

Francesco Flammini  
Sandro Bologna  
Valeria Vittorini (Eds.)

LNCS 6894

# Computer Safety, Reliability, and Security

30th International Conference, SAFECOMP 2011  
Naples, Italy, September 2011  
Proceedings



 Springer

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

*Lancaster University, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Alfred Kobsa

*University of California, Irvine, CA, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*TU Dortmund University, Germany*

Madhu Sudan

*Microsoft Research, Cambridge, MA, USA*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Gerhard Weikum

*Max Planck Institute for Informatics, Saarbruecken, Germany*

Francesco Flammini Sandro Bologna  
Valeria Vittorini (Eds.)

# Computer Safety, Reliability, and Security

30th International Conference, SAFECOMP 2011  
Naples, Italy, September 19-22, 2011  
Proceedings

Volume Editors

Francesco Flammini  
Ansaldo STS  
Via Argine, 425, 80147 Napoli, Italy  
E-mail: francesco.flammini@ansaldo-sts.com

Sandro Bologna  
Italian Association Critical Infrastructure (AIIC)  
Rome, Italy  
E-mail: s.bologna@infrastrutturecritiche.it

Valeria Vittorini  
Università di Napoli Federico II  
Dipartimento di Informatica e Sistemistica  
Via Claudio, 21, 80125 Napoli, Italy  
E-mail: valeria.vittorini@unina.it

ISSN 0302-9743 e-ISSN 1611-3349  
ISBN 978-3-642-24269-4 e-ISBN 978-3-642-24270-0  
DOI 10.1007/978-3-642-24270-0  
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011936469

CR Subject Classification (1998): K.6.5, C.2, D.2, H.3, D.4.6, E.3

LNCS Sublibrary: SL 2 – Programming and Software Engineering

© Springer-Verlag Berlin Heidelberg 2011

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

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

Printed on acid-free paper

Springer is part of Springer Science+Business Media ([www.springer.com](http://www.springer.com))

# Preface

Some outstanding writers of the last century have depicted an imaginary future in which intelligent machines ruled upon human beings. While most of the machines surrounding us can not be considered “intelligent” in the common sense (though they probably would to the eyes of people of some decades ago), a similar scenario can be considered nowadays as real: we do realize or not, nearly every activity we perform during our everyday life relies upon the dependability of computer-controlled devices, ranging from automatic transaction modules to brake-by-wire systems.

Furthermore, it is a matter of fact that the complexity and criticality of computer systems have grown substantially in the last years, and they are continuously increasing. Complexity is a result of three main factors: size, distribution and heterogeneity. Size refers to the number of functionalities requested to modern computers, which imply larger programs. Distribution is an effect of the need for networked devices, almost always required by the specific applications. Heterogeneity is given by the different hardware and software architectures involved in the design. The criticality attribute is related to the domains in which computer systems operate, whereas a failure can cause a significant loss of money, injuries, kills or even natural disasters. Please note that “critical” does not always imply “hard real-time”.

Such a scenario requires the adoption of novel techniques and tools in order to assure the dependability of computer systems, taking into account their interaction with other entities in terms both of the negative effects of the system upon the external environment (safety) and of the external environment upon the system (security).

The idea behind the choice of the main theme of the 30th edition of the International Conference on Computer Safety, Reliability and Security (SAFE-COMP 2011) has been the need for mastering complexity and criticality of modern computer-based systems. One of the best way to address that issue is the adoption of rigorous model-based engineering techniques, together with a holistic system-centric view, including all the components, abstraction layers and life-cycle phases.

As a result of this choice, the program of the conference reflects the contributor expertise in the following specialties, which are strictly related to the development of high-assurance systems:

- Computer dependability, studying the dynamics of propagation of random and systematic faults and the related protection mechanisms (fault-tolerance, error management, etc.).
- Software engineering, with special focus on simulative and analytical approaches of verification and validation (including software testing and formal methods).

- Risk analysis, addressing multi-disciplinary aspects of man-machine interaction and safety assessment procedures, using both qualitative and quantitative means.
- Multi-paradigm modeling, needed to master the increasing complexity of critical systems by integrating and evaluating heterogeneous models in cohesive system-level views.
- Information security, which plays an important role in high-integrity and business critical systems, needing robust authentication and communication protocols to protect against natural as well as malicious threats.

Such a mixture of topics has also helped to fill the “gap” existing between the research areas of computer dependability and critical infrastructure security.

All the aforementioned issues are addressed in this book, which represents the proceedings of the 30th edition of the International Conference on Computer Safety, Reliability and Security (SAFECOMP 2011), held in Naples, Italy, 19-21 September 2011. The proceedings includes 34 papers, but the response to the call for Papers was so high, that make all papers could be included in the volume.

As Chairpersons of the International Program Committee (IPC) and the National Organizing Committee, we would like to thank all authors who submitted their work, the presenters of the papers, the members of the IPC, the reviewers, the members of the National Organizing Committee, the session chairmen, and the sponsors for their efforts and support. Without their strong motivation and hard work we could not develop a succesfull and valuable conference as well as this book of proceedings.

September 2011

Francesco Flammini  
Sandro Bologna  
Roberto Setola  
Valeria Vittorini

# Organization

SAFECOMP 2001 has been organized by Dipartimento di Informatica e Sistemistica - University of Naples Federico II, Centro Regionale ICT (CeRICT), Ansaldo STS and European Workshop on Industrial Computer Systems Reliability, Safety and Security (EWICS).

## EWICS Chair

Francesca Saglietti (University of Erlangen-Nüemberg, Germany)

## Honorary Chairs

Giovanni Bocchetti (Ansaldo STS, Italy)

Giorgio Franceschetti (University of Naples Federico II, Italy)

Antonino Mazzeo (University of Naples Federico II, Italy)

## Program Chairs

Francesco Flammini (Ansaldo STS, Italy)

Sandro Bologna (ENEA, Italy)

## Local Chairs

Nicola Mazzocca (University of Naples Federico II, Italy)

Concetta Pragliola (Ansaldo STS, Italy)

Roberto Setola (University Campus Biomedico of Rome, Italy)

Valeria Vittorini (University of Naples Federico II, Italy)

## Program Committee

J.P. Blanquart (FR)

A. Bondavalli (IT)

R. Bloomfield (UK)

B. Buth (DE)

A. Coronato (IT)

D. Cotroneo (IT)

G. D'Agostino (IT)

P. Daniel (UK)

S. DAntonio (IT)

G. De Pietro (IT)

F. Di Giandomenico (IT)

W. Ehrenberger (DE)

M. Felici (UK)

N. Ferreira Neves (PT)

G. Franceschinis (IT)

G. Gigante (IT)

L. Glielmo (IT)

J. Gorski (PL)

W. Halang (DE)  
R.E. Harper (USA)  
M. Heisel (DE)  
J. Jurjens (DE)  
R. Jimenez (ES)  
K. Kanoun (FR)  
J. Karlsson (SE)  
T. Kelly (UK)  
F. Koornneef (NL)  
P. Ladkin (DE)  
S. Lindskov Hansen (DK)  
B. Littlewood (UK)  
G. Manco (IT)  
T. Margaria (DE)  
E. Meda (IT)  
P.J. Mosterman (USA)  
T. Nanya (Japan)  
O. Nordland (NO)  
F. Ortmeier (DE)

A. Pataricza (HU)  
D. Powell (FR)  
P. Prinetto (IT)  
L. Romano (IT)  
A. Romanovsky (UK)  
S. Russo (IT)  
F. Saglietti (DE)  
E. Schoitsch (AT)  
T. Seyfarth (DE)  
B. Siciliano (IT)  
L. Strigini (UK)  
M. Sujan (UK)  
N. Suri (DE)  
K. Trivedi (USA)  
M. van der Meulen (NL)  
V. Vittorini (IT)  
A. Vozella (IT)  
S. Zanero (IT)  
Z. Zurakowski (PL)

## Additional Reviewers

Flora Amato (IT)  
Fabrizio Baiardi (IT)  
Gianmarco Baldini (IT)  
Claudio Bareato (CH)  
Angelo Berghella (IT)  
Simona Bernardi (E)  
Andrea Bobbio (IT)  
Claudio Luigi Brasca (IT)  
Luigi Buonanno (IT)  
Audrey Canning (UK)  
Emiliano Casalicchio (IT)  
Valentina Casola (IT)  
Mario Ciampi (IT)  
Emanuele Ciapessoni (IT)  
Tadeusz Cichocki (PL)  
Alessandro Cilardo (IT)  
Carlo Alberto Clarotti (IT)  
Andrea Colini (IT)  
Antonio Di Pietro (IT)  
Stelios Dritisas (GR)  
Massimo Esposito (IT)  
Anna Rita Fasolino (IT)  
Andrea Fiaschetti (IT)

Roberto Filippini (IT)  
Vincenzo Fioriti (IT)  
Chiara Foglietta (IT)  
Luisa Franchina (IT)  
Giustino Fumagalli (IT)  
Andrea Gaglione (IT)  
Ilir Gashi (UK)  
Gokce Gorbil (UK)  
Dimitris Gritzalis (GR)  
Vincenzo Gulisano (E)  
Mauro Iacono (IT)  
Federico Maggi (IT)  
Loredana Mancini (IT)  
Stefano Marrone (IT)  
Fiammetta Marulli (IT)  
Francesca Matarese (IT)  
Oliver Meyer (DE)  
Paolo Nocito (IT)  
Gabriele Oliva (IT)  
Antonio Orazzo (IT)  
Stefano Panzieri (IT)  
Alfio Pappalardo (IT)  
Peter Popov (UK)



Andrey Povyakalo (UK)  
Erich Rome (DE)  
Guido Salvaneschi (IT)  
Damian Serrano (FR)  
Claudio Soriente (E)  
Federica Sorrentino (IT)  
Luigi Sportiello (IT)  
Marianthi Theoharidou (GR)

Alberto Tofani (IT)  
Bill Tolone (USA)  
Enrico Tronci (IT)  
Salvatore Venticinque (IT)  
Min Wu (USA)  
Christos Xenakis (GR)  
Loredana Zollo (IT)

## Organizing Committee

Flora Amato	University of Naples Federico II
Carmen C. Baruffini	University of Naples Federico II
Valentina Casola	University of Naples Federico II
Alessandra De Benedictis	University of Naples Federico II
Domenico Di Leo	University of Naples Federico II
Stefano Marrone	Second University of Naples
Roberto Nardone	University of Naples Federico II
Alfio Pappalardo	University of Naples Federico II
Antonio Pecchia	University of Naples Federico II
Sara Romano	University of Naples Federico II

# Table of Contents

## Session 1: Ram Evaluation 1

The Effect of Correlated Failure Rates on Reliability of Continuous Time 1-Out-of-2 Software . . . . .	1
<i>Peter Popov and Gabriele Manno</i>	
Model-Driven Availability Evaluation of Railway Control Systems . . . . .	15
<i>Simona Bernardi, Francesco Flammini, Stefano Marrone, José Merseguer, Camilla Papa, and Valeria Vittorini</i>	

## Session 2: Complex Systems Dependability 1

Vertical Safety Interfaces – Improving the Efficiency of Modular Certification . . . . .	29
<i>Bastian Zimmer, Susanne Bürklen, Michael Knoop, Jens Höfflinger, and Mario Trapp</i>	
DALculus – Theory and Tool for Development Assurance Level Allocation . . . . .	43
<i>Pierre Bieber, Rémi Delmas, and Christel Sequin</i>	
Towards Cross-Domains Model-Based Safety Process, Methods and Tools for Critical Embedded Systems: The CESAR Approach . . . . .	57
<i>Jean-Paul Blanquart, Eric Armengaud, Philippe Baufreton, Quentin Bourrouilh, Gerhard Griessnig, Martin Krammer, Odile Laurent, Joseph Machrouh, Thomas Peikenkamp, Cecile Schindler, and Tormod Wien</i>	

## Session 3: Formal Verification 1

From Probabilistic Counterexamples via Causality to Fault Trees . . . . .	71
<i>Matthias Kuntz, Florian Leitner-Fischer, and Stefan Leue</i>	
Rigorous Evidence of Freedom from Concurrency Faults in Industrial Control Software . . . . .	85
<i>Richard Bonichon, Géraud Canet, Loïc Correnson, Eric Goubault, Emmanuel Haucourt, Michel Hirschowitz, Sébastien Labbé, and Samuel Mimram</i>	

**Session 4: Risk and Hazard Analysis**

Evolutionary Risk Analysis: Expert Judgement ..... 99  
*Massimo Felici, Valentino Meduri, Bjørnar Solhaug, and  
Alessandra Tedeschi*

Computer-Aided PHA, FTA and FMEA for Automotive Embedded  
Systems ..... 113  
*Roland Mader, Eric Armengaud, Andrea Leitner, Christian Kreiner,  
Quentin Bourrouilh, Gerhard Griesßnig, Christian Steger, and  
Reinhold Weiß*

**Session 5: Cybersecurity**

A Statistical Anomaly-Based Algorithm for On-line Fault Detection in  
Complex Software Critical Systems ..... 128  
*Antonio Bovenzi, Francesco Brancati, Stefano Russo, and  
Andrea Bondavalli*

Security Analysis of Smart Grid Data Collection Technologies ..... 143  
*Luigi Coppolino, Salvatore D’Antonio, Ivano Alessandro Elia, and  
Luigi Romano*

**Session 6: RAM Evaluation 2**

Modeling Aircraft Operational Reliability ..... 157  
*Kossi Tiassou, Karama Kanoun, Mohamed Kaâniche,  
Christel Seguin, and Chris Papadopoulos*

An Integrated Approach for Availability and QoS Evaluation in  
Railway Systems ..... 171  
*Antonino Mazzeo, Nicola Mazzocca, Roberto Nardone,  
Luca D’Acierno, Bruno Montella, Vincenzo Punzo,  
Egidio Quaglietta, Immacolata Lamberti, and Pietro Marmo*

**Session 7: Case Studies**

Using a Software Safety Argument Pattern Catalogue: Two Case  
Studies ..... 185  
*Richard Hawkins, Kester Clegg, Rob Alexander, and Tim Kelly*

Integration of a System for Critical Infrastructure Protection with the  
OSSIM SIEM Platform: A Dam Case Study ..... 199  
*Luigi Coppolino, Salvatore D’Antonio, Valerio Formicola, and  
Luigi Romano*

A Case Study on State-Based Robustness Testing of an Operating System for the Avionic Domain . . . . .	213
<i>Domenico Cotroneo, Domenico Di Leo, Roberto Natella, and Roberto Pietrantuono</i>	

## Session 8: Formal Verification 2

Formal Methods for the Certification of Autonomous Unmanned Aircraft Systems . . . . .	228
<i>Matt Webster, Michael Fisher, Neil Cameron, and Mike Jump</i>	
Verifying Functional Behaviors of Automotive Products in EAST-ADL2 Using UPPAAL-PORT . . . . .	243
<i>Eun-Young Kang, Pierre-Yves Schobbens, and Paul Pettersson</i>	

## Poster Session

Establishing Confidence in the Usage of Software Tools in Context of ISO 26262 . . . . .	257
<i>Joachim Hillebrand, Peter Reichenpfader, Irenka Mandic, Hannes Siegl, and Christian Peer</i>	
Fault-Based Generation of Test Cases from UML-Models – Approach and Some Experiences . . . . .	270
<i>Rupert Schlick, Wolfgang Herzner, and Elisabeth Jöbstl</i>	
ISO/IEC 15504-10: Motivations for Another Safety Standard . . . . .	284
<i>Giuseppe Lami, Fabrizio Fabbrini, and Mario Fusani</i>	
Automatic Synthesis of SRN Models from System Operation Templates for Availability Analysis . . . . .	296
<i>Kumiko Tadano, Jiangwen Xiang, Masahiro Kawato, and Yoshiharu Maeno</i>	
A Collaborative Event Processing System for Protection of Critical Infrastructures from Cyber Attacks . . . . .	310
<i>Leonardo Aniello, Giuseppe A. Di Luna, Giorgia Lodi, and Roberto Baldoni</i>	
A Fault-Tolerant, Dynamically Scheduled Pipeline Structure for Chip Multiprocessors . . . . .	324
<i>Hananeh Alikee and Hamid Reza Zarandi</i>	
FloGuard: Cost-Aware Systemwide Intrusion Defense via Online Forensics and On-Demand IDS Deployment . . . . .	338
<i>Saman Aliari Zonouz, Kaustubh R. Joshi, and William H. Sanders</i>	

Reducing Complexity of Data Flow Testing in the Verification of a  
IEC-62304 Flexible Workflow System . . . . . 355  
*Federico Cruciani and Enrico Vicario*

Improvement of Processes and Methods in Testing Activities for  
Safety-Critical Embedded Systems . . . . . 369  
*Giuseppe Bonifacio, Pietro Marmo, Antonio Orazio, Ida Petrone,  
Luigi Velardi, and Alessio Venticinqu*

**Session 9: Formal Verification 3**

On the Adoption of Model Checking in Safety-Related Software  
Industry . . . . . 383  
*Alessandro Fantechi and Stefania Gnesi*

Equivalence Checking between Function Block Diagrams and  
C Programs Using HW-CBMC . . . . . 397  
*Dong-Ah Lee, Junbeom Yoo, and Jang-Soo Lee*

A Framework for Simulation and Symbolic State Space Analysis of  
Non-Markovian Models . . . . . 409  
*Laura Carnevali, Lorenzo Ridi, and Enrico Vicario*

**Session 10: Optimization Methods**

Model-Based Multi-objective Safety Optimization . . . . . 423  
*Matthias GÜdemann and Frank Ortmeier*

Tradeoff Exploration between Reliability, Power Consumption, and  
Execution Time . . . . . 437  
*Ismail Assayad, Alain Girault, and Hamoudi Kalla*

**Session 11: Complex Systems Dependability 2**

Criticality-Driven Component Integration in Complex Software  
Systems . . . . . 452  
*Antonio Pecchia, Roberto Pietrantuono, and Stefano Russo*

On the Use of Semantic Technologies to Model and Control Security,  
Privacy and Dependability in Complex Systems . . . . . 467  
*Andrea Fiaschetti, Francesco Lavorato, Vincenzo Suraci,  
Andi Palo, Andrea Tagliatela, Andrea Morgagni,  
Renato Baldelli, and Francesco Flammini*

**Author Index . . . . . 481**

# The Effect of Correlated Failure Rates on Reliability of Continuous Time 1-Out-of-2 Software

Peter Popov<sup>1</sup> and Gabriele Manno<sup>2</sup>

<sup>1</sup> Centre for Software Reliability, City University,  
Northampton Square, London, UK  
ptp@csr.city.ac.uk

<sup>2</sup> Department of Mathematics and Informatics, University of Catania  
Viale Andrea Doria 6, Catania, Italy  
gmanno@dmi.unict.it

**Abstract.** In this paper we study the effects on system reliability of the correlation over input space partitions between the rates of failure of two-channel fault-tolerant control software. We use a continuous-time semi-Markov model to describe the behavior of the system. We demonstrate via simulation that the variation of the failure rates of the channels over the partitions of the input space can affect system reliability very significantly. With a plausible range of model parameters we observed that the mean time to system failure may vary by *more than an order of magnitude*: positive correlation between the channel rates makes the system less reliable while negative correlation between the channel rates implies that the system is more reliable than assuming constant failure rates for the channels. Our observations seem to make a case for *more detailed reliability measurements* than is typically undertaken in practice.

## 1 Motivation

All systems need to be *sufficiently* reliable. There are two related issues here. In the first place there is the issue of *achieving* the necessary reliability. Secondly, there is the issue of *assessing* reliability that has actually been achieved, to convince oneself that it is 'good enough'.

In the light of the rather strict limitations to the levels of software reliability that can typically be achieved or claimed from observation of operational behavior of a single version program [1], fault tolerance via *design diversity* has been suggested as a way forward both for achieving higher levels of reliability, and for assisting in its assessment.

Design diversity has been studied thoroughly in the past 30+ years. For a relatively recent study the reader is referred to [2]. The focus, however, has been primarily on 'on demand' systems, e.g. a protection system called upon when a failure is detected in the operation of the system controlling a plant.

The focus of this paper is *control software*, i.e. which exercises control of an object of control and in the process executes a series of inputs (trajectories) coming directly from the controlled object, its environment and also the internal state of the software

itself. Assessing accurately reliability of control software is important not only for minimizing the losses due to downtime. In some cases, e.g. of critical applications, the control software reliability defines reliability requirements for the protection system designed and deployed to deal with situations of inadequate control. A protection system of given reliability may be adequate in some cases – e.g. when the control system is very reliable – or may be inadequate – e.g. if the control system is of modest reliability. Reliability of the total system (control and protection) depends on both reliability of the control and of the protection and therefore accurately assessing reliability of both systems is important.

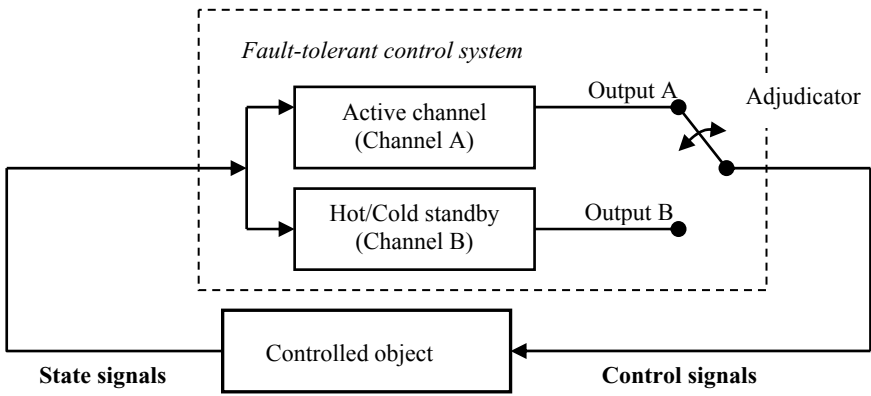
Our focus in this paper is a 2-channel control software for which the input space is divided in partitions, which represent different *modes of operation*. Examples of modes of operation might be an initialization, a normal control loop and terminating the control, e.g. to allow for maintenance. More refined scenario, e.g. as in robotics, might include a robot having to deal with different obstacles, which may require applying different algorithms of adaptive behavior to the current environment, etc. We address on purpose the problem at a sufficiently high level of abstraction – using a continuous time semi-Markov model – which will allow us to state the main result in a concise way.

Continuous-time semi-Markov models are typically used to model the behavior of control software: the modeling assumptions and the model details depend on the specific aspects of interest. For instance, failure clustering is typical for control software [3]. Modeling such behavior is impossible with models assuming that successive inputs are drawn independently from the input space. Instead, models, in which the failure rate changes significantly after the occurrence of the first failure proved to be useful [4].

The paper is organized as follows. Section 2 states the problem. Section 3 presents the model and the main result. In section 4 we compare our model with a model developed in the past by Littlewood for a single channel software with modular structure. In section 5 we discuss our findings and some parameter estimation techniques. Section 6 offers a survey of the relevant literature. The conclusions and directions for future research are presented in section 7.

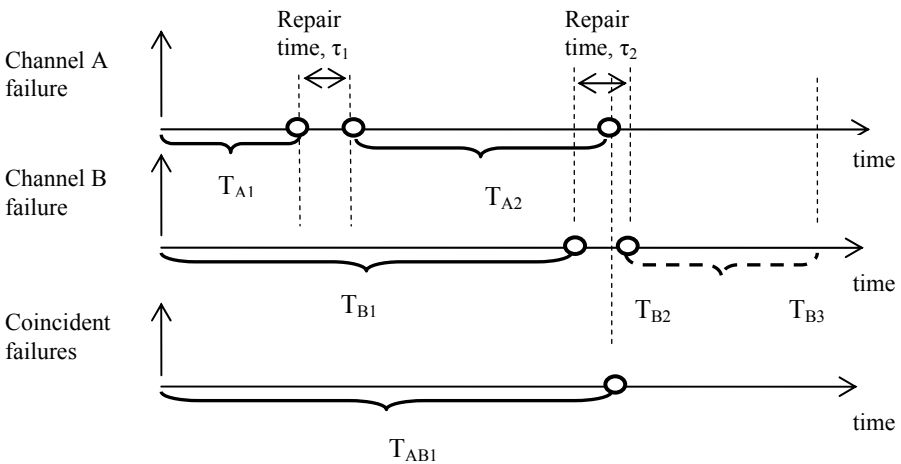
## 2 The Problem

Consider a 2-channel control system as shown in Figure 1. During the operation of the system each of the channels can fail and so can the adjudicator. In this paper we concentrate on the case of an *absolutely reliable adjudicator* and study reliability of the control system only. Once a channel failure is detected by the adjudicator an attempt is undertaken to ‘repair’ the failed channel, which eventually succeeds after time  $\tau$ , during which time the other channel will either work correctly or will also fail. Examples of repair envisaged here are the typical backward/forward recovery mechanisms used in practice such as retrying the execution with a slightly perturbed data [3] or merely restarting the failed channel.



**Fig. 1.** A typical architecture of a 2-channel control system. At any time the outputs to the controlled object (e.g. a nuclear plant) are generated by one of the channels, while the second channel is available as a hot/cold standby. An adjudicator is responsible to detect anomalies of the active channel and switch to the standby channel, if such is available. The failed channel is ‘repaired’ which takes finite time and becomes available to take over control again. The channels are diverse – if design faults are considered – or merely redundant if only hardware related faults are considered.

The channels are assumed to fail independently of each other: the chances of both channels failing simultaneously are, therefore, vanishingly small. The only source of coincident failure is the finite repair time  $\tau$  of the failed channel during which the second channel may also fail. We later will discuss relaxing the assumption of independent failure and discuss the model of a “common stress” that might cause simultaneous failure of both channels.



**Fig. 2.** The timing diagram illustrates the events of interest and the times associated with them. The times,  $T_{Ax}$ ,  $T_{Bx}$  and  $T_{ABx}$ , respectively, characterize the up times in the stochastic processes associated with the behavior of the individual channels and the control system.



We assume, further that the system's input space is divided into *partitions*, identical for both channels. Each of these partitions is associated with rates of failure of the channels, respectively. These rates may vary across partitions and it is the nature of this variation – whether the rates of channel failure are positively or negatively correlated or not correlated (e.g. the rate of one of the channels does not vary over the partitions at all) – that the study is focused on.

Figure 2 shows a typical timing diagram which illustrates the failure processes of interest.

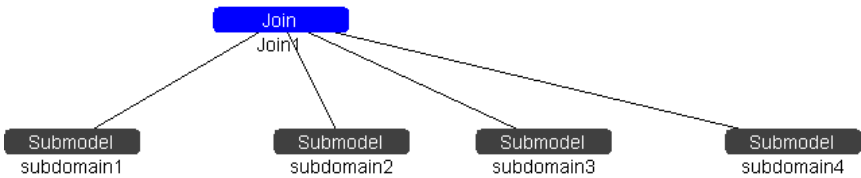
In this study we concentrate on the time to system failure (i.e. the times until both channels fail) starting from a state when both channels are operational. Clearly, the time to failure may include multiple cases of a single channel failure and successful repairs of the failed channel.

### 3 Model of the System

We studied the problem using the formalism of stochastic activity networks (SAN) and the tool support offered by the Möbius tool [5]. The results – the distribution of the time to system failure – are obtained *via simulation*.

#### 3.1 Diagrammatic Representation of the Model

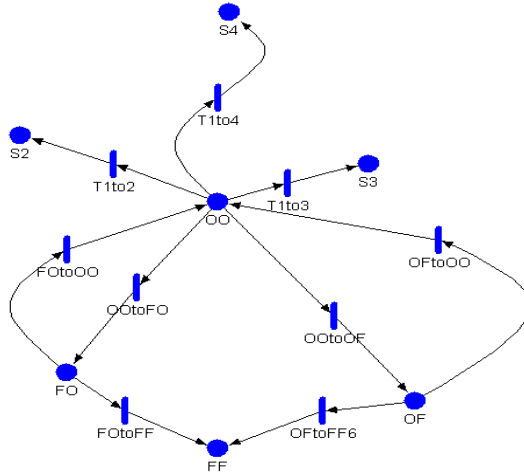
Now we defined the system model. Consider that the system can be represented as a stochastic activity network, in which there are several partitions as shown in Figure 3.



**Fig. 3.** Model of a system operating on 4 partitions of the input space, subdomain1 – subdomain4. The syntax of the graphical representation is Möbius specific. Each of the partitions is a detailed representation of the states that the system (the two channels) might be in while in the respective partition.

Figure 4 shows in detail the system behavior of the system in one of the partitions. The models of the other partitions are identical, but the parameters may differ. The system changes its state from both channels working correctly (OO) to states in which one of the channels has failed (OF or FO), from which it may either recover (i.e. return to OO) or instead the second channel may also fail (i.e. reach the state FF). While in OO state, the system may switch to a different partition: the other partitions are labeled  $S_2, S_3, S_4$ , which are really labels for the OO states in the respective partitions (**subdomain<sub>2</sub>**, **subdomain<sub>3</sub>** and **subdomain<sub>4</sub>**). One notices that in our model the system cannot switch to a different partition unless both channels work correctly. This simplifying assumption seems plausible. In a typical scenario of a fast repair (e.g. a restart) and a relatively infrequent change of modes of operation (modeled by the

partitions of the input space), the chances that the system will have moved to a different partition with one of the channels not working correctly are negligible. In some other cases, however, the transitions between the partitions may be fast and the simplification introduced in the model may be problematic. Relaxing this assumption, although possible, is beyond the scope of the paper.



**Fig. 4.** Detailed behavior of the 1-out-of-2 software in partition subdomain<sub>1</sub>. The model states are shown as places (nodes) OO, OF, FO and FF suitably named to indicate the state of the channels: both channels working correctly is represented by the place OO, ..., both states having failed is represented by the place FF, respectively. The transitions between the places are characterized by a set of ‘stochastic activities’, e.g. a change of the system state from OO to FO is represented by the stochastic activity O0toFO. A transition in the opposite direction (FO → OO) is represented by the stochastic activity F0toOO. The place FF is absorbing, i.e. there are no outgoing transitions (activities) from it to some other places.

### 3.2 Möbius Model Parameters

The model is parameterized under a set of assumptions:

- Failures of the channels are driven by independent Poisson processes, which are homogeneous conditional on sub-domains, but may be non-homogeneous across partitions.
- Repairs of the channels are perfect, but not instantaneous. Repairs are only undertaken if there is a channel working correctly.

Given these assumptions the model was parameterized as follows.

- The transitions between the partitions (between the respective OO states, that is) are all assumed *exponentially distributed* with a rate of 0.3. The uniformity of the rates here was chosen for *convenience*: we wanted to keep the channels equally reliable and be able to vary easily their rates of failure in partitions. Any difference, thus, in the system behavior observed between

the studied cases could be attributed solely to the correlation between the failure rates in the partitions. This objective is easily achieved if the domains are equally likely, which in turn is achieved by setting the same transition rates between the OO states of the partitions.

- The repair times were assumed of *fixed duration*, 0.01 units.
- The failure rates in the partitions are chosen from the set {0.01, 0.02, 0.03} in such a way that the marginal rates of failure of the channels remain unchanged (0.02 given the partitions are equally likely,  $\approx 0.25$ ).

The following cases (see Table 1) were studied, which represent different types of correlation between the failure rates of the channels over the partitions.

**Table 1.** Failure rates of the channels conditional on partitions ( $S_1$ - $S_4$ )<sup>1</sup>

		$S_1$	$S_2$	$S_3$	$S_4$
Experiment 1: ‘High’ Positive correlation between the rates.	Channel 1	0.03	0.01	0.03	0.01
	Channel 2	0.03	0.01	0.03	0.01
Experiment 2: ‘High’ Negative correlation between the rates.	Channel 1	0.03	0.01	0.03	0.01
	Channel 2	0.01	0.03	0.01	0.03
Experiment 3: Constant rates of both channels.	Channel 1	0.02	0.02	0.02	0.02
	Channel 2	0.02	0.02	0.02	0.02
Experiment 4: Constant rate of channel 1.	Channel 1	0.02	0.02	0.02	0.02
	Channel 2	0.01	0.03	0.01	0.03
Experiment 5: ‘Low’ positive correlation between the rates.	Channel 1	0.01	0.02	0.03	0.02
	Channel 2	0.01	0.02	0.03	0.02
Experiment 6: ‘Low’ Negative correlation between the rates.	Channel 1	0.03	0.02	0.01	0.02
	Channel 2	0.01	0.02	0.03	0.02

As one can see, a uniform profile on the set of partitions ( $P(S_1) = P(S_2) = P(S_3) = P(S_4) = 0.25$ ) guarantees that the marginal rates of failure of the channels indeed remains the same – 0.02.

### 3.3 Measure of Interest

The *time to system failure* was measured via simulation and the results are summarized in Table 2.

Clearly, the mean time to system failure is significantly affected by the covariance between the failure rates<sup>2</sup>. The greatest MTTF corresponds to Experiment 2 with high negative correlation between the rates of failure of the channels. The other extreme – the shortest MTTF – corresponds to the case with high positive correlation between the conditional rates of failure of the channels. A constant rate of failure of at least one of the channels (Experiment 3 and Experiment 4) forms the ‘case in the middle’, while more modest correlations – either positive or negative – place the respective cases between the ‘case in the middle’ and the respective cases with high correlation of the same sign.

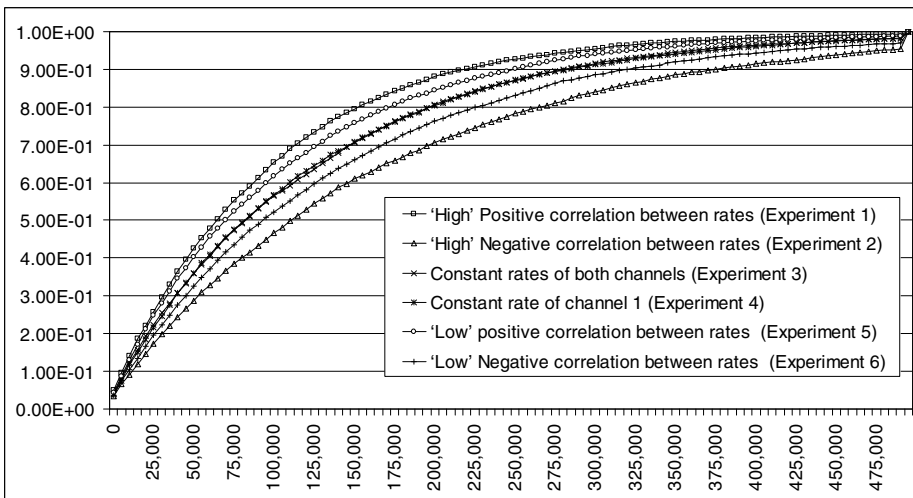
<sup>1</sup>  $S_1 - S_4$  are shortcuts for **subdomain**<sub>1</sub> – **subdomain**<sub>4</sub>, respectively.

<sup>2</sup> The MTTF of Experiment 3 and 4 are very close, but for these the covariance of the failure rates is 0, as for at least one of the channels the failure rates are constant over partitions.

**Table 2.** Mean time to system failure

	Mean		Runs
	Value	95% Confidence interval	
'High' Positive correlation between rates (Experiment 1)	97,569.53	+/- 2,000.9	12,000
'High' Negative correlation between rates (Experiment 2)	157,353.8	+/- 4,563.3	5,000
Constant rates of both channels (Experiment 3)	122,060.4	+/- 2,972.8	8,000
Constant rate of channel 1 (Experiment 4)	122,498.6	+/- 2,996.9	8,000
'Low' positive correlation between rates (Experiment 5)	107,831.7	+/- 2,426.1	10,000
'Low' Negative correlation between rates (Experiment 6)	137,377.9	+/- 3,504.0	7,000

We looked at the distributions associated with the experiments, which are presented in Figure 5. It turned out that the times to system failures are *stochastically ordered*: the ordering being the same as the ordering between the respective MTTFs (Table 2). Analyzing these distributions we established that in all 6 experiments they can be approximated very well using an exponential distribution with parameters equal to the reciprocal of the means defined in Table 2.

**Fig. 5.** Distribution of the time to system failure, truncated after 500,000 time units of simulation

We scrutinized further, via simulation, how the distribution of the activities associated with failures of the channels will affect the distribution of the time to

system failure. While the activities modeling the transitions between sub-domains were left exponentially distributed with a rate of 0.3 and fixed repair times of 0.01 were used, as before, we set the activities modeling the time to a channel failure to have Weibull and Gamma distributions with parameters which lead to non-constant hazard rate. The parameters were chosen in such a way that the transitions between the partitions remained significantly more frequent than the channel failures. The distributions of the activities did affect the time to system failure very significantly. The effect that we highlighted above, however, remained in place: negatively correlated rates would lead to longer times to system failure than constant rates, which in turn were longer than if the rates of failure of the channels were positively correlated. We observed that MTTF may differ up to an *order of magnitude* between positively and negatively correlated rates. The time to system failure in all simulated cases remained exponentially distributed despite the significant differences in the rates.

## 4 Littlewood's Semi-markov Model of Software Reliability

Littlewood studied [6] systems with modular structure. The structures he considered were defined by the software modules (functions, procedures, etc.) of which the software consists. He assumed that the failures of the software can happen within a module or during the invocation of a module by another module.

Littlewood's reasoning is based on *three essential assumptions*:

- the underlying stochastic process describing the software behavior is semi-Markov. The time that software spends in a module (the sojourn time) can have an arbitrary distribution, but the transition probabilities between the modules are *constant*.
- while occupying a module the program may fail randomly with a *constant failure rate*.
- the transition probabilities between the modules are *significantly greater* than the rates of failure (either within the module or during the module invocation). Otherwise the software would have been, Littlewood argues, very unreliable.

Under these assumptions Littlewood proved analytically that the failure process will be a Poisson process. Its parameter can be computed from the steady-state probabilities of the embedded Markov chain (after eliminating the failure state, which in his description was absorbing), the mean times the program spends in a module and the small failure rates.

How does the model described by Littlewood differ from the one used by us to model the behaviour of a 2-channel control system?

The first assumption by Littlewood is clearly sufficiently general to apply to our model too. Our model is a model of a semi-Markov process, too. Although we describe the model in terms of transitions between the partitions and ignore the internal structure of software (functions, procedures, etc.) the model is conceptually very similar: there are states represented by the partitions and transitions between these states. In each state our model is strictly a model of competing risks – the

shortest activity defines the next system state. However, one can easily transform the competing risk model with states to a semi-Markov process. Indeed, one can directly express the sojourn time as a function of the distributions chosen for the activities [7]. The marginal probabilities that the random variable representing a particular competing risk will be the shortest one can easily be derived, too (see the Appendix for details). These probabilities will form the transition probabilities for the embedded Markov chain associated with the semi-Markov model. In summary, the first assumption of the Littlewood theorem is satisfied.

The second assumption, however, is not always satisfied. For exponentially distributed activities representing the channel failure, the assumption is satisfied, but for Weibull and Gamma distributed activities – it is not. Thus, our model formally violates the second assumption made by Littlewood, that failures occur randomly.

The third assumption made by Littlewood is also plausible in our case: as evident from the used parameterization, the transitions to all non-absorbing states are significantly more frequent (including the repair) than the transition to the absorbing state of failure of both channels<sup>3</sup>.

Despite the violation of the second assumption made by Littlewood, his asymptotic result seems to apply: asymptotically the time to system failure is exponentially distributed. It is outside the scope of this paper to offer an analytic explanation why this is the case, a problem worth addressing in the future.

## 5 Discussion

The effect reported in this paper, that the correlation of failure rates over partitions of the input space matters, is not surprising. Similar effects, that variation of the probability of failure, conditional on partitions, have been studied extensively in the past for on-demand systems [8].

The practical implications of the work presented here seem significant. If one is to measure the marginal rates of failure of the channels and then use these to estimate, e.g. by simulation, how the speed of recovery will impact reliability of the system one will be implicitly assuming the situation described by our example 3 – no variation of failure rates of both channels. But such an evaluation may be incorrect – it may be

---

<sup>3</sup> There is a subtle difference between a semi-Markov process and the model of competing risks which is rarely discussed in the literature. For the competing risks model the transition probabilities are proportional to the respective *hazard rates* (of the random variables representing the competing risks), while in the semi-Markov process it is typically assumed that an embedded Markov chain exists with fixed transition probabilities. If all competing risks are exponentially distributed, then the hazard rates are constant and so are the transition probabilities – they remain the same irrespective of the length of the sojourn time. However, if at least some of the competing risks are not exponentially distributed then the hazard rate may vary over time (e.g. with Weibull distribution it may increase or decrease) and thus it becomes dependent on the duration of the sojourn time (see the Appendix for further details). This subtle difference, however, does not seem to matter, at least not for our studies. Despite exploring a wide range of scenarios (with activities assumed to have Gamma and Weibull distributions) the distributions of the times to system failure remained exponentially distributed.

optimistic or pessimistic depending on the variation of rates of failure in partitions. Results based on ignoring the variation of the failure rates will only be useful if one can demonstrate that at least one of the channels fails with the same failure rate in all partitions (as in experiment 4 in Table 2). Constant failure rate over the partitions, however, *does not seem realistic*. For various reasons the partitions are likely to be subjected to different scrutiny – some are less critical than others, or are less used by the users and hence problems are less likely to be reported, etc. The point in the end is that, ignoring the effect we report here may lead to overestimation or underestimation of system reliability and it is *impossible to know in advance* even the sign of the estimation error one will make by ignoring the correlation between the rates over the partitions. Overestimating system reliability may be dangerous, while underestimating may lead to waste of resources – e.g. unnecessarily insisting on further V&V to improve system reliability even if the system may already be ‘good enough’ (e.g. in the case the channel rates of failure are negatively correlated).

The model presented above assumes that the channel failure processes are independent processes. This assumption can be relaxed. An example is the model of ‘common stress’, which causes both channels to fail simultaneously, e.g. due to a specification fault. A useful and widely used example is the Marshall and Olkin model [9]. The joint *pdf* of the channels’ time to failure is defines as follows:

$$f(x, y) = \begin{cases} \theta_1(\theta_2 + \theta_3)e^{-\theta_1 x - (\theta_2 + \theta_3)y}, & \text{if } x < y, \\ \theta_2(\theta_1 + \theta_3)e^{-\theta_2 y - (\theta_1 + \theta_3)x}, & \text{if } y < x, \\ \theta_3 e^{-(\theta_1 + \theta_2 + \theta_3)y}, & \text{if } x = y, \end{cases}$$

where  $x > 0$ ,  $y > 0$ ,  $\theta_1 > 0$ ,  $\theta_2 > 0$ , and  $\theta_3 > 0$ . X and Y are the lifetimes of the two channels subjected to three kinds of shocks, assumed to be governed by three independent Poisson processes with parameters  $\theta_1$ ,  $\theta_2$ , and  $\theta_3$ , respectively. Shock 1 applies to channel 1 only, shock 2 applies to channel 2 only, while shock 3 applies to both channels (hence ‘common stress’). The model has been used widely in nuclear reactor safety, competing risks reliability, etc [10]. The marginal distributions of X and Y are exponential distributions with parameters  $\theta_1 + \theta_3$  and  $\theta_2 + \theta_3$ , respectively.

Clearly we could easily integrate the Marshall and Olkin model in the model presented in section 3 by adding a third activity, from place OO (Figure 4) to the absorbing place FF, to model the common stress, i.e. shock 3.

Accurately measuring system reliability will require more detailed measurement, which includes the following steps:

- estimating the probabilities of the partitions (the partition profile). Provided sufficient statistical testing is undertaken, one could easily arrive at a very accurate estimate of the probabilities of partitions. With more care one can even measure directly the transition rates between the partitions, which will be used directly in the model;
- estimating the rates of channel failure in partitions. This may require more effort, than measuring the partition profile, especially in case of very reliable channels. Failure rates can be estimated from the log of observed channel failures.

- relaxing the assumption that the channels fail conditionally independently will further complicate the measurements. Now one will need to quantify the strength of dependence between the channel failure processes.

Developing in details techniques for parameters estimation is beyond the scope of this paper. We notice in passing that the theory of competing risks is well developed and has been applied successfully in a wide range of applications. In our model every place (or state of the system) is associated with a set of competing risks – several activities compete to move the system to one of the states reachable by a single activity. For instance, in an OO place (Figure 4) several risks (represented by their respective activities) compete – to move the system to a new partition or to a state in which one of the channels has failed (or both in case the model is extended to include a common stress). The parameters associated with the risks may be unknown with certainty and need to be estimated from the available observations. The Appendix provides details, including the likelihood of any possible observation, sufficient for parameter estimation – either by maximizing the likelihood of the observations or by applying Bayesian inference. The point here is that every time a transition from a state takes place, we collect an observation associated with the realization of the competing risks defined for this place. Given the assumed Markov structure of the model, estimating the model parameters will consist of independent data collection and estimation of the model parameters associated with the individual states.

We note that estimating the parameters of different parts of the model can be done using different techniques. For instance, we can obtain the parameters of the transitions (activities) between the OO states of the partitions directly from the observations (as these will be likely to be frequent and many realizations can be observed in a short period of observation). The parameters (i.e. distributions) of the activities OotoFO of OotoOF in turn can be assessed using Bayesian inference (given the typically small number of observations one can collect within a limited statistical testing or operational exposure). Once the estimation of the parameters of OotoFO and OotoOF activities is done, one can use these to parameterize the activities FtoFF and OFtoFF, as in the model we assume them to have the same parameters as OotoOF and OotoFO, respectively.

Once the parameters of the model activities are estimated, one could run a simulation experiment to measure directly the time to system failure. Further, as new operational data becomes available, one could revise the model (by re-assessing periodically the model parameters) and then re-run a new simulation to estimate the time to system failure.

## 6 Relevant Literature

Probabilistic models of on-demand fault-tolerant software have attracted significant attention. The original work by Ekhardt and Lee [11] demonstrated that failure independence is unlikely for even independently developed software versions (channels of a fault-tolerant system). The reason for this is that the individual demands processed by software may differ in their “difficulty”, i.e. they will be problematic to independent developers and the chances of simultaneous failures on



these demands of independently developed channels are greater than what would be expected assuming independent failures. This was a very important insight, which affected the research and practical adoption of software fault-tolerance. The model by Ekhardt and Lee was extended by Littlewood and Miller [12], to the case of forced design diversity (e.g. different teams are forced to use different development methodologies which may lead to different difficulties of the demands). This work demonstrated the possibility for achieving system reliability better than assuming failure independence between the channels. Popov and Littlewood [13] extended the earlier models by allowing the channels reliability to grow, e.g. as a result of testing and compared the effect on system reliability of different testing regimes – testing the channels in isolation, testing them together on the same testing suite and back-to-back and ranked these testing regimes according to their impact on system reliability.

These models were models “on average” – they modeled the process of software development of fault-tolerant software as a random selection from populations of versions, which hypothetically can be developed to a given specification. The models, however, do not address the issue of assessing reliability of a particular fault-tolerant system. This problem was addressed in [8]: the authors developed a model of a fault-tolerant software operating on demand space with partitions and demonstrated that it can be used for *practical assessment* by establishing bounds on the probability of system failure based on estimates of the probabilities of failure of the channels in the partitions only, which are typically estimable.

The models summarized above are applicable to on-demand software only, i.e. in which the individual demands are drawn independently from the demand space. Another line of research addressed the characteristics specific for control software, e.g. the fact that control software is typically executed on trajectories of inputs, which are not independently drawn from the input space. An important implication of trajectory based execution is failure clustering due to the fact that failure regions usually occupy ‘blobs’ of individual inputs, [14] , [3]. Modeling explicitly failure clustering was done in a number of studies, e.g. [15].

## 7 Conclusion and Future Work

We presented a model of system reliability of a 2-channel control software operating over partitions of the input space. The failure rates of the channels may vary over the partitions. The model reveals a useful insight – the probability of system failure may be significantly affected by the correlation of failure rates of the channels over the partitions – we recorded up to an order of magnitude difference in the mean time to systems failure between assessment ignoring the effect of failure rate variation and taking it into account. The result seems important because it suggest the need for more accurate reliability measurement than is usually undertaken.

We further considered a model of reliability for software with modular semi-Markov structure developed by Littlewood in the past and established that our model, although generally very similar, deviates from the mathematical description provided by Littlewood. Despite the deviations, however, similarly to Littlewood, we observed

that the time to system failure is exponentially distributed. Providing an analytical proof for the cases when the failures in partitions are not random (i.e. do not occur as Poisson processes) or identifying the cases when the system failure process ceases to be a Poisson process itself, is an open research problem.

We also discuss the issue of model parameter estimation. The theory of competing risks offers a suitable framework for parameter estimation using either maximum likelihood or Bayesian inference. Developing detailed assessment techniques with illustrative examples to help practitioners will be addressed in the future.

## References

1. Littlewood, B., Strigini, L.: Validation of Ultra-High Dependability for Software-based Systems. *Communications of the ACM* 36(11), 69–80 (1993)
2. Littlewood, B., Popov, P., et al.: Design Diversity: an Update from Research on Reliability Modelling. In: *Safety-Critical Systems Symposium 2001*. Springer, Bristol (2001)
3. Ammann, P.E., Knight, J.C.: Data Diversity: An Approach to Software Fault Tolerance. *IEEE Transactions on Computers* C-37(4), 418–425 (1988)
4. Bondavalli, A., Chiaradonna, S., et al.: Dependability Models for Iterative Software Considering Correlation among Successive Inputs. In: *IEEE International Symposium on Computer Performance and Dependability (IPDS 1995)*, Erlangen, Germany (1995)
5. PERFORM, Möbius: Model Based Environment for Validation of System Reliability, Availability, SEcurity and Performance. User's Manual, v. 2.0 Draft (2006)
6. Littlewood, B.: A Semi-Markov Model for Software Reliability with Failure Costs. In: *MRI Symposium on Computer Software Engineering*, pp. 281–300. Polytechnic Press (Available from Wiley, London), Polytechnic of New York, New York (1976)
7. David, H.A., Moeschberger, M.L.: The theory of competing risks. *Griffin's Statistical Monographs & Courses*, ed. D.S.E. Prof. Alan Stuart, vol. 39, p. 103 (1978)
8. Popov, P., Strigini, L., et al.: Estimating Bounds on the Reliability of Diverse Systems. *IEEE Transactions on Software Engineering* 29(4), 345–359 (2003)
9. Marshall, A.W., Olkin, I.: A generalised bivariate exponential distribution. *Journal of Applied Probability* 4, 291–302 (1967)
10. Nadarajah, S., Kotz, S.: Reliability for Some Bivariate Exponential Distributions. *Mathematical Problems in Engineering*, 2006, 1–14 (2006)
11. Eckhardt, D.E., Lee, L.D.: A theoretical basis for the analysis of multiversion software subject to coincident errors. *IEEE Transactions on Software Engineering* SE-11(12), 1511–1517 (1985)
12. Littlewood, B., Miller, D.R.: Conceptual Modelling of Coincident Failures in Multi-Version Software. *IEEE Transactions on Software Engineering* SE-15(12), 1596–1614 (1989)
13. Popov, P., Littlewood, B.: The Effect of Testing on Reliability of Fault-Tolerant Software. In: *Dependable Systems and Networks (DSN 2004)*. IEEE Computer Society Press, Florence (2004)
14. Bishop, P.G., Pullen, F.D.: PODS Revisited - A Study of Software Failure Behaviour. In: *18th International Symposium on Fault-Tolerant Computing*. IEEE Computer Society Press, Tokyo (1988)
15. Bondavalli, A., Chiaradonna, S., et al.: Modelling the effects of input correlation in iterative software. *Reliability Engineering and System Safety* 57(3), 189–202 (1997)

## Appendix

The material and the notation used here are based on [7].

Let  $C_l$  ( $l = 1, \dots, k$ ) denote the  $k$  competing risks or causes of failure. Let the random variable  $Y_i$  denote the individual length of life if  $Y_i$  were the only risk present with cdf  $P_i(x) = \Pr\{Y_i \leq x\}$  and pdf  $p_i(x)$ . When all risks are present, we can only observe the random variable,  $Z$ , defined as follows:  $Z = \min(Y_1, \dots, Y_k)$ .

Clearly, if  $Z$  exceeds  $x$ , then every  $Y_i$  exceeds  $x$ , too, i.e.:

$$\Pr\{Z > x\} = \Pr\{Y_1 > x, \dots, Y_k > x\}, \text{ which we denote as } \overline{F_Z}(x) = 1 - F_Z(x).$$

An important characteristic is the *hazard* rate defined as:  $r_Z(x) = \frac{f_Z(x)}{F_Z(x)}$ . The hazard

rates of the individual competing risks are defined similarly:  $r_i(x) = \frac{p_i(x)}{P_i(x)}$

For the case of independent risks, the total hazard rate is equal to the sum of the

hazard rates of the competing risks:  $r_Z(x) = \sum_{i=1}^k r_i(x)$ .

Let  $\pi_i$  be the probability that a failure is caused by risk  $C_i$ .

A related measure is the conditional probability  $\Pr\{Y_l = \min(Y_1, \dots, Y_k) \mid Z = x\}$ ,

which is defined by the ratio  $\frac{r_i(x)}{r_Z(x)}$ . If this ratio is a constant (so called proportional

hazard rates) the probability does not depend on the value of  $x$  and is equal to  $\pi_i$ . But

this is not always the case and in the general case  $\frac{r_i(x)}{r_Z(x)} \neq \pi_i$ .

If  $N_i$  individuals fail from cause  $C_i$ , and  $X_{ij}$  denotes the lifetime of the  $j$ -th individual failing from clause  $C_i$  ( $j=1, \dots, n_i; i = 1, \dots, k$ ), then the joint pdf of the  $X_{ij}$  is:

$$f(x_{11}, \dots, x_{1n_1}, \dots, x_{k1}, \dots, x_{kn_k}) = \prod_{i=1}^k \frac{1}{\pi_i^{n_i}} \prod_{j=1}^{n_i} p_i(x_{ij}) \prod_{\substack{l=1 \\ l \neq i}}^k \overline{P}_l(x_{ij}).$$

This is conditional on the random variables,  $N_i = n_i$  ( $i = 1, \dots, k$ ), which have

multinomial distribution:  $f(n_1, \dots, n_k) = \frac{n!}{\prod_{i=1}^k n_i!} \prod_{i=1}^k \pi_i^{n_i}$ , where  $n = \sum_{i=1}^k n_i$ . Hence, the

likelihood function of interest is:  $L = \frac{n!}{\prod_{i=1}^k n_i} \prod_{i=1}^k \prod_{j=1}^{n_i} p_i(x_{ij}) \prod_{\substack{l=1 \\ l \neq i}}^k \overline{P}_l(x_{ij})$ .

This expression is sufficient for one to apply either maximum likelihood for parameter estimation associated with the individual risks,  $Y_i$ , or Bayesian inference directly to the distributions of  $Y_i$ .

# Model-Driven Availability Evaluation of Railway Control Systems

Simona Bernardi<sup>1</sup>, Francesco Flammini<sup>2</sup>, Stefano Marrone<sup>3</sup>, José Merseguer<sup>4</sup>,  
Camilla Papa<sup>5</sup>, and Valeria Vittorini<sup>5</sup>

<sup>1</sup> Centro Universitario de la Defensa, Academia General Militar (Spain)  
`simonab@unizar.es`

<sup>2</sup> AnsaldoSTS, Innovation and Competitiveness Unit (Italy)  
`francesco.flammini@ansaldo-sts.com`

<sup>3</sup> Seconda Università di Napoli, Dip. di Matematica (Italy)  
`stefano.marrone@unina2.it`

<sup>4</sup> Dep.to de Informática e Ingeniería de Sistemas, Universidad de Zaragoza (Spain)  
`jmerse@unizar.es`

<sup>5</sup> Università di Napoli “Federico II”, Dip. di Informatica e Sistemistica (Italy)  
`{camilla.papa, valeria.vittorini}@unina.it`

**Abstract.** Maintenance of real-world systems is a complex task involving several actors, procedures and technologies. Proper approaches are needed in order to evaluate the impact of different maintenance policies considering cost/benefit factors. To that aim, maintenance models may be used within availability, performability or safety models, the latter developed using formal languages according to the requirements of international standards. In this paper, a model-driven approach is described for the development of formal maintenance and reliability models for the availability evaluation of repairable systems. The approach facilitates the use of formal models which would be otherwise difficult to manage, and provides the basis for automated models construction. Starting from an extension to maintenance aspects of the MARTE-DAM profile for dependability analysis, an automated process based on model-to-model transformations is described. The process is applied to generate a Repairable Fault Trees model from the MARTE-DAM specification of the Radio Block Centre - a modern railway controller.

**Keywords:** Automated Model Generation, ERTMS/ETCS system, Model Transformation, Repairable Fault Trees, UML profiles.

## 1 Introduction

The development of mission-critical systems has to tackle several challenges, including the evaluation of RAMS (Reliability, Availability, Maintainability, Safety) attributes since early stages of system life-cycle till the possible final certification phase. Evaluation approaches by means of formal models have proven to be effective in assessing RAMS attributes. However, many formalisms (e.g., Fault Trees) suffer from limited expressive power when dealing with complex repairable

systems, or they are limited in usability and solving efficiency (e.g., Stochastic Petri Nets). To solve those issues, recently some “hybrid” approaches have been proposed, trying to combine the advantages of different formalisms [10].

One further step toward the simplification of the model-based RAMS evaluation is the use of high-level modeling languages, derived from the Unified Modeling Language (UML), and model transformation, which is the basis of the Model Driven Engineering (MDE) methodology. Model transformations processes transform a source model into a target model based on transformation rules [18]. A first attempt of defining such a process, in the dependability context, has been performed within the HIDE project [3]. More recently, specific profiles have been developed to specify non-functional properties (NFP) on UML diagrams, such as the OMG standard MARTE (Modeling and Analysis of Real-Time and Embedded Systems) UML profile [20].

An important aspect of the MDE related work is the availability of open source case tools and workbenches. These MDE platforms integrate tools which support the main technologies to implement Model-to-Model (M2M) transformations, such as ATL [11] or QVT [14]. The integration of formal methods and techniques into MDE based development process is still an open issue. MARTE introduces the possibility of annotating models in order to cope with NFPs but it does not provide support to dependability analysis. A recent work [2] proposes an extension of MARTE for dependability and modeling (MARTE-DAM). Nevertheless MARTE-DAM does not define M2M transformations for the automated generation of formal analysis models from MARTE artifacts.

In the past, several research efforts have focused on the derivation of formal models from UML diagrams, as surveyed in [2]. As specifically regards fault trees (FT) and their extensions, Pai and Dugan developed a method to derive Dynamic Fault Trees from UML system models [15] and D’Ambrogio et al. defined a method to generate FT models from a set of sequence diagrams in order to predict software reliability [5]. All these works are partial solutions to the problem and their systematic application appears to have been absent in the subsequent evolution of UML.

In this paper, we integrate the above mentioned approaches: we apply model-driven techniques to generate formal models of critical systems. Starting from a high level specification of the system expressed by an extended UML profile, we define and implement proper M2M transformations in order to automate the generation of availability models of a modern railway controller. The contribution of the paper is twofold: on the one hand, it shows how a model-driven approach may also promote the applicability of formal modeling in industrial settings; on the other hand, we extend the MARTE-DAM profile for dependability modeling and define the M2M transformations to generate Repairable Fault Tree [4] models from the extended profile. In particular, considering MARTE-DAM, we enrich the fault tolerance and maintenance aspects of the profile [2] to enable the specification of complex repairable systems.

It is worth to mention that, beside UML, SysML [19] and AADL [1] have been also considered as source specification languages in M2M transformations.

The work [8] proposes a joint use of UML-MARTE and SysML for the automatic generation of certification-related information in safety domain. Indeed, although SysML provides support to manage requirements and system design together, it lacks of standard concepts for dealing with specific dependability concerns. On the other hand, AADL enables the specification of dependability requirements and properties of software systems. The works [17] and [9] both propose transformation techniques to get formal dependability models from AADL specifications (respectively, Generalized Stochastic Petri Nets and probabilistic finite state-machines).

## 2 The MARTE-DAM Profile

The MARTE [20] profile provides a *lightweight* extension of UML (i.e., through the use of stereotypes, tagged-values and constraints) to specify system non-functional properties (NFPs), according to a well-defined Value Specification Language (VSL) syntax. Stereotypes extend the semantics of UML meta-classes with concepts from the target domain. They are made of tags whose types can be basic UML types (e.g., integer) or MARTE NFP types (e.g., *NFP\_Integer* in Tab. I). The latter are of special importance since they enable the description of relevant aspects of a NFP using several properties, such as *value*, a value or parameter name (prefixed by the dollar symbol); *source*, the origin of the NFP (e.g., a requirement - *req*) *statQ*, the type of statistical measure (e.g., mean).

The “Dependability Analysis and Modeling” (DAM) [2] profile is a MARTE specialization. A MARTE-DAM annotation *stereotypes* a UML design model element, then extending its semantics with dependability concepts (e.g., annotating a UML State Machine transition as a failure step). Moreover, DAM enriches the MARTE types with basic and complex dependability types. The latter (e.g., *DaRepair*) are composed of attributes (e.g., *MTTR*) that can be MARTE NFP types (e.g., *NFP\_Duration*) or simple types.

The DAM profile relies on the definition of the DAM *domain model* which represents the main dependability concepts from the literature according to a component-based view of the system to be analyzed [7]. In the domain model, the system is defined by a set of *components* bounded together through *connectors*, in order to interact. The system delivers a set of high-level *services*, that can be detailed - at finer grained level - by a sequence of *steps*, representing states of components, events or actions. The system can be affected by threats, i.e., *faults*, *errors*, *failures*. A fault is the original cause of errors and it affects system components. Errors are related to steps and they can be propagated from the faulty component to other components it interacts with. Errors may cause failures at different levels: at step level, when the service provided by the component becomes incorrect; at component level, when the component is unable to provide service; at service level, when the failure is perceived by external users.

The domain model includes also redundancy and maintenance concepts. The *Redundancy* model (Figure I) represents UML hw/sw *redundant structures* to

increase system fault tolerance (FT). These structures are made of components, among them *FT components* [12], which can play different roles.

The *Maintenance* model (Figure 2) concerns repairable systems and includes concepts that are necessary to support the evaluation of system availability, that is the *maintenance actions* undertaken to restore the system affected by threats. According to [7], we distinguish *repairs* of system components, that involve the participation of external agents (e.g., repairman, test equipment, etc) and *recovery* of services, which do not require the intervention of the latter.

In this paper, we aim at increasing modeling and analysis capabilities of DAM regarding redundancy and maintenance, as explained in the next Section.

### 3 A DAM Extension for Maintenance and Fault Tolerance

The DAM domain models of redundancy and maintenance provide the basis for the definition of proper extensions, i.e., stereotypes, tagged-values and OCL constraints. In particular, the stereotypes and tagged values will be used to annotate UML designs with fault tolerance and maintenance requirements/properties, while OCL constraints are assigned to UML extensions to guarantee UML annotations compliant to the DAM domain concepts.

The extension of DAM domain model consists in: 1) augmenting the maintenance model w.r.t. the one presented in [2] (grey classes in Figure 2 account for the extension). *Activation steps* initiate maintenance actions as consequence of component failures. An activation step defines its *priority* as well as a *preemption* policy, moreover it relates to a group of agents with the required *skills* to perform the step. The activation step also relates to the failures that caused it; 2) improving the redundancy model (Figure 1), keeping its original shape and only adding the *FTlevel* attribute in the redundant structure stereotype.

The stereotypes associated to the redundancy and maintenance domain models are shown in Table 1 (first column) and they corresponds to concrete classes in Figs. 1 and 2. For reason of space, we omit stereotypes that come from classes defined in the core domain model (e.g., component, service and step). A tag of a stereotype (Table 1, third column) can be derived, together with its multiplicity, from one of the following sources in the domain model: 1) an attribute of the corresponding class, i.e., the attribute *errorDetCoverage* in *Adjudicator* class (Fig. 1) has been mapped onto the tag *DaAdjudicator::errorDetCoverage*; 2) an association-end role, i.e., the *substituteFor* role in the association between *Spare* and *Component* classes (Fig. 1) has been mapped onto the tag *DaSpare::substitutesFor*. The types of the tags can be either simple types (e.g., the enumeration *skillType* assigned to the *skill* tag of the *AgentGroup* stereotype), MARTE-NFP types (e.g., *NFP\_Integer*), or complex dependability types. The latter are data types derived from classes in the domain models, they are characterized by a set of attributes corresponding to the ones of the mapped classes. Basically, they may represent either threat characterization (e.g., the

**Table 1.** Stereotypes and tags

<i>Stereotype</i>	<i>Inherits / Extends</i>	<i>Tags: type</i>
<b>Redundancy</b>		
DaAdjudicator	DAM::DaComponent	errDetCoverage: NFP_Percentage[*]
DaController	DAM::DaComponent	<i>none</i>
DaRedundantStructure	/ UML::Package	commonModeF: DaFailure[*] commonModeH: DaHazard[*] FTlevel: NFP_Integer[*]
DaSpare	DAM::DaComponent	dormancyFactor: NFP_Real[*] substitutesFor: DaComponent[1..*]
DaVariant	DAM::DaComponent	multiplicity: NFP_Integer[*]
<b>Maintenance</b>		
DaAgentGroup	/ UML::Classifier (e.g., Actor, Class)	skill: skillType correctness: NFP_Real[*] agentNumber: NFP_Integer[*]
DaActivationStep	DAM::DaStep	kind: {activation} preemption: NFP_Boolean[0..1] cause: DaStep[1..*] = (kind=failure) agents: DaAgentGroup[1..*]
DaReallocationStep	DAM::DaStep	kind: {reallocation} map: DaComponent[1..*] onto: DaSpare[1..*]
DaReplacementStep	DAM::DaStep	kind: {replacement} replace: DaComponent[1..*] with: DaSpare[1..*]

class *Failure* has been mapped onto the *DaFailure* complex type) or concrete maintenance actions (e.g. the *DaRepair* complex type corresponds to the class *Repair*).

Given a UML model of the system under analysis, the main issue - from the software engineer point of view - is which model elements in a UML diagram (e.g., a state in a state-machine diagram or a component in a component diagram) can be stereotyped in order to specify NFPs through tagged-values. As shown in Table 1 (second column), each stereotype may either specialize a previously defined MARTE-DAM stereotype or directly extend a UML meta-class. Then, a given model element can be stereotyped as *X* if the stereotype *X* eventually extends the meta-class the former belongs to (either directly or indirectly, through stereotype generalization). For example, all the sub-stereotypes of *DaComponent* can be applied to UML elements representing system software and hardware resources (e.g., classes, instances, components, nodes), since *DaComponent* specializes the MARTE *Resource* stereotype and the latter extends the corresponding UML meta-classes. On the other hand, the different *step* stereotypes (e.g., *DaReallocation*, *DaReplacement*, *DaActivation*) inherit from *DaStep*, which can be applied to a wide set of behavior-related elements, such as messages in sequence diagrams, and transitions, state, trigger events, effect actions in state-machine diagrams. Finally, the stereotypes *DaRedundantStructure* and *DaAgentGroup* directly extend the *Package* and *Classifier* UML meta-classes, respectively. While former can be applied to package elements, the latter can be applied to different kind of structure-related elements, such as actors in use case diagrams and classes in class diagrams.



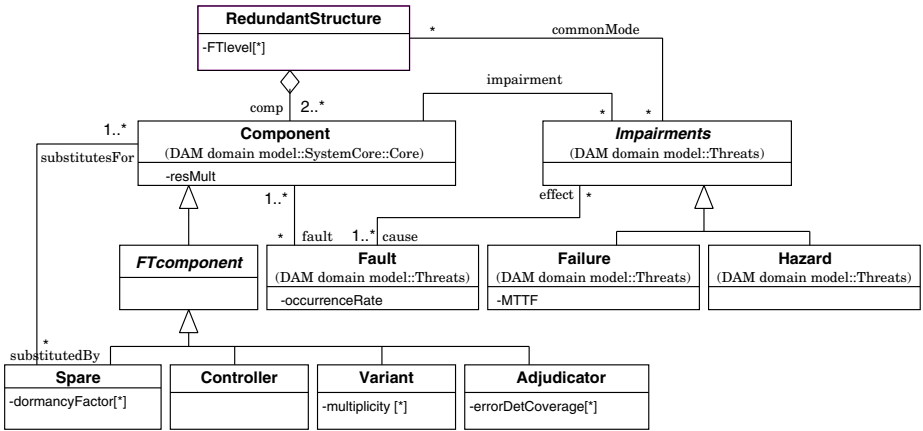


Fig. 1. Redundancy domain model

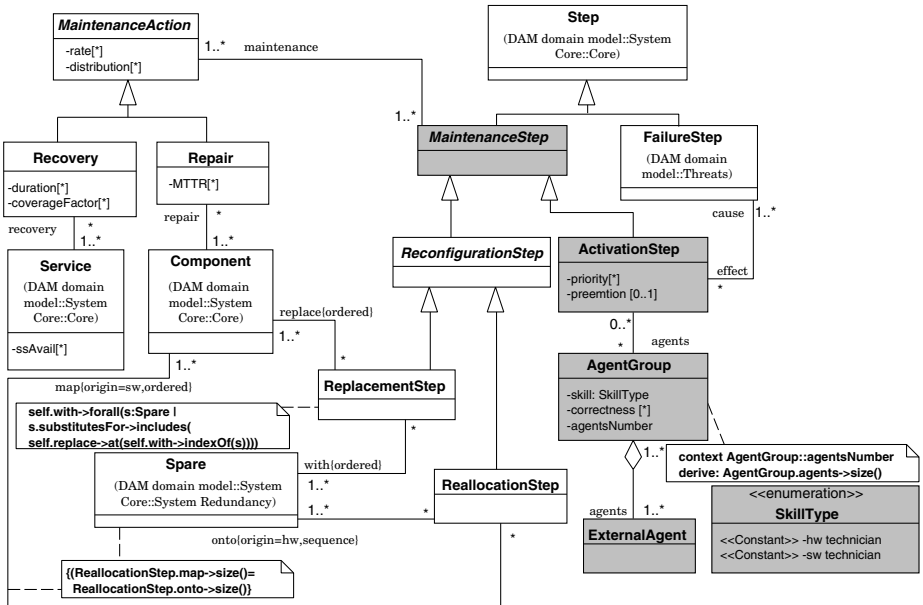


Fig. 2. Maintenance domain model

## 4 Automated Generation of RFT Models

Automated generation of formal models from UML models has already been studied and such technique can be considered part of the process schema depicted in Fig. 3. The first step is the definition of a design UML model by system designer. At this level dependability aspects have been not considered

yet. The next step is constituted by the application of MARTE-DAM profile to the UML model and the definition of dependability parameters that characterize the system. Then a MARTE-DAM model can be automatically translated into a formal model: this translation is conducted by means of model-to-model transformations that are defined on the base of a source and a destination metamodels (i.e. the formalizations of the languages in which source and destination models are expressed). Generated formal models can be finally analyzed, allowing the validation of the model or its eventual refining by parameters tuning and/or redefinition of the architecture. The formal language should be properly chosen according to the specific dependability aspect to be analyzed.

Since we are focusing on maintenance, source formalism is constituted by MARTE-DAM (containing the extension introduced in the previous Section) and destination formalism is the Repairable Fault Tree (RFT) that was introduced to ease the modeler's approach to complex repair policy modeling and evaluation [4]. The RFT formalism [4] integrates GSPNs and Fault Trees: repair policies are represented by nodes - called Repair Boxes - which encapsulate GSPN nets, and a Fault Tree describes the faults that may happen and their contribution to the occurrence of a failure. The Repair Box connected to a Fault Tree event models a repair action that can be performed on the related system sub-component. RFT metamodel is given in Figure 4 while the DAM meta-model has been described in Section 2 and Section 3.

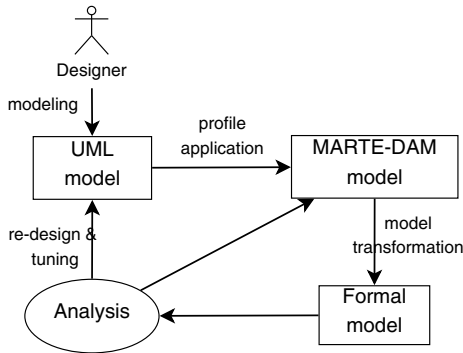


Fig. 3. A reference model-driven process

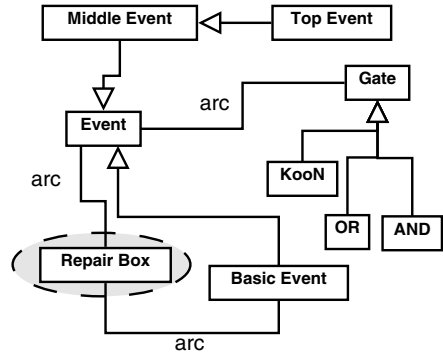
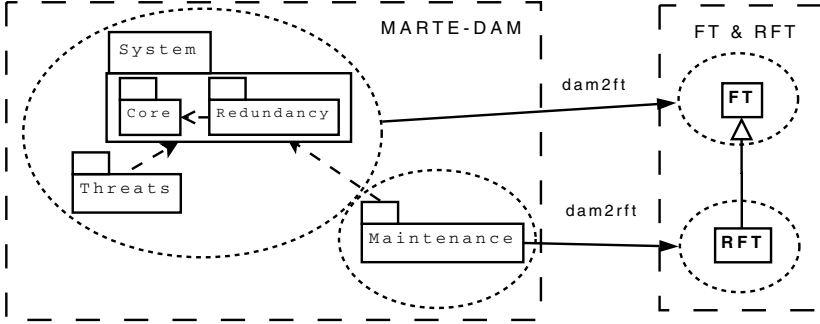


Fig. 4. RFT meta-model

Defining complex model transformations from scratch can be a hard task, so transformations composition and reuse are being widely investigated. In this paper we apply **module superimposition**, a widespread mechanism for coarse-grain composition of transformations which shifts the focus of reuse from rule to set of rules (transformation modules). In practice, superimposition allows for defining a new transformation by the union of the rules set of existing ones. Superimposition is well supported by the most important transformation languages (including ATL). Compositional approaches are enhanced by inheritance relationships between languages. In particular, the RFT language is an extension

of FTs obtained by adding the Repair Box element: the hierarchical nature of this formalism and its implications on building model transformations by composition has been already studied in [13]. On the other hand, MARTE-DAM has a decoupled structure due to the dependency relations among packages, as depicted in Figure 5. Hence, M2M transformations from MARTE-DAM to RFT may benefit from a *divide-et-impera* approach. According to the above considerations, the two transformations described in Figure 5 have been defined and implemented in ATL.



**Fig. 5.** DAM-to-RFT transformation schema

Starting from a DAM specification, *dam2ft* generates the Fault Tree from the System and Threats domain sub-models, and *dam2rft* adds the RBs and related arcs from the Maintenance domain sub-model.

The transformation implemented by *dam2ft* works as follows: the Top Event is associated to the failure of the system which provides the service (specified by the use case diagram). From the component diagram, events and gates are created by recursively applying the following rules:

1. *DaComponent* is translated into: a) one Middle Event or b) one Middle Event and as many Basic Events as specified by the *resMult* value (i.e., resource multiplicity) if the *fault* tagged value is not null;
2. *DaSpare* is translated into as many Basic Events as specified by *resMult* value if the *fault* tagged value is not null;
3. An input gate is generated for each Middle Event: an OR gate if *DaComponent* does not belong to a *DaRedundantStructure*, an AND gate if *DaComponent* belongs to a *DaRedundantStructure* with *FTlevel*=1, while a KooN gate is generated if *FTlevel*> 1;
4. An input arc is generated to specify an input Middle Event of gate if a sub-component relationship exists between *DaComponent* associated to the Middle Event and the one associated to the output Middle Event of the gate;
5. An arc is always generated from a Basic Event to a gate whose output Middle Event comes from *DaComponent* (point 1.b)) or from *DaComponent* substituted by *DaSpare* (point 2).

The transformation implemented by *dam2rft* works under the hypothesis that the DAM specification includes a model of the repairing process of a sub-component and information about its steps (this can be expressed by means of a state chart diagram, an activity diagram, or a sequence diagram). The existence of a repair model associated to a *DaComponent* is annotated by a *DaActivationStep* stereotyped element. First a RB is generated for each replica of *DaComponent* and of its *DaSpare*, if any. The RB is filled with information about the MTTR and the resources (the repairmen) needed to accomplish the activity. These information are retrieved by the diagram used to describe the repair dynamics and by navigating the component diagram.

Finally, each RB is connected to the Fault Tree generated by *dam2ft* through repair arcs: 1) between RB and its triggering event, i.e. the Middle Event or Basic Event from which the RB has been generated (the sub-system to be repaired); 2) between RB and all the Basic Events that are present in the sub-tree whose root is the RB triggering event. Once RFT model has been generated, a Generalized Stochastic Petri Nets model is derived by applying another M2M transformation in order to allow easy analysis of this formal model. The description of this last transformation is reported in [13].

## 5 The Radio Block Centre

The Radio Block Centre (RBC) is the vital core of the European Railway Traffic Management System / European Train Control System (ERTMS/ETCS) which is the reference standard of the new European railway signalling and control systems [6] ensuring the safe running of trains on different European railway networks. RBC is a computing system which controls the movements of the set of trains traveling across the track area under its supervision. At this aim, RBC elaborates messages to be sent to the trains on basis of information received from external trackside systems and on basis of information exchanged with on-board sub-systems. The unavailability of a RBC is critical, as there is no way for the signalling system to work without its contribution. In case of a RBC failure, all the trains under its supervision are compelled to brake and proceed in a staff responsible mode. This would lead to the most critical among the ERTMS/ETCS safe failures, that is the so called Immobilising Failure<sup>1</sup>. The ERTMS/ETCS standard requires compliance with the RAM requirements [16] whose fulfillment has to be properly demonstrated. Specifically, the quantifiable contribution of RBC system to operational unavailability must be not more than  $10^{-6}$  (see [16], §2.3.3). The standards do not impose constraints on the system architecture. Hence, different implementations are possible. A reference architecture of RBC must exhibit a high level of redundancy to improve the fault tolerance of the system. In this paper the system consists of three commercial CPU-RAM cards and a redundant FPGA based voter in a TMR (Triple Modular Redundancy) configuration. The GSM-R and WAN communication sub-systems

<sup>1</sup> An Immobilising Failure occurs when at least two trains are no more under ERTMS/ETCS supervision [16].

are also chosen as COTS (Commercial Off The Shelf). The RBC configuration in figure is completed by three commercial power supplies and a redundant standard backbone (used as system BUS).

Maintenance policies are a fundamental aspect of RBC life-cycle for their impact on system availability. ERTMS/ETCS gives no restrictive requirements for the maintainability parameters and this leaves much freedom in designing repair policies. Of course, it must be proved that the system still meet the availability requirement. The rest of this Section applies introduced modeling and transformational approach to the RBC case study. We limit our study to the hardware contribution to availability: as MARTE-DAM can be applied on both hw and sw UML models, we could apply this process on software systems.

## 5.1 DAM Model

The DAM specification of RBC used to generate the RFT consists of an use case diagram, a component diagram and a set of state chart diagrams. The diagrams are annotated with the DAM extensions introduced in Section 3.

The use case diagram in Figure 6 represents the main functionality of the RBC: the train outdistancing. The use case is stereotyped *DaService* to explicitly indicate (by the *usedResources* tagged value) which is the component in charge of providing the service, hence identifying the source of the failures that may cause a service interruption. The availability requirement is captured by the *ssAvail* tagged-value.

The actor stereotyped *DaAgentGroup* represents the set of hardware technicians (*skillType* tagged-value) who participate in the repair process; here two technicians (*agentsNumber*) are assumed to accomplish repair activities correctly (*correctness*).

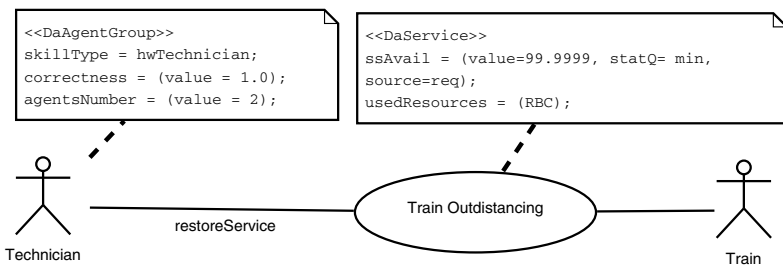
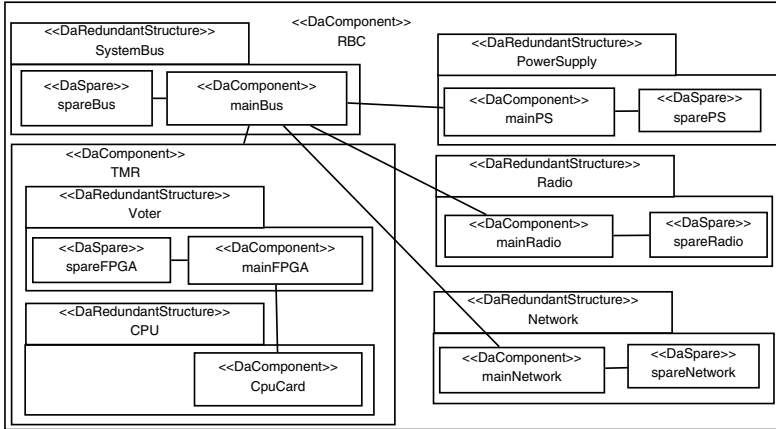


Fig. 6. Train Outdistancing Use Case

The component diagram in Figure 7 provides a high level description of the RBC components whose failures affect the system dependability. The main hardware components of the RBC system are stereotyped *DaComponent*: they can be either simple components (e.g., *MainBus*) or components with an internal structure (e.g., *TMR*). Each redundant sub-system is represented by a package stereotyped *DaRedundantStructure* (e.g., *SystemBus*) which includes several instances of the same hardware component: the *DaComponents* are the active



**Fig. 7.** RBC Component Diagram

replicas (e.g., `mainBus`), whereas the other components are stereotyped *DaSpare* (e.g., `spareBus`).

A detailed view of the `SystemBus` redundant sub-system is shown in Figure 8, where several tagged-values associated to the stereotyped elements have been specified:

- The *DaRedundantStructure* requires at least one operative component, either main or spare one, to guarantee the `SystemBus` functionality (*FTlevel* tagged-value);
- The `SystemBus` includes one main *DaComponent* bus instance and one *DaSpare* bus instance (*resMult* tagged-values);
- The *DaSpare* bus substitutes for the main bus, in case of failure of the latter (*substituteFor* tagged-value);
- Both the main and the spare buses are characterized by fault occurrence rate (*fault.occurrenceRate*) and Mean Time To Repair (*MTTR*) values;

Finally, the RBC specification includes several State Chart diagrams (SC), one for each repairable component. The SC models the dynamics of the repair process of a specific component. In the RBC system, a diagnostic mechanism is present for three components (specifically, RBC, `mainPS` and `CpuCard`); when one of the latter fails, a repair process may start. The three SCs have a common structure, Figure 9 shows the SC of the `CpuCard`. In particular, the transitions are stereotyped as DAM steps. The *DaStep* transition, triggered by the CPU `fail` event, models the failure occurrence step (*kind* tagged-value), and leads the component from the `running` to the `failed` state. The *DaActivationStep* transition occurs when the activation of a repair action becomes enabled; it specifies the number of agents needed to perform the repair (*agentsNumber* tagged-value) as well as the required repair skills (*agentSkill*). The *DaReplacementStep* transition models the step of replacing the failed component, then restoring the service.

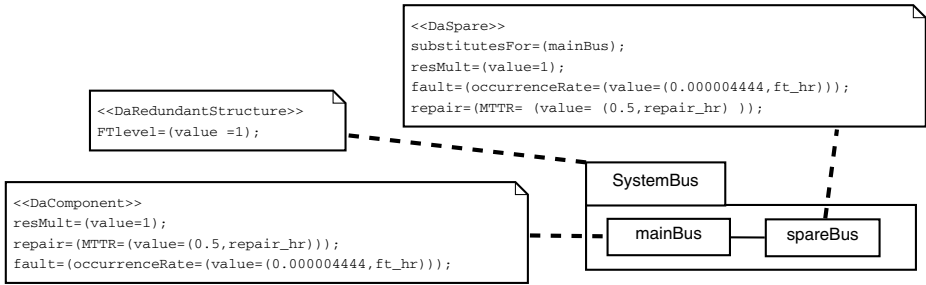


Fig. 8. System Bus

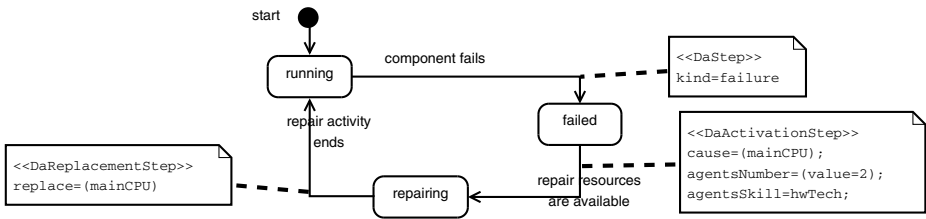


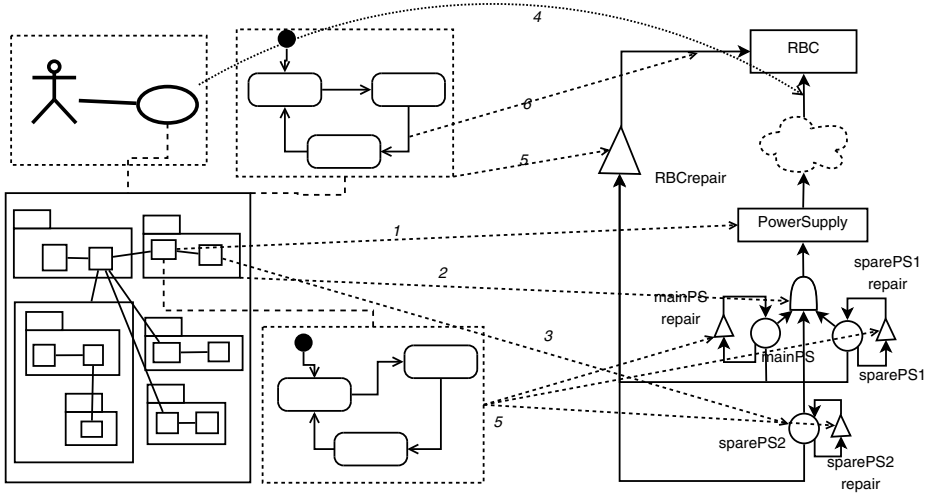
Fig. 9. State Chart Diagram

## 5.2 Generation of the RFT Model

The process by which transformations generate the RFT model is depicted in Figure 10 where only a part of the resulting RFT is shown, specifically the sub-tree obtained by translating the *PowerSupply* package.

First *dam2ft* rules are applied to the source model: they are represented by the dotted lines labeled from 1 to 4. Then the *dam2rft* transformation is applied (rules labeled 5 and 6). Rule 1 is applied to the *DaComponent* stereotyped elements of *PowerSupply* in the RBC component diagram and generates a Middle Event for each of them, hence in this case Rule 1 generates the *PowerSupply* FT event. Rule 2 generates the AND input gate because *FTlevel*=1, as described in Section 4. Rule 3 generates three Basic Events, where three is the number of replicas of *SparePS* (2) plus the number of *mainPS* (1). From the use case diagram, Rule 4 identifies the *DaComponent* representing the system to analyze (*DaService*) and generates gate-to-event arcs by recursively looking for sub-components relations.

Rules 5 is triggered by the *DaActivationStep* transitions present in the state charts, it generates one RB for the RBC component and three RBs from the *PowerSupply* package. These RBs are filled with relevant data (MTTR, necessary resources) by extracting maintenance related information from the state chart diagram and the component diagram. Rule 6 links RBs and events by recursive exploration of the sub-tree.



**Fig. 10.** The RBC Repairable Fault Tree generation

Once the RFT model has been generated it can be solved to perform the availability analysis and efficiently evaluate the probability of an immobilizing failure in presence of different repair policies. The solution process of RFT models is described in [4], the results of the availability analysis of RBC are reported in [10], where an hand-made RFT is proposed.

## 6 Conclusions and Future Work

In this paper, we have presented an enhancement of the MARTE-DAM profile in order to improve the capability of this profile to model complex repairable systems. We have also proposed an approach integrating DAM models and Repairable Fault Trees by means of M2M transformations. The suitability of the profile extension and the proposed transformations has been proved on the real case study of the Radio Block Centre. Next steps in this activity will include the development of meta-modeling, modeling and transformational techniques able to fully generate complex repair policies.

## References

1. SAE-AS5506/1 Architecture Analysis and Design Language Annex (AADL): Vol.1, annex E:Error Model, International Society of Automotive Engineers (2006)
2. Bernardi, S., Merseguer, J., Petriu, D.C.: A Dependability Profile within MARTE. *Journal of Software and Systems Modeling* (2009)
3. Bondavalli, A., Latella, D., Dal Cin, M., Pataricza, A.: High-Level Integrated Design Environment for Dependability (HIDE). In: *Proceedings of the Fifth International Workshop on Object-Oriented Real-Time Dependable Systems, WORDS 1999*, pp. 87–92. IEEE Computer Society, Washington, DC, USA (1999)



4. Codetta Raiteri, D., Iacono, M., Franceschinis, G., Vittorini, V.: Repairable fault tree for the automatic evaluation of repair policies. In: Proceedings of the 2004 International Conference on Dependable Systems and Networks, pp. 659–668. IEEE Computer Society, Washington, DC, USA (2004)
5. D’Ambrogio, A., Iazeolla, G., Mirandola, R.: A method for the prediction of software reliability. In: Proc. of the 6-th IASTED Software Engineering and Applications Conference, SEA 2002 (2002)
6. ERTMS/ETCS System Requirements Specification (SRS), SUBSET-026, Issue 3.0.0 (2008)
7. Avizienis, A., et al.: Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. on Dependable and Secure Computing* 1(1), 11–33 (2004)
8. Cancila, D., et al.: SOPHIA: a modeling language for model-based safety engineering. In: 2nd International Workshop on Model Based Architecting and Construction of Embedded Systems, Denver, Colorado, USA, October 6, pp. 11–26. CEUR (2009)
9. Bozzano, M., et al.: Safety, dependability and performance analysis of extended AADL models. *The Computer Journal* 54(5), 754–775 (2011)
10. Flammini, F., Mazzocca, N., Iacono, M., Marrone, S.: Using repairable fault trees for the evaluation of design choices for critical repairable systems. In: IEEE International Symposium on High-Assurance Systems Engineering, pp. 163–172 (2005)
11. Jouault, F., Kurtev, I.: On the architectural alignment of ATL and QVT. In: Proceedings of the 2006 ACM Symposium on Applied Computing, SAC 2006, pp. 1188–1195. ACM, New York (2006)
12. Lyu, M.R.: *Software Fault Tolerance*. John Wiley & Sons, Ltd., Chichester (1995)
13. Marrone, S., Papa, C., Vittorini, V.: Multiformalism and transformation inheritance for dependability analysis of critical systems. In: Méry, D., Merz, S. (eds.) IFM 2010. LNCS, vol. 6396, pp. 215–228. Springer, Heidelberg (2010)
14. MOF Query/Views/Transformations. Final Adopted Spec., ptc/05-11-01 (2005)
15. Pai, G.J., Dugan, J.B.: Automatic Synthesis of Dynamic Fault Trees from UML System Models. In: Proceedings of the 13th International Symposium on Software Reliability Engineering, pp. 243–254. IEEE CS, Washington, DC, USA (2002)
16. ERTMS/ETCS RAMS Requirements Specification. Ref. 96s1266 (1998)
17. Rugina, A.-E., Kanoun, K., Kaaniche, M.: A system dependability modeling framework using AADL and GSPNs. In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) Architecting Dependable Systems IV. LNCS, vol. 4615, pp. 14–38. Springer, Heidelberg (2007)
18. Sendall, S., Kozaczynski, W.: Model transformation: The heart and soul of model-driven software development. *IEEE Softw.* 20, 42–45 (2003)
19. Systems Modeling Language. SysML, <http://www.sysml.org>
20. UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE), Version 1.0, OMG document formal/2009-11-02 (November 2009)

# Vertical Safety Interfaces – Improving the Efficiency of Modular Certification\*

Bastian Zimmer<sup>1</sup>, Susanne Bürklen<sup>2</sup>, Michael Knoop<sup>2</sup>,  
Jens Höfflinger<sup>2</sup>, and Mario Trapp<sup>1</sup>

<sup>1</sup> Fraunhofer Institute for Experimental Software Engineering, Fraunhofer-Platz 1,  
67663 Kaiserslautern, Germany

<sup>2</sup> Robert Bosch GmbH, Postfach 30 02 40, 70442 Stuttgart, Germany

**Abstract.** Modular certification is a technique for transferring the modularity of an embedded system’s architecture to the traditionally monolithic craft of safety engineering. Particularly when applying integrated architectures like AUTOSAR or IMA, modular certification allows the construction of modular safety cases, which ensures the flexible handling of platforms and applications. However, the task of integrating these safety cases is still a manual and expensive endeavor, lowering the intended flexibility of an integrated architecture. We propose a tool-supported semi-automatic integration method that preserves the architecture’s flexibility and helps to lower the integration costs. Our method is based on a language capable of specifying the conditions for a valid integration of a platform and of an application using a contract-based approach to model safety case interfaces. This paper presents the language in detail.

## 1 Introduction

Over the course of the last ten years integrated architectures like IMA and AUTOSAR have been gaining ground. An integrated architecture considers general purpose execution platforms and function specific applications as modular pieces of an embedded system. This facilitates extending and changing the system and consequently increases the system’s flexibility. However, when developing safety-critical systems, the monolithic certification approach of most safety standards counteracts this flexibility, because changes usually lead to expensive re-certifications.

Modular certification enables pre-certifying components independently of their later usage scenario [19]. This allows the system integrator to exchange components along with their modular certificates, reducing the certification costs through reuse. In order to modularly reason about the safety of his product, the developer of a component typically has to specify demands on the behaviour of other components. As a result, the system integrator has to check if these

---

\* This work was funded by the German Federal Ministry of Education and Research (BMBF), grant SPES2020, OIIS08045I.

demands are met before he is able to use the certificates. This certificate composition can become a costly job, especially if the components are developed by different organizations, using diverse methods and vocabulary to define the certificate dependencies. Since integration costs are incurred at every change of the system, they pose a threat to flexibility. To lower these costs, we propose a semi-automated integration method.

As the objective of the integration is to check whether all demands are fulfilled, any kind of automation requires a formalized description of the dependencies specified at the certificate interface. A formal description covering all possible interfaces would have to be very expressive, since a large variety of safety-related dependencies can exist among the modules of a system. Still, feasibility seems to be much better by narrowing the objectives and seeking to cover only the interface between an application and its platform, from now on referred to as the vertical interface. Compared to the horizontal interface, covering the dependencies between applications, a vertical interface is much more regular. In case of AUTOSAR [2] and the ARINC 653 [3] Integrated Modular Avionics (IMA) standard, the services of a platform are even standardized. This provides reason to believe that the dependencies at the interface of the certificates can be standardized to a large extent as well.

This paper presents the Vertical Safety Interface (VerSaI) language. The VerSaI language is a meta-model based formalization of the typical dependencies between the certificates of an application and a platform. The language uses the notion of contracts, comprised of demands and guarantees.

This paper is structured as follows. In the second chapter we give an overview of the related work. The third chapter is dedicated to the introduction of our running example. The example is used to illustrate our approach described in chapter four. Chapter five contains an evaluation of the applicability of the approach from an industrial viewpoint. We summarize the paper with a conclusion and a description of possible future work.

## 2 Related Work

The notion of contracts is widely used in computer science, to model the dependencies between interacting elements of a system. It is, for example, used in formal verification by Jones [13], and in the area of Object Oriented Programming by Meyer [16]. Comparable contract-based approaches have also been used in modular certification, to capture the dependencies between the certificates of system components, for example by Rushby [19]. His approach extends assume-guarantee reasoning known from verification so it can capture failure behavior and operates on the horizontal level, describing dependencies between applications.

Motivated by preserving the flexibility of IMA systems, Bate and Kelly [5] propose the use of modular safety cases for the certification of modular systems. The approach introduces a safety case structure that aligns with the architecture of the system, in order to reduce rework in case of change. The safety case

architecture contains cases for platforms and application and describes vertical as well as horizontal dependencies. The dependencies between modular safety cases are expressed by a contract-based method [11] using relies and guarantees and is based on the Goal Structuring Notation (GSN) [14]. A method for analysing an IMA system in order to establish suitable contracts is provided by Conmy et. al. [7].

Our work is based on the afore-mentioned approaches and pursues the idea of reducing re-certification costs further, by providing a formalized language to specify contracts between an application and a platform and therewith, lay the ground for a future automation. One cornerstone of our approach is the integration of the language into the design models of the system. This follows an apparent trend in the are of modular safety analyses, regarding approaches like CFTs [10], Hip-HOPS [17], ConSerts [20] or SaveCCM [12]. But also in the area of modular certification there have been examinations regarding the utilization of model-driven techniques. Conmy and Paige [8] concluded, for example, that the use of the Model Driven Architecture (MDA) has the potential to facilitate modular certification.

There has also been research about the tightly coupled use of contracts and design models, like the development of the HRC framework [9] or the work done in [4]. These approaches have a focus on supporting the design and development of safety-critical systems, but do not directly touch the subject of certification.

### 3 Running Example

The example under consideration is an excerpt of an automotive integrated system, comprising one application and one platform. The application provides the simplified functionality of a fictive cruise control (CrCtl). The task of a cruise control is to control the car’s velocity to match it to a desired velocity, previously set by the driver. The notation used to illustrate the application is a slightly adapted version of the notation used in AUTOSAR as shown in figure 1. The application consists of four software-components called *ui\_sensor*, *brakePedal\_sensor*, *crctl\_main* and *crctl\_monitor*.

The job of *ui\_sensor* and *brakePedal\_sensor* is to read and process raw sensor data, and to provide the inputs as utilizable communication signals. The component *crctl\_main* uses these signals together with signals provided by other applications (like the car’s current velocity (*vCur*)) to calculate the desired acceleration (*aSet*).

As a starting point to reason about the safety of the system we considered two typical safety goals:

**Safety Goal 1.** After an operation of the brakes, the CrCtl must deactivate within 300ms (ASIL x)

**Safety Goal 2.** The demanded acceleration of the CrCtl must never lead to a destabilization of the vehicle (ASIL y)

To achieve these safety goals, we specified the following exemplary safety concept: For the fulfillment of the second goal, the CrCtl needs the aid of the car’s electronic stability control (ESC). Under the assumption that the ESC is preventing the destabilization of the car, both safety goals can be achieved by transitioning into the application’s safe state, the inactive state. This safety strategy is jointly implemented by the software components *crctl\_main* and *crctl\_monitor*. If the former notices the activation (*escActive*), or the unavailability (*escAvailable*) of the ESC, or an operation of the brakes (*braPedStatus*) it will deactivate the CrCtl (*crctlActive*). The same functionality is redundantly performed by the *crctl\_monitor*, using the signal *crctlValid* to indicate the status of the CrCtl. Any following application is only allowed to use the demanded acceleration (*aSet*) if *crctlActive* and *crctlValid* indicate that the CrCtl is active.

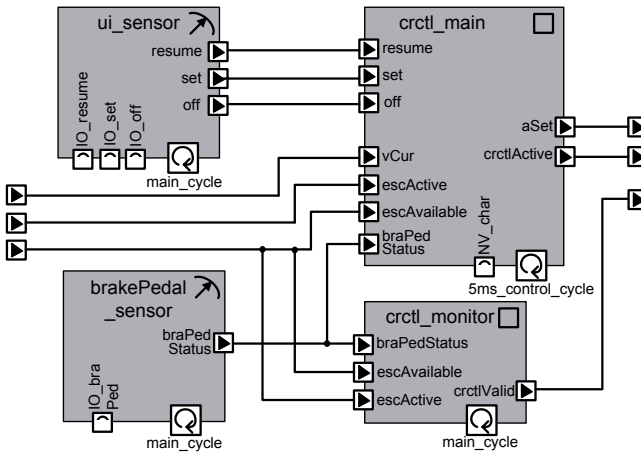


Fig. 1. A model of the application under consideration

Platforms, as they are referred to in this paper, consist of software and hardware, providing general purpose services that enable the execution of an application. I/O-wise, the exemplary platform has three channels to read digital signals (*DI\_Channel<sub>x</sub>*), two channels to do analog digital conversion (*ADC\_Channel<sub>x</sub>*) and two digital output channels. The platform can be attached to a CAN bus (*CAN\_ComChannel*) and also supports communication between application software components on the ECU (*IntraECU\_ComChannel*). Furthermore, the platform has the capabilities to guarantee a memory partitioning and offers three partitions, one CPU and non-volatile memory.

## 4 The Interface Language

Regarding the example introduced in section 3, the reader may observe that an application often cannot implement its safety concept without the aid of other

parts of the system. Therefore, the application developer needs to specify the dependencies on these parts, before being able to modularly certify the application. As shown in section 2, it is a common approach to capture these dependencies in contracts. These contracts again can have a horizontal/functional orientation, like the demand to inform the *CrCtl* about the availability of the *ESC*. Nevertheless, the focus of the VerSaI language lies on the vertical/technical interface.

When the safety concept of the application becomes more detailed, and thus more technical, certain measures can be implemented more efficiently or perhaps exclusively by the platform. To capture these vertical dependencies in the certificate, the application developer must specify demands on the platform. In the context of an integrated architecture, however, the application is developed without knowing the platform. Therefore, the demands must be specified referring only to elements of an application, for example, referencing to a signal or an application software component. Regarding our example, a typical requirement could demand the detection of corruptions of the communication signal *escAvailable*.

The VerSaI language allows the application developer to specify these demands by offering a fixed set of demand classes that are typically required by an application. These demand classes are contained in the *application language* package shown in figure 2. As also indicated in the figure, the model of the *application language* references to the model of the application’s design specification. This allows the application developer to attach the specified demands formally to the model elements they refer to. This integration on model-level enables certain consistency checks and the evaluation of completeness indicators, like, for example, to check the existence of the usual demands for all signals of a safety-critical application component.

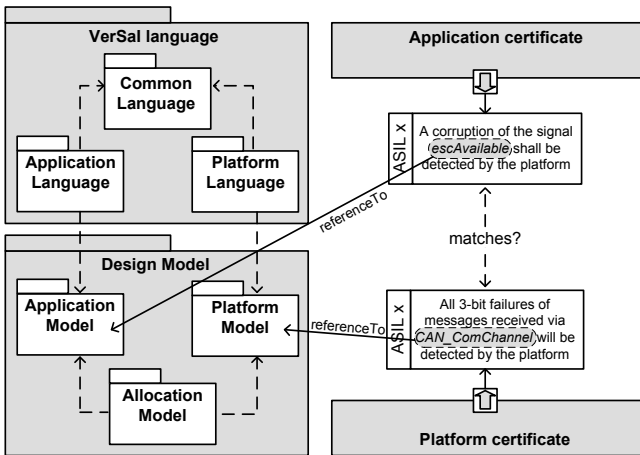


Fig. 2. An overview of the VerSaI language

The platform developer, on the other hand, needs to specify the platform’s guarantees without knowing the application. Consequently, the respective guarantees have to be more technical than the application demands and must refer only to elements of the platform. Again making reference to our example, the platform might give the guarantee to detect all 3-bit failures of messages received via the *CAN\_ComChannel*. Analogously to the content and the structure of the *application language*, the *platform language* package contains a set of guarantee classes which are typically provided by a platform, and is integrated into the design model of the platform. This allows to attach platform guarantees to the model elements of a platform in the same way the application demands are attached to the model elements of an application.

After the developers of applications and platforms deliver their products, each bundled with a modular certificate, the integrator of the system uses the *allocation model* to specify the deployment of the application elements to the devices and services of the platforms. As the interface requirements are attached to the design elements, the allocation implicitly associates the application demands with the relevant platform guarantees as well. If, for instance, the signal *escAvailable* is allocated to the channel *CAN\_ComChannel* it is clear that the demands of this specific signal have to be fulfilled by the guarantees provided for this specific communication channel.

In order to help answering the question whether the identified guarantees suffice to implement a specific demand, the VerSaI approach utilizes a *strategy repository*. The strategies in the repository describe established approaches on how to fulfill a certain type of demand and how to close the gap between the application level demand and the available, more technical platform guarantees. A strategy is always linked with the demand type it addresses, which allows the VerSaI language to present the strategy to the integrator when it is potentially applicable.

The following subsections contain a more detailed description of the language we developed. The structure of the VerSaI language reflects the different classes of dependencies identified from analysis of common safety standards and platform specifications like AUTOSAR [2], ARINC 653 [3], and non standardized general purpose platforms like the IFM CR7201 [1]. Subsection 4.2 describes the identified application-specific demands, whereas the platform-specific language elements are specified in 4.3. The common language package contains language aspects used in both module-specific language packages and is described in 4.1.

Please remark that although we utilize design models, they are not considered to be part of the VerSaI language. We intend to use existing notations, like we used the AUTOSAR meta-model to conduct the example in this paper.

## 4.1 Common Language

This subsection describes the language elements used in both, the application- and the platform-specific part of the language. These elements are contained in the *common language* package. Furthermore, this subsection illustrates the basic structure of the language.

Besides the dissection of the language in the different packages, the structure of the language can be described as a hierarchical classification of interface requirements, with a limited number of instantiable elements in the leaves of the hierarchy. The top-level classification, which differentiates guarantees and demands, has already been introduced. The next level of the classification contains several application and platform-specific demand and guarantee classes and are described in subsection 4.2 and 4.3.

To tailor a requirement class during instantiation, each class has certain quantitative and qualitative parameters. One mandatory qualitative parameter is the criticality level<sup>1</sup> of a requirement, classifying the risks caused by not meeting the requirement. Certain requirements also need quantification. If, for example, the CrCtl demands the detection of a delay of the signal *escAvailable*, it needs to specify the tolerable delay in ms.

Exemplary applications show that the available requirement classes, together with the notion of parameters, are able to cover a high percentage of the necessary vertical dependencies. We are yet aware that specific domains or specific applications may require special interface requirements that are currently not covered by the language. For that reason, the VerSaI language contains the notion of *custom requirements*. A *custom requirement* basically allows the developer to insert a requirement into the interface using natural language. Of course, those requirements do not have the benefits of the formalized requirements and can, for example, not be linked with strategies in the *strategy repository*. If the developer of an application or platform realizes that there are recurring custom demands, there is the possibility to extend the language. When inserting a new requirement into the classification, the requirement inherits all the characteristics of its super-classes, reducing the required modeling effort.

Up to this point, we have been regarding interface requirements as individual and isolated language elements. To increase the expressiveness, the VerSaI language allows aggregating multiple requirements to *requirement groups*. This language aspect is used for two reasons. First, it allows the flexible combination of single demands or guarantees to form complex language elements. The demand to detect a certain failure can, for example, be combined with either the demand to indicate the failure or to react to the failure. Furthermore, *requirement groups* allow the combination of demands and guarantees, to express that certain guarantees can only be provided after the demand has been fulfilled.

Finally, every interface requirement class in the meta-model has a representation in natural language. This allows the user to transform an interface specification into a human readable documentation, which is, for instance, needed for a final safety assessment. Such a representation is a concatenation of static and variable strings, where the variable parts enable capturing the variation points of a requirement class, like parameters or references to design model elements. Natural language representations are modeled using the Extended BackusNaur

---

<sup>1</sup> Specifying criticality levels is a concept used by several standards like DAL in DO-178B, SIL in IEC 61508 or ASIL in ISO 26262.



Form (EBNF). Terminal symbols are denoted with quotation marks. The following shows an example before and after the final production:

```
Abstract natural representation = "If a delay of the signal",
signal, "longer than", delay, "has been detected, the platform
shall indicate the failure within", reaction_time, "(",
criticality_level, ")"
```

```
Instantiated natural representation = "If a delay of the signal
escAvailable longer than 50ms has been detected, the platform
shall indicate the failure within 300ms (ASIL x)"
```

## 4.2 Application Language

This subsection presents the different demand classes the *application language* comprises. To identify the classes needed, we classified the different roles a platform can play in the safety concept of an application, and derived demand classes for each of these roles. From the viewpoint of an application, the platform can act as a provider of high integrity platform services(demand no.1), a monitor of the application's behaviour(no. 2), a failure handler(no.3 and 4), a provider of resource protection(no. 5) and an insurer of independence concepts(no. 6). These roles result in the following demand classes which are successively described hereafter.

1. Platform service demands
2. Application monitor demands
3. Simple reaction demands
4. Complex reaction demands
5. Resource protection demands
6. Independence demands

The platform needs to provide services to the application with a high integrity. Because of this, *platform service demands* allow specifying requirements to detect or avoid certain failure modes of platform services that could have safety-critical effects on the application. The approach taken to identify all relevant failure modes is comparable to the work done by Conmy[6]. First, we identified the core platform services needed by the applications, and then we used the guide words described in [15] to derive the relevant failure modes. Figure 3 shows the representation of *platform service demands* in the meta-model.

You can see that the model allows choosing the *demand type* (detection or avoidance) and the addressed failure mode, when instantiating a *platform service demand*. The meta-model contains a hierarchical representation of the identified failure modes on application level. The first level of the hierarchy differentiates between the failure-modes of the different platform services. Each failure mode of the service classes can be related to an element of the application design model, as exemplarily shown for the *ComFailure* class (*relatedSignal*). The next level

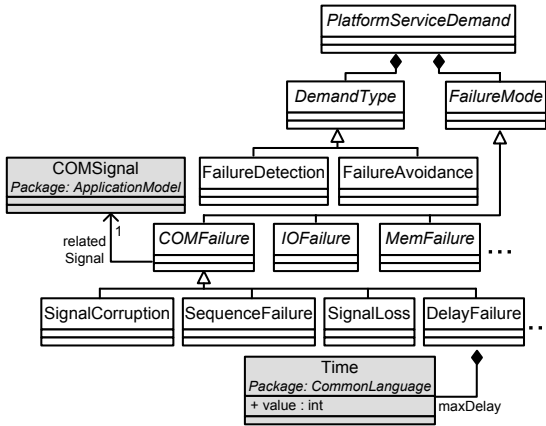


Fig. 3. Meta-Model Extract Showing Platform Service Demands

of the hierarchy shows the instantiable communication failure-modes and the concept of parameters (*maxDelay*) on the example of the *DelayFailure*.

Due to space restrictions, the following language elements will be described using the more concise EBNF production rules of the respective natural language representation. The top-level production rule for a *platform service demands* is described as follows:

```
platform service demand = failure_mode, "shall be", "detected" |
"avoided", "by the platform"
```

It is a common strategy to use the platform as a monitor of the application, because the platform provides a certain degree of independence. Those monitoring mechanisms can be implemented in software or hardware, and allow observing the applications temporal behaviour (e.g. watchdogs) and, to a lesser extend, its functional behaviour (e.g. sequence monitor). To capture this aspect, the VerSaI language contains *application monitor demands*. Since the general purpose platform does not have the knowledge to differentiate between correct and incorrect application behavior, the application typically has to parameterize standard mechanisms provided by the platform. The description of an *application monitor demand* is shown below:

```
application monitor demand = deviation, "shall be detected by the
platform"
```

To handle failures, the platform can execute several reactions that an application cannot perform. If, for example, an application component detects that another component is behaving erroneously, the application that is aware of the failure can ask the platform to shut it down. These requests are covered using *reaction demands*. Furthermore, some platforms can be configured to react on failures "automatically". It especially makes sense not to leave failure handling

to the application if the detected failure poses the risk that the application is not working properly anymore. *Complex reaction demands* basically allow the application to demand a specific reaction, subsequent to a detection of a failure or a deviation in application behavior. Because it is usually critical to indicate or react to a failure within a given time interval, the demands described allow specifying a *failure-reaction-time*.

```
simple reaction demand = "If demanded, the platform shall",
reaction, "within", reaction_time
```

```
complex reaction demand = "If", failure_mode | reaction, "has been
detected, the platform shall", reaction, "within", reaction_time
```

A specific thread of integrated architectures stem from shared resources. Even applications that have no functional relationship can interfere with each other via resources used by both parties. Typical interferences are excessive use of computational resources or the corruption of another application's memory. Because of this risk, most safety standards demand either that applications with different criticality are not allowed to execute on the same platform or that they must be protected from each other. *Resource protection demands* enable the application developer to demand freedom from interference between any of the application's software components and all other software components on the same platform that have a lower criticality. These demands are defined using the following language pattern:

```
resource protection demand = application_sw_component, "must be
protected from interference from other application software
components with lower criticality"
```

Finally, we have to consider demands to enforce an application's independence concept. An application may contain functions which are redundantly developed to mitigate the effect of failures in single components. This strategy is only effective if the redundant functions fail independently, which means they have neither random nor systematic common-cause failures. As a shared resource is an obvious threat to this independence, an application developer can use *independence demands* to demand the platform to enforce the application's safety concept. We differentiate between demands for the independent execution of two application software components, typically fulfilled by deploying them to different controllers, and the demand of independence between incoming or outgoing signals.

```
independence demand = "Independence between",
(application_sw_component, "and", application_sw_component)
| (IO_signal, "and", IO_signal) | (COM_signal, "and",
COM_signal), "must be preserved by the platform"
```

### 4.3 Platform Language

In order to complement the *application language*, the *platform language* needs to cover as many guarantees as possible, which a platform can give to support an application's safety concept. Since we regard general purpose platforms, no platform can make application specific guarantees. This means that platform guarantees cannot ensure that specific signals are not corrupted or that a specific runnable is scheduled in time.

However, general purpose platforms typically implement services that are configurable. A configurable service can be adapted by the integrator in order to fit it to the specific demands of the actual application. Take, for instance, a watchdog service that has been configured to raise an alarm if a specific application has not shown its activity within a specific time window. The presence of configurable services has two consequences for our language. First, a platform guarantee can make references to classes of application elements, like application software components, but not to specific application elements, like *crtcl\_monitor*. Second, before such a guarantee can be assumed to hold, a correct configuration of the service must be provided. Therefore, the *platform language* provides the *configuration demand* class, which can be used together with the notion of requirement groups to express this correlation.

Besides the afore-mentioned demand classes, we tried to keep the platform language structurally compatible with the application language and therefore, developed six guarantee classes to match the introduced application demand classes.

*Platform service guarantees* follow the same principle as platform service demands. The platform developer can specify guarantees to assure either the avoidance or the detection of certain failure modes of the platform services. Looking at specifications of available platforms, platform providers tend to specify the technical safety measures their platform implements, rather than to provide abstract guarantees. The failure modes detected by these measures are, therefore, more technical in nature than the failure-modes identified in the application language. *Platform service guarantees* are represented in natural language using the following pattern:

```
platform service guarantees = technical_failure_mode, "will be",
"detected" | "avoided", "by the platform"
```

The term health monitor is taken from ARINC 653<sup>[3]</sup>, where the health monitor service is responsible for detecting platform or application failures and subsequently, depending on the type of failure, takes action to handle or to indicate the failure. The health monitor specified in ARINC 653, as well as the AUTOSAR services that are comparable in functionality (FIM and DEM), are highly configurable. In contrast to that, the IFM CR7201 provides a limited range of failure reactions and almost no configurability. Based on this observation, the platform language needs to allow for a specification of fixed as well as variable health monitor capabilities. *Application monitor guarantees* specify the behavior deviations that can be detected by the platform, whereas *simple reaction guarantees*

specify the reactions that can be directly demanded by the application. *Complex reaction guarantees* specify a list of possible configurable reactions for each detectable failure or deviation. The illustrated platform guarantees contain the following natural language representations:

```
application monitor guarantee = deviation, "will be detected by
the platform"
```

```
simple reaction guarantee = "If demanded, the platform will
perform", reaction, "within". reaction_time
```

```
complex reaction guarantee = "If", technical_failure_mode |
deviation, "is detected, the platform can perform the
following reactions:", {reaction, "within", reaction_time}
```

Rushby [18] presents different threats that have to be taken care of before a safe partitioning can be assumed. Among those threats are the previously mentioned memory corruptions, monopolizations of computational resources or other threats like interference via devices (e.g. IO peripherals or communication channels). *Resource protection guarantees* allow the platform developer to separately guarantee the absence of these threats.

```
resource protection guarantee = "The platform guarantees
freedom from intereference considering", interference_class
```

A platform can provide independent services that might be needed in order to support an application's safety concept. A platform can, for example, have heterogeneously developed input peripherals (maybe one current the other voltage-based) and independently developed software stacks to process the inputs to provide independent input services. Therefore, the platform language contains *independence guarantee* to specify independent communication and IO services provided by the platform.

```
independence guarantee = "The platform provides independence
between", (IO_peripheral, "and", IO_peripheral) | (COM_channel,
"and", COM_channel)
```

## 5 Industrial Evaluation

Based on the running example of the fictive cruise control illustrated in chapter 3, the feasibility of the VerSaI method was to be analysed from an industrial point-of-view. Our goal was to ascertain whether or not the proposed method can be employed with manageable effort using readily-available information and producing meaningful results. While manual safety certification is, of course, well-known and utilized in current practice, we did not aim to quantify possible improvements or reach a comparison between the two approaches.

The method’s focus on the vertical interface restricts the domain of addressable safety aspects. However, it does cover a majority of aspects that are well-defined and can be handled in a formalized and thus potentially automatable manner. Horizontal dependencies between applications are more complex and mostly characterize ”custom requirements” resulting in the necessity of manual safety evaluations.

By systematically specifying safety dependencies on the vertical interface between applications and platforms, the method especially supports the conscientious verification of those dependencies that would generally be regarded as tedious (simple, but formalizable). Using the language in conjunction with the *strategy repository*, the knowledge of system experts can be documented and ultimately reused for subsequent evaluations, thus possibly reducing necessary effort and avoiding planning conflicts resulting from the restricted availability of such resources.

Upon integration of modular certificates between applications and platforms, the VerSaI method performs a certain matching of demands and guarantees and presents strategies containing arguments explaining the fulfilment of the interface requirements. Due to the fact that not all feasible safety interface requirements at the vertical interface are easily projected into an interface language, a certain share of these requirements will remain *custom requirements* and thus will require manual evaluation by engineers. Therefore, a fully automatable solution cannot be achieved, but this does not lessen the viability of the method as it does ensure the systematic documentation of these requirements and provides a useful share of automatable aspects.

We expect a more reproducible and reliable evaluation of safety interface requirements while potentially reducing effort through reuse and automation. Continuing evaluation of the methodology will show how it can be employed on a wider scale and identify possible benefits in comparison with current practice.

## 6 Conclusion and Future Work

This paper has described the VerSaI language, a model-based language to specify the demanded and guaranteed requirements between applications and general purpose platforms. The language was introduced gradually, by illustrating the different language elements and how they are related to each other. Finally we presented an evaluation of the feasibility and the benefits of the language from an industrial perspective.

Future work includes the development of a semi-automatic mediation algorithm. The mediation algorithm is based upon a formalization of strategies, which contain the information on how to fulfill a certain class of demands using guarantees provided by a platform. The algorithm will check the applicability of the strategy under the given deployment and, in the case of success, constructs a human readable argument explaining how the demands have been met.

## References

1. Website of ifm electronics, <http://www.ifm.com/>
2. Website of the autosar standard, <http://www.autosar.org/>
3. ARINC: Arinc 653, avionic application software standard interface, part 1 (2005)
4. Bate, I., Hawkins, R., McDermid, J.: A contract-based approach to designing safe systems. In: Proceedings of the 8th Australian Workshop on Safety-Critical Systems and Software (SCS 2003), pp. 25–36 (2003)
5. Bate, I., Kelly, T.: Architectural considerations in the certification of modular systems. In: Anderson, S., Bologna, S., Felici, M. (eds.) SAFECOMP 2002. LNCS, vol. 2434, pp. 321–324. Springer, Heidelberg (2002)
6. Conmy, P., McDermid, J.: High level failure analysis for integrated modular avionics. In: Proceedings of the 6th Australian Workshop on Safety Critical Systems and Software (SCS 2001), pp. 13–22. ACM, New York (2001)
7. Conmy, P., Nicholson, M., McDermid, J.: Safety assurance contracts for integrated modular avionics. In: Proceedings of the 8th Australian Workshop on Safety-Critical Systems and Software (SCS 2003), pp. 69–78 (2003)
8. Conmy, P., Paige, R.: Challenges when using model driven architecture in the development of safety critical software. In: Proceedings of the Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES 2007), pp. 127–136. IEEE, Los Alamitos (2007)
9. Damm, W., Metzner, A., Peikenkamp, T., Votintseva, A.: Boosting re-use of embedded automative applications through rich components. In: Proceedings of the Workshop on Foundations of Interface Technologies 2005, FIT 2005 (2005)
10. Domis, D., Trapp, M.: Integrating safety analyses and component-based design. In: Harrison, M.D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 58–71. Springer, Heidelberg (2008)
11. Fenn, J., Hawkins, R., Williams, P., Kelly, T.: Safety case composition using contracts - refinements based on feedback from an industrial case study. In: Proceedings of the 15th Safety Critical Systems Symposium (SSS 2007). Springer, Heidelberg (2007)
12. Grunske, L.: Towards an integration of standard component-based safety evaluation techniques with saveccm. In: Hofmeister, C., Crnković, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214, pp. 199–213. Springer, Heidelberg (2006)
13. Jones, C.B.: Tentative steps toward a development method for interfering programs. *ACM Trans on Programming Languages and Systems* 5(4), 596–619 (1983)
14. Kelly, T., Weaver, R.: The goal structuring notation – a safety argument notation. In: Proceedings of the Dependable Systems and Networks Conference 2004 (DSN 2004). IEEE, Los Alamitos (2004)
15. McDermid, J., Pumfrey, D.: A development of hazard analysis to aid software design. In: Proceedings of the 9th Annual Conference on Computer Assurance (COMPASS 1994), pp. 17–25. IEEE, Los Alamitos (1994)
16. Meyer, B.: Applying "design by contract". *IEEE Computer* 25(10), 40–51 (1992)
17. Papadopoulos, Y., McDermid, J., Sasse, R., Heiner, G.: Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Elsevier - Reliability Engineering & System Safety* 3(71), 229–247 (2001)
18. Rushby, J.: Partitioning in avionics architectures: Requirements, mechanisms and assurance (1999)
19. Rushby, J.: Modular certification (2001)
20. Schneider, D., Trapp, M.: Conditional safety certificates in open systems. In: Proceedings of the 1st Workshop on Critical Automotive Applications: Robustness & Safety, CARS 2010, pp. 57–60. ACM, New York (2010)

# DALculus – Theory and Tool for Development Assurance Level Allocation

Pierre Bieber, Rémi Delmas, and Christel Seguin

ONERA, 2 avenue Edouard Belin,  
31055 Toulouse, France

{Pierre.Bieber, Remi.Delmas, Christel.Seguin}@onera.fr

**Abstract.** The Development Assurance Level (DAL) indicates the level of rigor of the development of a software or hardware function of an aircraft. We propose a theory formalizing the DAL allocation rules found in the ARP4754a recommended practices. A tool implementing this theory was developed in order to assist the safety specialists when checking or optimizing a DAL allocation.

**Keywords:** Dependability Assessment, Aerospace Systems, Avionics.

## 1 Introduction

The Development Assurance Level (DAL) indicates the level of rigor of the development of a software or hardware function of an aircraft. The DAL guides the assurance activities that should be applied at each stage of development. These activities aim at eliminating design and coding errors that would have a safety effect on the aircraft.

The revised version of Aeronautical Recommended Practices ARP4754a [1] establishes rules to allocate the DAL to functions. The allocation is primarily based on the severity of the effects of a function implementation error. But new rules introduce the possibility to downgrade the DAL levels based on the independence at requirement, implementation or hardware deployment level.

It can be tedious to check that downgrading rules were applied correctly. Furthermore, designers are trying to allocate the smallest DAL possible to functions in order to decrease development costs. Consequently, we have investigated means to assist the safety specialists when checking or optimizing a DAL allocation.

We undertook the formalization of DAL allocation rules as constraints, that link the maximal allowed reduction of DAL and the independence of functions appearing in the minimal combination of function errors leading to an aircraft function loss. Optimization criteria are also defined to help minimizing the allocated DAL and the number of functions required to be independent.

This problem is solved using very efficient constraint solvers. A valid DAL allocation and a set of function independence requirements are extracted from the solution found by the solver and are proposed to the designers. The approach also takes into account user provided constraints that help the tool to focus on more interesting allocations.



The following chapter describes the Development Assurance Level Allocation Process according to ARP4754a. Then, in chapter 3, we detail the proposed theory of DAL allocation. Finally, we explain how the approach is implemented and the tool was applied on various critical aircraft systems including the Electrical Generation and Distribution System. The paper concludes with the first lessons learnt from these experimentations.

## 2 Development Assurance Level Allocation Process

### 2.1 Aims of the DAL

The design of aeronautics safety critical systems deals with two families of faults:

- random faults of equipments, such as an electrical wire rupture that would cause the loss of power supply for a computer cabinet,
- systematic faults in the development of the equipment, which include errors in the specification, design and coding of hardware and software. An instance of a development fault could be a specification of the “wheel on ground” function that would not make a difference between the landing gear being compressed or decompressed resulting in a incorrect indication of whether the aircraft is on ground or flying.

Two very different approaches are used when assessing whether the risk associated with these two types of faults is acceptable. Quantitative requirements (thresholds of fault occurrence probabilities) are associated with random equipment faults whereas non-quantitative design assurance requirements (Development Assurance Level) are associated with development faults. This approach is not unique to the aeronautics industry, it also applied in other safety-critical domains such as space, nuclear, railway or automotive industries [2].

In the aeronautics industry, a DAL ranging from E to A (where A is the higher level) is allocated to functions, software and hardware items. According to the ARP4754, *“The Development Assurance Level is the measure of rigor applied to the development process to limit, to a level acceptable for safety, the likelihood of Errors occurring during the development process of Functions (at aircraft level or system level) and Items that have an adverse safety effect if they are exposed in service.”*

The DAL associated with a software item guides the assurance activities that have to be performed during its development following DO178B [3]. The higher the DAL, the more detailed and rigorous are the assurance activities to be performed. For instance, the following table describes three objectives of the Software Coding and Integration Process. It indicates which objectives are applicable at a given DAL level. A cell containing R means that this is a Required objective at this level, a blank cell means the objective is not required and a cell containing I means that the objective should be achieved with independence. Independence is achieved when the activity is performed by a team different from the software development team.

**Table 1.** DO178B Software Coding Objectives (extract)

Id	Objective	DAL Applicability			
		A	B	C	D
1	Source Code complies with low-level requirements	I	I	R	
2	Source Code complies with software architectures	I	R	R	
3	Source code is verifiable	R	R		

At level E nothing is required. At level D none of these three objectives has to be achieved. Traceability at this level of detail (between source code and requirements) is not needed, but other traceability (between high-level and system requirements) is required. At level C, objectives 1 and 2 are required without independence. Independence is almost never required at level C. At levels A and B, the three objectives have to be fulfilled. The difference between these two levels is that more objectives shall be achieved with independence at level A than at level B.

High DALs require a great number of assurance activities. The increase in the level of rigor, level of detail and the need to involve independent teams increase the development cost of software and hardware items. Consequently, designers aim at allocating a DAL to software and hardware as low as possible, within the bounds imposed by safety regulation, in order to reduce the development cost of their systems.

## 2.2 DAL Allocation Rules According to ARP4754a

DAL allocation is a part of the System Safety Assessment Process which comprises several steps. First, the Functional Hazard Analysis identifies the Failure Conditions (e.g. safety critical situations of the system) and assesses their severity on a scale going from No Safety Effect (NSE) to Catastrophic (CAT). Then, during the Preliminary System Safety Assessment, fault-trees (or alternatively fault propagation models [4]) are built and analyzed. A fault-tree describes the way that item faults propagate inside the system architecture in order to cause a Failure Condition. Minimal combinations of item faults leading to the Failure Conditions are extracted from the fault-tree, these combinations are called Minimal Cut Sets (MCS). These combinations are used to compute the mean probability of the Failure Condition in order to assess whether the designed architecture is safe enough. Usually the mean probability of a Failure Condition whose severity is Catastrophic shall be smaller than  $10^{-9}$  per flight-hour.

DAL allocation is based on a qualitative assessment of the minimal cut sets computed for the Failure Conditions of the system. We consider that an item fault contributes to a Failure Condition if it appears (through its failure modes) in one of the MCS of the failure condition. A DAL is associated with an item according to the classification of the most severe Failure Condition that this item fault contributes to. Actually, due to DAL downgrading rules, it is not necessarily the most severe FC that constrains the DAL allocation the most.

Table 2 gives the basic DAL assignment rules. **Sev** is the severity of the most critical Failure Condition an item fault is involved into. According to this table, an item is assigned DAL B if the most severe consequence of this item fault is classified as Hazardous (HAZ).

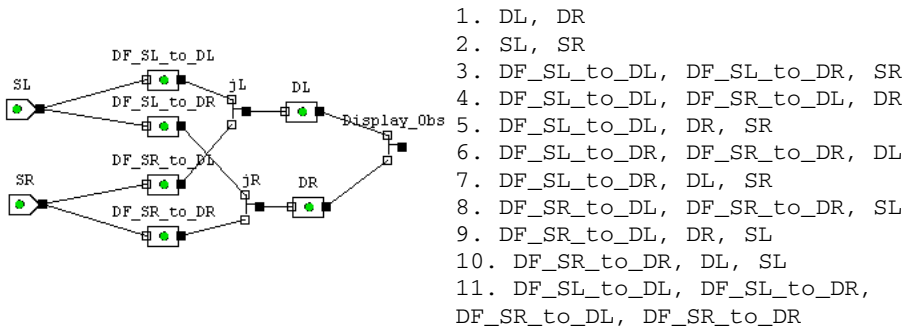
**Table 2.** Basic DAL Allocation

Sev	NSE	MIN	MAJ	HAZ	CAT
DAL	E	D	C	B	A

New DAL allocation rules introduced in the revised ARP4754a allow to downgrade the original DAL allocated using the basic allocation rule, in cases when items involved in the minimal cut sets are known to be pair-wise independent (we will discuss in detail the notion of independence in section 2.3). Each minimal cut set is analyzed using one of the 2 following downgrading options:

- **Option 1:** The original DAL of one item in the minimal cut set is preserved and the DAL of remaining items may be downgraded of at most two levels.
- **Option 2:** The original DAL of one pair of items in the minimal cut set may be downgraded of at most one level and the DAL of remaining items may be downgraded of at most two levels.

Once all minimal cut sets are analyzed the lowest DAL consistent with the authorized downgrading by the selected option may be allocated to the items.



**Fig. 1.** Data Display (Graphical view and Minimal Cut Sets)

**Example:** To illustrate the various DAL allocation rules, we introduce a very basic Data Measurement and Display example that is graphically described in figure 1. Data is measured by two sensors SL (Left sensor) and SR (Right Sensor) and displayed on two Display units DL and DR, measured data is sent by the sensors to both displays (via data flows named DF\_SL\_to\_DL, DF\_SL\_to\_DR, DF\_SR\_to\_DL and DF\_SR\_to\_DR). Each of these items may be lost. Measured data can be displayed on one display unit as long as the display unit is working and data measured by at

least one sensor is transmitted to the display. The Failure Condition of interest is the loss of data on both displays, and it is classified Hazardous.

Figure 1 also provides the list of the 11 Minimal Cut Sets leading to the loss of data on both displays. MCS 1 states that the loss of both displays leads to the failure condition, and MCS 11 states that the loss of 4 data flows leads also to the failure condition.

As the Failure Condition is classified Hazardous, the original DAL allocated to each of the items of the Data Measurement and Display system is equal to B.

Downgrading rules can be applied if we suppose that the groups of Left items (SL, DL, DF\_SL\_to\_DL, DF\_SL\_to\_DR) and Right items (SR, DR, DF\_SR\_to\_DL and DF\_SR\_to\_DR) are mutually independent. We can check that each minimal cut set of each FC contains at least a pair of faults from items that are mutually independent. For instance MCS 1 contains the faults of items DL and DR that are mutually independent. Hence, if we apply option 1 then we may allocate DAL B to DL and downgrade the DAL of DR to D or, conversely, downgrade the DAL of DL and allocate DAL B to DR. After analyzing all the minimal cut sets, allocating DAL B to all Left items and DAL D to all Right items would be allowed by option1.

If we now apply option 2 and we analyze MCS 1 then the DAL of both DL and DR may be downgraded to C. When MCS 11 is analyzed, as DF\_SL\_to\_DR and DF\_SR\_to\_DL are independent their DAL may be downgraded to C and the DAL of remaining items DF\_SL\_to\_DL and DF\_SR\_to\_DR may be downgraded to D. When MCS 10 is analyzed, as DL and SL are not independent but DL and DF\_SL\_to\_DR are independent, the DAL of DL and DF\_SL\_to\_DR may only be downgraded to C. Consequently, there is a conflict between the analysis of MCS 11 and MCS 10 with respect to the downgrading of the DAL of DF\_SL\_to\_DR. To solve this conflict, the higher DAL allocation has to be used. Here, DAL C should be allocated to DF\_SL\_to\_DR.

But, if we go back to the analysis of MCS 11, it was also possible to downgrade the level of DF\_SL\_to\_DR and DF\_SR\_to\_DL to D and the level of DF\_SL\_to\_DL and DF\_SR\_to\_DR to C. In that case the analysis of MCS 10 and 11 would not be conflicting.

As it was shown by this simple example, verifying that a DAL allocation is correct is not an easy task, and generating a correct DAL allocation is even more difficult when systems become large. For each minimal cut set there are several downgrading possibilities depending on the selected option. Furthermore, the downgrading possibilities have to be consistent for all minimal cut sets. We think that a DAL allocation and verification tool would be very helpful for the safety analyst. The first step in developing such a tool is to formalize the principles of DAL allocation.

### 2.3 Item Independence According to ARP4754A

Independence plays a crucial role in the DAL allocation process as it is required in order to apply DAL downgrading. According to the ARP4754a *“item Development Independence ensures that the development of items on which the Function(s) is(are) implemented, should not suffer from a common mode Error.”*

Item Development Independence is achieved by relying on items that use different types of software and hardware technologies to implement a Function. For instance, a general-purpose computer-based LCD screen and an ad-hoc electro-mechanical display could be used to display the fuel quantity on-board. These two set of items would be considered as independent as they do not depend on similar technologies. On the contrary, several occurrences of a screen with the same technology (LCD screen for instance) could be considered as non-independent.

From a system development perspective, the use of independent items increases the development costs. In some cases, it might just be impossible to achieve item independence as only one technology is available to implement the items. So system designers will be interested by architectures with a limited use of independent items.

**Example:** Let us consider again the Data Measurement and Display system studied in the previous section. We now suppose that the four data flows of the Data Measurement and Display System are no longer mutually independent because the same communication protocol is used to transmit them. In that case, when MCS 11 is analyzed, we cannot downgrade the DAL level of the data flows as they are not independent. Consequently DAL B would be allocated to data flows (DF\_SL\_to\_DL, DF\_SL\_to\_DR, DF\_SR\_to\_DL and DF\_SR\_to\_DR).

As it was shown by the previous example, various assumptions on item independence lead to very different DAL allocation. This makes the DAL allocation even more complex. So in order to assist the DAL allocation process we should also help the safety analyst to find which items have to be independent.

### 3 DAL Allocation as a Constraint Satisfaction Problem

We formalize DAL allocation as a sequence of two constraint satisfaction problems. The first problem consists in identifying a minimal set of necessary independence relations between items. The second problem allocates the DAL to items taking into account the various downgrading options enabled by the item independence relations identified by solving the first problem.

#### 3.1 Independence Identification

Besides the classical quantitative safety requirement, a qualitative safety requirement is also usually associated with a failure condition. This qualitative requirement could be of the form “*no combination of item faults of size strictly less than NSev leads to the failure condition*”, where the relation between NSev and the severity of the failure condition is given by Table 2.

To check the qualitative safety requirements associated with a failure condition, we want to establish that the size of each minimal cut set is greater than or equal to NSev. The size of an MCS including N item faults is not necessarily equal to N because these item faults could be non-independent. If none of the items appearing in the MCS are independent, a single common cause fault could lead to the failure condition. According to Table 2, this would not be acceptable for failure conditions whose severity ranges from MAJ to CAT. We consider that the size of a minimal cut set is

equal to the cardinal of the maximal subset of mutually independent items contained in the minimal cut set. So for a MAJ or HAZ failure condition, each minimal cut set should contain at least a pair of independent items. For a CAT failure condition each minimal cut set should contain at least three items that are mutually independent. We name this maximal subset the *core* of the MCS from now on.

**Table 2.** Qualitative Safety Requirements

Sev	MIN	MAJ	HAZ	CAT
NSev	1	2	2	3

**Example:** According to the definition of the size of a minimal cut set, MCS 1 of the Data Measurement and Display system is of size 2 if DF and DL are independent. MCS 11 is of size 4 if the four data-flows are mutually independent but it is of size 2 if only DF\_SL\_to\_DL and DF\_SR\_to\_DR are independent. Both independence relations would be acceptable as the failure condition if HAZ.

In the following we describe the Constraint Satisfaction Problem (CSP) that has to be solved in order to identify independence relations between items so that qualitative requirements are enforced. The CSP is defined by a set of constants that are inputs of the problem such as the set of minimal cut sets, a set of variables whose values are searched by the solver, and by the constraints that should be satisfied on these variables. We also use abbreviations to make the constraints more readable. Finally we propose a quantitative criterion that is optimized by the solver during its search.

- Constants

- I: set of items,
- MCS: set of minimal cut sets for the failure condition,
- Nsev: minimal size required for minimal cut sets to satisfy the qualitative requirement.

- Variables

–indep( $p, p'$ ) is true if items  $p$  and  $p'$  are independent,

- Abbreviations

- The size of a minimal cut set  $mc$  is greater than or equal to  $n$  iff there exists a set of  $n$  items in  $mc$  that are mutually independent:

$$\text{size}(mc) \geq n \iff (\exists m \subseteq mc, \text{card}(m) = n \ \& \ \text{indep}_m(m))$$

- Items in the set  $m$  are mutually independent iff all pairs of distinct items  $(p, q)$  in  $m$  are independent.

$$\text{indep}_m(m) \iff \forall (p, q) : m * m, p \neq q \Rightarrow \text{indep}(p, q)$$

- Constraint

- The size of each minimal cut set shall be greater or equal to Nsev:

$$\forall mc : \text{MCS}, \text{size}(mc) \geq \text{Nsev}$$

• Optimization Criterion

Minimize the number of pairs of independent items:

$$\Sigma_{(p,q) : I \times I} \text{indep}(p, q)$$

**Example:** Let us consider again the Data Measurement and Display system. A minimal and valid independence relation found by the solver has cardinal 7 :

$$\begin{aligned} &\text{indep}(\text{DR}, \text{DL}), \text{indep}(\text{SL}, \text{SR}) \\ &\text{indep}(\text{SR}, \text{DF\_SL\_to\_DR}) \\ &\text{indep}(\text{SL}, \text{DF\_SR\_to\_DL}) \\ &\text{indep}(\text{SL}, \text{DF\_SR\_to\_DR}) \\ &\text{indep}(\text{DR}, \text{DF\_SL\_to\_DL}) \\ &\text{indep}(\text{DF\_SL\_to\_DR}, \text{DF\_SR\_to\_DR}). \end{aligned}$$

**3.2 DAL Allocation**

This section describes the CSP that formalizes the DAL allocation problem. Table 3 shows the correspondence between DALs and numerical values used to encode the order relation between DALs and simplify the expression of constraints.

**Table 3.** Numerical values for the DAL

<b>NDAL</b>	0	1	2	3
<b>DAL</b>	D	C	B	A

• Constants

- I: set of items,
- MCS: set of minimal cut sets,
- indep: independence relation,
- NDAL: numerical value of the DAL of FC

• Variables

- $\text{ndal}(f)$  : numerical value of the possibly downgraded DAL of item f

• Constraints

- The DAL level of an item cannot be downgraded by more than two levels:

$$\forall mc : \text{MCS}, \forall p : mc, \text{ndal}(p) \geq \text{NDAL} - 2$$

- **option 1.** For each MCS and each of its items, either the DAL is not downgraded, or the DAL is downgraded. In this case a core of mutually independent items of size NSev must exist in the considered MCS such that one of its elements has at least the DAL NDAL, and the downgraded item must be part of this core:

$$\begin{aligned}
& \forall mc:MCS, \forall p:mc, \\
& \quad ndal(p) \geq NDAL \text{ or} \\
& \quad (ndal(p) < NDAL \Rightarrow \\
& \quad \quad (\exists m \subseteq mc, indep(m) \ \& \ card(m) = Nsev \ \& \ p:m \ \& \\
& \quad \quad (\exists q:m, ndal(q) \geq NDAL)))
\end{aligned}$$

- **option 2.** For each MCS and each of its items, either the DAL is not downgraded, or the DAL is downgraded. In this case a core of mutually independent items of size Nsev must exist in the considered MCS such that two of its elements have at least the DAL NDAL-1, and the downgraded item must be part of this core.

$$\begin{aligned}
& \forall mc:MCS, \forall p:mc, \\
& \quad ndal(p) \geq NDAL \text{ or} \\
& \quad (ndal(p) < NDAL \Rightarrow \\
& \quad \quad (\exists m \subseteq mc, indep(m) \ \& \ card(m) = Nsev \ \& \ p:m \ \& \\
& \quad \quad (\exists (q, r) : m*m, q \neq r \ \& \ ndal(q) \geq NDAL-1 \ \& \ ndal(r) \geq NDAL-1)))
\end{aligned}$$

- **Criterion:** Minimize the sum of numerical values of allocated DALs:

$$\Sigma_f : \mathbb{I} \ ndal(f)$$

**Example:** We consider the minimal cut sets of the Data Measurement and Display system and the `indep` relation presented in the previous section. If we apply option 1, an optimal DAL allocation is:

$$\begin{aligned}
dal(SR) &= dal(DL) = D \\
dal(SL) &= dal(DR) = dal(DF\_SL\_to\_DR) = B \\
dal(DF\_SR\_to\_DL) &= dal(DF\_SL\_to\_DL) = dal(DF\_SR\_to\_DR) = D
\end{aligned}$$

If we apply option 2, an optimal DAL allocation is:

$$\begin{aligned}
dal(SR) &= dal(DL) = dal(SL) = dal(DR) = dal(DF\_SL\_to\_DR) = C \\
dal(DF\_SR\_to\_DL) &= dal(DF\_SL\_to\_DL) = dal(DF\_SR\_to\_DR) = C
\end{aligned}$$

## 4 Tool Support and Experimentations

### 4.1 Pseudo-Boolean Constraint Solving

The two constraint satisfaction problems presented in the previous section can be solved very efficiently by solvers such as Sat4J [6] or WBO [7] that deal with pseudo-Boolean logic (also known as  $\{0,1\}$  linear integer constraints). In this section we explain how the DAL allocation problem is modeled using pseudo-Boolean logic.

In pseudo-boolean logic, the DAL level allocation is encoded by a predicate `hasDal(p, i)` which is true if and only if `ndal(p) >= i`. Let `Subsets(mc, Nsev)` denote the collection of all possible subsets of size Nsev of the minimal cut set `mc`, called the Nsev-subsets of `mc`. We introduce the predicate `dalOk(mc, s)` to represent that the subset `s` of `mc` satisfies the chosen DAL allocation constraints.



For each MCS  $mc$ , there must be at least one of its Nsev-subsets such that the  $dalOk$  predicate holds. This is modeled in pseudo-boolean logic by instantiating the following constraint for each MCS, where  $\{s_1, \dots, s_n\} = \text{Subsets}(mc, Nsev)$ :

$$dalOk(mc, s_1) + \dots + dalOk(mc, s_n) \geq 1;$$

For each item, the DAL is downgraded by at most two levels with respect to the default case :

$$hasDal(f, NDAL-2) \geq 1;$$

Listed below are the constraints instantiated for each Nsev-subset to encode its  $dalOk$  semantics when  $Nsev=2$ , for each possible DAL allocation options.

**DAL option1** is satisfied for subset  $s = \{p_1, p_2\}$  in MCS  $mc$  if and only if:

- Items of the subset are mutually independent:

$$-1*dalOk(mc, \{p_1, p_2\}) + 1*indep(p_1, p_2) \geq 0;$$

- One item in the subset has the Default DAL:

$$-2*dalOk(mc, \{p_1, p_2\}) + 2*hasDal(p_1, NDAL) + 2*hasDal(p_2, NDAL) \geq 0;$$

- Each item  $p_3$  in  $mc - \{p_1, p_2\}$  has the default DAL:

$$-1*dalOk(mc, \{p_1, p_2\}) + 1*hasDal(p_3, NDAL) \geq 0;$$

**DAL option2** is satisfied by subset  $s = \{p_1, p_2\}$  in MCS  $mc$  if and only if:

- Items of the subset are mutually independent:

$$-1*dalOk(mc, \{p_1, p_2\}) + 1*indep(p_1, p_2) \geq 0;$$

- A pair of items in the subset has at least the default DAL minus 1:

$$-2*dalOk(p_1, p_2) + 1*hasDal(p_1, NDAL-1) + 1*hasDal(p_2, NDAL-1) \geq 0;$$

- Each item  $p_3$  in  $c \setminus \{p_1, p_2\}$  has the default DAL:

$$-1*dalOk(mc, \{p_1, p_2\}) + 1*hasDal(p_3, NDAL) \geq 0;$$

## 4.2 The DALculator

A tool called the DALculator supporting independence identification and DAL allocation was developed. Figure 2 shows its graphical user interface. The user first selects one or several files containing the minimal cut sets of failure conditions then and indicates the failure condition severity. The user also selects the DAL downgrading option and the solver to be used. The tool parses the minimal cut sets and first generates a pseudo-Boolean instance encoding the independence identification problem and solves it. If a solution is found, the tool extracts the list of all mutually independent cores from it and generates a pseudo boolean instance encoding the the DAL allocation problem, and solves it using the chosen solver. If a solution is found, the DALculator proposes a DAL allocation to the user in a text file.

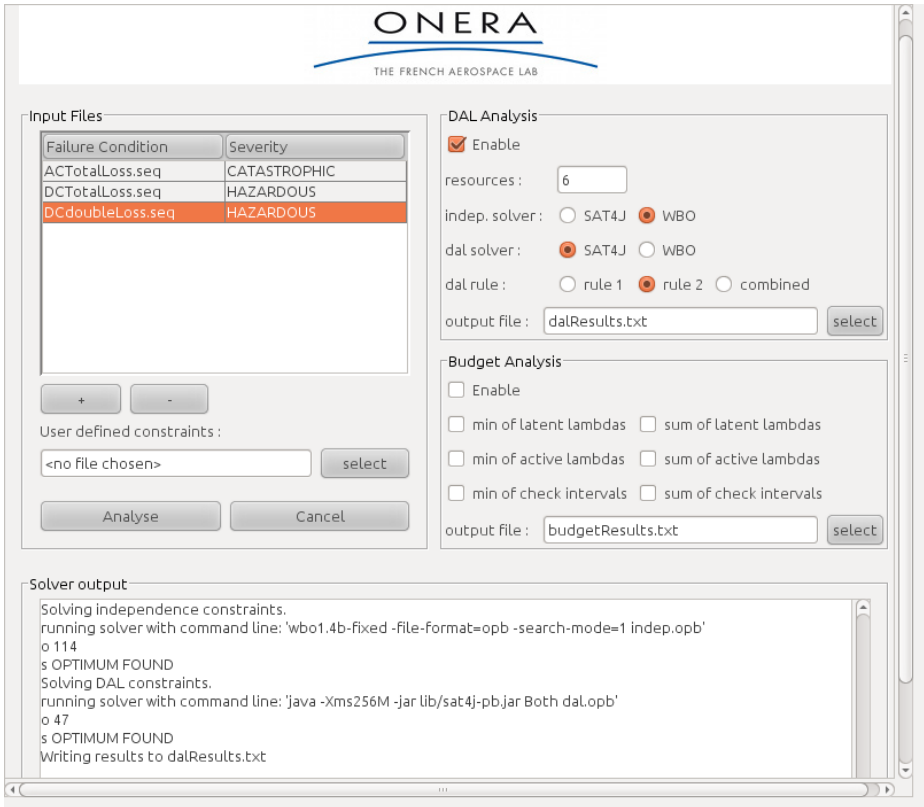


Fig. 2. Screen Capture of the DALculator GUI

### 4.3 Lessons Learnt from the First Experimentations

We tested the DALculator on two classes of examples. The first class contains simple examples in the style of the Data Measurement and Display system. They were used in order to validate the results of the DALculator. The size of the minimal cut sets in this class ranges from 11 to 51, making it possible to check by hand that the results are correct. The second class contains more complex systems extracted from industrial models. This includes minimal cut sets computed for an Electrical Distribution and Generation system as well as a Flight Control system. These models were used in order to test the performance of the tool.

The first experiments showed that DAL allocation and independence proposed by the DALculator can be optimal but not very intuitive. For instance, in the Data Measurement and Display system we first hand-built a solution with 16 independent pairs, considering that all Left items are independent from the Right items. This is far from the optimal solution with only 7 independent pairs found by the tool and presented in section 3.1.

To help the user explore various independence relations and DAL allocations, the tool can load directives that will be taken into account in the constraint problem. So

far, the user can specify that two items have to be independent, or that they shall not be independent. Similarly, the user can specify that the DAL of an item shall be upper bounded to a given level.

Thanks to these user directives it is also possible to use the DALculator to check that an existing allocation is correct. The user enters a complete independence relation or a complete DAL allocation and if the resulting constraints are consistent it means that the proposed allocation is consistent with the DAL allocation rules.

**Table 4.** Performances of DALculator

Name	MCS	size	Sev	Perfs		Idp	DAL		
				Idp	DAL		max	R1	R2
DataDisplay0	11	4	HAZ	0.004	0.004	7	12	12	8
DataDisplay1	39	6	HAZ	0.011	0.005	7	9	9	9
DataDisplay2	51	9	HAZ	0.039	0.007	7	12	12	8
Elec	165	3	HAZ	0.007	0.006	16	18	15	11
Elec	447	4	CAT	0.060	0.066	102	70	60	56
FlightControl	3746	3	CAT	0.093	0.445	903	183	168	155

Table 4 gives performance figures on representative examples. The **MCS** column gives the number of cut sets of the considered failure condition. The **order** column gives the maximal cut order. The **Sev** column gives the classification of the failure condition. The **Perf** column gives the solving time in seconds for the independence constraint problem (**idp** column) and the DAL allocation problem (**DAL** column). Despite the combinatorial nature of the problem, the run times are relatively low, even on real world examples. This validates our choice of constraint formalism and solving technology, and indicates a good scalability potential to real world problems. The **Idp** column gives the optimal number of independence requirements generated by the independence analysis. This number can be surprisingly lower than in trivial hand made solutions, which demonstrates the added value of optimization. Last, the **DAL** column lists the values of the criterion that is optimized on the **DAL** allocation analysis: the **max** column gives the worst value for a valid allocation encountered by the solver during optimization. **O1** and **O2** columns give the optimal solution computed by the solver for option1 and option2 respectively. Again, the quantitative distance between valid allocations, from the worst encountered to optimal, can be quite important, demonstrating the added value of automatic and formal analysis over hand built solutions.

As often in standards, there is room for interpretation of the rules. In the case of ARP4754A downgrading rules, we estimated that two notions were subject to interpretation:

- How many items in the minimal cut set have to be independent in order to apply the downgrading rules?
- What does the “*remaining items*” cover in the rules that state that “*all remaining items may be downgraded*”?

Our formalization proposes an interpretation of these two notions. We considered that the number of items that needs to be independent is defined by Table 2. And we considered that “remaining items” that can be downgraded are items that are independent from the non-downgraded item, other items shall not be downgraded. We plan to discuss with people in the aeronautics industry and in the certification authorities in order to check that our interpretation is valid.

## 5 Conclusion and Perspectives

### 5.1 Related Work

In [8] several authors including Y. Papadopoulos have proposed an approach for the automated allocation of Automotive Safety Integrity Level (ASIL). The ASIL has a role similar to DAL. Their approach is quite different from ours as it computes the ASIL directly on the basis of the fault-tree structure. We have not considered this approach because we wanted to support safety assessment that use safety propagation models instead of fault-trees. Nevertheless this approach seems interesting to overcome one of the drawback of the approach based on minimal cut sets analysis. The number of MCS can become huge for complex systems and could cause problems for the constraint solvers that we use.

Another interesting feature of this work is that ASIL is allocated to an item fault and not to a function as for the DAL. Consequently different ASIL could be allocated to the loss of an item and to its undetected erroneous behavior. This fine grained allocation could help to select the level of rigor of development techniques; For instance, it could be interesting to use formal techniques to check the absence of errors leading to a fault with a high SIL and use more conventional techniques when considering other faults of the item that were allocated a smaller SIL.

The paper does not describe whether the independence between items is taken into account in the allocation process. In the aeronautics safety process the identification of item independence is very important and cannot be neglected.

In [9] the author describes an approach to allocate the SoftWare Assurance Level SWAL. Again the SWAL plays in Air-Traffic Management a role similar to DAL. The approach analyzes the dependencies among the entities and resources of a distributed system related with their interactions. Then inequalities between the SWAL of these entities are defined. The resolution of these inequalities leads to the allocation of SWAL levels for the components of the distributed system. This approach needs a more detailed description of the system design than what is used in our approach (e.g. minimal cut sets). It could be used at a later stage of the design of the systems, when the software design and implementation is established, in order to check that dependencies are consistent with the DAL that were allocated at the preliminary design stages.

### 5.2 Perspectives

We are currently working on a similar approach for the allocation of Quantitative budgets to function and item faults (see [5]). The tool analyzes minimal cut sets and proposes a mean probability of occurrence for each item fault such that the probability

of the failure condition is acceptable according to its severity. The approach also formalizes the allocation as a Constraint Satisfaction Problem but we use Mixed Integer Linear Programming tools instead of pseudo-Boolean tools to solve the CSP.

More generally one of our research objectives would consist in investigating the integration of models and tools based on constraint satisfaction problems for avionics design assistance. This could contain tools for independence and DAL allocation, allocation of Modular avionics modules and communication path [11], real-time scheduling of tasks [12].

**Acknowledgements.** The research described by this paper was partially supported by the European Union FP7 project MISSA. The authors would like to thank Jean-Pierre Heckmann and Alessandro Landi for their explanations about DAL allocation rules.

## References

1. SAE S-18 and EUROCAE WG-63 committees: ARP4754a - Guidelines for Development of Civil Aircraft and Systems, SAE aerospace (2010)
2. Baufreton, P., Blanquart, J.-P., Boulanger, J.-L., Delseny, H., Derrien, J.-C., Gassino, J., Ladier, G., Ledinot, E., Leeman, M., Quéré, P., Ricque, B.: Multi-Domain Comparison of Dependability and Safety Standards. In: Proceedings of ERTS (2010), <http://www.erts2010.org>
3. RTCA SC167 and EUROCAE WG-12 committees: RTCA/DO-178B - Software Considerations in Airborne Systems and Equipment Certification, RTCA Inc. (1992)
4. Bozzano, M., Villafiorita, A., Åkerlund, O., Bieber, P., Bougnol, C., Böde, E., Bretschneider, M., Cavallo, A., Castel, C., Cifaldi, M., Cimatti, A., Griffault, A., Kehren, C., Lawrence, B., Luedtke, A., Metge, S., Papadopoulos, C., Passarello, R., Peikenkamp, T., Persson, P., Seguin, C., Trotta, L., Valacca, L., Zacco, G.: ESACS: an integrated methodology for design and safety analysis of complex systems. In: Proceedings of ESREL 2003. Balkema publisher, Rotterdam (2003)
5. Bieber, P., Delmas, R., Seguin, C.: Derivation of Qualitative and Quantitative Safety Requirements. To appear in: ESREL 2011. Balkema, Rotterdam (2011)
6. Manquinho, V., Martins, R., Lynce, I.: Improving Unsatisfiability-based Algorithms for Boolean Optimization. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 181–193. Springer, Heidelberg (2010)
7. SAT4J, <http://www.sat4j.org>
8. Papadopoulos, Y., Walker, M., Reiser, M.-O., Weber, M., Chen, D.-J., Törngren, M., Servat, D., Abele, A., Stappert, F., Lönn, H., Berntsson, L., Johansson, R., Tagliabo, F., Torchiario, S., Sandberg, A.: Automatic allocation of safety integrity level. In: Workshop on Critical Automotive Applications: Robustness & Safety, CARS 2010 (EDCC Workshop), Valencia, Spain (April 27, 2010)
9. Pecchia, A.: Una metodologia per la definizione dei livelli di criticità dei componenti di un sistema software complesso, Master Thesis, Università degli Studi di Napoli Federico II, Italy (2008)
10. Sagaspe, L., Bieber, P.: Constraint-Based Design and Allocation of Shared Avionics Resources. In: 26th AIAA-IEEE Digital Avionics Systems Conference, Dallas (2007)
11. Aleti, A., Bjoernander, S., Grunske, L., Meedeniya, I.: ArcheOpterix: An extendable tool for architecture optimization of AADL models, in Model-based Methodologies for Pervasive and Embedded Software (MOMPES), Workshop at ICSE 2009 ACM and IEEE Digital Libraries (2009)

# Towards Cross-Domains Model-Based Safety Process, Methods and Tools for Critical Embedded Systems: The CESAR Approach

Jean-Paul Blanquart<sup>1,\*</sup>, Eric Armengaud<sup>2,3</sup>, Philippe Baufreton<sup>4</sup>,  
Quentin Bourrouilh<sup>3</sup>, Gerhard Griessnig<sup>3</sup>, Martin Krammer<sup>2</sup>, Odile Laurent<sup>5</sup>,  
Joseph Machrouh<sup>6</sup>, Thomas Peikenkamp<sup>7</sup>, Cecile Schindler<sup>5</sup>, and Tormod Wien<sup>8</sup>

<sup>1</sup> Astrium Satellites, 31 rue des cosmonautes, 31402 Toulouse Cedex 4, France  
jean-paul.blanquart@astrium.eads.net

Tel.: +33 5 62 19 69 56, Fax: ...71 58

<sup>2</sup> Virtual Vehicle Research and Test Center, Graz, Austria  
martin.krammer@v2c2.at

<sup>3</sup> AVL, Graz, Austria

{eric.armengaud,quentin.bourrouilh,gerhard.griessnig}@avl.com

<sup>4</sup> Sagem Défense Sécurité, Massy, France

philippe.baufreton@sagem.com

<sup>5</sup> Airbus Operations, Toulouse, France

odile.laurent@airbus.com, cecile.schindler@apsys.eads.net

<sup>6</sup> Thales, Palaiseau, France

joseph.machrouh@thalesgroup.com

<sup>7</sup> OFFIS, Oldenburg, Germany

peikenkamp@offis.de

<sup>8</sup> ABB, Billingsstad, Norway

tormod.wien@no.abb.com

**Abstract.** The CESAR project<sup>1</sup> aims at elaborating a Reference Technology Platform usable across several application domains (Aeronautics, Automotive, Industrial Automation, Railway and Space) for the cost effective development and validation of safety related embedded systems. Safety and, more generally, dependability are therefore major topics addressed by the project. This paper focuses on the work performed on safety requirements and approaches to be supported by a common Reference Technology Platform. We analyse and compare the industrial practice, applicable standards and state of the art so as to identify which and how safety views should be supported. We focus in particular on the major axes investigated by the project, formal model-based techniques for requirements engineering and component-based engineering. Preliminary realisations and case studies confirm the interest and provide refined requirements for the final version of the platform.

**Keywords:** Safety, dependability, embedded systems, standards, multi-domains, development and validation platform.

---

\* Corresponding author.

<sup>1</sup> The CESAR project (“Cost efficient methods and processes for safety relevant embedded systems”) has received funding from the ARTEMIS Joint Undertaking under grant agreement n° 100016 and from specific national programs and/or funding authorities.

## 1 Introduction

The industry of safety critical embedded systems faces difficult challenges with more and more complex systems with more and more functions and interacting functions, very strong requirements on safety and safety justification, and very strong constraints on cost and time-to-market.

In this context, advanced engineering methods appear as very promising, and in particular formal model-based methods addressing requirements engineering and component-based engineering. Building on these approaches, the CESAR European project (Joint Undertaking ARTEMIS) gathers more than 50 partners from academia, technology providers and industrial end-users from five application domains (Aeronautics, Automotive, Industrial Automation, Railway and Space). They aim at elaborating a Reference Technology Platform (RTP) usable across several application domains for the cost effective development and validation of safety related embedded systems. Safety and, more generally, dependability are therefore major topics addressed by the project.

This paper reports on the objectives, work and current achievements of the project from the safety point of view. This encompasses a survey of standards, state of the practice and of the art, the elicitation of requirements for the RTP, and the elaboration and implementation of solutions in requirements engineering and component-based engineering. Preliminary assessments on industrial use cases provide refined requirements for the final version of the RTP.

## 2 State of the Art and of the Practice

Altogether the academic, technology and industrial partners of the CESAR project elaborated a collection and synthesis of the state of the practice and state of the art in safety related embedded systems in all five application domains covered by the project: aeronautics, automotive, industrial automation, railway and space, addressing the applicable standards, the industrial practice, state of the art, and identifying the support expected from the CESAR Reference Technology Platform, in particular under the form of “safety views”.

### 2.1 Safety Standards

In terms of safety, the concern of this paper, we especially focused on the safety standards applicable to the target application domains and to their analysis and comparison:

- Aeronautics: Eurocae/SAE documents ED-79A/ARP-4754A “Guidelines for Development of Civil Aircraft and Systems” [1] with the complements on methods and techniques ED-135/ARP-4761 “Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment” [2], completed by the Eurocae/RTCA documents on software (ED-12B/DO-178B “Software considerations in airborne systems and equipment certification” [3] and ED-80/DO-254 “Design Assurance Guidance for Airborne Electronic Hardware” [4]);

- Automotive: the new standard, in Final Draft International Standard (FDIS) status expected to be published in 2011, ISO/FDIS 26262 “Road vehicles – Functional safety” [5];
- Industrial automation: the generic standard IEC 61508 “Functional safety of electrical/electronic/ programmable electronic safety-related systems” [6] and its derived standards such as IEC 61511 “Functional safety – Safety instrumented systems for the process industry sector” [7];
- Railway: the EN CENELEC standards 50126 (“Railway applications – The specification and demonstration of reliability, availability, maintainability and safety (RAMS)”) [8], 50128 (“Railway applications – Communications, signalling and processing systems – Software for railway control and protection systems”) [9] and 50129 (“Railway applications – Communications, signalling and processing systems – Safety related electronic systems for signalling”) [10];
- Space the ECSS standards Q30 (“Space product assurance – dependability”) [11], Q40 (“Space product assurance – safety”) [12] and Q80 (“Space product assurance – software”) [13].

Even though there are differences in the standards applicable to the various domains, there are also many strong common principles and approaches [14], confirming the interest of a common platform to support the development and validation of safety-critical or safety-related embedded systems in various domains. In particular all analyzed standards propose a top-down risk-based approach and the consideration of several levels or categories for the consequences of failures and therefore categories for the systems and elements. This leads in particular to categories of requirements applicable to the development and assurance in consideration of the severity of failure consequences and associated overall occurrence probability of a system failure.

There are however also many differences, for instance in the details of the rules to allocate categories along the design, as well as on the architecture principles (e.g., with focus on “integrated safety” in aeronautics and automotive versus “external safety monitoring” in automation, rail and space), on the nature of the standards (e.g., prescriptive in terms of objectives versus means, or promoting explicitly or not the safety demonstration under the form of a “safety case”). Consequently the CESAR Reference Technology Platform cannot be a single common platform shared “as is” by all users, but rather a generic platform with all necessary facilities to be instantiated in each domain taking in consideration also the various industrial practices as well as the most promising methods and tools to support the development of safety critical embedded systems and in particular formal model-based and component-based approaches for requirements engineering and development.

## 2.2 Safety Process and Methods: The Safety Views

Industrial practice, following applicable standards, has developed and implemented a safety approach based on a strong process combining a global top-down safety construction and assurance process, supported by several methods and techniques aiming at supporting the elaboration and evaluation of architecture solutions and their implementation. The support that the CESAR Reference Technology Platform is



expected to provide to these safety process, methods and techniques can be described through the notion of safety views and viewpoints.

Following IEEE 1471 [15] we define a view as “a representation of a whole system from the perspective of a related set of concerns”, and a viewpoint as “a specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis”. Therefore a view conforms to a viewpoint that establishes the conventions by which this view is depicted, created and analyzed. The viewpoint determines the languages to be used to describe the view, and any associated modelling method or analysis technique to be applied to the view. So, a viewpoint is described by: objectives, a set of concerns, modelling and language features and analytic methods. So, a view may consist of several models. And a model may participate in several views.

In the context of this work we propose the relevant safety and dependability views and viewpoints based on the identification of the relevant safety and dependability analysis techniques, and on which models these analyses could be performed. Therefore a safety and dependability viewpoint is defined as:

- A model i.e., a set of information and their inter-relationships, with formal semantics of information and relations (the modelling and language features that allow performing the analyses of concern),
- A set of analytic methods that can be applied to the model so as to analyse and assess some predefined safety and dependability properties.

Various categories of safety and dependability analysis techniques can be identified, corresponding to various categories of objectives, depending on the different stakeholders concerned with the safety of the system to be developed. Depending on the domain the stakeholders may be represented by different roles in the development process: e.g. a dedicated quality or safety manager is compliant with the standards applicable for the domain. Within the development process they also represent certification authorities, as well as customers concern with respect to safety.

Additionally, safety is a concern across the different engineering phases, implying various approaches depending on the development phase addressed. Early stages of the safety lifecycle aim at the identification of requirements and exploration of the implications of design whereas later lifecycle stages focus on the successful implementation of the requirements. Typical safety analysis techniques as in particular required by safety standards include:

- Hazard analysis and risk assessment,
- System Safety Assessment, generally supported by Fault Tree Analysis, Failure Modes, Effects (and Criticality) Analysis, Common Cause Analysis, etc.,
- Verification and validation, in particular of the implementation of safety and functional safety requirements.

These various safety analysis techniques can be classified, taking into consideration different categories of their objectives:

**Quantitative (Probabilistic) Assessment of Safety and Dependability Properties**

The purpose is to evaluate global probabilistic properties of a system (or of a part of it) based on its architecture (especially in terms of redundancies) and a set of assumptions on the stochastic distribution of failures that can affect the system and its elements. There are numerous modelling approaches and associated tools such as Markov Chains, Reliability Block Diagrams, Fault Trees etc., and component failure rate data bases. Even though some difficulties and limitations may exist, they are mainly related to the relevance of the component failure rates (for which model-based approaches in the sense of the study are of little support if any) rather than on the capability to combine them and evaluate safety and dependability probabilistic properties at a higher level. Nevertheless in case of complex system architecture and reconfiguration mechanisms, elaborating a model to conduct these evaluations remain tedious and error prone. It is therefore expected that the CESAR RTP will provide facilities to support the elaboration of such models.

**Qualitative (Descriptive and/or Deterministic) Assessment of Propagation of Faults and Failures and of the Effects of this Propagation**

This corresponds to a very important and large set of analyses of the effectiveness of detection and protection mechanisms against faults and their combinations, analyses of common mode failures or common cause faults and their effects, etc. This must be a major target for CESAR considering the importance of the associated objectives and the limitations of the currently used techniques (e.g., Failure Modes, Effects and Criticality Analysis, Fault Tree Analysis and various check-lists). An assisted, if not automated, generation of the analysis reports from an engineering model annotated with the relevant information about fault occurrence and propagation is expected to be particularly useful.

**Assessment (Correctness, Performance) of Fault Tolerance Mechanisms**

Similarly to the correctness or performance assessment of any function, model-based approaches can be used provided they represent the behaviour of the function as well as, for performance evaluation, its utilisation of the relevant resources with respect to the target evaluation (time, memory, etc.). This can also be identified as an important target for CESAR because of the current difficulties and limitations of the definition and validation techniques applicable to fault tolerance mechanisms. Precisely these difficulties come from the increasing complexity of the fault tolerance mechanisms and their detailed behaviour, taking into account multiple interactions between a large number of elements (and moreover in nominal and degraded conditions).

**Soundness, Completeness of the Safety and Dependability Arguments**

Formal models could be used to represent and check the safety and dependability logical argumentation (how the various pieces of evidence are logically combined to support a high level safety or dependability claim). This is mentioned here mainly for completeness because despite its interest in general and its possible inclusion among the topics addressed by the CESAR project, it is very different in nature from the techniques addressed by the study in the sense that the concerned models are specific

to the considered safety and dependability analysis, rather than engineering models (even augmented and annotated with safety and dependability specific information).

Finally we can identify three main categories of objectives of safety analyses, corresponding to a priori three categories of safety models, structural safety models, behavioural safety models and logical safety models.

### **Structural Safety Models**

Structural models are often used as a support to the engineering activities, to represent the organisation of the various elements composing a system (at various abstract levels: functions, sub-systems, equipment, software, etc. and from the early definition and design phases to the implementation, verification and validation). Structural models can be extended to represent how the faults and failures affect the various elements of the structure, and how these faults and failures propagate along the structure. Such augmented models are good candidates to support, possibly through coupling with classical existing safety and dependability analysis tools, the analysis of fault propagation and demonstration of the related requirements (e.g., all single faults can be handled through automatic reconfiguration, or switch to survival mode, no combination of two independent faults can have catastrophic consequences, etc.). Of course not all the characteristics of fault propagation can be easily represented on structural engineering models (e.g., thermal, electromagnetic compatibility, etc.). The methods and support, object of the project, are not expected to solve all the difficulties but at least a significant part.

At minimum it is expected that the CESAR RTP provides support to the classical analysis techniques as requested by the various safety standards and in particular Hazard Analysis, Fault Tree Analysis, FMECA (Failure Modes, Effects and Criticality Analysis). It is worth noting that according to the definition we proposed, the above mentioned analysis techniques or more precisely their underlying formalism (fault trees, FMECA sheets) could be considered as “safety viewpoints” in the sense that they describe information and their relation, with a (more or less) rigorous semantics, and support analysis in the sense of the assessment of properties such as for instance the minimum number of faults leading to some feared event. However, all these techniques (and some other such as Event Trees, HAZOP (Hazard and Operability studies), etc.) are based on the same fundamental principle (the expression and processing of the propagation of faults). Previous work has confirmed that fault propagation can be appropriately described on structural models, very close to the models of the system architecture as used in system engineering with some additional properties associated to the system elements (how they fail, how they react to faults) and links. In particular it is possible to extract automatically from such enriched structural models the more specialised models for safety analysis such as fault trees or FMECA sheets.

We therefore propose to not focus on these specialised models and consider principally this more general structural model as a safety viewpoint. It is expected from the CESAR RTP to support the elaboration of this viewpoint, in full consistence with the engineering models, and support the extraction from it of specialised models such as FMECA and fault trees, and the assessment of safety properties as needed by end-users.

### **Behavioural Safety Models**

Behavioural models are often used as a support to the engineering activities and they are in particular more and more used thanks to the improvements of the associated tools allowing formal verification of behavioural properties i.e., the correctness of the behaviour even in case of complex behaviour with many interactions between a large number of elements and a large number of possible combinations of events and states. Fault tolerance is generally a particularly difficult case in this respect, and moreover the severity of consequences of potential failures of the fault tolerance mechanisms may be very high.

### **Logical Safety Models**

Though not required explicitly by all safety standards (but notable exceptions are in railway and automotive domains), the notion of safety case is very useful to organise and check the safety arguments and claims. Be it called or not a “safety case”, the formal expression of the logical structure of the arguments (safety objectives, claims, assumptions etc.) corresponds clearly to a “safety viewpoint” and is of practical interest for end-users. Moreover experience, formalisms and tools (such as around the Goal Structured Notation) are available in this area.

### **Coupling Safety Models**

A complete safety view encompasses structural, behavioural and logical models and also, in terms of behaviour, the complete detailed model of the behaviour of the system and of its fault tolerance mechanisms, in presence and in absence of faults. However modelling has something to do with the notion of abstraction and the aim should not be to model everything, but what is necessary as regards the modelling objectives. Here we are mainly concerned with safety analysis techniques and the safety views identified and proposed above correspond to the needs related to safety analysis practice and standards in CESAR application domains.

Therefore, due to the particular nature of fault tolerance behaviour (with inputs and outputs directly from and to the system structure and fault propagation), it is of major interest that the CESAR RTP supports the expression and processing of fault tolerance behaviour directly and formally coupled to the structural safety model, itself directly and formally coupled to the (engineering) architectural model(s) of the system, properly enriched with the necessary information on fault propagation.

## **3 Safety Requirements for the Reference Technology Platform**

The goal of this process is to collect the safety requirements that will be taken into account within the CESAR RTP. These requirements were collected from different sources and especially the safety experts in each application domain (Aeronautics, Automotive, Industrial Automation, Railway, and Space) involved in the project. During this phase, all requirements were reviewed one by one, analysed and classified (figure 1). Each requirement was given the status: accepted, pending or rejected. Requirements needing clarification or rewording, were assigned to one or more partners.

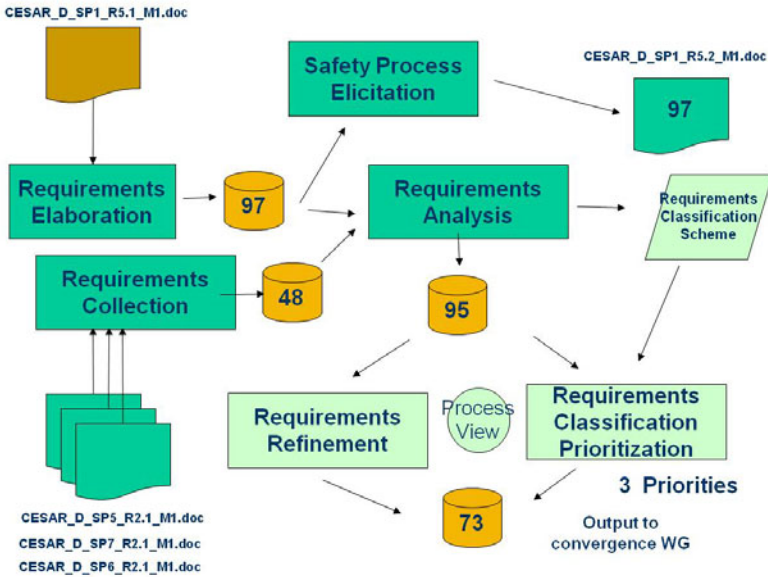


Fig. 1. Safety requirements refinement process

In order to have common cross-domains requirements, we first analyzed all previously collected requirements. This analysis consisted in eliminating the requirements already taken into account. After this step, only 95 requirements have been selected. These requirements were then classified according to their priority, understood here as a combination of priority in terms of time (referring to the CESAR project timescale and development plan) and of priority in terms of importance (referring to the expected business impact for CESAR end-users).

The last step consisted in classifying the resulting requirements within 5 categories:

- Management
- Process
- Modelling
- Safety Analysis
- Diagnosability

## 4 Safety Process

One of the safety activities within the CESAR project is the modelling of a safety process, as enforced by applicable standards and implemented in industrial practice. Processes are usually arranged orthogonal to organizational structures, thus spreading across several organizational units of different domains. In terms of embedded systems, the development of hardware and software usually follows well-proven development processes, often based on several years of experience. Therefore

changes are to be made carefully. The assurance of functional safety features is achieved by embedding related activities to the currently running development processes. To achieve this, several steps need to be carried out.

#### 4.1 Challenges of Process Modelling

First, a detailed analysis of the safety process is performed. The first part of this analysis, for automotive, is based on the upcoming *ISO 26262: Road vehicles – Functional Safety* [5]. This new standard covers all activities during the life cycle of safety related electrical/electronic systems in passenger cars, and will be released in 2011. While IEC 61508 [6] was considered for a wide range of electrical/electronic systems, ISO 26262 clearly aims at the automotive industry and affects original equipment manufacturers (OEMs) and suppliers. The core parts of ISO 26262 are covering main areas of product development, starting with a *concept phase* and the *product development at the system level*, including system specification, integration and validation activities. Furthermore, *product development at the hardware level* and *product development at the software level* are covered. Additional parts are describing requirements and recommendations for production and operation, supporting processes and ASIL oriented safety analyses. The concept of ASILs (automotive safety integrity level) is derived from the idea of SILs (safety integrity levels, IEC 61508), and is used to classify hazards, based on their severity (S), controllability (C) and exposure (E) for a number of relevant situations. Depending on the necessary level of confidence, the methods applied during the item development may vary. This fact introduces great variability to development processes. ISO 26262 defines *work products* as results of single tasks, which may be input to other tasks. Therefore, work products create strong dependencies between all parts and domains of the standard. A work product can be a new separate document or just a reference to an existing document. For the latter case, an appropriate mapping is required.

For the analysis of ISO 26262, all clauses and requirements of the standard were translated to applicable practices and concrete activities to be performed by the industrial organizations. This means that the requirements of the standard were regrouped to activities that are described, assigned to different roles, and associated with their corresponding inputs, outputs and potential necessary tools. To perform this step, a spreadsheet, so-called “safety framework” was developed, based on the previously outlined core parts of ISO 26262.

Figure 2 shows an excerpt of the safety framework. Note that the information contained in the spreadsheet is split into two different font types: The **bold** information evolves directly from standard (e.g. columns “Phase”, “Sub-phase”, “Objectives”, “Work products”, “Activities”), whereas normal font is the interpretation and explanation that is developed in CESAR (e.g. columns “Related project input / documentation”, “What Method / Action brief description”, “Tools”). For legibility reasons only an excerpt of the framework is presented here: For example, only the column “Inputs” is included. However, the same kind of information is provided in the complete safety framework for the outputs as well. Other information not visible in figure 2 is the assignment of roles. ISO 26262 makes no assumptions on roles within the safety life cycle. Therefore we introduce a role model, featuring all necessary roles from different technical domains, categorizing them to *responsible* and *supporting* roles, reducing the imminent risk of ambiguities.

Phase	Sub phase	Objectives	Inputs		Activities	What: Method/Action brief description	Tools
			Workproducts (as defined in the standards)	Related project input / documentation			
<b>System level Specification</b>							
Product Development-System Level	System design	The first objective of this subphase is to develop the system design and the technical safety concept that comply with the functional requirements and the technical safety requirements specification of the item.	Technical safety requirements specification Item integration and testing plan (Optional: Preliminary architectural assumptions, Functional concept (external), Functional safety concept)	Customer specifications	Specification of system design and technical safety concept	The system design specification is developed according to company-internal guidelines (standard process) and is reviewed / updated / extended to ensure that the safety aspects are covered. The technical safety concept is specified based on the technical safety requirements. ASIL decomposition can be used if applicable (see functional safety concept above).	MS Word, MS Visio
		The second objective of this subphase is to verify that the system design and the technical safety concept comply with the technical safety requirements specification.			Definition of architecture requirements	Definition of system architecture (data flow diagram, block diagram, network, etc). The architecture design is developed according to company-internal guidelines (standard process) and is reviewed / updated / extended to ensure that the safety aspects are covered. ASIL decomposition can be used if applicable (see functional safety concept above).	MS Word, MS Visio, Mathworks, MATLAB / Simulink / Stateflow (Structure/ Architecture modelling tools)
					Measures for the avoidance of systematic failures	FMEA and/or FTA are performed to identify causes and effects of systematic failures. Based on	Safety analysis tools: APIS IQ-FMEA,

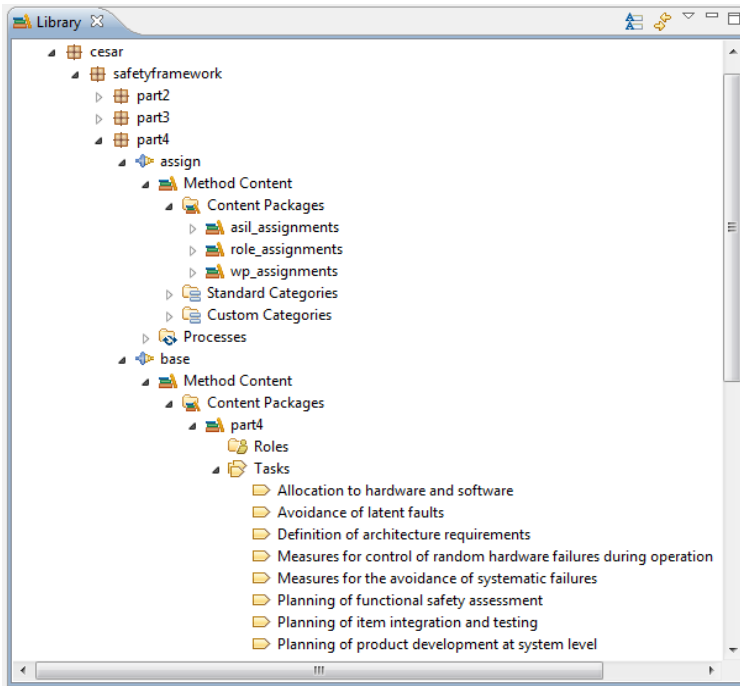
Fig. 2. Excerpt of Safety Framework

After this first analysis focused on the automotive domain, the resulting safety framework is now being extended to other domains with the analysis of their applicable standards [1-4], [6-13]. After that, common properties and methods will be identified, in order to broaden the framework’s spectrum.

Spreadsheets provide a great overview and carry valuable information, however it is difficult for engineers to derive further information from them: Relationships between activities, roles, and other process artefacts are hard to track. Moreover, the representation of workflows is hardly possible and element instances may not be created. Process modelling languages do provide these features, bringing the safety framework closer to its application.

#### 4.2 The CESAR Approach to Process Modelling

The *CESAR Practice Framework* aims at finding new ways for the description and modelling of development processes and practices. The *Software and Systems Process Engineering Metamodel* (SPEM) [16] is used to define all necessary entities. Based on SPEM, the *Eclipse Process Framework (EPF) Composer* [17] is used to realize the vision of a versatile process framework, assisting process engineers and developers from the early beginning of safety related projects. The *Automotive Safety Framework* is targeted at the needs of the automotive domain and will integrate seamlessly to the practice framework, thanks to agreed modelling standards and concepts [18]. Starting with the previously described safety framework table, a mapping between the contained descriptions and the elements provided by the practice framework/EPF was established.



**Fig. 3.** Process elements implementing the safety process in EPF

EPF supports various modelling concepts, allowing storing elements in *packages* and *method plugins*. Most important concepts are affecting reusability and extensibility. These properties are preserved by creating base plugins, which hold generic definitions. These are extended later on by the creation of *assign plugins*. EPF's *variability* feature was used to create relations between the elements of these two plugins. This modelling strategy was used to achieve a complete library of tasks, roles and work products, separated into different packages according to the core parts of ISO 26262. Furthermore, EPF has proven to be usable for the representation of quality related attributes [19], thus all ASIL related methods of ISO 26262 are represented as guidelines, organized in a hierarchy of packages. An overview of the resulting framework structure is shown in figure 3.

Another contribution to the safety process is the addition of work flows. The resulting library was used to arrange generic work flows, introducing sequences of tasks and activities.

Finally, the resulting safety process library, the work flows and an appropriate process configuration are used to publish processes. In EPF, a publication typically consists of HTML (Hyper Text Markup Language) output files, including full graphics showing all dependencies between tasks, roles, work products and other artefacts. All information is available in XML (Extensible Markup Language) format as well, in order to support other means of automated process enactment. The *RTP*



*Desktop* for example could use this type of process description for automated orchestration and enactment.

## 5 Requirements Engineering

Safety is, besides of the functional aspect, one of the major areas in the field of requirements engineering of the CESAR Project. It has been approached from different perspectives like ontology based support, formalization support and different analyzes ranging from virtual safety integration tests to fault tree generation. Furthermore special requirements from safety standards like ISO 26262 have been considered and the developed methods adapted to suit the needs.

Main principle is always the early detection of safety relevant requirements and early analysability even without having an architecture available.

The supported process of requirement formalization allows to transform requirements from natural language text to semi formal boilerplates (BP) and finally to strictly formal patterns. At each degree of formalization a different set of safety analyses can be applied.

### Ontology Based Analysis

- How to get from NLR to BPs
- TODO HazOP

### Formalizing Safety Requirements

- how to get from BPs to Patterns
  - o Usage of formalization ontology (planned work)
  - o special focus on safety elements in this ontology
- importance of contracts for safety design, always assumptions on surrounding environment
- Safety Patterns, semantic etc...

A : {CMD AS1 fail, CMD AS2 fail} does not occur

{CMD AS1 fail, VALID1 fail} does not occur

G : {perm(CMD AS fail)} does not occur

This contract expressing the expectation that a permanent failure on the CMD output (of a braking system controller) does not occur provided none of the two double failures in the assumption occur. The benefit of the patterns is that – although they have a lot of natural language elements – they come equipped with a formal semantics. For instance, the above assumption is equivalent to the following LTL formula:

$$\neg(F(\text{CMD AS1 fail}) \wedge F(\text{CMD AS2 fail})) \wedge \neg(F(\text{CMD AS1 fail}) \wedge F(\text{VALID1 fail}))$$

Thanks to these features it is possible to perform analyses on these patterns such as completeness and consistency, or analyses of fault propagation and impact on safety such as fault Tree Analysis.

## 6 Preliminary Assessment and Way Forward

In addition to the process modelling activities described in section 4 and to the elaboration of safety oriented requirement engineering support as described in section 5, both being incorporated in the Reference Technology Platform, several use cases and scenarios have been defined by the industrial end-users so as to perform an assessment of the proposed technologies and tools.

At this stage only preliminary assessment has been performed, on a first version of the RTP, mainly to support the refinement of the requirements towards the final version. These experiments include pilot applications from all covered domains e.g., door management system or flight warning system (aeronautics), on recuperation and on power train control unit for hybrid vehicles, airbag control unit, brake by wire (automotive), safe controllers in industrial automation, on-board traffic management unit in railway, or FDIR (Failure Detection, Isolation and Reconfiguration) definition and validation and incremental safety assessment (space).

Though this preliminary assessment was performed in parallel to the actual incorporation of the innovations and their integration in the platform, and therefore mainly focussed on the assessment of basic techniques, methods and tools, it already provided important insights and better understanding of user expectations and remaining work.

It appears in particular that some model-based safety languages and tools such as AltaRica are very powerful and well suited to the addressed problems [20]. However their specialisation may be a drawback and other modelling approaches and languages more widely used for system and software engineering such as AADL (Architecture Analysis and Design Language), UML (Unified Modelling Language) or SysML (Systems Modelling language) could also be used, possibly with some adaptations and extensions [21]. We could for instance elaborate a prototype of an automatic transformation from space systems and FDIR AADL-based models (with specific adaptations and extensions for the purpose of the experiment) to timed automata so as to perform behavioural analysis and demonstration of properties on the temporal behaviour of the FDIR mechanisms [22]. These experiments and case studies confirm the validity and feasibility of the approaches investigated in the CESAR project, towards their integration in the final version of the Reference Technology Platform to be released early 2012 for final assessment by the partners.

## References

1. Guidelines for Development of Civil Aircraft and Systems, EUROCAE ED-79A and SAE Aerospace Recommended Practice ARP 4754A (December 21, 2010)
2. Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment, EUROCAE ED-135 and SAE Aerospace Recommended Practice ARP 4761 (December 1996)
3. Software considerations in airborne systems and equipment certification, EUROCAE ED-12 and RTCA DO-178, issue B (December 1, 1992)
4. Design Assurance Guidance for Airborne Electronic Hardware, EUROCAE ED-80 and RTCA DO-254 (April 2000)

5. Road vehicles – Functional safety, Final Draft International Standard ISO/FDIS 26262: (Parts 1-10) (2010)
6. Functional safety of electrical/electronic/ programmable electronic safety-related systems, IEC 61508 Parts 1-7, Edition 2.0, (April 2010)
7. Functional safety – Safety instrumented systems for the process industry sector, IEC 61511 Parts 1-3, Edition 1.0 (March 2003)
8. Railway applications – The specification and demonstration of reliability, availability, maintainability and safety (RAMS), CENELEC, EN 50126 (February 28, 2007)
9. Railway applications – Communications, signalling and processing systems – Software for railway control and protection systems, CENELEC, EN 50128 (May 15, 2001)
10. Railway applications – Communications, signalling and processing systems – Safety related electronic systems for signalling, CENELEC, EN 50129 (May 7, 2003)
11. Space product assurance – Dependability, European Cooperation for Space Standardisation, ECSS-Q-ST-30C (March 6, 2009)
12. Space product assurance – Safety, European Cooperation for Space Standardisation, ECSS-Q-ST-40C (March 6, 2009)
13. Space product assurance – Software product assurance, European Cooperation for Space Standardisation, ECSS-Q-ST-80C (March 6, 2009)
14. Baufreton, P., Blanquart, J.P., Boulanger, J.L., Delseny, H., Derrien, J.C, Gassino, J., Ladier, G., Ledinet, E., Leeman, M., Quéré, P., Ricque, B.: Multi-domain comparison of safety standards. In: Proceedings of the 5th International Conference on Embedded Real Time Software and Systems (ERTS<sup>2</sup> 2010), Toulouse, France (May 19-21, 2010)
15. Recommended Practice for Architectural Description of Software-Intensive Systems, ANSI/IEEE Std 1471, ISO/IEC 42010:2007 (2007)
16. Object Management Group, Software and Systems Process Engineering Meta-Model, v2.0 (2008)
17. Haumer, P.: Increasing Development Knowledge with EPFC. Eclipse Review (Spring 2006)
18. Cifaldi, M., Lanteri, F.: CESAR Practices Framework – SPEM Mapping Guidelines, Draft 1, CESAR internal document (2010)
19. Chiam, Y.K., Staples, M., Zhu, L.: Representing Quality Attribute Techniques Using SPEM and EPF Composer. In: EuroSPI 2009 (2009)
20. Bieber, P., Blanquart, J.P., Durrieu, G., Lesens, D., Lucotte, J., Tardy, F., Turin, M., Seguin, C., Conquet, E.: Integration of formal fault analysis in ASSERT: Case studies and lessons learnt. In: Proceedings of the 4th International Conference on Embedded Real Time Software (ERTS 2008), Toulouse, France (January 29-February 1, 2008)
21. Rugina, A.E., Blanquart, J.P.: Formal Methods in Space Systems: Lessons Learnt. In: Data Systems in Aerospace, DASIA Conference, Budapest, Hungary (June 1-4, 2010)
22. Blanquart, J.P., Valadeau, P.: Model-based approaches for an improved FDIR development and validation process. In: Data Systems in Aerospace, DASIA Conference, Malta (May 17-20, 2011)

# From Probabilistic Counterexamples via Causality to Fault Trees

Matthias Kuntz<sup>1</sup>, Florian Leitner-Fischer<sup>2</sup>, and Stefan Leue<sup>2</sup>

<sup>1</sup> TRW Automotive GmbH, Germany

<sup>2</sup> University of Konstanz, Germany

**Abstract.** In recent years, several approaches to generate probabilistic counterexamples have been proposed. The interpretation of stochastic counterexamples, however, continues to be problematic since they have to be represented as sets of paths, and the number of paths in this set may be very large. Fault trees (FTs) are a well-established industrial technique to represent causalities for possible system hazards resulting from system or system component failures. In this paper we suggest a method to automatically derive FTs from counterexamples, including a mapping of the probability information onto the FT. We extend the structural equation approach by Pearl and Halpern, which is based on Lewis counterfactuals, so that it serves as a justification for the causality that our proposed FT derivation rules imply. We demonstrate the usefulness of our approach by applying it to an industrial case study.

## 1 Introduction

In recent joint work [1] with our industrial partner TRW Automotive GmbH we have proven the applicability of stochastic formal analysis techniques to safety analysis in an industrial setting. In [1] we showed that counterexamples are a very helpful means to understand how certain error states representing hazards can be reached by the system. While the visualization of the graph structure of a stochastic counterexample [2] helps to analyze the counterexamples, it is still difficult to compare the thousands of paths in the counterexample with each other, and to discern causal factors during fault analysis. In safety analysis, fault tree analysis (FTA) [21] is a well-established industrial method and graphical notation to break down the hazards occurring in complex, technical systems into a combination of what is referred to as basic events, which represent system component failures. The main drawback of fault tree analysis is that it relies on the ability of the engineer to manually identify all possible component failures that might cause a certain hazard. In this paper we present a method that automatically generates a fault tree from a probabilistic counterexample. Our method provides a compact and concise representation of the system failures using a graphical notation that is well known to safety engineers. At the same time the derived fault tree constitutes an abstraction of the probabilistic counterexample since it focuses on representing the implied causalities rather than enumerating all possible execution sequences leading to a hazard. The causality expressed by the fault tree is rooted

in the counterfactual notion of causality that is widely accepted in the literature. Our approach can be described by identifying the following steps:

- Our fault tree computation method uses a system model given in the input language of the PRISM probabilistic model checker [14].
- For this model we compute counterexamples for stochastic properties of interest, representing system hazards, using our counterexample computation extension of PRISM called DiPro [2]. The counterexamples consist of potentially large numbers of system execution paths and their related probability mass information.
- In order to compute fault trees from these counterexamples we compute what is commonly referred to as basic events. Those are events that cause a certain hazard. The fault tree derivation is implemented in a tool called CX2FT.
- The justification for the fault tree computation is derived from a model of causality due to Halpern and Pearl [12] that we modify and extend to be applicable to our setting.
- The path probabilities computed by the stochastic model checker are then mapped on the computed fault tree.
- Finally, the obtained fault tree is represented graphically by an adapted version of the FaultCAT tool [1].

All analysis steps are fully automated and do not require user intervention. We demonstrate the usefulness of our approach by applying it to a selection of case studies known from the literature on stochastic model checking.

This paper is organized as follows: In Section 2 we briefly introduce the concepts of counterexamples in stochastic model checking and fault trees. In Section 3 we describe the model of causality that we use, and how probabilistic counterexamples can be mapped to fault trees. In Section 4 we demonstrate our approach on a case study known from the literature. A discussion of related work follows in Section 5. Finally, Section 6 concludes the paper.

## 2 Counterexamples and Fault Trees

In stochastic model checking, the property that is to be verified is specified using a variant of temporal logic. The temporal logic used in this paper is Continuous Stochastic Logic (CSL) [4]. Given an appropriate system model and a CSL property, stochastic model checking tools such as PRISM [14] can verify automatically whether the model satisfies the property. Stochastic model checkers do not automatically provide counterexamples, but the computation of counterexamples has recently been addressed in, amongst others, [3, 13]. For the purpose of this paper it suffices to consider only upper bounded probabilistic timed reachability properties. They require that the probability of reaching a certain state, often corresponding to an undesired system state, does not exceed a certain upper probability bound  $p$ . In CSL such properties can be expressed by formulae

---

<sup>1</sup> <http://www.iu.hio.no/FaultCat/>

of the form  $\mathcal{P}_{\leq p}(\varphi)$ , where  $\varphi$  is path formula specifying undesired behavior of the the system. A counterexample for an upper bounded property is a set  $\Sigma_C$  of paths leading from the initial state to a state satisfying  $\varphi$  such that the accumulated probability of  $\Sigma_C$  violates the probability constraint  $\leq p$ . If the CSL formula  $\mathcal{P}_{=?}(\varphi)$  is used, the probability of the path formula  $\varphi$  to hold is computed and the counterexample contains all paths fulfilling  $\varphi$ . The probability of the counterexample is computed using a stochastic model checker, in our case PRISM. Notice that in the setting of this paper the counterexample is computed completely, i.e., all simple paths leading into the undesired system state are enumerated in the counterexample. Fault trees (FTs) [21] are being used extensively in industrial practice, in particular in fault prediction and analysis, to illustrate graphically under which conditions systems can fail, or have failed. In our context, we need the following elements of FTs, for an in-depth discussion of FTs we refer the reader to [21].

1. Basic event: represents an atomic event.
2. *AND*-gate: represents a failure, if all of its input elements fail.
3. *OR*-gate: represents a failure, if at least one of its input elements fails.
4. Priority-AND (*PAND*): represents a failure, if all of its input elements fail in the specified order. The required input failure order is usually read from left to right.
5. Intermediate Event: failure events that are caused by their child nodes. The probability of the intermediate event to occur is denoted by the number in the lower right corner. A top level event (TLE) is a special case of an intermediate event, representing the system hazard.

### 3 Computing Fault Trees from Counterexamples

**Inferring Causality.** Fault Trees express causality, in particular they characterize basic events as being causal factors in the occurrence of the top-level event in some Fault Tree. The counterexamples that we use to synthesize these causal relationships, however, merely represent possible executions of the system model, and not explicitly causality amongst event occurrences. Each path in the counterexample is a linearly ordered, interleaved sequence of concurrent events. The question is hence how, and with which justification, we can infer causality from the sets of linearly ordered event sequences that we obtain in the course of the counterexample computation. We use the concept of *structural equations* as proposed by Halpern and Pearl [12] as a model of causality. It is based on *counterfactual* reasoning and the related *alternative world* semantics of Lewis [17,9]. The counterfactual argument is widely used as the foundation for identifying faults in program debugging [22] and also underlies the formal fault tree semantics proposed in [20]. The "naive" counterfactual causality criterion according to Lewis is as follows: event  $A$  is causal for the occurrence of event  $B$  if and only if, were  $A$  not to happen,  $B$  would not occur. The testing of this condition hinges upon the availability of alternative worlds. A causality can be inferred if there is a world in which  $A$  and  $B$  occur, whereas in an alternative world neither  $A$  nor  $B$

occurs. The naive interpretation of the Lewis counterfactual test, however, leads to a number of inadequate or even fallacious inferences of causes, in particular if causes are given by combinations of multiple events. The problematic issues include common or hidden causes, the disjunction and conjunction of causal events, the non-occurrence of events, and the preemption of failure causes due to, e.g., repair mechanisms. A detailed discussion of these issues is beyond the scope of this paper, and we refer to the critical literature on counterfactual reasoning, e.g., [9]. Since we are considering concurrent systems in which particular event interleavings may be the cause of errors, e.g., race conditions, the order of occurrence of events is a potential causal factor that cannot be disregarded. Consider a railroad crossing model in which  $G$  denotes the gate closing,  $O$  the gate opening,  $T$  the train crossing the road, and  $C$  the car crossing the tracks. A naive counterfactual test will fail to show that the event sequence  $\langle G, O, T, C \rangle$  is a potential cause of a hazard, whereas  $\langle G, T, O, C \rangle$  is not. In addition, the naive counterfactual test may determine irrelevant causal events. For instance, the fact that the train engineer union has decided not to call for a strike is not to be considered a cause for the occurrence of an accident at the railroad crossing. Halpern and Pearl extend the Lewis counterfactual model in [12] to what they refer to as *structural equation model* (SEM). It encompasses the notion of causes being logical combinations of events as well as a distinction of relevant and irrelevant causes. However, the structural equation model does not account for event orderings, which is a major concern of this paper. We now sketch an actual cause definition adopted from [12]. An actual cause is a cause in which irrelevant events are factored out. A causal formula in the SEM is a boolean conjunction  $\psi$  of variables representing the occurrence of events. We only consider boolean variables, and the variable associated with an event is true in case that event has occurred. The set of all variables is partitioned into the set  $U$  of *exogenous* variables and the set  $V$  of *endogenous* variables. Exogenous variables represent facts that we do not consider to be causal factors for the effect that we analyze, even though we need to have a formal representation for them so as to encode the "context" ([12]) in which we perform causal analysis. An example for an exogenous variable is the train engineer union's decision in the above railroad crossing example. Endogenous variables represent all events that we consider to have a meaningful, potentially causal effect. The set  $X \subseteq V$  contains all events that we expect jointly to be a candidate cause, and the boolean conjunction of these variables forms a causal formula  $\psi$ . The causal process comprises all variables that mediate between  $X$  and the effect that  $\psi$  is causing. Those variables are not root causes, but they contribute to rippling the causal effect through the system until reaching the final effect. Omitting a complete formalization, we assume that there is an actual world and an alternate world. In the actual world, there is a function  $val_1$  that assigns values to variables. In the alternate world, there is a function  $val_2$  assigning potentially different values to the variables. In the SEM, a formula  $\psi$  is an actual cause for an event represented by the formula  $\varphi$ , if the following conditions are met:

AC1: Both  $\psi$  and  $\varphi$  are true in the actual world, assuming the context defined by the variables in  $U$ , and given a valuation  $val_1(V)$ .

AC2: The set of endogenous events  $V$  is partitioned into sets  $Z$  and  $W$ , where the events in  $Z$  are involved in the causal process and the events in  $W$  are not involved in the causal process. It is assumed that  $X \subseteq Z$  and that there are valuations  $val_2(X)$  and  $val_2(W)$  assigning values to the variables in  $X$  and  $W$ , respectively, such that:

1. Changing the values of the variables in  $X$  and  $W$  from  $val_1$  to  $val_2$  changes  $\varphi$  from true to false.
2. Setting the values of the variables in  $W$  from  $val_1$  to  $val_2$  should have no effect on  $\varphi$  as long as the values of the variables in  $X$  are kept at the values defined by  $val_1$ , even if all the variables in an arbitrary subset of  $Z \setminus X$  are set to their value according to  $val_1$ .

AC3: The set of variables  $X$  is minimal: no subset of  $X$  satisfies conditions AC1 and AC2.

AC2(1) corresponds to the Lewis counterfactual test. However, as [12] argue, AC2(1) is too permissive, and AC2(2) constrains what is admitted as cause by AC2(1). Minimality in AC3 ensures that only those elements of the conjunction that are essential for changing  $\varphi$  in AC2(1) are considered part of the cause; inessential elements are pruned.

**Formal Representation of Events and their Order.** To logically reason about the causality of events in our setting we need to allow for the description of conjunctive and disjunctive occurrence of events and represent, at the same time, the order in which the events occur. In the common description of the structural equation model the occurrence of events is encoded as boolean formulae. In these formulae, boolean variables represent the occurrence of an event (true = event occurred, false = event did not occur). These variables are connected via the boolean and- or or-operators to express conjunctive or disjunctive constraints on their occurrence. Note that this representation does not yet allow for expressing logical constraints on the order in which events need to occur. We first define a mathematical model that allows us to logically reason about the occurrence of events in sets of execution sequences forming counterexamples in stochastic model checking. Technical Systems evolve in discrete computation steps. A system state  $s$  is defining a valuation of the system state variables. In our setting, we limit ourselves to considering systems that only contain Boolean state variables representing the occurrence of events, as described above. We use a set of atomic propositions that represent the Boolean state variables we consider. A computation step is characterized by an instantaneous transition which takes the system from some state  $s$  to a successor state  $s'$ . The transition from  $s$  to  $s'$  will be triggered by an action  $a$ , corresponding to the occurrence of an event. Since we wish to derive causality information from sets of finite computations, which we obtain by observing a finite number of computation steps, our main interest will be in sets of state-action sequences. We define the following model as a basis for our later formalization of the logical connection between events.



**Definition 1.** *State-Action Trace Model.* Let  $S$  denote a set of states,  $AP$  a finite set of atomic state propositions, and  $Act$  a finite set of action names.

- A finite sequence  $s_0, a_0, s_1, a_1, \dots, a_{n-1}, s_n$  with, for all  $i$ ,  $s_i \in S$  and  $a_i \in Act$ , is called a state-action trace over  $(S, Act)$ .
- A State-Action Trace Model (SATM) is a tuple  $M = (S, Act, AP, L, \Sigma)$  where  $\Sigma = \{\sigma_1, \dots, \sigma_k\}$  such that each  $\sigma_i$  is a state-action trace over  $(S, Act)$ , and  $L : S \rightarrow 2^{AP}$  is a function assigning each state the set of atomic propositions that are true in that state.

We assume that for a given SATM  $M$ ,  $AP$  contains the variables representing the events that we wish to reason about. We also assume that for a given state-action trace  $\sigma$ ,  $L(s_i)$  contains the event variable corresponding to the action  $a_{i-1}$ . Notice that we consider event instances, not types. In other words, the  $n$ -th occurrence of some event of type  $E$  will be distinct in  $AP$  from the  $n+1$ st occurrence of this event type. We next define an event order logic allowing us to reason about boolean conditions on the occurrence of events. The logic is using a set  $\mathcal{A}$  of event variables as well as the boolean connectives  $\wedge$ ,  $\vee$  and  $\neg$ . To express the ordering of events we introduce the ordered conjunction operator  $\wedge$ . The formula  $A \wedge B$  is satisfied if and only if events  $A$  and  $B$  occur in a trace and  $A$  occurs before  $B$ . The formal semantics of this logic is defined on SATMs. Notice that the  $\wedge$  operator is a temporal logic operator and that the SATM model is akin to a linearly ordered Kripke structure.

**Definition 2.** *Semantics of event order logic.* Let  $M = (S, Act, AP, L, \Sigma)$  a SATM,  $\phi$  and  $\psi$  formulae of the event order logic, and let  $\mathcal{A}$  a set of event variables, with  $A \in \mathcal{A}$ , over which  $\phi$  and  $\psi$  are built. Let  $\sigma = s_0, a_0, s_1, a_1, \dots, a_{n-1}, s_n$  a state-action trace over  $(S, Act)$ . We define that a formula is satisfied in state  $s_i$  of  $\sigma$  as follows:

- $s_i \models A$  iff  $A \in L(s_i)$ .
- $s_i \models \neg\phi$  iff not  $s_i \models \phi$ .
- $s_i \models \phi \wedge \psi$  iff  $\exists j, k : i \leq j, k \leq n . s_j \models \phi$  and  $s_k \models \psi$ .
- $s_i \models \phi \vee \psi$  iff  $\exists j, k : i \leq j, k \leq n . s_j \models \phi$  or  $s_k \models \psi$ .
- $s_i \models \phi \wedge \psi$  iff  $\exists j, k : i \leq j \leq k \leq n . s_j \models \phi$  and  $s_k \models \psi$ .

We define that a sequence  $\sigma$  satisfies a formula  $\phi$ , written as  $\sigma \models \phi$  iff  $\exists i . s_i \models \phi$ . We define that the SATM  $M$  satisfies the formula  $\phi$ , written as  $M \models \phi$ , iff  $\exists \sigma \in \Sigma . \sigma \models \phi$ .

In order to perform comparison operations between paths we define a number of path comparison operators.

**Definition 3.** *Path Comparison Operators.* Let  $M = (S, Act, AP, L, \Sigma)$  a SATM, and  $\sigma_1$  and  $\sigma_2$  state-action traces in  $M$ .

- $=$ :  $\sigma_1 = \sigma_2$  iff  $\forall e \in Act . \sigma_1 \models e \equiv \sigma_2 \models e$ .
- $\doteq$ :  $\sigma_1 \doteq \sigma_2$  iff  $\forall e_1, e_2 \in Act . \sigma_1 \models e_1 \wedge e_2 \equiv \sigma_2 \models e_1 \wedge e_2$ .

- $\subseteq$ :  $\sigma_1 \subseteq \sigma_2$  iff  $\forall e \in \text{Act} . \sigma_1 \models e \Rightarrow \sigma_2 \models e$ . Furthermore,  $\sigma_1 \subset \sigma_2$  iff  $\sigma_1 \subseteq \sigma_2$  and not  $\sigma_1 = \sigma_2$ .
- $\dot{\subseteq}$ :  $\sigma_1 \dot{\subseteq} \sigma_2$  iff  $\forall e_1, e_2 \in \text{Act} . \sigma_1 \models e_1 \wedge e_2 \Rightarrow \sigma_2 \models e_1 \wedge e_2$ . Furthermore,  $\sigma_1 \dot{\subset} \sigma_2$  iff  $\sigma_1 \dot{\subseteq} \sigma_2$  and not  $\sigma_1 \dot{=} \sigma_2$ .

We are now ready to adopt the SEM to event orders. We interpret the SEM equations over a given SATM  $M$ . Again, without providing a detailed formalization, we assume the existence of a function  $order_1$  assigning an order to the occurrence of the events  $M$  in the actual world, as well as a function  $order_2$  which assigns a potentially different order in the alternate world. An event order logic formula  $\psi$  is considered a cause for an event represented by the event order logic formula  $\varphi$ , if the following conditions are satisfied:

AC1: Both  $\psi$  and  $\varphi$  are true in the actual world, assuming the context defined by the variables in  $U$ , given a valuation  $val_1(V)$  and an order  $order_1(V)$ .

AC2: The set of endogenous events  $V$  is partitioned into sets  $Z$  and  $W$ , where the events in  $Z$  are involved in the causal process and the events in  $W$  are not involved in the causal process. It is assumed that  $X \subseteq Z$  and that there exist valuations  $val_2(X)$  and  $val_2(W)$  and orders  $order_2(X)$  and  $order_2(W)$  such that:

1. Changing the values of the variables in  $X$  and  $W$  from  $val_1$  to  $val_2$  and the order of the variables in  $X$  and  $W$  from  $order_1$  to  $order_2$  changes  $\varphi$  from true to false.
2. Setting the values of the variables in  $W$  from  $val_1$  to  $val_2$  and the order of the variables in  $W$  from  $order_1$  to  $order_2$  should have no effect on  $\varphi$  as long as the values of the variables in  $X$  are kept at the values defined by  $val_1$ , and the order as defined by  $order_1$ , even if all the variables in an arbitrary subset of  $Z \setminus X$  are set to their value according to  $val_1$  and  $order_1$ .

AC3: The set of variables  $X$  is minimal: no subset of  $X$  satisfies conditions AC1 and AC2.

If a formula  $\psi$  meets the above described conditions, the occurrence of the events included in  $\psi$  is causal for  $\varphi$ . However, condition AC2 does not imply that the order of the occurring events is causal. We introduce the following condition to express that the order of the variables occurring in  $\psi$ , or an arbitrary subset of these variables, has an influence on the causality of  $\varphi$ :

OC1: Let  $Y \subseteq X$ . Changing the order  $order_1(Y)$  of the variables in  $Y$  to an arbitrary order  $order_2(Y)$ , while keeping the variables in  $X \setminus Y$  at  $order_1$ , changes  $\varphi$  from true to false.

If for a subset of  $X$  OC1 is not satisfied, the order of the events in this subset has no influence on the causality of  $\varphi$ .

**Fault Tree Generation.** In order to automatically synthesize a fault tree from a stochastic counterexample, the combinations of basic events causing the top level event in the fault tree have to be identified. Using a stochastic model checker we compute a counterexample which contains all paths leading to a state corresponding to the occurrence of some top level event  $T$ . This is achieved by computing the counterexample for the CSL formula  $P =?(true \ U \ t)$ , where  $t$  is a

state formula representing the top level event  $T$ . We interpret counterexamples in the context of an SATM  $M = (S, Act, AP, L, \Sigma)$ . We assume that  $\Sigma$  is partitioned in disjoint sets  $\Sigma_G$  and  $\Sigma_C$ , where  $\Sigma_C$  contains all traces belonging to the counterexample, whereas  $\Sigma_G$  contains all maximal simple system traces that do not belong to the counterexample. The disjointness of  $\Sigma_C$  and  $\Sigma_G$  implies that  $M$  is deterministic with respect to the causality of  $T$ . Furthermore, we define  $M_C = (S, Act, AP, L, \Sigma_C)$  as the restriction of  $M$  to only the counterexample traces, and refer to it as a counterexample model. W.l.o.g. we assume that every trace  $\sigma \in M_C$  contains a last transition executing the top level event  $T$ , so that its last state  $s_n \models T$ , which implies that  $M_c \models T$ . In our interpretation of the SEM, actual world models will be derived from  $\Sigma_C$ , whereas alternate world models are part of  $\Sigma_G$ . Notice that in order to compute the full model probability of reaching  $T$  it is necessary to perform a complete state space exploration of the model that we analyze. We hence obtain  $M_C$  at no additional cost. We next define the candidate set of paths that we consider to be causal for  $T$ . We define this set in such a way that it includes all minimal paths. Paths are minimal if they do not contain a subpath according to the  $\sqsubseteq$  operator that is also a member of the candidate set.

**Definition 4 (Candidate Set).** *Let  $M_C = (S, Act, AP, L, \Sigma_C)$  a counterexample model, and  $T$  a top level event in  $M_C$ . We define the candidate set of paths belonging to the fault tree of  $T$  as  $CFT(T)$ :*

$$CFT(T) = \{\sigma \in \Sigma_C \mid \forall \sigma' \in \Sigma_C . \sigma' \sqsubseteq \sigma \Rightarrow \sigma' = \sigma\}. \quad (1)$$

Notice that the candidate set is minimal in the sense that removing an event from some path in the candidate set means that the resulting path is no longer in the counterexample  $\Sigma_C$ . Given a counterexample model  $M_C$ , we state the following observations regarding the paths included in  $\Sigma_C$ :

- Each  $\sigma \in \Sigma_C$  can be viewed as an ordered conjunction  $A_1 \wedge \dots \wedge A_{n-1} \wedge T$  of events, where  $T$  is the top level event that we consider.
- On each path in the counterexample, there has to be at least one event causing the top level event. If that was not the case, the top level event would not have occurred on that path and as a result the path would not be in the counterexample.

The algorithm that we propose to compute fault trees is an over-approximation of the computation of the causal events  $X$  since computing the set  $X$  precisely is not efficiently possible [11]. Instead of computing the set  $X$  of events that are causal for some  $\varphi$ , we compute the set  $Z$  of events, which consists of all events that are part of the causal process of  $\varphi$ .  $Z$  will then be represented by  $\psi$ . Since  $X$  is a subset of  $Z$  we can assure that no event that is causal is omitted from the fault tree. It is, however, possible that due to our overapproximation events that are not in  $X$  are added to the fault tree. This applies in particular to those events that are part of the causal process, and hence mediate between  $X$  and  $\varphi$ . However, as we show in Section 4, adding such events can be helpful to illustrate

how the root cause is indirectly propagating by non-causal events to finally cause the top level event. We do not account for exogenous variables, since we believe them to be less relevant in the analysis of models of computational systems since the facts represented in those models all seem to be endogenous facts of the system. However, should one wish to consider exogenous variables, those can easily be retrofitted. We now define tests that will identify the set  $Z$  of ordered conjunctions of events that satisfy the conditions AC1 to AC3 and OC1, and which hence can be viewed as part of the causal process of the top level event. The starting point for this computation is the candidate set of Definition 4.

*Test for AC1:* The actual causal set  $Z$  that our algorithm computes is a subset of the events included in the candidate set  $CFT(T)$  for some given top level event  $T$ . Since we assume that every path includes at least one causal event,  $Z$  is not empty. We may hence conclude that  $CFT(T) \models \psi$  and  $CFT(T) \models \varphi$ .

*Test for AC2(1):* We check for each path  $\sigma \in CFT(T)$  whether the ordered conjunctions of events that it is representing fulfills the condition AC2(1). We assume that the set of events  $Z$  is equal to the events occurring on the path  $\sigma$ . We further assume that  $W = V \setminus Z$  and that  $V = Act$ .  $W$  hence contains all events that are possible, minus the ones that occur on path  $\sigma$ . More formally, for a given  $\sigma$ ,  $Z = \{e \in V \mid \sigma \models e\}$ . This corresponds to using the valuation  $val_1$  to assign true to all variables in  $Z$  and false to all variables in  $W$  in the formulation of AC2(1). Changing the valuation of the variables in  $Z$  to move from  $val_1$  to some  $val_2$  can imply removing variables from  $Z$ . Due to the minimality of  $\sigma$  this implies that the resulting trace  $\sigma'$  is no longer in  $\Sigma_C$ . Testing of this condition is hence implicit and implied by the minimality of the candidate set.

*Test for AC2(2):* We need to check that moving  $W$  from  $val_1$  to  $val_2$  and from  $order_1$  to  $order_2$  has no effect on the outcome of  $\varphi$  as long as the values of  $X$  are kept at the values defined by  $val_1$  and the order defined by  $order_1$ . Recall that  $W$  denotes all events that are not currently considered to be causal, and that we compute  $Z$  as an overapproximation of  $X$ . For a path  $\sigma \in CFT(T)$  changing  $W$  from  $val_1$  to  $val_2$  and from  $order_1$  to  $order_2$  implies that events are added to  $\sigma$ . Thus, we check for each path  $\sigma \in CFT(T)$  whether there exists some path  $\sigma' \in \Sigma_C$  for which  $\sigma \dot{c} \sigma'$  holds. If there is no such path, there are no  $val_2$  and  $order_2$  of  $W$  that change the outcome of  $\varphi$ , and as a consequence AC2(2) is fulfilled by  $\sigma$ . If we do find such a path  $\sigma'$ , it contains all variables in  $Z$  with  $val_1$  and  $order_1$  and some events  $W$  with  $val_2$  and  $order_2$  that change the outcome of  $\varphi$ . In other words, the non-occurrence of the events in  $W$  on  $\sigma$  was causal for  $\varphi$ . In order to identify those events, we search for the minimal paths  $R = \{\sigma' \in \Sigma_C \mid \sigma \dot{c} \sigma'\}$ . For each path in  $R$  we negate the events that are in  $\sigma$  but not in  $\sigma'$  and add them to the candidate set. Subsequently, we remove  $\sigma$  from the candidate set. Consider the case  $Z = G \wedge O \wedge T \wedge C$  in our rail road crossing model. It is necessary that no event  $G$  occurs between  $O$  and  $T$  for this ordered conjunction of events to lead to a hazard. If the system execution  $G \wedge O \wedge G \wedge T \wedge C$  is possible, which means that there is a path representing this execution in the set  $NCX(A)$  for top level event  $A$ , we hence have to replace  $Z$  by  $Z' = G \wedge O \wedge \neg G \wedge T \wedge C$ .

*Test for AC3:* Due to the minimality property of the candidate set, no test for AC3 is necessary.

*Test for OC1:* We need to decide whether for all ordered conjunctions in  $CFT(T)$  the order of the events is relevant to cause  $T$ . For each path  $\sigma \in CFT(T)$ , we check whether the order of the events to occur is important or not. If the order of events in  $\sigma$  is not important, then there has to be at least one path  $\sigma' \in CFT(T)$  for which  $\sigma = \sigma'$  and not  $\sigma \dot{=} \sigma'$ . For each event  $e_i$  in  $\sigma$  we check for all other events  $e_j$  with  $i < j$  whether  $\sigma' \models e_i \wedge e_j$  for all  $\sigma' \in CFT(T)$ . If  $\sigma' \models e_i \wedge e_j$  for all  $\sigma' \in CFT(T)$ , we mark this pair of events as having an order which is important for causality. If we can not find such a  $\sigma'$ , we mark the whole path  $\sigma$  as having an order which is important for causality.

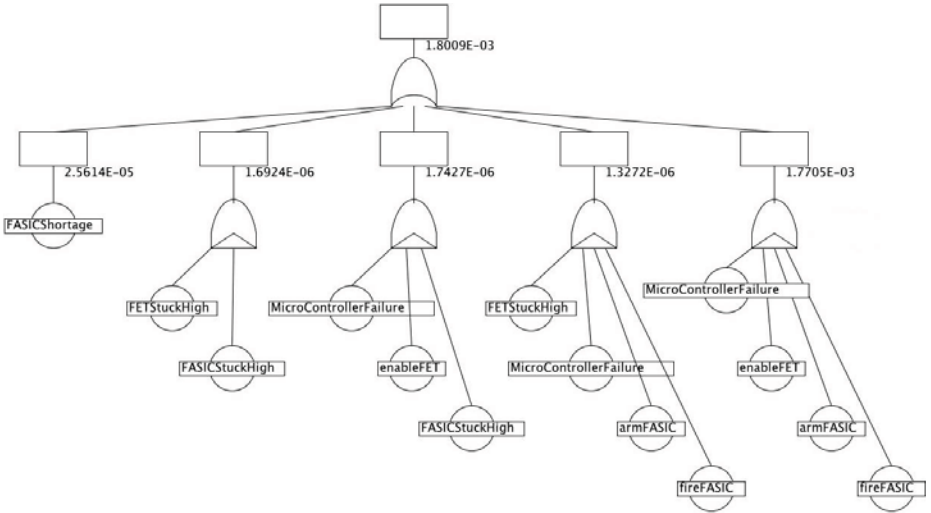
**Adding Probabilities.** In order to properly compute the probability mass that is to be attributed to the TLE  $T$  in the fault tree it is necessary to account for all paths that can cause  $T$ . If there are two paths  $\sigma_1, \sigma_2 \in \Sigma_C$  which, when combined, deliver a path  $\sigma_{12} \in \Sigma_C$ , then the probability mass of all three paths needs to be taken into account when calculating the probability for reaching  $T$ . To illustrate this point, consider an extension of the railroad example introduced above. We add a traffic light indicating to the car driver that a train is approaching. Event  $R$  indicates that the traffic light on the crossing is red, while the red light being off is denoted by event  $L$ . The top level event  $A$  denoting the hazard is expressed as a state proposition applicable to the underlying stochastic model that states that neither the red light is on nor the gate is closed, and that the train approaches and the car is in the crossing. Assume that the above described test would identify the following ordered conjunctions of events to be causal:  $\neg G \wedge T \wedge C$  and  $\neg L \wedge T \wedge C$ . Due to the minimality property of  $CFT(A)$  the ordered conjunction  $\neg G \wedge \neg L \wedge T \wedge C$  would be missing. We would hence lose the probability mass associated with the corresponding trace in the counterexample, as well as the qualitative information that the simultaneous failure of the red light and the gate also leads to a hazardous state. To account for this situation we introduce the combination condition CC1. CC1: Let  $\sigma_1, \sigma_2, \dots, \sigma_k \in CFT(L)$  paths and  $\psi_1, \psi_2, \dots, \psi_k$  the event conjunctions representing them. A path  $\sigma$  is added to  $CFT(L)$  if for  $k \geq 2$  paths in  $CFT(L)$  it holds that  $\sigma \models \psi_1 \wedge \sigma \models \psi_2 \wedge \dots \wedge \sigma \models \psi_k$ . We can now assign each path in the candidate set the sum of the probability masses of the paths that it represents. This is done as follows: The probability of a path  $\sigma_1$  in  $CFT(L)$  is the probability sum of all paths  $\sigma'$  for which  $\sigma_1$  is the only subset in  $CFT(L)$ . The last condition is necessary in order to correctly assign the probabilities to paths which were added to the fault tree by test CC1. All paths still in the candidate set are part of the fault tree and have now to be included in the fault tree representation. The fault trees generated by our approach all have a normal form, that is they start with an *intermediate*-gate representing the top level event, that is connected to an *OR*-gate. The paths in the candidate set  $CFT(L)$  will then be added as child nodes to the *OR*-gate as follows: Paths with a length of one and hence consisting of only one basic event are represented by the respective basic event. A path with length greater than one that has

no subset of labels marked as ordered is represented by an *AND*-gate. This *AND*-gate connects the basic events belonging to that path. If a (subset of a) path is marked as ordered it is represented by a *PAND*-gate that connects the basic events in addition with an *Order Condition* connected to the *PAND*-gate constraining the order of the elements. The probability values of the *AND*-gates are the corresponding probabilities of the paths that they represent. In order to display the probabilities in the graphical representation of the fault tree, we add an intermediate event as parent node for each *AND*-gate. The resulting intermediate events are then connected by an *OR*-gate that leads to the top event, representing the hazard. Since the path probabilities are calculated for a path starting from an initial state to the hazard state, the probability of the *OR*-gate is the sum of the probability of all child elements.

**Scalability and Complexity.** As we show in detail in [15], the complexity of our algorithm is cubic in the size of the counterexample. The case studies presented in Section 4 show that the fault tree computation finishes in several seconds, while the computation of the counterexample took several minutes. Hence, the limiting factor of our approach is the time needed for the computation of the counterexample.

## 4 Case Study

We now briefly discuss a case study illustrating the applicability of our approach. Notice that for reasons of limited space we have to refer the reader to [15] for more detail, and also for more case studies. This case study is taken from [1] and models an industrial size airbag system. The airbag system architecture that we consider consists of two acceleration sensors whose task it is to detect front or rear crashes, one microcontroller to perform the crash evaluation, and an actuator that controls the deployment of the airbag. Although airbags save lives in crash situations, they may cause fatal behavior if they are inadvertently deployed. This is because the driver may lose control of the car when this deployment occurs. It is therefore a pivotal safety requirement that an airbag is never deployed if there is no crash situation. We are interested in generating the fault tree for an inadvertent ignition of the airbag. In CSL, this property can be expressed using the formula  $\mathcal{P}_{=?}(noCrash \ U^{\leq T} \ AirbagIgnited)$ . Notice that we assume the PRISM models in practical usage scenarios to be automatically synthesized from higher-level design models, such as for instance by our QuantUM tool [16]. However, the case study presented in this paper was directly modeled in the PRISM language. We computed the counterexamples using our counterexample generation tool DiPro [2], which in turn uses the PRISM model checker. Figure 1 shows the fault tree generated by CX2FTA. For better readability we have omitted the order constraints of the *PAND*-gates. While the counterexample consists of 738 paths, the fault tree comprises only 5 paths. It is easy to see by which basic events, and with which probabilities, an inadvertent deployment of the airbag is caused. For instance, there is only one single fault that can lead to an inadvertent deployment, namely *FASICSshortage*. It is also easy to



**Fig. 1.** Fault Tree of the Airbag System

$t$	Runtime CX (sec.)	Paths in CX	Mem. CX	Runtime FT	Paths in FT	Mem. FT
10	1 147 ( $\approx 19.12$ min.)	738	29.17 MB	1.3 (sec.)	5	27 MB
100	1 148 ( $\approx 19.13$ min.)	738	29.20 MB	1.3 (sec.)	5	27 MB
1000	1 263 ( $\approx 21.05$ min.)	738	29.49 MB	1.8 (sec.)	5	27 MB

**Fig. 2.** Experiment results for  $T=10$ ,  $T=100$  and  $T=1000$

see that the combination of the basic events *FETStuckHigh* and *FASICStuckHigh* only lead to an inadvertent deployment of the airbag if the basic event *FETStuckHigh* occurs prior to the basic event *FASICStuckHigh*. The case study shows that the fault tree is a compact and concise visualization of the counterexample which allows for an easy identification of the basic events that cause the inadvertent deployment of the airbag, along with the corresponding probabilities. If the order of the events is important, this can be seen in the fault tree by the *PAND*-gate. In the counterexample computed by DiPro one would have to manually compare the order of the events in all 738 paths, which is a tedious and time consuming task. Figure 2 shows the memory and run time consumption of the fault counterexample and fault tree computation. The computational effort is dominated by the counterexample computation. Increasing the parameter  $t$  (mission time) in the process model has only a marginal influence on the computational effort needed.

## 5 Related Work

Work described in [7,20] interprets fault trees in terms of temporal logic. This is the opposite direction of what we aim to accomplish, namely to derive fault trees

<sup>2</sup> Experiments were performed on a PC with an Intel Core2Duo CPU (3.06 Ghz) and 8 GBs RAM.

from system execution models. Various approaches to derive fault trees semi-automatically or automatically from various semi-formal or formal models have been proposed, e.g. [19,8,18]. Contrary to our method, none of these methods uses sets of system execution sequences as the basis of the fault tree derivation, or provides an automatic probabilistic assessment of the synthesized fault tree nodes. These approaches also lack a justification of the causality model used. Our work extends and improves on the approach of [6] in the following ways: We use a single set of system modeling and specification languages, namely PRISM and CSL. Whereas in the approach of [6] only minimal cut sets are generated, we generate complete fault trees. Contrary to [6], we support PAND-gates and provide a justification for the causality model used. Work documented in [5] uses the Halpern and Pearl approach to determine causality for counterexamples in functional CTL model checking. However, this approach considers only functional counterexamples that consist of single execution sequences. [11] contains a careful analysis of the complexity of computing causality in the SEM. Most notable is the result that even for an SEM with only binary variables computing causal relationships between variables is NP-complete.

## 6 Conclusion

We presented a method and tool that automatically generates a fault tree from a probabilistic counterexample. We demonstrated that our approach improves and facilitates the analysis of safety critical systems. The resulting fault trees were significantly smaller and hence easier to understand than the corresponding stochastic counterexample, but still contain all information to discern the causes for the occurrence of a hazard. The justification for the causalities determined by our method are based on an adoption of the Structural Equation Model of Halpern and Pearl. We illustrated how to use this model in the analysis of computing systems and extended it to account for event orderings as causal factors. We presented an over-approximating implementation of the causality tests derived from this model. To the best of our knowledge this is the first attempt at using the structural equation model in this fashion. In future work, we plan to further extend our approach, in particular to support the generation of dynamic fault-trees [10]. We are also interested in incorporating causality analysis directly into model checking algorithms.

**Acknowledgments.** The authors thank Mark Burgess for giving them access to the FaultCAT source code.

## References

1. Aljazzar, H., Fischer, M., Grunske, L., Kuntz, M., Leitner-Fischer, F., Leue, S.: Safety Analysis of an Airbag System Using Probabilistic FMEA and Probabilistic Counterexamples. In: Proc. of QEST 2009. IEEE Computer Society, Los Alamitos (2009)



2. Aljazzar, H., Leue, S.: Debugging of Dependability Models Using Interactive Visualization of Counterexamples. In: Proc. of QEST 2008. IEEE Computer Society, Los Alamitos (2008)
3. Aljazzar, H., Leue, S.: Directed explicit state-space search in the generation of counterexamples for stochastic model checking. *IEEE Trans. Soft. Eng.* (2009)
4. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P.: Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. Soft. Eng.* (2003)
5. Beer, I., Ben-David, S., Chockler, H., Orni, A., Trefler, R.: Explaining counterexamples using causality. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 94–108. Springer, Heidelberg (2009)
6. Böde, E., Peikenkamp, T., Rakow, J., Wischmeyer, S.: Model Based Importance Analysis for Minimal Cut Sets. In: Cha, S(S.), Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 303–317. Springer, Heidelberg (2008)
7. Bozzano, M., Cimatti, A., Tapparo, F.: Symbolic Fault Tree Analysis for Reactive Systems. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 162–176. Springer, Heidelberg (2007)
8. Chen, B., Avrunin, G., Clarke, L., Osterweil, L.: Automatic Fault Tree Derivation From Little-Jil Process Definitions. In: Wang, Q., Pfahl, D., Raffo, D.M., Wernick, P. (eds.) SPW 2006 and ProSim 2006. LNCS, vol. 3966, pp. 150–158. Springer, Heidelberg (2006)
9. Collins, J. (ed.): Causation and Counterfactuals. MIT Press, Cambridge (2004)
10. Dugan, J., Bavuso, S., Boyd, M.: Dynamic Fault Tree Models for Fault Tolerant Computer Systems. *IEEE Trans. Reliability* (1992)
11. Eiter, T., Lukasiewicz, T.: Complexity results for structure-based causality. *Artificial Intelligence* (2002)
12. Halpern, J., Pearl, J.: Causes and explanations: A structural-model approach. Part I: Causes. *The British Journal for the Philosophy of Science* (2005)
13. Han, T., Katoen, J.-P., Damman, B.: Counterexample generation in probabilistic model checking. *IEEE Trans. Softw. Eng.* (2009)
14. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A Tool for Automatic Verification of Probabilistic Systems. In: Hermanns, H. (ed.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
15. Kuntz, M., Leitner-Fischer, F., Leue, S.: From probabilistic counterexamples via causality to fault trees. Technical Report soft-11-02, Chair for Software Engineering, University of Konstanz (2011), <http://www.inf.uni-konstanz.de/soft/research/publications/pdf/soft-11-02.pdf>
16. Leitner-Fischer, F., Leue, S.: QuantUM: Quantitative safety analysis of UML models. In: Proc. of QAPL 2011 (2011)
17. Lewis, D.: Counterfactuals. Wiley-Blackwell, Chichester (2001)
18. McKelvin Jr, M., Eirea, G., Pinello, C., Kanajan, S., Sangiovanni-Vincentelli, A.: A Formal Approach to Fault Tree Synthesis for the Analysis of Distributed Fault Tolerant Systems. In: Proc. of EMSOFT 2005. ACM, New York (2005)
19. Pai, G., Dugan, J.: Automatic synthesis of dynamic fault trees from UML system models. In: Proc. of ISSRE 2002. IEEE Computer Society, Los Alamitos (2002)
20. Schellhorn, G., Thums, A., Reif, W.: Formal fault tree semantics. In: Proc. IDPT 2002. Society for Design and Process Science (2002)
21. U.S. Nuclear Regulatory Commission. Fault Tree Handbook, NUREG-0492 (1981)
22. Zeller, A.: Why Programs Fail: A Guide to Systematic Debugging. Elsevier, Amsterdam (2009)

# Rigorous Evidence of Freedom from Concurrency Faults in Industrial Control Software

Richard Bonichon<sup>1</sup>, Géraud Canet<sup>1</sup>, Loïc Correnson<sup>1</sup>, Eric Goubault<sup>1</sup>,  
Emmanuel Haucourt<sup>1</sup>, Michel Hirschowitz<sup>1</sup>,  
Sébastien Labbé<sup>2</sup>, and Samuel Mimram<sup>1</sup>

<sup>1</sup> CEA, LIST,  
Gif-sur-Yvette, F-91191, France  
`firstname.lastname@cea.fr`  
<sup>2</sup> EDF Research & Development  
6 quai Watier, Chatou, F-78401, France  
`firstname.lastname@edf.fr`

**Abstract.** In the power generation industry, digital control systems may play an important role in plant safety. Thus, these systems are the object of rigorous analyzes and safety assessments. In particular, the quality, correctness and dependability of control systems software need to be justified. This paper reports on the development of a tool-based methodology to address the demonstration of freedom from intrinsic software faults related to concurrency and synchronization, and its practical application to an industrial control software case study. We describe the underlying theoretical foundations, the main mechanisms involved in the tools and the main results and lessons learned from this work. An important conclusion of the paper is that the used verification techniques and tools scale efficiently and accurately to industrial control system software, which is a major requirement for real-life safety assessments.

**Keywords:** Digital control systems, software dependability, formal verification, concurrency, deadlock.

## 1 Introduction — Intrinsic Software Faults

Dependability assessment of digital control systems require elements from control systems designers in order to establish the excellence of production; e.g. evidence of systematic, fully documented and reviewable engineering process, quality assurance, test and simulation at different stages of development, operational experience and demonstration of conformity to applicable standards.

Complementary measures may be taken in order to demonstrate properties, and provide rigorous evidence of the freedom from postulated categories of faults. Considered faults include *intrinsic software faults*, i.e. faults in a software design that can be identified independently of functional specifications.

Technical studies and surveys performed in recent years have led to consider three main categories of software intrinsic faults as relevant to this domain, i.e. intrinsic faults that might be postulated in software important to availability. The taxonomy we rely on [24] should be comprehensive, though not exhaustive. It has been established by using various sources:

- Experience gained in software formal verification at EDF (e.g. [28] and internal technical reports) and other institutions or companies sharing similar interest in such methods [1, 15, 25];
- Community lists of software faults, like CWE [12];
- Lists of addressed faults published by software analysis tool vendors.

The mentioned categories of faults are the following:

- Faults in concurrency and synchronization (detailed below);
- Faults in dynamic handling of memory, namely: memory leaks, segmentation faults and other memory-related undesirable behaviors;
- Other, more “basic”, faults such as divisions by zero, out-of-bounds array access, use of uninitialized variables, other numeric manipulation errors, etc.

Depending on the importance to safety of a system software, some of the above categories of faults can be ruled out by design. For instance in critical software, dynamic management of the memory is usually not allowed by design rules. In this paper, only the concurrency aspects are developed; specifically when considering these aspects, it is of interest to demonstrate the absence of the following intrinsic faults:

- Resource starvation such as deadlocks and livelocks, e.g. points where no further progress is possible for a given program run;
- Non-determinism or race condition, e.g. points where a program may behave differently given the same inputs (because side effects depend upon synchronization);
- Incorrect protection of shared resources, e.g. concurrent and non protected accesses to a shared variable;
- Incorrect handling of priorities;
- Unexpectedly unreachable program states, e.g. no path may lead to a given state while it was supposed to be reachable.

In some cases, intrinsic faults are unlikely to be detected via classical V&V methods<sup>1</sup>, for instance faults that might be triggered in specific configurations of variables (e.g. division by zero, arithmetic overflow), or after long runs (e.g. overrun in a very large buffer, out-of-memory error due to a small leak), etc.

Tool-based analytic approaches to systematically identify intrinsic faults preferably rely on formal verification techniques. The gained confidence can then be used in higher level assessments, e.g. to support claims about I&C software contribution to the reliability of a safety function, or to alleviate concerns regarding digital Common Cause Failures.

---

<sup>1</sup> Functional correctness still needs to be addressed with appropriate approaches, e.g. functional testing, theorem proving or simulation.

## 2 Tool-Based Methodology — Outline

This paper reports on the development of a tool-based methodology to demonstrate the freedom from intrinsic software faults related to concurrency in industrial control software. The following two related projects are involved (both projects have eponymous tools):

- **MIEL**: Interactive model extraction from software — Analyze software written in C language in order to facilitate code understanding and extract representative models to be analyzed by third-party tools;
- **ALCOOL**: Analysis of coordination in concurrent software — Develop a theory, algorithms and a static analysis tool with the ability to verify synchronization properties (particularly, freedom from intrinsic faults) in complex software.

The tool-based methodology developed in these two projects aims at analyzing software that can be found in digital control systems either safety related, or important to availability, in power plants. Compared to the most critical software parts, complex programming mechanisms might be more freely used in such software, e.g. concurrent interactions and synchronization.

On the other hand, the dependability requirements for such systems are high; then the system lifecycle must (at least partly) establish required properties, e.g. predictability. In particular, design measures are generally applied to effectively reduce the complexity and restrict the set of potential vulnerabilities. Memory allocation, task and synchronization resources creation are indeed to be performed *only* during a dedicated initialization phase. Then, the behavior in normal operation is intended mostly cyclic and steady. For instance, an iteration of a loop is expected to “know” as little as possible from previous iterations. Also, communication and synchronization are expected to be restricted to the necessary. The presented tool-based methodology is intended to take advantage of those characteristics, to provide rigorous verification tools with high efficiency.

The rest of the paper is structured as follows. The tools MIEL and ALCOOL, together with their theoretical foundations are described in Sec. 3. An industrial software case study and practical experiments are described in Sec. 4. Some related works are presented in Sec. 5. Finally, the main results, lessons learned and perspectives to this work are synthesized in Sec. 6.

## 3 Theoretical Framework and Tools

### 3.1 Static Analysis for Model Extraction and the MIEL Tool

The MIEL tool is a model extractor for C programs. It runs as a plug-in of the **Frama-C** static analysis platform [11, 17], which is dedicated to the analysis of software source code written in the C programming language.

The main requirement for models extracted by MIEL is to be *conservatively* representative with respect to the specified point of interest, i.e. behaviors related to a specific aspect of interest in the original program must be included in the behaviors denoted by the corresponding model. Various intrinsic aspects of

<pre>function pthread_create of type thread creation has arg 3 of type function name</pre>	<pre>function pthread_mutex_unlock of type release has arg 1 of type semaphore  function pthread_mutex_lock of type lock has arg 1 of type semaphore</pre>
--	--

**Fig. 1.** Excerpts of MIEL description file: POSIX thread creation (left), locks (right)

software can be of interest, including calls to memory management, concurrency, synchronization or communication primitives.

Accordingly, extracted sets of information may then be provided as models to dedicated external tools, with the purpose of demonstrating or refuting properties. The MIEL tool is primarily geared toward extracting models from concurrent programs; its algorithm applies also to sequential programs.

Model extraction is based on a description file, which indicates the points of interests in the program: a list of functions, together with a signature (using types known by MIEL)<sup>2</sup>. Only arguments of interest should be specified in a signature. For example, a user interested in threads or processes created in a program using POSIX primitives, might write the description of Fig. 1 (left). This declares to the analyzer, first, that every occurrence of thread creation must appear in the model, and second, that the starting functions of threads are given in the third argument of `pthread_create`. Now, considering synchronization faults, e.g. deadlocks, the model must also encompass all events where locks are taken and released. Fig. 1 (right) suggests a suitable description.

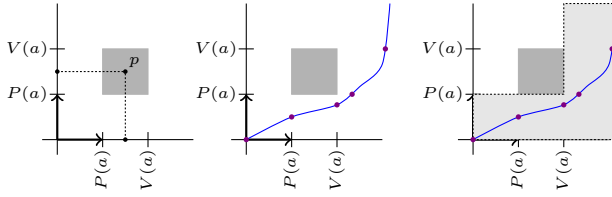
The MIEL tool implements a syntactic model extraction algorithm, which is sound for a specific class of software where the interesting function identifiers are syntactically reachable. That is to say, the functions of interest must not be aliased, and resource requests of interest must be syntactically distinct. We will see in the following how these soundness requirements are consistent with the characteristics of the targeted systems software.

Considering the typical design features of the targeted classes of software (safety related or important to availability, cf. Sec. 2), it is expected that synchronization or memory management primitives can be easily identified (no aliases on these function names). It is also expected that loop unrolling is sufficient to make sure accesses to different synchronization or memory resources (threads, mutexes, semaphores, memory cells, etc.) can be syntactically distinguished. This can be achieved through code annotation within **Frama-C**. Detailed examples of the annotations needed for the presented case study are given in Sec. 4.2.

### 3.2 Geometric Semantics for Concurrency Analysis, ALCOOL Tool

The ALCOOL tool is based on the directed algebraic topology of cubical areas. Roughly speaking, a cubical area of dimension  $n \in \mathbb{N}$  is a finite union of hyper-rectangles of dimension  $n \in \mathbb{N}$  (i.e. finite products of  $n$  non empty intervals of the real line  $\mathbb{R}$ ). As mathematical objects, the cubical areas enjoy a structure which

<sup>2</sup> The types used to classify functions include: thread creation, process creation, lock, release, delay, priority, interrupt handler, ...



**Fig. 2.** The forbidden square

is implemented as a library used by the **ALCOOL** tool. Basically it takes as input a program written in a specific input language (in practical cases, models are preferably automatically generated by **MIEL**) and produces a geometric model which is a cubical area. From this model, **ALCOOL** then can identify forbidden states, deadlocks, unreachable states, critical sections. Using an algorithm which performs the “prime” decomposition of a cubical area [3], **ALCOOL** can also find the subgroups of processes which actually run independently from each others.

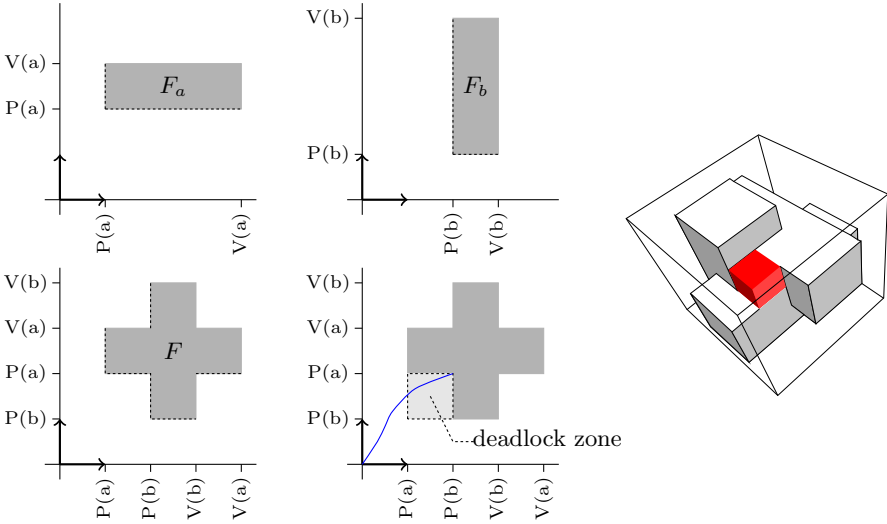
The *PV* language was introduced by E.W. Dijkstra to illustrate the problems which may arise when designing parallel programs [13]. In the original version, *P* and *V* respectively stand for the dutch words “pakken” (take) and “vrijlaten” (release). The processes indeed share a pool of resources, and each of them may request the authorization to access a resource *a* by executing the instruction *P(a)*. If *a* is available, then it is granted to the process until it releases it by executing the instruction *V(a)*. Otherwise, the process stops and waits until the resource is dropped by the previous holder.

On the practical side, the development of the **ALCOOL** tool was originally driven by the need for dealing with industrial C programs. The **ALCOOL** input language (called *PV* in the following), therefore contains a broader set of concurrency primitives, including features from **POSIX** and **VxWorks**.

We now illustrate the abilities of **ALCOOL** through some examples. Consider the sequential process  $\pi := P(a).V(a)$ , where *a* is a resource that can be held by a single process at a time (a mutex). Then run two copies of  $\pi$  simultaneously:  $\pi|\pi$  (cf. Fig. 2). Generally, each running process is associated with a dimension, therefore the geometric model is here 2-dimensional. The point *p* represents a state in which both instruction pointers, which are the projections of *p*, lie between *P(a)* and *V(a)*. That is, in this state both copies of the process  $\pi$  hold the resource *a*, which is by definition forbidden (not feasible). The geometric model is thus  $\mathbb{R}^2 \setminus [1, 2]^2$ : all the points in the gray square are forbidden.

The fact intervals are open or closed depends on whether an instruction is executed exactly when it is reached by the instruction pointer. In this framework, the program execution traces correspond to increasing continuous paths in the geometric model (cf. Fig. 2). The library dealing with cubical areas allows **ALCOOL** to identify these, all represented by the same cubical area.

For the next example in Fig. 3 (left), we consider two processes  $\pi_0 := P(a).P(b).V(b).V(a)$  and  $\pi_1 := P(b).P(a).V(a).V(b)$  running concurrently. We have two resources *a* and *b* each of which generates a forbidden rectangle. In this



**Fig. 3.** Deadlock examples: The Swiss flag (left), The 3 dining philosophers (right)

example we have a potential deadlock: if the first process takes the resource  $a$  while the second one takes  $b$ , then both won't be able to progress further.

The 3-dimensional models arise from the study of programs made of 3 processes, such as the well-known 3 dining philosophers, cf. Fig. 3 (right). In this picture, the central (red) cube represents the deadlock zone.

As we shall see, **ALCOOL** can also perform the factorization of a PV program from its geometric model. In the next example we introduce counting semaphores: resources that can be shared by more than 2 processes. We suppose  $a$  and  $b$  are mutexes and  $c$  is a 3-semaphore, meaning  $c$  can be held by 2 processes at a time but not 3. Then we consider the following processes:

$$\begin{aligned} \pi_a &:= P(a).P(c).V(c).V(a) \\ \pi_b &:= P(b).P(c).V(c).V(b) \end{aligned}$$

Then we consider the PV program  $\pi_a|\pi_b|\pi_a|\pi_b$ . A handmade analysis reveals that the semaphore  $c$  is in fact useless. The program can indeed be split into two groups of processes  $\{1, 3\}$  and  $\{2, 4\}$ . Each group cannot hold more than one occurrence of the  $c$  resource, so it cannot run out of stock. The **ALCOOL** tool detects this situation by performing an algebraic factorization, proving that the geometric model of the program can be written as a 2-fold Cartesian product:

$$\left( [0, 1[ \times [0, \infty[ \cup [4, \infty[ \times [0, \infty[ \cup [0, \infty[ \times [0, 1[ \cup [0, \infty[ \times [4, \infty[ \right)^2$$

From a theoretical point of view, a cubical area can be written as a Cartesian product in a unique way (compare with natural numbers and prime numbers). This feature is extremely important since it allows, when the decomposition of

the geometric model is not trivial, to split the analysis of a program into the analysis of several simpler subprograms.

The cubical area library lies upon some facts we now state. In the sequel, a cubical area should be understood as a finite union of hyperrectangles.

- The intersection of finitely many cubical areas of dimension  $n$  is a cubical area of dimension  $n$ ;
- The complement (in  $\mathbb{R}^n$ ) of a cubical area of dimension  $n$  is a cubical area of dimension  $n$ ;
- Any cubical area has a “normal form” given by the collection of all its maximal sub-hyperrectangles. A hyperrectangle contained in a cubical area  $X$  is said to be maximal (in  $X$ ) when any strictly bigger hyperrectangle is not contained in  $X$ ;
- The previous assertions over the  $n$  dimensional cubical areas are compatible with the action of the symmetric group  $\mathfrak{S}_n$  (in other words the permutation of coordinates in a “coherent” way);
- There is a notion of directed homotopy so that each equivalence class of directed path over a cubical area is characterized by a cubical area  $X$ . Indeed, a path  $\gamma$  is in the class  $E_X$  if and only if its image  $\{\gamma(t) \mid t \in [0, 1]\}$  is contained in  $X$ .

More details about the way theoretical facts are exploited by `ALCOOL` may be found in [19]. The cubical areas are special cases of pospaces, they are introduced in [26] without any mention to their directed homotopy aspects. Gentle introductions to the latter can be found in [18] and [16]. An abstract treatment of Directed Algebraic Topology can be found in [20].

## 4 Case Study

This section reports on a practical application of the verification approach presented in Sec. 3. Section 4.1 briefly describes a real-world software unit, which is embedded in a programmable logic controller. Then, we explain in Sec. 4.2 how the verification framework and tools support rigorous evidence of freedom from synchronization faults in this software.

### 4.1 Industrial Control System Software

Software under analysis in the present case study is a processing unit of an industrial programmable logic controller (PLC) used in digital control systems. The source code is written in C language; its size is approximately 85 kloc, where 1 kloc = 1 000 lines of code (107 kloc including specific header files and 135 kloc including all header files).

This software is involved in processing inputs and outputs (interfacing control systems with sensors and actuators, including local processing of I/O data), handling network communications, self-monitoring and maintenance functions. For other low-level internal resource management and processing, the software relies



on a commercial real-time kernel threads API: for scheduling, handling of priority and interruption, creation and management of threads and communication resources such as semaphores and queues.

The following is a succinct description of the software architecture and nominal dynamic behavior. After kernel initialization, the main thread configures interruptions, connects interruption routines, and creates all the needed threads and communications resources (there are about 10 of each item created). Each thread is created with one of the following purposes: handling process inputs, screening and detecting state changes in I/O boards; handling time-controlled inputs and outputs and network communications; PLC configuration; update of redundant processing unit, self-checking, etc. These threads run concurrently until the end of the main thread. Shared data include queues, semaphores (some of which are gathered in arrays), some events and configuration values.

## 4.2 Verification of Synchronization Properties

**Model Extraction.** As explained in Sec. 3.1, users of MIEL need to check that each call to a function of interest in the source file can be syntactically seen as such in the source files. Concerning our case study, code inspection indicates that some parts need annotations for the abstracted model to be correct. As argued earlier, two conditions might invalidate that correction: aliases for synchronization primitives names, and calls to synchronization primitives nested within loops. In the former case, we have seen in Sec. 2 that relying on design considerations, it is expected that there is no alias on synchronization primitives names (we have also confirmed this assumption by code inspection). In the latter case, we rely on loop unrolling to extract a correct model.

More precisely, where semaphores are locked then unlocked, the encompassing loop has to be syntactically expanded by `Frama-C` before the analysis by MIEL, as in Fig. 4. Actually, every part of the original code that is both a remote parent of a call to a function of interest, and located within a loop will need to have its loop unrolled in this way. With the additional use of semantic constant folding (done by a `Frama-C` plug-in) the semaphores of Fig. 4 can be identified in the model without ambiguity (because syntactically different after constant folding).

After the initial phases of specifying the aspects of interest in the program — in this case: primitives related to semaphores, threads and message queues — and inserting the appropriate annotations, MIEL performs an automatic analysis. Fig. 4 shows the whole description file needed for the case study. No additional code modifications or annotations are needed.

Given the code snippet of Fig. 4 (right), MIEL yields a model sketched in Fig. 5. During the analysis, the entry points of threads are automatically found. Launching MIEL analysis of the case study files takes a few seconds on a recent Linux workstation. It yields a quite large model file:  $\approx 2000$  lines in the PV language (fifty times smaller than the original C code).

**Verification.** Whether programs are concurrent or not, the analysis of programs with loops is a problem known to be undecidable. Hence, `ALCOOL` analyzes are parametrized by the number of synchronization steps to be unrolled to find

```

function semTake of type lock
  has arg 2 of type delay
  arg 1 of type semaphore

function semGive of type release
  has arg 1 of type semaphore

function taskSpawn
  of type thread creation
  has arg 5 of type function name

function msgQSend of type send
  has arg 5 of type priority
  arg 4 of type delay
  arg 1 of type queue

function msgQReceive of type receive
  has arg 4 of type delay
  arg 1 of type queue

```

```

/*@loop pragma UNROLL 10; */
for( i = 0 ; i < 10 ; i++ ) {
  if( message[i] ) {
    if(semTake(sem_array[i],
              NO_WAIT))
    {
      ...
      message[i] = 0;
      new_frame[i] = ... ;
      semGive(sem_array[i]);
    }
  }
}

```

**Fig. 4.** Inputs: MIEL description file for the case study (left), Frama-C annotation for loop expansion (right)

```

exec_loop =
  ((P(sem_array_0).V(sem_array_0))
   + ...) + ...).
  ((P(sem_array_1).V(sem_array_1))
   + ...) + ...).
  ...
  ((P(sem_array_9).V(sem_array_9))
   + ...) + ...).

```

```

Geometric Model (forbidden area):
[0, +∞[2 × [1, +∞[ × [0, +∞[10
∪ [0, +∞[3 × [1, +∞[ × [0, +∞[9
∪ [0, +∞[7 × [1, +∞[ × [0, +∞[5
∪ [0, +∞[9 × [1, +∞[ × [0, +∞[3
∪ [0, +∞[10 × [1, +∞[ × [0, +∞[2
∪ [0, +∞[11 × [1, +∞[2

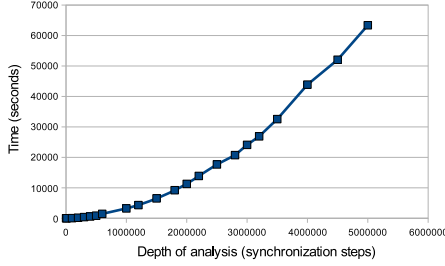
Deadlock Attractor: ∅
Critical sections: No conflict.
Unreachable area: ∅

```

**Fig. 5.** Outputs: Part of the PV model for the code of Fig. 4 (left), ALCOOL output on the case study (right)

possible intrinsic faults in concurrency. During the analysis, ALCOOL priorly chooses branches of “if then else” on which resource primitives appear. The idea is to focus the verification effort on scenarios that might lead to a synchronization fault. The results are displayed as in Fig. 5. Here, the geometric model has dimension 13 (no graphical representation available). Back to the definition of intrinsic faults related to synchronization and concurrency in Sec. 1, the results displayed by ALCOOL give evidence of freedom from deadlocks (item “*Deadlock attractor*”), incorrect protection of shared resources (item “*Critical sections*”), and unexpectedly unreachable program states (item “*Unreachable area*”). Non-determinism and priorities are currently not supported.

When the depth of analysis varies, the qualitative results remain the same as in Fig. 5. As shown in Fig. 6 the computation time grows non-linearly with the depth of analysis. The analysis takes less than one hour at depth  $10^6$ , and less than one minute at depth  $10^5$ . Depth  $5 \times 10^6$  can be practically reached (around 17 hours; computation times obtained on a Z600 Linux workstation).



**Fig. 6.** ALCOOL: Computation time *versus* depth of analysis

We recall that depth of analysis is here expressed in terms of calls to synchronization primitives; the associated concrete traces in the original program are consequently far longer, referring to the 1 : 50 ratio between model and source code (cf. Sec. 4.2, part “model extraction”).

As we have seen, analyzes with ALCOOL can be practically performed at a significant depth. The significance is consolidated under the assumption that loop iterations have limited memory from previous iterations (cf. Sec. 2). More generally, these results confirm the considerations in Sec. 2 about how the characteristics of the addressed classes of programs can be helpful when designing or using formal analysis methods. Code inspection in this software case study indeed shows that synchronization primitives are moderately used (few occurrences of resource requirements, limited interactions between tasks...). The resulting concurrent model is quite simple, given the size of the software, and compared to what absolute concurrency may allow.

In cases where ALCOOL finds a synchronization fault, the variables behaviors have to be thoroughly studied in order to check whether the execution traces that lead to the fault are in fact feasible or not.

No intrinsic fault related to concurrency has been found in the original source code; this outcome is likely when considering high integrity software. In the remainder of this section, we will see how voluntarily inserted faults can be detected by the tools (the instance presented is a deadlock).

**Voluntary Insertion of a Deadlock.** This short presentation is meant as an example; the fault detection ability of the tools is based on the framework in Sec. 3 and is validated elsewhere (against sets of sample codes). We actually insert three threads in the original annotated code of the case study. Fig. 7 (left) shows the additional code accordingly. These additional threads implement three dining philosophers in a configuration known to lead to starvation. Fig. 7 (right) shows a snippet of the PV model generated by MIEL, focused on the additional threads. Fig. 8 shows that ALCOOL indeed finds the deadlock induced by the three additional threads.

```

void Ph1(void) {
    semTake(mutex_a, WAITFOREVER) ;
    semTake(mutex_b, WAITFOREVER) ;
    semGive(mutex_a) ;
    semGive(mutex_b) ;
}
void Ph2(void) {
    semTake(mutex_b, WAITFOREVER) ;
    semTake(mutex_c, WAITFOREVER) ;
    semGive(mutex_b) ;
    semGive(mutex_c) ;
}
void Ph3(void) {
    semTake(mutex_c, WAITFOREVER) ;
    semTake(mutex_a, WAITFOREVER) ;
    semGive(mutex_c) ;
    semGive(mutex_a) ;
}

static void create_tasks(void) {
    mutex_a = semBCreate() ;
    mutex_b = semBCreate() ;
    mutex_c = semBCreate() ;

    taskSpawn("Russell",90,
              VX_NO_STACK_FILL,1000,Ph1,
              0,0,0,0,0,0,0,0,0,0);
    taskSpawn("Goedel",90,
              VX_NO_STACK_FILL,1000,Ph2,
              0,0,0,0,0,0,0,0,0,0);
    taskSpawn("Hilbert",90,
              VX_NO_STACK_FILL,1000,Ph3,
              0,0,0,0,0,0,0,0,0,0);
    /*Rest of the initial code */
}

do_Ph1 =
(P(mutex_a).P(mutex_b)
.V(mutex_a).V(mutex_b))

do_Ph2 =
(P(mutex_b).P(mutex_c)
.V(mutex_b).V(mutex_c))

do_Ph3 =
(P(mutex_c).P(mutex_a)
.V(mutex_c).V(mutex_a))
....

init: do_Task1 | do_Task2
      | do_Task3 | do_Task4
      | do_Task5 | do_Task6
      | do_Task7 | do_Task8
      | do_Task9 | do_Task10
      | do_Task11 | do_Task12
      | do_Task13 | do_Ph3
      | do_Ph2 | do_Ph1

```

**Fig. 7.** Inserting philosophers in the original source code (left), Philosophers in the PV model (right)

```

Geometric Model (forbidden area):
[0, +∞[2 × [1, +∞[ × [0, +∞[13 ∪ [0, +∞[3 × [1, +∞[ × [0, +∞[12
∪ [0, +∞[9 × [1, +∞[ × [0, +∞[6 ∪ [0, +∞[11 × [1, +∞[2 × [0, +∞[3
∪ [0, +∞[13 × [1, 3[ × [2, 4[ × [0, +∞[ ∪ [0, +∞[13 × [2, 4[ × [0, +∞[ × [1, 3[
∪ [0, +∞[14 × [1, 3[ × [2, 4[

Local Deadlock Attractor: [0, +∞[13 × [1, 2[3

```

**Fig. 8.** ALC00L output on the modified case study

## 5 Related Work

Tools implementing model checking techniques [2, 9] usually work on a representative *model* of the program to be analyzed, e.g. SPIN [21], FAST [5], UPPAAL [6]. While SPIN addresses general concurrent systems and their synchronization issues, UPPAAL is dedicated to real-time systems and is thus more focused on to timing issues, e.g. *delays*. The FAST tool is dedicated to the analysis of infinite systems. It mainly aims at computing the exact (infinite) set of configurations

reachable from a given set of initial configurations. In some cases, the verification models can be generated by auxiliary tools, e.g. the MODEX tool for SPIN [22]; the TOPICS tool for FAST [23]. As seen in sections 3 and 4, the ALCOOL tool can similarly be used in conjunction with the MIEL automatic model extractor.

Automata theory is a widely used framework for the theoretical foundations of model checking tools; for instance SPIN, FAST and UPPAAL respectively work on Büchi automata, counter automata and timed automata.

In contrast, the ALCOOL tool is based on the *topological* notion of directed spaces. Generally speaking, the parallel composition operator is modeled by the Cartesian product in a well-suited category:  $\llbracket A|B \rrbracket = \llbracket A \rrbracket \times \llbracket B \rrbracket$

An automaton is a directed graph endowed with some extra structure. Directed graphs form a category in which any Cartesian product exists though they do not fit to concurrency theory. Our claim is that using directed spaces is natural since the Cartesian product in the category of topological spaces behave as concurrency theory expects.

Other existing approaches include predicate abstraction and refinement (ARMC [27], BLAST [7], SLAM [4]), symbolic model checking (NuSMV [8]), or combining model checking and theorem proving (SLAB [14]).

## 6 Conclusion

This paper reports on the development of a tool-based methodology to demonstrate the freedom from certain types of software faults, and its practical application to an industrial control software case study. Two main phases are involved: automatically extract a correct and representative model from C code, and then check for properties in the model.

**Lessons Learned.** The methodology presented in this paper aims at analyzing software that can be found in systems either safety related, or important to availability, in power plants. Experience in assessments of control systems has lead us to identify generic characteristics for such software (cf. Sec. 2). For instance, memory allocation, task and synchronization resources creation are usually performed only during a dedicated initialization phase. We have also relied on a taxonomy of synchronization faults that might be postulated in control systems software, established in previous works (cf. Sec. 1), and discussed how the tools can give rigorous evidence against a part of it. Finally, Sec. 4 shows that the tools scale up efficiently to analyze a real-world control system software unit. Also, voluntarily inserted faults have been identified.

An important learning is that generic characteristics of targeted classes of software can be taken in account in order to provide rigorous verification tools with high efficiency.

The soundness of this approach depends on the following requirements. The main requirement (*R1*) is that there is no dynamic creation of threads nor synchronization resources. The other requirements hold only for model extraction with MIEL: (*R2*) the programming language must be C, (*R3*) there must be no

aliases on the names of the functions of interest, and (R4) calls to functions of interest must be syntactically distinct. An important claim regarding applicability of the methodology is that these requirements are compatible with the generic characteristics of the targeted class of software. Under these assumptions, the methodology should widely apply within the considered class.

Additionally, let us examine the following cases. If one wants to analyze software where:

- only (R1) holds, then ALCOOL can be applied to a model extracted by other means than MIEL (e.g. expert knowledge or another tool);
- only (R1) to (R3) hold: the required user manipulation for having a sound model extraction with MIEL should remain fairly reasonable and practicable, i.e. unrolling loop so that each access to synchronization resources becomes syntactically noticeable and distinct for MIEL, as in Sec. 4.2. If user intervention is thought too demanding, one would consider the case below.
- only (R1) and (R2) hold: we are currently working on extending the methodology with an enhanced model extractor, to deal with this case; cf. Sec. 6, part “ongoing work”.

**Ongoing Work.** We are currently experimenting the use of semantic analyzes using Framac’s value analysis [10] to provide a sound value analysis for concurrent programs in order to formally and accurately identify synchronization variables and threads, and use this information to refine model extraction.

The ALCOOL tool is based on a mathematical library which allows it to deal with geometric models drawn on a higher dimensional torus. However the representation of a finite directed graph on a hypertorus, which is known to be theoretically possible, has not been implemented yet. Roughly speaking, ALCOOL is meant to provide any concurrent program with a mathematical structure which generalizes the notion of control flow graph. Once this structure is determined, standard methods from static analysis can be applied. The tool-based approach should then provide a finer-grained analysis of message passing mechanisms.

## References

1. Aiken, A., Foster, J.S., Kodumal, J., Terauchi, T.: Checking and inferring local non-aliasing. In: PLDI, pp. 128–140 (2003)
2. Baier, C., Katoen, J.P.: Principles of Model-Checking. MIT Press, Cambridge (2008)
3. Balabonski, T., Haucourt, E.: A geometric approach to the problem of unique decomposition of processes. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 132–146. Springer, Heidelberg (2010)
4. Ball, T., Rajamani, S.K.: The SLAM project: debugging system software via static analysis. In: POPL, pp. 1–3 (2002)
5. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: acceleration from theory to practice. STTT 10(5), 401–424 (2008)
6. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)

7. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker BLAST. *STTT* 9(5-6), 505–525 (2007)
8. Cimatti, A., Clarke, E.M., Giunchiglia, F., Roveri, M.: NUSMV: A new symbolic model checker. *STTT* 2(4), 410–425 (2000)
9. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: *Model Checking*. MIT Press, Cambridge (1999)
10. Cuoq, P., Prevosto, V.: FRAMA-C's value analysis plug-in. CEA LIST Technical Report (2010), <http://frama-c.com/download/frama-c-value-analysis.pdf>
11. Cuoq, P., Signoles, J., Baudin, P., Bonichon, R., Canet, G., Correnson, L., Monate, B., Prevosto, V., Puccetti, A.: Experience report: OCaml for an industrial-strength static analysis framework. In: *ICFP*, pp. 281–286 (2009)
12. CWE Common Weakness Enumeration —, <http://cwe.mitre.org/>
13. Dijkstra, E.W.: Cooperating sequential processes. In: *Programming Languages: NATO Advanced Study Institute*, pp. 43–112. Academic Press, London (1968)
14. Dräger, K., Kupriyanov, A., Finkbeiner, B., Wehrheim, H.: SLAB: A certifying model checker for infinite-state concurrent systems. In: Esparza, J., Majumdar, R. (eds.) *TACAS 2010*. LNCS, vol. 6015, pp. 271–274. Springer, Heidelberg (2010)
15. Emanuelsson, P., Nilsson, U.: A Comparative Study of Industrial Static Analysis Tools. Linköping University Technical Report (2008)
16. Fajstrup, L., Goubault, E., Raußen, M.: Algebraic topology and concurrency. *Theoretical Computer Science* 357, 241–278 (2006)
17. FRAMA-C Software Analyzers —, <http://frama-c.com/>
18. Goubault, E.: Geometry and concurrency: a user's guide. *Mathematical Structures in Computer Science* 10(4), 411–425 (2000)
19. Goubault, E., Haucourt, E.: A practical application of geometric semantics to static analysis of concurrent programs. In: Abadi, M., de Alfaro, L. (eds.) *CONCUR 2005*. LNCS, vol. 3653, pp. 503–517. Springer, Heidelberg (2005)
20. Grandis, M.: *Directed Algebraic Topology*. New Mathematical Monographs. Cambridge University Press, Cambridge (2009)
21. Holzmann, G.J.: *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, Reading (2003)
22. Holzmann, G.J., Ruys, T.C.: Effective bug hunting with SPIN and MODEX. In: Godefroid, P. (ed.) *SPIN 2005*. LNCS, vol. 3639, p. 24. Springer, Heidelberg (2005)
23. Labbé, S., Sangnier, A.: Formal verification of industrial software with dynamic memory management. In: *IEEE PRDC*. pp. 77–84 (2010)
24. Labbé, S., Thuy, N.: Formal verification of freedom from intrinsic software faults in digital control systems. In: *ANS NPIC&HMIT*, pp. 2191–2201 (2010)
25. Larochelle, D., Evans, D.: Statically detecting likely buffer overflow vulnerabilities. In: *USENIX Security Symposium*, pp. 177–190 (2001)
26. Nachbin, L.: *Topology and Order*. Mathematical Studies, vol. 4. Van Nostrand, Princeton (1965)
27. Podelski, A., Rybalchenko, A.: ARM: The logical choice for software model checking with abstraction refinement. In: Hanus, M. (ed.) *PADL 2007*. LNCS, vol. 4354, pp. 245–259. Springer, Heidelberg (2006)
28. Thuy, N., Ourghanlian, A.: Dependability assessment of safety-critical system software by static analysis methods. In: *DSN*, pp. 75–79 (2003)

# Evolutionary Risk Analysis: Expert Judgement

Massimo Felici<sup>1</sup>, Valentino Meduri<sup>1</sup>,  
Bjørnar Solhaug<sup>2</sup>, and Alessandra Tedeschi<sup>1,\*</sup>

<sup>1</sup> Deep Blue S.r.l.  
Piazza Buenos Aires 20, 00198 Roma, Italy  
[alessandra.tedeschi@dblue.it](mailto:alessandra.tedeschi@dblue.it)

<http://www.dblue.it/>

<sup>2</sup> SINTEF ICT  
P.O. Box 124 Blindern, 0314 Oslo, Norway

**Abstract.** New systems and functionalities are continuously deployed in complex domains such as Air Traffic Management (ATM). Unfortunately, methodologies provide limited support in order to deal with changes and to assess their impacts on critical features (e.g. safety, security, etc.). This paper is concerned with how change requirements affect security properties. A change requirement is a specification of changes that are to be implemented in a system. The paper reports our experience to support an evolutionary risk analysis in order to assess change requirements and their impacts on security properties. In particular, this paper discusses how changes to structured risk analysis models are perceived by domain experts by presenting insights from a risk assessment exercise that uses the CORAS model-driven risk analysis in an ATM case study. It discusses how structured models supporting risk analysis help domain experts to analyse and assess the impact of changes on critical system features.

**Keywords:** Air Traffic Management, Change Requirements, Security Requirements, Evolutionary Risk Analysis, CORAS.

## 1 Changes and Risks

Standards, guidelines and best practices advise to assess the impact of changes. In safety-critical contexts, or other domains characterised by stringent critical non-functional requirements (e.g. reliability, security, safety), it is necessary to assess how changes affect system properties. This aspect concerns system artifacts at any developmental stage. For instance, safety cases need to be adapted in order to take into account any emergent system knowledge (e.g. system failures), system requirements need to change in order to accommodate evolving environmental factors, testing activities need to be performed again in order to assess software and configuration changes. Similarly, risk analysis needs to take into account changes and emergent hazards because changes and evolution may affect the risk picture. On the one hand, changes and evolution may introduce new or

---

\* Corresponding author.



different threats and stress system vulnerabilities. On the other hand, changes and evolution may provide opportunities for enhancing system dependability.

Unfortunately, support throughout the system lifecycle for systematically dealing with changes and evolution is patchy. Recent research (e.g. see [18, 19, 24, 25, 26] for work concerned in particular with the ATM domain) highlights some challenges for risk assessment methodologies. This paper is concerned with challenges for current risk assessment methodologies in dealing with changes in particular for safety-critical domains such as ATM. It discusses an evolutionary risk analysis by means of structured models. Changes may affect various system artifacts (e.g. requirements, design models). They require such artifacts to be updated and reassessed eventually. This inevitably increases project costs associated with maintaining a valid risk assessment for the system. It may affect reusing strategies as well as any effort to localise changes into specific arguments (hence, increasing intrinsic system complexities). However, although different structured models (e.g. design models, risk models, safety arguments, etc.) are used to support risk analysis, this paper is concerned with whether structured models provide suitable support in order to acquire expert judgement while risk analysis deals with changes — *How do models support assessing the impact of changes? How do changes into models shift risk perception?* This paper reports our experience of evolutionary risk analysis supported by the CORAS approach. Section 2 reviews relevant work on risk analysis, and highlights guidelines and methodologies drawn from the ATM domain. Section 3 describes a case study drawn from ongoing activities within the ATM domain. Section 4 introduces the basic concepts of the CORAS approach to model-driven risk analysis. Section 5 reports our evolutionary risk analysis and the investigation results taking into account expert judgements during dedicated risk analysis sessions. Section 6 summarises our lessons learned.

## 2 Related Work on Risk Analysis

The ISO 31000 risk management standard [1] defines risk management as coordinated activities to direct and control an organisation with regard to risk, where risk can be understood as the combination of the likelihood and consequence of an unwanted incident. The risk management process includes the phases of context establishment, risk assessment and risk treatment. Context establishment involves defining the target of analysis and setting the risk criteria, whereas risk assessment involves risk identification, risk analysis and risk evaluation. The risk analysis estimates the likelihoods and consequences of risks, and the risk evaluation compares the resulting risk levels with the criteria in order to determine which risks must be mitigated by treatment options. The ISO 31000 standard stresses the importance of handling changes. However, the standard comes with no explicit guidelines for how to manage and assess changing risks. Other established risk analysis methods such as OCTAVE [2] and CRAMM [3, 4] follow a process similar to ISO 31000. Such methods focus on a particular configuration of the target at a particular point in time, and the results are therefore valid

only for this particular snapshot and for the assumptions being made. When addressing changing systems, there is a need for risk analysis methods that comes with guidelines and techniques for how to understand, to assess and to document risks as changing risks. There is a need for risk modelling techniques to facilitate the tasks of assessing changing risks. Structured risk models represent unwanted incidents with their causes and consequences by graphs (e.g. Cause-consequence analysis and Bayesian networks [9]), trees (e.g. Fault Tree Analysis [6], Event Tree Analysis [7] and Attack trees [8]) or block diagrams [5].

CORAS threat diagrams [11] describe how threats exploit vulnerabilities to initiate threat scenarios and unwanted incidents, as well as the likelihoods and consequences of incidents. Risks graphs represent an abstraction of each of the above mentioned risk modelling techniques in the sense that each of them can be understood as a risk graph instantiation [12]. Risk graphs facilitate the structuring of events that lead to incidents, as well as the estimation of likelihoods. The notation is provided a formal semantics and comes with a calculus for reasoning about likelihoods. Unfortunately, risk modelling techniques provide limited support for the identification, modelling and assessment of changing risks. This paper presents an evolutionary risk analysis that generalises the risk graph notation in order to support the modelling of risks that evolve. This generalisation is in turn instantiated in CORAS, thus supporting a CORAS risk analysis process with methods and techniques for assessing and documenting evolving risks.

The risk associated with the *high-couple* and *complex interactions* emerging among system 'components' is characterising for many technology systems [15], in particular ATM systems. The socio-technical nature of such systems involves diverse entities interacting within operational environments. The SHELL model characterises the socio-technical nature of ATM systems and their distributed nature [16]. Causal analysis of failures in such systems highlights that failures are often *interaction* or *organisational* failures. The Cheese model provides a characterisation how failures emerge within organisations [17]. Safety mechanisms and barriers address to a certain extent threats and vulnerabilities across organisational layers [17]. Such concepts underlie safety nets in the ATM domain [20]. The EUROCONTROL Permanent Commission approved a number of ATM safety regulatory requirements, known as ESARRs, but these represent only one element of a wider framework for ATM safety regulation. These requirements are mandatory for all EUROCONTROL Member States and aim at harmonising the ATM safety regulation across the ECAC area. ECAC States not member of EUROCONTROL are strongly encouraged to adopt the ESARRs as well. EUROCONTROL, through the Safety Regulation Commission (SRC), is developing a harmonised framework for the safety regulation of ATM, for implementation by States. The core of the framework is represented by harmonised safety regulatory requirements, ESARRs. ESARR 4 (Risk Assessment and Mitigation in ATM) [21] and ESARR 6 (Software in ATM systems) [22] are of particular relevance. In order to comply with the ESARRs and to support the deployment of ATM systems, EUROCONTROL is developing the Integrated Risk Picture

Methodology (IRP) [23]. Relevant guidelines and requirements stress that risk analysis needs to deal with changes, hence, an evolutionary risk analysis.

### 3 ATM Case Study

In Air Traffic Management the increase of air traffic is pushing the human performances to the limit, and the level of automation is growing dramatically to deal with the need for fast decisions and high traffic load. There is an increase in data exchange between aircraft and ground and between Area Control Centers (ACCs) due to new systems, equipments and ATM strategies. There is a growing relevance for dependability, security and privacy aspects. Software and devices must adapt to evolution of processes, introduction of new services, and modification of the control procedures. This adaptation shall preserve safety, security and dependability and be able to face new and unexpected security problems arising from evolution. Introducing Safety and Security relevant methodologies in the ATM context requires us to understand the risk involved in order to mitigate the impact of possible future threats and failures. The ATM 2000+ Strategic Agenda [29] and the Single European Sky ATM Research [30] (SESAR) Initiative, involve a structural revision of ATM processes, a new ATM concept and a system approach for the ATM Network. This requires ATM services to go through significant structural, operational and cultural changes that will contribute towards SESAR. The SESAR Operational Concept is a trajectory based system, which relies on precise trajectory data, combined with cockpit displays of surrounding traffic. The execution of such trajectory by Air Traffic Management services will ensure that traffic management is carried out safely and cost efficiently within the infrastructural and environmental constraints.

Changes to the business trajectory must be kept to a minimum. Modifications to the business trajectory are best met through maintenance of capacity and throughput rather than optimisation of an individual flight. Changes will ideally be performed through a Collaborative Decision Making mechanism but without interfering with the pilots' and controllers' tactical decision processes required for separation provision, for safety or for improvement of the air traffic flow, thanks to the new tools that will be introduced in the Controller Working Position (CWP). Sharing access to accurately predicted, business trajectories information will reduce uncertainty and give all stakeholders a common reference, permitting collaboration across all organisational boundaries.

Fundamental to the entire ATM Target Concept is a net-centric operation based on: (1) a powerful information handling network for sharing data; (2) new air-air, ground-ground and air-ground data communications systems, and; (3) an increased reliance of airborne and ground based automated support tools. The ATM case study is concerned with changes to operational processes of managing air traffic in Terminal Areas. Arrival management is a very complex process, involving different actors. Airport actors are private organisations and public authorities with different roles, responsibilities and needs. The subsequent introduction of new tools (e.g. Queue Managers) and the introduction of a new ATM

network for the sharing and management of information affect the ATM system at an organisational level.

### 3.1 Organisational Level Change

The introduction of the AMAN (Arrival Manager) affects Controller Working Positions (CWPs) as well as the Area Control Center (ACC) environment as a whole. The main foreseen changes in the ACC from an operational and organisational point of view are the automation of tasks (i.e. the usage of the AMAN for the computation of the Arrival Sequence) that in advance were carried out by Air Traffic Controllers (ATCOs), a major involvement of the ATCOs of the upstream Sectors in the management of the inbound traffic. These changes will also require the redefinition of the Coordinator of the Arrival Sequence Role, who will be responsible for monitoring and modifying the sequences generated by the AMAN, and providing information and updates to the Sectors. The AMAN's interfaces provide access to different roles, and authorisations need to make information available only to authorised personnel or trusted systems.

### 3.2 Security Properties

Main aspects of security in ATM relate to self protection of facilities and resources of the ATM system as well as coordination with Air Defense authorities for exchange of information and coordination in case of aviation security incidents. The ATM is above all a cooperative system, based on mutual trust primarily between airspace users and ATM staff. Traffic surveillance relies currently on sensors that can bring additional confidence to the integrity of information received. Surveillance of traffic and monitoring of information may be used to detect civil aircraft operating in such a manner as to raise suspicion of seizure by terrorists or hijackers. The introduction of new systems and the reorganisation of ATM services are facing new security issues. Both ATM security and safety are concerned with protecting ATM assets and services, that seeks to safeguard the overall airspace from unauthorised use, intrusion or other violations. EUROCONTROL has recently issued several guidelines highlighting security as a critical factor for future ATM developments and identifying relevant security

**Table 1.** Security Properties

<b>Security Property</b>	<b>Description</b>
Information Protection	Unauthorised actors (or systems) are not allowed to access confidential queue management information.
Information Provision	The provisioning of information regarding queue management sensitive data by specific actors (or systems) must be guaranteed 24 hours a day, 7 days a week, taking into account the kind of data shared, their confidentiality level and the different actors involved.

methodologies [27, 28]. Table 1 identifies critical security properties to be guaranteed at the process and organisational level. Our risk analysis study focuses on such security properties.

## 4 Model-Driven Risk Analysis: The CORAS Approach

CORAS [11] is an approach to risk analysis that consists of three tightly integrated parts, namely the CORAS method, the CORAS language and the CORAS tool. The method is based on the ISO 31000 risk management standard [1] and consists of eight steps. The four first steps correspond to the context establishment, whereas the remaining four are risk identification, risk estimation, risk evaluation and risk treatment. The method comes with concrete tasks and practical guidelines for each step, and is supported by several risk analysis techniques. The CORAS language consists of five kinds of diagrams, each of which provides support for specific tasks throughout the whole risk assessment process. The method is supported by the tool, which is an editor for on-the-fly risk modelling. The most important kind of CORAS diagram is threat diagrams which are used for risk identification and risk estimation. The language constructs are firmly based on an underlying well-defined conceptual framework for reasoning about risk, and includes: human and non-human threats, vulnerabilities, threat scenarios, unwanted incidents and assets. Threat diagrams are used for on-the-fly risk modelling during structured brainstorming that involves personnel with expert knowledge about the target of analysis. In such a setting, the diagrams must be intuitive and easy to understand, also for people with little technical background and little experience in risk analysis. For this reason, the CORAS language constructs are graphical, easily understandable symbols. In the following we describe and exemplify selected parts of the generalised CORAS approach and the risk analysis of the ATM case study, focusing on the identification and modelling of changing risks since this is the core part of the process [13].

*Context Establishment.* The context establishment includes making the target description, setting the focus and scope of the analysis, identifying the assets, and setting the risk evaluation criteria. In the setting of evolving systems, the context establishment moreover includes the specification of the changes to the target, the changes in assets or asset values, and the changes to the evaluation criteria, if any. Figure 1 shows the risk evaluation criteria partially based on the EUROCONTROL safety regulatory requirement (ESARR4) [21].

	Insignificant	Minor	Moderate	Major	Catastrophic
Rare	Green	Green	Yellow	Yellow	Red
Unlikely	Green	Green	Yellow	Yellow	Red
Possible	Green	Yellow	Yellow	Red	Red
Likely	Green	Yellow	Yellow	Red	Red
Certain	Green	Yellow	Red	Red	Red

Fig. 1. Risk evaluation criteria

Our target of analysis, both its structure and its behavior before and after the changes, were specified by UML 2.0 diagrams [14]. In the risk analysis, the identified assets of confidentiality and availability correspond to the security properties of Information Protection and Information Provision, respectively.

*Risk Identification.* The risk identification was conducted as a structured brainstorming involving personnel with first hand knowledge about the target of analysis and strong background from ATM. By conducting a walkthrough of the UML target description, the risks were identified by systematically identifying unwanted incidents, threats, threat scenarios and vulnerabilities. The results were documented by means of CORAS threat diagrams. So far, the methods and techniques are as for traditional risk analyses. However, a guiding principle for our risk analysis method generalised to handle evolving systems and risks is that only the risk analysis results that are affected by the system changes should be addressed anew. In our generalisation of CORAS we provide techniques and language support for tracing changes from the target system to the risk model so as to enable the identification of the parts of the risk models that are not affected by changes and therefore maintain their validity. Because our main concern in this paper is to present the insights from the evolutionary risk assessment case study regarding expert judgement, we refer to the full report for further details about the method and techniques [13]. Figure 2 shows a fragment of a CORAS threat diagram resulting from the identification of changing risks.

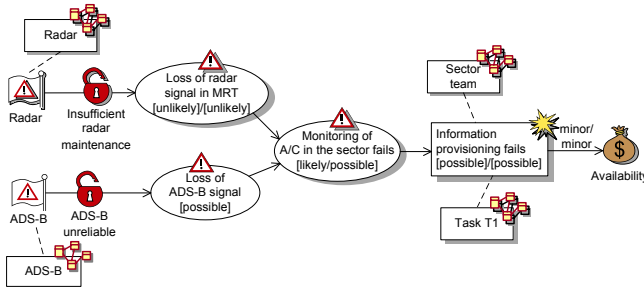


Fig. 2. Threat diagram with changing risks

Compared with the standard CORAS language, there are two main language extensions to support the risk analysis of evolving systems. First, the rectangle icons with the system diagram symbol (e.g. the one named Task T1 - the first task in the arrival management work process) exemplify the new construct for referring to the target of analysis. Second, the threat diagram language constructs of threat, unwanted incident, asset, etc. are generalised to three modes with different appearances, namely the modes before, after and before-after. The before constructs are in grey shade and dashed outline and represent parts of the risk picture that are valid only before the changes. The after constructs are in colour and solid outline and represent parts that are valid only after the changes.

The before-after constructs are two-layered and represent parts that are valid both before and after changes. The explicit references to the target system in the threat diagrams facilitate the identification of the parts of the risk picture that are affected by system changes. For example, in the ATM risk analysis, the radar was not subject to the ATM system changes. Hence, the vulnerability Insufficient radar maintenance and the threat scenario Loss of radar signal in MRT (multi-radar tracking) are maintained under change. The threat scenario Monitoring of A/C (aircraft) in the sector fails, on the other hand, is affected due to the introduction of the ADS-B (automatic dependent surveillance-broadcast). Notice that we take into account here the dependencies of elements on their preceding elements in the threat diagrams. The different appearance of the three modes of the language constructs facilitates the immediate recognition of the risk changes that are modelled. This feature is an important part of supporting the risk identification brainstorming and for appropriately documenting the results. In order to highlight the risk changes, the CORAS tool implements the functionality of changing between the views of before, after and before-after. Figure 3 shows such feature on an extract of the threat diagram.

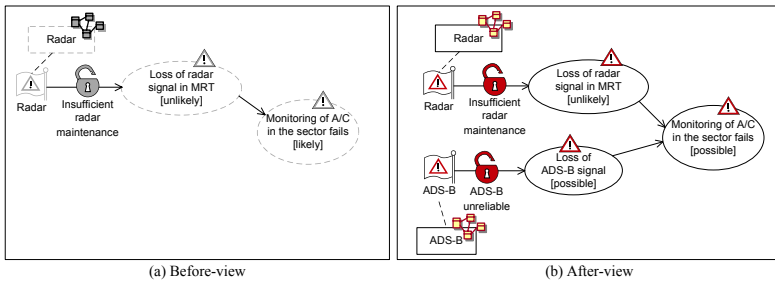


Fig. 3. Two views on changing risks

*Risk Estimation.* The risk estimation basically amounts to estimating likelihoods and consequences for unwanted incidents. Usually, we also estimate likelihoods for threat scenarios in order to get a better basis for estimating the likelihood of unwanted incidents and to understand the most important sources of risks. The CORAS calculus provides rules for calculating and reasoning about likelihoods. Diagram elements of mode before-after are assigned a pair of likelihoods. The former denotes the likelihood before the changes. The latter denotes the likelihood after the changes. Diagram elements of mode before or after are assigned only a single likelihood. The distinction is likewise for the consequence estimates. Hence, the threat diagrams document not only risks that emerge, disappear or persist, but also how risk levels change. For example, the threat scenario Monitoring of A/C in the sector fails is assigned the likelihood likely before the changes and the likelihood possible after the changes. The likelihood drops due to the introduction of the ADS-B. Information provisioning fails is an unwanted incident, and therefore constitutes a risk. Its likelihood is possible both before

and after the changes, while its consequence for the Availability asset is minor as annotated on the relation between the unwanted incident and the asset.

*Risk evaluation.* During the risk evaluation we first calculate the risk levels by using the risk matrix exemplified in Figure 1 and the likelihood and consequence estimates from the risk estimation. We then compare the risk levels with the risk evaluation criteria to determine which risks that must be treated or evaluated for treatment. The risk estimation is supported by CORAS risk diagrams which we do not show here due to space constraints. These diagrams show the changing risks together with the threats that initiate them and the assets they harm. The unwanted incident Information provisioning fails, for example, has the likelihood possible and the consequence minor before and after the ATM system changes, which yields a low risk level.

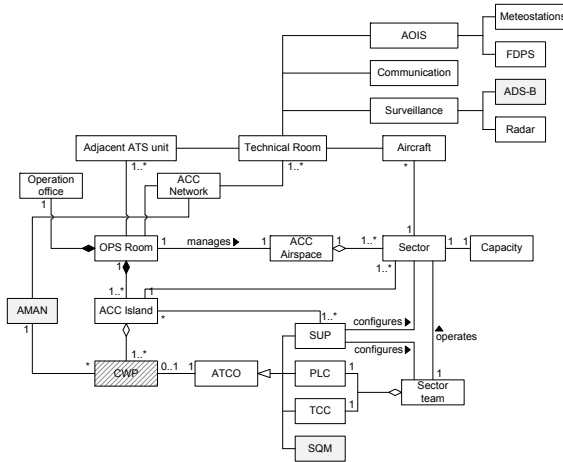
*Risk treatment.* The purpose of the risk treatment is to identify options for risk mitigation for the unacceptable risks. In the presence of changes, the treatments should ensure that an acceptable level of risk is maintained under planned changes or foreseen evolutions. This final step of the process is conducted as a structured brainstorming with a walkthrough of the threat diagrams documenting the unacceptable risks. The CORAS treatment diagrams support such task.

## 5 Expert Judgement in Evolutionary Risk Analysis

This section discusses further the risk analysis concerned with the Organisation Level Change and the security properties of information protection and information provision. The technical solutions we use in the ATM case study are the modelling language for documenting and reasoning about changing risks, and the assessment method for conducting and documenting the risk analyses of changing and evolving systems. Our work is concerned with supporting structured approaches to changes, capturing security properties affected by changes, and providing mechanisms dealing with subsequent changes. The investigation involved a focused risk analysis of the ATM Changes Requirements and their relevant Security Properties. The risk analysis was conducted by means of design models capturing the main entities characterising an ATM domain. In order to take into account how change requirements, i.e. planned changes that are to be implemented, affect the ATM contexts and their organisations, the risk analysts produced structured (UML) models capturing the ATM settings before and after the changes. These models were reviewed and revised by ATM experts who are currently involved in various activities concerning the SESAR project. The models were used as starting point for the risk analysis in order to have a common understanding of the change requirements among the people (i.e. ATM experts) involved in the risk analysis exercise. Figure 4, for instance, shows a conceptual model of an ACC after changes. The shaded elements represent parts that are introduced to the ACC, whereas the diagonally striped element represents a part that is modified. Similar models have been drawn for other aspects characterising ATM settings and practices (e.g. different UML models capturing different







roles and procedures). These models supported discussion and communication between ATM experts and Risk Analysis modellers. Moreover, they have been used to focus and organise the risk analysis on both before and after changes.



**Fig. 4.** Conceptual overview of ACC after changes

The risk analysis trial was conducted during a dedicated two-day workshop. The first day of the workshop was dedicated to the risk analysis of the before case. The second day of the workshop was dedicated to the risk analysis of the after case, that is, to the risk analysis regarding the change requirements and how they potentially affect security properties. Table 2 shows examples of the identified hazardous situations modelled and analysed by CORAS diagrams.

**Table 2.** Examples of hazardous situations

 <b>Who/what caused it?</b>	 <b>What is the scenario or incident? What is harmed?</b>	 <b>What makes it possible?</b>	 <b>Target element</b>
System Failure	Loss of the AMAN leads to loss of provisioning of information to ATCO		AMAN
Attacker	Attacker broadcasts false ADS-B signals, which lead to the provisioning of false arrival management data.	Use of ADS-B; dependence on broadcasting	ADS-B
Software failure	Provisioning of unstable or incorrect sequence by the AMAN leading to ATCO reverting to manual sequencing	Immature (unreliable) software	AMAN

The first activity involved a high-level risk analysis of the AMAN introduction. The structured models were used in order to support a walkthrough analysis of the change requirements and to identify potential hazardous situations. The subsequent risk analysis phases involved risk identification, risk estimation and risk evaluation. Figure 5 shows sample risk analysis models for the after case. The model supports a structured risk analysis of change requirements and their impact on critical security properties. Among the risk analysis outcomes were models assessing emergent risk due to the change requirements and their impact on critical security properties. These models supported a systematic way of analysing the risk of changes and their impact on security aspects.

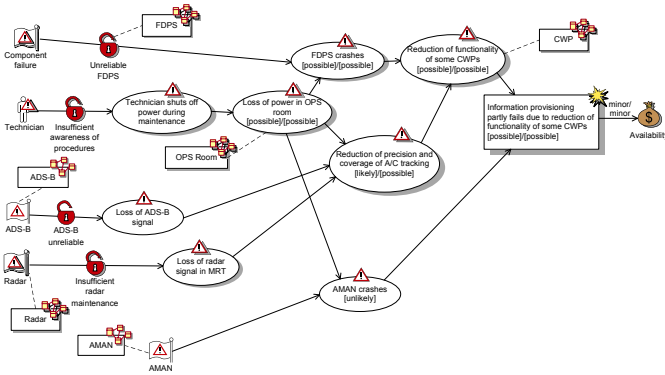
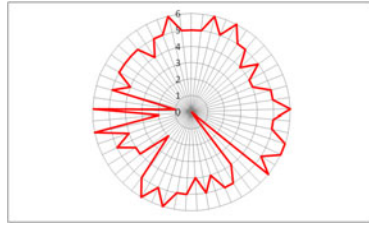


Fig. 5. A sample risk model for reduction of functionality

Note that the model captures different hazards and relate them to the target of analysis as well as to other relevant hazards. The resulting network of causalities is used in order to assess the impact of changes on the risk picture and relate them to specific security properties. The same network of causalities is then used to assess the risk in terms of frequency of events and their severities. This is useful to revise risks with respect to emergent hazards related to the change requirements. The final phases involved the identification and discussion of suitable mitigations for the analysed hazards. ATM experts were involved in the risk analysis. They reviewed the models describing the change requirements and actively participated in the risk analysis trial. In order to account for model effectiveness as a means to investigate risk analysis with respect to change requirements, we collected relevant information about the experts' profiles and perceptions. At the beginning of the risk analysis trial, ATM experts as well as other project partners filled in a Safety Culture Questionnaire. The questionnaire has been developed and tailored by Deep Blue taking into account relevant information drawn from the ATM domain [31, 32]. It covered ten different areas (e.g. Regulation and Standards, Safety Assessment) by fifty three questions contributing to Safety Culture. Figure 6 shows a Safety Culture Profile for one of the ATM experts taking parts in the risk analysis trials.



**Fig. 6.** Safety Culture Profile

The reason we collected expert knowledge with respect to Safety Culture is because Risk Management and Change Management are often critical practices. This allows us to understand further the relationship between safety and risk with respect to change requirements and relevant security properties. After each one of two risk analysis sessions, we collected other information by an Evolutionary Risk Questionnaire. Figure 7 shows some of the questionnaire statements.

	Deep Blue					Evangelix	
1.1 This AoC increases the likelihood of well-understood current hazards that will exist in the future	0	1	2	3	4	5	6
1.2 This AoC creates new hazards synergistically with other AoCs or with the Future that would not have come into being without the presence of the AoC	0	1	2	3	4	5	6
1.3 This AoC increases the subjective likelihood of Future hazards to an unacceptable level	0	1	2	3	4	5	6
1.4 This AoC creates increased potential for human error, procedural non-compliance or equipment failure	0	1	2	3	4	5	6
1.5 This AoC decreases the resilience of the projected safety system	0	1	2	3	4	5	6

**Fig. 7.** Sample questionnaire statements

The questionnaire has been developed and tailored by Deep Blue in order to account of perceived hazards, hence risk perception, as captured by risk analysis models concerning current and future change requirements. The questionnaire consists of twelve different points drawn from relevant work in the ATM domain [33], and is concerned with Area of Changes (AoC) as a means to discuss relevant changes requirements and hazards pertinent to current and future ATM. Figure 8 shows the questionnaires’ outcomes (for the same expert). It is interesting to notice how risk perceptions change with respect to current situation and future ones. The dedicated risk analysis sessions helped to capture this shift in perception with respect to change requirements. The specific points highlighted by the questionnaire identify aspects for further investigation in order to refine and gain confidence on the risk analysis concerning future changes requirements.

The identification of specific areas of concerns for changes supports the use of structured models in order to assess the impact of changes. However, evolutionary risk analysis needs to be organised and supported adequately.

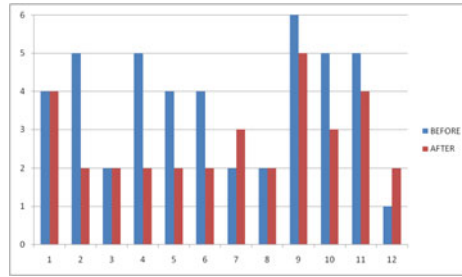


Fig. 8. Evolutionary risk perception

## 6 Conclusions

This paper enhances how structured models may support expert judgement while conducting an evolutionary risk analysis. The use of structured models tailored and organised for an evolutionary risk analysis helps to identify potential areas of concerns due to changes. The evolutionary risk analysis presented in this paper consists of different activities: (1) identify relevant design models, (2) build risk assessment models for before and after the changes, (3) run dedicated before and after risk analysis sessions, (4) monitor (by means of qualitative assessment) risk perception shifts in order to identify areas of concerns. Our empirical results provide insights supporting evolutionary risk analysis by means of structured models and expert judgement. The generality of the different activities would support the evolutionary risk analysis across different domains. Further work intends to involve an increasing number of experts in order to gain further evidence supporting evolutionary risk analysis, but also to support statistical accounts of how changes affect risk perceptions in risk analysis.

**Acknowledgements.** This work has been supported by the *Security engineering for lifelong evolvable systems* (SecureChange) project, FP7-EC-GA-231101.

## References

1. ISO 31000, Risk Management: Principles and Guidelines, International Organization for Standardization (2009)
2. Alberts, C.J., Davey, J.: OCTAVE criteria version 2.0. Technical report CMU/SEL-2001-TR-016. Carnegie Mellon University (2004)
3. Barber, B., Davey, J.: The use of the CCTA risk analysis and management methodology CRAMM in health information systems. In: 7th International Congress on Medical Informatics, MEDINFO 1992, pp. 1589–1593 (1992)
4. CRAMM - The total information security toolkit, <http://www.cramm.com/> (accessed March 2, 2011)
5. Robinson, R.M., Anderson, K., Browning, B., Francis, G., Kanga, M., Millen, T., Milman, C.: Risk and Reliability. An Introductory Text, 5th edn. R2A (2001)
6. IEC 61025, Fault Tree Analysis (FTA), International Electrotechnical Commission (1990)

7. IEC 60300-3-9, Dependability management - Part 3: Application guide - Section 9: Risk analysis of technological systems - Event Tree Analysis (ETA), International Electrotechnical Commission (1995)
8. Schneier, B.: Attack trees: Modeling security threats. *Dr. Dobb's J.* 24(12), 21–29 (1999)
9. Nielsen, D.S.: The cause/consequence diagram method as basis for quantitative accident analysis. Technical report RISO-M-1374, Danish Atomic Energy Commission (1971)
10. Ben-Gal, I.: Bayesian networks. In: Ruggeri, F., Kenett, R.S., Faltin, F.W. (eds.) *Encyclopedia of Statistics in Quality and Reliability*. John Wiley & Sons, Chichester (2007)
11. Lund, M.S., Solhaug, B., Stølen, K.: *Model-Driven Risk Analysis: The CORAS Approach*. Springer, Heidelberg (2011)
12. Brændeland, G., Refsdal, A., Stølen, K.: Modular analysis and modelling of risk scenarios with dependencies. *Journal of Systems and Software* 83(10), 1995–2013 (2010)
13. Lund, M.S., Solhaug, B., Stølen, K., Innerhofer-Oberperfler, F., Felici, M., Meduri, V., Tedeschi, A.: *Assessment Method, SecureChange deliverable* (2011)
14. OMG Unified Modeling Language, Superstructure, version 2.2, Object Management Group (2009)
15. Perrow, C.: *Normal accidents: living with high-risk technologies*. Princeton University Press, Princeton (1999)
16. Edwards, E.: Man and machine: Systems for safety. In: *Proceedings of British Airline Pilots Associations Technical Symposium*, British Airline Pilots Associations, pp. 21–36 (1972)
17. Reason, J.: *Managing the Risks of Organizational Accidents*, Ashgate (1997)
18. Pasquini, A., Pozzi, S.: Evaluation of air traffic management procedures - safety assessment in an experimental environment. *Reliability Engineering & System Safety* 89(1), 105–117 (2005)
19. Pasquini, A., Pozzi, S., Save, L.: A critical view of severity classification in risk assessment methods. *Reliability Engineering & System Safety* 96(1), 53–63 (2011)
20. EUROCONTROL. *Safety Nets - Ensuring Effectiveness* (2009)
21. EUROCONTROL safety regulatory requirements (ESARR), ESARR 4 - risk assessment and mitigation in ATM, Edition 1.0 (2001)
22. EUROCONTROL safety regulatory requirements (ESARR), ESARR 6 - Software in ATM Systems, Edition 1.0 (2003)
23. EUROCONTROL, *Baseline Integrated Risk Picture for Air Traffic Management in Europe*, EEC Note No. 15/05 (2005)
24. Brooker, P.: The Überlingen accident: Macro-level safety lessons. *Safety Science* 46(10), 1483–1508 (2008)
25. Felici, M.: Evolutionary safety analysis: Motivations from the air traffic management domain. In: Winther, R., Gran, B.A., Dahll, G. (eds.) *SAFECOMP 2005*. LNCS, vol. 3688, pp. 208–221. Springer, Heidelberg (2005)

# Computer-Aided PHA, FTA and FMEA for Automotive Embedded Systems

Roland Mader<sup>1,2</sup>, Eric Armengaud<sup>1,3</sup>, Andrea Leitner<sup>2</sup>,  
Christian Kreiner<sup>2</sup>, Quentin Bourrouilh<sup>1</sup>, Gerhard Griebnig<sup>1,2</sup>,  
Christian Steger<sup>2</sup>, and Reinhold Weiß<sup>2</sup>

<sup>1</sup> AVL List GmbH

{roland.mader,quentin.bourrouilh,gerhard.griessnig}@avl.com

<sup>2</sup> Institute for Technical Informatics, Graz University of Technology

{andrea.leitner,christian.kreiner,stege,rweiss}@tugraz.at

<sup>3</sup> Virtual Vehicle Competence Center (ViF)

eric.armengaud@v2c2.at

**Abstract.** The shift of the automotive industry towards powertrain electrification introduces new automotive sensors, actuators and functions that lead to an increasing complexity of automotive embedded systems. The safety-criticality of these systems demands the application of analysis techniques such as PHA (Preliminary Hazard Analysis), FTA (Fault Tree Analysis) and FMEA (Failure Modes and Effects Analysis) in the development process. The early application of PHA allows to identify and classify hazards and to define top-level safety requirements. Building on this, the application of FTA and FMEA supports the verification of a system architecture defining an embedded system together with connected sensors and controlled actuators. This work presents a modeling framework with automated analysis and synthesis capabilities that supports a safety engineering workflow using the domain-specific language EAST-ADL. The contribution of this work is (1) the definition of properties that indicate the correct application of the workflow using the language. The properties and a model integrating the work products of the workflow are used for the automated detection of errors (property checker) and the automated suggestion and application of corrective measures (model corrector). Furthermore, (2) fault trees and a FMEA table can be automatically synthesized from the same model. The applicability of this computer-aided and tightly integrated approach is evaluated using the case study of a hybrid electric vehicle development.

## 1 Introduction

Nowadays automotive embedded systems incorporate up to 70 microcontrollers that communicate via bus systems, gather sensor data and command actuators of the vehicle. This complexity still increases. One of the reasons is the shift of the automotive industry towards powertrain electrification that goes along with the introduction of new sensors, actuators and functions. The automotive embedded system is responsible for the management of the components (e.g.

high voltage battery, electric motor) that can be found in electrified vehicles and components (e.g. transmission, engine) which are parts of traditional vehicles as well. It is obvious that the correct and safe operation of an electrified vehicle depends on the correct operation of its embedded system.

Due to the safety-criticality of automotive embedded systems, they are developed according to rigorous development processes such as defined by ISO 26262, the functional safety standard for the automotive domain. These development processes incorporate the application of analysis techniques. Among the applied techniques are the following:

- **PHA (Preliminary Hazard Analysis):** PHA [12] is an analysis technique that is qualitatively applied early in the development process by a team of people with a wide variety of expert knowledge and skills. The purpose of PHA is the identification, classification and assessment of potential hazards of a newly developed vehicle, caused by failures. The early knowledge about these hazards allows to define top-level safety requirements, even if less detailed and quantitative information about the vehicle is available.
- **FTA (Fault Tree Analysis):** FTA [12] belongs to the group of deductive analysis techniques. FTA starts with the identified hazards and tracks them back to possible faults that can lead to the occurrence of the top faults. Relationships between effect and cause are defined using logical operators that combine the effects of events. This analysis technique can be applied to verify a system architecture defining an embedded system together with connected sensors and controlled actuators.
- **FMEA (Failure Modes and Effects Analysis):** FMEA [12] belongs to the group of inductive analysis techniques. Individual failures of system components are considered and their causes (e.g. fault of a component) are identified. Then the effects on the complete system in terms of hazards are determined. This analysis technique can be applied to verify a system architecture as well.

This work presents a modeling framework with automated analysis and synthesis capabilities. This modeling framework supports an ISO 26262-compatible automotive safety engineering workflow. Results are annotated using the domain-specific language EAST-ADL [1]. The contribution of this work is (1) the definition of properties that indicate the correct application of the workflow using this language. The properties and a model integrating the work products of the workflow are used for the automated detection of errors (property checker) and the automated suggestion and application of corrective measures (model corrector). Furthermore, (2) fault trees and a FMEA table can be automatically generated from the model allowing the qualitative application of FTA and FMEA. The fault trees and the FMEA table are consistent to the PHA results. Minimum cut sets can be automatically extracted from the synthesized fault trees.

The remainder of this work is organized as follows. Section 2 reviews related work. Section 3 describes the ISO 26262-compatible safety engineering workflow. Section 4 describes how the workflow can be supported by the property checker.

Section 5 describes how the model corrector can be used to correct errors and how fault trees and a FMEA table can be automatically generated. Section 6 describes the experimental evaluation of the approach using the case study of a hybrid electric vehicle development. Finally Section 7 concludes this work.

## 2 Related Work

Approaches that aim on supporting safety engineering by fault tree generation and/or FMEA generation are reviewed in this section. An approach that combines system architecture modeling and FTA is described in [14]. The approach allows continuous assessment of an evolving system design. A system model is input to HAZOP (Hazard and Operational Studies). Each component of the system model is analyzed and component failure modes are determined. The HAZOP result is a model that defines failure modes that can be observed at the component outputs as results of internal component malfunctions as well as deviating component inputs. In [13] an extension of [14] is presented that allows FMEA table generation. In [2] the extended approach is integrated with an EAST-ADL modeling tool using a model transformation technique. This allows synthesis of fault trees and FMEA tables from EAST-ADL models.

In [4] tool support for automated FMEA generation is presented. Input to the presented method is a component model of a system including so called safety interfaces that can be automatically generated. Safety interfaces can be seen as formal descriptions of the components in terms of failures affecting the components. From the safety interface descriptions cFMEAs (Component Failure Modes and Effects Analysis) can be created for each component. Subsequently the cFMEAs are input to the generation of a system-level FMEA.

A methodology that combines safety analyses and a component-oriented, model-based software engineering approach is described in [3]. The authors aim on supporting safety analyses in the earlier stages of development. A hierarchical model for component-based software engineering is available. The model allows to define a failure specification and a failure realization as well as a functional specification and a functional realization for each software component. Fault trees can be generated from the component model.

In [10] a computer-aided approach to fault tree generation is described. The approach requires the creation of a model of the system under investigation. This model describes system structure, system behavior as well as the flows of information and energy through the system. Moreover top events are defined for system parameters such as component inputs or component outputs. This model is input to a trace-back algorithm that generates a fault tree.

The authors of [11] integrate architectural modeling languages with safety analysis languages to improve consistency. When a safety-critical software architecture is developed an initial architecture is proposed. This architecture is annotated and enriched with safety-relevant information. Safety analysis of the architecture is carried out. Results influence the software architecture. This design and analysis process is cyclic. A meta model for component-based, safety-aware



architectures (SAA) is available allowing to complement architectural descriptions with safety-relevant information such as safety objectives and mitigation means. Meta models for FTA and FMECA (Failure Modes, Effects and Criticality Analysis) are proposed. A tool implementation is presented that allows the generation of FTA models and FMECA models from a SAA model.

Each of the reviewed approaches uses a system model describing the system components complemented with safety-relevant information (typically about faults and failures and their propagation). This underlying model is used by all approaches as input to fault tree generation and/or for FMEA table generation, supporting the application of FTA and/or FMEA. In none of these approaches the application of the workflow for creation of the underlying model is aided by automated checking or model correction. Our approach supports this, supporting fault tree generation and FMEA table generation and furthermore elaboration of the underlying model that integrates the work products of the presented workflow. This strongly supports coping with the complexity imposed by the embedded system of an electrified vehicle.

### 3 Safety Engineering Workflow

We present an ISO 26262-compatible, automotive safety engineering workflow that is based on the workflows described in [15] and [9]. The workflow can be subdivided into multiple phases. Iterations between phases are possible. The presented workflow is illustrated in Figure 1. In the course of the workflow an EAST-ADL model is annotated, systematically enhanced and refined using a modeling framework. The elaborated model integrates the work products (e.g. analysis results, requirements, system architecture) of the workflow phases. EAST-ADL is a domain-specific language and tailored to the needs of the automotive domain. It is *diagrammatic* [5] such as UML. It consists of syntactic elements such as boxes, ovals, lines or arrows. Its *abstract syntax* is defined by its meta model and its *semantic domain* and *semantic mapping* are defined using natural language [5]. The workflow phases are thereafter described:

1. **Definition of the Analysis Subject:** First information about the vehicle under development is collected and modeled. Functions of the vehicle (e.g. motoring or recuperative braking) are defined. Requirements to these functions are determined and allocated (e.g. conditions for activation or deactivation). In addition relevant modes (e.g. drive, creep or acceleration) are identified for each function and associated with the requirements.
2. **Identification of Hazards and Hazardous Events:** Based on the definition of the analysis subject, PHA (for more details see also [9]) is carried out. Possible malfunctions are identified. Hazards are derived for each malfunction (e.g. unintended acceleration of the vehicle). Thereafter operational situations such as traffic situations (e.g. oncoming traffic on a highway in a curve) and maintenance situations (e.g. vehicle at lifting ramp) are defined. Moreover use cases describing the behavior (e.g. overtaking or

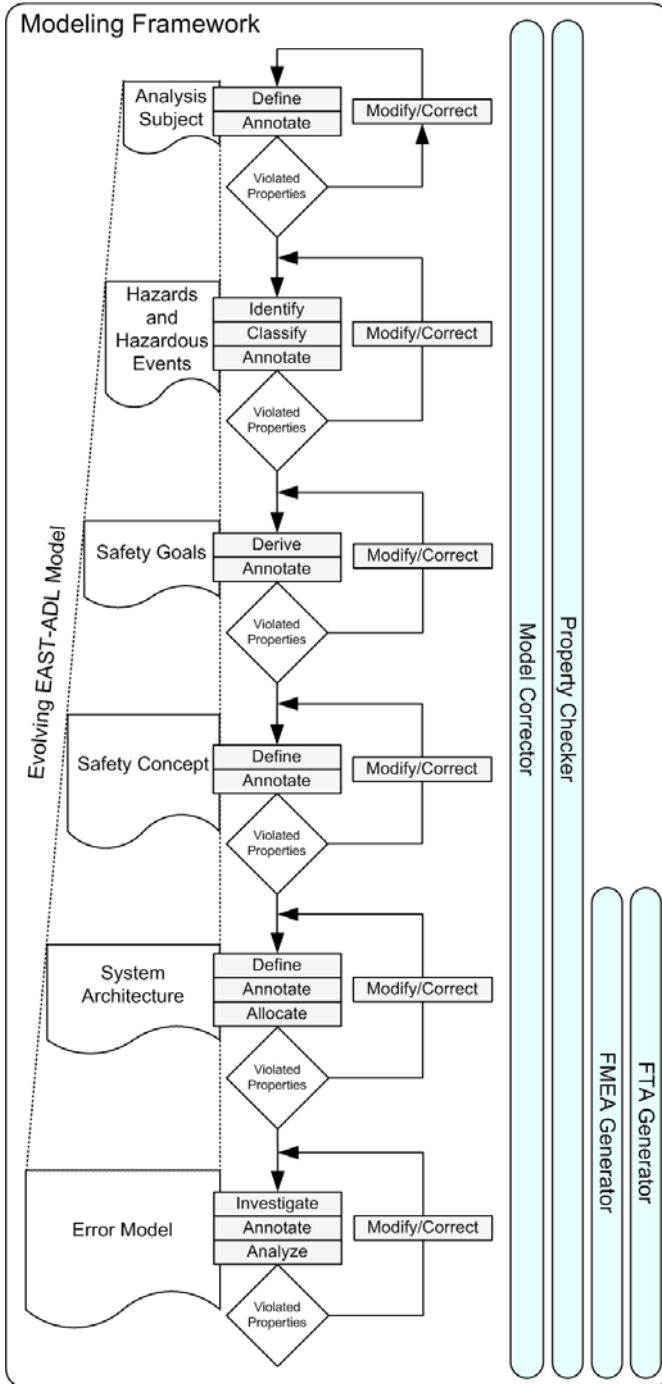


Fig. 1. Computer-Aided Safety Engineering Workflow

changing oil) of the related actors (e.g. driver or mechanic) are described. Hazardous events are determined for relevant combinations of hazards, use cases and operational situations. Moreover relevant modes are identified for each hazardous event. The criticality of each hazardous event is assessed in terms of its controllability, severity and exposure and an ASIL (Automotive Safety Integrity Level) [7] is determined.

3. **Derivation of Safety Goals:** For each hazardous event that has an ASIL assigned (ASIL A, ASIL B, ASIL C or ASIL D), a safety goal is derived and associated. Furthermore, a safe state is defined (e.g. switch open) for each safety goal. Alternatively a safe mode (e.g. limp home mode) is determined. The determined safety goals are top-level safety requirements.
4. **Definition of Safety Concept:** The safety concept is derived from the safety goals. This safety concept consists of functional and technical safety requirements to the automotive embedded system, connected sensors and controlled actuators. Traces are created between safety goals, functional safety requirements and technical safety requirements.
5. **Definition of System Architecture:** The system architecture is defined in terms of the embedded system, connected sensors and controlled actuators. Moreover the parts of the environment are modeled that interact with the sensors and actuators. Thereafter the functional and technical safety requirements are allocated to the components of the system architecture. Furthermore functions are allocated to the components of the system architecture.
6. **Investigation and Annotation of Faults and Failures:** Information flows and energy flows through the embedded system, connected sensors, controlled actuators and their environment are investigated. Possible faults and failures are estimated and their propagation is analyzed and annotated. Moreover it is investigated and annotated how the failures lead to the malfunctions that were identified during PHA. Thereafter FTA and FMEA are applied.

After the completion of these working steps, requirements to software and hardware are derived from the safety concept. Software and hardware are fully specified, implemented, integrated, verified and validated. However these working steps are beyond the scope of this work.

Due to the complexity of contemporary vehicles, the application of the workflow is cumbersome and error-prone. Therefore we propose to aid the safety engineering workflow defined above by automated property checking (property checker), automated model correction (model corrector), automated fault tree synthesis and automated FMEA table synthesis (see Section 4 and Section 5). This allows to early identify erroneously applied working steps and enables the automated suggestion and application of corrective measures. Moreover it is not necessary to construct fault trees and FMEA tables manually. Instead, they are consistently generated from the EAST-ADL model. While property checker and model corrector aid the entire workflow, FTA generator and FMEA generator are especially useful for the verification of the system architecture defining an

embedded system together with connected sensors and controlled actuators. The automated analysis and synthesis capabilities of the modeling framework provide guidance and strongly support the application of the workflow and the creation of a complete and consistent set of work products.

## 4 Computer-Aided Checking

Properties are defined that indicate the correct application of the activities of the safety engineering workflow (see Section 3). A property checker is part of the modeling framework (see Figure 1) and continuously checks the evolving EAST-ADL model and presents violating modeling elements to the user. If no properties are violated, the EAST-ADL model indicates the correct application of the workflow. If the property checker identifies violated properties, the erroneous application of the workflow is unveiled. The property checker does not only allow the early identification of errors, it is also a valuable guide, while the workflow is applied. In addition to properties for the earlier phases of the safety engineering workflow (see 9), properties for the later phases are presented in Table 1.

Assume  $M$  is an EAST-ADL model,  $M_{MM}$  is the EAST-ADL meta model and  $P$  is the set of properties an EAST-ADL model is expected to hold. Assume  $e$  is a modeling element of the EAST-ADL model,  $t$  is a type defined by the EAST-ADL meta model and  $p$  is a property (Expression 1).

$$e \in M, t \in M_{MM}, p \in P \quad (1)$$

Moreover,  $I(e, t)$  pertains, if  $e$  is of type  $t$ ,  $D(t, p)$  pertains, if  $p$  is defined for  $t$  and  $H(e, p)$  pertains if  $p$  holds for  $e$ . If  $M$  indicates the correct application of the workflow, Expression 2 is valid. In this case no modeling elements violate properties.

$$\neg \exists e \neg \exists t \neg \exists p (I(e, t) \wedge D(t, p) \wedge \neg H(e, p)) \quad (2)$$

If a model  $M$  shows the erroneous application of the workflow, Expression 3 is valid. In this case at least one modeling element violates a property.

$$\exists e \exists t \exists p (I(e, t) \wedge D(t, p) \wedge \neg H(e, p)) \quad (3)$$

## 5 Automated Synthesis

### 5.1 Model Correction

Correction rules are defined that impose possible solutions to problems indicated by the property checker (see also Section 4). A model corrector is part of the modeling framework (see Figure 1). Possible solutions are suggested on demand by the model corrector, based on the evolving EAST-ADL model and the correction rules. A user can decide to accept a suggestion of the model corrector

**Table 1.** Properties of the EAST-ADL model are automatically checked

ID	Meta Class	Property Definition
28	SafetyGoal	At least one safety requirement is derived
29	QualityRequirement	Traceable to a safety requirement or a SafetyGoal, if it is a safety requirement
30	QualityRequirement	Is allocated to at least one AnalysisFunctionPrototype, if it is a safety requirement
32	Environment	An environmentModel is defined
34	AnalysisLevel	A functionalAnalysisArchitecture has been defined
39	FunctionFlowPort	Has at most one FunctionConnector to a FunctionFlowPort of type out or inout associated, if type in
40	FunctionPort	A type is defined
40a	FunctionPort	Connected by at least one FunctionConnector
40c	FunctionPort	A complementary description has been defined
40b	FunctionConnector	Connector is connected to two FunctionPorts
41	AnalysisFunctionPrototype	A type is defined
42	AnalysisFunctionPrototype	Has a complementary description
42a	AnalysisFunctionPrototype	An ErrorModelPrototype is defined for every AnalysisFunctionPrototype
37	AnalysisFunctionType	At least one FunctionPort has been defined
42b	AnalysisFunctionType	An ErrorModelType is defined for every AnalysisFunctionType
48	FaultInPort	Has only one FaultFailurePropagationLink to a FailureOutPort associated
51	FaultFailurePort	A functionTarget_path is defined
51a	FaultFailurePort	A type is defined
52a	FailureOutPort	Has a complementary description
53	ErrorModelPrototype	A type is defined
54	ErrorModelPrototype	A functionTarget is defined
55	ErrorBehavior	An externalFailure is defined
56	ErrorBehavior	The defined failureLogic is legal and recognized
57	ErrorBehavior	An owner is defined
58	InternalFaultPrototype	Has a complementary description
59	InternalFaultPrototype	Is owned by at least one ErrorBehavior
60	VehicleFeature	Every function is allocated to at least one AnalysisFunctionPrototype
62	FeatureFlaw	Is mapped onto a FailureOutPort
63	EABoolean	A note is defined
64	RangeableDatatype	A note is defined
65	EAFloat	The lower threshold is defined
66	EAFloat	The upper threshold is defined

**Table 2.** Possible solutions to problems that are automatically suggested

ID	Meta Class	Suggested Solution
28	SafetyGoal	Creation and association of safety requirement
28	SafetyGoal	Associate one of the untraceable safety requirements
29	QualityRequirement	Associate to existing SafetyGoal, if it is a safety requirement
29	QualityRequirement	Associate to existing safety requirement, if it is a safety requirement
30	QualityRequirement	Allocation to existing AnalysisFunctionPrototype, if it is a safety requirement
32	Environment	Creation and association of AnalysisFunctionPrototype
34	AnalysisLevel	Creation and association of AnalysisFunctionPrototype
40	FunctionPort	Association of existing EAInteger
40	FunctionPort	Association of existing EAFloat
40	FunctionPort	Association of existing EABoolean
40c	FunctionPort	Creation and association of Comment
40b	FunctionConnector	Remove connector
41	AnalysisFunctionPrototype	Association of existing AnalysisFunctionType
42	AnalysisFunctionPrototype	Creation and association of Comment
42a	AnalysisFunctionPrototype	Association of existing ErrorModelPrototype
37	AnalysisFunctionType	Creation and association of FunctionPort
42b	AnalysisFunctionType	Creation and association of ErrorModelType
42b	AnalysisFunctionType	Association of existing, unassociated ErrorModelType
51	FaultFailurePort	Association of existing AnalysisFunctionPrototype
51a	FaultFailurePort	Association of existing EAInteger
51a	FaultFailurePort	Association of existing EAFloat
51a	FaultFailurePort	Association of existing EABoolean
52a	FailureOutPort	Creation and association of Comment
53	ErrorModelPrototype	Creation and association of ErrorModelType
53	ErrorModelPrototype	Association of existing ErrorModelType
54	ErrorModelPrototype	Association of existing AnalysisFunctionPrototype
55	ErrorBehavior	Association of existing, unassociated FailureOutPort
56	ErrorBehavior	Change to and (type OTHER)
56	ErrorBehavior	Change to or (type OTHER)
57	ErrorBehavior	Creation and association of ErrorModelType
57	ErrorBehavior	Association of existing ErrorModelType without ErrorBehavior
58	InternalFaultPrototype	Creation and association of Comment
59	InternalFaultPrototype	Association of existing ErrorBehavior
60	VehicleFeature	Allocation to existing AnalysisFunctionPrototype
62	FeatureFlaw	Allocation to existing FailureOutPort

or to solve the problem in another way. If a suggested, possible solution is accepted, the EAST-ADL model is automatically modified and corrected making a manual modification superfluous. If a suggestion is rejected the EAST-ADL model remains unchanged. In addition to correction rules for the earlier phases of the safety engineering workflow (see [9]), correction rules for the later phases are presented in Table 2.

Assume  $M$  is an EAST-ADL model,  $M_{MM}$  is the EAST-ADL meta model,  $P$  is the defined set of properties and  $S$  is the set of defined suggestions. Assume  $e_1$  is a modeling element of the EAST-ADL model,  $t_1$  is a type defined by the meta model,  $p_1$  is a property and  $s_1$  is a suggestion (Expression 4).

$$e_1 \in M, t_1 \in M_{MM}, p_1 \in P, s_1 \in S \quad (4)$$

Assume that before an automated model correction is carried out (precondition),  $e_1$  is of type  $t_1$  and violates  $p_1$  that is defined for  $t_1$  (Expression 5).

$$I(e_1, t_1) \wedge D(t_1, p_1) \wedge \neg H(e_1, p_1) \quad (5)$$

If the user accepts suggestion  $s_1$ , the EAST-ADL model  $M$  is automatically corrected and transformed to EAST-ADL model  $M'$  by function  $\gamma$  depending on  $M$ ,  $e_1$ ,  $t_1$ ,  $p_1$  and  $s_1$  (Expression 6).

$$\gamma(M, e_1, t_1, p_1, s_1) \rightarrow M' \quad (6)$$

After the modification (postcondition)  $e_1$  is an element of  $M'$ ,  $e_1$  is still of type  $t_1$  and does not violate  $p_1$  any more (Expression 7).

$$e_1 \in M', I(e_1, t_1) \wedge D(t_1, p_1) \wedge H(e_1, p_1) \quad (7)$$

## 5.2 Fault Tree and FMEA Table Synthesis

The modeling framework (see Figure 1) contains a FTA generator and a FMEA generator. The EAST-ADL model that is created in the course of the safety engineering workflow (see Section 3) is input to them. The FTA generator is able to synthesize fault trees (see Figure 2). The fault trees show, how each safety goal can be violated by the faults and failures of the embedded system, connected sensors or controlled actuators. The FMEA generator is able to synthesize a FMEA table (see Figure 3). The FMEA table shows failure modes of the components, causative faults for these failure modes and effects of these failure modes in terms of violated safety goals.

The FTA generator considers the recommendations of IEC 61025 [6]. Therefore the shapes of the symbols of the fault trees are adapted to the shapes of the symbols recommended by IEC 61025. Basic events (faults, failures) are represented by circles, complex events (faults, failures, malfunctions, hazards, hazardous events, violated safety goals) are represented by rectangles and gates (and, or) are represented by the corresponding logic symbols. The FTA generator uses the identified safety goals as top events of the generated fault trees (one





Component	Failure Mode	Possible Causative Faults	Violated Safety Goal
HCU	UnintendedDemandOfPosi...	EESystem::UntrulyPresse... HCU::HCUMicrocontrollerF... HCU::HCURAMFault AccelerationPedalSensor::... HCU::HCUPeripheryFault ECU::ECUPeripheryFault ECU::ECUMicrocontrollerF... ECU::ECURAMFault	AvoidUnintendedPositiveT...
	UnintendedDemandOfNeg...	HCU::HCUMicrocontrollerF... HCU::HCURAMFault EESystem::UntrulyPresse... HCU::HCUPeripheryFault ECU::ECUPeripheryFault ECU::ECUMicrocontrollerF... BrakePedalSensor::Brake... ECU::ECURAMFault	AvoidUnintendedNegative...
	UnintendedDemandOfNeg...	HCU::HCUMicrocontrollerF... HCU::HCURAMFault BMU::BMUPeripheryFault BMU::BMUMicrocontrollerF... HCU::HCUPeripheryFault Battery::BatteryVoltage5... BMU::BMURAMFault	AvoidBatteryOvercharging

**Fig. 3.** A FMEA table can be synthesized from the EAST-ADL model

$$\rho(S) \rightarrow \Upsilon \quad (9)$$

The FMEA generator creates a FMEA table containing four columns denoting the names of the components (*Component*), the component failure modes leading to the violation of safety goals (*Failure Mode*), faults that potentially cause the component failure modes (*Possible Causative Faults*) and the violated safety goals (*Violated Safety Goal*). The generated FMEA table is consistent to the fault trees and the earlier elaborated PHA results, because FTA generator and FMEA generator use the same model  $S$  as input.

Given that Expression 2 holds for all  $e \in S$ , FMEA generator  $\alpha(S)$  can generate a graphical FMEA table  $\Xi$  that allows to examine how component failures can lead to the violation of safety goals (see Expression 10).

$$\alpha(S) \rightarrow \Xi \quad (10)$$

## 6 Experimental Evaluation

A plugin for the open source tool Papyrus [8] was created that allows property checking, model correction, fault tree generation and FMEA table generation such as described in Section 4 and Section 5. Thereafter the approach was experimentally evaluated using the case study of a hybrid electric vehicle development. This type of vehicle contains an additional electric motor that supplements the internal combustion engine providing substitutive or additive torque. This electric motor is controlled by the automotive embedded system. The safety engineering workflow such as defined in Section 3 was carried out for a part of a

hybrid electric vehicle powertrain being aided by the property checker and the model corrector.

Although the safety engineering workflow was carried out only for a part of the hybrid electric vehicle powertrain, the resulting EAST-ADL model contains 457 interconnected modeling elements. Each of them contains numerous attributes. The property checker identifying erroneously applied activities (see Section 4) and the model corrector suggesting and applying model corrections (see Section 5.1) strongly supported the application of the workflow and allowed coping with the complexity. Illustrations of property checker and model corrector can be found in 9.

During PHA the hybrid electric vehicle was identified to be safety-critical, because its failures can cause malfunctions such as battery overcharging (*BatteryOvercharging*). This malfunction can lead to hazards such as fire or explosion of the battery (*FireExplosion*). Fire or explosion of the battery during vehicle operation imposes a hazardous event (*FireExplosionDuringCityTraffic*). Therefore the safety goal *AvoidBatteryOvercharging* was defined to control or mitigate the corresponding hazard.

In later phases of the workflow, a part of the system architecture including networked ECUs (electronic control unit), connected sensors and controlled actuators was defined. Furthermore the relevant parts of the interacting environment were modeled. The propagation of faults and failures was estimated and annotated. The failure *UnintendedNegativeTorque2* of the component *EMotor* was identified to be causative for the malfunction *BatteryOvercharging*. This failure can occur due to a failure of the E-motor or faults propagated from a sensor and networked ECUs such as the BMU (Battery Management Unit).

Fault trees and a FMEA table were synthesized from the annotated model (see Section 5.2). Figure 2 depicts a fault tree that shows the relations between safety goal *AvoidBatteryOvercharging*, hazardous event *FireExplosionDuringCityTraffic*, hazard *FireExplosion*, malfunction *BatteryOvercharging* as well as the causative faults and failures of the components. The extracted minimum cut sets that can cause the violation of the safety goal are also depicted.

Figure 3 shows a part of the synthesized FMEA table. The table shows that a failure mode of the HCU (Hybrid Control Unit) can lead to the violation of the safety goal *AvoidBatteryOvercharging*. Moreover possible causative faults are listed.

## 7 Conclusion

This work presents a modeling framework with analysis and synthesis capabilities. This modeling framework supports a safety engineering workflow. In the course of the workflow a model is annotated using the domain-specific language EAST-ADL. This model integrates the work products of the workflow phases. The modeling framework contains a property checker that allows to unveil the incorrect application of the workflow and a model corrector that suggests and automatically performs corrections of the evolving model. Moreover fault trees and

a FMEA table can be automatically synthesized allowing the application of qualitative FTA and FMEA. This tightly integrated approach ensures consistency of PHA results, fault trees and FMEA table. The approach was evaluated using the case study of a hybrid electric vehicle development. While the analysis and synthesis capabilities of the modeling framework did not replace the intellectual process of applying the workflow, they strongly supported its application.

**Acknowledgment.** The authors wish to thank the "COMET K2 Forschungsförderungs-Programm" of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economics and Labour (BMWA), Österreichische Forschungsförderungsgesellschaft mbH (FFG), Das Land Steiermark and Steirische Wirtschaftsförderung (SFG) for their financial support. Additionally we would like to thank the supporting company and project partner AVL List GmbH as well as Graz University of Technology. Further information about the MEPAS project can be found at <http://www.v2c2.at/mepas>.

## References

1. ATESS2 Project Consortium: EAST-ADL Domain Model Specification, version 2.1, Release Candidate 3 (2010)
2. Biehl, M., DeJui, C., Törngren, M.: Integrating Safety Analysis into the Model-based Development Toolchain of Automotive Embedded Systems. In: Proc. of the Conference on Languages, Compilers and Tools for Embedded Systems, pp. 125–131 (2010)
3. Domis, D., Trapp, M.: Integrating Safety Analyses and Component-Based Design. In: Proc. of the 27th International Conference on Computer Safety, Reliability and Security, pp. 58–71 (September 2008)
4. Elmqvist, J., Nadjm-Tehrani, S.: Tool Support for Incremental Failure Mode and Effects Analysis of Component-Based Systems. In: Proc. of the Design, Automation and Test in Europe Conference and Exhibition (DATE 2008), pp. 921–927 (April 2008)
5. Harel, D., Rumpe, B.: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Transactions on Computers 37, 64–72 (2004)
6. International Electrotechnical Commission: IEC 61025 - Ed. 2.0 Fault tree analysis (FTA) (2006)
7. International Organization for Standardization: ISO/DIS 26262-3 Road vehicles - Functional safety - Part 3: Concept phase (2009)
8. Lanusse, A., Tanguy, Y., Espinoza, H., Mraidha, C., Gerard, S., Tessier, P., Schneckeburger, R., Dubois, H., Terrier, F.: Papyrus UML: an open source toolset for MDA. In: Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009), pp. 1–4 (June 2009)
9. Mader, R., Griebnig, G., Leitner, A., Kreiner, C., Bourrouilh, Q., Armengaud, E., Steger, C., Weiß, R.: A Computer-Aided Approach to Preliminary Hazard Analysis for Automotive Embedded Systems. In: Proc. of the IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS), pp. 169–178 (2011)

10. Majdara, A., Wakabayashi, T.: A New Approach for Computer-Aided Fault Tree Generation. In: Proc. of the 3rd Annual IEEE Systems Conference, pp. 308–312 (2009)
11. de Miguel, M., Briones, J., Silva, J., Alonso, A.: Integration of safety analysis in model-driven software development. *IET Software* 2, 260–280 (2008)
12. Leveson, N.G.: *Safeware: system safety and computers*. Addison-Wesley Publishing Company, Reading (1995)
13. Papadopoulos, Y., Grante, C.: Evolving car designs using model-based automated safety analysis and optimisation techniques. *The Journal of Systems and Software* 76, 77–89 (2004)
14. Papadopoulos, Y., Maruhn, M.: Model-Based Synthesis of Fault Trees from Matlab - Simulink models. In: Proc. of the International Conference on Dependable Systems and Networks (DSN 2001), pp. 77–82 (July 2001)
15. Sandberg, A., Chen, D.J., Lönn, H., Johansson, R., Feng, L., Törngren, M., Torchiaro, S., Kolagari, R.T., Abele, A.: Model-Based Safety Engineering of Interdependent Functions in Automotive Vehicles Using EAST-ADL2. In: Proc. of the 29th International Conference on Computer Safety, Reliability and Security, pp. 332–346 (September 2010)

# A Statistical Anomaly-Based Algorithm for On-line Fault Detection in Complex Software Critical Systems

Antonio Bovenzi<sup>1</sup>, Francesco Brancati<sup>2</sup>, Stefano Russo<sup>1</sup>, and Andrea Bondavalli<sup>2</sup>

<sup>1</sup> Dipartimento di Informatica e Sistemistica (DIS),  
Università degli Studi di Napoli “Federico II”, Napoli, Italy  
{antonio.bovenzi,sterusso}@unina.it

<sup>2</sup> Dipartimento di Sistemi e Informatica (DSI), Università degli Studi di Firenze, Italy  
{Francesco.brancati,bondavalli}@unifi.it

**Abstract.** The next generation of software systems in Large-scale Complex Critical Infrastructures (LCCIs) requires efficient runtime management and reconfiguration strategies, and the ability to take decisions on the basis of current and past behavior of the system. In this paper we propose an anomaly-based approach for the detection of online faults, which is able to (i) cope with highly variable and non-stationary environment and to (ii) work without any initial training phase. The novel algorithm is based on Statistical Predictor and Safety Margin (SPS), which was initially developed to estimate the uncertainty in time synchronization mechanisms.

The SPS anomaly detection algorithm has been experimented on a case study from the Air Traffic Management (ATM) domain. Results have been compared with an algorithm, which adopts static thresholds, in the same scenarios [5]. Experimental results show limitations of static thresholds in highly variable scenarios, and the ability of SPS to fulfill the expectations.

**Keywords:** Anomaly detection, SPS, on-line software fault diagnosis.

## 1 Introduction

Large scale Complex Critical Infrastructures (LCCI), such as transport infrastructures (e.g., the novel European Air Traffic Management federated system<sup>1</sup>) or power grids, play a key role into several fundamental human activities. It is easy to think about their economic and social impact: the consequences of an outage can be catastrophic in terms of efficiency, economical losses, consumer dissatisfaction, and even indirect harm to people.

LCCIs are the result of the integration of heterogeneous stand-alone subsystems and their scale is strongly increasing, due to deregulation and the development of “mixed market infrastructures” [10] and technological improvement. Such interconnection requires not only designing a way to interconnect heterogeneous systems, but also imposes that legacy systems have to operate beyond the original design parameters [9].

---

<sup>1</sup> <http://www.sesarju.eu/>

These systems rely on the efficacy of services, such as system management, replication, load balancing and group communication, which require suitable algorithms able to take runtime decisions on the basis of actual and past behavior of the system. All these characteristics exacerbate the complexity of the infrastructure, making crucial the designing of intelligent on-line monitoring and detection mechanisms to infer (i) if the whole system is performing well and, if not, (ii) how to face with possible failures.

Huge amount of data, coming from different probes spread over the system, need to be analyzed in order to reveal that something is not working properly. In other words, it must be possible to identify the case where anomalies occurred in the system. With the term anomaly we refer to changes in the variable characterizing the behavior of the system caused by specific and non-random factors [11], e.g., overload, the activation of faults, malicious attacks, etc.

It is an interesting open issue to detect relevant anomalies in systems that exhibit variable and non-stationary behavior, and may be affected by perturbations. Moreover, the detection problem is exacerbated when the anomaly detector has to support timely decisions based on online instead of offline analysis.

Detection systems usually assume worst-case thresholds to allow distinguishing between nominal behaviors and anomalies [5][6]. However these thresholds are typically tuned in a preliminary training phase and cannot fit all dynamically changing situations in which the system could evolve. For instance, these thresholds may depend on the application requirements, on system operational parameters and on the current environment, and usually there are not a-priori fixed for the entire system, and for all the system life-cycle; therefore, detectors can take advantage from the possibility to adapt the expected thresholds online, e.g., because of operational conditions modifications.

This work proposes an anomaly-based approach for software fault detection in complex critical system, exploiting statistical analysis on data gathered at the Operating System level. The proposed detector is able (i) to cope with variable and non-stationary behavior, (ii) to perform an online (instead of offline) analysis and (iii) to work without any initial training phase.

The detector receives data coming from a monitoring infrastructure, described in [5], and it exploits the Operating System tracking mechanisms to collect different kinds of information (e.g., syscall errors, signals, scheduling time of processes), which reveal to be useful for detection.

The statistical analysis is performed by means of a recent algorithm, i.e., Statistical Predictor and Safety Margin (SPS), which was initially designed to estimate the synchronization uncertainty of a software clock [7]. Our intuition was that such algorithm could be exploited to detect anomalies, due to software faults activation, in high variable context. Such intuition is confirmed by our experimental results, which encourage further research.

The detector performance have been evaluated by means of an experimental campaign (see section 5) on a case study coming from the ATM (Air Traffic Management), namely the SWIM-BOX<sup>®</sup>, which is a prototype developed at SESM<sup>2</sup> to allow the cooperation and the interoperability of future ATM systems. Results have

---

<sup>2</sup> SESM s.c.a.r.l., a Finmeccanica company. <http://www.sesm.it/>

been compared with an algorithm, which adopts worst-case thresholds (in the following we refer to this algorithm as Static Thresholds Algorithm), in order to explore possible improvements adopting this algorithm in the same situations [5]. The experimental campaign involved a complete and sound testing activity, which explores the performance of both the algorithms using a large set of possible configurations. In particular, we execute fault injection experiments to accelerate the failure related data collection, which allows labeling the relevant anomalies (namely, those due to fault activation).

To evaluate performance we relied on the metrics for failure prediction used by Malek in [1] and the Quality of Service (QoS) metrics for failure detectors provided by Chen, Tuogeg and Aguilera in [12], furthermore an analysis of which class of metrics best describes this class of algorithm has been provided.

The paper is organized as follows. Section 2 gives a brief description of the Detection Framework, section 3 introduces the main steps to adapt the SPS algorithm to this context, section 4 gives a survey of the most used metrics in literature, section 5 describes the experimental campaign and section 6 analyzes the obtained results. Finally conclusion and future work are in section 7.

## 2 The Detection Framework

The Detection Framework was proposed in [5]. The Authors propose an approach based on indirectly (and locally) inferring the health of the monitored component by observing its behavior and interactions with the external environment. The basic idea is to shift the observation perspective and to leverage OS support to detect application failures.

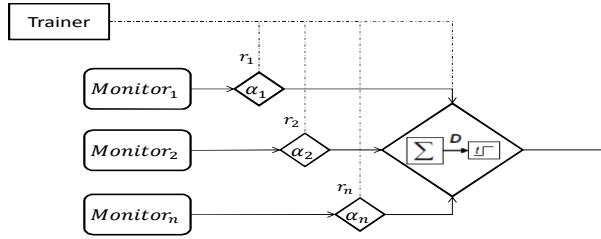
### 2.1 Assumptions

**System Model.** The aim is to detect failures in complex Off-The-Shelf (OTS) based safety critical software systems. These are often distributed on several nodes, which communicate through a networking infrastructure. However, we focus on a single node of the system to perform failure detection and we do not care of system topology. In this context, failure detection is performed at process (thread) level.

**Failure Model.** According to the failure classification proposed in [1] we focus on (i) crash failures and (ii) hang failures, i.e., failures which cause the delivered service to be halted and the external state of the service to be constant. In our context, a service is crashed when the process terminates unexpectedly (e.g., due to run-time exceptions). Thus we refer to systems whose failures are to an acceptable extent halting failures only, namely fail-halt (or fail-stop) system. For a more detailed description of this class of failures we refer to [5].

### 2.2 The Detection Approach

The detection framework is based on the combination of several OS level monitors. As suggested by intuition, combining multiple alarms coming from several sources allows revealing a higher number of failures, as well as to gain a better accuracy, if compared to detector without combination.



**Fig. 1.** The detection framework architecture

As depicted in Figure 1, multiple monitors  $M_i$  keep track of OS data related to a given process (thread). Each monitor is followed by an alarm generator  $\alpha_i$ . If the monitored value  $n$  does not belong to the specified range  $r_i = [r_i^-, r_i^+]$ , an alarm is triggered. Actual thresholds have to be preliminarily tuned for each monitored variable, during a so-called training phase (see the Trainer block in Figure 1). Training is performed by means of an initial profiling phase analyzing both normal and faulty runs (namely, runs when a failure occurs). As previously stated, alarms, triggered by alarm generators, are combined in order to improve detection quality.

The detector  $D$  performs the overall detection by means of a simple heuristic defined as the weighted sum of single alarms, where weights are defined after the training phase. A failure is finally detected if the output of  $D$  exceeds a given threshold  $t$  tuned during the training phase too.

### 2.3 Limitations of the Static Thresholds

As described in previous section static threshold algorithms perform well enough if the environmental conditions in which the system operates are similar to the training phase. Performance goes worse if the operational conditions of system differ from those of the training phase since the evaluated thresholds may no longer be able to model the nominal behavior. This last situations is no far from real scenarios if we consider Large scale Complex Critical Infrastructures (LCCI) as possible application field, in which detection algorithms have to deal with highly variable scenarios, in which the whole system evolves in time and space dimension, alternating periods of heavy workloads, which involved high number of nodes and heterogeneous type of service requests, with periods of low computational activities. In this type of environment, training phases need to be performed periodically in order to keep tuned the algorithm. Since this kind of dynamical environment changes are often very hard to predict an algorithm that computes adaptive, instead of static, thresholds could overcome such limitations because, as shown in Section 6, it does not depend from an initial training phase.

## 3 Using SPS Algorithm to Estimate Adaptive Thresholds

In this section we first give an overview of SPS algorithm, by discussing its assumptions and by showing how it can be used to provide adaptive thresholds to the



detector, then we illustrate the fundamental differences between static and adaptive thresholds.

### 3.1 SPS-Based Detection Algorithm

The SPS algorithm was initially designed to compute uncertainty interval at a time  $t$  within a given coverage, namely the probability that the next value of the time series will be inside the uncertainty interval. This algorithm can be adapted to compute adaptive bounds for anomaly detection activities with minor changes. As shown in [7], the uncertainty computed by SPS algorithm consists in a combination of left and right bounds. These bounds are computed starting from three quantities: (i) the *last value of the series*, (ii) the output of a *predictor* function and (iii) the output of a *safety margin* function. The output of the SPS at  $t \geq t_0$  is constituted by the two values:

$$U_r(t) = \max(0, \tilde{\Theta}(t_0)) + P_r(t) + SM_r(t_0) \quad U_l(t) = \min(0, \tilde{\Theta}(t_0)) + P_l(t) + SM_l(t_0)$$

Where  $\tilde{\Theta}(t_0)$  is the last value of the time series (i.e., the estimated offset for the time synchronization environment). In computing uncertainty in time synchronization the requirements state that the uncertainty interval must contain the global time, so we distinguish left from right uncertainty by considering the offset only if it is negative in the former, positive in the latter.

We adapted the computation of the bounds assuming symmetrical values for the predictor and the safety margin functions ( $P_r(t) = -P_l(t)$  and  $SM_r(t) = -SM_l(t)$ ) and computing the adaptive bounds as:

$$T_u(t) = x(t_0) + P(t) + SM(t_0) \quad T_l(t) = x(t_0) - P(t) - SM(t_0)$$

Where  $T_u(t)$  and  $T_l(t)$  are the upper and the lower bounds at time  $t$ , and  $x(t_0)$  is the last value of the series.

The *predictor* function provides an estimation of the behavior of the time series. The *safety margin* function aims at compensating possible errors in the prediction and/or in the measurement of the actual value of the time series. The *safety margin* is computed at  $t_0$  and it is updated only when new measurements arrive. We refer to [7] for technical details about the predictor and the safety margin functions.

The set-up parameters used by SPS are: four probabilities  $p_{ds}, p_{dv}, p_{os}, p_{ov}$ , that can be combined in one single parameter  $c$  (coverage of the algorithm) and the two different values for memory depth  $M_1 = M_2 = m$ . The performance achieved by SPS depends on these parameters [7].

Computational cost of the SPS algorithm depends on the computation of a population-weighted variance. Since variance is computed using sums of the elements, the computational cost of the algorithm is linear with the number of samples. If we use accumulators to store the value of the sums in memory, the computational cost of SPS becomes constant. This last solution is obviously preferred when we want to use the algorithm at runtime considering a large set of samples.

### 3.2 SPS Assumptions

The measurements provided by monitors are received at regular interval of time. Let be  $k$  the number of different monitored variables.

**Def. 3.1 (timeseries).** A time series  $T_i = x_{i1}, \dots, x_{ij}$  is an ordered set of  $j$  real-valued variables.

**Def. 3.2 (anomaly).** With respect to a monitored time series  $T_i$ , an **anomaly** is a change in the characteristics of  $T_i$ , caused by specific and non-random factors.

The continuous stream of data points  $x \in R^k$  constitutes the collection of measurements. These measurements correspond to certain physical events in the event space  $S$ , which we assume can be divided into two subspaces corresponding to normal events ( $S_N$ ) and anomalous events ( $S_A$ ).

In order to apply SPS algorithm to time series  $T_i$ , we make the following assumptions.

**Random Walk Model.** We assume that the monitored process behavior can be modeled as a random walk, with or without drift. Namely, the variability of the process is the result of the cumulative effect of small but unavoidable constant and casual factors.

**Interleaved Behavior.** We assume that the environment alternates *stable periods*, during which the monitored process has some stability properties (i.e., it is under control), with *transient periods* (smaller compared with the stable), during which a variation of environmental or system condition occurs (due to workload, new configuration) involving a change in the characteristics of the monitored process. This assumption is supported by the results of many recent works [13].

**Transient Period Changing.** We assume that, during the transient period, changes in the monitored processes behavior consist in continuous increments or decrements with respect to previous values. These changes are due to some specific factors, which are not casual, such as: a modification of system structural parameters (e.g., the number of active nodes), overloading conditions (e.g., due to a burst of requests), the activation of a residual fault leading to system failures (e.g., crashes, hangs).

It is worth noting that, the proposed detection approach does not make any assumptions about the stationarity of the monitored variables. Namely, if the statistical properties of the monitored process (e.g., mean and variance) change over time, the detection of anomalies is still possible. Relaxing the stationary hypothesis makes the detector more suitable for real variable contexts with respect to the case in which these properties are statically derived by means of preliminary profiling phase.

### 3.3 The Detector Equipped with SPS

The detector can be easily equipped with SPS modifying the alarm generator component, (i.e.,  $\alpha_i$  in Figure 1). SPS continuously processes data received from the associated monitor, thus it will be in charge to provide adaptive thresholds to each  $\alpha_i$ . In this way the training phase, which in the previous version of the detector was necessary to compute static thresholds and to find weights for each monitor, is totally avoided.

### 3.4 Comparison between Adaptive and Static Thresholds Algorithm

Figure 2 shows adaptive thresholds computed by SPS compared with Static thresholds for the same monitored variable (total number of timeouts expired for scheduling of processes).

SPS thresholds signals the failure (at about 160 s), while, as we can observe in the left part of Figure 2, the monitored value is very often above the upper static threshold, producing a lot of False Positive.

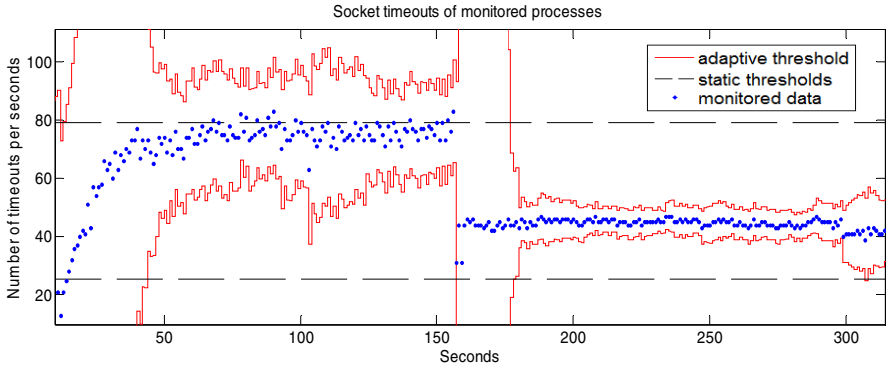


Fig. 2. Static thresholds and Adaptive thresholds on monitored data

## 4 Metrics for Performance Evaluation

In order to provide a fair and sound comparison between SPS and Static Thresholds Algorithm we analyze criteria that should be used to characterize on-line detectors performance for our target applications. We first summarize some of the most used metrics in literature then we must specify, which metrics are the most representative in our scenario and why.

Roughly speaking the goal of an on-line failure detector is i) to reveal all the occurring failures, ii) to reveal them timely and ii) not to trigger false alarms.

To ease the description of metrics and to better understand their meaning we introduce some definitions:

- *True Positive (TP)*: if a failure occurs and the detector triggers an alarm;
- *False Positive (FP)*: if no failure occurs and an alarm is given;
- *True Negative (TN)*: if no real failure occurs and no alarm is raised;
- *False Negative (FN)*: if the algorithm fails to detect an occurring failure.

Clearly many TPs and TNs are good, while the vice versa for FPs and FNs.

Metrics coming from diagnosis literature are usually used to compare the performance of detectors [9]. For instance *coverage* measures the detector ability to reveal a failure, given that a failure really occurs; *accuracy* is related to mistakes that a failure detector can make. Coverage can be measured as the number of detected

failures divided by the overall number of failures, while for accuracy there are different metrics.

Basseville et al. [3] consider the *mean delay for detection* (MDD) and the *mean time between false alarms* (MTBFA) as the two key criteria for on-line detection algorithms. Analysis are based on finding algorithms that minimize the mean delay for a given mean time between false alarms and on other indexes derived from these criteria.

In [2] metrics borrowed from information retrieval research are used, namely *precision* and *recall*. In their context recall measures the ratio of failures that are correctly predicted, i.e.,  $TP/(TP+FN)$ , while precision measures the portion of the predicted events, which are real failure, i.e.,  $TP/(TP+FP)$ . Thus perfect recall (recall=1) means that all failures are detected and perfect precision (precision=1) means that there are no false positives. A convenient way of taking into account precision and recall at the same time is by using *F-measure*, which is the harmonic mean of the two quantities. Since in diagnosis the ratio of failures correctly detected (recall) is also called coverage, in the following we refer to it as coverage.

However using solely precision and coverage is not a good choice because they do not account for true negatives, and since failures are rare events we need to evaluate the detector mistake rate when no failure occurs. Hence, in combination with precision and coverage, one can use *False Positive Rate* (FPR), which is defined as the ratio of incorrectly detected failures to the number of all non-failures, thus  $FP/(FP+TN)$ . Fixing Precision and Coverage, the smaller the false positive rate, the better. Another metric is *Accuracy* [2], which is defined as: the ratio of all correct decisions to the total number of decisions that have been taken, i.e.,  $(TP+TN)/(TP+TN+FP+FN)$ .

Chen, Toueg and Aguilera [12] propose three primary metrics to evaluate detectors quality, in particular their accuracy. The first one is *Detection Time* (DT), which, informally, accounts for the promptness of the detector. The second one is the *Mistake Recurrence Time* ( $T_{MR}$ ), which accounts for time elapsed between two consecutive erroneous transitions from Normal to Failure. Finally they define *Mistake Duration* ( $T_M$ ), which is related to the time that detector takes to correct the mistake. Other metrics can be simply derived from the previous one. For instance, *Average Mistake Rate* ( $\lambda_M$ ), represents the number of erroneous decisions in the time unit; *Good period duration* ( $T_G$ ) measures the length of period during which the detector does not trigger a false alarm; *Query accuracy probability* ( $P_A$ ) is the probability that the failure detector's output is correct at a random time.

Bearing in mind classes of our target applications we believe that, when dealing with long running and safety critical systems, mistake duration (and thus  $T_G$ ) is less appropriate than Coverage, accuracy and  $\lambda_M$ , since just an alarm may be sufficient to trigger the needed actions (e.g., put the system in a safe state). Coverage is essential because if the detector does not reveal a failure, then more severe (and potentially catastrophic) consequences may happen. Accuracy and  $\lambda_M$  are useful to take into account false positives because each failure detector mistake may result in costly actions (such as shut down, reboot, etc.).

The query accuracy probability is not sufficient to fully describe the accuracy of a failure detector, in fact, as discussed in [12], for applications in which every mistake causes a costly interrupt the mistake rate is an important accuracy metric too.

We point out the differences between Accuracy, defined in [2], and  $P_A$  defined in [12]. Since we consider a fail stop model,  $TP \ll TN$ , so if  $FP \ll TN$ , then  $\text{Accuracy} \approx 1$ . For these reasons we consider  $P_A$  as more representative than Accuracy to compare SPS algorithm with Static Thresholds Algorithm.

Finally we introduced an additional parameter for performance evaluation, the time-to-detection  $d$ . This parameter represents the maximum delay to detect a failure. Thus, considering  $T_f$  the time when a failure occurs, if an alarm is raised after  $T_f + d$  we do not account it as a TP. To be sure that a true positive is effectively related to the failure we set  $d \leq m$  (where  $m$  is the memory depth of SPS algorithm).

**Table 1.** Metrics for performance evaluation of failure detectors

<i>Metric</i>	<i>Formula</i>	<i>Metric</i>	<i>Formula</i>
Coverage (C)	$TP/(TP + FN)$	Accuracy-Coverage TradeOff	$C \cdot A$
Precision (P)	$TP/(TP + FP)$	MTBFA (Mean Time Between Mistakes)	$E(T_{MR})$
F-Measure	$(2 \cdot P \cdot C)/(P + C)$	MDD (Mean Delay for Detection)	$E(T_D)$
FPR (False Positive Rate)	$FP/(FP + TN)$	$\lambda M$ (Average Mistake Rate)	$1/E(T_{MR})$
Accuracy (A)	$\frac{TP + TN}{TP + FP + TN + FN}$	$P_A$ (Query accuracy probability)	$\frac{E(T_{MR} - T_M)}{E(T_{MR})}$

## 5 Experimental Campaign

We performed an experimental campaign with the aim of comparing performance of two Detection frameworks using both Static Thresholds and SPS algorithm as alarm generators.

The testing activity was performed analyzing a large amount of data monitored in a real and complex case application, namely the SWIM-BOX®. The application is made of several OTS, e.g., OS, the application Server (JBoss) and the data distribution middleware (OpenSplice). We executed tests under different workloads and faultloads, using fault injection technique described in [4], in order to accelerate the failure related data collection. The monitored data of each test was loaded into a well-structured data-repository following an OLAP approach [13]. The two algorithms are then applied and evaluated in a post-processing phase.

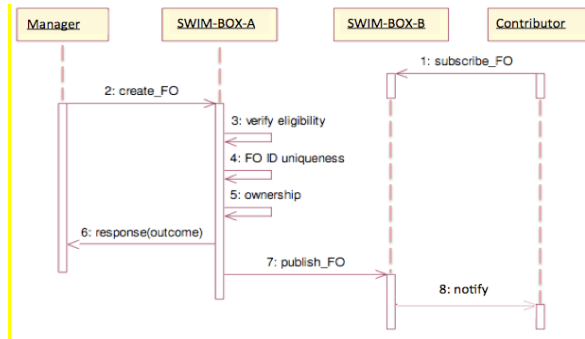
### 5.1 Case Study Description

The SWIM-BOX® is actually a pilot prototype, which has been implemented in the framework of the SWIM SUIT FP6 European project<sup>3</sup>, to support global

<sup>3</sup> <http://www.swim-suit.aero/swimsuit/>

interoperability for the novel Air Traffic Management (ATM) systems. It is a complex OTS-based application, which offers several facilities to SWIM-BOX users: synchronous/asynchronous communication pattern (i.e. request/reply, publish/subscribe), security services (e.g., authentication, authorization, encryption) and distributed and transactional data storage.

The case study scenario consists of two legacy entities, named the Contributor and the Manager, which collaborate by means of the SWIM-BOX to manage Flight Data Plan.



**Fig. 3.** Simplified interaction scenario

Figure 3 describes an example of the interaction between the legacy systems. The Contributor acts as the subscriber, waiting for Flight Object updates (i.e., an entities including several data related to a flight) to be published. Also, it periodically reads all the available Flight Object summaries. Conversely, the Manager is in charge of (i) executing a given number of operations (e.g., Flight Data Object creations and updates) at a variable rate (20 to 300 operations/min in the experiments), as well as of (ii) distributing data over the network. Once all the operations have been completed the Contributor requires unsubscribing. Two legacy entities Manager and Contributor, represent the system under test.

## 5.2 The Experimental Activity

Detection Framework analyzes several variables by means of the monitoring infrastructure described in [5]. Monitors are based on OS instrumentation facilities, which allow implanting probes into the kernel and register corresponding probe handlers. Probes are breakpoints inserted dynamically into the kernel module avoiding recompiling and rebooting. When a breakpoint is hit, a handler routine is launched to register an event with the needed information (e.g., input parameters or return values of called functions).

The events collected during this experimental campaign are shown in Table 2.

Monitors are associated to probes and provide measurements by aggregating the number of events recorded by a probe in a given time period. Other monitors,

measure the disk/network throughput, by summing bytes read or write, respectively on the disk and on the network interfaces. The total number of types of monitor is 18.

This infrastructure is completely configurable such that one can choose (i) processes and network interfaces to observe, (ii) timeouts associated to a particular probe (when needed) and (iii) the sample period of monitors (i.e., the interval of time between two consecutive measurements).

**Table 2.** Probe description

Probe	Trigger condition for events registration
System call error code	An error code is returned
Time scheduling of process	Timeout exceeded since the process is preempted
Signal	A signal is received
Process/Thread creation/termination	Creation or Termination of a Process (Thread)
I/O on Disk	Timeout exceeded since last disk read/write
I/O on Socket	Timeout exceeded since last socket read/write
Holding time for Mutex/Semaphore	Timeout exceeded for mutex/semaphore possession
Waiting time for Mutex/Semaphore acquisition	Timeout exceeded for mutex/semaphore acquisition
Disk Throughput	A byte is read/write
Network Throughput	A byte is send/received

We evaluate the overhead of the monitoring infrastructure by measuring the execution time of our application when the monitors are turned off and when they are enabled. Overhead results almost negligible (about 3%).

**Table 3.** Experimental activity dimensions

Dimensions	Description
Target System	Characteristics of the Target System,(CPU, RAM, disk speed, ...)
Events	Monitored events
Run	Information on the executed run (start time, end time, ...)
Scenario	In this campaign we considered only the scenario described in 5.1: Two Entity, one Manager and one Contributor.
Workload	Adopted Workloads differs from message rate, message burst rate and message per burst.
Faultload	Several faults injected (one per injection) by means of code mutation technique [4].

Monitored data were stored in an online data repository following an OLAP approach [13]. Dimensions and description of the OLAP repository are shown in Table 3.

We considered a subset of the mathematical combinations of all dimensions, and performed 17 faulty runs and 19 nominal runs. After the execution of the runs both algorithms were applied in a post-processing phase, varying several configurations.

Combinations of runs and considered configurations give 6300 different sets of data (see sec.5.3).

### 5.3 The Post Processing Phase

After the execution of all the runs both Algorithms are applied in a post-processing phase, varying several configurations. The evaluation of metrics is carried out by applying algorithms to dataset, which have not been used for parameters tuning. For these reasons the dataset is divided into a training set and a validation set, which consist respectively in 6 and 30 runs.

**Post Processing SPS.** SPS algorithm was applied varying the following parameter: coverage  $c$ :  $\{0.9, 0.99, 0.9999\}$ , memory depth  $m$  (in terms of number of data considered in the statistics):  $\{10, 20\}$ , time for detection  $d$ :  $\{m, \lfloor \frac{m}{2} \rfloor, \lfloor \frac{m}{3} \rfloor\}$ . To be sure that a true positive alarm is effectively related to the failure we set  $d \leq m$ . Combining these tree dimension we obtains 18 different configurations for SPS.

**Post Processing Static Thresholds.** Static Thresholds algorithm was applied by varying (i) method for the evaluation of thresholds  $r_i$  for each monitor  $M_i$  and the time for detection  $d$ :  $\{3, 6, 10, 20\}$ . In particular we estimate the thresholds on the training set using 3 different methods:  $r_i = \{\{min, max\}, [\mu - \sigma, \mu + \sigma], [\mu - 2\sigma, \mu + 2\sigma]\}$ . The total number of configurations is 12.

For Both SPS and Static Threshold we consider 5 values for parameter  $t$  in the global detector, so we have a total of 150 different configurations. All 90 SPS configurations was post processed for the validation set (30 runs), while the 60 Static Algorithm configurations were post processed first for validation set by carrying out the training phase of algorithm, and then to the validation set without the training phase, for a total of 6300 sets of analyzed data.

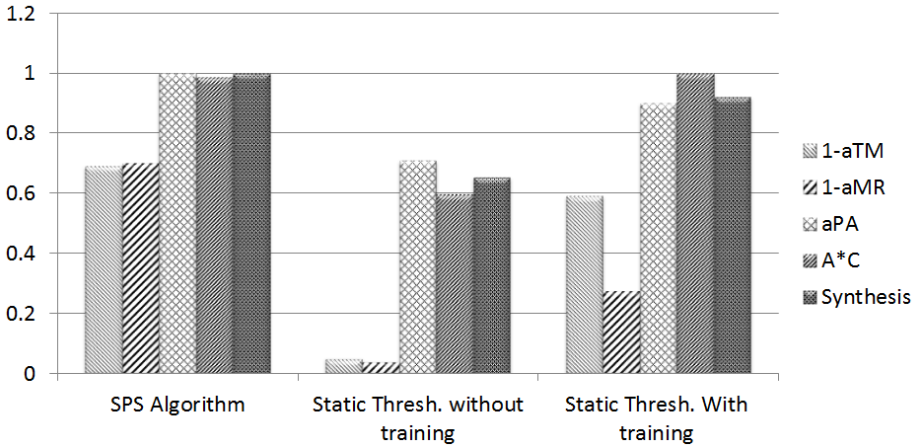
## 6 Results Analysis

We pointed out 3 different scenarios to be compared: SPS Algorithm; Static Thresholds Algorithm with a training phase; Static Thresholds Algorithm without a training phase.

Since it could be very difficult and confusing but also not useful to compare the two algorithms for all possible configurations and all metrics we decided to compare only the best configuration for the two algorithms. To select the best configuration among all the metrics described in section 4, we introduced a synthetic measure  $Synthesis = mean(|aPA| + |Coverage| + |Accuracy| + (1 - (|aTM|)) + (1 - |aMR|))$  that takes into account most relevant metrics that are not correlated together. Synthesis allows weighting the metrics according to the type of system at hand. For our purposes we considers metrics with the same weight.

Configurations with best Synthesis measure were  $c = 0.99, m = 20, d = 20, t=0.4$  for SPS,  $d = 3, t = 0.5$  for Static Thresholds with the training phase and  $d = 3, t = 0.3$  for Static thresholds without a training phase. It's noteworthy that the absence of the training phase for Static algorithm involves setting all weight to 1 in the global detector.





**Fig. 4.** Compared experimental results

The following metrics: average Mistake Duration (aTM), average Mistake Rates (aMR), average Query Accuracy (aPA), Accuracy-Coverage tradeoff (A\*C), and the Synthesis metric has been normalized in Figure 4 to ease the comparison, while the absolute values are showed in Table 4.

**Table 4.** Experimental results

	aTM (sec)	aMR	aPA	A*C	Synthesis
SPS Algorithm	3.611111	0.009804	0.95098	0.96004	0.914551
Static T. without training	11.63951	0.032724	0.674494	0.582356	0.59768
Static T. with training	4.75	0.023715	0.85584	0.974436	0.842296

The weakness of static thresholds algorithm described in section 2.3 has been proved by the poor results achieved without the training phase. Moreover, SPS-based detection algorithm shows better results, proving that adaptive thresholds give an effective gain in this class of detectors.

## 7 Conclusion and Future Work

This work proposes an anomaly-based approach for software fault detection in complex critical systems, exploiting statistical analysis on data gathered at the Operating System level. The proposed approach relies on the SPS algorithm, which was initially designed to compute uncertainty in clock synchronization [7]. Performance of this algorithm has been evaluated by means of an experimental campaign and compared with a static threshold algorithm proposed in [5]. The comparison has been made, first exploring the existing metrics in literature [2],[12] and then applying the most representative to both algorithms. Experimental results

confirm the limitations of static threshold algorithms in highly variable scenarios, where the operational conditions of system differ from those of the training phase. Moreover results show that, compared to static threshold based algorithms, the SPS algorithm is able (i) to cope with variable and non-stationary behavior, (ii) to work without any initial training phase, and (iii) to perform better even when a training phase is applied, in terms of coverage, query accuracy probability, mistake rate and mistake duration.

As future work we will implement this algorithm to test and validate it when executed online and compare it with other techniques used in failure detection field, such as machine learning algorithm, and ARIMA models [3].

**Acknowledgment.** This work has been partially supported by the Italian Ministry for Education, University, and Research (MIUR) in the framework of the Project of National Research Interest (PRIN) “DOTS-LCCI: Dependable Off-The-Shelf based middleware systems for Large-scale Complex Critical Infrastructures” (dots-lcci.prin.dis.unina.it).2008. It is also in the context the “Iniziativa Software” Project, an Italian Research project which involves Finmeccanica company and several Italian universities (www.iniziativasoftware.it).

## References

1. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. Dependable Secure Computing* (2004)
2. Salfner, F., Lenk, M., Malek, M.: A survey of online failure prediction methods. *ACM Computing Surveys, CSUR* (2010)
3. Basseville, M., Nikiforov, I.V.: *Detection of abrupt changes: theory and application*. Prentice-Hall, Inc., Englewood Cliffs (1993)
4. Natella, R., Cotroneo, D.: Emulation of transient software faults for dependability assessment: A case study. In: *Proceedings of the Eighth European Dependable Computing Conference, EDCC* (2010)
5. Carrozza, G., Cinque, M., Cotroneo, D., Natella, R.: Operating System Support to Detect Application Hangs. In: *International Workshop on Verification and Evaluation of Computer and Communication Systems, VECoS* (2008)
6. Irrera, I., Duraes, J., Vieira, M., Madeira, H.: Towards Identifying the Best Variables for Failure Prediction Using Injection of Realistic Software Faults. In: *Pacific Rim International Symposium on Dependable Computing. IEEE, Los Alamitos* (2010)
7. Brancati, A., Bondavalli, A., Ceccarelli, A.: Safe estimation of time uncertainty of local clocks. In: *International Symposium on Precision Clock Synchronization for Measurement, Control and Communication* (2009)
8. Salfner, F.: *Event-based failure prediction: an extended hidden Markov model approach*, Dissertation.de, Berlin (2008)
9. Daidone, A.: *Critical infrastructures: a conceptual framework for diagnosis, some applications and their quantitative analysis*. PhD thesis, Università degli studi di Firenze (December 2009)
10. Johnson, C., Malek, M.: *Progress achieved in the research area of Critical Information Infrastructure Protection by the IST-FP6 Projects CRUTIAL, IRRIS and GRID*. Technical report, EU Report (March 2007)

11. Montgomery, D.C.: *Controllo statistic della qualità*, 1st edn. McGraw-Hill italia, New York (2000)
12. Chen, W., Toueg, S., Aguilera, M.K.: On the Quality of Service of Failure Detectors. In: *Proceedings of the 2000 International Conference on Dependable Systems and Networks (formerly FTCS-30 and DCCA-8)* (2000)
13. Casimiro, A., Lollini, P., Dixit, M., Bondavalli, A., Verissimo, P.: A framework for dependable QoS adaptation in probabilistic environments. In: *Proceedings of the 2008 ACM Symposium on Applied Computing* (2008)
14. Madeira, H., Costa, J., Vieira, M.: The OLAP and data warehousing approaches for analysis and sharing of results from dependability evaluation experiments. In: *2003 International Conference on Dependable Systems and Networks, 2003*, pp. 86–91 (June 22-25, 2003)

# Security Analysis of Smart Grid Data Collection Technologies

Luigi Coppolino, Salvatore D'Antonio,  
Ivano Alessandro Elia, and Luigi Romano

University of Naples "Parthenope", Department of Technology, Italy  
{luigi.coppolino,salvatore.dantonio,  
ivano.elia,luigi.romano}@uniparthenope.it

**Abstract.** In the last few years we are witnessing a dramatic increase in cyber-attacks targeted against Critical Infrastructures. Attacks against Critical Infrastructures are especially dangerous because they are tailored to disrupt assets which are essential to the functioning of the society as a whole. Examples of Critical Infrastructure sectors include transportation, communication, and utilities. Among these, power grids are possibly the most critical, due to the strong dependency of virtually all Critical Infrastructures on the power infrastructure. We have conducted a security analysis of two key technologies which enable data collection in Power Grids, namely synchrophasor devices and Phasor Data Concentrators. We emphasize that the study has been conducted on a commercial synchrophasor produced by a major vendor, and on a widely used open source product for the Phasor Data Concentrator application. We describe the experimental setup, present the main results, and comment the findings of our research.

**Keywords:** Phasor Measurement Units, Synchrophasors, Phasor Data Concentrators, Security Analysis, Power Grids, Smart Grids.

## 1 Rationale and Contribution

Information Technology (IT) and Operational Technology (OT)<sup>1</sup> are progressively converging, as reported by the independent analyst Gartner in [1]: *"the nature of the OT systems is changing, so that the underlying technology such as platforms, software, security and communications is becoming more like IT systems"*. If on the one hand such a convergence is a source for new opportunities, on the other it results in new challenges and threats, especially when it occurs in systems providing critical functions (such as energy distribution, e-health, transportation, etc.). Indeed weaknesses and vulnerabilities of the underlying IT systems, can be exploited to compromise the correct provisioning of critical services. Evidence is showing that Critical Infrastructures (CIs) are already exposed to Cybersecurity attacks [2][3] and they will be even more so in

---

<sup>1</sup> The term Operational Technology in this report is used as a synonym of physical-equipment-oriented technology.

the future, as suggested by both McAfee and Symantec in some recent reports [4] [5]. In particular, in [4] McAfee describes a major hacking initiative, called “the night dragon” (since it originated in China), specifically targeting power grid systems. Given that Power Grids will evolve towards Smart Grids, the possibility of successful attacks against Power Grid IT management layers might have a dramatic impact in the future, when Smart Grids will be provided of reconfiguration abilities able to trigger automated reactions in case of detection of abnormal behavior of the network. The correct selection of a reconfiguration strategy depends on the ability to precisely monitor the status of the grid. Phasor Data Concentrator (PDC) and Phasor Measurement Unit (PMU) represent key technologies for Power Grid data monitoring, where electrical waves in the power distribution infrastructure are analyzed to evaluate the system status and to diagnose possible problems and faults. A more precise view of the power distribution network can be obtained by correlating information gathered by multiple PMUs deployed in a single grid. This correlation is made possible by the adoption of synchrophasors, that is synchronized PMUs with a common time source from a GPS radio clock. In this context, guaranteeing the integrity of collected measurements is of paramount importance, since their alteration may result in wrong reconfiguration actions and possibly in money losses and blackouts with unpredictable cascade effects, possibly affecting multiple countries [28] [29] [30].

In this paper we have conducted a security analysis of two key technologies which enable data collection in Power Grids, namely synchrophasor devices and Phasor Data Concentrators. Among the projects that rely on the use of these technologies, we can cite the NASPI network (NASPInet) [13], a framework project involving members of the NERC [14] corporations and the U.S. Department of Energy and the North American consumers and utilities, and the Frequency Monitoring Network (FNET) [15] project which is aimed to detect power systems anomalies by means of Frequency Disturbance Recorders (FDRs) and Information Management Systems (IMs).

We emphasize that the study has been conducted on a commercial product by a major vendor (as far as the synchrophasor is concerned), and on a widely used open source product (as far as the Phasor Data Concentrator is concerned). We have set up a simplified - yet realistic - testbed, and we have conducted a penetration testing campaign against the two aforementioned components. As a result of the testing sessions, we exposed several vulnerabilities, some of which can be easily exploited for conducting attacks to current smart grid data collection infrastructures if proper measures are not taken and additional protection devices are not integrated in the system. We explicitly note that major synchrophasor vendors and power operators take security very seriously, and that is why they provide a plethora of products, which allow security enhanced deployment of smart grid data collection infrastructures. These solutions pursue security at the system level, rather than at the level of individual devices/components. Commercial products and/or best practices which provide answers to some of the problems which we have pointed out include: [19] [20] [21] [22] [23] [24] [25] [26] [27].

The rest of the paper is organized as follows. In Section 2 we introduce synchrophasor and other related technologies (including the most used standard for PMU communications, namely the C37.118 protocol). In Section 3 we present the experimental testbed, and the key results of our security analysis. Finally, we provide some concluding remarks in Section 4.

## 2 Smart Grids and Phasor Measurement Units

Even if the phasor mathematical concept dates back to more than one hundred years ago, the first evaluation of phase differences started in the early 80s, when several instrumentations helped to have the same time-synchronized reference in different locations (e.g. LORAN-C, GOES satellite, and HBG radio transmission). The synchrophasors are the evolution of the first-generation devices, traditionally known as PMU [12], that add real time monitoring capabilities, which are enabled by the use of a Global Positioning System (GPS) synchronized time source.

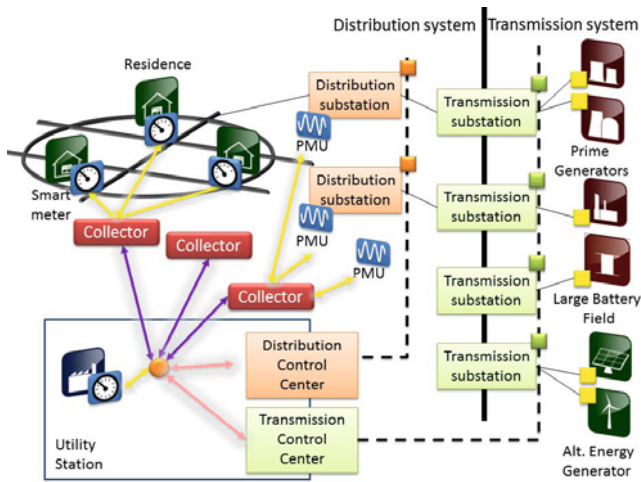
Observing the power line dynamics needs real time measurement, which is not achievable with current state of the art SCADA technology (since the most common versions of traditional SCADA systems are unable to offer real time dynamics and angle evaluations). The use of GPS gives the opportunity of exploiting an UTC timestamp and using it as reference to evaluate the phasor angle in the same time slice in different locations of the Power Grid. This UTC time is given with a fractional seconds precision. Among the others, the GPS clock is also a reference for the PMU signal sampling rate.

The need for increasing the reliability and efficiency of power distribution services led to improve the capabilities and security properties of power distribution systems. The adoption of interconnected infrastructures has allowed the power companies to exchange information, maintain frequency and voltages stabilities and control undesirable load shed events; moreover systems maintenance costs were reduced, and new emergency plans and techniques could be developed.

Smart grids allow new paradigms in power generation, consumption, and delivery by the adoption of new emerging communication and information technologies and frameworks. Smart grid networks (Fig. 1) are made of several functional units, each one in charge of different tasks (specifically: each consumer manages a smart controller and meter connected to a collector node; the collector node receives inputs from several residences and transmits them to an utility station through the Internet; the utility is in charge of transmitting data to the distribution and the transmission system, usually through an intranet).

Today synchrophasors are used for substation analysis, power systems analysis, and wide area control.

The most important monitoring activities for the power stakeholders can be divided in two main categories: normal operations and emergency operations. In the first case the power supplier should check for frequency deviations: the generators have a tight frequency bandwidth operativity, so larger deviations can induce transmission trips also from several energy suppliers, due to the



**Fig. 1.** A schematic representation of the main components of a smart grid

interconnection of generation sources. During normal operations stakeholders consider also the voltage deviations: even if less restrictive than the frequency ones, these deviations must be controlled to avoid overcoming the tolerance range and check for events like uncontrolled brownout or voltage surges.

During the emergency operations the most important events to face are the loss of generation and the loss of transmission (input and output): in the case of generation loss and input transmission outages, the power load is greater than the supplied power and the working frequency shows a huge deviation.

In the substation analysis synchrophasors are used for verifying the voltage and current phasing of the substation, giving a reference for the equipment. They can also help in voltage measurement refinement, solving issues involving the meter equipment or devices. Moreover synchrophasors can interact with a legacy SCADA system for measurement verifications, since they are more accurate and work at higher rate than SCADA systems.

In the field of power system analysis synchrophasors allow wide area disturbance monitoring with timeliness capabilities: since there is a common time reference the synchrophasors are easily correlated and processed in real-time. Synchrophasors voltage measurements support the system state estimation: typical SCADA evaluation were based on magnitude only, inaccurate and not time aligned measurements. Since the monitoring of a wide area can induce communication difficulties, the synchrophasors data are more often used as an input for a disturbance recorder (SDR), so that some events are recorded and evaluated locally and then reported to wide area collectors.

The wide area control consists in detecting known events or anomalies and take countermeasures for them. Among the others we have the anti-islanding control (islanding is a phenomenon affecting power grids, as a result of a power system failure, where the power grid is separated in two or more fragments called

islands): in this case the system tries to disconnect the generator from the grid in timely manner. Synchrophasors are used also for the so called black start of a generation unit.

One of the most interesting analyses developed by means of synchrophasors coordinated observations is the evaluation of propagating generator trips [16]. Events observation puts in evidence that a generator trip event causes a frequency reduction. The frequency variation, observed in a certain point, propagates on the transmission lines, showing the same frequency reduction in other sites with a certain delay. Using the triangulation technique, at the locations with the same phase, we can identify the site of the event. Moreover we can estimate the power trip imbalancing, since it is proportional to the frequency reduction: the propagating effect and the power reduction estimation can be used to forecast serious events like blackouts, or simply to prepare the remote power supplier with storage energy sources.

In 2005 the C37.118 protocol was produced as a evolution of the 1344 standard [6] to facilitate integration of synchrophasor technology. The C37.118 is an IEEE standard that defines the protocol for phasor data transmission and synchrophasors devices configuration.

The C37.118 communication standard specifies data types and classes identities; however it does not specify the underlying data communication protocol or medium; usually C37.118 messages are transmitted over TCP, UDP or higher level protocols.

The C37.118 is a lightweight protocol: it has a real-time data stream, occasionally interleaved with configuration information messages. There are five types of messages: Data Frame, Configuration Frame 1, Configuration Frame 2, Header Frame, Command Frame. All the messages contain binary data, except the header messages and some fields in the Configuration Frame that contain human readable (ASCII) coded words. The Data Frames must contain synchrophasor and frequency measurements and optionally other analog values (i.e. power flows) or status words (i.e. breaker status). The C37.118 standard is usually implemented with the client/server paradigm, where the PMU acts as server and the PDC host as client. The Command Frame is sent by the PDC to start or stop the data transmission or to request configuration information. The standard specifies that, if the lower level protocol is the TCP-UDP/IP, each message must be encapsulated in one frame and completed with the CRC word. The standard has been conceived with the possibility to collect measurements coming from different PMUs or PDCs. For this reason the data frames have an initial ID field, used by the collector to identify different sources. Moreover each frame could contain a different number of PMU measurements: each PMU is identified with an ID and a station name in ASCII code (16 bytes); moreover each PMU has different phasor and status channels: each channel or phasor could be identified with an ASCII label of 16 bytes. The Header Frame contains labels about each PMU/PDC that are transmitted in ASCII format within 16 bytes words.

We highlight that the standard does not inherently grant security in the message exchange operations. Moreover there is no specification for the underlying



transmission protocol. So the standard could suffer for vulnerabilities at lower (transmission) and upper (message content) level. These vulnerabilities can be exposed in the case of devices in some way joined to public networks.

### 3 Experimental Testbed and Campaign

#### 3.1 Testbed Setup

Figure 2 shows the testbed we used for our security analysis.

Even though the testbed architecture is a simplified version of a real set-up (which would typically consist of multiple hierarchical levels of PDCs, and also include additional components with the capability of enforcing specific protection mechanisms), we emphasize that our testbed is based on components which are actually used in Smart Grid networks which are currently being deployed. Thus, many of the vulnerabilities which we expose in our study may well be present in real set-ups, especially those - which are not rare indeed - where security-related best practices have been disregarded.

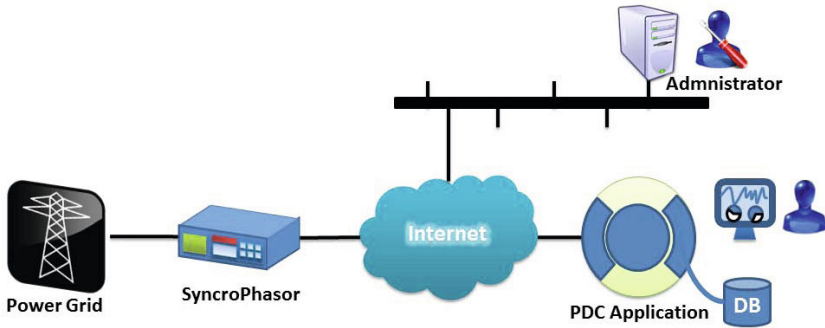
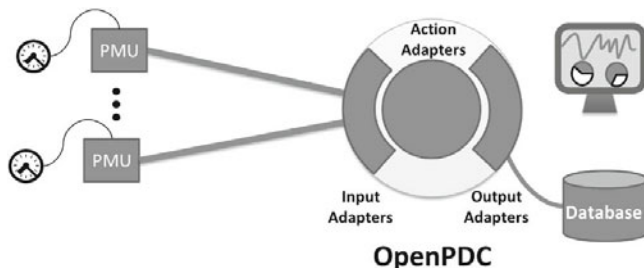


Fig. 2. The schema of the testbed used for our analysis

In smart grid data collection infrastructures, multiple Phasor Measurement Units (PMU) are deployed across a power grid measuring different parameters which provide indications of its health, including phasor measurements. Each measurement is timestamped using a GPS clock. The collected measurements and the associated timestamps are sent to a central PDC, where data is analyzed, processed, stored, and presented to the operator. The timestamping of collected measures enables time-based cross correlation of data.

As already mentioned, our setup is based on realistic components (in particular, we have used a commercial synchrophasor device, which is widely used in this type of applications). The PDC receives phasor data from the synchrophasor using the C37.118 protocol which is the IEEE standard protocol for phasor data management. It uses the OpenPDC [7] platform developed by the TVA (Tennessee Valley Authority) [17], which is the most widely used open source platform for phasor data concentrator applications development.

The OpenPDC platform is compatible with almost all the standard protocols in this application domain (FNET, IEEE 1344 and C37.118 among others) and allows generation time-sorted concentrated data streams. The platform has a modular architecture that gives the developer a great level of freedom allowing her to create applications that perform complex elaboration over the received data. The architecture is composed of three layers (Fig. 3):



**Fig. 3.** The schema of a Phasor Data Concentrator application based on OpenPDC

- Input Adapter Layer
- Action Adapter Layer
- Output Adapter Layer

The Input Adapter Layer allows the development of a module for the interaction with any protocol used to send data to the application. The Action Adapter Layer is where the data elaboration and analysis is performed. The Output Adapter Layer allows integration with different presentation and storage media. In our application phasor data from different locations is transmitted using the IEEE C37.118 standard protocol and collected by an Input Adapter in the application. The data are then transferred to an Output Adapter that stores all the measurements in a MySQL database.

### 3.2 Key Findings and Results

In this subsection we describe the procedures performed in our security analysis and highlight the weak points we have discovered in the different parts of the synchrophasors based PDC monitoring network. Figure 4 summarizes our key findings highlighting the four main areas that have been analyzed in our work: the remote management of the synchrophasor, the passwords management and maintenance policies, the synchrophasor communication with the PDC application, and the PDC implementation.

**Remote Management of the Synchrophasor.** The first part of our analysis has targeted the administrator communications with the synchrophasor. In our setup the synchrophasor is connected to the network through an Ethernet port. The synchrophasor allows remote managing by its administrator over the TCP/IP through Telnet [8], FTP [9], IEC 61850 [10], DNP3 [11] protocols.

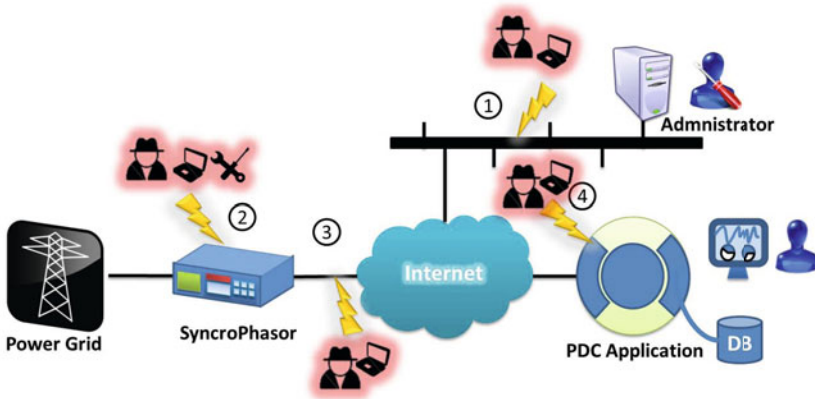


Fig. 4. The main attack points highlighted by our analysis

We have analyzed the protocols behavior using a network sniffer to assess the security of the communications. Our tests highlighted that Telnet communications are totally in cleartext and, as shown in Figure 5, also the authentication passwords are sent without any encryption. Also the other supported protocols do not provide any inherent security performing all communications in cleartext and exposing the synchrophasor to the same vulnerabilities of Telnet.

*Password Eavesdropping:* the lack of encryption on this communications is a great vulnerability to the confidentiality and integrity of the remote management of the synchrophasor. All the passwords that grant administrator privileges to the device are completely exposed to any malicious eavesdropper.

```

prompt password:
0000 00 1d 92 9c 23 7c 00 30 a7 02 1c 52 08 00 45 00  ...#|.0...R...E.
0010 00 37 13 e9 00 00 40 06 e3 84 c0 a8 01 02 c0 a8  .7...@.....
0020 01 01 00 17 04 76 11 74 59 a0 a8 64 9c b4 50 18  ...v.tY..d..P.
0030 22 34 17 a6 00 00 0a 02 0d 0a 50 61 73 73 77 6f  "4.....Passwo
0040 72 64 3a 20 3f                                     rd: ?

sending password ("TAIL"):
0000 00 30 a7 02 1c 52 00 1d 92 9c 23 7c 08 00 45 00  .0...R...#|..E.
0010 00 29 15 2b 40 00 80 06 62 50 c0 a8 01 01 c0 a8  .)/@...bP.....
0020 01 02 04 76 00 17 a8 64 9c b4 11 74 59 af 50 18  ...v...d...tY.P.
0030 00 fe 22 b0 00 00 54                                     .."....
0000 00 30 a7 02 1c 52 00 1d 92 9c 23 7c 08 00 45 00  .0...R...#|..E.
0010 00 29 15 2e 40 00 80 06 62 4d c0 a8 01 01 c0 a8  .)/@...bM.....
0020 01 02 04 76 00 17 a8 64 9c b5 11 74 59 b0 50 18  ...v...d...tY.P.
0030 00 fe 35 ae 00 00 43                                     ..5...
0000 00 30 a7 02 1c 52 00 1d 92 9c 23 7c 08 00 45 00  .0...R...#|..E.
0010 00 29 15 2f 40 00 80 06 62 4c c0 a8 01 01 c0 a8  .)/@...bL.....
0020 01 02 04 76 00 17 a8 64 9c b6 11 74 59 b1 50 18  ...v...d...tY.P.
0030 00 fe 2d ac 00 00 49                                     ..-...
0000 00 30 a7 02 1c 52 00 1d 92 9c 23 7c 08 00 45 00  .0...R...#|..E.
0010 00 29 15 31 40 00 80 06 62 4a c0 a8 01 01 c0 a8  .)/@...bJ.....
0020 01 02 04 76 00 17 a8 64 9c b7 11 74 59 b2 50 18  ...v...d...tY.P.
0030 00 fe 2a aa 00 00 4e                                     ..*...
    
```

Fig. 5. A Telnet session highlighting the cleartext authentication with the synchrophasor

An attacker may sniff all the passwords and then use them to gain complete access to the synchrophasor. With this kind of privileges an attacker would be enabled to modify both the synchrophasor's configurations and the acquired phasor measurements.

**Passwords Management and Maintenance.** The accounting system of the device has a hierarchical structure: there aren't formal users in the system but the only existing account may be granted different level of privileges using different passwords for each level. To access to the higher level privileges the user must escalate through all the lower levels by sequentially providing the appropriate password. The privileges are granted for a limited period of time that is configurable, with a default value set to 5 minutes. After this period of time the privileges are withdrawn and the user is brought back to the lowest privileges level. The system is provided with some protection against password brute forcing since after three consecutive failed attempts the access is forbidden for 30 seconds and an alarm is raised. The system allows to exclude or limit remote access from certain ports.

*Weak policies on password selection and maintenance:* the main security weaknesses of the passwords management and maintenance subsystem are:

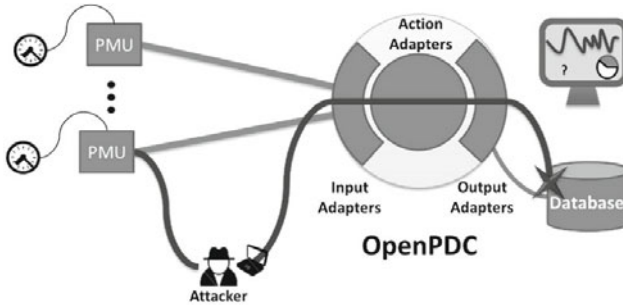
1. The default passwords are very common and simple alphabetic strings which are prone to dictionary attacks.
2. Passwords are editable but no constraints is given for the strength of new passwords. Unwary users are enabled to set very weak or guessable passwords.
3. Multiple levels can share a common password. Since users are usually lazy when dealing with passwords they could use the same password for all the level of privileges and often they will do.
4. Passwords can be totally disabled by hardware intervention by tampering the front panel and setting off a jumper.

**Synchrophasors Communication with the PDC Application.** As we highlighted in section 2, the C37.118 protocol does not provide encryption and thus is not inherently protected from eavesdropping. Any user accessing a node that is in between the two endpoints of the C37.118 communication (the synchrophasor and the PDC application) is capable of reading all the messages exchanged that contain measurements and configuration information. Moreover no authentication or other means of verification are used to identify the source of the messages.

*Channel reliability and integrity weaknesses:* as depicted in Fig. 6 this approach is prone to man-in-the-middle attacks: a malicious eavesdropper can easily read the message exchanged between one of the PMUs and the PDC for a certain period of time in order to learn the relevant parameters of the communication. Then he can either modify the communication parameters as they flow or impersonate the PMU by accurately forging all the subsequent messages.

**PDC Implementation.** We have analyzed the code provided in the OpenPDC's C37.118 input adapter that was used as input point for our PDC application.

We have discovered that the content of the messages was not verified by the input adapter and that it was possible to include any string in the text fields of the C37.118 messages without producing errors on the adapter. Moreover no sanitization was performed in the other adapters allowing the forged content to reach database used for the storage of the Phasor measurements.



**Fig. 6.** An attacker compromising an OpenPDC application

*Lack of input validation and sanitization:* the PDC application is capable of receiving data streams from many different PMUs deployed across the monitored smart grid and the application stores the data from each source PMU in a different table. The number of source PMUs can vary during the operation of the PDC application as new synchrophasors are installed and connected the monitoring system. Since the C37.118 protocol is a smart protocol it is expected that the PMU and the PDC exchange some configuration messages before the PMUs starts sending its measurements. The configuration messages announce the identity of the PMU to the PDC and provide details about the structure and the type of the messages that will follow. For this reason the application uses this information obtained from the PMU's configuration messages to create a new table in the database to store the data from the new PMU. The new table is created according to the information provided by the PMU in the configuration messages: the name of the table is generated using the identifier associated to the PMU and the structure of the records of the table are created according to the data structure that the PMU is supposed to send.

In this situation the attacker can leverage the unjustified trust that the application designer has put in the source of the messages it receives (see section 3.2) to inject SQL code in the application's database. The attacker may forge the PMU identification name at his will and the application will use that name to generate the name of the new table in the database that will be used to store the measurements from that PMU. The attacker can perform an SQL Injection attack to the application simply attaching some SQL code to the identification name it provides to the application and that SQL code will be executed on the

database. The possible malicious exploitations of this attack are countless and we describe some in the following.

The more immediate exploitation could be the deletion of a whole table containing the measurement of one of the PMUs. The attacker can add a “DROP table\_name;” command to its name leading to the complete deletion of a measurements table. This type of attack could be used repeatedly to erase all the collected data from all the PMUs and thus preventing any assessment of the health status of the grid.

Another possible exploitation could be based on injecting “DELETE” commands. In this way the attacker could selectively erase only some records from the database. This is a much more stealthy approach as it doesn’t completely remove all the stored data but only those records that the attacker wants to hide. For instance a terrorist attacker could use this selective deletion to erase evidences of a physical attack brought to the power grid without completely taking down the database which would probably alert the operators.

A smarter attacker could use a more subtle approach: injecting well-crafted SQL “ALTER” commands she can change the names of the tables containing the measurements of other PMUs. In this way she can smartly swap the names of different tables logically assigning measurements collected from one PMU to another PMU and so deceive the operator that is monitoring the grid. As we described in Section 2 the phasor measurements are used not only to foresee but also to locate possible malfunctions and blackouts in the grid through triangulation. Using a very finely tuned orchestration of the attack the application can be tricked so that it incorrectly locates the position of an upcoming malfunction. The operator will react to a dangerous situation basing his planning on information that was maliciously modified by the attacker and thus focusing his attention on the wrong area of the grid. A possible consequence of an attack that follows this approach is that a technical team is sent to a specific position in the grid to solve a problem while that problem is being generated in completely different part of the grid.

## 4 Conclusions

In this paper we have conducted a security analysis of two key technologies which enable data collection in Power Grids, namely synchrophasor devices and Phasor Data Concentrators. The study has been conducted on a commercial product by a major vendor (as far as the synchrophasor is concerned), and on a widely used open source product (as far as the Phasor Data Concentrator is concerned). We have used a simplified - yet realistic - testbed, and we have conducted a penetration testing campaign against the two aforementioned components. We have collected evidence proving that state of the art components for building smart grid data collection infrastructures have several vulnerabilities, some of which can be easily exploited if proper measures are not taken and additional protection devices are not integrated in the system.

This is a major security risk, since attacks to Power Grids may dramatically impact the operation of virtually all human activities.

Our analysis has been conducted on a simplified yet realistic scenario, which is in many respects similar to the setup which is being used in the development of the largest Phasor Network actually deployed, namely NASPInet, operated by the North American Synchrophasor Initiative (NASPI [18]).

In our study we have highlighted how even modern, cutting edge technologies for smart grid data collection may be affected by traditional security issues such as lack of encryption of the communication channels, lack of input validation and sanitization, weak password policies, and man-in-the-middle and dictionary attacks.

In accordance with the findings of some recent reports about attacks targeting critical infrastructures ([4], [5]) we have proved that some of the most common attacks from the domain of classic IT security may well affect critical infrastructures, and the level of risk will probably become higher and higher in the future.

With this work we want to raise the awareness of practitioners about the existing threats, and emphasize the need for increased attention in the design, development, and deployment phase of a smart grid data collection network. As a final comment, we want to emphasize that it is crucial that developers of critical applications take security in due account. In particular, the training of practitioners with a strong security background is a necessary pre-requisite to face the challenges the increasing adoption of Information Technology (IT) in Smart Grids in particular and in Critical Infrastructures in general.

**Acknowledgments.** The research leading to these results has received funding from the European Commission within the context of the Seventh Framework Programme (FP7/2007-2013) under Grant Agreement No. 225553 (INSPIRE Project), Grant Agreement No. 248737 (INSPIREINCO Project) and Grant Agreement No. 257475 (MANagement of Security information and events in Service Infrastructures, MASSIF Project). It has been also supported by the Italian Ministry for Education, University, and Research (MIUR) in the framework of the Project of National Research Interest (PRIN) “DOTS-LCCI: Dependable Off-The-Shelf based middleware systems for Large-scale Complex Critical Infrastructures”.

## References

1. IT and Operational Technology: Convergence, Alignment and Integration, Gartner (February 2011), <http://www.gartner.com/it/page.jsp?id=1590814> (last accessed 30/05/2011)
2. Beech E.: Cyberspies penetrate electrical grid: report. Reuters top ten news stories (April 2009), <http://www.reuters.com/80/article/topNews/idUSTRE53729120090408> (last accessed 30/05/2011)
3. Details of the first-ever control system malware, cnet, [http://news.cnet.com/8301-27080\\_3-20011159-245.html](http://news.cnet.com/8301-27080_3-20011159-245.html) (last accessed 30/05/2011)

4. McAfee, Global Energy Cyberattacks: Night Dragon (February 2011)
5. Symantec Intelligence Quarterly Report: October- December - Targeted Attacks on Critical Infrastructures (December 2010)
6. 1344 IEEE Standard for Synchrophasors for Power Systems, IEEE (1995)
7. openPDC, <http://openpdc.codeplex.com/>
8. Postel J. B., Reynolds J. K.: TELNET Protocol Specification, (RFC854). IETF Network Working Group (May 1983)
9. Postel J. B., Reynolds J. K.: File Transfer Protocol (FTP) (RFC959). IETF Network Working Group (October 1985)
10. Zhang J., Gunter C. A.: IEC 61850 - Communication Networks and Systems in Substations: An Overview of Computer Science, University of Illinois at Urbana-Champaign
11. DNP3 (Distributed Network Protocol), <http://www.dnp.org/>
12. Phadke, A.: Synchronized phasor measurements in power systems, vol. 6(2), pp. 10–15 (April 1993)
13. Dagle, J.: The north american synchrophasor initiative (naspi). In: 2010 IEEE Power and Energy Society General Meeting, pp. 1–3 (2010)
14. The North American Electric Reliability Corporation (NERC), <http://www.nerc.com/>
15. Zhong, Z., Xu, C., Billian, B., Zhang, L., Tsai, S.-J., Connors, R., Centeno, V., Phadke, A., Liu, Y.: Power system frequency monitoring network (fnet) implementation, vol. 20(4), pp. 1914–1921 (2005)
16. Gardner, R., Liu, Y.: Fnet: A quickly deployable and economic system to monitor the electric grid. In: 2007 IEEE Conference on Technologies for Homeland Security, pp. 209–214 (May 2007)
17. Tennessee Valley Authority (TVA), <http://www.tva.gov>
18. NASPI Network (NASPINet), <http://www.naspi.org/naspinet.stm>
19. Secure Communications, Schweitzer Engineering Laboratories, Inc., <http://www.selinc.com/securecommunications/> (last accessed 30/05/2011)
20. Cybersecurity, Schweitzer Engineering Laboratories, Inc., <http://www.selinc.com/cybersecurity/> (last accessed 30/05/2011)
21. Stewart J., Maufer T., Smith R., Anderson C., Ersonmez E.: Synchrophasor Security Practices, Schweitzer Engineering Laboratories, Inc., <http://www.selinc.com/WorkArea/DownloadAsset.aspx?id=8502> (last accessed 30/05/2011)
22. Smith R.: Cryptography Concepts and Effects on Control System Communications, Schweitzer Engineering Laboratories, Inc., <http://www.selinc.com/WorkArea/DownloadAsset.aspx?id=5200> (last accessed 30/05/2011)
23. Hurd S., Smith R., Leischner G.: Tutorial: Security in Electric Utility Control Systems, Schweitzer Engineering Laboratories, Inc., <http://www.selinc.com/WorkArea/DownloadAsset.aspx?id=3491> (last accessed 30/05/2011)
24. Mix S.: Primer Discussion on Cyber Security: What do the CIP Standards Mean for SynchroPhasors in the future?, North American Electric Reliability Corporation (NERC), [http://www.naspi.org/meetings/workgroup/2009\\_february/presentations/nerc\\_cyber\\_security\\_mix\\_20090205.pdf](http://www.naspi.org/meetings/workgroup/2009_february/presentations/nerc_cyber_security_mix_20090205.pdf) (last accessed 30/05/2011)



25. Introduction to NISTIR 7628, Guidelines for Smart Grid Cyber Security, The Smart Grid Interoperability Panel Cyber Security Working Group, [http://www.nist.gov/smartgrid/upload/nistir-7628\\_total.pdf](http://www.nist.gov/smartgrid/upload/nistir-7628_total.pdf) (last accessed 30/05/2011)
26. Braendle M.: Cyber security - effectively and efficiently tackling the challenges ahead, ABB, <http://www.abb.com/cawp/seitp202/a6a42387602e83828525784200766310.aspx> (last accessed 30/05/2011)
27. Hadley M.D., McBride J.B., Edgar T.W., O'Neil L.R., Johnson J.D.: Securing Wide Area Measurement System, Pacific Northwest National Laboratory, [http://www.oe.energy.gov/DocumentsandMedia/Securing\\_WAMS.pdf](http://www.oe.energy.gov/DocumentsandMedia/Securing_WAMS.pdf) (last accessed 30/05/2011)
28. Larsson, S., Danell, A.: The black-out in southern Sweden and eastern Denmark September 23, 2003. In: Power Systems Conference and Exposition, pp. 309–313 (2006)
29. Corsi, S., Sabelli, C.: General blackout in Italy Sunday September 28, 2003. Power Engineering Society General Meeting, 2, 1691–1702 (2004)
30. U.S. - Canada Power System Outage Task Force, Final report on the august 14, 2003 blackout in the united states and canada: Causes and recommendations (April 2004)

# Modeling Aircraft Operational Reliability

Kossi Tiassou<sup>1,2</sup>, Karama Kanoun<sup>1,2</sup>, Mohamed Kaâniche<sup>1,2</sup>,  
Christel Seguin<sup>3</sup>, and Chris Papadopoulos<sup>4</sup>

<sup>1</sup> CNRS, LAAS, 7 Avenue du Colonel Roche, F-31077 Toulouse Cedex 4, France

<sup>2</sup> Université de Toulouse, UPS, INSA, INP, ISAE, UT1, UTM, LAAS,  
F-31077 Toulouse Cedex 4, France  
firstname.lastname@laas.fr

<sup>3</sup> ONERA/DCSD/CD, 2 Avenue Edouard Belin, 31055 Toulouse Cedex 4, France  
christel.seguin@onera.fr

<sup>4</sup> AIRBUS Operations Ltd., New Filton House, Golf Course Lane, Filton, Bristol, BS99 7AR,  
United Kingdom  
Chris.Papadopoulos@Airbus.com

**Abstract.** The success of an aircraft mission is subject to the fulfillment of some operational requirements before and during each flight. As these requirements depend essentially on the aircraft system components and the mission profile, the effects of failures can be very significant if they are not anticipated. Hence, one should be able to assess the aircraft operational reliability with regard to its missions in order to be able to cope with failures. This paper addresses aircraft operational reliability modeling to support maintenance planning during the mission achievement. We develop a modeling approach to represent the aircraft system operational state taking into account the mission profile as well as the maintenance facilities available at the flight stop locations involved in the mission. It is illustrated using Stochastic Activity Networks (SANs) formalism, based on an aircraft subsystem.

**Keywords:** operational reliability, model-based assessment, aircraft system, maintenance planning.

## 1 Introduction

With the increasing interest in air transportation and the competitive market aircraft operators have to deal with, aircraft operational disruptions become a key concern in the aviation field. In order to avoid economical losses due not only to inoperability but also to customer dissatisfaction, airlines need to anticipate on the events that may disrupt the achievement of their aircraft missions. Aircraft missions are achieved in compliance with operational requirements depending principally on the current operational state of the aircraft system components and the mission profile. Thus, an attention must be paid to the effects of the aircraft system component failures and the corresponding maintenance actions. Failures that may disturb the achievement of the aircraft mission must be handled with adequate corrective actions. However, the ability to promptly cope with these failures depends on the location where they occur.

Maintenance facilities are not the same at all airports. Generally, airlines have more facilities at their main base than at the other airports. Therefore, the maintenance resources must be adapted to the aircraft missions. The issue is to have an assessment method that can support mission assignments and maintenance activities forecasting. Model-based dependability assessment is well suited to support this process.

Our work aims at developing an assessment approach, based on dependability modeling, that makes it possible to continuously assess the ability to keep operating up to a given time or location. The model will be used while planning the missions and during their achievement. To plan the mission, the model can be used to estimate the period of time during which the aircraft system can be operated without reaching adverse states. This allows to determine the mission profile the aircraft must be assigned. Once a mission is assigned to the aircraft, the model can be used during its achievement to assess the ability to succeed in continuing on the remaining part of the mission. The model can also support maintenance activities planning. The best maintenance strategy can be determined comparing the probabilities to accomplish the missions considering different alternatives.

To cover these issues, the model should be able to take into account the various situations in which it may be used. Our approach consists in developing generic stochastic sub models that can be dynamically updated and configured to represent the current state of the aircraft, with regard to the mission to achieve.

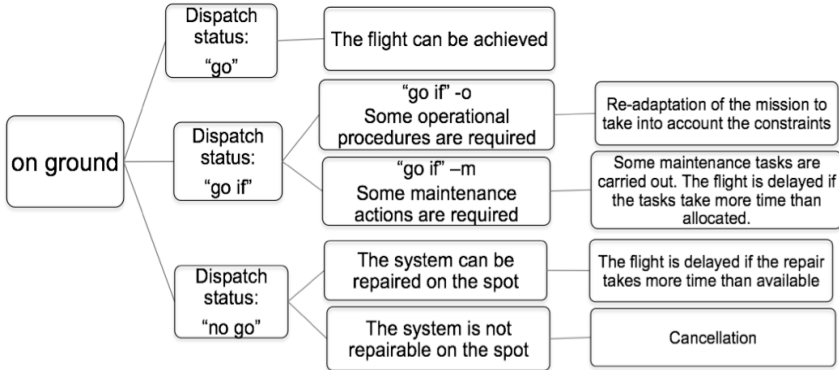
The remainder of this paper is structured as follows. Section 2 describes how aircraft missions are carried out together with the verification of the operational requirements fulfillment. Section 3 presents some related works. Section 4 is devoted to the modeling approach, which is implemented in section 6 using an aircraft subsystem as example. The subsystem is presented in section 5. Section 7 presents an example of evaluation result. Finally Section 8 concludes the paper.

## 2 Description of Mission Achievement

The achievement of the mission is such that each flight is followed by a stop where the aircraft is prepared for next flight. The preparation for the next flight consists of routine maintenance activities, cabin cleaning, catering, baggage and cargo processing, and passenger boarding.

At each stop, the aircraft is inspected and the discrepancies that are reported during the previous flight are checked. If a component is found inoperative, a dispatch decision is taken regarding the next flight. The flight captain refers to an approved document called *Minimum Equipment List* (MEL) where the components are listed with the status “go”, “go if” or “no go”:

- The “go” status is the case where the aircraft can fly with the component failed.
- The “go if” status allows the flight provided some conditions (on other components, operational performance and maintenance activities) are fulfilled. This includes a given deadline to repair the component.
- The “no go” status prevents the aircraft from flying. The failed component must be repaired before any flight.



**Fig. 1.** Dispatch status outcomes

The dispatch is allowed if there is no “no go” and all “go if” conditions are acceptable. When the aircraft does not meet the dispatch requirements following a failure, maintenance activities are initiated in order to solve the problem. The magnitude of the failure effect depends thus on the ability to solve the problem at the considered location before the planned departure time. Actually, the flight is considered delayed only after exceeding a given tolerable time frame. Figure 1 summarizes the possible outcomes of the dispatch decision.

When the dispatch is allowed, the aircraft can depart after passenger, cargo and the other ground service processing. Then, the flight begins by the taxiing of the aircraft to runway where the takeoff is initiated. During this period or even after the takeoff, the flight can be aborted as a result of a critical failure. The aircraft then returns back to the departure airport. Actually, during the entire flight, it may be diverted if the aircraft capability is degraded. Procedures, stated in the Flight Manual (FM), the Flight Crew Operating Manual (FCOM) or the Quick Reference Handbook (QRH), are used to determine whether the flight must be diverted or not [1].

The adverse situations while operating an aircraft are operational interruptions, namely flight delays, cancellations, in flight turn-back and diversions. Delays and cancellations occur on ground, while turn-back and diversion occur in flight.

### 3 Related Work

To the best of our knowledge, aircraft operational reliability modeling has been seldom addressed in the literature. The studies carried out are rather concentrated on safety aspects (see [3, 10, 14] for instance), and most works about operational reliability are for design enhancement purpose [2, 13]. In [6], the issues of delays and safety in airline maintenance are addressed. A probabilistic risk analysis model is developed in order to quantify the effect of airlines maintenance policies on their aircraft operability. A decision support approach to maintenance planning is presented in [7]. That is, thanks to redundancy, the aircraft can continue operating with some equipment inoperative, however, it is time limited and can increase the risk of

occurrence of an interruption. The approach proposes a method to schedule the repairs taking into account some optimization criteria: cost, remaining useful life and operational risks. The approach is based on generating alternatives on which is defined a utility function. It is worth noting that the work does not account for reliability measure. It uses the reliability measure as input. In [1], the operational consequences of system failures are studied using event tree analysis. The paper discusses the possible consequences of failures taking into account the flight phase during which they have occurred. A modeling approach based on the fault trees of the targeted aircraft system is presented in [2], together with a computing algorithm to estimate the bounds of the considered probability measure. The approach considers a series of flight cycles and provides a means to evaluate the probability of occurrence of one of three events at each cycle: “No Go dispatch”, “Accepted Degraded Mode” which corresponds to the case where a “go if” occurs and the airline accepts to perform the corresponding tasks, “Refused Degraded Mode” which is a “go if” that is not accepted by the airline. Note that the paper only deals with dispatch events and does not consider in-flight operational consequences. The probability of failure of more than one component during a flight is also neglected.

Concerning modeling aspects, the problem is generally categorized, with regard to the system, as a Phase Mission System (PMS) problem. Mura and Bondavalli [8] analyze the PMS and present a dependability modeling approach. It is shown that, under some given conditions, the model can be processed using an analytical method. Chew et al. [9] address the problem using the concept of maintenance-free operating periods; the system evolves through a series of phases with no possible maintenance. The developed model is solved by simulation.

Of all these works, none is aimed directly at modeling aircraft operability during its missions’ achievement. The closest works [1, 2] are carried out for long-term operational dependability analysis and are based on event trees and fault trees. This paper addresses aircraft operational reliability using stochastic state-based models. Our work is intended to develop a reliability model that one can use to cope with operability issues during aircraft missions’ achievement. The modeling approach is presented in the following section.

## 4 Modeling Approach

As presented in section 2, the aircraft has to fulfill some operational requirements (dispatch requirements) before flying and some requirements (in-flight requirements) during the flight. We distinguish:

- the minimal system requirements given by MEL (Min\_Sys\_Req) that are independent of the mission profile and which must be fulfilled in order to operate the aircraft whatever the mission.
- the requirements (M\_Prof\_Req) that are specific to the mission profile. These include the mission dependent dispatch requirements and the requirements in flight. They are composed of the requirements specific to the flights composing the mission.

The evaluation is based on the fulfillment of these requirements. The objective is to evaluate the probability of occurrence of the adverse events that may lead to an operational interruption. We distinguish two reliability measures:

- While planning a mission, the aircraft system reliability (SR) is evaluated with regard to *Min\_Sys\_Req* in order to determine the maximum number of flight hours that can be achieved without maintenance. This is used to determine the length of the mission or to plan maintenance activities.
- Once a mission is assigned to the aircraft and during its achievement, the reliability measure (MR) which corresponds to the probability to achieve the mission without an operational interruption, is evaluated with regard to *Min\_Sys\_Req* and *M\_Prof\_Req* in order to determine whether a preventive action must be initiated or not.

### 4.1 Structure of the Model

Figure 2 shows the overall structure of the model composed of four levels.

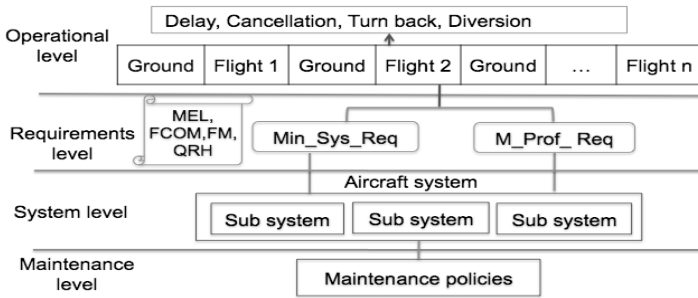


Fig. 2. Overall structure of the model

**Operational Level:** it represents the succession of periods during which the aircraft is either flying or on ground.

**Requirements Level:** it consists of the aggregation of the requirements from the potential contributors to the continuity of the mission. These requirements are the representation of *Min\_Sys\_Req* and *M\_Prof\_Req*. These requirements are formulated as complements of Boolean expressions, representing the different combinations leading to an operational interruption.

**System Level:** It describes the aircraft system. The system is decomposed into subsystems and atomic components according to its design logic or its functions. This level describes the components failure scenarios.

**Maintenance Level:** It describes the maintenance possibilities at the various airports involved in the mission profile. It is intended to represent the predefined maintenance policies related to the airports. This has an impact on the repair time of the system components at a given stop. The maintenance activity itself is modeled at the system level.

We build generic sub models corresponding to each of the main levels in the above structure. The composition of these sub models will form an initial model, which is to be configured and parameterized with online data in order to obtain the overall model.

The approach can be implemented using an appropriate formalism. In this paper, we consider the Stochastic Activity Networks (SANs) formalism and the associated Möbius tool [5], which provide compositional operators that are convenient to master the complexity of the model. A brief description of SANs is given in the followings.

### 4.2 SANs Formalism

Stochastic activity networks are an extension of Petri nets (PN). SANs consist of four primitive objects: places, activities, input gates, and output gates. Activities are the equivalent of transitions in PN. They are either timed or instantaneous. Timed activities have durations and a time distribution function. Instantaneous activities represent actions that complete immediately when enabled. Input gates are used to control the enabling of activities and define the marking changes that will occur when an activity completes. Each input gate is defined with an enabling predicate and a function. Output gates are like input gates and are used to change the state of the system when an activity completes. An output gate is defined only with a function. The function defines the marking changes that occur when the activity completes. Input gates and output gates are represented graphically as triangles (see Figure 3).

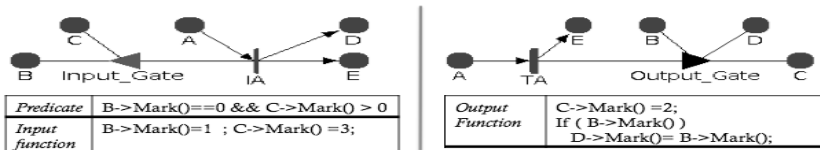


Fig. 3. Input and output gates

An activity is enabled when the predicates of all input gates connected to the activity are true, and all places connected to incoming arcs contain tokens, i.e., have non zero markings. Once enabled, the activity samples its delay distribution function to determine the time delay before the activity fires. When the activity fires, it updates the state of the model by subtracting tokens from places connected by incoming arcs, adding tokens to places connected by outgoing arcs, and executing the functions in input and output gates.

Möbius allow the construction of composed models. Indeed, for a large system, it may be helpful to compose the overall model based on sub-models that have less complexity. This is feasible using the Join and Replicate operators. The Join operator combines several models sharing some state variables. The Replicate operator is used to create copies of models; the copies are combined into a global model. The copies may hold some state variables in common. A Join node may have other Joins, Replicates, or other sub models defined as its children.

This formalism is used to develop a case study implementing the modeling approach. The case study concerns a subsystem that controls one of the movable

surfaces of the aircraft [11], referred to as CMS in the rest of the paper. The subsystem is described in the following section.

### 5 CMS Presentation

The subsystem is composed (see Figure 4) of three primary computers (P1, P2, P3), a secondary computer S1, three servo-controls (ServoCtrl\_G, ServoCtrl\_B and ServoCtrl\_Y), a backup control module (BCM) and two backup power supplies (BPS\_B and BPS\_Y).

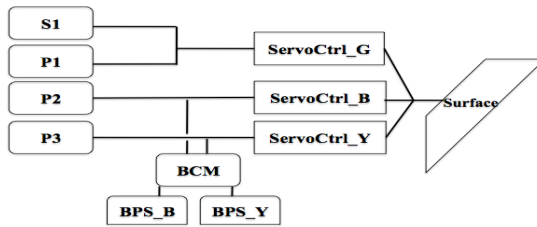


Fig. 4. The subsystem

The computers are connected to the servo-controls, which move the surface. S1 and P1 are connected to the servo-control ServoCtrl\_G, P2 is connected to ServoCtrl\_B, and P3 is connected to ServoCtrl\_Y. The connection between a computer and a servo-control form a control line that can act on the surface. We have:

- P1 control line (PL1): formed by the connection between P1 and ServoCtrl\_G,
  - P2 control line (PL2): formed by the connection between P2 and ServoCtrl\_B,
  - P3 control line (PL3): formed by the connection between P3 and ServoCtrl\_Y,
  - S1 control line (SL): formed by the connection between S1 and ServoCtrl\_G.
- We have also Backup control line (BCL), which is based on BCM, BPS\_B, BPS\_Y, ServoCtrl\_Y and ServoCtrl\_B.

Initially the secondary computer S1, the backup control module BCM and the backup power supplies BPS\_B and BPS\_Y are inhibited. The surface is then controlled by the three primary control lines (PL1, PL2, PL3). When the three primary control lines fail, S1 is activated and the system switches to SL. If the latter also fails, BCM, BPS\_B and BPS\_Y are activated enabling the backup control. Therefore, three control modes can be distinguished: the primary control (PC), the secondary control (SC) and the backup control (BC). Figure 5 summarizes the control modes.

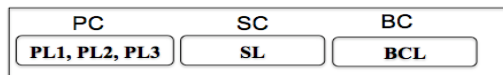


Fig. 5. The control modes and associated control lines



**Related Operational Requirements:** According to [4]<sup>1</sup>, the failure of P2, ServoCtrl\_G, ServoCtrl\_Y, ServoCtrl\_B, BCM, BPS\_B or BPS\_Y leads to “no go” status. P1, P3 and S1 are “go if” items with “go if” conditions stated respectively at sections MMEL 27-93-01-1, MMEL 27-93-01-3 and MMEL 27-94-01-1 of the document. The dispatch conditions resulting from these sections are respectively  $(P1=ok) \vee (S1=ok \wedge P3=ok)$ ;  $(P3=ok) \vee (S1=ok \wedge P1=ok)$ ;  $(S1=ok) \vee (P1=ok \wedge P2=ok \wedge P3=ok)$ <sup>2</sup>. In the three cases, the failed component must be repaired before the deadline of 10 days. These conditions are not dependent on any mission profile. Therefore, they are part of Min\_Sys\_Req.

$$\begin{aligned} \text{Min\_Sys\_Req} = & ( P2 =ok \wedge \text{BCM} =ok \wedge \text{BPS\_B} =ok \wedge \text{BPS\_Y} =ok \wedge \\ & (P1 =ok \vee (S1 =ok \wedge P3 =ok)) \wedge \text{ServoCtrl\_G} =ok \wedge \\ & (P3 =ok \vee (S1 =ok \wedge P1 =ok)) \wedge \text{ServoCtrl\_Y} =ok \wedge \\ & (S1 =ok \vee (P1 =ok \wedge P3 =ok)) \wedge \text{ServoCtrl\_B} =ok ). \end{aligned} \quad (1)$$

Using the control lines previously defined, this expression becomes:

$$\begin{aligned} \text{Min\_Sys\_Req} = & ( PL2 =ok \wedge (PL1 =ok \vee (PL3 =ok \wedge SL =ok)) \wedge \\ & (PL3 =ok \vee (PL1 =ok \wedge SL =ok)) \wedge \text{BCL} =ok \wedge \\ & (SL =ok \vee (PL1 =ok \wedge PL3 =ok)) ). \end{aligned} \quad (2)$$

There is no operational requirement related to the subsystem in the FCOM.

## 6 The Model

The model is the aggregation of sub models corresponding to the levels presented in section 4.1. Note that only one subsystem is considered here. Due to space limitations only the operational, requirements and system levels sub models are shown.

### 6.1 The System Level Sub Model

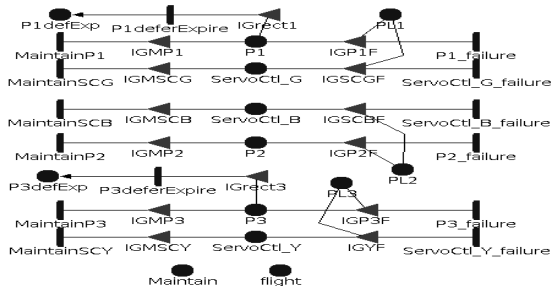
The system level sub model consists of the representation of CMS. To simplify its presentation, it is decomposed into three sub models corresponding to the control modes given in Figure 5. In all the three sub models, places representing the subsystem’s components functional state (ok or failed) are named after these components. Activities named *xxx\_failure* represent failures events. Their enabling is conditioned by the presence of a token in place *flight*. Activities *Maintainxx* represent maintenance activities and their enabling is conditioned by the presence of a token in place *Maintain*. Places *flight* and *Maintain* represent respectively whether a flight is ongoing or not, and whether a maintenance period is ongoing or not. Their markings are controlled by the operational level sub model (Figure 10). For clarity purpose, some places involved in the predicate or function of the input gates are not explicitly linked to them; this is allowed by SANs.

<sup>1</sup> [4] is actually a Master MEL(MMEL). MELs result from the completion of MMELs with airline specific policies and are not public documents. MMELs are established by the aircraft’s manufacturer.

<sup>2</sup> These are not actually the full conditions, we only consider the conditions related to the components involved in the subsystem described.

**Primary control (PC)** model is given in Figure 6. The transitions representing the maintenance activities (*Maintain<sub>xx</sub>*) are at the left side and the failure events (*xxx<sub>failure</sub>*) at the right side of the places. Their associated input gates control their firings. For example *IGP1F* and *IGMP1* are defined as follows:

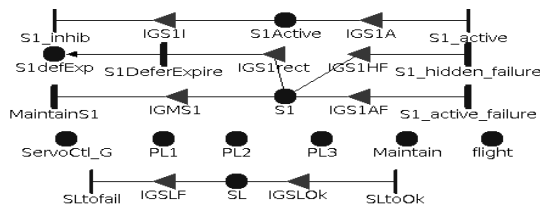
*IGP1F* Predicate :  $P1 \rightarrow \text{Mark}() \ \&\& \ \text{flight} \rightarrow \text{Mark}()$  Function :  $P1 \rightarrow \text{Mark}()=0; PL1 \rightarrow \text{Mark}()=0;$   
*IGMP1* Predicate :  $P1 \rightarrow \text{Mark}()=0 \ \&\& \ \text{Maintain} \rightarrow \text{Mark}()$  Function :  $P1 \rightarrow \text{Mark}()=1;$   
 if ( $\text{ServoCtrl\_G} \rightarrow \text{Mark}()$ )  $PL1 \rightarrow \text{Mark}()=1; P1\text{defExp} \rightarrow \text{Mark}()=0;$



**Fig. 6.** PC sub model

Transition *P1(3)deferExpire* represents the expiration of the deadline before which the computer must be repaired after being failed. This doesn't concern P2 since its status is "no go". Places *PL<sub>i</sub>* represents the state of the lines *PL<sub>i</sub>*. *PL<sub>i</sub>* is marked when *P<sub>i</sub>* and the corresponding *ServoCtrl<sub>x</sub>* in the line are marked. The markings of places *Maintain* and *flight* are used in the predicates of the input gates to enable the failure and maintenance activities as explained above.

**Secondary control (SC)** is represented in Figure 7. Place *S1Active* represents the activation state of S1. That is when PC fails, the instantaneous activity *S1<sub>active</sub>* fires in order to mark place *S1Active*, representing the failover to SL. *S1<sub>inhib</sub>* models the inhibition event. It fires when one of *PL1*, *PL2* and *PL3* becomes marked again, removing the token from *S1Active*. *PL1*, *PL2* and *PL3* are shared with the PC sub model, which controls their makings. They are only used in the predicates of *IGS1A* and *IGS1I* to express whether PC is failed or not. *S1<sub>hidden\_failure</sub>* and *S1<sub>active\_failure</sub>* model respectively the failure events of S1 while inhibited and activated. *SL* represents the functioning state of the secondary control line. It holds when *S1* and *ServoCtrl<sub>G</sub>* hold. *ServoCtrl<sub>G</sub>* is shared with PC sub model.



**Fig. 7.** SC sub model

The Backup control (BC) model is depicted in Figure 8. *BPS\_BActive* and *BPS\_YActive* describe the inhibition and the activation of *BPS\_B* and *BPS\_Y*. That is, when *PL1* and *SL* are inoperative, *BPS\_B* and *BPS\_Y* are activated to supply power to *BCM*. They are inhibited when *PL1* or *SL* is operative. *BPS\_BActive* and *BPS\_YActive* are updated by their associated instantaneous transitions, which fire according to the marking of *PL1* and *SL* as described above.

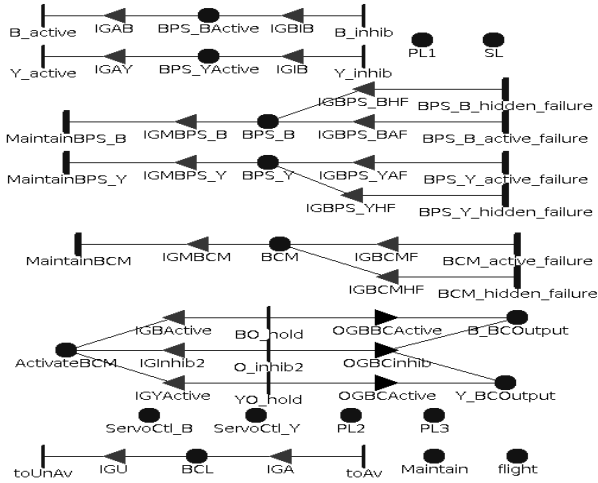


Fig. 8. BC sub model

*ActivateBCM* represents the use of the *BCM* to control the surface; when none of the primary and secondary control lines is operative and *BPS\_B* or *BPS\_Y* supply the *BCM* with electric power, the *BCM* is activated to attempt to control the surface via *ServoCtrl\_Y* or *ServoCtrl\_B*. *B\_YCoutput* and *B\_BCoutput* represent respectively the use of power from *BPS\_Y* and *BPS\_B*. *BCL* represents the fulfillment of the requirements on the components of the line. It is marked when *BCM*, *BPS\_B*, *BPS\_Y*, *ServoCtrl\_B* and *ServoCtrl\_Y* are marked. Places *Maintain* and *Flight* are shared with the operational level sub model; *PL1*, *PL2*, *PL3*, *ServoCtrl\_B* and *ServoCtrl\_Y* with *PC* sub model; and *SL* with *SC* sub model. Their marking are used as input to the *BC* sub model as they are involved in the activation and inhibition of the *BC*.

As only one subsystem is considered in this case study, the system level sub model corresponds to the composition of *PC*, *SC* and *BC* sub models (see Figure 11).

### 6.2 The Requirement Level Sub Model

Figure 9 shows the aggregation of the requirements fulfillments from the system level sub models.

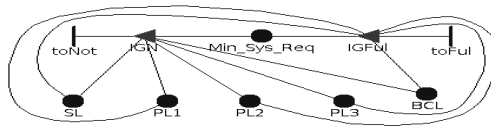


Fig. 9. Requirement level sub model

Place *Min\_Sys\_Req* models the requirements fulfillment. The firings of the instantaneous activities *toFul* and *toNot* update the place according to the condition expressed in section 5 (expression (2)). *Min\_Sys\_Req* is used at the operational level. Places *PL1*, *PL2*, *PL3*, *SL* and *BCL* are shared with the system level sub model. *M\_Prof\_Req* is not represented here due to the fact that the subsystem has no mission profile related requirement. Nevertheless, its representation will be similar.

### 6.3 The Operational Level Sub Model

The operational level sub model is shown in Figure 10. The upper part represents a flight and the lower part represents the activities on ground at a stop. Place *Maintain* is shared with the system level sub model indicating the ongoing of a maintenance period. Place *Req\_fulfilment* is an extended place representing the requirements fulfillment. It should be composed of *Min\_Sys\_Req* and *M\_Prof\_Req*, which are shared with the requirements level sub model. Since no mission profile related requirement is considered here, the share concerns only *Min\_Sys\_Req* at the requirements level. A flight is represented by three phases *Taxing\_to\_Climb*, *In\_Flight* and *Landing*. During the *Taxing\_to\_Climb* phase the flight can be aborted and it can be diverted during the *In\_Flight* phase. The input gates *AbortCondition* and *Diversion\_Condition* represent the conditions under which these interruptions can occur (in-flight requirements fulfillment). The conditions are stated using the marking of *Req\_fulfilment*. Place *flight* indicates whether a flight is ongoing or not. It is shared with the system level sub model.

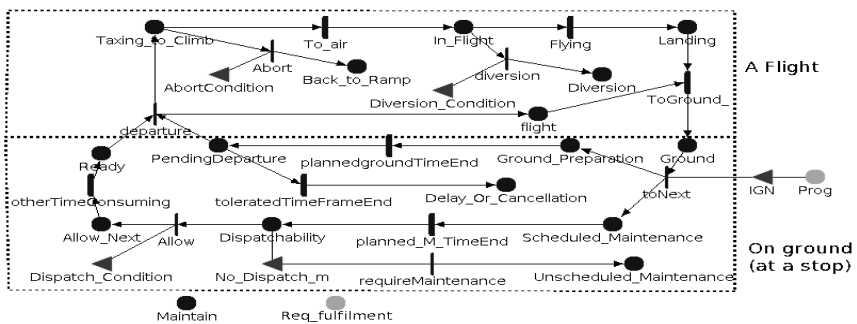


Fig. 10. Operational level sub model

The sub model of a ground period consists of the representation of the preparation for the next flight and the readiness for departure on time. The beginning of the preparation for the upcoming flight is represented by the marking of places *Ground\_Preparation* and *Scheduled\_Maintenance*, stating that the scheduled ground

period is ongoing and the system is under scheduled maintenance (routine check for instance)<sup>3</sup>. When the scheduled maintenance is finished (activity *planned\_M\_TimeEnd* fires), the place *Dispatchability* then holds and the instantaneous activity *Allow* can fire if the dispatch requirements, stated in the predicate of *Dispatch\_Condition*, are fulfilled. Otherwise the instantaneous activity *requireMaintenance* fires if the corrective action requires maintenance tasks (stated by the predicate of *No\_Dispatch\_m*), place *Dispatchability* still holds until the corrective action succeeds (predicate of *Dispatch\_Condition* becomes true) and the flight is allowed. In the current illustration, the dispatch requirements fulfillment consist of testing if the marking of field *Min\_Sys\_Req* in the extended place *Req\_fulfillment* is zero or not. Until then, the scheduled ground duration may have elapsed (firing of activity *plannedgroundTimeEnd* moving the token to place *PendingDeparture*) and the tolerable delay may be running out. A delay or cancellation occurs if the tolerated time to dispatch is exceeded. The timed transition *OtherTimeConsuming* represents the other activities (passengers and baggage processing ...) that may consume time, causing delay. Place *Prog* (at right) is an extended place representing the list of flights to be achieved. The input gate *IGN* indicates whether there is a next flight to achieve or not (end of the mission or not).

### 6.4 The Global Model

The global model results from the composition of the sub models corresponding to the four levels. It is shown in Figure 11.

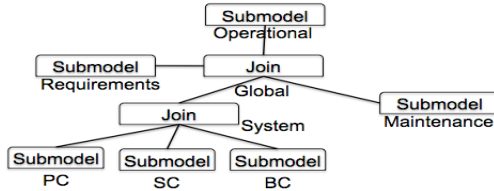


Fig. 11. The global model

## 7 Example of Results

Since the model is intended to be used during the achievement of the mission, the initial markings and the parameters such as the distribution laws of the timed activities are to be set online using online data. In order to provide an example of evaluation, some values of the parameters are assumed here. We assume that all the events represented by timed activities at system level (Figures 6, 7, 8) have exponential distributions, except *P1deferExpire* and *P3deferExpire*, which have deterministic durations. The values of failure rates used for the example are between  $10^{-4}$ /hour and  $10^{-6}$ /hour. For the parameters of the operational level sub model, we consider a mission of 4 flights per day over a week. We assume that the timed activities of the operational level sub model have deterministic durations. Each flight takes 3 hours. The planned duration of a ground period is of 1.5 hour during the day

<sup>3</sup> These tasks are aimed at detecting failures, and not to repair any failed component.

and 7.5 hours at the end of the day (after 4 flights). We evaluate the mission reliability;  $MR(t)$ . For illustration purposes, we consider that the in-flight requirements are the same as the dispatch requirements (*Min\_Sys\_Req*). The mission reliability  $MR(t)$  is the probability to have no tokens in places *Delay\_Or\_Cancellation*, *Back\_to\_Ramp* and *Diversion* of Figure 10. Figure 12 shows the mission reliability considering two initial states of the primary computer P1: P1-OK (P1 is OK at the starting of the mission), and P1-KO (P1 is in failure at the starting of the mission), the other components are assumed to be OK at the starting of the mission.

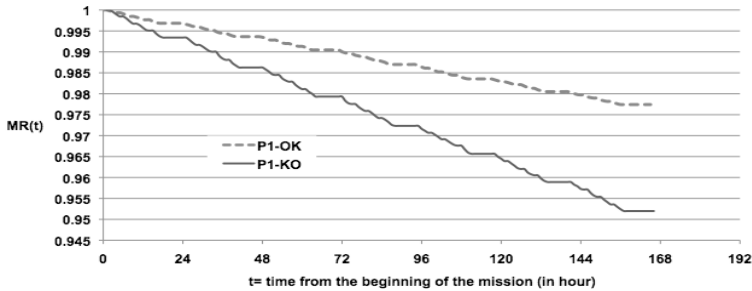


Fig. 12. Mission Reliability

From the evaluation, the time from which the reliability becomes lower than a given threshold can be determined. For example, considering 0.98 as reliability threshold, one has to consider strengthening its ability to maintain after 144h in case of P1-OK and 72h in case of P1-KO. The curves also illustrate a situation where one has to decide on whether it is preferable to defer the maintenance of computer P1, knowing that there is one week remaining mission to achieve. With the assumed parameters, the reliability of the one-week mission will increase from 0.952 to 0.978 if P1 is repaired before the starting of the mission. Other examples of missions and of system reliability measures are given in [12].

## 8 Conclusion

This paper is aimed at developing a model that one can use to assess aircraft operational reliability. The model is intended to be used before and during aircraft mission achievement. A modeling approach has been developed considering aircrafts systems particularities and how the missions are achieved. The proposed model is composed of generic sub-models corresponding to components that may be involved in aircraft operability. An illustration of the modeling approach with SANs formalism has been given using an aircraft subsystem.

The current work is focused on the construction of the initial model that will be used to assess the operational reliability. The model, however, must be updated during the achievement of the missions in order to take into account the current situation during which it will be used. The modeling approach is designed to facilitate these updates. Changes concerning the aircraft system components states and failures distributions will be taken into account in the system level sub model. Missions' update will be managed with the operational level sub model. It is expected that the

system level sub model update will rely on diagnosis and prognosis modules. Data from the flight plans will be used to configure the operational level sub model.

The model update is currently achieved manually. Methods to dynamically integrate the updates and automatically re-assess the reliability, after the occurrence of a major event, are under investigation [15].

## References

1. Ahmadi, A., Soderholm, P.: Assessment of Operational Consequences of Aircraft Failures: Using Event Tree Analysis. In: 2008 IEEE Aerospace Conference, pp. 1–14 (2008)
2. Saintis, L., et al.: Computing in-service aircraft reliability. *International Journal of Reliability, Quality and Safety Engineering* 16(02), 91 (2009)
3. Prescott, D., Andrews, J.: Aircraft safety modeling for time-limited dispatch. In: Proceedings of Annual Reliability and Maintainability Symposium, pp. 139–145 (2005)
4. Master Minimum Equipment List, Airbus A-340-200/300,  
<http://fsims.faa.gov/wdocs/mmel/a340-200-300%20original%2005-30-08.htm>,  
<http://fsims.faa.gov/wdocs/mmel/a340-200-300original05-30-08.pdf>
5. Daly, D., et al.: Möbius: An extensible tool for performance and dependability modeling. In: Schaumnurg, I.L., Haverkort, B.R., Bohnenkamp, H.C., Smith, C.U. (eds.) TOOLS 2000. LNCS, vol. 1786, pp. 332–336. Springer, Heidelberg (2000)
6. Sachon, M., Paté-Cornell, E.: Delays and safety in airline maintenance. *Reliability Engineering & System Safety* 67(3), 301–309 (2000)
7. Papakostas, N., et al.: An approach to operational aircraft maintenance planning. *Decision Support Systems* 48(4), 604–612 (2010)
8. Mura, I., Bondavalli, A.: Markov Regenerative Stochastic Petri Nets to Model and Evaluate Phased Mission Systems Dependability. *IEEE Transactions on Computers* 50(12), 1337–1351 (2001)
9. Chew, S., et al.: Phased mission modelling of systems with maintenance-free operating periods using simulated Petri nets. *Reliability Engineering & System Safety* 93(7), 980–994 (2008)
10. Kehren, C., et al.: Advanced Simulation Capabilities for Multi-Systems with AltaRica. In: Proceedings of the 22nd International System Safety Conference (ISSC), System Safety Society, pp. 489–498 (2004)
11. Bernard, R., et al.: Experiments in model-based safety analysis: flight controls. In: First IFAC Workshop on Dependable Control of Discrete Systems, Cachan (2007),  
[http://sites.google.com/site/pierrebieber/publications/DCDS07\\_FlightControlsModel\\_RB.pdf](http://sites.google.com/site/pierrebieber/publications/DCDS07_FlightControlsModel_RB.pdf)
12. Tiassou, K., Kaâniche, M., Kanoun, K., Seguin, C.: DIANA Operational Reliability — Modelling the Rudder System Using AltaRica and Stochastic Activity Networks, LAAS report No 11001
13. Bineid, M., Fielding, J.P.: Development of an aircraft systems dispatch reliability design methodology. *The Aeronautical Journal* 110(1108), 345–352 (2006)
14. Ramesh, A., et al.: Advanced methodologies for average probability calculation for aerospace systems. In: 26th International Congress of the Aeronautical Sciences (2008)
15. Tiassou, K., Kanoun, K., Kaâniche, M., Seguin, C., Papadopoulos, C.: Operational Reliability of an Aircraft with Adaptive Missions. In: Proceedings of the 13th European Workshop on Dependable Computing - EWDC 2011, p. 9 (2011)

# An Integrated Approach for Availability and QoS Evaluation in Railway Systems

Antonino Mazzeo<sup>1</sup>, Nicola Mazzocca<sup>1</sup>, Roberto Nardone<sup>1,3</sup>, Luca D'Acerno<sup>2</sup>,  
Bruno Montella<sup>2</sup>, Vincenzo Punzo<sup>2</sup>, Egidio Quaglietta<sup>2</sup>,  
Immacolata Lamberti<sup>3</sup>, and Pietro Marmo<sup>3</sup>

<sup>1</sup> Dipartimento di Informatica e Sistemistica, University of Naples "Federico II"  
Via Claudio 21, 80125 Naples, Italy

{mazzeo, nicola.mazzocca, roberto.nardone}@unina.it

<sup>2</sup> Dipartimento di Ingegneria dei Trasporti, University of Naples "Federico II"  
Via Claudio 21, 80125 Naples, Italy

{luca.dacerno, bruno.montella, vinpunzo,  
egidio.quaglietta}@unina.it

<sup>3</sup> Ansaldo STS, RAMS - Transportation Solutions Business Unit  
Via Argine 425, 80147 Naples, Italy

{Immacolata.Lamberti, Pietro.Marmo}@ansaldo-sts.com

**Abstract.** Prediction of service availability in railway systems requires an increasing attention by designers and operators in order to satisfy acceptable service quality levels offered to passengers. For this reason it is necessary to reach high availability standards, relying on high-dependable system components or identifying effective operational strategies addressed to mitigate failure effects. To this purpose, in this paper an innovative architecture is proposed to simulate railway operation in order to conduct different kinds of analysis. This architecture encompasses a set of components considering, in an integrated way, several system features. Finally an application to a first case study demonstrates the impact on quality of service and service availability of different recovery strategies. Complexity of a railway system requires a heterogeneous working group composed of experts in transport and in computer science areas, with the support of industry.

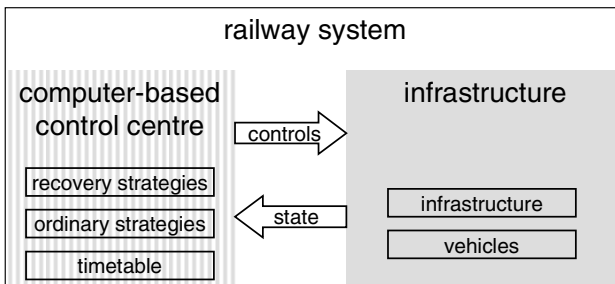
**Keywords:** quality of service prediction, service availability prediction, railway simulation, failures mitigation.

## 1 Introduction

Planning and designing phases of both railway infrastructure components (stations, rail tracks, vehicles, signalling equipments, etc.) and operation strategies at control centre (e.g. service timetable, recovery strategies after service breakdowns, shunting movements, etc.) aim at satisfying both dependability constraints as imposed by customer specifications and overall the total transportation demand (people, goods or both) foreseen for a considered area, respecting certain requirements of service quality. Railway networks can be considered in fact as complex demand-oriented



systems, since the reasons why they are built are strongly related to the contentment of certain transportation demand levels as well as the improvement of economic and environmental conditions of surrounding areas. Therefore it is clear that the attainment of determined standards of service quality offered to passengers, represents a fundamental issue to increase both system attractiveness and global benefits to railway operators and their customers. To this purpose train operator companies and infrastructure managers, have the hard task of assuring established service availability levels: such aim can be reached not only equipping railway infrastructure with high-dependable components (using high-reliability and high-maintainability items) but also determining effective operational strategies which minimize impacts of components failures on service levels. In particular decisional phases addressed to identify amongst several alternatives, the most appropriate designing or operational solution, are strongly supported by opportune railway simulation models, since the high complexity degree of railway networks prevents system behaviour to be described by analytical closed-form solutions.



**Fig. 1.** Railway system composition

As illustrated in figure 1 railway systems contain computer-based elements which aim at controlling both infrastructure (e.g. interlocking areas, telecommunications, switch machines, etc.) and vehicles to perform safe train movements on the track. Nowadays different models have been developed which lead to different and non-integrated models which are separately addressed to simulate railway system operations or analyze its respective components. The first kind of models can be classified on the basis of the considered level of detail in macroscopic, mesoscopic and microscopic. Macroscopic models ([1], [2]) depict at high abstraction level railway infrastructure. They are mostly used during long-term planning tasks to determine preliminary physical characteristics of the network (e.g. number of stations, inter-station lengths, etc.) as well as system capacity values to satisfy the level of transportation demand estimated. Mesoscopic models ([3]), thanks to their "multi-scale" structure containing both areas modelled on a macroscopic level and areas modelled on a microscopic level, consent to realize "simplified" railway simulations minimizing computational efforts. Microscopic models ([4], [5]), instead represent with high detail level each element of railway infrastructure allowing for the precise evaluation of system operations. The second type of models, instead, are not addressed to system operations simulation, since they are used especially for availability evaluation at subsystem level.

The accurate assessment of service levels induced by designing and operational alternatives, strongly need to consider the overall interactions amongst the different items composing railway system. In particular railway operators need to analyze how interventions on mitigation strategies (implemented by computer-based elements) addressed to reach certain availability targets, affect Quality of Service perceived by passengers. To this purpose an integrated approach for railway service simulation is proposed to contemporary estimate both QoS and service availability, consenting to dominate the complexity of real railway systems. An innovative architecture encompasses different interacting modules, each one depicting a specific feature of both infrastructure and computer-based control centre. Such approach will therefore conduct to have a more complete view when analyzing different designing solutions and overall strategic alternatives.

The second paragraph of this paper will briefly summarize theory and methodologies to estimate parameters of both quality of service offered to passengers and operational availability, while the third paragraph will deal with an accurate description of the proposed simulation architecture. In the fourth paragraph a practical application on a Mass Rapid Transit case study will illustrate the effect on both service availability and quality, induced by different operation strategies to recover normal service after a failure of a train. Finally concluding comments will be explained.

## 2 Quality of Service and Service Availability for a Railway System

In this section Quality of Service and Service Availability will be defined, specifically referring to the field of railway transportation systems. Policies and regulations which introduce such questions at international level are considered. Then a paragraph dedicated to clearly illustrate the strong dependencies between such issues is reported.

### 2.1 Quality of Service and Passengers Satisfaction

Quality of Service (QoS) offered by a railway system to its customers, certainly represents one of the most important issues that a railway operator has to consider. In fact, also in accordance with the international literature on marketing science, only the achievement of passengers' satisfaction as well as an acceptable quality of service will let railway operators be competitive in the domain of people transportation ([6]).

The term "satisfaction" for a customer of a railway line, involves different issues such as: the cost of the service (e.g. price of the ticket), the journey time, quality parameters like punctuality, cleanness, safety, travel time variability (connected to service reliability) and so on. According to classical literature ([7]), customer's satisfaction is reached when the so called "user's generalized cost" is minimized. In particular, the generalized cost  $C_i$  of a single customer for choosing alternative  $i$ , can be expressed as a linear combination of the  $K$  attributes concerning that alternative  $X_{K,i}$  weighted by their respective homogenization coefficients  $\beta_{K,i}$ , which mostly represent specific costs of the attribute:

$$C_i = \sum_K \beta_{K,i} \cdot X_{K,i} \quad (1)$$

For a passenger railway line, specific cost attributes can be constituted for example by quantitative variables like average passenger waiting time at stations, total on-board travel time, cost of the ticket to access the service, arrival delays at destination, as well as qualitative variables (often considered as dummy variables) such as comfort of the journey, cleanness of trains and stations, travel safety, and so on.

Moreover, in literature, applications of quality measuring methodologies to real case studies, show how the estimation of quality parameters (through direct detections or surveys to passengers) can lead to the identification of quality-critical infrastructure sections which call for improvements. On this basis, it can be possible to branch also complex railway systems in smaller and less complex regions which are homogeneous from the service quality point of view. In such a way more effective analyses could be conducted only for the most critical regions to identify designing or operational solutions addressed to improve the respective quality levels. Anyway in this paper QoS perceived by passengers will be measured by means of the users' generalized cost which in turn will be calculated considering the only "total journey time" (i.e. the sum of the on-board travel time and the average waiting time at stations) as specific attribute.

## 2.2 Service Availability in Railway Systems

The norm EN 50126 ([8]) defines for railway networks the term "availability" as the ability of a certain system to perform required functions under given conditions, over a certain time period assuming that the needed sources of help are provided. This means that a system with a high level of availability will mostly fulfill the requested kind of service under the defined framework conditions, and therefore will assure high levels of Service Availability (SA). As is evident, the availability of service is strongly related to system availability, since a failure of a system component will tend to reduce railway system functions and consequently its operational availability.

In order to guarantee acceptable SA levels railway operators can achieve high values of system availability employing high-dependability components (i.e. high reliable and maintainable items) or adopt operational strategies addressed to minimize failures effects on system availability. To this purpose a set of regulations have been recently introduced in the field of railway systems to define a list of management procedures which aim at performing the so called RAMS (Reliability, Availability, Maintainability and Safety) throughout the railway system lifecycle. Such procedures have the objective of achieving certain qualitative and quantitative targets for each element of railway system (subsystems and components), in order to guarantee determined standards of availability and therefore of SA, which are reliable and safe at the same time. Anyway SA is usually represented by mathematical indexes often described by the ratio between performed (actual) and target (designed) service measures. In particular such indexes are defined and specified within contracts between customers (train operators or infrastructure managers) and railway systems designers or manufacturers. Common SA indexes are for example: "system availability" expressed as the ratio between the time of performed and scheduled service (in minutes), or "punctuality" defined as the ratio between the number of on-time trips and the total number of trips arrived at a certain station. In particular punctuality can be expressed as:

$$\text{punctuality} = (t_s - t_l) / t_s \cdot 100 \tag{2}$$

where  $t_s$  is the number of scheduled trips within a certain time period and  $t_l$  is the number of lost trips (i.e. the number of trips which does not arrive at the considered station) calculated over the same time interval. In this paper, punctuality index will be considered to measure SA.

### 2.3 Relationships between QoS and SA in Railway Systems

As is evident, QoS perceived by passengers of a railway system is strongly dependent on levels of SA achieved by railway operators on the system itself. In fact passengers' satisfaction is related to the gap between expected (ideal) and perceived (actual) level of QoS. In turn perceived QoS levels strongly depend on the discrepancy between targeted and delivered SA levels achieved by service providers. Such relationship is clearly depicted in the loop shown in figure 2, as illustrated by AFNOR, the French Organization for Standardisation ([9]). Therefore only if this loop is retained the service is considered as successfully offered. It seems clear that in order to increase the attractiveness of a passenger railway system, railway operators must seriously take into account this matter starting to adequate both infrastructure characteristics (e.g. signalling system, number of stations, etc.) and operation strategies (e.g. timetable, train movements to recover normal service after a system breakdown) in a demand-orientated way.

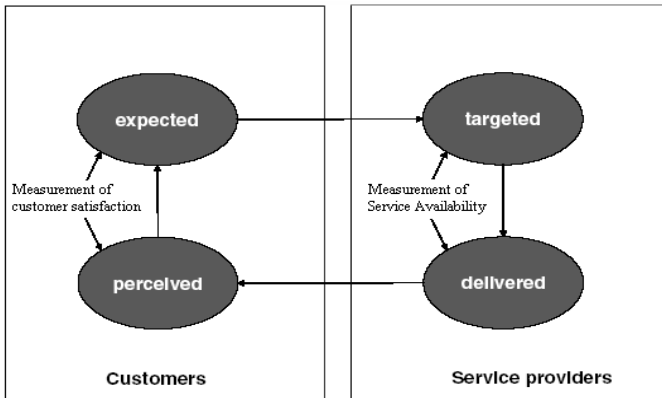


Fig. 2. Service level loop in railway transportation systems (AFNOR, 2006)

For these reasons, decisional phases at each level, require not only the investigation of the effects induced by a certain management solution (infrastructural or operational) on network performances, but also their respective impacts on passengers flows. Since the high level complexity of this problem, decisional activities both during designing and real-time rescheduling stages, strongly needs to be supported by opportune simulation systems which consent the evaluation of repercussions on both SA and QoS, induced by different solutions to identify the most effective one.

### 3 Simulation System Architecture

Here an integrated architecture is introduced for the simulation of railway system operations and the contemporary assessment of both SA and QoS levels. Outputs returned by such architecture can be analyzed with three different goals: evaluating the impact of disturbances on service, estimating the variation induced to the SA related to QoS; estimating the efficiency having different configurations of track layout allowing to choose the one that has the greater impact on cost reduction; identify the optimal configuration of the plants and of operation and maintenance staff.

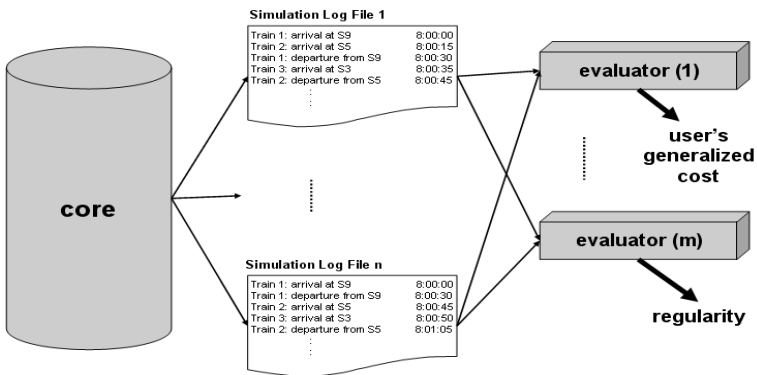
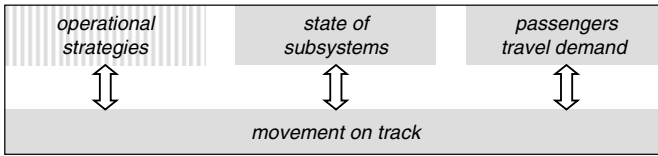


Fig. 3. Simulation system architecture

The necessity to estimate a non-completed (and project dependant) set of indexes makes necessary the decomposition of the architecture in a *core* component on which it is possible to connect different *evaluators* for measurement of interest parameters. The explicit architecture is depicted in figure 3: the *core* component is the kernel of the architecture and it reproduces system operational conditions service taking into account any kind of stochastic disturbance on service; the *Simulation Log Files* developed in output, in a very large number, contain the list of all instant of arrival and departure of each train at each station; a group of *evaluator* make use of *Simulation Log Files* to evaluate interest parameters. For the present paper scope of work we have connected two particular *evaluators* for the estimation of the two indexes showed in previous paragraph: *user's generalized costs* and *punctuality*.

High dimensions and complex internal relationships of a railway system requires the integration of heterogeneous modules in the *core*, each one manage the evolution of a particular feature during the simulation. In particular, as depicted in figure 4, these modules are: *operational strategies*, *state of subsystems*, *passengers travel demand* and *movement on track*. With respect to the railway system composition, as already shown in figure 1, the *operational strategies* module implements the control-centre functions while the others modules simulate infrastructure components.



**Fig. 4.** Internal organization of *core* component

### 3.1 Operational Strategies Module

This module is responsible for the implementation of operational strategies controlled by computer-based control centre. In particular, it is possible to distinguish three main functionalities: the respect of the timetable, the execution of ordinary strategies and the implementation of the recovery strategies, when needed. The module authorizes train departures according to timetable and obviously respecting signalling system aspects which safely regulates train movements on the track. Moreover such module enables the activation of apposite train movements which aim at performing specific operational strategies mostly addressed to restore ordinary service after a component failure.

### 3.2 State of Subsystems Module

The *state of subsystems* module deals with the simulation of operating mode of all subsystems involved in service fulfilment. In particular, this module simulates the evolution between different operating modes according to the failure rates and probability of state passing of each failure-prone component (vehicles, on board signalling system, central signalling system, infrastructure, etc.). The model use a Markov chains: the states represent different operating modes of the component and on the arcs there is the probability of state passing between a couple of them, calculated from component failure rates. Some of the arcs of the complete graph may be associated with null probability when those transitions are not feasible during operation. Obviously, this module permits to simulate the nominal operation, in case, for example, of timetable validation, setting null all the failure probability.

### 3.3 Passengers Travel Demand Module

This module is dedicated to the simulation of passengers demand. In particular passengers origin-destination matrix relative to a certain time period, is considered as input datum. Such module is constituted of an assignment model which by means of consistency equations calculates, during simulation, passengers boarding and alighting flows at stations, returning as output the on-board flows for each station and for each train run.

### 3.4 Movement on Track Module

This module is responsible for simulation of train movements on track. Inputs of this module are all physical and mechanical characteristics of both track and vehicles. Such module assumes the master role during simulation, collaborating with other modules. A classical compositional strategy ([10]) based on the cooperation of basic elements (e.g. stations, block sections, terminals, pocket tracks, etc.) is used; thanks to

well-defined composition rules, these basic elements can be joined together in order to produce the track layout you want to simulate in an enough simple way.

To simulate both ordinary and degraded train service, a dynamic integration between different simulation approaches has proved to be an effective solution to overcome the limits of applicability of models at high level of detail, but computationally inefficient (i.e. microscopic), and models unable to describe local and transient train dynamics even though very efficient (i.e. mesoscopic) ([16]). In fact a combined approach, which dynamically integrates micro- and mesoscopic models allows to efficiently simulate large-scale networks or deal with a large number of simulations (e.g. when performing probability analyses), preserving the accuracy needed to evaluate QoS and SA. In particular, the mesoscopic approach is implemented by means of Stochastic Activity Networks (SAN) formalism and uses timed transitions to model travel times between stations, whilst the microscopic approach, designed in C++, explicitly simulate train dynamics integrating the Newton's motion formula .

#### 4 Case Study: A Mass Rapid Transit System

To clearly understand the usefulness of the simulation architecture proposed in this paper, it is necessary to implement a practical application on a case study, in particular a Mass Rapid Transit system has been considered. As shown in figure 5, this network is constituted of a 12 km long double-track layout with 15 stations and two terminals to let trains change their path or reverse direction. An ETCS level 1 signalling system type regulates train movements on the track. Furthermore a pocket track to store away corrupted trains is located between station 7 and 8. Scheduled train headway is set to 6 minutes while train dwell times are all equal to 20 seconds for each station. The depot is pinpointed between first and second station, it has three connections with the line: one is used for entering vehicles in service, another one is used instead to allow the service entrance of hot spare vehicles, and the last one is used for train admission to the depot. Total train running time (including dwell times at stations) is 1251 sec. for 1-15 direction and 1257 sec. for 15-1 direction. Minimum train inversion time at is about 20 sec. at terminal 15 and 140 sec. at terminal 1, while the maximum synchronization time awaited at terminals to perform a constant headway is 66 sec.

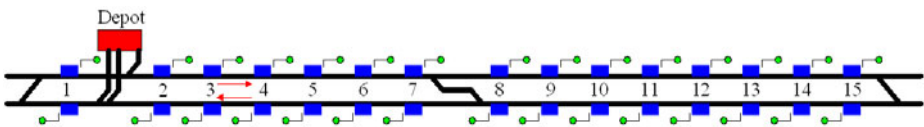


Fig. 5. Schematic layout of the considered MRT system

Passenger travel demand considered for the line is reported in terms of on-board passengers flows (both for 1-15 and 15-1 direction) referring to a working-day peak-hour in the morning. As can be seen, the maximum passenger flow assessed for 1-15 direction is equal to 8500 pax/h, while for the opposite direction (15-1) this value is 3189 pax/h (figure 6).

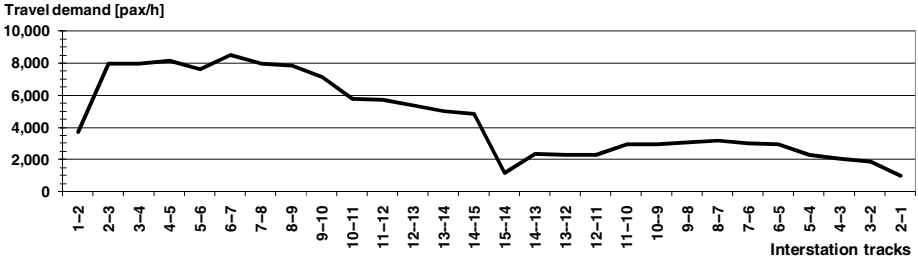


Fig. 6. Passenger travel demand data loading the system

In this application a total time interval of 3 hours has been observed, considering that hourly passenger flows previously described, preserve exactly the shown trend within each hour of the considered period. According to the scheduled train headway defined by timetable, a total of 30 train runs for each direction have been analyzed. In particular during ordinary service conditions, simulation outputs return total on-board passengers flows for each train run and for each inter-station track (figure 7). As can be seen, each train run shows the same passengers load (in fact on-board passengers diagrams relative to each train run are overlapped).

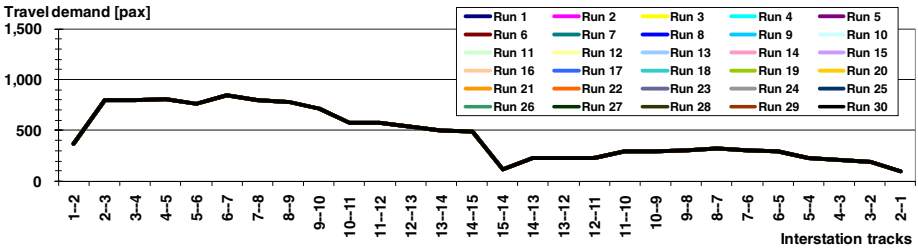


Fig. 7. On-board passengers for each train run for ordinary service condition

In particular no limits have been set for train capacity (i.e. the maximum number of passengers that a train can contain). Moreover for the calculation of the users’ generalized cost the “total journey time” (i.e. the sum of the on-board travel time and the average waiting time at stations) has been considered as cost function attribute, using a value of the specific cost  $\beta$  of 5€/h. Within ordinary service the total passengers’ generalized cost estimated over all passengers flows during the entire time interval is 63893 €.

Successively a failure scenario has been considered, supposing that the second train run along 15-1 direction experiences a breakdown after his departure from station 15 causing a degraded functioning state of the train no. 1 which increases the respective travel time of 2 times. Then in such conditions three different recovery strategies have been analyzed and for each one the effects on both QoS offered to passengers and SA have been assessed. Descriptions and simulation results obtained for each strategy are reported in the following paragraphs.



### 4.1 First Operational Strategy: Return to Depot and Successively Substitution

This strategy consists in keeping on service the corrupted train in degraded conditions, until it reaches the depot near station no. 1. Once this has entered the depot it is substituted by a hot spare vehicle (train no. 9) which starts its service from station no. 1 (obviously along 1-15 direction). As shown in figure 8, due to the higher travel time experienced by the broken vehicle, a consistent delay is transferred to other trains. The delay suffered by trains induces a passenger overloading of runs (figure 9), especially for those which enter on service after the occurring of the failure event.

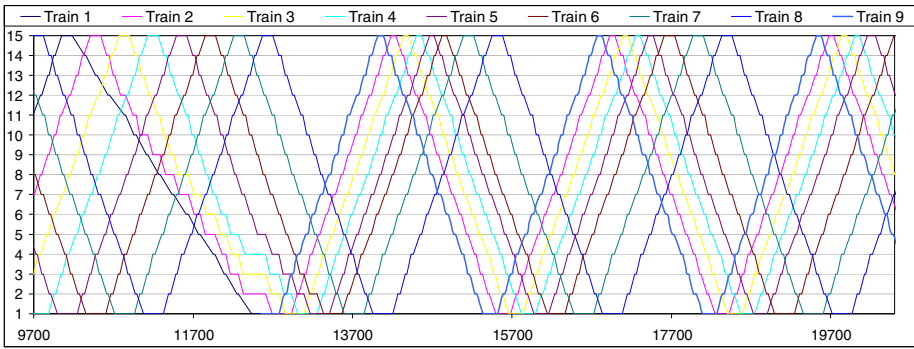


Fig. 8. System operation for the first recovery strategy

Moreover this effect is obviously higher for train runs along 1-15 direction (because of the higher demand level) and slowly tend to fade away after the entrance of the spare. Anyway for this strategy the total generalized cost estimated is 83905 €. Therefore with respect to ordinary service conditions such strategy determines an increase in the total generalized cost (i.e. a decrease in passengers’ satisfaction) of about 31%. The effect induced by such strategy on SA has been detected through measuring the punctuality index at terminal station no. 1. In particular such index is equal to 76.57%.

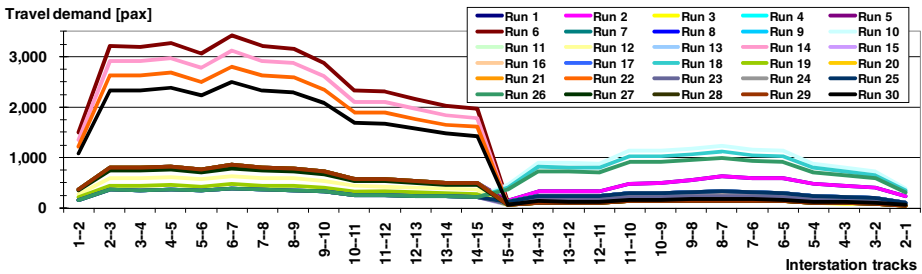


Fig. 9. On-board passengers for each train run for the first recovery strategy

### 4.2 Second Operational Strategy: Preventive Insertion

Such strategy instead considers that a minute after the failure has occurred a hot spare from the depot is put on service from station no.1 along 1-15 direction, while the corrupted train, although is degraded, continues its service along 15-1 direction until it reaches station no. 1 and enters the depot.

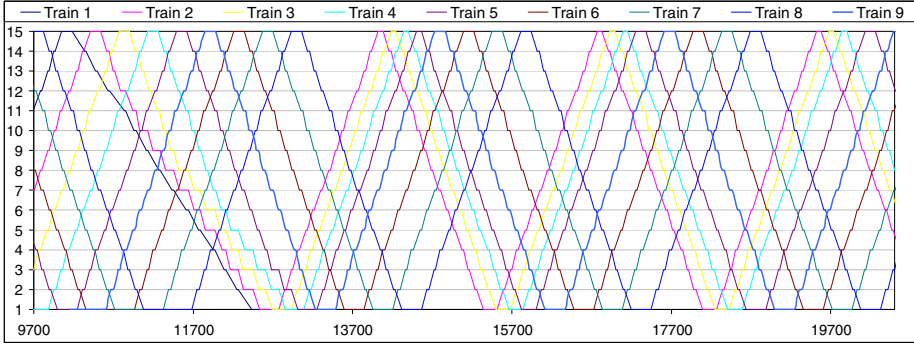


Fig. 10. System operation for the second recovery strategy

As in the first strategy, also for this one a passenger overloading of train runs happens (figure 11) especially for 1-15 direction, but in this case train loading rates are lower, since the delay transferred from the corrupted train to other trains is lower. However for this second strategy the assessed total passengers’ generalized cost is 70614 €. This means that with respect to regular conditions such strategy induces an increase of about 11% of the total generalized cost. The immediate introduction of the spare vehicle makes punctuality index be 83.23%, reducing the overdue runs from 6 to 4 (figure 10).

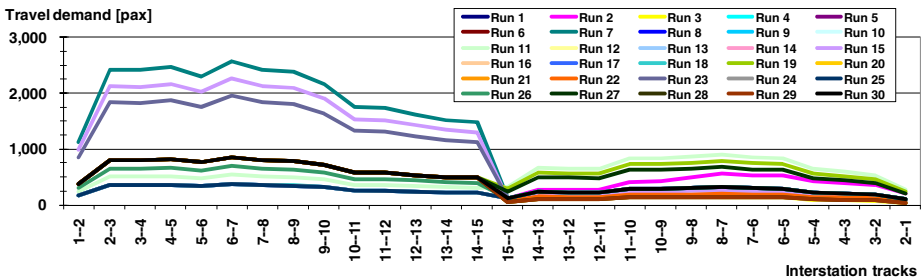


Fig. 11. On-board passengers for each train run for the second recovery strategy

### 4.3 Third Operational Strategy: Store Away on Pocket Track

The last analyzed strategy consists in keeping on service the broken train in degraded conditions along 15-1 direction, until it reaches section between station 8 and 7 where it is stored away on the pocket track there located. Then a hot spare from the depot is put on service from station no.1 along 1-15 direction (figure 12).

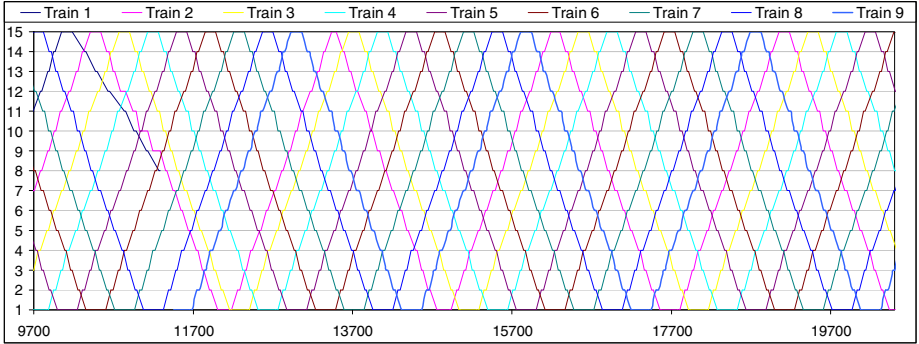


Fig. 12. System operation for the third recovery strategy

This highlights that such strategy strongly mitigates the impacts of train knock-on delays transferred from the corrupted train to the other runs (figure 13). Anyway the total passengers’ generalized cost calculated for this strategy is 64234 €, and with respect to ordinary conditions determines a cost increase (i.e. a satisfaction decrease) of only 0.5%. The value of the punctuality index during the considered simulation period is 94,57%, but higher costs for the installation of the pocket track are necessary to put in practice this strategy.

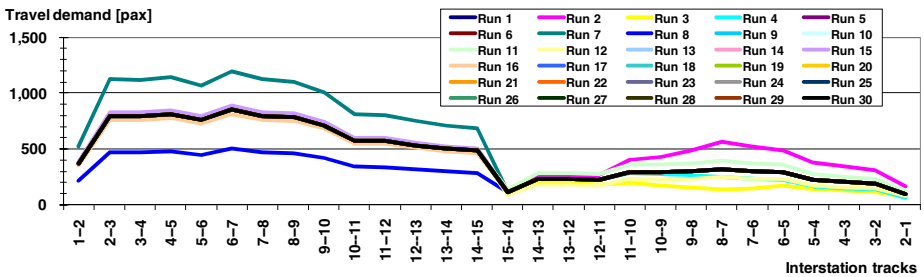


Fig. 13. On-board passengers for each train run for the third recovery strategy

## 5 Conclusions and Future Works

This paper proposes a modular architecture for railway service simulation. This architecture allows to assess several indexes in particular relative to SA and QoS, evaluating the impact of stochastic disturbances on service. Given the complexity of a

railway system each module which constitutes the proposed simulation architecture can only be developed by a heterogeneous working group made of computer science and transportation experts with a remarkable support of the industry. The application of such simulation architecture to a real-scale MRT case study, demonstrates that the impacts on service due to a certain failure scenario can be mitigated in several ways adopting different recovery strategies. Obtained results show how the proposed architecture enables the contemporary evaluation of several service level indexes and consequently verify the attainment of contract requirements and functional specifications in general. Such aspect, together with the capability of this approach to dominate complexity of real scale railway systems, confirm its relevance in industrial research area.

Moreover the simulation can be performed during all the project lifecycle, providing important information since the early stages of project design, when almost 20% of the total ownership cost of the system is already “frozen”. The proposed simulation system evaluates capability of a railway system to fulfil target indexes, thus permitting to identify proper remedial actions, allowing also to avoid penalty payments. In addition to this, it is important to remember that, on average, a retrofit cost is almost 4 times higher than a cost sustained in the design phase. Moreover it can be said that such integrated approach could support both railway operators and railway industry to acquire more confidence on requirements compliance during decisional phases at each level.

Prototype versions of the modules described in this architecture have been developed to date, using a multiformal approach (mainly made of SAN networks) combined with C++ programs. Future work will be addressed to describe and develop interfaces in a formal way in order to permit easy replacement and reuse.

## References

1. Kettner, M., Prinz, R., Sewcyk, B.: NEMO – Netz – Evaluations-Modell bei den OBB. Eisenbahntechnische Rundschau (ETR) 3, 117–121 (2001)
2. Kettner, M., Sewcyk, B.: A model for Transportation Planning and Railway Network Evaluation. In: Proceedings of the 9th World Congress on Intelligent Transport Systems, Chicago, USA (October 14-17, 2002)
3. Marinov, M., Viegas, J.: A mesoscopic simulation modelling methodology for analyzing and evaluating freight train operations in a rail network. *Simulation Modelling Practice and Theory* 19, 516–539 (2011)
4. Nash, A., Huerlimann, D.: Railroad Simulation using Open-Track. In: *Computers in Railways IX*. WIT Press, Southampton (2004)
5. Siefer, T., Radtke, A.: Railway Simulation, Key for Operation and Optimal Use. In: Proceedings of the 1st International seminar of Railway and Operations Modelling and Analysis, Delft, the Netherlands (June 8-10, 2005)
6. Kotler, P.: *Marketing Management, Analysis, Planning, Implementation and Control*. Prentice Hall, Englewood Cliffs (1991)
7. Cascetta, E.: *Transportation System Analyses, models and applications*. Springer, New York (2009)
8. CENELEC: Railway applications – Specification and demonstration of reliability, availability, maintainability and safety (RAMS). EN 50126 (1999)

9. AFNOR Group, <http://www.afnor.org> (last access 15.03.2011)
10. Hagalisletto, A.M., Bjork, J., Chieh Yu, I., Enger, P.: Constructing and Refining Large-Scale Railway Models Represented by Petri Nets. *IEEE Trans. On System, Man and Cybernetics-Part C: Applications and Reviews* 37(4), 444–460 (2007)
11. Grupe, P., Nunez, F., Cipriano, A.: An event-driven simulator for multi-line metro system and its application to Santiago de Chile metropolitan rail network. *Simulation Modelling Practice and Theory* 19(1), 393–405 (2009)
12. Kaakai, F., Hayat, S., El Moudni, A.: A hybrid Petri nets-based simulation model for evaluating the design of railway transit stations. *Simulation Modelling Practice and Theory* 15(8), 935–969 (2007)
13. Vittorini, V., Iacono, M., Mazzocca, N., Franceschinis, G.: The OsMoSys approach to multi-formalism modeling of systems. *Software and Systems Modeling* 3(1), 68–81 (2003)
14. Flammini, F., Marrone, S., Mazzocca, N., Vittorini, V.: A new modeling approach to the safety evaluation of N-modular redundant computer systems in presence of imperfect maintenance. *Reliability Engineering & System Safety* 94(9), 1422–1432 (2007); ESREL 2007, the 18th European Safety and Reliability Conference (2007)
15. Flamini, F., Mazzocca, N., Moscato, F., Pappalardo, A., Pragliola, C., Vittorini, V.: Multiformalism techniques for critical infrastructure modelling. *International Journal of System of Systems Engineering* 2(1), 19–37 (2010)
16. Quaglietta, E., Punzo, V., Montella, B., Nardone, R., Mazzocca, N.: Towards a hybrid mesoscopic-microscopic railway simulation model. In: 2nd International Conference on Models and Technologies for ITS (2011)

# Using a Software Safety Argument Pattern Catalogue: Two Case Studies

Richard Hawkins, Kester Clegg, Rob Alexander, and Tim Kelly

The University of York, York, U.K.

{richard.hawkins,kester.clegg,rob.alexander,tim.kelly}@cs.york.ac.uk

**Abstract.** Software safety cases encourage developers to carry out only those safety activities that actually reduce risk. In practice this is not always achieved. To help remedy this, the SSEI at the University of York has developed a set of software safety argument patterns. This paper reports on using the patterns in two real-world case studies, evaluating the patterns' use against criteria that includes flexibility, ability to reveal assurance deficits and ability to focus the case on software contributions to hazards. The case studies demonstrated that the safety patterns can be applied to a range of system types regardless of the stage or type of development process, that they help limit safety case activities to those that are significant for achieving safety, and that they help developers find assurance deficits in their safety case arguments. The case study reports discuss the difficulties of applying the patterns, particularly in the case of users who are unfamiliar with the approach, and the authors recognise in response the need for better instructional material. But the results show that as part of the development of best practice in safety, the patterns promise significant benefits to industrial safety case creators.

## 1 Introduction

Providing a compelling software safety argument is a fundamental but challenging part of demonstrating that a system is safe. Part of the problem is providing evidence for low-level argument claims, but there are also difficulties in structuring the argument in an intelligible and maintainable way. To help with this latter problem, we have developed a catalogue of software safety argument patterns which guide engineers in structuring safety arguments.

The pattern catalogue is summarized in [1] and documented fully in Appendix B of [2]. The philosophy underpinning these patterns is that developers must demonstrate assurance in the same fundamental safety claims for all software used in a safety related role; the difference between arguments for different systems is in the way in which these claims are ultimately supported. The patterns we have created define the expected structure of a software safety argument which supports all of the fundamental safety claims.

We intend for the patterns to provide benefits to several different stakeholders. When a developer uses them during the earlier stages of a systems lifecycle, they should find it easier to identify areas where the assurance of the system may be

weak. They can then make changes (to the system or its operating restrictions) to address these areas of concern. The patterns can also help reviewers of a software system to identify where assurance deficiencies may exist, and provide a common baseline for agreeing acceptability. In essence, the patterns attempt to encourage best practice in creating and reviewing software safety arguments.

In order to check the effectiveness of the patterns in achieving these aims we applied the patterns to a number of industrial case studies to determine their effectiveness. In this paper we describe some of our experiences of applying the patterns on two of these safety-critical software projects.

In particular, we wanted to assess the patterns in the software safety argument pattern catalogue against the following desirable criteria:

- The patterns should be easy to understand and apply by software development teams.
- The patterns should be flexible enough to apply to any safety-critical software system.
- The patterns should ensure that the resulting software safety argument is explicitly focused on controlling the software contribution to system hazards.
- It should be easy to judge the sufficiency of an argument created using the patterns.

In the next section we give an overview of the pattern catalogue. Sections 3 and 4 then describe our experiences in two case studies: a prototype autonomous vehicle controller, and an aircraft avionics software system. Finally, Section 5 draws some conclusions from these experiences and outlines the future for the pattern catalogue.

## 2 Software Safety Argument Pattern Catalogue

Prior to the development of our pattern catalogue, the main extant work in the area was that of Weaver [3]. Weaver's catalogue was unique in its time in that, unlike that of Kelly [4], it was specifically aimed at software systems, and specifically designed to connect its patterns together in order to form a single coherent argument. However, Weaver's catalogue has a number of weaknesses. First, the patterns take a fairly narrow view of assuring software safety, in that they focus on the mitigation of known failure modes in the design. Mitigation of failure modes is important, but there are other aspects of software assurance which should be given similar prominence. Second, issues such as safety requirement traceability and mitigation were considered at a single point in Weaver's patterns. This is not a good approach; it is clearer for the argument to reflect the building up of assurance relating to traceability and mitigation over the decomposition of the software design (see later discussion on the tiered approach). Finally, Weaver's patterns have a rigid structure that leaves little scope for any alternative strategies that might be needed for novel technologies or design techniques.

The other relevant existing patterns are those developed by Fan Ye [5] specifically to consider arguments about the safety of systems including COTS software

products. Ye’s patterns provide some interesting developments from Weaver’s, including patterns for arguing that the evidence is adequate for the assurance level of the claim it is supporting. Although we do not necessarily advocate the use of discrete levels of assurance, the patterns are useful as they support arguing over both the trustworthiness of the evidence and the extent to which that evidence supports the truth of the claim.

The patterns we created were deliberately constructed such that they make no assumptions about project, application or domain specific details. For example they are designed to be applicable for any software development process, any software design methodology, diverse types of system-level hazards and diverse software requirements.

The key organizing assumption for the patterns was that as the software system moves through the development lifecycle there are numerous assurance considerations against which evidence must be provided. Jaffe et al [8] proposed an extensible model of development which captures the relationship between components at different “tiers” (a set of tiers for one project might be for example the software architecture, the software high-level and low-level designs, and the source code). For our purposes we can note that at each tier, different assurance considerations arise. Our patterns are explicitly based on a view of software development as this process of refinement through tiers, and they consider the relationship between the design information at various tiers and the resulting assurance considerations. The number and type of tiers used in the specific design process being used is irrelevant, so long as the assurance considerations are sufficiently addressed at each tier.

Figure 1 summarises the assurance considerations that are repeated at each tier of a software development lifecycle. At each tier, the pattern instantiator must provide evidence which is sufficient to address each of these considerations, and they must provide a compelling argument which explains how the evidence addresses each assurance consideration. It follows that as software development progressed through more detailed design tiers, more assurance evidence is generated.

From Figure 1, the assurance considerations defined for each tier can be seen to be:

1. The safety requirements placed upon the software have been met.
2. Those safety requirements are appropriate for the design at this tier and are traceable to higher tiers.
3. Hazardous errors have not been introduced into design at this tier.
4. Hazardous failure behaviour has been assessed — it has been determined what could go wrong at this tier and how it is mitigated.

If a safety argument is to be compelling, then it is crucial that the high-level structure of the argument is correct. This requires that the argument focuses explicitly on how the software can contribute to system level hazards. Our patterns provide this structure by forcing users to consider each system hazard in which



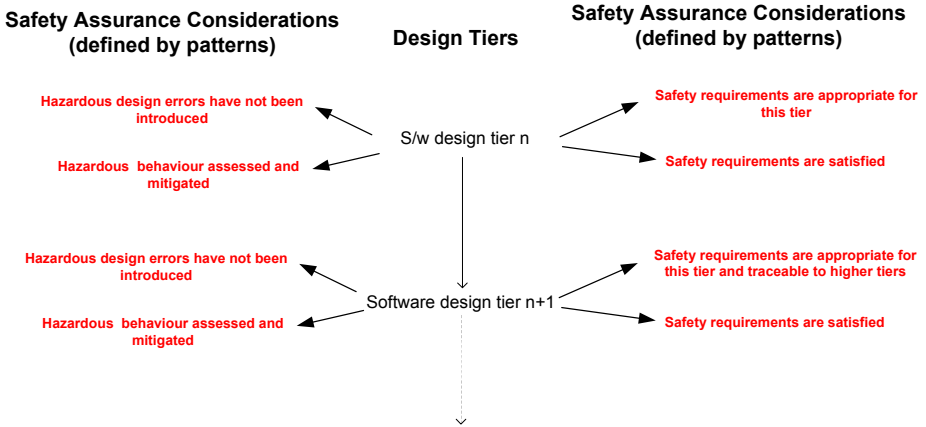


Fig. 1. System safety requirements

software may play a role and to identify the specific software behaviour which may contribute to the hazard. This might involve, for example, working systematically over the base events in fault tree. If an argument creator understands the specific functions or properties of the software in which assurance is required, then they can focus the argument and evidence on those things. This structured approach should help to discourage spurious information being included in the safety argument “just in case”.

### 2.1 Identifying Assurance Deficits

There will be aspects of all safety-related software systems for which assurance is not demonstrated with complete certainty; there will always be things relating to the behaviour of the software system which remain unknown or unclear. We refer to such uncertainties as assurance deficits since they can undermine the assurance that can be demonstrated. It is important to note that assurance deficits do not necessarily correspond to faults or defects in the system, but instead to an inability to demonstrate complete certainty in each safety claim in the argument.

For example, it may be known that the compiler being used has some undefined behaviour. It may be known that a design model used may not accurately represent certain real environmental features or that a component has not been exhaustively tested. Or it may be known that an assumption that has been made about partitioning of some software modules may not actually hold in all cases.

The argument patterns can help identify where such assurance deficits exist. These assurance deficits may relate to the safety evidence generated or to the safety argument itself. It is through managing assurance deficits that the required assurance can be achieved.

### 3 Case Studies

The next two sections describe case studies where the software safety case patterns were used on real products. The first is the control software for a prototype autonomous vehicle and the second for an aircraft avionics system. Each of the case studies highlights different ways that using the patterns can benefit the production of safety case arguments by indicating where those arguments are either missing evidence to back up safety claims or failing to identify clearly the software contribution to the hazards being considered. The case studies differ in the stage of technology readiness, the type of software deployment and the derivation of safety properties. They also differ in terms of the level of experience of the engineers creating the safety case. Despite these differences, the patterns were sufficiently flexible to be used and provide benefits to both safety cases.

#### 3.1 Prototype Autonomous Vehicle Case Study

As part of a SEAS DTC [9] project on safety of autonomous systems we created a safety argument for an autonomous system drawn from the SEAS DTC Exemplar 2 scenario [10]. The system chosen was a prototype Unmanned Ground Vehicle (UGV) that formed part of a larger System of Systems (SoS), including an Unmanned Aerial Vehicle (UAV) and ground control units. As a high-level hazard and safety analysis for the SoS had already been completed using previously developed techniques [11], we decided to construct a partial safety argument that would start at a system level hazard for the UGV and end with arguments justifying the safety of software that could contribute to that hazard. The UGV in question is an adapted all-terrain vehicle (the Wildcat) produced by the Advanced Technology Centre of BAE Systems. The current prototype is able to operate without a driver, to follow an off-road GPS waymarked route by calculating the best path within the waymarked corridor and is able to avoid static objects. As with many UGVs it is heavily reliant on GPS signals for their autonomous operation. However, in cases where the GPS signal is lost or jammed, the vehicle is able to continue to plan its path by taking measurements from the Inertial Measurement Unit (IMU) in conjunction with other on-board sensors (such as LIDAR). Unfortunately the IMU measurements (and therefore estimates of the vehicle's position) are subject to drift over time, giving an ellipse of uncertainty with regard to the vehicle's true position that can grow in an unbounded fashion in some scenarios (e.g. after entering a long tunnel). This could result in the vehicle colliding with objects or the side of the road as it miscalculates its position.

While there are many potential collisions that could be described, the approach adopted was first to identify potential accidents and the hazards that could lead to those accidents occurring. Based on the hazards, we next identified a set of top level system safety requirements. These requirements were then further decomposed using techniques derived from the Goal-Oriented Requirements Engineering (GORE) approach of Lamsweerde and others [12]; this proved fairly straightforward and intuitive.

The act of decomposing the system safety requirements gave rise to safety requirements over the software that form the starting point for instantiating the software safety case patterns discussed in this paper. Our intention was not to conduct an exhaustive safety review of a particular hazard and its mitigation; instead we chose to restrict ourselves to a particular aspect of one hazard (object collision after loss of GPS signal, see Table I, safety requirement SR2-4) and to follow the instantiation of a software safety case pattern to the point at which evidence would normally be presented to meet the software safety requirements defined at the lowest levels. This naturally excluded much peripheral work that would have been essential to a full safety case, as either the technical information was not to hand or we felt it was not directly related to the part of the safety case we were covering.

**Table 1.** System safety requirements

Safety requirement	Description	Notes	System or software components
SR2 – top level	While moving, the UGV avoids collisions with objects.	Top level SR2 depends on availability of GPS, sensor ranges and quality of their data, path planning and vehicle driver functions.	All of UGV
SR2-1	Vehicle restricts speed such that stopping is possible before collision occurs with objects.	Stopping can be affected by traction, gradient, steering, hardware or vehicle damage.	Sensors. Actuators. SW: Driver.
SR2-2	Vehicle restricts speed such that it can manoeuvre to avoid objects in its path before collision occurs.	Steering can be affected by speed, camber, traction, rate of turn or hardware and vehicle damage.	Sensors. Actuators. SW: Pilot, Driver.
SR2-3	Vehicle restricts speed such that it can plan a new path to avoid objects.	Sufficient time is allowed for vehicle to plan new path even in complex environments.	Sensors. SW: Planner.
SR2-4	In cases where GPS is lost or blocked, vehicle to maintain last good path until GPS signal is re-established.	New plans can be formed relying on IMU data but this carries significant risk.	Sensors. Actuators. SW: Planner, Platform Manager.
SR2-5	On re-establishing a GPS, the vehicle converges path differences between estimated position and true position in a safe manner.	Convergence carries significant risk, as the degree of positional error is impossible to predict and object avoidance may be impossible if vehicle needs to "teleport" to true position.	Sensors. SW: Planner.
SR2-6	If vehicle is unable to maintain a planned path, vehicle is brought to emergency stop	Ability to plan a new path is limited by CPU processing and sampling speeds, current speed of vehicle and complexity of environment.	Sensors. SW: Pilot, Driver, Platform Manager.

The software safety case patterns require that the software contributions to the hazard in question have been identified. There are various ways this top level contribution can be obtained; for example, it might occupy one node in a causal model, such as a fault tree analysis. For the instantiations of the software safety case pattern to be as easily as possible, it is important that the high level software contribution to the hazard is clearly understood and defined, as this forms the starting point of the software safety case and defines the context in which the case is made.

The first thing that the use of the patterns helped to do was to highlight that the top level software contribution to the hazards had not been clearly identified for the prototype UGV. The reason for this is that safety requirements are not typically expressed by describing how the system or software can contribute to a hazard. Instead, safety requirements tend to be framed in language that states the requirement as “necessary to mitigate” the hazard. Thus from our system safety requirements we have SR2 decomposed to SR2-4 (see Table reftab2, note have selected just that which is related to the loss of GPS signal referred to above):

### **System Safety Requirements**

#### **SR2**

While moving, the UGV avoids collisions with objects.

#### **SR2-4**

In cases where GPS is lost or blocked, vehicle maintains last good path until GPS signal is re-established.

Neither of these explicitly defines the hazard or mentions the software contribution to it. In fact SR2-4 does not specify how the vehicle should maintain a good path; it could be through hardware, software or a combination of both. The use of the patterns highlights the importance of making the software contributions to hazards explicit. Note that for our purposes, we are only concerned with the software contribution with regards to loss of the GPS signal (there are other software contributions we do not define here). We were able to transform the requirement above into the following expression of a contribution to a hazard.

#### **Hazard described in SR2**

UGV collides with static object.

#### **Software contribution to Hazard in SR2**

Software fails to plan safe path for vehicle when GPS signal is lost or blocked.

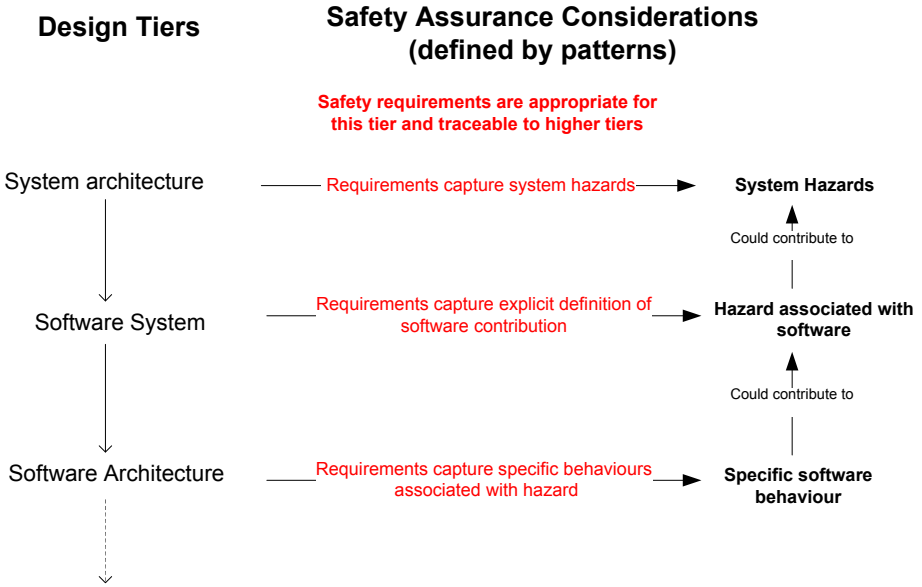
This rewriting of SR2-4 gives a clear starting point from which to construct a safety argument using the patterns. From this point downward in the decomposition we can refer to safety requirements over the software, as shown by the software safety requirements extract in Table 2 that decomposes SR2-4 by apportioning software safety responsibilities (note that the full decomposition is much more detailed and goes down to the level of individual program functions and variable declarations).

**Table 2.** Transition from system to software safety requirements

Software Safety requirement	Description	Notes / mitigation in design / other risks.	Risks / hazards introduced as a function of design decision
SR2-4	In cases where GPS is lost or blocked, vehicle to maintain last good path until GPS signal is re-established.	New paths can be formed using IMU data but this carries unspecified levels of risk.	
SR2-4-1	Where GPS signal is momentarily lost, software takes positional input from IMU to continue planning new paths until GPS signal is re-established.	Paths planned without GPS data become increasingly inaccurate. If vehicle starts to skid or loses traction, IMU data becomes unreliable.	Positional "drift" can grow in an unbounded fashion during loss of GPS signal, therefore vehicle could collide with an object it knew about and had planned to avoid.
SR2-4-2	If vehicle experiences loss of GPS signal for longer than 30 seconds, software brings vehicle to halt.	Bringing vehicle to halt within a GPS "tunnel" may result in vehicle being unable to continue mission.	Vehicle falls into enemy hands or becomes "lost" to accompanying UAV.

**Case Study Findings.** Before the patterns were used to guide the development of the safety case argument an explicit distinction had not been made between the system level safety requirements and the software contribution to the hazard. This led to some confusion when first instantiating the patterns, as the patterns require that the software contributions to hazards are identified. This is a strength of the software safety argument patterns, as they encourage the derivation of the software contributions and this represents good practice for software safety. The diagram in Figure 2 illustrates how the patterns force this distinction to be made - at the different design tiers, it must be demonstrated that the safety requirements are appropriate for that tier and are traceable to higher tiers. It can be seen that as the design tiers become more detailed and more software-specific, so the safety requirements for that tier must do also. Figure 2 shows how clear traceability can thus be established up to the system hazards.

We should note, here, that the system in this case study was a prototype product whose design is yet to be finalised. Perhaps we should not expect such a product to have something as specific as a fault tree (which would isolate the software contribution to the hazard in language that is more suited to the pattern); this type of safety analysis is more common on products with a higher level of technology readiness. Here, we carried out a fairly intuitive system to software safety requirements decomposition - given our prototype's operational scenario, and the fact it is undergoing further development, this is probably not inconsistent with real industrial practice. If this is the case, then care needs to be taken to explicitly define both the hazard and the top level software contribution to it, perhaps outside the main decomposition tables. Indeed, whether using the



**Fig. 2.** Transition from system to software safety requirements

patterns or not, it is beneficial to carry this exercise out so as to have a clear understanding of the software contribution across the system. As everything below this point in the decomposition will be a safety requirement on the software rather than the system, the software safety requirements can be inserted into the corresponding tiers of the pattern alongside the evidence selected to meet those requirements.

It should be noted that this Wildcat UGV case study was carried out by researchers who had no prior experience of using the patterns and limited experience with safety cases in general. Despite this, implementing the patterns was relatively easy, and helped ensure we had adequately covered the necessary assurance considerations. The patterns themselves provide a well structured framework within which to document design rationale regarding mitigation of hazards and justification of evidence. By tying this to tiers within the software architecture, the patterns make it obvious where to locate arguments about design decisions at a particular level. This, in turn, ties argument claims closely to specific elements in specific design or implementation artefacts, which helps argument assessors judge the sufficiency of the resulting argument.

### 3.2 Aircraft Safety Critical Software System Case Study

The system considered in this case study was a safety critical aircraft avionics system. The system comprised of a single line replacement item; the software for this was the subject of the case study.

The potential safety hazards associated with the system are partially mitigated by means of hardware safety interlocks independent of the system software. This approach minimises the contribution to safety from the software. Software involved hazards can also be addressed by ensuring that the integrity of the CPU commands to the hardware is sufficient to mitigate these hazards. This was achieved by the use of a high integrity Safety Monitor component within the main application software. The application software is split into two components, the Controller and the Safety Monitor. The Controller implements all of the actual system functionality, but all critical outputs are routed as requests to the Safety Monitor. The Safety Monitor sees the same set of real world inputs as the Controller and continuously calculates the safety state of the system. All critical outputs which are passed from the Controller to the Safety Monitor are checked against defined System Level Safety Properties, and the Safety Monitor vetos any outputs which would infringe any of the safety properties. Any safe outputs (those which are determined not to infringe any safety properties) are routed by the Safety Monitor onto the software device drivers.

The system level safety properties are the necessary conditions under which the behaviour of the system is considered to be safe. The properties were identified from the system hazards during system Preliminary Hazard Identification and Analysis. A formal definition (using the Z specification language) of the safety properties was provided in order to state the necessary conditions precisely and unambiguously.

This case study was undertaken at a fairly early point in the software development lifecycle. However, even at this early stage enough information was available about the design and development of the software, and plans in place for the later stages of the development, that a detailed software safety argument could be formed using the software safety argument patterns as guidance, particularly for the aspects of the argument relating to the Safety Monitor

**Case Study Findings.** The use of the software safety argument patterns highlighted a number of potential assurance deficits associated with the software. Thus identified, the significant assurance deficits could be dealt with. If the patterns had not been used then the assurance deficits may well not have been discovered until later in the development process, increasing the cost and possibly causing schedule slips of the system. Here, we will focus our discussion on one particular deficit, which relates the provision of sufficient evidence for certain safety properties.

The issue is illustrated in Figure 3. The left-hand side of the figure shows the tiers of design for the Safety Monitor software, while the right-hand side shows one of the safety assurance considerations at that tier (as determined from the patterns) and how that consideration is met. The argument patterns demand that direct evidence of satisfaction of safety properties is provided. The diagram shows the way in which this evidence was provided for the software at each design tier.

It can be seen that at the software system level, evidence is generated by performing system tests that check the behaviour of the software is as defined by the

safety properties specification. At the level of the software architecture, a separate specification is defined for each architectural element (module). Evidence can be generated at this tier through module testing against the specification for each module. Note that this evidence is not directly checking the behaviour of the module against that defined by the safety properties specification. This is acceptable as long as the safety properties required of the safety monitor module have been correctly captured in the safety monitor specification. The patterns highlight the importance of demonstrating that the safety properties are adequately interpreted for the Safety Monitor module.

For the class design of the safety monitor module there can be seen to be no evidence which can directly show that the classes behave in accordance with the safety properties. Although unit testing is performed, this is evidence only that the Safety Monitor behaves according to the design specification. In order for unit testing to meet the safety assurance consideration, we also need assurance that the class design correctly captures the required high-level safety properties. Again, the patterns highlighted the importance of adequately interpreting the safety properties for each of the classes in the safety monitor module design.

Finally in Figure 3, it can be seen that static analysis is provided as evidence at the level of the source code. The analysis was conducted using SPARK proof annotations [6] included in the safety monitor code. Again, for this evidence to be effective from an assurance perspective, it must be demonstrated that the proof annotations completely and correctly capture the required safety properties.

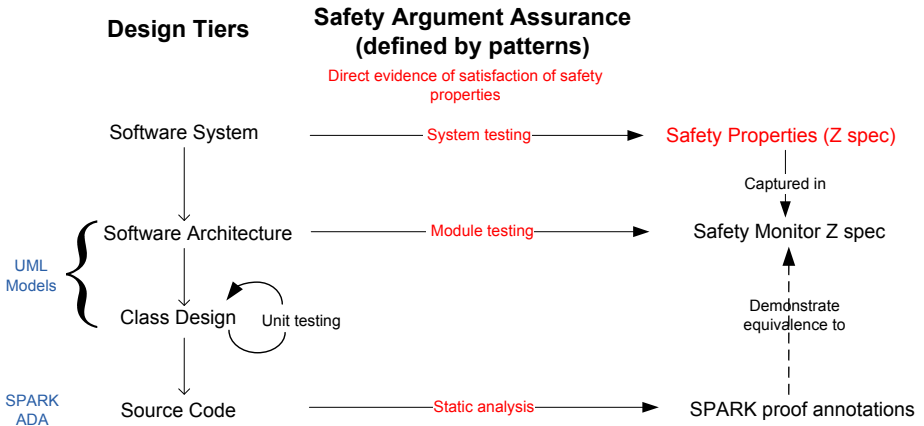


Fig. 3. Transition from system to software safety requirements

This example illustrates how, by encouraging the developer to consider explicit assurance claims relating to the required safety properties at every tier of design decomposition, the software safety patterns highlighted the potential pit-falls of any “gap” between the safety properties themselves and the tier against which evidence is provided. The patterns identified that the most effective strategy



from an assurance perspective would be to explicitly interpret the required safety properties at each level of design decomposition.

Other potential assurance deficits were highlighted by the use of the patterns in this case study. A lack of assurance regarding potential hazardous failures at each design tier was identified. Every time there is a decomposition in the design, there is the potential to introduce erroneous behaviour into the design which could manifest itself as a hazardous software failure. There had been analysis conducted to identify new or additional failure modes of the software at the architecture level (although this was fairly unstructured), but application of the patterns highlighted a need to perform similar analysis at other levels of design.

## 4 Conclusions

The case studies we have undertaken have given us confidence that they have the desirable criteria defined at the start of this paper:

- The patterns should be easy to understand and apply by software development teams.
- The patterns should be flexible enough to apply to any safety-critical software system.
- The patterns should ensure that the resulting software safety argument is explicitly focused on controlling the software contribution to system hazards.
- It should be easy to judge the sufficiency of an argument created using the patterns.

It is clear that the patterns are fairly easy to understand. This has been demonstrated through the relative ease with which they were applied to the autonomous system by people completely unfamiliar with the patterns.

It has been shown that the patterns are very flexible. The two case studies reported here were on very different types of software system, but the patterns proved to be equally applicable. This was particularly reassuring in the case of the prototype autonomous vehicle, which is a novel system at an early stage of development.

Both case studies made it clear that the resulting safety assurance argument is very focused on demonstrating how the software contributions to system hazards are controlled. This is an advantage over the unfocussed safety arguments that are often produced. It was seen in the case of the aircraft software system that the development team noted how the structure of the generated argument helped to clearly highlight to them which of their software safety and development activities were most important from a safety assurance perspective. In particular, they commented that the case study revealed that many of the things that they focus their attention on are general assurance activities, rather than activities that explicitly help to address specific software contributions. This could help to focus attention on the activities which are most important to software safety assurance. In addition, it makes it easier to judge the sufficiency of the resulting

argument, since the relationship between the generated evidence and the safety of the system was clear and explicit.

Most importantly, the case studies have shown that applying the patterns can identify potential assurance issues, which can then be addressed as early as possible. If left unidentified, such issues could lead to safety problems during operation.

The case studies have also identified areas where more work on the patterns would be beneficial. In particular we think that clearer guidance is required on the process of instantiating the patterns for a particular application. This would seem to be particularly required when creating large, complex safety cases. For such large complex software systems, it would also be beneficial to provide guidance on how to group arguments with respect to the corresponding design elements in order to keep a clear relationship between the software design structures, and the structure of the argument.

The argument structures in the software safety argument patterns discussed in this paper are broadly in line with the new “assured safety case” structure presented by Hawkins et al in [7]. The patterns will be reviewed to ensure they are completely consistent with that structure. The constraints in this new format for safety cases have the potential to further focus software safety arguments on those claims and evidence that matter the most.

**Acknowledgements.** The authors would like to thank the Systems Engineering for Autonomous Systems Defence Technology Centre (SEAS DTC) and the Software Systems Engineering Initiative (SSEI), both funded by the UK Ministry of Defence, which supported the two case studies described in this paper.

## References

1. Hawkins, R., Kelly, T.: A Systematic Approach for Developing Software Safety Arguments. In: Proceedings of the 27th International System Safety Conference, Huntsville, AL (2009)
2. Menon C., Hawkins R., McDermid J.: Interim standard of best practice on software in the context of DS 00-56 Issue 4. Technical Report SSEI-BP-000001. Software Systems Engineering Initiative, York (2009), <https://ssei.org.uk/documents/>
3. Weaver, R.A.: The safety of Software - Constructing and Assuring Arguments. PhD thesis, Department of Computer Science, The University of York (2003)
4. Kelly, T.: Arguing Safety - A Systematic Approach to Managing Safety Cases. PhD thesis, Department of Computer Science, The University of York (1998)
5. Ye, F.: Justifying the Use of COTS Components within Safety Critical Applications. PhD thesis, Department of Computer Science, The University of York (2005)
6. Barnes, J.: High Integrity Ada - The SPARK Approach. Addison Wesley, Reading (1997)
7. Hawkins, R., Kelly, T., Knight, J., Graydon, P.: A New Approach to Creating Clear Safety Arguments. In: Proceedings of the Nineteenth Safety-Critical Systems Symposium (SSS 2011), Southampton (2011)
8. Jaffe, M., Busser, R., Daniels, D., Delseny, H., Romanski, G.: Progress Report on Some Proposed Upgrades to the Conceptual Underpinnings of DO178B/ED-12B. In: Proceedings of the 3rd IET International Conference on System Safety (2008)

9. Systems Engineering for Autonomous Systems (SEAS) Defence Technology Centre (DTC) <http://www.seasdtc.com/>
10. Bardo B.: Autonomous Systems — A New Partnership Between Man and Machine. Presentation to SEAS DTC (2010), <http://www.innovate10.co.uk/uploads/BillBardo-theSEASDTC.pdf>
11. Alexander, R., Herbert, N., et al.: Deriving Safety Requirements for Autonomous Systems. In: Proceedings of the 4th SEAS DTC Technical Conference, Edinburgh (2009)
12. Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice. In: Proceedings of the Requirements Engineering Conference, 12th IEEE International (2004)

# Integration of a System for Critical Infrastructure Protection with the OSSIM SIEM Platform: A dam case study

Luigi Coppolino<sup>1</sup>, Salvatore D'Antonio<sup>2</sup>,  
Valerio Formicola<sup>2</sup>, and Luigi Romano<sup>2</sup>

<sup>1</sup> Epsilon S.r.l., Naples, Italy

luigi.coppolino@epsilononline.com

<sup>2</sup> University of Naples "Parthenope", Department of Technology, Italy  
{salvatore.dantonio,valerio.formicola,luigi.romano}@uniparthenope.it

**Abstract.** In recent years the monitoring and control devices in charge of supervising the critical processes of Critical Infrastructures have been victims of cyber attacks. To face such threat, organizations providing critical services are increasingly focusing on protecting their network infrastructures. Security Information and Event Management (SIEM) frameworks support network protection by performing centralized correlation of network asset reports. In this work we propose an extension of a commercial SIEM framework, namely OSSIM by AlienVault, to perform the analysis of the reports (events) generated by monitoring, control and security devices of the dam infrastructure. Our objective is to obtain evidences of misuses and malicious activities occurring at the dam monitoring and control system, since they can result in issuing hazardous commands to control devices. We present examples of misuses and malicious activities and procedures to extend OSSIM for analyzing new event types.

**Keywords:** Critical Infrastructure Protection, SIEM, dam, OSSIM.

## 1 Introduction

Misuses and malicious activities occurring at systems for Critical Infrastructure Protection (CIP) can have catastrophic consequences, such as financial losses and danger for life [1]. We refer to misuses as unintentional incorrect uses of the system: for example, unintentional violations of the operating procedures guaranteeing safety for users, staff and people in general. Instead, we refer to malicious activities as conscious activities aimed at compromising the correct operation of the system: for example, cyber attacks to communication networks supporting the critical infrastructures.

In recent years the monitoring and control devices in charge of supervising the critical processes of Critical Infrastructures have been victims of cyber attacks [2][3]. Typically the supervision of critical processes (i.e. key processes for

the critical infrastructure operation and for providing services) is realized by means of devices able to measure and modify process state parameters (namely sensors and actuators). Particularly the attacks have turned into intrusions, by exploiting the vulnerabilities of the Commercial-Off-The-Shelf (COTS) components, such as the legacy Supervisory Control And Data Acquisition (SCADA) systems: the SCADA systems are hardware and software solutions widely used to perform monitoring and control operations.

In such a scenario, the organizations that provide critical services, such as energy, water, oil, gas distribution, transportation, have to face several challenges: avoid regulation, policy and procedure violations; protect their network infrastructure from cyber attacks; guarantee proper operation of monitoring and control systems for the safeguard of population, staff and users.

To provide network protection, currently adopted solutions are based on management tools that assess the global level of security of the network infrastructure. Particularly interesting from this perspective are frameworks based on Security Information and Event Management (SIEM) systems, since they are able to analyze in a single point the reports produced by several kinds of devices deployed over the network infrastructure. Specifically the analysis of the SIEM framework is based on gathering and correlating the operating and security reports (also called "events") generated by Information and Communication Technology (ICT) appliances, applications and security tools, finally producing detailed and concise reports about the security level of the occurred events.

In this work we propose an extension of a commercial SIEM framework, namely OSSIM by AlienVault, to perform the analysis of the events reported by the components responsible for monitoring, controlling and protecting the processes and the operation of a critical infrastructure, specifically a dam. Our objective is to obtain evidences of malicious activities and misuses on the dam monitoring and control system, since they can result in issuing hazardous commands to the control devices.

We present our work in three main tasks. (1) We provide some examples of misuses and malicious activities that could result in issuing hazardous commands to the dam control devices. (2) We show how to extend the SIEM framework to process events generated by security, monitoring and control devices of the dam infrastructure (such as structural and environmental sensors): we have adopted the open source product OSSIM, developed and maintained by the AlienVault company, since it is extensible and highly customizable and allows to build components able to analyze new kinds of events. (3) We show how to implement new correlation rules in OSSIM in order to exploit the information of the events generated by security and process control systems and obtain evidences of misuses and malicious activities.

In Section 2 we present related works about most advanced technologies for dam and critical infrastructure monitoring and to supervise systems for CIP by means of SIEM based tools. In Section 3 we give more details about the dam monitoring and control systems. In Section 4 we describe the SIEM framework technology and the OSSIM product. In Section 5 we provide some examples of

misuses and malicious activities and give details about the implementation of rules and plugins in OSSIM.

## 2 Related Work

This section shows that works proposing innovative approaches and technologies to monitor and control dam infrastructures do not address security issues [4] [5] [6]. On the other hand, works proposing to enforce the security of systems for CIP by means of SIEM based frameworks, give no relevance to the events related to critical process domain.

### 2.1 Use of SIEMs for Critical Infrastructure Protection

The DATES (Detection and Analysis of Threats to the Energy Sector) project, sponsored by the National Energy Technology Laboratory (NETL) of the U.S. Department of Energy, develops several intrusion detection technologies for control systems [7]. DATES adopts the commercial SIEM ArcSight to detect a network "traversal" attack to a corporate or enterprise network: the paper describes how to detect, by means of a SIEM framework, the attempt of controlling the system managing the critical processes; the attack is described as sequential violation of machines on the network hosting the field monitoring and control devices. The SIEM correlates intrusion events related to the hosts on the field network: anomaly based Intrusion Detection Systems are in charge of revealing attack attempts by looking at deviations from the normal behavior of field devices. In [8] is presented a joint work of Universidad ICESI and Sistemas TGR S.A. to implement the Security Operations Center (SOC) Colombia product. The system extends the OSSIM SIEM to evaluate reports produced by two different kinds of physical devices, specifically a fire alarm panel and an IP surveillance camera. Moreover it introduces a new interface to facilitate the creation of new correlation rules.

### 2.2 Advanced Monitoring and Control for Dam Infrastructure

In [4], the Korean Water Resource Corporation (Kwater) multi-purpose dam safety management system (KDSMS) has been proposed. KDSMS implements a workflow to coordinate the surveillance activities of dam field engineers, dam staff located in headquarter offices and experts in remote research centers. The framework manages different personnel profiles: each identity is responsible for transmitting reports, approving and notifying actions and for correlating different kinds of evidences or to require proper controls at the dam infrastructure.

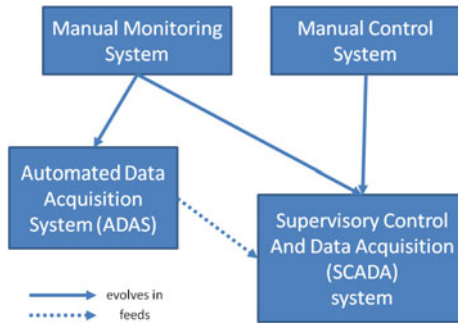
In [5] a work by the Technical University of Lisbon is presented. The paper presents the prototype of a knowledge-based system to support engineers responsible for dam safety assessment. The system, named SISAS, is composed of a centralized management tool in charge of analyzing sensor data to evaluate the dam health, by comparing the measures with alert thresholds computed from

reference models. The final diagnosis is reported to the operator by means of graphical interfaces, providing suggestions to recover from dangerous situations.

In [6], is presented a work financed by the Swiss Nation Center of Competence in Research (NCCR) Mobile Information & Communication Systems (MICS) and the European FP6 Wirelessly Accessible Sensor Populations (WASP) project. SensorScope faces the issue of effective monitoring in harsh weather conditions by means of wireless sensors. The prototype is composed of a sink station box and a wireless sensor network made of TinyOS based devices. The framework is in charge of retrieving measures of meteorological and hydrological parameters, such as wind speed and direction.

### 3 Dam Monitoring and Control

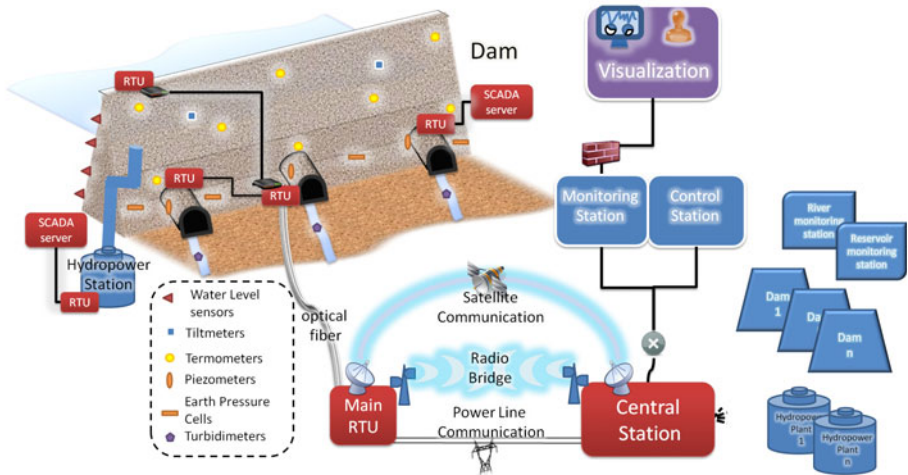
Dam infrastructure is designed to provide services related to the usage of water reservoirs: food water supplying, hydroelectric power generation, irrigation, water sports, wildlife habitat granting, flow diversion, navigation are just some examples.



**Fig. 1.** Automation evolution in dam monitoring and control systems

Since dam infrastructure has large geographical extension, monitoring and control operations must be performed in distributed fashion. In addition, some critical controls need to be orchestrated among several remote sites: for example the mechanisms related to the production of hydroelectric power require to control the reservoir discharges feeding the downriver plants. To monitor and control such a complex system, a large number of devices are deployed. Typically these devices are technologically heterogenous and present several levels of automation: old control systems require heavy manual interaction with human operators; more advanced systems allow real-time environmental analysis and perform automatic control procedures. In Figure 1 we show the evolution trend for these systems: the Automated Data Acquisition System (ADAS) is designed to acquire measures and data from sensor devices deployed over the dam infrastructure and store and transmit them to personal computers for assessment.

Typically personal computers are placed next to sensor devices or in remote locations. In addition to automatic acquisition of measures, the Supervisory Control And Data Acquisition (SCADA) systems are designed to supervise and control the processes, by issuing commands to configure the actuator operations. Since ADAS is designed to perform monitoring operations, its components can be adapted to the SCADA system: for example, sensors can provide measures to the SCADA devices. As a matter of fact, currently deployed ADASs include devices able to perform control operations autonomously.



**Fig. 2.** Deployment of the dam monitoring and control system

Figure 2 shows some components of a SCADA-based dam monitoring and control system. We have not represented the ADAS, since its functionalities are implemented by the SCADA system. Both SCADA systems and ADASs include instruments to measure geotechnical parameters related to dam structure, water quality, water flows, environment, mechanism states, device states [9] [10]. In the context of automated monitoring, the instrumentation is based on sensors producing analog or digital signals. Measures are collected by devices placed next to or inside the dam facilities: Remote Terminal Units (RTUs) in the context of SCADA systems, and Remote Monitoring Units (RMUs) in the context of the ADASs. RTUs are in charge of converting sensor signals to digital data and sending them to remote SCADA system components or master RTUs (Main RTUs). Similarly, RMUs are in charge of transferring the measures to local or remote personal computers. Typical RMU devices are the Data Loggers.

ADASs include Monitoring and Control Units (MCUs) able to perform control operations autonomously. Similarly, more advanced RTUs and the Programmable Logic Controllers (PLCs) are able to control actuator devices.

Typically the ADAS parameter assessment is performed by application specific softwares installed on general purpose personal computers. PCs can be hosted in dam facilities or in remote locations. The SCADA system adopts a



supervisor server (SCADA server) displaced next to the monitored process or in remote locations. Typically the SCADA server manages the dynamic process database, system logics, calculations, alarm database. The SCADA system includes client applications (SCADA clients) hosting process specific Human Machine Interfaces (HMIs). In figure 2 the Monitoring station represents the machine hosting the SCADA client; the Control station hosts the SCADA server and other components such as storage units and databases.

To realize short distance connections (for example RTU-main RTU, RTU-local SCADA server, RMUs-PCs), both SCADA systems and ADASs rely on several kinds of solutions: typically Local Area Networks based on fiber optic, telephone and Ethernet cables. For long distance connections Wide Area Networks based on power line communications, radio bridges, satellite links, cellular networks, telephone lines.

Central stations host SCADA components in charge of orchestrating the monitoring and control operations of several infrastructures using the water reservoirs. As shown in the figure, the Central station supervises several dams, hydropower plants, flow monitoring stations and other remote monitoring and control sites. The Visualization stations are public access zones, deployed to show the dam "health" to population living near to the reservoir or at downriver. Visualization station and some components of the Monitoring, Control and Central stations are composed of typical ICT devices like routers, firewalls, Intrusion Detection Systems, Intrusion Prevention Systems, Web Server, Databases, Storage Units, Application servers, Gateways, Proxies. Moreover, identification devices perform access control to the SCADA units.

### 3.1 Dam Sensors

Physical sensors are adopted to monitor the operating environment conditions of the dam: monitoring some parameters is necessary to guarantee safe execution of critical processes, avoid hazardous controls and prevent possible critical failures or damages to the dam infrastructure. To monitor environmental parameters, several instruments and sensor devices exist. We provide a short list of them and a brief explanation of their principal usage in Table 1.

**Table 1.** Dam instrumentation

<i>Instrument</i>	<i>Parameter or physical event</i>
Inclinometer/Tiltmeter	Earth or wall inclination or tilt
Crackmeter	Wall/rock crack enlargement
Jointmeter	Joint shrinkage
Piezometer	Seepage or water pressure
Pressure cell	Concrete or embankment pressure
Turbidimeter	Fluid turbidity
Thermometer	Temperature

With regards to the most advanced technologies, we mention the smart sensors. These devices have increased the capabilities of the metering process in several aspects. Indeed, smart sensors have introduced the possibility to process the measures on the sensor boards, sending alarm messages in case of suspect environmental conditions. Other kinds of smart devices, such as the sensors of the Wireless Sensor Network (WSN), have provided capabilities in terms of protection mechanisms able to isolate faulted and misbehaving nodes (also named "motes").

## 4 SIEMs Overview

Security Information and Event Management (SIEM) systems are tools in charge of assessing the security level of the network infrastructure, by processing the reports generated by ICT applications, appliances and security devices deployed over the network. One of the most negative aspects of currently deployed security systems is the generation of too false positives: this limits their effectiveness since some relevant events pass unnoticed to the administrator behind the multitude of events. Main objective of the SIEM is to reduce this high false positive rate and emphasize the occurrence of events otherwise unnoticed. Its main functionality is to centralize the event analysis and produce a detailed and effective report by a multi-step correlation process.

Follows a description of the SIEM framework components.

*Source Device* is the component producing information to feed the SIEM; reports of normal or suspicious activities are generated by applications (Web Server, DHCP, DNS,...), appliances (router, switch,...) or operating systems (Unix, Mac OS, Windows,...). Even if not strictly part of the SIEM architecture, the Source is a fundamental component for the SIEM framework. Typically, most of the reports are logs in application specific format.

*Log Collection* component is responsible for gathering logs from Source Devices. It works adopting push or pull based paradigm.

*Parsing and Normalizing* component is in charge of parsing the information contained in the logs and to traduce this from the native format to a format manageable by the SIEM engine. Moreover, the Normalization component is in charge of filling the reports with extra information required during the correlation process.

*Rule and Correlation Engines* trigger alerts and produce detailed reports; they work on the huge amount of logs generated by the Source Devices. The Rule engine raises the alert after the detection of a certain number of conditions, while the Correlation Engine correlates the information within the evidences to produce a more concise and precise report.

*Log Storage* component stores logs for retention purposes and historical queries; usually the storage is based on a database, a plain text file or binary data.

*Monitoring* component allows the interaction between the SIEM user and the SIEM management framework. Interactions include report visualization, incident handling, policy and rule creation, database querying, asset analysis, vulnerability view, event drilling down, system maintaining.

As we will see soon, these components are all implemented and customizable in the OSSIM SIEM.

## 4.1 OSSIM

OSSIM (Open Source Security Information Management) [11] is an open source SIEM released under the GPL licence and developed by AlienVault [12]. OSSIM does not aim at providing new security detection mechanisms but at exploiting already available security tools. OSSIM provides integration, management, and visualization of events of more than 30 [13] open source security tools. More important, OSSIM allows the integration of new security devices and applications in a simple way.

All the events collected by OSSIM undergo a process of normalization in order to be managed by the SIEM core. After the normalization, OSSIM performs event filtering and prioritization by means of configurable policies.

OSSIM provides capabilities in terms of event correlation, metric evaluation and reporting. Indeed, OSSIM performs per event risk assessment and correlation process. Correlation process produces events more meaningful and reliable than those generated by single security tools. The objective of the correlation is to reduce the number of false positives to produce less reports for human operator.

OSSIM performs three types of correlations:

**Inventory Correlation.** performs event filtering by assessing the possibility that a given attack may affect a specific kind of asset (i.e. a Windows threat to a Linux box).

**Cross Correlation.** re-evaluates the event "reliability" by comparing each event with the result of the vulnerability analysis (i.e. if an event reports an attack to an IP and that host is vulnerable to that attack, the event reliability raises).

**Logical Correlation.** executes the correlation directives defined by condition trees. Conditions are built on Boolean logic expressed in hierarchical structures. Correlation directives are customizable and configurable by the user.

**Architecture.** OSSIM architecture is based on software components that can be deployed in several ways across many networks: in this way, OSSIM can be used to monitor different network domains. Main components of OSSIM are represented in Figure 3:

**Detector:** it is any tool that supervises the assets. Detectors are in charge of reporting operating or security-related "events". OSSIM is already enabled to be connected with a huge number of Detector tools and the Collectors of these tools come already packaged within the framework. Other Detectors can be added to OSSIM by developing new Collectors enabling OSSIM to accept new types of events.

**Collector:** it is the component in charge of: (1) gathering events form different sensors; (2) parsing and normalizing the events; (3) forwarding the

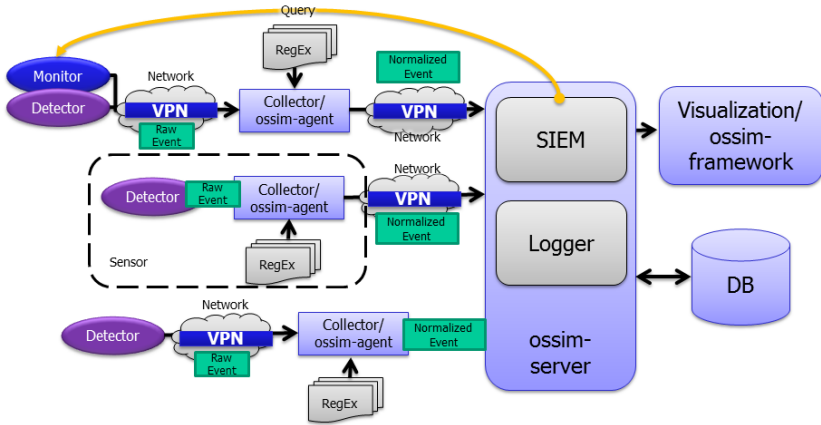


Fig. 3. Deploymet of OSSIM components

normalized events to the Server component. The software component that implements these tasks is the OSSIM-Agent. Event collection is organized in a plugin based system. Each different source of events is associated with a plugin able to parse and normalize a specific data format. Event parsing and normalization is performed using Python style regular expressions.

**Monitor:** this component is very similar to the Detector, but it’s activated only after a request (query) by the Server. Monitors generate ”indicators”. The OSSIM server can query a Monitor to gather additional information and perform a more precise correlation process. Indicators reach the Server by means of the same mechanisms used for the ”events”.

**Sensor:** it is the combination of the Detector tool and the related Collector.

**Server:** the OSSIM-Server component implements the intelligence of the SIEM. The Server component has two main functionalities: SIEM and Logging. The SIEM subsystem performs risk assessment, correlation, and real-time monitoring; moreover, it allows mechanisms for vulnerability scanning. The SIEM behavior is totally configurable through policies. Polices are used to set up event management and alert creation. The Logger subsystem is used to store raw events for forensic analysis. All the events are provided with a digital signature that allows their use for legal evidence. OSSIM supports encrypted channels from event source to Data Logger component.

**Database:** the OSSIM-Database is a MySQL database used to store both configurations (handled by means of the web interface) and the asset inventory.

**Web interface:** it is implemented in the OSSIM-Framework software. It provides the visualization interface of the entire framework. It allows the handling of all the events and alarms generated by Sensors and OSSIM-Server. The Web interface is used to configure policies, to perform network scanning, to query the database. The web interface is implemented in PHP and HTML code and runs on a Python daemon process.

Figure 3 represents a typical architecture of the OSSIM framework. Detectors can be deployed either along with their Collectors or separately. In the latter case the raw events produced by Detectors must reach the Collector through the network. To transfer these events, OSSIM adopts several protocols: Rsyslog, FTP, SAMBA, SQL, OSSEC, SNARE among others. Collector components can be deployed together with the OSSIM-Server or remotely. In the latter case, to protect the Collector-Server communication, Virtual Private Networks are set up. Monitors are deployed just like Detectors, but are triggered by the Server.

## 5 Changing the SIEMs to Provide Safety

As seen, OSSIM actions can be described in the following steps: extract information from events (or indicators) generated by the Source Devices deployed over the network infrastructure; apply a policy and execute the correlation process to perform risk assessment; finally, raise an alert message (and a *ticket*) to the administrator. Our main idea is to detect misuses and malicious actions (such as cyber attacks) occurring at the monitoring and control system of the dam infrastructure. We correlate reports produced by ICT appliances and applications, monitoring and control devices (physical sensors, SCADA servers, PLCs, RTUs, RMUs), security devices and applications (identification, authentication mechanisms,...). We describe how to implement new plugins for the OSSIM-Agent and write new correlation rules (*directives*) to detect misuses and malicious activities.

### 5.1 Examples of Misuses and Malicious Activities on the Dam Monitoring and Control System

In this section we provide some examples of misuses and malicious activities occurring at the dam monitoring and control system. Moreover we indicate some possible events to be correlated by the SIEM. We remark that this list is far from being exhaustive and is a hint for future considerations about the safety and cyber security relationship in systems for CIP.

**Alteration of Measurement Data:** Physical sensors measure unexpected values: for example the piezometer measures water levels out of the structural range or expected profile (in specific environmental conditions). The SIEM framework correlates this event with events or indicators produced by security Source Devices and evaluates the probability of sensor device tampering. Examples of security events are: changes in the sensor devices' Operating System fingerprint, traffic profile anomalies on the field network, connection attempts to the machines controlling the sensors. Altering measurement data is dangerous because parameters out of range can trigger automatic control procedures.

**Malicious Control Commands:** Parameters have sudden changes further to controls operated by actuators. The SIEM correlates this physical event with events reporting controls issued by the SCADA server. Moreover the SIEM verifies if the control issued by the SCADA is consequence of a predefined

behavior, like a scheduled operation or a change in the automatic control procedure. In the latter case the SIEM correlates these events with security events related to the SCADA system components.

**Missing Control Commands:** Sensors measure physical parameter values inside the expected ranges and/or the sensor supervisor unit does not report any alerting event (or reports a normal event). In case of critical conditions for the infrastructure, altered measures or missing reports can result in missing safety controls causing damages to the dam infrastructure ("dam failures"). Events about malicious activities against the SCADA system components can be correlated with the analysis of model-based physical predictors.

**Control Station Hacking:** The control station issues control commands modifying the procedures of the RTU or PLC systems. Controls or changes are issued by an operator enabled to access the control station, but not to perform these operations, as reported by the identification procedure. The identification procedure can be realized by means of Radio Frequency Identification (RFID) or by biometric recognition devices.

## 5.2 Customizing OSSIM to Process New Events

To perform comprehensive event analysis by means of the SIEM framework, our first task has been to extend OSSIM with new Source Devices, in particular sensors, control units and security devices. Typically Source Devices generate two kinds of information, namely "events" and "indicators". Events are generated after specific occurrences: for example, SCADA servers and PLCs generate events reporting the issue of control commands or the detection of anomalous parameter profiles (i.e. threshold exceeding); security devices produce evidences of security related events (i.e. access to control station). Indicators are sent to the SIEM framework after a query by the SIEM server: RTUs, Data Loggers and SCADA servers can be in charge of retrieving measures of physical parameters; security devices can provide indicators about specific vulnerabilities (i.e. open ports). Indicators and events can be used to define the correlation rules to detect misuses and malicious activities, as shown in the following lines.

As seen in the previous section, data from "field" devices are transferred by RTUs with several communication protocols: DNP3, ModBus (TCP/IP), Profibus (DP/PA), Profinet, IEC 60870-5-104, ICCP, OLE for Process Control (OPC), OPC Unified Architecture, just to cite some. Even if OSSIM provides a large number of parsers for the most common log formats or communication protocols, users can provide new plugin modules to make the SIEM able to work with legacy or custom message formats. To integrate new parsers with the OSSIM SIEM, users extend the plugin set within the Collector component. The source type that feeds a plugin can be classified as "monitor" or "detector". Monitors are pull based sources and produce indicators, while detectors are push based and produce events. To feed the Collector agent, transfer methods or protocols must be specified (log, mysql, mssql, wmi): it's suggested to adopt the Syslog protocol, since it is more suitable to be parsed by the Regular Expression

```

<directive id="50000" name="Unauthorized user command issue from control station" priority="3">
  <rule type="detector" name="RFID_Authentication_not_field_engineer" reliability="2"
  occurrence="1" from="ANY" to="ANY" port_from="ANY" port_to="ANY"
  plugin_id="1001" plugin_sid="1,2">
    <rules>
      <rule type="detector" name="Control command issued" reliability="5"
      occurrence="1" from="1:SRC_IP" to="1:DST_IP"
      port_from="ANY" time_out="600" port_to="ANY"
      plugin_id="1002" plugin_sid="1">
    </rules>
  </rule>
</directive>

```

**Fig. 4.** Example of OSSIM rule

engine; moreover the Syslog server can be configured to filter events and drop values (as measures) out of valuable ranges.

RegExp (Regular Expression) plugins are in charge of extracting useful information and filling the *Normalized event*. Normalized events contain optional and mandatory fields useful to the correlation process. Each plugin is identified by the Id (unique) and more Sub-Ids (Sids); Sids are useful to create rules and correlation directives. Indeed, rules use Sids to identify different kinds of event messages generated by the data source (for example: plugin Id identifies logs produced by Apache servers in general, Sids identify specific events on the Apache server).

To feed OSSIM with environmental parameters we have adopted a commercial Data Logger. Data loggers can perform measurements and store or forward them to remote servers. The following string contains a sample log of the piezometer measure reporting the seepage or groundwater level.

```

D,088303,"JOB1",2011/03/11,11:27:02,0.016113,1;
A,0,8.621216e-06,-1.4952594;0075;CC8C

```

Normalized events are sent to the SIEM Server. The Server applies a policy to the event (Correlation, Forwarding, Action, Discard), basing on: Time Range, Plugin Group, Source and Destination Addresses, Ports. If not discarded, the event undergoes the *Risk Assessment* process and becomes *Enriched event*. The Enriched event has several metrics: Reliability, that represents how much the reported event is probably a suspect activity; Priority value, that represents the absolute importance of the event with no reference to a specific host or environment; Asset value, that states what is the importance of the assets involved in that event. The Risk is computed by combining metrics in a single value. After Risk Assessment, the event can be processed by the Correlation engine. Correlation produces a new event that, as such, is subjected to new Risk Assessment process. If the Risk is bigger than one, the Alarm event is generated. We give full scale value to Asset and Priority metrics involving events related to physical sensors.

Most interesting Correlation mechanism we considered is the Logical Correlation, implemented by means of directives written in xml syntax or, more easily, by means of visual editors (in the Web Interface). Main objective of the directive is to assess Priority and Risk for a certain number of collected events. Directives

specify the Id and Sid of the events involved in the rule: since the Id is unique per event source, we can correlate the Id of physical sensor messages with the Id of security devices, ICT applications, SCADA components and so on. Monitor directives can verify the met of several *conditions* (equal, less than,...) related to the event fields, such as measures; Detector directives are focused on event "occurrence", that is the repetition of the same event. In the latter case, the environmental monitoring units are in charge of transmitting to the SIEM the results of their analysis.

In Figure 4 we show a simple rule of the "Control station hacking" misuse: the first event is produced by RFID device (id=1001), where Sids 1,2 indicate the access by two employees with no authorization to issue commands; the second event is reported by an application on the Control station (id=1002) and is triggered by the execution of a new control command.

## 6 Conclusion and Future Works

Dams infrastructures are designed to provide services related to the use of water reservoirs (typically in conjunction with different infrastructures, such as hydropower plants). These complex systems are monitored and controlled by several components in charge of providing supervision during the processes. The monitoring and control procedures, orchestrated among several sites, are performed issuing commands and processing data by means of a large number of components (legacy COTS SCADA systems, ICT appliances and applications): typically these components are not designed with security in mind. In such a complex scenario, misuses and malicious activities can represent threats to society, safety and business. Indeed, malicious activities like cyber attacks, can be aimed at changing the automatic control procedures, alter the measures produced by sensor devices, issue control commands.

In this work we propose an extension of the OSSIM SIEM by AlienVault, to perform the analysis of events generated by the security devices (recognition devices, authentication tools,...) and process specific devices (SCADA servers, RTUs, ...) responsible to supervise the operations and the processes of the dam infrastructure. Our objective is to obtain evidences of misuses and malicious activities on the monitoring and control systems, since they can result in issuing hazardous commands to the control devices of the dam infrastructure.

In next works we reserve to perform a more detailed analysis about the safety and cyber security relationship within the systems for CIP and assess the reliability of the reports produced by our prototype.

**Acknowledgments.** The research leading to these results has received funding from the European Commission within the context of the Seventh Framework Programme (FP7/2007-2013) under Grant Agreement No. 225553 (INSPIRE Project), Grant Agreement No. 248737 (INSPIRE-INCO Project) and Grant Agreement No. 257475 (Management of Security information and events in Service Infrastructures, MASSIF Project). It has been also supported by the Italian



Ministry for Education, University, and Research (MIUR) in the framework of the Project of National Research Interest (PRIN) "DOTS-LCCI: Dependable Off-The-Shelf based middleware systems for Large-scale Complex Critical Infrastructures".

## References

1. Regan, P.J.: Dams as systems - a holistic approach to dam safety. In: 30th Annual USSD Conference Sacramento, California (2010)
2. White Paper, Global Energy Cyberattacks: "Night Dragon", McAfee@Foundstone@Professional Services and McAfee Labs (2011)
3. White Paper, Symantec@Intelligence Quarterly Report, Targeted Attacks on Critical Infrastructures, <http://bit.ly/g8kpzv> (October-December, 2010)
4. Jeon, J., Lee, J., Shin, D., Park, H.: Development of dam safety management system. *Advances in Engineering Software* 40(8), 554–563 (2009) ISSN 0965-9978
5. Farinha, F., Portela, E., Domingues, C., Sousa, L.: Knowledge-based systems in civil engineering: Three case studies. In: *Advances in Engineering Software. Selected papers from Civil-Comp 2003 and AICivil-Comp 2003*, vol. 36(11-12), pp. 729–739 (November-December 2005) ISSN 0965-9978
6. Ingelrest, F., Barrenetxea, G., Schaefer, G., Vetterli, M., Couach, O., Parlange, M.: SensorScope: Application-specific sensor network for environmental monitoring. *ACM Trans. Sen. Netw.* 6(2) Article 17 (2010)
7. Briesemeister, L., Cheung, S., Lindqvist, U., Valdes, A.: Detection, correlation, and visualization of attacks against critical infrastructure systems. In: Eighth Annual International Conference on Privacy Security and Trust (PST), 2010, August 17-19, pp. 15–22 (2010), doi:10.1109/PST.2010.5593242
8. Madrid, J.M., Munera, L.E., Montoya, C.A., Osorio, J.D., Cardenas, L.E., Bedoya, R., Latorre, C.: Functionality, reliability and adaptability improvements to the OS-SIM information security console. In: *IEEE Latin-American Conference on Communications, LATINCOM 2009*, September 10-11, pp. 1–6 (2009)
9. Myers, B.K., Dutson, G.C., Sherman, T.: City of Salem Utilizing Automated Monitoring for the Franzen Reservoir Dam Safety Program. In: 25th USSD Annual Meeting and Conference Proceedings (2005)
10. Parekh, M., Stone, K., Delborne, J.: Coordinating Intelligent and Continuous Performance Monitoring with Dam and Levee Safety Management Policy. In: Association of State Dam Safety Officials Conference Proceedings, at the 2010 Dam Safety Conference (2010)
11. Karg, D., Casal, J.: Ossim: Open source security information management. Tech. report, OSSIM (2008)
12. AlienVault®, <http://alienvault.com/>
13. AlienVault OSSIM Available Plugins, <http://alienvault.com/community/plugins>

# A Case Study on State-Based Robustness Testing of an Operating System for the Avionic Domain

Domenico Cotroneo, Domenico Di Leo, Roberto Natella, and Roberto Pietrantuono

Dipartimento di Informatica e Sistemistica, Università degli Studi di Napoli Federico II,  
Via Claudio 21, 80125, Naples, Italy  
{cotroneo, domenico.dileo, roberto.natella,  
roberto.pietrantuono}@unina.it

**Abstract.** This paper investigates the impact of state on robustness testing, by enhancing the traditional approach with the inclusion of the OS state in test cases definition. We evaluate the relevance of OS state and the effects of the proposed strategy through an experimental campaign on the file system of a Linux-based OS, to be adopted by Finmeccanica for safety-critical systems in the avionic domain. Results show that the OS state plays an important role in testing those corner cases not covered by traditional robustness testing.

**Keywords:** Robustness Testing, Operating Systems, Safety-Critical Systems, DO-178B, FIN.X-RTOS.

## 1 Introduction

The importance of high robustness in safety-critical systems is well recognized [1][2][3]. The Operating System (OS) is the foundation of any software system, and an OS failure may affect the system as a whole; thus, assessing its robustness is one of the most important tasks to perform during verification of critical software systems. Robustness testing techniques are conceived to assess the system's ability to not fail in presence of invalid inputs and unforeseen conditions.

Due to the kind of bugs for which robustness testing is conceived, and to the complexity and size of OS code, performing an effective robustness testing campaign is challenging. Robustness bugs are characterized by rare and subtle activation conditions, which are hard to find during functional testing [1][2][3]. Unfortunately, in OSs there are so many factors potentially involved in bug activation (such as I/O events and task scheduling), that it is difficult to include all of them when generating robustness test cases. As a result, many OS defects found in the field are related to boundary conditions and error handling, as shown in [3]. This difficulty is further exacerbated by the OS architecture, whose subsystems are tightly coupled, making it hard to isolate the reproduction of rare conditions for each of them. Considering these issues, it is clear that achieving high test coverage in OSs becomes a really tricky task.

In the literature, much effort has been devoted to assess robustness of OSs through the injection of invalid inputs via APIs (i.e., system calls/driver interfaces) with the goal of assessing their robustness. From results of these studies, an important

emerging aspect is that, to improve the effectiveness of robustness testing, test cases should consider one more variable, other than exceptional inputs; that is, the current *state* of the OS. Indeed, the OS state can significantly affect its execution, as well as the test case outcomes. Executing a given robustness test case in different states increases the probability to explore those parts of the code most rarely reached, i.e., it increases the final coverage: states representing “unusual” conditions combined with exceptional inputs can produce very rare execution patterns.

Starting from these studies, this paper investigates the impact of OS state, or part thereof, on robustness testing. We introduce a robustness testing strategy that accounts for the state of the *file system* (in terms of resource usage, concurrent operations, and other aspects). First, a model of the file system is presented that considers both entities a file system is composed of, and resources it uses and that contribute to determine its state. Then, the impact of the state on the achieved coverage is assessed through experiments performed on an industrial case study. Finmeccanica is in the process of developing a certifiable Linux-based OS, namely *FIN.X-RTOS*, compliant to the recommendations of the DO-178B standard [4], that is the reference standard in the avionic domain. In this process, evidences should be provided that the OS underwent a thorough robustness assessment campaign in terms of coverage. Results show that testing the OS by accounting for its state improves the final coverage, and hence the confidence in OS robustness, allowing to reach those corner cases not covered by traditional robustness testing.

After a related work section, the approach is described in Section 3. Section 4 shows obtained results in terms of coverage and via examples of achieved corner cases in the OS code; Section 5 concludes the paper.

## 2 Related Work

Past work approached robustness testing of operating systems from different points of view; they differ with respect to the OS interface under test (system calls or device driver interface), the assumed fault model, and the failure modes that were analyzed.

BALLISTA [1][2] was the first approach for evaluating and benchmarking the robustness of commercial OSs with respect to the POSIX system call interface [5]. BALLISTA adopts a *data-type based* fault model, that is, it defines a subset of invalid values for every data type encompassed by the POSIX standard. Examples of invalid inputs for three data types are provided in Table 1. Test cases are generated by all the combinations of invalid values of the system call’s data types: a test case consists of a small program that invokes the target system call using a combination of input values.

**Table 1.** Examples of invalid input values for the three data types of the `write(int filedes, const void *buffer, size_t nbytes)` system call

File descriptor (filedes)	Memory buffer (buffer)	Size (nbytes)
FD_CLOSED	BUF_SMALL_1	SIZE_1
FD_OPEN_READ	BUF_MED_PAGESIZE	SIZE_16
FD_OPEN_WRITE	BUF_LARGE_512MB	SIZE_PAGE
FD_DELETED	BUF_XLARGE_1GB	SIZE_PAGEx16
FD_NOEXIST	BUF_HUGE_2GB	SIZE_PAGEx16plus1

Test outcomes are classified by severity according to the *CRASH* scale: a Catastrophic failure occurs when the failure affects more than one task or the OS itself; Restart or Abort failures occur when the task launched by BALLISTA is killed by the OS or stalled; Silent or Hindering failures occur when the system call does not return an error code, or returns a wrong error code. BALLISTA found several invalid inputs not gracefully handled (Restarts and Aborts), and some Catastrophic failures related to illegal pointer values, numeric overflows, and end-of-file overruns [1].

OS robustness testing evolved in *dependability benchmarks* in the framework of the DBench European project [6][7]. A dependability benchmark has been proposed to assess OS robustness in terms of OS failures, reaction time (i.e., mean time to respond to a system call in presence of faults) and restart time (i.e., mean time to restart the OS after a test). Valid inputs are intercepted and replaced with invalid ones, by using a data-type based fault model, as well as by *fuzzing* (i.e., random values) and *bit-flips* (i.e., a correct input is corrupted by inverting one bit). In dependability benchmarking, the *workload* has a key role: it is used for exercising the system in order to assess its reaction. To obtain realistic measures, the workload should be representative of the expected usage profile (e.g., database or mail server [6][7]). In [8], a stress test campaign on the Linux kernel assessed the influence of the workload on kernel performance and memory consumption over long time periods. In this work, we further investigate the influence of the external environment, and propose a state model for generating tests that includes the OS workload.

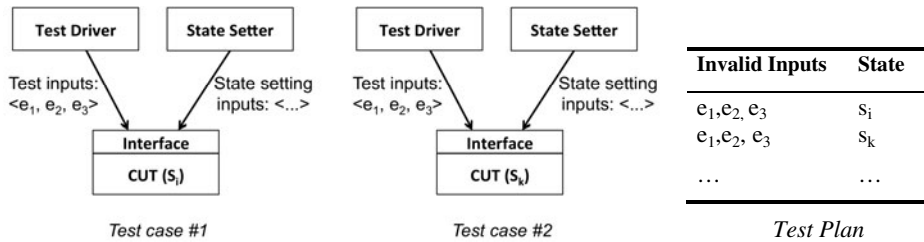
Robustness testing has been adopted for assessing OSs with respect to its interfaces to device drivers, since drivers are usually provided by third party developers and are buggier than other OS components [9]. The fault models mentioned above were adopted also in this case [10], and have been compared in terms of their ability to expose robustness bugs [11]. In [12], a fault injection approach is proposed that mutates the device driver code (by artificially inserting bugs) instead of injecting invalid values at the OS interface. These studies found that OSs are more prone to failures in case of device driver faults than application faults, since developers tend to omit checks in the device driver interface to improve performance, and because they trust device drivers more than applications. Other works assessed the robustness of OSs with respect to hardware faults (e.g., CPU or disk faults), by corrupting code and data [13][14][15][16]. Similarly to system call testing, these approaches either rely on a representative workload for exercising the system, or neglect the system state.

The influence of OS state gained attention in recent work on testing device drivers [17][18]. In [17], the concept of *call blocks* is introduced to model repeating subsequences of OS function calls made by device drivers, since they issue recurring sequences of function calls (e.g., when reading a large amount of data from a device): therefore, robustness testing is more efficient when it is focused on call blocks instead of injecting invalid inputs at random time. Sarbu et al. [18] proposed a state model for device driver testing, using a vector of boolean variables. Each variable represents an operation supported by the device driver: at a given time  $t$ , the  $i^{\text{th}}$  variable is true if the driver is performing the  $i^{\text{th}}$  operation. Case studies on Microsoft Windows OSs found that the test space can be reduced using the state model. Prabhakaran et al. [15] proposed an approach for testing journaling file systems, which injects disk faults at specific states of file system transactions. These studies showed that the OS state has an important role in testing such complex systems; however, they model a specific

part of OS state (e.g., device drivers or journaling) and do not consider the overall state of the OS components, such as the file system and process scheduling.

### 3 Testing Approach

Since OS components can be very complex and their state has a significant influence on the OS correct behavior, it is necessary to take the states of the Component Under test (CUT) into account, and assess its robustness as the state changes. According to this view, a hypothetical test plan is expressed through two dimensions: the exceptional inputs and the states. Inputs are selected as usual (e.g., boundary values) while the state varies in  $S = \{s_1, s_2 \dots s_n\}$ . In order to apply this strategy, we need to test the CUT with both a *test driver* and a *state setter*. The former injects invalid inputs to its interface, whereas the latter is responsible for producing the state transition or keeping the component in a given state  $s_k$  (see Figure 1).



**Fig. 1.** Robustness testing conducted with the CUT in two different states  $s_i$  and  $s_k$

In complex components the state representation (i.e., the state model) plays a key role. It can be considered at several levels of abstraction, hence determining the number of potential states the *state setter* should cope with. This aspect is relevant for our approach, since it can affect the efficiency and the feasibility of robustness testing. Thus the state model should satisfy these requirements: *i)* it should be easy to set and control by the tester, *ii)* it should represent the state at a level of abstraction high enough to keep the number of test cases reasonably small and *iii)* it should include those configurations that are the most influential on the component behavior. Thus, with this regard, the model that we define expresses the state of an OS component without detailing its internals, since they are not always easy to understand and to manage, and would inflate the number of states.

#### 3.1 Modeling the File System

In this work, we experiment the described strategy by applying it to the File System (FS) component. We choose the FS because it is a critical and bug-prone component [8][19] (its failure can corrupt persistent data or lead to unrecoverable conditions). Furthermore, the behavior of the FS is influenced by its internal state and the other components with which it interacts (e.g., virtual memory manger, scheduler). Following the previous requirements, we conceived a model for the FS (Figure 2).

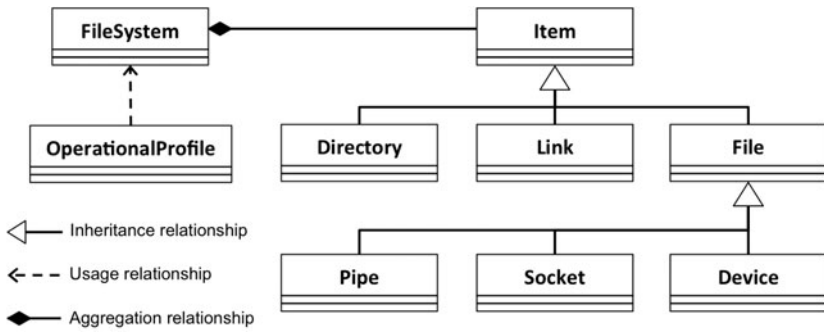


Fig. 2. File System model

Moreover, the model is easily adoptable across different FS<sup>1</sup> implementations; as a consequence, the proposed model does not take specific “internal design” of a FS into account (e.g., *inode* that are adopted in some UNIX file system, but not in others).

The model is a UML representation of the FS, with three main classes: *Item*, *FileSystem* and *OperationalProfile*. *FileSystem* represents the contents of data on the disk as a whole. It includes the state attributes that are not specific of a file. The class attributes are reported in Table 2.

Table 2. *FileSystem* attributes

Attributes	Description	Type
Partition Type	Typology of the partition	Primary, Logical
Partition size	Size of the partition on which is installed the FS	Byte
Partition allocated	The Current size of the allocated partition	Byte
Max file size	The maximum dimension of a file on the FS	Byte
Block size	The dimension of a block	Byte
FS implementation	The type of file system	NTFS, ext2, ext3
# of files allocated	The number of files in the FS	Integer
# of directories	The number of directories in the FS	Integer
FS layout	The tree that represents the FS	Balanced, Unbalanced
# of items allocated	The current number of items allocated in the FS	Integer

The choice of attribute values defines the test cases. Attributes like *Partition Allocated* can assume values from a minimum (e.g., 1MB) to the maximum allowable (e.g., 2TB). Therefore, the number of test cases, just for one parameter, grows rapidly. However, test cases in which the values of *Partition Allocated* varies with very small increments (e.g., from 1MB to 2MB) can be of little interest (e.g., 1MB or 2MB both are values for a small partition). Thus, it is necessary to define criteria to keep the number of test cases reasonably low and cover a reasonable

<sup>1</sup> In this work, the term “File System” refers to the OS component for managing files. The term “filesystem” refers to the contents on the storage, e.g., the structure of tree.

set of test scenarios. Hereafter, we illustrate potential choices for those attributes that the tester can set except for the attributes assigned by OS (e.g., `Max file size`).

The attributes `Block size` and `Partition size` are typically set when the file system is formatted for the first time. In a hypothetical test campaign, these values could assume minimum, maximum and intermediate values. The attribute `Partition allocated` can be expressed as a percentage of `Partition size`, therefore the tester can set scenarios in which the file system is totally full, partially full or empty.

The attribute `FS layout` deals with the tree representing the directory hierarchy on the FS. In particular, it can assume the values: *balanced*, i.e., trees in which the number of sub-directories is almost the same on each directory, and *unbalanced*, i.e., trees in which the number of sub-directories significantly differs. In order to generate balanced and unbalanced trees, we introduce  $P(\{d_{k+1}d_j\})$ , i.e., the probability that a new directory,  $d_{k+1}$ , is a child of a directory,  $d_j$ , already present in the tree. This probability allows, to some extent, to control the structure of the hierarchy, once `Number of Directory allocated` is fixed. For  $P(\{d_{k+1}d_j\})$ , we provide the following formulas for generating balanced and unbalanced trees, although other choices are possible (e.g., to use a well-known statistical distribution):

$$P_{unbalanced}(\{d_{k+1}d_j\}) = \frac{depth(d_j)}{\sum_1^k depth(d_i)} \quad (1)$$

$$P_{balanced}(\{d_{k+1}d_j\}) = \frac{1}{depth(d_j)} \frac{1}{\sum_1^k 1/depth(d_i)} \quad (2)$$

$$ParentDirectory = d : \max\{P(\{d_{k+1}d_1\}) \dots P(\{d_{k+1}d_k\})\} \quad (3)$$

where  $k$  is the number of current directories in the tree, and  $N$  the number of directories to be created;  $k$  is increased until  $k=N$ . In (1), new directories are more likely to form an unbalanced tree, since the higher the depth of a node is, the higher the probability to have children. In (2), new directories are more likely to group at the same depth. The parent directory (3) is the one with the highest value of  $P(\{d_{k+1}d_j\})$ .

As for the `FileSystem` class, it is possible to conceive several criteria for assigning values to the attributes. For instance, the attribute `Name` can assume alphabetical and numerical characters with equal probability or the length should not overpass a given value. The attributes `Permission` and `Owner` can be assigned in such a way that a given percentage of files are executable by the owner only, another percentage is readable by all users and so on. The attribute `Size` can be fixed for all files, generated according to a statistical distribution.

The `Item` class represents the entity which a `FileSystem` is made of. For this class, we define typical attributes that are available in every OS. Such attributes are: `name` of the item, `permission` (e.g., readable, writeable, executable), `owner` (root, nobody, user) and `size`. The classes that inherit from `Item` represent the different types of file in a UNIX file system. Files are randomly generated to populate the

directory tree mentioned above; the location and type of file can be determined according to statistical distributions.

The FS, like other OS subcomponents, uses resources such as cache, locks and buffers. We refer to these resources as *auxiliary resources*, that is, resources that serve for managing an Item of a FS. For instance, if a thread performs I/O operations it is likely to stimulate auxiliary resources: indeed, buffers are instantiated; locks to control the access to them are used, and so forth. These resources are part of the internal state of the FS, although they are not included in our model, since (i) they cannot be easily controlled by the tester, and (ii) they are dependent on the FS internals. Moreover, most of these resources are instantiated at run-time, and they are not part of the filesystem on the disk. The presence of these resources, however, cannot be neglected because they may influence the state of the FS and potentially change test outcomes. Therefore, in order to include both the behavior of the auxiliary resources in our model and the manner in which the FS is exercised, we introduce the `OperationalProfile` class. It expresses *the degree of usage* of the auxiliary resources and more generally, the way the FS is stimulated. This class does not directly model the auxiliary resource, but it allows to know the way in which the FS is invoked while performing a test. Thus the tester, *indirectly*, is aware of the mechanisms that are stimulated, e.g., if there are threads invoking I/O operations it is likely that caching and mutex mechanisms are invoked. The `OperationalProfile` attributes are reported in Table 3.

**Table 3.** `OperationalProfile` attributes

Attributes	Description	Type
Number of tasks invoking FS ops.	Number of tasks that invokes I/O operations (like read, write, open).	Integer
Average number of ops/s	Average number of operations made by a task	Integer
Ratio of read/write ops.	Ratio of read/write operations made by a task	Float

The `OperationalProfile` attributes are related to the performance of the File System and the hardware, which can limit the rate of FS operations that can be served by the system within a reasonable latency. Therefore, the selection of these attributes should be preceded by a capacity test aiming at assessing the maximum operation rate allowed by the system. A capacity test consists in gradually increasing the operations rate, given a fixed number of concurrent tasks (e.g., 2, 4 or 16), until the I/O bandwidth is saturated, i.e., the amount of transferred data per second reaches its peak [14]. After that the I/O bandwidth is known, the tests can select a discrete set of usage levels (e.g., 10% and 90% of I/O bandwidth) and the ratio between read and write operations (e.g., 2 read operations per 1 write operation).

## 4 Experimentation

In this section, we present an experiment aimed at analyzing the effects of the state on robustness testing, by comparing the proposed approach with a “stateless” approach



and with stress testing. The proposed approach has been applied on the FIN.X Real-Time Operating System (RTOS) developed by Finmeccanica. It is a Linux-based OS aimed at industrial applications in the avionic domain. The original Linux kernel has been enhanced by providing hard real-time and scalability on multi-core architectures and removing unessential parts. FIN.X-RTOS is accompanied with documentation and evidences recommended by the DO-178B safety standard [4]. At time of writing the requirements of the standards at level D have been fulfilled (software functions that may cause "a minor failure condition"), and FIN.X-RTOS is currently on the assessment process for the more stringent requirements of level C (software functions that may cause "a major failure condition"), which encompass robustness testing and full statement coverage.

#### 4.1 Experimental Setup

The proposed approach has been applied to the ext3 file system available in FIN.X-RTOS. We selected a set of system calls to test, described in Table 4. The system calls are commonly used by applications and exercise different parts of the FS code.

**Table 4.** System calls tested

System Call	Description
access	check user's permissions for a file
dup2	duplicate a file descriptor
lseek	reposition read/write file offset
mkfifo	make a FIFO special file (a named pipe)
mmap	map files or devices into memory
open	open and possibly create a file or device
read	read from a file descriptor
unlink	delete a name and possibly the file it refers to
write	write to a file descriptor

To apply the proposed strategy, we selected, without loss of generality, two well-known tools for supporting testing execution, namely *Ballista* and *Filebench*<sup>2</sup>. With regard to Figure 1, *Ballista* plays the role of *test driver*, while *FileBench* is the *state setter*. The *Ballista* tool is currently distributed with the Linux Test Project tool suite. We ported the original version to FIN.X-RTOS. *FileBench* is a tool for FS benchmarking: the user can customize a workload by configuring I/O access patterns in terms of number of threads, access type and so on. In our test campaign, we choose a realistic scenario in which the partition of filesystem is partially full (75% of Partition size) and there are tasks invoking FS operations, e.g., read and write. Leveraging on the model introduced in section 3, we create a logical partition with a balanced tree and the number of directories is 10 each one populated with 100 small files. No other items have been considered. Table 5 summarizes the values that we selected for the FileSystem entity's attributes. Table 6 shows the values selected for the File entities; all the files, apart from Name, have the same values. Table 7 specifies the attributes of OperationalProfile, which are typical values for FS benchmarking [6][8].

<sup>2</sup> <http://www.ece.cmu.edu/~koopman/ballista/> - <http://www.fsl.cs.sunysb.edu/~vass/filebench/>

**Table 5.** `FileSystem` values

Attribute	Value
Partition type	Logical
Partition size	2GB
Partition allocated	1,5GB
Block size	4096
File system implementation	ext3
Number of files allocated	1000
Number of directories allocated	10
Number of items allocated	1010

**Table 6.** `File` values

Attribute	Value
Name	Numeric string with length equals to five
Permission	Readable, Writeable, Executable
Owner	Root
Size	1500Kb

**Table 7.** `OperationalProfile` values

Attributes	Values
Number of tasks invoking FS operations	16
Average number of operations per second	10
Ratio of read/write operations	1

Those instances of `File`, `FileSystem`, and `OperationalProfile` reproduce stressful conditions in which to test the FS. By stressing the FS with read and write operations on a full allocated partition, we aim at creating exceptional conditions: in fact, with this setting, it is more likely to experiment conditions in which disk blocks are not available, seek operations have to traverse several directories, and so on.

#### 4.1.1 Definition of Test Campaigns

We carry out three experimental campaigns:

1. **Stateless robustness testing.** *Ballista* injects faults to the selected system calls (Table 4). The faultload to apply to the parameters of the system call belongs to the default *Ballista* configuration. An example of such a faultload is represented in Table 1. This test campaign lasts 15 minutes.
2. **Stress testing.** *FileBench* invokes the system calls `read` and `write` on the files previously allocated for 1 hour. The operations produced by *FileBench* reflect the attributes of `OperationalProfile` (Table 7). *Ballista* is not executed.

3. **Stateful robustness testing.** *FileBench* and *Ballista* work at the same time. *Ballista* and *FileBench* use the same configuration (faultload and operations executed) of the previous campaigns. The entire test campaign lasts 1 hour.

The experimental duration for the first test campaign is the time that *Ballista* spends to execute all the test cases. The second campaign lasts the time necessary for *Ballista* to execute all the tests while *FileBench* is running. The time for the third test campaign is set to 1 hour in order to compare the results between the second and third campaign over the same duration time.

## 4.2 Results

We first analyze the outcomes of robustness tests, which are classified according to the CRASH scale (see Section 2). Table 8 provides the summary produced by *Ballista* in the default configuration (i.e., all potential test cases are generated). We did not observe any *Catastrophic* failure, and only a small number of *Restart* and *Abort* failures occurred. This result was expected, since the OS is a mature and well-tested system, and is consistent with past results on POSIX OSs [1], in which only a small number of corner cases led to *Catastrophic* failures (e.g., an OS crash). The relevance of *Restart* and *Abort* failures is a controversial subject, since OS developers tend to consider them as a “robust” behavior of the OS [1]. According to this point of view, we do not consider *Restarts* as severe failures: several OSs (e.g., QNX, Minix) intentionally deal with a misbehaving task by killing it in some specific cases (e.g., manipulation of an invalid memory address, or lack of privileges for performing an operation), in order to avoid further error propagation within the system. Similarly, *Abort* failures can represent an expected (and desirable) behavior of the OS, such as in the case of the `read()` and `write()` system calls that can bring a task in a “waiting for I/O” state. For these reasons, a “Restart” or “Abort” outcome cannot be considered as a “failure” without a detailed analysis of the expected behavior. It should be noted that stateful robustness testing differs from stateless robustness testing with respect to the number of *Restart* outcomes, mostly due to failed memory and disk allocations. Although we cannot conclude that these outcomes represent OS failures, this result points out that OS state can affect test outcomes and the assessment of OS robustness.

**Table 8.** Results of robustness tests

Function	# Tests	Stateless robustness testing		Stateful robustness testing	
		# Restart	# Abort	# Restart	# Abort
<code>access()</code>	3,986	0	4	1	4
<code>dup2()</code>	3,954	0	0	1	0
<code>lseek()</code>	3,977	0	0	0	0
<code>mkfifo()</code>	3,870	0	5	1	5
<code>mmap()</code>	4,003	0	0	0	0
<code>open()</code>	3,988	0	8	40	8
<code>read()</code>	3,924	0	253	1	253
<code>unlink()</code>	500	0	1	0	1
<code>write()</code>	3,989	0	68	4	68
Total	32,191	0	339	48	339

However, the stateful tests cover a scenario not considered by stateless tests, and therefore they represent an additional evidence of the robust behavior of the OS. As a result, we observed an increased coverage of kernel code after executing the stateful tests; this aspect is relevant since coverage is a measure of test confidence and a requirement for software in safety-critical systems (e.g., DO-178B at level C [4]).

We analyzed statement coverage of file system code, which is the target of our tests. The file system code is arranged in three directories: the code in the "fs/" directory is independent from the specific file system implementation (i.e., it is shared among several implementations such as ext3 and NTFS); the "ext3" directory provides the implementation of the ext3 file system; finally, the "jbd" directory provides a generic support for journaling file systems. Data about coverage was collected using GCOV. Table 9 compares the statement coverage with respect to the three considered scenarios. We observed differences in coverage between stateless (second column) and stateful robustness testing (fourth column), ranging between 0.49% and 15.11%. Part of the code is covered by the plain state setter (i.e., without using Ballista); the remaining part is covered due to interactions between Ballista and the OS state (some examples are provided in the following). In particular, stateful testing exercised those parts of the file system that interact with other subsystems (e.g., interactions between "fs/buffer.c" and the memory management subsystem, and between "fs/fs-writeback.c" and disk device drivers). The coverage improvement is more significant for the journal-related code (i.e., the JBD component in "fs/jbd"). This effect can be attributed to the interactions between file system transactions and the state of I/O queues. For instance, a transaction commit can be delayed due to concurrent I/O operations, therefore affecting the management of data buffers within the kernel and the file system image on the disk. Although the improvement is less significant for the implementation-independent code, the proposed approach has been useful for improving test coverage with no human effort. This aspect is relevant since

**Table 9.** Statement coverage

Source file	Stateless robustness testing	Stress testing	Stateful robustness testing
fs/binfmt_elf.c	319/850 (37.53%)	331/850 (38.94%)	332/850 (39.06%)
fs/buffer.c	529/1320 (40.08%)	553/1320 (41.89%)	565/1320 (42.80%)
fs/dcache.c	371/880 (42.16%)	341/880 (38.75%)	387/880 (43.98%)
fs/exec.c	479/807 (59.36%)	392/807 (48.57%)	486/807 (60.22%)
fs/fs-writeback.c	146/273 (53.48%)	169/273 (61.90%)	174/273 (63.74%)
fs/inode.c	252/527 (47.82%)	307/527 (58.25%)	316/527 (59.96%)
fs/namei.c	918/1392 (65.95%)	626/1392 (44.97%)	925/1392 (66.45%)
fs/select.c	237/402 (58.96%)	237/402 (58.96%)	239/402 (59.45%)
fs/ext3/balloc.c	384/556 (69.06%)	385/556 (69.24%)	398/556 (71.58%)
fs/ext3/dir.c	140/219 (63.93%)	143/219 (65.30%)	144/219 (65.75%)
fs/ext3/ialloc.c	181/337 (53.71%)	186/337 (55.19%)	189/337 (56.08%)
fs/ext3/inode.c	719/1204 (59.72%)	729/1204 (60.55%)	737/1204 (61.21%)
fs/ext3/namei.c	607/1088 (55.79%)	654/1088 (60.11%)	781/1088 (71.78%)
fs/jbd/checkpoint.c	102/263 (38.78%)	141/263 (53.61%)	142/263 (53.99%)
fs/jbd/commit.c	300/362 (82.87%)	302/362 (83.43%)	318/362 (87.85%)
fs/jbd/revoke.c	108/228 (47.37%)	105/228 (46.05%)	116/228 (50.87%)
fs/jbd/transaction.c	489/697 (70.16%)	500/697 (71.74%)	545/697 (78.19%)

FIN.X-RTOS is mostly composed by third-party code re-used from the Linux kernel; covering this code can be very costly, due to the lack of knowledge of kernel internals and the inherent complexity of OS code (e.g., heuristics for memory management).

In order to better understand the interactions between OS state and test cases, we analyzed more in depth part of the kernel code only covered by stateful robustness testing. Figure 3 shows an example of corner case in the kernel code not covered in stateless testing (the code is highlighted in bold font; part of the code was omitted; we kept some comments from developers). The `real_lookup()` routine is invoked when file metadata are not in the page cache, and the FS needs to access to the disk. It blocks the current task on a semaphore (using the `mutex_lock()` primitive) until a given directory can be accessed in mutual exclusion. It then checks if metadata have been added to the cache during this wait period. Usually, metadata are not found, and the routine performs an access to the disk. In stateful testing, a different behavior was observed, since the cache has been re-populated during the wait period (developers refer to this situation as "nasty case"), and additional operations are executed (e.g., to check that metadata are not expired due to a timeout in distributed file systems). This code was only executed in stateful testing due to interactions with the cache that occur when concurrent I/O operations are taking place.

```
static struct dentry * real_lookup(struct dentry * parent,
    struct qstr * name, struct nameidata *nd) {
    /* --- OMISSIS (declarations) --- */
    mutex_lock(&dir->i_mutex);
    result = d_lookup(parent, name);
    if (!result) {
        /* --- OMISSIS (performs lookup) --- */
        mutex_unlock(&dir->i_mutex);
        return result;
    }
    /* Uhuh! Nasty case: the cache was re-populated while
    we waited on the semaphore. Need to revalidate.*/
    mutex_unlock(&dir->i_mutex);
    if (result->d_op && result->d_op->d_revalidate) {
        result = do_revalidate(result, nd);
        if (!result)
            result = ERR_PTR(-ENOENT);
    }
    return result;
}
```

**Fig. 3.** Example of kernel code covered due to interactions between the file system and caching (from `real_lookup()`, `fs/namei.c:478`)

Another example is provided in Figure 4, which is related to concurrency of kernel code. The `ll_rw_block()` routine performs several low-level accesses to the disk, and each access is controlled by a “buffer head” data structure. During the inspection of the list of buffer heads, one of them could have been locked by another concurrent task; this condition is detected by the `test_set_buffer_locked()` primitive, which may fail to lock the buffer head in some cases. Stateful testing covered this rare scenario, and it is worth being tested to verify that pending I/O is correctly managed.

```

void ll_rw_block(int rw, int nr, struct buffer_head *bhs[]) {
    int i;
    for (i = 0; i < nr; i++) {
        struct buffer_head *bh = bhs[i];
        if (rw == SWRITE)
            lock_buffer(bh);
        else if (test_set_buffer_locked(bh))
            continue;
        /* --- OMISSIS (performs I/O op.) --- */
    }
}

```

**Fig. 4.** Example of kernel code covered due to concurrent I/O requests (from `ll_rw_block()`, `fs/buffer.c:2941`)

Finally, we analyzed an example of kernel code interacting with memory management, which is provided in Figure 5. The `try_to_free_buffers()` routine is invoked by the file system when the cache for file system data (the "page cache") gets large and pages need to be freed for incoming data. It may occur that a file system transaction involves I/O buffers allocated over several pages, and these pages cannot be de-allocated until the transaction commits. Pages are then marked with "mapping == NULL" in order to be reclaimed later (the `drop_buffers()` routine checks that I/O buffers in the page are not being used). As suggested by the comment in the code, this condition is unlikely to occur; the code has been executed in stateful testing since memory management has been put under stress.

```

int try_to_free_buffers(struct page *page) {
    /* --- OMISSIS (declarations) --- */
    BUG_ON(!PageLocked(page));
    if (PageWriteback(page))
        return 0;
    if (mapping == NULL) { /* can this still happen? */
        ret = drop_buffers(page, &buffers_to_free);
        goto out;
    }
    /* --- OMISSIS (page writeback and deallocation) --- */
}

```

**Fig. 5.** Example of kernel code covered due interactions between the file system and memory management (from `try_to_free_buffers()`, `fs/buffer.c:3057`)

## 5 Conclusion and Future Work

This paper investigated the impact of OS state on robustness testing through an experiment on the File System of a Linux-based OS for critical applications. In order to include the OS state in the robustness test plan, we introduced a model of the File System by including a set of factors (such as file tree layout and concurrent I/O operations) that are most influential on the File System behavior, and that can be controlled by the tester. We performed an experiment using the proposed model, which highlighted the influence OS state on the test outcomes and on statement coverage. In particular, robustness tests were able to reach corner cases with complex

interactions with other subsystems (such as scheduling, caching and memory management), which are not covered by traditional robustness testing. In turn, this approach comes in handy to achieve an increased confidence in OS robustness with low human effort, since both robustness test cases and OS states can be automatically generated once programmed by the tester.

Future work encompasses an experimental campaign with more robustness tests and OS states, in order to assess the full potential of robustness testing. Moreover, we plan to analyze test planning strategies in order to achieve the best trade-off between time and the code coverage or the explored states. Another direction is to extend the approach to other subsystems. For instance, a model similar to the FS could be introduced for the virtual memory manager, by including the amount and type of memory areas allocated by processes, physical free memory, swap usage and so on.

**Acknowledgements.** We would like to thank Mariana Esposito for her valuable contributions, and Francesco Rogo and MBDA Systems for their technical support with FIN.X-RTOS. This work has been funded by the FP7 European project CRITICAL-STEP (<http://www.critical-step.eu>) IAPP no. 230672, and by the Italian research project “Iniziativa Software”, which involves the Finmeccanica company and Italian universities (<http://www.iniziativasoftware.it>).

## References

1. Koopman, P., DeVale, J.: The exception handling effectiveness of POSIX operating systems. *IEEE Trans. on Software Engineering* 26(9) (2002)
2. Koopman, P., Sung, J., Dingman, C., Siewiorek, D., Marz, T.: Comparing operating systems using robustness benchmarks. In: *SRDS* (1997)
3. Sullivan, M., Chillarege, R.: Software Defects and their Impact on System Availability-A Study of Field Failures in Operating Systems. In: *FTCS* (1991)
4. RTCA Inc., Software considerations in airborne systems and equipment certification, RTCA DO-178B, EUROCAEED-12B (1992)
5. IEEE Standard for Information Technology-Portable Operating System Interface (POSIX). IEEE Std 1003.1b-1993, IEEE CS (1994)
6. Kanoun, K., Cruzet, Y., Kalakech, A., Rugina, A.-E., Rumeau, P.: Benchmarking the Dependability of Windows and Linux using PostMarkTM Workloads. In: *ISSRE* (2005)
7. Kalakech, A., Kanoun, K., Cruzet, Y., Arlat, J.: Benchmarking The Dependability of Windows NT4, 2000 and XP. In: *DSN* (2004)
8. Cotroneo, D., Natella, R., Pietrantuono, R., Russo, S.: Software Aging Analysis of the Linux Operating System. In: *ISSRE* (2010)
9. Chou, A., Yang, J., Chelf, B., Hallem, S., Engler, D.: An empirical study of operating systems errors. In: *SOSP* (2001)
10. Albinet, A., Arlat, J., Fabre, J.C.: Characterization of the Impact of Faulty Drivers on the Robustness of the Linux Kernel. In: *DSN* (2004)
11. Johansson, A., Suri, N., Murphy, B.: On the selection of error model(s) for OS robustness evaluation. In: *DSN* (2007)
12. Duraes, J., Madeira, H.: Multidimensional characterization of the impact of faulty drivers on the operating systems behavior. *IEICE Trans. on Information and Systems* 86(12) (2003)

13. Gu, W., Kalbarczyk, Z., Iyer, R.K., Yang, Z.: Characterization of Linux kernel behavior under errors. In: DSN (2003)
14. Skarin, D., Barbosa, R., Karlsson, J.: GOOFI-2: A tool for experimental dependability assessment. In: DSN (2010)
15. Bairavasundaram, L.N., Rungta, M., Agrawa, N., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H., Swift, M.M.: Analyzing the effects of disk-pointer corruption. In: DSN (2008)
16. Dreges, R.J., Nanya, T.: Analysis of Inter-Module Error Propagation Paths in Monolithic Operating System Kernels. In: EDCC (2010)
17. Johansson, A., Suri, N., Murphy, B.: On the impact of injection triggers for OS robustness evaluation. In: ISSRE (2007)
18. Sarbu, C., Johansson, A., Suri, N., Nagappan, N.: Profiling the operational behavior of OS device drivers. *Empirical Soft. Eng.* 15(4) (2009)
19. Prabhakaran, V., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Model-based failure analysis of journaling file systems. In: DSN (2005)



# Formal Methods for the Certification of Autonomous Unmanned Aircraft Systems

Matt Webster<sup>1,\*</sup>, Michael Fisher<sup>2</sup>, Neil Cameron<sup>1</sup>, and Mike Jump<sup>1,3</sup>

<sup>1</sup> Virtual Engineering Centre, Daresbury Laboratory, Warrington, UK  
Tel./Fax: +44 (0) 1925 864850  
matt@liv.ac.uk

<sup>2</sup> Department of Computer Science, University of Liverpool, UK

<sup>3</sup> School of Engineering, University of Liverpool, UK

**Abstract.** In this paper we assess the feasibility of using formal methods, and model checking in particular, for the certification of Unmanned Aircraft Systems (UAS) within civil airspace. We begin by modelling a basic UAS control system in PROMELA, and verify it against a selected subset of the CAA's Rules of the Air using the SPIN model checker. Next we build a more advanced UAS control system using the autonomous agent language Gwendolen, and verify it against the small subset of the Rules of the Air using the agent model checker AJPF. We introduce more advanced autonomy into the UAS agent and show that this too can be verified. Finally we compare and contrast the various approaches, discuss the paths towards full certification, and present directions for future research.

**Keywords:** Model Checking, Formal Methods, Unmanned Aircraft System, Autonomous Systems, Certification.

## 1 Introduction

An Unmanned Aircraft System (UAS, plural UAS) is a group of elements necessary to enable the autonomous flight of at least one Unmanned Air Vehicle (UAV) [8]. For example, a particular UAS may comprise a UAV, a communication link to a ground-based pilot station and launch-and-recovery systems for the UAV. UAS are now routinely used in military applications, their key advantages coming from their ability to be used in the so-called “dull, dangerous and dirty” missions, e.g., long duration/persistence flights and flights into hostile or hazardous areas (such as clouds of radioactive material) [20]. There is a growing acceptance, however, that the coming decades will see the integration of UAS into civil airspace for a variety of similar applications: security surveillance, motorway patrols, law enforcement support, etc. [21][15]. However, in order for this integration to take place in a meaningful way, UAS must be capable of routinely flying through “non-segregated” airspace. Today, for most useful civil applications, UAS can fly in UK civil airspace but in what is known as segregated airspace, that is, airspace which is for the exclusive use of the specific user. For routine UAS operations, this will not be an acceptable solution if the demand for UAS usage increases as is envisaged. The UK projects ASTRAEA and ASTRAEA II and the FAA's Unmanned

---

\* Corresponding author.

Aircraft Program Office (UAPO) are tasked with meeting this regulatory challenge, but a summary of the issues is considered pertinent. Guidance on the UK policy for operating UAS is given in [8]. The overarching principle is that, “UAS operating in the UK must meet at least the same safety and operational standards as manned aircraft.” A UAS manufacturer must therefore provide evidence to the relevant regulatory authority that this is indeed the case.

For manned aircraft, there is a well understood route for manufacturers to demonstrate that their vehicle and its component systems meet the relevant safety standards (see, for example, [12]). However, the manufacturer does not have to concern itself with certification of the pilot: it is assumed that a suitably qualified crew will operate the aircraft. For a UAS, however, the human operator may be out of the control loop and therefore the manufacturer must demonstrate that any *autonomous* capabilities of the aircraft, in lieu of an on-board human pilot, do not compromise the safety of the aircraft or other airspace users. The acceptable means to achieve this end, i.e., regulatory requirements, have yet to be formalised even by the regulators.

In this paper, we investigate the potential usefulness of model checking in providing formal evidence for the certification of UAS. The work described here develops a new approach and describes a study examining the feasibility of using formal methods tools to prove compliance of an autonomous UAS control system with respect to a small subset of the “Rules of the Air” [7]. Demonstrating that the decisions made by the autonomous UAS are consistent with those that would be made by a human pilot (in accordance with the Rules of the Air), could provide powerful evidence to a regulator that the UAS would not compromise the safety of other airspace users. Thus, the work described herein may be a first step in answering the question as to whether or not formal verification tools have the potential to contribute to this overall ambition.

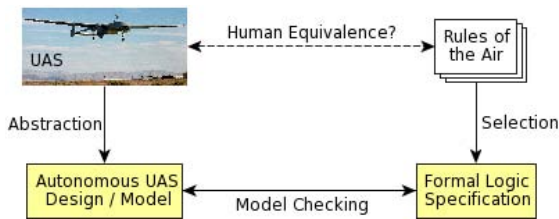
This is but one step towards the certification of autonomous UAS in non-segregated UK airspace, yet it allows us to show how a route to full certification might be relieved of some of the burden of analysis/testing required at present. This can save time and increase reliability, but might come at the cost of an increased level of expertise required of the analysts involved in the certification process. In particular, we focus on using formal methods to verify the high-level “decision-making” aspects of autonomous UAS control which may eventually complement or replace human decision-making for UAS. The model checking approaches we describe could help to establish the robustness of a given decision-making system, and when combined with existing approaches to aircraft software engineering, could provide a route to certification of autonomous UAS.

## 1.1 Approach

Since the route to airframe and automatic flight control system certification is already established, the main, and possibly the only, difference between a UAS and a human-piloted aircraft is the core autonomous control system, plus all of the systems that are directly associated with it, e.g., power supplies, etc. Thus, a vital part of certification would be to show that this core autonomous control (in the form of an “intelligent” agent) would make the same decisions as a human pilot/controller *should* make (this is, after all, one of the piloting skills that a human pilot must obtain to be awarded a licence). In general, analysing human behaviour is, of course, very difficult. However,

in the specific case of aircraft certification, pilots should abide by the Rules of the Air. Thus, our approach here is to verify that all of the choices that the agent makes conform to these Rules. It should be recognised that demonstrating that an autonomous agent’s decisions will conform to the Rules of the Air is not the same as providing sufficient evidence for certification. However, demonstrating that this is the case will provide one piece of evidence that will support any application for certification of a system.

To show how this might be done, we chose a small subset of the Rules of the Air and encoded these in a formal logic. (“The Rules of the Air Regulations 2007,” is large: around 15,000 words plus accompanying images [7].) We modelled a UAS control system as an *executable agent model* (initially using PROMELA [13], but later in a higher-level agent language [1]), and applied model checking to verify that the UAS agent satisfied the selected subset of the Rules of the Air.



**Fig. 1.** An approach to certification via the Rules of the Air. (Image: SSgt. R. Ramon, USAF.)

Our approach is summarised in Fig. 1. Clearly, the closer the UAS design/model is to the actual UAS control system implementation and the closer the logical specification is to the actual *meaning* of the “Rules of the Air”, the more useful model checking will be in generating analytical evidence for certification. Ideally, the UAS model/design should be a description of *all* the decisions/choices the UAS can possibly make. For the purposes of this study, we assume that standard verification and validation (V&V) techniques for high integrity software have been used to ensure that the UAS control system does actually correspond to this design/model. Ideally, we would also like to capture *all* of the Rules of the Air in a precise, logical form. However, there are several problems with this. First, the Rules of the Air are neither precise nor unambiguous — thus it is very hard to formalise their *exact* meaning without making the formulation very large. Next, the number of rules is too large to tackle them all within this study. Finally, some of the rules implicitly use quite complex notions, such as “likelihood”, “knowledge”, “the other pilot’s intention”, “expectation”, and so on (see below for some examples). While extending our formalisation to such aspects will be tackled in the second half of this study, our initial step is to *select* a small number of rules that are clear, unambiguous, and relevant to UAS.

## 1.2 Paper Structure

In Section 2 we describe the software tools to be used for UAS agent verification and describe how the small subset of the Rules of the Air for verification was chosen. Then,

in Section 3 we model a basic UAS agent in PROMELA, and verify it against a small subset of the Rules of the Air using the SPIN model checker. The concept of an “agent” is a popular and widespread one, allowing us to capture the core aspects of autonomous systems making informed and rational decisions [27]. Indeed, such agents are typically at the heart of the hybrid control systems prevalent within UAS. We will say more about the “agent” concept later but, initially, we simply equate “agent” with “process”. Thus, we model the UAS’s choices/decisions as a single process in PROMELA, and use SPIN to show that the UAS agent satisfies the selected subset of the Rules of the Air.

In Section 4 we construct a UAS control system based on a rational agent model. This is written using the autonomous agent language Gwendolen [10], and we show that it can be verified against the same Rules of the Air using the agent model checker AJPF [211]. We introduce more advanced autonomous behaviour into the UAS agent, and verify that this acts in accordance with the subset of the Rules of the Air.

There are two main reasons for using a rational agent model. The first was to allow more “intelligence” in the UAS agent itself. This extended the agent’s choices to take into account not only the UAS’s situation but also the agent’s beliefs about the intentions of other UAS/aircraft. The second reason is to consider more than the literal meaning of the Rules of the Air. Specifically, we noticed that there is often an implicit assumption within these rules. For example, “in situation A do B” might have an implicit assumption that the pilot will assess whether doing B in this particular situation would be dangerous or not. Really such rules should be: “in situation A do B, unless the UAS believes that doing B will be likely to lead to some serious problem”. In piloting parlance, the agent needs to demonstrate *airmanship*. Thus, in Section 4 we show how we might “tease” out such aspects into formal specifications involving intentions/beliefs that could then be checked through our verification system.

Finally, in Section 5 we compare the different approaches to UAS agent modelling and verification, and we present directions for future research.

## 2 Model Checking

Model checking is a variety of formal verification in which a logical property is exhaustively evaluated against all executions of a system [9]. Typically, the logical property is expressed within a *temporal* logic. This allows us to refer to properties that occur now, in the next moment, and at selected moments in the future. As well as classical logic operators, temporal logic also provides operators such as ‘ $\square$ ’, meaning “at all future moments”. Thus, “ $\square(x \Rightarrow y)$ ” means that at all future moments within the execution, if  $x$  is true then  $y$  must be true. This is distinct from “ $x \Rightarrow \square y$ ” which means that, if  $x$  is true then  $y$  must be true at all future moments.

In the model checker we first utilise, called SPIN [13], the program to be checked is written in the PROMELA programming language. The SPIN model checker then exhaustively checks our required temporal formula against all possible executions of the program. If successful, this means that no matter how the program executes, the required property will still be true. However, if the model checker finds a specific execution that violates the required property, it identifies this to the user.

Although we begin by using the PROMELA language and SPIN for verification, we later use a more sophisticated language, Gwendolen [10], a high-level agent-based

programming language, to develop more advanced UAS control. We check the Gwendolen program against the same logical requirements, but as SPIN only checks PROMELA programs, we must use a different model checker called AJPF [211] to establish correctness of the Gwendolen program with respect to the logical properties.

## 2.1 Selecting Rules of the Air for Model Checking

We chose a small subset of just three Rules of the Air [7] which were relevant for a straightforward flight of a powered UAS vehicle (e.g., taxiing to take-off, navigation, sense-and-avoid, and landing). It was also desirable to choose rules which might potentially come into conflict, as this would present a greater challenge for engineering and verification of the UAS. We also had to leave out certain rules concerning specific heights and distances, as we did not intend to describe such detailed information within our UAS model. In addition we wanted to focus on two key scenarios for UAS engineering: (i) “sense-and-avoid”, where the UAS must detect objects that it may collide with and take evasive action; and (ii) partial autonomy, where the UAS proceeds autonomously but checks with a human for permission to perform certain actions. Both are essential abilities of autonomous UAS [21]. Thus, the rules chosen were as follows:

1. **Sense and Avoid:** “...when two aircraft are approaching head-on, or approximately so, in the air and there is danger of collision, each shall alter its course to the right.” (Section 2.4.10)
2. **Navigation in Aerodrome Airspace:** “[An aircraft in the vicinity of an aerodrome must] make all turns to the left unless [told otherwise].” (Section 2.4.12(1)(b))
3. **Air Traffic Control (ATC) Clearance:** “An aircraft shall not taxi on the apron or the manoeuvring area of an aerodrome without [permission].” (Section 2.7.40)

The first rule is relevant for the sense-and-avoid scenarios (see (i) above), and the third rule is relevant for partial autonomy (see (ii) above). The second rule is interesting because it may conflict with the first rule under certain circumstances, e.g., where an object is approaching head-on and the UAS has decided to make a turn. In this case, the UAS vehicle may turn left or right depending on which rule (1 or 2) it chooses to obey.

Simplification was necessary to encode the above “rules” so that they could be model checked. For instance, in the second rule, there are a number of factors which could “tell” the UAS vehicle to make a turn to the right, such as the pattern of traffic at an aerodrome, ground signals, or an air traffic controller. We chose to model all of these under the umbrella term “told otherwise”, and not to model these factors separately.

## 3 Reactive UAS Agents

Through consultations with researchers from the Autonomous Systems Research Group at BAE Systems (Warton, UK) we have modelled fragments of a typical UAS agent relevant to our selected scenario. Here, it is assumed that the UAS agent will be composed of a set of rules concerning the successful completion of the mission and the safe flight of the aircraft. Each rule has a condition which must be satisfied for that rule to be applied, and a consequence of applying that rule. For example, a rule might look like:

IF aircraft\_approaching\_head\_on THEN turn\_right

This would be the part of the agent designed to deal with the “Sense and Avoid” scenario described in Section 2.1. Clearly there would be many other rules in the agent to deal with other situations, such as running low on fuel, take off, landing, etc. The idea is that the complete set of rules would enable the flight of the UAS, so that the UAS would respond appropriately in every situation. Another such rule could be:

IF ATC\_clearance\_rcvd THEN set\_flight\_phase\_taxi; taxi\_to\_runway\_and\_wait

This rule would specify that when the UAS receives clearance from the ATC, it will set its flight phase to “taxi” and start taxiing to the runway where it will wait for take-off clearance. In general, this kind of agent is known as a *reactive agent*, as it reacts to situations without reasoning about them. (In later sections we will also consider a *practical reasoning*, or *rational*, agent for controlling a UAS.)

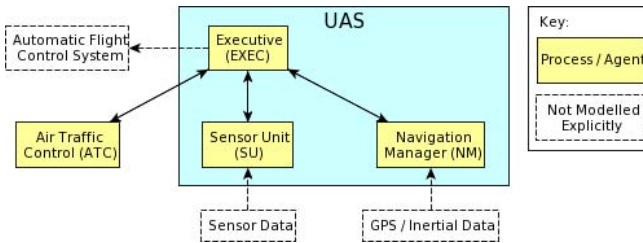


Fig. 2. UAS Models in PROMELA and Gwendolen. Arrows represent information flow.

### 3.1 Modelling a Reactive UAS Agent in PROMELA

A simple model of a partial UAS control system has been written using PROMELA, the process modelling language for the SPIN model checker [13]. The UAS is divided into a number of components: the Executive, the Sensor Unit (SU) and the Navigation Manager (NM). In Fig. 2, the role of the Executive is to direct the flight of the UAS based on information it receives about the environment from the SU and the NM. The NM is an independent autonomous software entity (i.e., an agent) on-board the UAS which detects when the UAS is off-course and needs to change its heading; it sends messages to the Executive to this effect. When the UAS’s heading is correct, the NM tells the Executive so that it can maintain course. The SU is another agent on-board the UAS whose job it is to look for potential collisions with other airborne objects. When it senses another aircraft, it alerts the Executive; the SU then notifies the Executive when the detected object is no longer a threat. Another essential part of the model is the ATC. The Executive communicates with the ATC in order to request clearance to taxi on the airfield. The ATC may either grant or deny such clearance. Thus, our simple reactive UAS models sense-and-avoid scenarios as well as navigation and ATC clearance.

In PROMELA, we model the Executive, the SU, the NM and the ATC as processes, which communicate using message-passing *channels* (see Fig. 2). For simplicity we specify the NM and the SU as non-deterministic processes which periodically (and arbitrarily) choose to create navigation and sensory alerts. The Executive process has a

variable, called `state`, which has different values to represent the different parts of the UAS’s mission: `WaitingAtRamp` (start of mission), `TaxiingToRunwayHoldPosition`, `TakingOff`, `EmergencyAvoid`, etc.

Each step in the process is modelled by a different value of the `state` variable. Once the UAS model becomes “airborne”, the Executive may receive messages from both the SU and the NM. If the Executive receives a message from the SU saying that there is an object approaching head-on, then it changes state to “Emergency Avoid” and alters the course of the UAS to the right (by updating a variable `direction`). When the SU tells the NM that the object approaching head-on has passed, the Executive will continue on the heading and in the state it was in before the alert, e.g., if it was changing heading and turning left then it will go back to this behaviour. At any point the Executive may receive a message from the NM advising it to alter its heading, maintain its current heading or, eventually, land.

Certain elements of a real-life UAS are not modelled here. We do not model the “real world” environment of the UAS explicitly; rather we use the SU to send sensory alerts on a non-deterministic basis. Likewise, the NM does not really navigate, as there is no “real world” in the model to navigate through, and so it sends navigation alerts on a non-deterministic basis. Also, we do not model the flight control systems of the UAS or any aspects of the vehicle itself, as without a “real world” model these are unnecessary. However, we make these simplifications without loss of accuracy in the verification process: our aim is verify the *behaviour* of the Executive, to ensure that it adheres to the “Rules of the Air” according to the information it possesses about the current situation, and so using the SPIN model checker we can ascertain whether the Executive behaves in the desired manner.

### 3.2 Model Checking the Rules of the Air in SPIN

As we have a system capturing selected behaviour within a UAS, together with elements of its environment (e.g., ATC), we can check its compliance with the Rules of the Air identified in Section 2.1 using the SPIN model checker. The temporal logic form of these three rules are as follows.

1. **Sense and Avoid:**  $\Box(\text{objectIsApproaching} \implies \{\text{direction} = \text{Right}\})$

2. **Navigation in Aerodrome Airspace:**

$$\Box \left[ \left( \begin{array}{l} \text{changeHeading} \wedge \neg \text{objectIsApproaching} \\ \wedge \text{nearAerodrome} \wedge \neg \text{toldOtherwise} \end{array} \right) \implies \neg \{\text{direction} = \text{Right}\} \right]$$

3. **ATC Clearance:**

$$\Box(\{\text{state} = \text{TaxiingToRunwayHoldPosition}\} \implies \text{haveATCTaxiClearance})$$

The UAS agent model was found to satisfy all three properties.

## 4 Rational UAS Agents

The reactive UAS agent model presented so far, written in PROMELA, is quite basic in terms of autonomy. The UAS follows a series of reflexive responses to environmental

changes, e.g., a message has come from ATC saying taxi clearance has been given, so update the UAS state to “Taxiing.” It may be desirable to encode more complex autonomous behaviours based on ideas from intelligent agent theory, such as the Beliefs–Desires–Intentions (BDI) framework for autonomous agents [22]. As suggested by the name, agents comprise *beliefs* (i.e., their information about the world), *desires* (i.e., their long term aims), and *intentions* (i.e., the things the agent is doing to try to achieve its desires). Such approaches offer a natural way of specifying, engineering and debugging high-level autonomous behaviour [27]. Another advantage is model checking autonomous behaviour: we can see the state of the agent’s beliefs, desires and intentions at the point a particular logical property is violated.

To model BDI agents we use a BDI agent language as PROMELA is not designed for this purpose. To use PROMELA in this way, beliefs, desires and intentions would have to be constructed from the native PROMELA constructs such as processes, variables, etc. In contrast, BDI agent languages have these features “built-in”. Therefore the software engineer is more able to focus on the behaviours of the autonomous system when using a BDI agent language than when using PROMELA. Likewise, the SPIN model checker used to verify PROMELA programs does not contain any operators concerning agents’ beliefs, desires or intentions, whereas agent model checkers let us specify different agents’ beliefs, desires and intentions within the property being checked.

Gwendolen [10] is a BDI agent programming language designed specifically for agent verification. Gwendolen agents consist of beliefs, goals, intentions and plans. (Goals are desires which are being actively pursued.) Each plan consists of a triggering event, a guard and a number of “deeds” which are executed if the plan is triggered and the guard is satisfied. A Gwendolen agent begins with sets of initial beliefs and goals, and a set of plans. The agent selects a subset of plans based on its beliefs about the current situation and its current goals, i.e., what it wants to achieve.

We have constructed a model of a UAS agent written in Gwendolen. Our UAS agent consists of 44 different plans, several of which are shown below. The UAS is similar in behaviour to the agents written in PROMELA: it taxis, holds, lines up and takes off, and once airborne it performs simple navigation and sense/avoid actions. Finally, it lands. The UAS agent believes initially that it is waiting at the ramp at the beginning of its mission, and that it has no forward direction. It has an initial goal (here, “!p” means a goal to perform some action) — to run the “startup procedure” — and a set of plans. For instance, the first plan says that if a belief that normal flight is underway is added (the trigger,  $+normalFlight$ ), the agent will delete the last message from the sensor unit ( $-su(S)$ ) and will undertake an action to send a message to the sensor unit requesting information ( $send(su, poll)$ ). The “{...}” in this case is a guard condition on the plan. Here ‘T’ always evaluates to “True”.

**Agent:** *exec*

**Initial Beliefs:** *waitingAtRamp, direction(none)*

**Initial Goals:** *!pstartup*

**Plans:**

$+normalFlight$ : { T }  $\leftarrow -su(S), send(su, poll)$  ;  
 $+!p pollAgents$  : { T }  $\leftarrow -su(S), send(su, poll)$  ;  
 $+su(S)$ : { T }  $\leftarrow -nm(N), send(nm, poll)$  ;



```

+nm(N): {B su(S), ¬G makeDecision(S,N) } ← +!p makeDecision(S,N);
+nm(X,N): {B su(X,S), ¬G makeDecision(S,N) } ← +!p makeDecision(S,N);
+!p makeDecision(objAppr,headingOk) : {B normalFlight} ← +!p handleObjAppr;
+!p makeDecision(objAppr,changeHeading) : {B normalFlight} ← +!p handleObjAppr;
+!p handleObjAppr : {B normalFlight, B direction (D)} ← -normalFlight, lock,
    -direction(D), +direction(right), unlock, +emergencyAvoid, +objectIsApproaching,
    +!p pollAgents;

```

The chief difference between the Gwendolen and PROMELA models is that the Executive’s behaviours are specified in terms of beliefs, desires and intentions, which provide a richer language for describing autonomous behaviour. For instance, “the UAS is taxiing”, “the UAS wants to taxi”, “the UAS believes it is taxiing”, and “the UAS intends to taxi”, are all distinct for a BDI agent. Furthermore it is possible to reason about other agents’ beliefs, such as “the UAS believes that the ATC believes the UAS is taxiing”, allowing for richer interactions between different parts of the model than is found with similar processes in PROMELA.

The trade-off is that whilst BDI agent software is more representative of natural-world intelligent systems and provides improved expressiveness for describing autonomous systems, the added complexity of the agent programs makes subsequent model checking *much* slower. In general, we talk in terms of minutes and hours for verifying UAS agent programs, as opposed to milliseconds for the simpler PROMELA programs.

In our implementation the architecture of the Gwendolen UAS model is slightly different from the PROMELA model. Firstly, we modelled the Executive as a Gwendolen agent, but the ATC, NM and SU were modelled within the agent’s Java environment. The reason for this was that it makes intuitive sense; the Executive is the autonomous part of the model on which we focus our model checking efforts, and therefore is programmed in Gwendolen, a language for autonomous agents. Also, a future objective is to be able to connect the Executive to simulated sensors and navigation systems within a networked simulation environment, and replacing the Java simulated environment with a networked simulated environment was simpler than connecting Gwendolen agent models of the SU, NM and ATC to the networked simulation environment. In model checking terms there is little difference; the simulated SU, NM and ATC in Java perform the same function as the corresponding processes in PROMELA, and enable the full state space of the Executive to be explored.

#### 4.1 Model Checking Reasoning UAS Agents

Agents are often written in agent programming languages, so we need an agent model checker to verify agent programs [4]. We use AJPF (for Agent JPF), which works by providing a Java interface for BDI agent programming languages called the Agent Infrastructure Layer (AIL) [17]. Interpreters for agent programming languages are written using the AIL, and the resulting Java program can then be verified via AJPF [2][1]. AJPF is, in turn, built on JPF, the Java PathFinder model checker developed at NASA Ames Research Center [25][16]. For example, an agent program written in Gwendolen is executed by an interpreter written in Java and using the AIL. Temporal properties can then be checked against the model using AJPF. We verified our UAS agent model using

this method. For consistency we used the same subset of the Rules of the Air earlier used for the PROMELA UAS model. The properties verified are as follows.

1. **Sense and Avoid:**  $\Box(B(\text{exec}, \text{objectIsApproaching}) \implies B(\text{exec}, \text{direction}(\text{right})))$
2. **Navigation in Aerodrome Airspace:**  
 $\Box(B(\text{exec}, \text{changeHeading}) \wedge B(\text{exec}, \text{nearAerodrome}) \wedge \neg B(\text{exec}, \text{toldOtherwise})$   
 $\implies \neg B(\text{exec}, \text{direction}(\text{right})))$
3. **ATC Clearance:**  $\Box(B(\text{exec}, \text{taxiing}) \implies B(\text{exec}, \text{taxiClearanceGiven}))$

Here we use the belief operator ‘B’ to specify beliefs about the agents being verified, e.g., property 1 translates as, “It is always the case that if the agent ‘exec’ believes that an object is approaching, then it also believes that its direction is to the right.”

In order to test the usefulness of our UAS model, we introduced a minor error into the code to simulate a typical software engineering error. Normally, when the UAS has discovered that there is an object approaching head-on and that it should also change heading it prioritises the former, as avoiding a potential collision takes precedence over navigation. However, our error caused the UAS to have no such priority. The net effect on the UAS behaviour is that it would start to turn right to avoid the object, but would then turn left to navigate (as it was within aerodrome airspace). Specifically, the errant code was as follows:

```
+!_p makeDecision(objAppr,changeHeading){ B¬normalFlight(X) } ← +!_p handleObjAppr(X),
+!_p handleChangeHeading(X);
```

The model checker found the fault when we verified the “Sense and Avoid” property.

## 4.2 Model Checking More Advanced Autonomy in UAS Agents

The UAS agent model constructed so far will always turn right when an object is approaching head-on. This is in accordance with the Rules of the Air. However there *may* be occasions when it is advantageous (or indeed necessary) for the UAS agent to disobey certain Rules of the Air in order to maintain a safe situation. For instance, consider the case where an object is approaching head-on, and the UAS agent “knows” it should turn to the right. However, the approaching aircraft may indicate that its intention is to turn to the left (e.g., by initiating a roll to the left, manifested by its left wing dropping). At this point a rational pilot would assume that the other aircraft is going to turn left, and would realise that turning right would greatly increase the possibility of a collision. Turning left would be the more rational action to take. Likewise, if the other aircraft’s intention is to turn right, the rational action is to turn right. If the intention is unknown, then the rational action is to follow the Rules of the Air, i.e., turn right.

We added several plans to our UAS agent model in order to make the agent adopt this more advanced autonomous behaviour. The sensor unit was re-written, so that instead of sending an “object approaching head-on” message, it now sends information about intentions, e.g., “object approaching head-on and its intention is to go left.” The UAS was then enhanced to take into account beliefs about the other object’s intentions when making a decision about which way to go when an object is approaching head-on:

```
+!_p makeDecision(objectApproaching(intentionTurnLeft),changeHeading) :
B normalFlight(X) <- +intention(turnLeft),+_p handleObjAppr(X)
```

In other words, “When the Executive has to decide between an object approaching head-on (and intending to turn left) and a directive from the navigation manager to change heading, and the Executive believes it is in normal flight mode, it will add the belief that the object’s intention is to turn left, and will add as a goal to handle the object approaching by taking evasive action.” Adding such advanced autonomy will cause the UAS agent to disobey the Rule of the Air concerning turning right when an object is approaching head-on in the name of safety. The reason is that there will be times when there is an object approaching head-on, but the UAS turns left because it has detected the intention of the object is to turn left. For this reason we must modify the properties being checked. For instance the rule in Section 4.1 concerning turning right when there is an object approaching head-on becomes:

$$\Box(B(\text{exec}, \text{objectIsApproaching}) \wedge B(\text{exec}, \text{intention}(\text{right})) \implies B(\text{exec}, \text{direction}(\text{right})))$$

In other words, “It is always the case that if the Executive believes there is an object approaching head on and the intention of the object is to turn right, then the UAS turns right.” We verified similar properties for the cases where the intention is to turn left and where the intention is unknown, finding that the agent satisfied all three cases, as well as the “Navigation in Aerodrome Airspace” and “ATC Clearance” properties.

It is important to note that, in practice, there is no conflict between this advanced autonomous behaviour and the Rules of the Air, as the advanced behaviour is similar to what would be expected of a human pilot. All Rules of the Air are subject to interpretation, i.e., the previously mentioned *airmanship*; there are times when the strict Rules of the Air must be disobeyed in order to maintain safe operations.

## 5 Conclusions

We have constructed basic agent models of Unmanned Aircraft Systems for two different model checking platforms: PROMELA / SPIN for standard model checking and Gwendolen / AJPF for agent model checking. In each case we tested our UAS model against a small subset of the Rules of the Air corresponding to the following cases:

1. Sense and Avoid;
2. Navigation in Aerodrome Airspace; and
3. Air Traffic Control Clearance.

These rules were chosen as interesting cases of UAS autonomy: “Sense and Avoid” and “human in the loop” cases (rules 1 and 3 respectively) are essential for UAS engineering [21]. In addition, rules 1 and 2 are interesting because they are potentially conflicting, presenting an interesting challenge for engineering and verification.

The model we constructed in SPIN / PROMELA was very fast in terms of verification, requiring only milliseconds and megabytes to model-check a Rule of the Air. However, its low-level process-modelling and state-transition systems presented problems when it came to modelling more advanced autonomy, as this is something for which those verification systems were not designed. Agent languages in the BDI tradition (Gwendolen being one such example) allow faster and more accurate engineering of autonomous systems, but this comes at a price: in our example, the time required for verification of a single Rule of the Air property increased to minutes and hours.

The models and temporal requirements we have used are relatively straightforward. However, since most of the elements within the UAS control system are likely to be similarly simple and since quite a number of Rules of the Air are similarly straightforward, then our preliminary results suggest that it is indeed feasible to use formal methods (and model checking in particular) to establish UAS compliance with at least *some* of the Rules of the Air. The areas where the models/designs might be more sophisticated and where the Rules of the Air go beyond a straightforward representation are considered in the subsequent section of future work. We are confident that this approach can move us towards acceptable certification for autonomous UAS.

A possible disadvantage of our approach, from the perspective of certification of airworthiness, is that for an existing UAS agent (written in a compiled language such as SPARK Ada) any models written in PROMELA or Gwendolen may not be accurate, so that the verification process will not lead to useful evidence for certification. A well-known way to avoid this problem is to specify the agent architecture using a process modelling language, and then use a formal software development methodology to accurately implement the specification. Alternatively, in the case of AJPF, implementation may not even be necessary as the result of the verification process is code executable within a Java virtual machine — the agent is effectively already implemented.

Another possible difficulty is in justifying the abstractions made during the modelling process. Applying our approach to a given autonomous UAS control system requires modelling the system, e.g., using PROMELA or the Gwendolen agent language. The conclusions drawn from model checking are only as useful as our confidence in the model itself; therefore model validation is important when applying our approach to implemented autonomous UAS systems. For similar reasons, the properties used for model checking would need to be validated with respect to required standards of behaviour.

## 5.1 Impact

Two principal questions for UAS manufacturers are whether Formal Methods has anything to offer autonomous UAS, and if so, what kind of approaches should be used and in what manner? These are the questions that we have started to answer but the answer is by no means complete; the construction of the models described in the paper has shown that the SPIN and Agent JPF model checkers are well-suited to the task of specifying and analysing autonomous UAS behaviour. Furthermore, the paper demonstrates that these models can be checked to be in accordance with a small subset of the Rules of the Air, a statutory document specifying many of the requirements of pilots and aircraft in UK airspace. Therefore the paper has demonstrated that Formal Methods could indeed be useful for providing evidence to regulatory authorities that a given autonomous UAS is airworthy and presents no additional risks beyond those currently encountered by traditional manned aircraft. This is a small but crucial first step on the road to certification, which is likely to require intensive investigation by both academic and industrial researchers over the coming years. This work has begun to show how the problem of verifying that an autonomous computer system is equivalent to a human might be tackled.

## 5.2 Related and Future Work

This paper has focused on the problem of engineering and certification of autonomous UAS, with the emphasis on verification of high-level decision making. However there is a wealth of literature in the field of control engineering concerning automatic flight control systems (e.g., autopilot, autoland) designed to assist the safe operation of manned vehicles [18]. In addition there is much in the literature concerning Airborne Collision Avoidance Systems (ACAS) which have tackled the sense-and-avoid problem, primarily in the arena of manned aircraft [26]. In this paper we attempted to formalise Rules of the Air (written in natural language) to derive properties describing the desired behaviour of autonomous UAS. These properties could then be checked against a model of an autonomous UAS control system. Deriving formal specifications from requirements written in natural language has also been examined elsewhere, e.g., [19].

There have been several uses of formal methods in UAS. For example: Sward used SPARK Ada to prove correctness of UAV cooperative software [24]; Chaudemar et al. use the Event-B formalism to describe safety architectures for autonomous UAVs [6]; Jeyaraman et al. use Kripke models to model multi-UAV teams and use SPIN to verify safety and reachability properties amongst others [14]; Sirigineedi et al. use Kripke models to model UAV cooperative search missions, and use the SMV model checker to show that the UAVs do not violate key safety properties [23]. Formal methods have also been applied to autonomous systems in the aerospace domain: Pike et al. describe an approach to V&V of UAVs using lightweight domain-specific languages; Brat et al. use the PolySpace C++ Verifier and the assume-guarantee framework to verify autonomous systems for space applications [5]; while Bordini et al. proposed the use of model checkers to verify human-robot teamwork in space [3]. Importantly, none of these use formal verification to establish that an autonomous systems is “equivalent” (even to a limited extent) to a human pilot, as we do here.

In this paper we have only modelled a very basic UAS. Adding functionality would add complexity to the model and likely increase verification time, although quantifying this is difficult without having a more complete model to hand. For a complete test of UAS airworthiness we also need to verify the UAS subsystems with which our “Executive” communicates: various avionics systems including sensors, actuators and automatic flight control systems would all need to be certified separately and together, presumably using existing methods such as SPARK Ada.

However, an obvious next step is to expand the functionality of the UAS as we have described it, and test whether it is possible to verify it against increasingly large subsets of the Rules of the Air. Another interesting avenue would be to obtain “real-life” UAS source code, or an abstract state transition system describing the behaviour of an already-operational UAS, and generate a model of its control system in order to verify different aspects of its airworthiness.

A key area for future research is in the management of complexity: as the complexity of the model of autonomous UAS behaviour increases, so will the time and space required for verification by the model checker. However it is possible that novel abstractions, modelling techniques and advances in computer technology and model checking software will mitigate this problem.

An immediate aim is to use the formally verified Executive agent within a virtual prototype of an autonomous UAS, including agent(s), UAV, complex flight control system, sensors and ground control station, and test whether Monte Carlo methods can be used to quantify UAS behaviour and provide evidence for certification.

**Acknowledgements.** The authors would like to thank Charles Patchett and Ben Gorry of BAE Systems (Warton) for their guidance and support.

This work is supported through the Virtual Engineering Centre (VEC), which is a University of Liverpool initiative in partnership with the Northwest Aerospace Alliance, the Science and Technology Facilities Council (Daresbury Laboratory), BAE Systems, Morson Projects and Airbus (UK). The VEC is funded by the Northwest Regional Development Agency (NWDA) and European Regional Development Fund (ERDF) to provide a focal point for virtual engineering research, education and skills development, best practice demonstration, and knowledge transfer to the aerospace sector.

## References

1. Bordini, R., Dastani, M., Dix, J., El Fallah-Seghrouchni, A. (eds.): *Multi-Agent Programming: Languages, Tools and Applications*. Springer, Heidelberg (2009)
2. Bordini, R.H., Dennis, L.A., Farwer, B., Fisher, M.: Automated Verification of Multi-Agent Programs. In: *Proc. 23rd Int. Conf. Automated Software Engineering (ASE)*, pp. 69–78. IEEE Computer Society Press, Los Alamitos (2008)
3. Bordini, R.H., Fisher, M., Sierhuis, M.: Formal Verification of Human-Robot Teamwork. In: *Proc. 4th Int. Conf. Human-Robot Interaction (HRI)*, pp. 267–268. ACM, New York (2009)
4. Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M.: Model Checking Rational Agents. *IEEE Intelligent Systems* 19(5), 46–52 (2004)
5. Brat, G., Denney, E., Giannakopoulou, D., Frank, J., Jonsson, A.: Verification of Autonomous Systems for Space Applications. In: *Proc. IEEE Aerospace Conference* (2006)
6. Chaudemar, J.-C., Bensana, E., Seguin, C.: Model Based Safety Analysis for an Unmanned Aerial System. In: *Proc. Dependable Robots in Human Environments, DRHE* (2010)
7. Civil Aviation Authority. CAP 393 Air Navigation: The Order and the Regulations (April 2010), <http://www.caa.co.uk/docs/33/CAP393.pdf>
8. Civil Aviation Authority. CAP 722 Unmanned Aircraft System Operations in UK Airspace — Guidance (April 2010) <http://www.caa.co.uk/docs/33/CAP722.pdf>
9. Clarke, E., Grumberg, O., Peled, D.: *Model Checking*. MIT Press, Cambridge (1999)
10. Dennis, L.A., Farwer, B.: Gwendolen: A BDI Language for Verifiable Agents. In: *Logic and the Simulation of Interaction and Reasoning. AISB 2008 Workshop* (2008)
11. Dennis, L.A., Fisher, M., Webster, M.P., Bordini, R.H.: *Model Checking Agent Programming Languages*. Automated Software Engineering (in press)
12. European Aviation Safety Agency. Certification Specifications for Large Aeroplanes CS-25 (October 2003) ED Decision 2003/2/RM Final 17/10/2003.
13. Holzmann, G.: *The Spin Model Checker: Primer and Reference Manual*. AW (2004)
14. Jeyaraman, S., Tsourdos, A., Zbikowski, R., White, B.: Formal Techniques for the Modelling and Validation of a Co-operating UAV Team that uses Dubins Set for Path Planning. In: *Proc. American Control Conference* (2005)
15. Johnson, C.: Computational Concerns in the Integration of Unmanned Airborne Systems into Controlled Airspace. In: Schoitsch, E. (ed.) *SAFECOMP 2010*. LNCS, vol. 6351, pp. 142–154. Springer, Heidelberg (2010)

16. Java PathFinder, <http://javapathfinder.sourceforge.net>
17. Model-Checking Agent Programming Languages, <http://mcapl.sourceforge.net>
18. McRuer, D., Graham, D.: Flight control century: Triumphs of the systems approach. *Journal of Guidance, Control and Dynamics* 27(2), 161–173 (2004)
19. Nikora, A.P., Balcom, G.: Automated identification of LTL patterns in natural language requirements. In: *Proceedings of the 20th International Symposium on Software Reliability Engineering, ISSRE 2009*, pp. 185–194. IEEE Computer Society, Los Alamitos (2009)
20. Office of the Secretary of Defense. *Unmanned Aircraft Systems Roadmap 2005–2030*. US DoD Publication (2005)
21. Patchett, C., Ansell, D.: The Development of an Advanced Autonomous Integrated Mission System for Uninhabited Air Systems to Meet UK Airspace Requirements. In: *Proc. International Conference on Intelligent Systems, Modelling and Simulation* (2010)
22. Rao, A., Georgeff, M.: Modeling Agents within a BDI-Architecture. In: *Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pp. 473–484. Morgan Kaufmann, San Francisco (1991)
23. Sirigineedi, G., Tsourdos, A., Zbikowski, R., White, B.A.: Modelling and Verification of Multiple UAV Mission Using SMV. In: *Proc. FMA 2009. EPTCS*, vol. 20 (2009)
24. Sward, R.E.: Proving Correctness of Unmanned Aerial Vehicle Cooperative Software. In: *Proc. IEEE International Conference on Networking, Sensing and Control* (2005)
25. Visser, W., Havelund, K., Brat, G.P., Park, S., Lerda, F.: Model Checking Programs. *Automated Software Engineering* 10(2), 203–232 (2003)
26. Williams, E.: Airborne Collision Avoidance System. In: Cant, T. (ed.) *Proc. 9th Australian Workshop on Safety Critical Systems and Software, SCS 2004. Conferences in Research and Practice in Information Technology*, vol. 47, pp. 97–110 (2004)
27. Wooldridge, M.: *An Introduction to Multiagent Systems*. John Wiley & Sons, Chichester (2002)

# Verifying Functional Behaviors of Automotive Products in EAST-ADL2 Using UPPAAL-PORT

Eun-Young Kang<sup>1,2</sup>, Pierre-Yves Schobbens<sup>1</sup>, and Paul Pettersson<sup>2</sup>

<sup>1</sup> Computer Science Faculty, University of Namur, Belgium

<sup>2</sup> MDH PROGRESS Research Centre, Västerås, Sweden

{eun-young.kang, pierre-yves.schobbens}@fundp.ac.be  
paul.petterson@mdh.se

**Abstract.** We study the use of formal modeling and verification techniques at an early stage in the development of safety-critical automotive products which are originally described in the domain specific architectural language EAST-ADL2. This architectural language only focuses on the structural definition of functional blocks. However, the behavior inside each functional block is not specified and that limits formal modeling and analysis of systems behaviors as well as efficient verification of safety properties. In this paper, we tackle this problem by proposing one modeling approach, which formally captures the behavioral execution inside each functional block and their interactions, and helps to improve the formal modeling and verification capability of EAST-ADL2: the behavior of each elementary function of EAST-ADL2 is specified in UPPAAL Timed Automata. The formal syntax and semantics are defined in order to specify the behavior model inside EAST-ADL2 and their interactions. A composition of the functional behaviors is considered a network of Timed Automata that enables us to verify behaviors of the entire system using the UPPAAL model checker. The method has been demonstrated by verifying the safety of the Brake-by-wire system design.

## 1 Introduction and Main Themes

EAST-ADL2 is an architecture description language for the development of automotive embedded systems [1]. Advanced automotive functions [15,6] are increasingly dependent on software and electronics. These automotive embedded systems are becoming progressively complex and critical for the entire vehicle. Model-based development (MBD) is a means to manage this complexity and develop embedded systems in a way that increases safety and quality. The EAST-ADL2 modeling approach addresses this topic and provides means to integrate the engineering information from documents, spreadsheets and legacy tools into one systematic structure, an EAST-ADL2 system model.

Our aim is to use formal modeling techniques at an early stage in the development life cycle of automotive embedded systems, and to use symbolic simulators and model checkers as debugging and verification tools to ensure that the predicted function behaviors of the modeled system in EAST-ADL2 satisfy certain requirements under given assumptions on the environment where the system is supposed to operate.

EAST-ADL2 expresses the structure and interconnection of the system. System behavior is defined based on the definition of a set of elementary functional blocks and



their triggers and interfaces. However, the behavioral definition inside each elementary functional block is not specified, which limits the automatic translation from EAST-ADL2 models to other formal models for efficient verification. Instead, the execution of each function is described with external behavioral annexes and legacy tools including general UML tool and domain-specific tools, e.g., Simulink or UML [13]. This restricts the construction of a complete system behavior model and verification of the behavior of the entire system model with verification tools.

To achieve our goal by improving the aforementioned restriction, we propose a formal approach which facilitates the verification of system function behaviors in EAST-ADL2 by using UPPAAL-PORT model-checker [8]: this approach specifies a behavior inside of each elementary function (block) in Timed Automata (TA) and constructs a complete system behavior model by the parallel composition of local behaviors. In particular, we specify the execution of each function behavior in the UPPAAL-PORT TA model and consider a composition of the function behaviors as a network of TA so that the behaviors of the entire system in EAST-ADL2 can be formally defined. Then this network TA can be analyzed and verified by UPPAAL-PORT model checker.

This work is organized as follows. Section 2 introduces technology and background, EAST-ADL2 and UPPAAL-PORT toolkit as used in our approach. Section 3 presents our approach for verifying system behaviors in EAST-ADL2 by using UPPAAL-PORT model checker: this approach formally captures the behavior inside each functional block and their parallel compositional interactions. Furthermore, the formal definition enables transformation of the given model to models of UPPAAL-PORT tool for model checking. In section 4, our method is demonstrated in verifying the safety of the Brake-by-wire system design. We discuss further work and conclude in Section 5.

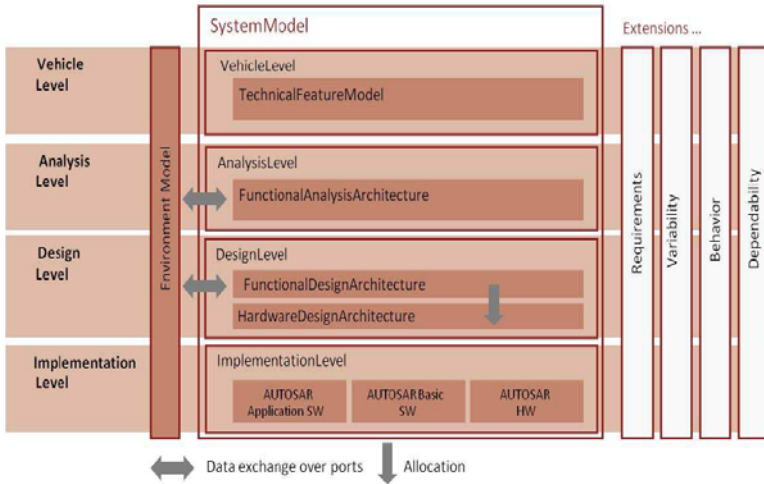
## 2 Background

### 2.1 EAST-ADL2

The goals of modeling with EAST-ADL2 are to deal with complexity control and improve safety, reliability, cost, and development efficiency through MBD. For this, EAST-ADL2 structures a system model into multiple abstraction levels in terms of the development life cycle of automotive embedded systems.

EAST-ADL2 is an information model, connecting different views of the system. The views are influenced by the different engineering traditions and backgrounds. This concept allows EAST-ADL2 to handle various types of information including requirements, vehicle features, system environment, application functions, deployment of software and hardware resources, behaviors, non-functionality properties such as variability, timing constraint, dependability, and V&V related information. Abstract solution, design, and implementation details are found in different abstraction levels in the model: the highest abstraction level, *Vehicle(Feature) level*, characterizes a vehicle by means of features and defines implementation-independent information such as features and requirements. Fig. 1 depicts an overview of the system model and the abstraction levels of EAST-ADL2.

At *Analysis level*, functionality is realized based on the features and requirements. These features and requirements are refined by the decision of logical design with the



**Fig. 1.** The structure of an EAST-ADL2 System Model

definition of logical abstract functions of features and their interactions, and requirements. The model at this level is used for the analysis of control requirements, timing constraints, data consistency between interfaces, hazard identification, etc. *Design level* contains concrete functional definition according to the realized logical design. In particular, functional definition of application software, functional abstraction of hardware and middleware are presented, as well as hardware architecture being captured and function-to-hardware allocation being defined. *Implementation level*, i.e. the software architecture, is represented using the AUTOSAR standard and allocates software modules to a network of Electronic Control Units (ECUs) according to the AUTOSAR standard [2]. As in Fig. 1, EAST-ADL2 extensions are constructs for requirements, variability, behaviors, dependability, and V&V activities, etc. EAST-ADL2 is intended to be an integration framework for functionality defined in different notations and tools. The behavioral definition therefore relies on the definition of a set of elementary functions that are executed based on the assumption of run-to-completion execution (read inputs from ports, compute, and write outputs on ports). This is chosen to enable analysis and behavioral composition and make the function execution independent of behavioral notations. Details of those issues are explained in Section 3.

## 2.2 UPPAAL-PORT

UPPAAL-PORT is a model checking tool for component based modeling, simulation, and verification of real-time and embedded systems modeled as real-time components. It can be used as an Eclipse plugin together with the SAVE integrated development environment (IDE) [16,17] in order to support graphical modeling of internal component behaviors as an UPPAAL-PORT TA and composition of components. The model checker of UPPAAL-PORT verifies properties expressed in a subset of timed *computational tree logic* (TCTL). The current input file format for UPPAAL-PORT is a component

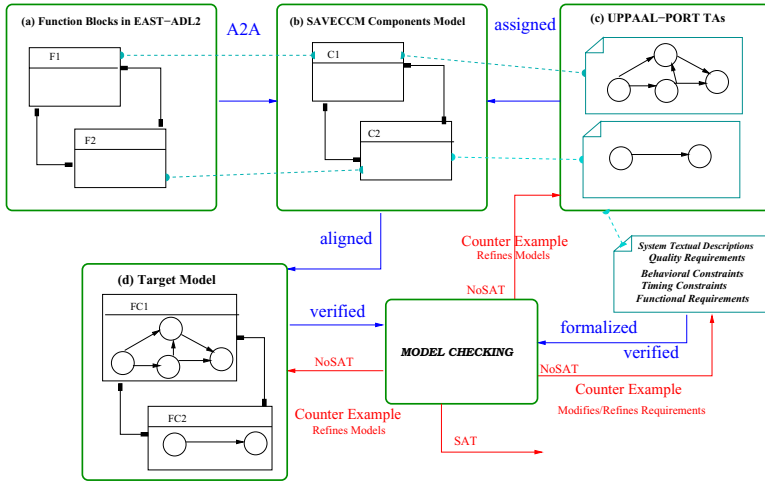


Fig. 2. Methodology Roadmap

modeling language, SAVE-CCM [3], which describes the architectural framework for modeling real time embedded applications with particular emphasis on automotive domain and safety concerns. In particular, SAVE-CCM is used to create components and interconnections among them, and supports run-to-completion semantics. We use this architectural framework of SAVE-CCM for mapping from functional blocks and their interconnectors in EAST-ADL2 to components and their interconnections in SAVE-CCM respectively.

For analysis purpose, an UPPAAL-POR TA model is assigned to each of the SAVE-CCM components in order to describe timing and functional behaviors of the component. Since EAST-ADL2 is intended for use with different behavioral notations, UPPAAL-POR TA is perfectly appropriate to use. This UPPAAL-POR TA communicates with other ones through ports and the values of the ports defined by binding TA variables to the ports of components, and supports synchronous execution with other regular UPPAAL TA models. Thus, by defining local behaviors of each EAST-ADL2 function block with UPPAAL-POR TAs, their synchronous run-to-completion execution semantics should make it possible to integrate the contained TA into a model representing the complete system. Since EAST-ADL2 also supports requirements, the invariants and other logical criteria used for modeling function behaviors with UPPAAL-POR TAs refer to requirements in EAST-ADL2. The SAVE-CCM/UPPAAL-POR TA is not an intermediate step. It is the target model we want to build as a long term goal. So the behaviors of a given system (functions and their interactions) will be more effectively analyzed.

### 3 Approach and Proposed Solution

To achieve our aforementioned goal in section 1, we propose a formal approach which facilitates the verification of system behaviors in EAST-ADL2 by using UPPAAL-POR model-checker independently of any hardware constraints and topology mapping. It

mainly focuses on the higher level of functional behavior of applications at *Analysis level* in terms of its *Feature level* with three distinct phases – architectural and behavioral mapping, behavior specification, and verification (model checking). We will discuss those phases in more detail in following sections.

### 3.1 Architecture and Behavioral Semantics Mapping

This architectural mapping step, called A2A, is an architecture-to-architecture representation from (a) to (b) in our methodology roadmap Fig. 2. The EAST-ADL2 model architecture frame at *Analysis level* in the Papyrus UML (Fig. 2-(a)) is mapped to SAVE-CCM architecture frame (Fig. 2-(b)). However, this stage is not concerned with the actual representation of the data.

This stage performs a semantic anchoring between the domains of EAST-ADL2 and that of SAVE-CCM. The purpose of the semantic anchoring is to map concepts from EAST-ADL2 to SAVE-CCM in a way that preserves the semantics of the original model without changing the structure of the model heavily. Each elementary *AnalysisFunction* (AF) has its own logical execution and no internal concurrency, therefore it maps well to a SAVE-CCM component. In this case, there is a convenient and obvious mapping possibility: we design a system model in SAVE-CCM from the given system at *Analysis level* in EAST-ADL2. We assign one SAVE-CCM component per AF element in the EAST-ADL2 system (i.e. BreakController, ABS, etc). The original interconnectors and associated ports in EAST-ADL2 are mapped to the interconnections between ports of SAVE-CCM components respectively and that enables communication with other components according to their original AF element. One or more ports and elements of EAST-ADL2 models may be realized by one port and one component of SAVE-CCM models, as these may have several signals or data elements per interface that are simplified (as abstracted design) in one port and one component in our SAVE-CCM design model.

Inside each AF, the data transformation and its own behaviors are described as TAs based on the assumption of synchronous run-to-completion execution. There are two types of function interactions in EAST-ADL2: either a *FlowPort* interaction whereby a function performs a computation on provided data, or a *ClientServer* interaction whereby the execution of a service is called upon by another function. The *FlowPort* interactions are matched to the interconnections of SAVE-CCM components. The *ClientServer* interactions are explained by the execution of the TA inside a component and its synchronization with other TAs.

The triggering of each AF is defined either as time-driven or event-driven on one of the input ports. There are two types of function entities, time-discrete function and time-continuous function. Time-discrete function is done after a computational delay, i.e. execution time. Time-continuous function defines the transfer function from input to output, and the computation rate is infinite. Since the semantic of AF is run-to-completion, there should be no infinite delays in the local UPPAAL-TA model, the time-continuous function is not concerned in our semantic anchoring and we deal only with time-discrete function. The time-discrete function is invoked either by time-triggered in which time alone causes execution to start, or event-triggered, which is caused by data arrival or calls on the input ports. Those trigger conditions are matched to those of a *clock* component, trigger ports and data type ports in SAVE-CCM, respectively.

### 3.2 Behavior Specification

We have shown a straightforward mapping from the informal semantics of EAST-ADL2 to the formal semantics of SAVE-CCM. Since EAST-ADL2 allows the use of different behavioral notations, we capitalize on this advantage to specify *FunctionBehavior* by assigning an UPPAAL-PORT TA model to each SAVE-CCM component mapped from its corresponding AF (especially *ADLFunctionPrototype*) respecting the triggering definition, and execution time of each AF as well as its requirements. This TA model encapsulates the "execution behaviors" of AF and is used for verification in terms of real-time properties by using UPPAAL-PORT model-checker. Our tooling composes such local automata in parallel to a composed TA (network TA). The purpose of this phase is to construct a target model (Fig 2(d)) by filling the architectural frame model (Fig 2(b)) with the corresponding UPPAAL-PORT TAs (Fig 2(c)). In this case, textual system description, quality/functional requirements and behavioral/timing constraints are referred to specify *FunctionBehavior* in UPPAAL-PORT TAs (Fig 2(c)).

We define an EAST-ADL2 model below. Essentially, this model is a tuple  $\langle N, CE \rangle$ , where  $N$  is a set of *ADLFunctionalPrototypes* AFs, and  $CE \subseteq N \times N$  is the set of interconnectors between AFs. Output variables of one AF may be connected to input variables of another AF. The clock component in SAVE-CCM [18] is used to define time-triggered activations. It periodically generates the triggering event to activate the component and its connected components in a sequence by sending a trigger signals through the ports. For detailed semantics of the SAVE-CCM language (the subset of ProSave), we refer the reader to [18]. These triggering (or data) signals arrive at a port with a one-place buffer. It is stored in that buffer, and for other ports it is forwarded to connected ports.

The behavior  $\mathcal{B}$  inside an AF, noted  $\llbracket \mathcal{B} \rrbracket_{AF}$ , is modeled as an UPPAAL-PORT TA  $= \langle L, l_0, l_f, V_C, V_D, E, I \rangle$ , where  $L$  is a set of locations,  $l_0 \in L$  is the initial location,  $l_f \in L$  is the final location, such that no edges in  $E$  are leading out from  $l_f$ , and is used to model the termination of an execution of AF.  $V_C$  and  $V_D$  is a set of clock and data variables respectively.  $I$  assigns an invariant to each of the locations.  $E$  is a set of edges, represented as  $l \xrightarrow{g, a, u} l'$ , where  $l$  is a source location,  $l'$  is a destination location,  $g$  is a guard,  $a$  is an action,  $u$  is an update.

The execution of behavior inside FA is determined, (i.e., a SAVE-CCM component is triggered) in terms of triggering values, which can be generated from the clock component of SAVE-CCM (named *active*). When the triggering value is *active*, the component is triggered via its input trigger port and its input data ports are mapped to data variables.  $V_D$  in TA are updated with those variables by *read-input-from-ports* action, noted  $READ(P_{in})$ , (respectively *write-output-to-ports*, noted  $WRITE(P_{out})$ ), which are atomic and urgent (in the sense that time is not allowed to pass when a component reads or writes). A component is initially *idle* after the read action it switches to its executing locations until its internal computation is done. After the write action, which forwards data in variables via interconnections from the output ports, the component becomes *idle* again and the trigger port is updated to *inactive*. Formally the behavior of AF is defined as follows:

**Definition 1 (Behavior of AF).** *The behavior of AnalysisFunction AF is a tuple  $\mathcal{B} = \langle L \cup \{l_\perp\}, l_\perp, l_0, l_f, V_D \cup P, V_C, E \cup \{e_r, e_w\}, I_\perp \rangle$  where*

- $l_{\perp}$  is the idle location.
- $P$  is the set of ports of the component described as  $P_{in} \cup P_{out} \cup P_{trig}$ , where  $P_{in}$  is a set of input ports,  $P_{out}$  is a set of output ports,  $P_{trig} \subseteq P_{in}$  is the set of trigger input ports.
- $e_r = l_{\perp} \xrightarrow{g,r,u} l_0$ , if  $g$  is triggered,  $r$  is the "read-input-from-ports" action,  $READ(P_{in})$ , and  $u$  updates  $V_D$  with input values  $(P_{in} \setminus P_{trig})$ .
- $e_w = l_f \xrightarrow{g,w,u} l_{\perp}$ , if  $g$  is true,  $w$  is the "write-outputs-on-ports" action,  $WRITE(P_{out})$ , and  $u$  resets  $P_{trig}$  to "inactive"
- $I_{\perp}(l_{\perp}) = true$ ,  $I_{\perp} = I(l)$  for  $l \neq l_{\perp}$

The TA of a composition  $C$ ,  $TA(C)$ , is defined as a network of local TA. For  $AF_i$  and its corresponding component  $C_i \in C$ , the write action in  $TA(C_i)$  is extended to update the input ports (noted  $P_{in,j}$ ) of a target component  $C_j \in C$  according to interconnections from the out ports of  $C_i$  (noted  $P_{out,i}$ ). An interconnection connects a source port  $p \in P_{out,i}$  to a target port  $p' \in P_{in,j}$  whenever variables in  $P_{in}$  of  $C$  are enabled in a way that if  $p'$  is a trigger port then  $p'$  is activated, otherwise  $p' = p$ . The edges  $e$  of  $TA(C_i)$  are explained with extended write actions as follows.

**Definition 2 (Extended Write Actions).** *The behavior  $\mathcal{B}$  inside  $AF_i$ ,  $\llbracket \mathcal{B} \rrbracket_{AF_i}$ , is  $TA(C_i) = \langle L, l_0, l_f, V_D, V_C, \{XWRITE_i(e) \mid e \in E\}, I \rangle$  such that*

- $XWRITE_i(l \xrightarrow{g,a,u} l') \triangleq (l \xrightarrow{g,w,u} l' ; WRITE(P_{out,i}))$ , if  $a = w$  and  $g$  is triggered (holds). Note that  $;$  is defined as sequential execution
- $XWRITE_i(l \xrightarrow{g,a,u} l') \triangleq l \xrightarrow{g,a,u} l'$ , for  $a \neq w$

The automata  $TA(C)$  is then the network of each  $TA(C_i)$  for  $C_i \in C$ .

An environment is modeled as  $TA_{Env}$  in a similar way. The resulting composition is thus defined as the network  $TA(C) \times TA_{Env}$ , where any edge in  $TA_{Env}$  updating ports  $P_{in}$  of  $C$ , is extended with an update  $WRITE(P_{out.Env})$ . This is similar to the adaption of the  $XWRITE$  action that is used to build  $TA(C_i)$  in Definition 2.

### 3.3 Verification: Model Checking

The execution of each *FunctionBehavior* in EAST-ADL2 is specified by UPPAAL-PORT TA in SAVE-CCM and its composition is considered as the network TA: the formal semantics of SAVE-CCM used in this paper was given in section 3.2. For the semantics of the full SAVE-CCM language, we refer the reader to [7]. The entire system (network TA) is considered in terms of a timed transition system [18], then this entire system is verified by UPPAAL-PORT model-checker. Quality requirements (e.g, timing, safety, deadlock freedom) in terms of functional requirements (e.g, behavioral constrains, timing constraints), see Fig 2. Requirements aspect, are formalized in linear time logics based on the UPPAAL logic, which can be verified over the target model (Fig 2.(d)) by UPPAAL-PORT model checker.

In particular, the quality requirements are derived from a given system's textual descriptions. One may verify certain delay, reaction and synchronization constraints (i.e, overall behavioral constraints of a system) according to the quality requirements. For

example, a plausible reaction constraint is 250 ms. In contrast, functional requirements describe particular constraints of a function such as timing constraints and trigger elements linked to an AF block. They define the triggering and execution time of the AF. UPPAAL-PORT model checker verifies those two types of requirements as safety properties in a way that (a) if a property is satisfied by the target model, then a functional requirement linked to an AF is updated to a satisfy relation and generic constraints of the AF are stored as valid invariants in the V&V structure (VVOutcome linked to the explained requirement, VVCase, etc) of the EAST-ADL2 model. (b) If a property is violated (depicted as NoSAT arrows in Fig.2) then our UPPAAL-PORT model checker returns some counterexamples that can help analysts to refine the behavioral constraints of the system model or modify generic constraints, and identify correct constraints for the AF that it concerns. Thus, the models in EAST-ADL2 are updated with the timing assumptions analysts make as well as the analysis results.

### 4 Current Result and Example

Our approach has been applied and demonstrated on a case study, the Break-by-Wire System (BWS), from our industrial partner VOLVO. It has been first modeled using Papyrus UML [11] for EAST-ADL2 in the ATESS2 project [1]. First, the BWS Papyrus UML model at *Analysis level* in EAST-ADL2 domain is translated to a SAVE-CCM model. This step is depicted in Fig. 2-(a) and the result is shown in Fig. 3.

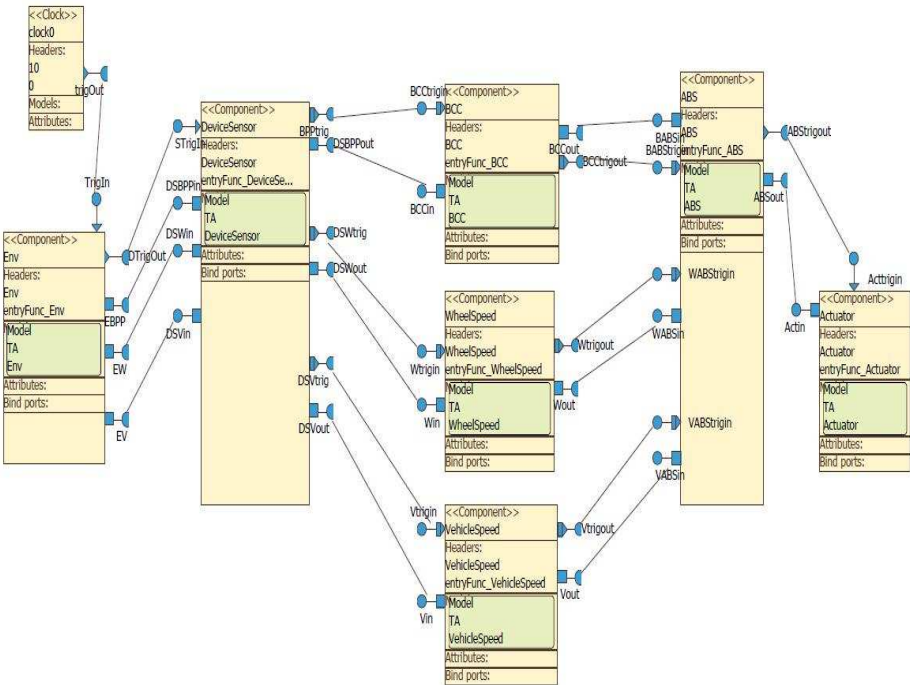


Fig. 3. The BWS architectural model (actual screenshot from the SAVE-CCM modeling tool)

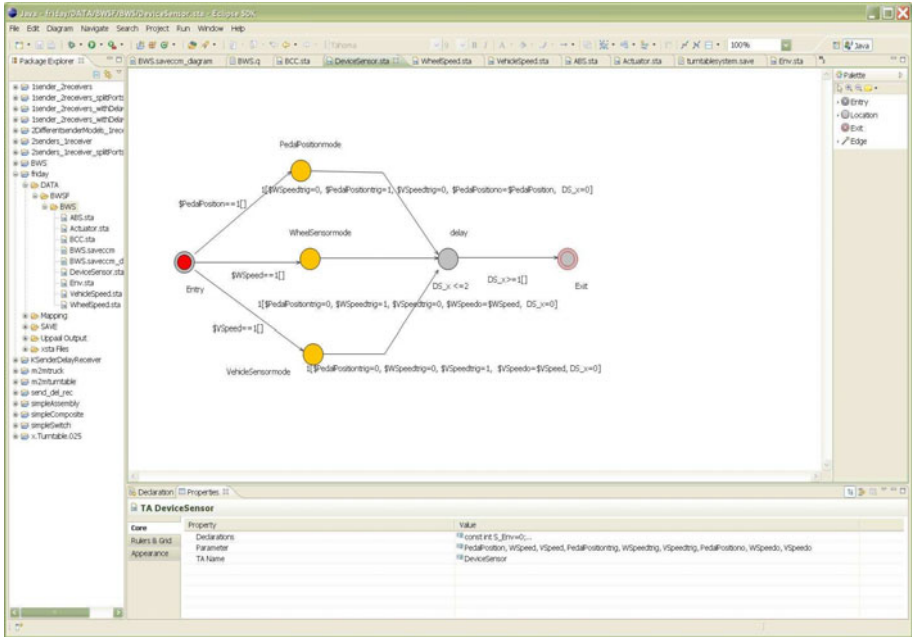


Fig. 4. BWS DeviceSensor TA

Secondly, *FunctionBehaviors* are specified in UPPAAL-PORT TAs and then are assigned to components in the SAVE-CCM model in terms of their corresponding function entities in EAST-ADL2. The functional and quality requirements of the system were given as either informal description in ATESS2 project case study reports or as timing/trigger constraints requirement entities linked to AFs in EAST-ADL2. The architecture of the system is represented by SAVE-CCM components filled with UPPAAL-PORT TAs. One of the TAs in this step is shown in Fig 4

Finally, the requirements formalized in UPPAAL logics over the result from the second step are verified by model-checking with some assumptions we make regarding timing: there is a data flow from a pedal to a brake actuator. The functions are periodic and mutually unsynchronized. A perfect clock is assumed in the sense that it generates periodic triggering in order to activate (run) the components with a periodicity of one time unit. Each function has its execution time which is modeled with a delay location in its TA. Based on those assumptions, properties of safety, deadlock freedom and liveness are verified successfully.

Data flows through ports between function blocks of BWS are simulated by using the UPPAAL-PORT plug-in for the Eclipse IDE in Fig 5. The direction of data flow is indicated by the arrow. We use this simulator in order to trace or detect fault flow paths. This is facilitated by its intuitive graphical interface that allows analysts to step forward and backward along the simulation. Apart from the simulation, we have so far verified 28 properties of the system. A list of selected properties is given below and their verification results are established as valid:



\\* *Definition of each component*

*C1 = Environment*

*C2 = DeviceSensor*

*C3 = BCC : Brake calculator and controller*

*C4 = WheelSpeed*

*C5 = VehicleSpeed*

*C6 = ABS : Anti lock Braking System*

*C7 = Actuator* \*\

- *Deadlock freedom*:  $A[\ ] \not\models$  not deadlock
- *Leads-to property* based on the internal variables of function components: every time the system is invoked by its environment it will eventually execute ABS which calculates the brake force according to the brake pedal position, wheel speed, vehicle speed:
  - $(C1.WheelDynamic \vee C1.BrakePedal \vee C1.VehicleDynamic)$   
 $\rightarrow (C6.mode == ABS \wedge C6.ForceCtr)$
- *Leads-to property* based on the values of ports: if the BrakePedal function device sends out a value of its position then the value should be received by the BrakeController function:
  - $(C1.BrakePedal \wedge C1.EBPP == 1) \rightarrow (C3.BrakeCtr \wedge C3.BCCin == 1)$
- *State correspondence check*: one internal state of a component corresponds to what is happening in the states of other environment components. The following three properties describe that while one of the function components, BCC, WheelSensor, VehicleSensor is executing, the other two function components are not executing:
  - $A[\ ] C5.VSensormode \implies (\neg C4.WSensormode \wedge \neg C3.BrakeCal)$
  - $A[\ ] C4.WSensormode \implies (\neg C3.BrakeCal \wedge \neg C5.VSensormode)$
  - $A[\ ] C3.BrakeCal \implies (\neg C5.VSensormode \wedge \neg C4.WSensormode)$
- *Execution time property*: each function component should execute within its given local execution time ( $t = 2$ ),  $0 \leq clock \leq 2$ . In other words, it should not exceed its given local execution time:
  - $A[\ ] C7.exec \implies (C7.clock \leq 2 \wedge C7.clock \geq 0)$

Since the current version of UPPAAL tool only provides reachability analysis, we first verified a certain delay in an AF component, such as its local execution time, as an invariant property. In order to verify *bounded response time properties* formula of the form  $f1 \rightarrow_{\leq T} f2$ , meaning *if a request (f1) becomes true at a certain time point, a response (f2) must be guaranteed to be true within a time bound (T)*, we apply the early experiments in [9], which showed how to check such properties with a certain syntactical manipulation on the system model, to our work either by (1) adding observer components syntactically to the system model or (2) making observer automata and synchronizing them with the actual system automata. Then we verify if both observers success states can be reached in parallel with the main actual system under the synchronization constraints.

<sup>1</sup>  $A[\ ] P$ : "P holds for any reachable configuration" is written  $A[\ ]$  in UPPAAL format.

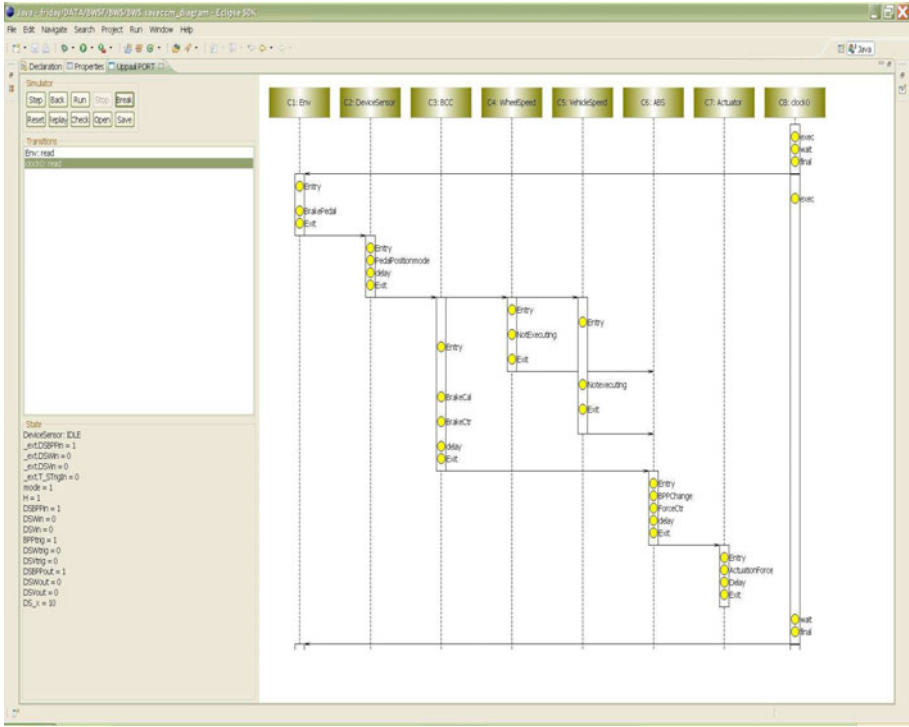


Fig. 5. BWS data flow simulation trace using UPPAAL-PORT

We construct one observer TA, illustrated in Fig 6, which contains an observer clock constraint (obsClock) as an invariant. This observer restricts the time bound of response time (MAX\_TIME). By applying this observer TA in our experiment, we successfully evaluate bounded response time properties in a way that the error location, which violates the bounded time condition, is never reached from any location of the main actual system model. The verification result is given below:

- When the brake pedal mode is activated, the actuator reacts timely under its given time bound (MAX\_TIME) as a failsafe against serious accident. i.e.,
  - $A[ ] C1.BrakePedal \implies (\neg ObsTA.error \wedge C7.Actuator)$ . The property is valid. In other words, if the BrakePedal function component is invoked, it should not reach the error location of the observer TA, which violates the MAX\_TIME bounded time condition, while the Actuator component is executing.

Search order is breadth first and uses conservative space optimization. The state space representation uses difference bound matrices (DBM). Verifying properties takes an average of around 2 seconds per verified property on an Intel T9600 2.80 GHz processor. The verification tool only needs to explore a maximum of 3584 states to verify properties such as deadlock freedom.

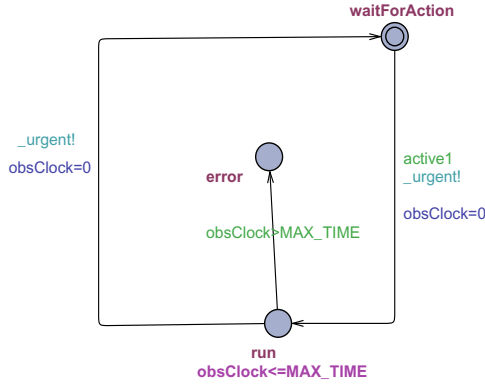


Fig. 6. Observer TA of bounded response time properties

## 5 Related Work

For safety-driven system development in the automotive domain, feature based analysis is prescribed by ISO standard as the state-of-the art approach to functional safety. However at early stage it is difficult to see function dependencies that would result in updated function requirements. Therefore, A. Sandberg et al. [14] provide one approach that performs iterative analysis to manage changes in the safety architecture at analysis level and still meet function specific safety goals derived at vehicle level. In Comparison to our work, their main concern is to define the semantics for requirement selection in order to ensure correct inclusion of requirements for a function definition. There is no formal modeling approach to the behavioral definition of the language.

L. Feng et al. [5] bring modeling formalisms to the existing behavioral principle of the system by transforming EAST-ADL2 behavior model to the SPIN model. Thus the requirements on the system design can be verified by model checking. In contrast to our work, there is no notion for the timing constraints in the behavior model. Indeed, formal analysis on the real-time properties of the behavior model is not considered at all.

## 6 Conclusion and Future Work

In this paper, we studied the use of formal modeling and verification techniques at an early stage in the development of safety-critical automotive products which are originally described in the EAST-ADL2 architectural language. While EAST-ADL2 focuses on the structural definition of functional block, we propose a method to formally specify behaviors inside each functional block in TAs mainly at *Analysis level* in terms of *Feature level* in EAST-ADL2, and verify them by using the UPPAAL-PORT model checker. The formal syntax and semantics of functional behaviors are defined. A composition of those behaviors is considered as a network of TA that allows us to verify the entire system using the UPPAAL-PORT model checker. Moreover, this paper presents a technique to verify *bounded response time properties* by adding observer components

or TA syntactically to the system model and synchronize them in parallel with the actual system automata. The contribution improves behavior modeling, verification and analysis capability of EAST-ADL2, and the result shows the applicability of model checking in safety-critical automotive products. We started from the informal description of quality and functional requirements in order to model the execution behaviors of *ADLFunctions* and manually specified them in TAs. A possible further work would be to define a formal, real-time semantics of UML diagrams so that engineers can use this familiar language. They can then be translated automatically to TAs.

In future works, we plan to extend our work: (1) From tooling perspective, in Papyrus (an Eclipse based tool platform for EAST-ADL2), UML modeling tools and domain-specific tools, e.g. Simulink, are used as external tools. They describe the data transformation inside each AF and exchange information via Plugins. Thus, an UPPAAL-PORT, which is in fact also an Eclipse based Plugin tool, would be developed as an EAST-ADL2 Plugin and be integrated with other EAST-ADL2 tools for analysis. The analysis invariants and outcome should be recorded in the EAST-ADL2 structure as valid constraints for requirements and V&V. Ideally this process should be done fully automatically. (2) Another future work includes more elaborated verification of non-functional properties, and more refined configurations of the generated model. For example, minimizing the use of certain resources, such as CPU, energy, memory, etc, while preserving functional correctness, timing requirements and other resource constraints. The results presented here are promising steps towards these goals. (3) Since the current UPPAAL tool only provides reachability analysis, observer TAs were used to verify *bounded response time properties* in this work. In order to extend this restriction, Memory Event Clocks Temporal Logic (MECTL) formula, created in our early work [12,10], will be adapted to improve fully decidable real-time expressiveness, using a tool chain that employs the UPPAAL model checker to verify properties on a system.

Furthermore, we plan to study a new design interface theory for timed-component systems [4], considered as Timed I/O automata with game semantic, that would support compositional design and verification of timed component-based embedded systems. We will employ an extended UPPAAL-TIGA, which is an engine for solving timed games in order to manipulate this design methodology.

**Acknowledgement.** This work was funded by PROGRESS Research Centre at MDH in Sweden, FUNDP PRECISE Research Center in Information Systems Engineering (CERUNA project) Namur University, and Belgian Science Policy (MoVES project). We also wish to acknowledge the participation of collaborator Volvo Technology Corporation from Sweden. Special thanks to Henrik Lönn and Lei Feng (Volvo Technology Corporation, Gothenburg, Sweden) for their valuable feedback.

## References

1. Advancing Traffic Efficiency and Safety through Software Technology Phase 2, European project (2010), <http://www.atesst.org>
2. AUTomotive Open System Architecture (2010), <http://www.autosar.org>

3. Carlson, J., Håkansson, J., Pettersson, P.: SaveCCM: An analysable component model for real-time systems. In: Liu, Z., Barbosa, L. (eds.) *Proceedings of the 2nd Workshop on Formal Aspects of Components Software (FACS 2005)*. *Electronic Notes in Theoretical Computer Science*, vol. 160, pp. 127–140. Elsevier, Amsterdam (2006)
4. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed i/o automata: a complete specification theory for real-time systems. *Hybrid Systems*, 91–100 (2010)
5. Feng, L., Chen, D., Lönn, H., Törngren, M.: Verifying system behaviors in east-adl2 with the SPIN model checker. In: *IEEE International Conference on Mechatronics and Automation, Xi'an China (August 2011)*
6. Grimm, K.: Software technology in an automotive company - major challenges. In: *International Conference on Software Engineering*, p. 498 (2003)
7. Håkansson, J.: Design and verification of component based real-time systems. PhD thesis, Uppsala University (2009)
8. Håkansson, J., Carlson, J., Monot, A., Pettersson, P., Slutej, D.: Component-based design and analysis of embedded systems with UPPAAL PORT. In: Cha, S., Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) *ATVA 2008*. LNCS, vol. 5311, pp. 252–257. Springer, Heidelberg (2008)
9. Lindahl, M., Pettersson, P., Yi, W.: Formal design and analysis of a gear controller. In: Steffen, B. (ed.) *TACAS 1998*. LNCS, vol. 1384, pp. 281–297. Springer, Heidelberg (1998)
10. Jerson Ortiz, J., Legay, A., Schobbens, P.-Y.: Memory event clocks. In: Chatterjee, K., Henzinger, T.A. (eds.) *FORMATS 2010*. LNCS, vol. 6246, pp. 198–212. Springer, Heidelberg (2010)
11. Open Source Tool for Graphical UML2 Modeling (2010). <http://www.papyrusuml.org>
12. Raskin, J.-F., Schobbens, P.-Y.: State clock logic: A decidable real-time logic. In: Maler, O. (ed.) *HART 1997*. LNCS, vol. 1201, pp. 33–47. Springer, Heidelberg (1997)
13. Rumbaugh, J., Jacobson, I.: *United Modeling Language User Guide*, 2nd edn. Addison-Wesley, Reading (1998)
14. Sandberg, A., Chen, D., Lönn, H., Johansson, R., Feng, L., Törngren, M., Torchiaro, S., Tavakoli-Kolagari, R., Abele, A.: Model-based safety engineering of interdependent functions in automotive vehicles using EAST-ADL2. In: Schoitsch, E. (ed.) *SAFECOMP 2010*. LNCS, vol. 6351, pp. 332–346. Springer, Heidelberg (2010)
15. Sangiovanni-Vincentelli, A., Di Natale, M.: Embedded system design for automotive applications. *Computer* 40(10), 42–51 (2007)
16. SAVE-IDE project at source net, <http://sourceforge.net/projects/save-ide/>
17. Sentilles, S., Håkansson, J., Pettersson, P., Crnkovic, I.: SAVE-IDE, an integrated development environment for building predictable component-based embedded systems. In: *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering, ASE 2008 (September 2008)*
18. Suryadevara, J., Kang, E.-Y., Seceleanu, C., Pettersson, P.: Bridging the semantic gap between abstract models of embedded systems. In: Grunske, L., Reussner, R., Plasil, F. (eds.) *CBSE 2010*. LNCS, vol. 6092, pp. 55–73. Springer, Heidelberg (2010)

# Establishing Confidence in the Usage of Software Tools in Context of ISO 26262

Joachim Hillebrand<sup>1</sup>, Peter Reichenpfader<sup>1</sup>, Irenka Mandic<sup>2</sup>,  
Hannes Siegl<sup>2</sup>, and Christian Peer<sup>2</sup>

<sup>1</sup> Virtual Vehicle Research and Test Center, Graz, Austria  
{joachim.hillebrand,peter.reichenpfader}@v2c2.at

<sup>2</sup> Magna E-Car Systems GmbH & Co OG, Graz, Austria  
{irenka.mandic,hannes.siegl,christian.peer}@magnaecar.com

**Abstract.** The development of safety-critical electric/electronic (E/E) automotive systems is performed by an increasing number of software tools. Hence it is very important that software tool malfunctions do not have an impact on the final product. This paper proposes a systematic methodology to establish confidence in the usage of software tools. The approach has been developed on the basis of an industrial development project and is compliant to the framework required by the standard ISO 26262. The methodology is based on a multi-layered analysis that systematically identifies the risk of tool-introduced errors and error detection failures and allows for the derivation of the tool confidence level (TCL). The benefit of this methodology is to identify and reuse already existing verification measures in the development process for establishing confidence in the usage of software tools. Furthermore, the approach allows introducing new verification measures to optimize the overall development process.

**Keywords:** tool qualification, ISO 26262, automotive, tool confidence, functional safety, embedded systems.

## 1 Introduction

In the development of electric/electronic (E/E) automotive systems a multitude of software development tools is employed. A typical automotive development process involves the use of tools for requirement engineering, configuration management, hardware development (e.g. PCB layout), modeling of control systems, code generation, testing and so forth. However, there is no guarantee that software tools are free of errors. It is essential to avoid that tool malfunctions affect the developed E/E system without being detected in the following development phases according to the established development process. This is of particular importance for the development of safety-critical automotive systems where safety-relevant system failures may cause dangerous situations for drivers, passengers and other traffic participants. The new standard for functional safety of road vehicles - ISO 26262 [5] - demands measures to establish confidence in software tools, but the formulation leaves freedom to the industry to find a suitable

way to apply it in a dedicated development process. In this context automotive companies and not the tool vendors are responsible for establishing confidence in the tools. The reason for this is that only the automotive companies themselves have knowledge about the actual application of the tools in their development process.

The scope of this paper is the development of a suitable classification methodology for software tools. The methodology not only allows to establish confidence in the usage of software tools, but it also derives verification measures that can be applied in the development process.

## 2 State of the Art

So far, there is no commonly accepted approach for tool qualification or certification across standards [2]. One of the most stringent safety standards is the avionics standard DO-178B [11] [6] that requires tool qualification for all tools involved in the creation of airborne software. This standard does not permit software development with unqualified tools. In a similar way, software developed for industrial automation and machinery industry needs to be qualified according to IEC 61508 Edition 2.0 [4]. When applied in this field, the literature speaks of tool certification instead of tool qualification. The automotive standard ISO 26262 [5] is derived from IEC 61508 Edition 2.0 and speaks in this context of *confidence in the usage of software tools*. The tool qualification approaches between these two standards differ significantly.

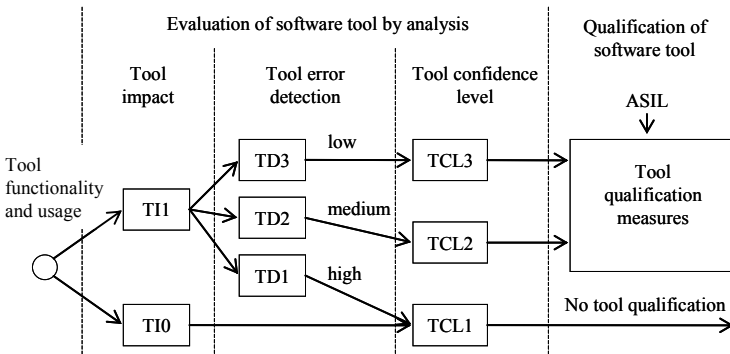


Fig. 1. ISO 26262 Tool Classification Scheme

In order to determine the required level of confidence in a software tool, ISO 26262 requires the tools to be assessed in a project-specific usage context. First, the use cases for a tool need to be documented. Based on the documented use cases, it shall be evaluated if and how a malfunction in the software tool or an erroneous output produced by the tool can violate a safety requirement. Then, the probability of preventing or detecting these malfunctions and erroneous outputs of the tool need to be evaluated. As a result of this analysis, a required

tool confidence level (TCL) is determined. For a  $TCL=1$  no further activities need to be performed. For  $TCL>1$ , ISO 26262 requires further tool qualification measures. These measures are out of scope of this paper. Figure 1 shows the tool classification scheme defined in ISO 26262. The standard leaves freedom to the industry to define the way of its implementation. Some tool vendors propose *reference workflows* [7] as a solution to establish confidence in the usage of tools. This approach facilitates the classification of the tool in the workflow, because the workflow usually contains a set of measures for error prevention and detection. On the other hand, reference workflows are based on strictly defined sets of tools which are not tailored to project specific needs and cause a strong dependency on the tool vendor and its proposed partners. The focus of tool error detection is to a large extent just shifted onto specific verification tools. Some tool vendors provide pre-defined templates [12] [8] in addition to reference workflows that help the tool user for establishing tool confidence. However, this approach is tailored to specific tools and can not be used for an entire development project involving a multitude of software tools. Besides these approaches, there are at the time of writing no established methods for best practices in industry [2]. Due to those reasons, a tool-vendor independent approach is proposed to establish tool confidence that complies with the requirements from ISO 26262.

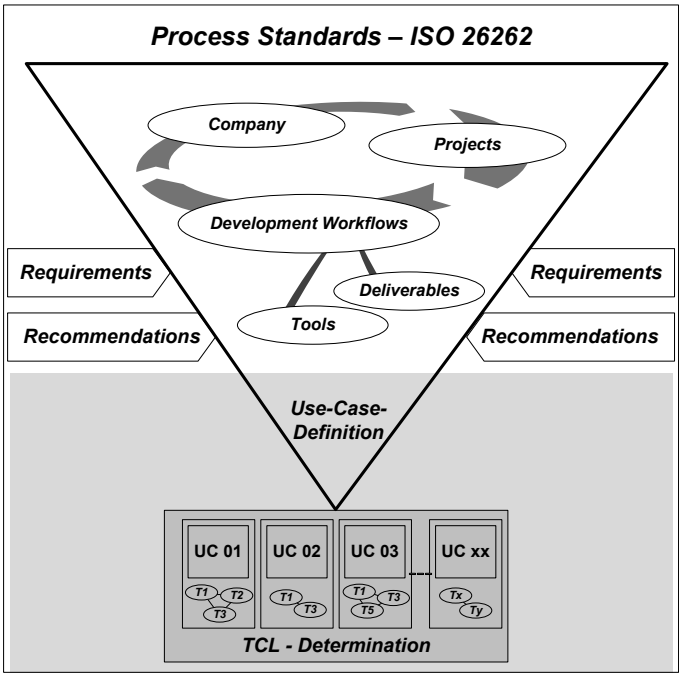
### 3 Problem Statement

In a typical industrial development process it is difficult to consider software tools separately in the context of tool classification. This can be illustrated in the following example.

First, there is a software package that provides an environment for numerical calculus. Another tool is using this software package for the modeling of dynamic systems. A third tool extends the second with methods for automated code generation. All tools are controlled with scripts executed in the first tool. Due to the close and not self-evident interaction of these different tools, it is challenging to determine the origin of a malfunction in this tool compound. Consequently, the versions of employed tools and compatibility issues also have a major impact when a tool chain is set up.

Traditionally, it is avoided to change the tool version during a development project and introduce a new tool version at the start of a project. In addition, it is quite common that users are involved in several development projects simultaneously in order to apply their expertise efficiently. This leads to the fact that users need to work with several versions of the same tool and also with several coordinated tool bundles. It can be seen that there is a strong dependency between employed tools and specific development projects. The establishment of tool confidence also depends on other influence factors like project deliverables with development workflows and requirements and recommendations demanded by ISO 26262, see Figure 2.





**Fig. 2.** Characteristics influencing the confidence in the usage of software tools

For establishment of tool confidence, ISO 26262 requires the definition of following information [5]:

1. Identification and version number of the software tool
2. Configuration of the software tool
3. Use case(s) of the software tool
4. Environment in which the software is executed
5. The maximum ASIL among the safety requirements
6. Methods to qualify the software tool, if required based on the level of confidence

Most of that information can be defined easily, except the use cases that are ambiguously defined. A use case describes the user’s interaction with a software tool or an applied subset of the software tool’s functionality. However, ISO 26262 does not define how the use cases can be derived in an industrial context in an environment of tightly interlinked tools within a development process. The main challenge for establishing tool confidence is to find a systematic way to break down the tool landscape used within a company into manageable segments. In the following section, we will present a methodology that specifies use cases and thereupon derives the tool confidence levels (TCL) for a software tool.

### 4 Proposed Approach

The focus of this approach is to establish a methodology that provides guidance for systematically identifying software tool malfunctions in a specific development project. The methodology has been developed on the basis of a representative industrial development project for automotive electric/electronic systems. The motivation for the approach is practical acceptance and applicability in the industry. Therefore, the methodology has been iteratively derived by means of interviews that involved all development groups for the chosen project.

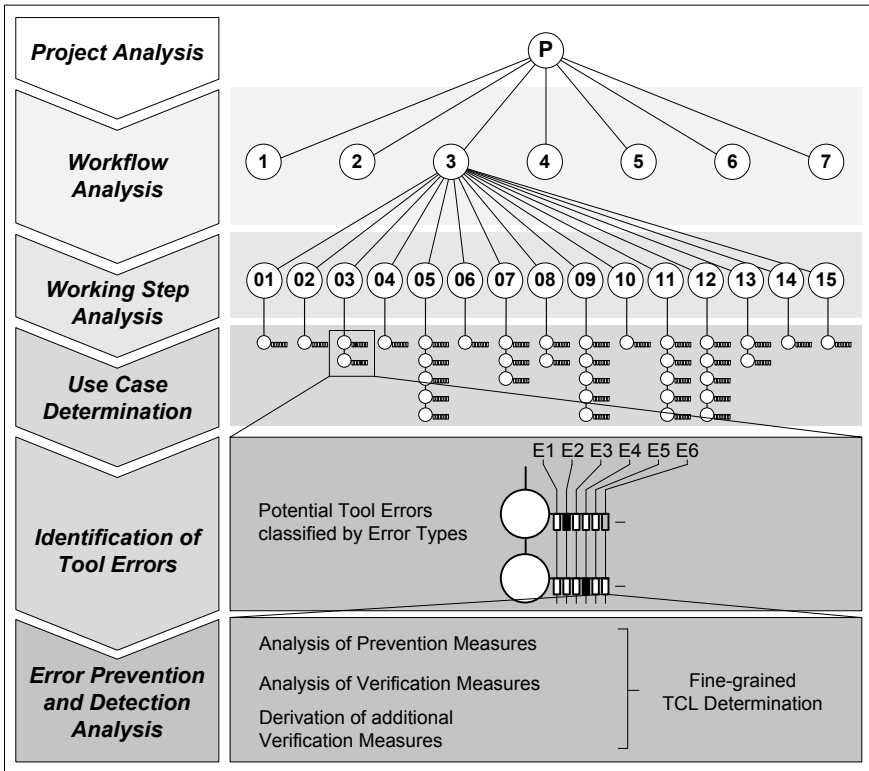


Fig. 3. Methodology

The methodology consists of following five phases:

- Project analysis
- Workflow analysis
- Working step analysis
- Use case determination

- Identification of Tool Errors
- Analysis of Error Prevention and Detection

The whole methodology is performed in a top-down sequence where each following phase refines its predecessor. Figure 3 shows a graphical illustration of the structure of the methodology. The depicted tree structure gives an example of a real industrial project where the methodology has been applied to. One project is in that case segmented into seven workflows. Each of them is assigned to a development team within an organization. In the given example, the chosen workflow number three is in turn divided into 15 working steps, whereas each of them is segmented into up to five use cases.

Each analysis step is compliant to ISO 26262 and can be mapped to the relevant parts with the respective requirements and work products. This enables a continuative basis for an assessment by accredited auditors.

#### 4.1 Project Analysis

The first step is a mapping of development processes to adequate parts of the V-model. The development processes in a company are typically well defined in internal process standards. Such standards give information on essential process steps and their process owners for product development. Considering different customers and obligation to use customer specific tools, such standards are insufficient as a basis for tool classification. The mapping to parts of the V-model is important because ISO 26262 also standardizes the respective phases. One workflow can therefore represent one section of ISO 26262, e.g. *software unit design and implementation* that is defined in part 6, chapter 8 of ISO 26262. In the organizational structure of a company, one or several workflows may be handled by one team. The top of Figure 4 shows an example for a part of the workflow analysis. This part shows the V-model and the breakdown of respective tasks concerning software development. Not included in this example are additional workflows from e.g. hardware development or system development.

#### 4.2 Workflow Analysis

The next step is the detailing of workflows that are executed within the development project. This is done by interviewing responsible persons of each development step. This gives a deeper understanding of essential requirements that are given by ISO reference on the specific part of the workflow. Furthermore, descriptions for all essential results (work products) are given and also recommendations for reviewing and testing them. As a first result such a workflow map shows:

- Description of all essential workflow steps
- Description of used tools and their relevant input and output files within a workflow step

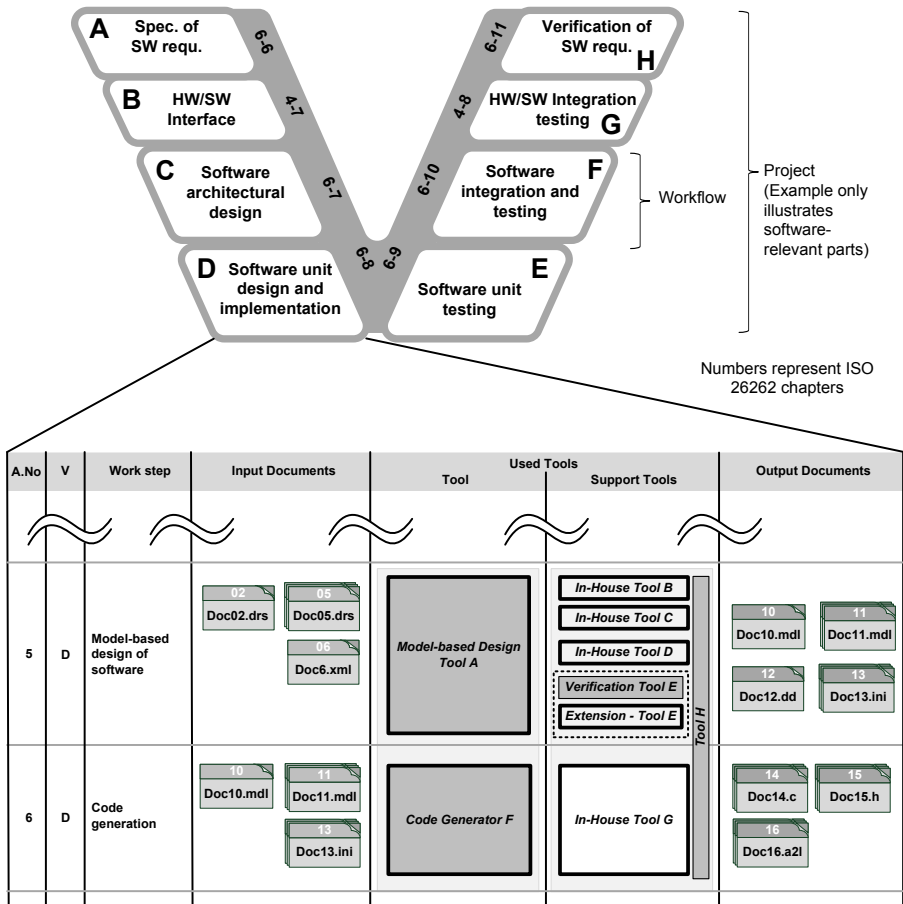


Fig. 4. Example for Project Analysis (top) and Workflow Analysis (bottom)

The workflow illustration is mainly based on typical workflow descriptions, such as eEPK [3] or UML [1]. In order to give a basis for tool classification it is necessary to focus on tools in usage. For this purpose there are distinguishing factors for the used tools:

- Distinction of tools by producer (Inhouse-tool or contract manufacturer)
- Software Licence (freeware, open source or commercial)
- Distinction of intended use (usage as main tool or support tool)

In the workflow analysis these tool types are considered by additional text (e.g. In-house tool) or by use of colored boxes. The reason for adding this information is to illustrate the maturity level, support level and the impact on the product being developed in the analysis. For illustration of used tools swimlanes are used, where on the left side the main tools are displayed and on right side support

tools are shown. There are also two columns for input and output data used in each working step which are arranged on left and right side of tool swimlanes. Tools and files are combined together in working steps.

Such a working step provides following characteristics:

- a unique identifier (activity number)
- a short description of working step
- Mapping to relevant ISO V-model
- Assignment of relevant input and output documents
- Illustration of used tools on inner swimlane

When applying the workflow analysis in the chosen industrial project, it has been detected that it is sometimes difficult to clearly separate the software tools from each other and hence create separate workflows (e.g. due to iterative loops leading to high interaction). For such cases, the workflow combines software tools to a tool compound. An example can be seen in Figure 4, activity number five.

The result of this step is a collection of all relevant workflows within development with reference to ISO 26262. These maps are the basis for the working step analysis.

### 4.3 Working Step Analysis

In this step all workflow maps are analyzed in detail. In particular input and output files of each step are checked. Each file, that is referenced in the different workflow maps is described in a file list. The file list shows the following distinctive features:

- whether it is a single file or a collection of files
- whether a file is reviewed or not reviewed within development
- Mapping to adequate parts of ISO 26262 requirements
- linkage to relevant work products of ISO 26262

The list also gives a short description on file content and allocates a unique identifier number to each file. This number is referenced in each workflow step. The working step analysis can also be used to derive measures for the detection and prevention of errors in the development process. An example are available review checklists for specific work products in an organization that can be analyzed and upgraded with this input. These checklists provide criteria for requirement specification review, architecture review, implementation review and verification and validation. The criterias have been collected from ISO 26262 as well as other standardized works such as MISRA-C [9]. These checklists are the basis for the verification of all documents to be reviewed.

### 4.4 Use Case Determination

The purpose of the use case determination is a further break-down of the work-steps into use cases. The detailing of such use cases is done with interviews

of responsible persons. According to ISO 26262, use cases describe *the user's interactions with a software tool or an applied subset of the software tool's functionality*. Even when a single workflow cannot be broken down any further in the workflow analysis due to the interconnectedness of software tools, there is some potential by separating a workflow step according to input and output files or the sequence of tool usage. If e.g. a whole software tool compound is used to create two output files, the use case determination may lead to one use case per output file. As opposed to the workflow analysis, the use case analysis may consider intermediate results that are not filed work products in the development process. The practical application of the use case determination in the industrial project showed that about half of the worksteps can be broken down further into several use cases. For the remaining worksteps there is a one-to-one mapping to use cases. The results of the use case determination are the basis for the tool error classification.

### 4.5 Identification of Tool Errors

After the workflows are broken down into atomic use cases, the proposed methodology identifies error possibilities in each use case. This is also performed by use of interviews with responsible persons. Since errors can occur in manifold ways,

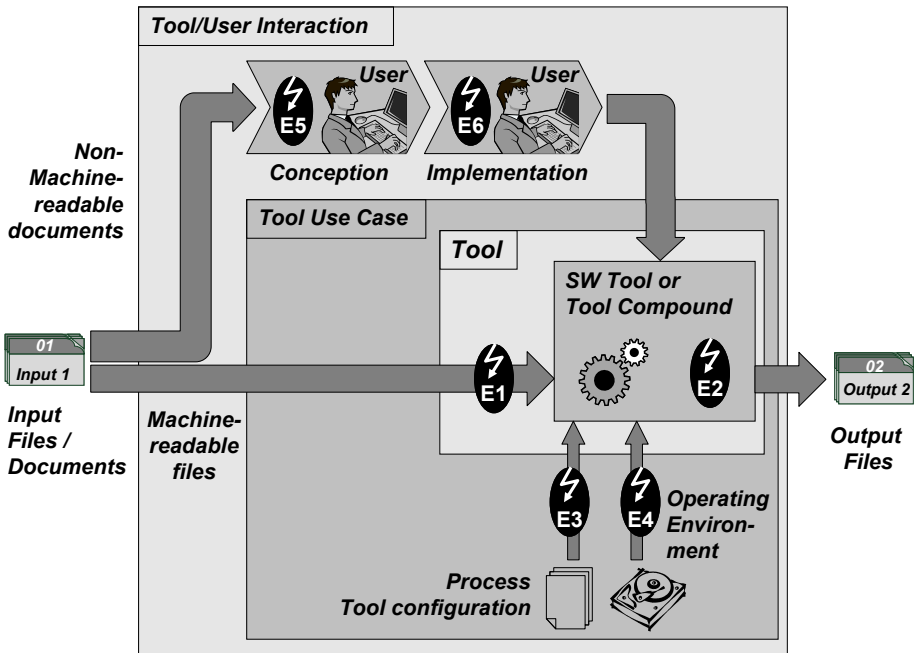


Fig. 5. Error Model

**Table 1.** Error Types

Error Type	Description	Example (Spreadsheet application)
E1	Input error, errors when reading input files	Parsing error when importing CSV files
E2	Processing error of tool or tool compound	Calculation of incorrect sum in some circumstances
E3	Error in process configuration Errors caused by various configuration files and settings	Use of wrong or outdated template files
E4	Error in operating environment Errors caused by operating system, hardware and network failures	Language settings in operating system lead to different decimal point presentation in spreadsheet application
E5	Misconception by user Tool-independent errors introduced by user due to wrong interpretation	Selection of wrong cell range due to user misinterpretation
E6	Implementation errors by user Tool-specific handling and implementation errors	Typing or pointing errors, choice of wrong function or creating the wrong formula, accidentally overwritten formulae, copy/paste errors [10]

the main errors are derived by using an error model for each use case as depicted in Figure 5.

The generic error model illustrated in Figure 5 has been developed in a way that it can describe arbitrary tool/user scenarios. As shown in Table 1, it allows deriving six basic error types that can be used to classify errors. The error type E1 and E2 are specific errors that can occur with the respective software tool. These errors include all serious intricate problems tools can cause, e.g. mistakes on generated code, incomplete coverage during testing and so forth. E3 and E4 are general errors that consider the embedding of the software tools into the project including process and environment. E5 and E6 take into account errors that occur in interaction of users with software tools. The latter two error types are strictly speaking not part of a use case analysis. They are added in order to get the complete overview of software tool and user interaction. The benefit of this extension is that consistent verification measures can be derived.

The error types have been defined as generic as possible in order to apply it to various different types of software tools. Table 2 shows an example how the tool error classification can be performed. In the depicted case the selected use case is a spread sheet tool that is used by a developer for the management of test vectors. Potential tool errors are identified by listing all errors that apply for the respective use case. In a typical software development process this methodology will reveal more than one potential error per error type. In Table 2, these errors are treated in a combined way, i.e. there is one line per error type. For a larger number of potential errors, a more fine-grained approach may be feasible.

**Table 2.** Tool Error Classification Check list (exemplary spreadsheet application)

Identification of Tool Errors			Analysis of Error Prevention and Detection							
Use Case	Error Type	Potential Errors	Measures			Detection probability	TD	TI	TCL	
			Prevention	Review	Test					Detailed description
Use Case X.Y Management of Test Vectors	E1	Search path of tool includes several input files with same name	X			Versioning	high	1	1	1
	E2	Software bug when calculating values in cells		X		Review of output files	high	1	1	1
	E3	Use of wrong templates; use of different language settings; Use of different tool version	X			Prevention by configuration management	high	1	1	1
	E4	Saving on network drive without quota; Hard disk failure	X			Versioning; hard disk failure is random failure	high	1	1	1
	E5	User fills in wrong cell due to misunderstanding of requirements		X		Review of output files	high	1	1	1
	E6	Typing mistakes of user		X		Plausibility check in extra cells in table	medium	2	1	2

#### 4.6 Analysis of Error Prevention and Detection

The right side of Table 2 illustrates the derivation of verification measures. The goal of this task is to achieve a high detection probability of software tool malfunctions. The verification measures are systematically grouped into three categories:

- Prevention: The error can be avoided by preventive measures due to the development process or configuration management. In an industrial context, the analysis of prevention measures must be based on existing documentation of process information.
- Review: The error can be detected by a review of work products. In a rigorous analysis the review examines the availability of checklists for specific development steps and verifies the quality and completeness of the review protocols.
- Test: The error can be detected by a test with another software tool within the product-specific tool chain. The analysis of tests verifies the quality of performed tests, e.g. if test cases are generated systematically.



Many of these verification measures may already be present in the respective development process. Therefore, one benefit of this methodology is to identify and reuse already existing verification measures for establishing confidence in the usage of software tools. In a mature industrial development process, formal and systematic checklists can be a basis for the derivation of verification measures. Another benefit of the methodology is the ability to optimize the overall development process for cases where verification measures are not sufficient before applying this methodology. When applying in industrial context, Table 2 needs to be further refined in order to achieve a rigorous and comprehensive checklist. One possibility is to strictly split up several potential errors listed under one error type into separate lines for derivation of separate measures.

The detection probability of tool errors is estimated in interviews with experts that are most familiar with the respective measures. In the shown example the detection probability is classified in three levels that are directly mapped to tool error detection (TD) levels, tool impact level (TI) and consequently tool confidence levels (TCL) defined in ISO 26262. The estimation of detection probabilities have been performed by using a conservative way as defined in ISO 2626.

## 5 Conclusion and Outlook

This paper gives an overview of a systematic methodology to establish confidence in the usage of software tools in an industrial project for the development of automotive systems. The methodology involves a multi-layered analysis that allows to systematically break-down the complex interaction of software tools into lists of use cases and classify the potential errors on a fine-grained level. The tool confidence levels (TCL) can be derived in compliance with ISO 26262. Development processes are linked to ISO reference processes described in ISO 26262 and all requirements, recommendations and work products given by ISO 26262 are referenced by internal deliverables. The industrial application of the approach has shown that the systematic methodology is deployable for all kind of software tools in the development of electric/electronic automotive systems. The benefit of the methodology is the derivation of verification methods that allows to optimize the overall development process. The applicability of the methodology has been proven in the course of an industrial development project. Future work will analyze how the methodology can be applied to large scale industrial systems where tool confidence needs to be established in families of projects.

**Acknowledgments.** The authors would like to acknowledge the financial support of the *COMET K2 - Competence Centres for Excellent Technologies Programme* of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG). We would furthermore like to express our thanks to our supporting industrial and scientific project

partners, namely Magna E-Car Systems GmbH & Co OG and to the Graz University of Technology.

## References

1. Alhir, S.S.: UML in a Nutshell. O'Reilly, Sebastopol (1998) ISBN 1-56592-448-7
2. Conrad, M., Munier, P., Rauch, F.: Qualifying Software Tools According to ISO 26262. In: MBEEES, pp. 117–128 (2010)
3. Kindler, E., et al.: On the semantics of EPCs: A vicious circle. In: Proceedings of the EPK 2002: Business Process Management using EPCs. pp. 71–80 (2002)
4. IEC 61508-2.0 Functional safety of electrical/electronic/programmable electronic safety-related systems (2010)
5. ISO 26262 - Draft International Standard Road Vehicles - Functional Safety - Part 8: Supporting Processes (2009)
6. Kornecki, A.J., Zalewski, J.: Experimental evaluation of software development tools for safety-critical real-time systems. ISSE 1(2), 176–188 (2005)
7. Beine, M.: A Model-Based Reference Workflow for the Development of Safety-Critical Software. In: Embedded Real Time Software and Systems (ERTS 2010), Toulouse (2010)
8. Conrad, M., Sauler, J., Munier, P.: Experience Report: Two-Stage Qualification of Software Tools. In: Proc. 2. EUROFORUM ISO 26262 Conference, Stuttgart, Germany (September 27-28, 2010)
9. MISRA: MISRA-C:2004 Guidelines for the Use of the C Language in Vehicle Based Software. Motor Industry Research Association, Nuneaton CV10 0TU, UK (2004)
10. Powell, S., Baker, K., Lawson, B.: Errors in operational spreadsheets. Journal of Organizational and End User Computing 21(3), 24–36 (2009)
11. RTCA Special Committee 167: Software considerations in airborne systems and equipment certification. Recommendation DO-178B, RTCA, Inc, Washington DC, USA (December 1992)
12. The Mathworks, Inc: IEC Certification Kit product page, <http://www.mathworks.com/products/iec-61508>

# Fault-Based Generation of Test Cases from UML-Models – Approach and Some Experiences

Rupert Schlick<sup>1</sup>, Wolfgang Herzner<sup>1</sup>, and Elisabeth Jöbstl<sup>2</sup>

<sup>1</sup> AIT Austrian Institute of Technology, Safety & Security Department,  
A-1220 Vienna, Austria

{Rupert.Schlick,Wolfgang.Herzner}@ait.ac.at

<sup>2</sup> Graz Univ. of Technology, Institute for Software Technology  
A-8010 Graz, Austria

ejoebstl@ist.tugraz.at

**Abstract.** In principle, automated test case generation – both from source code and models – is a fairly evolved technology, which is on the way to common use in industrial testing and quality assessment of safety-related, software-intensive systems. However, common coverage measures such as branch or MC/DC<sup>1</sup> for source code and states or transitions for state-based models provide only very limited information about the covered (implementation) faults. Fault-based test case generation tries to improve this situation by looking for detecting faults explicitly. This paper describes an approach combining fault- and model-based testing which has been realized in the European project MOGENTES<sup>2</sup>, using UML state machines for representing requirements, and discusses results of its application to a use case from the automotive domain.

**Keywords:** model-based test case generation, fault models, MOGENTES, UML, action systems, mutation testing, ioco (input/output conformance).

## 1 Introduction and Motivation

The continuously increasing complexity and pervasiveness of safety-related (embedded) systems requires adequate means for assessing and asserting their correct operation and reliability, in particular of their software. While formal verification of software – clearly the strongest quality assurance – is successfully applied in a growing number of cases (e.g. [9][22]), the nature of embedded systems – namely their close interaction with the physical environment as well as the specific constraints and limitations of embedded computing platforms – renders the application of formal verification methods of limited value for embedded software. One reason is that it is not sufficient to only prove the correctness of source code or the model input for code generation itself, but also that of every step/tool along the chain from code development (either

---

<sup>1</sup> “Modified Condition / Decision Coverage”: each sub-condition of a Boolean expression in a decision has at least once contributed to the value of the whole expression.

<sup>2</sup> “MOdel-based GENeration of Tests for Embedded Systems”, EU Frame Programme 7, contract number 216679; [www.mogentes.eu](http://www.mogentes.eu)

manually or automated from models) down to the execution within the target environment [17]. This is possible (see e.g. [20]), but very expensive, and hence pays only for the most critical parts of a system. Consequently, testing remains the major quality assessment technique in industry.

Testing, of course, needs *test cases*, which specify at least the input to the system and the expected result or behavior. Developing and maintaining smart test cases manually is expensive and often requires deep domain knowledge, which contributes significantly to the overall testing costs of a system development. For safety-critical applications, these testing costs can make up to 50% and more of the overall design and development costs [14].

In the last decades a number of automated test case generation methods and tools have been developed, either from source code (e.g. [15]) or from models (a state-of-the-art survey is given in [2]). It turns out, however, that the usual metrics used for test suite quality evaluation and driving the test case generation bear the risk of missing faults. Consider, for example, the following code fragment for deciding the type of a triangle with edge lengths *a*, *b*, and *c* (which is gratefully taken from [3]):

```
if ((a = b) and (b = c)) then r := "equilateral"
else if ((a = b) or (a = c) or (b = c))
  then r := "isosceles"
  else r := "scalene"
```

If it is tested with the three test cases (full branch coverage)

```
<a: 1, b: 1, c: 1, r: equilateral>
<a: 2, b: 2, c: 1, r: isosceles>
<a: 2, b: 3, c: 4, r: scalene>
```

then a typo such as “(a = a)” instead of “(a = b)” in the first line would remain undetected. However, replacing the second test case by

```
<a: 1, b: 2, c: 2, r: "isosceles">
```

would detect this fault.

The EU-project MOGENTES aimed at improving this situation by developing methods for the automated generation of test cases, which deliberately try to find implementation faults. This paper describes one of its approaches and its application to one of the four use cases addressed in MOGENTES (from automotive and railway interlocking domains), and discusses results and experiences, in particular with respect to requirements modeling. By representing requirements as UML-models, this approach combines fault- and model-based test case generation.. It is a *black-box* testing technique; the generated test cases are intended to be applied at the target (embedded) system.

Therefore, after discussing related work in the next section, section 3 gives an overview about the approach and section 4 about the use case. Section 5 discusses results and experiences, with a focus on requirements modeling, while the last section contains the conclusion and an outlook on future work.

## 2 Related Work

The large number of publications on mutation testing shows that this is an active field of research. Jia and Harman give a good overview of existing work on mutation

testing in their survey [13]. Another survey closely related to our work was conducted in the beginning of the MOGENTES project. It gives an overview of the state of the art in model-based testing [2].

Test case generation from UML state charts has also already been a topic of various publications. Since we cannot name all, here are some exemplary pieces of related work: Gnesi et al. [12] concentrate on a subset of UML state charts, formulate the semantics for this subset by transition systems with input/output pairs and present a test case generation algorithm for these systems. Seifert et al. [16] use Compact Semantic Automaton (CSA) to formalize state charts. Fröhlich et al. [11], employ AI planning to derive test suites with a given coverage. Weißleder et al. [21] derive boundary values for testing from UML class and state diagrams and OCL expressions.

To our knowledge, Weiglhofer was the first who came up with the idea of using an *ioco* checker(input/output conformance, [18]) for mutation testing. Weiglhofer tested against Lotos specifications [4]. We apply the technique to UML state machines, which, to our knowledge, has not been done before. In fact, mutation testing of UML state machines has been a topic before but in terms of model validation [10]. [8] presents results for testing state charts by state-based testing of classes. Classical mutation testing is used for an evaluation of the approaches.

A complementary approach to mitigate the lack of completeness of testing is runtime verification (see e.g. <http://runtime-verification.org>). It aims at checking the final system against correctness properties (assertions, pre/post-conditions, contracts etc.) while running. If a conflict is detected, the system can react and execute a recovery strategy or a safe termination procedure.

Such features help to improve fault-tolerance and reliability, such as conventional (and presumably more expensive) techniques like redundancy and diversity serve for. While the used formal representation can be verified, the recovery functionality itself again needs to be tested. Further, in safety- and/or time-critical applications, it may be crucial that requirement violations simply do not occur.

### 3 Fault-Based Test Case Generation (FBTCG) from UML in MOGENTES

#### 3.1 Terms and Concepts

A *mutation* is a syntactically correct modification of some code or model, such that the modified artifact remains executable (interpretable).

A *fault model* describes the kind of mutation (*mutation operator*), and may contain additional information such as parameters – e.g. the replacing item, possible application locations (within the artifact) and related semantics.

They of course depend on the specification language where they are applied. For instance, for UML state machines following fault models:

In general, small local mutations are used, based on two assumptions:

1. competent programmers do not make big mistakes, but are not immune against typos or small faults,
2. there exists a coupling effect so that complex errors will be found by test cases that were designed to detect simple errors.

**Table 1.** Examples of fault models for state diagrams

<i>Fault model</i>	<i>Parameter</i>	<i>Typical Meaning and result</i>
Replacing trigger event by another one	New trigger event	Confusion of triggers – transition is activated by another event
Setting transition guard to TRUE	--	Equivalent to forgetting a guard condition - transition will always execute on trigger event
Setting transition guard to FALSE	--	Equivalent to forgetting a transition - transition will never be executed
Aiming transition at another state	New target state	Confusion of transition result – trigger event will lead to another (system) state

In principle, fault models can be applied to all locations in the artifact where the respective mutation is possible. For instance, the first, third and last one in table 1 can be applied to all transitions in a state graph, while the second one can only be applied to transitions with a guard.

### 3.2 Test Case Generation Steps

It was decided to use UML state diagrams for representing use case requirements for several reasons: (a) one objective of MOGENTES was to generate test cases for assessing fulfillment of requirements (rather than for a specific source code coverage). A motivation is that the source code was not available or the use case providing partners were not in the position to provide the source code to the research partners. (b) State diagrams were well suited to represent most of the requirements. (c) For UML, not only many (free) tools are at hand, but also it is increasingly used by the industrial partners. (d) Generating test cases from requirements automatically delivers also the test oracles, i.e. the means to decide if the results match the expected results.

In short, following steps are needed for fault-based generation of test cases from UML models:

1. Definition of UML classes and their relationships, which denote basic structural aspects given in the requirements.  
Definition of the state diagrams for these classes, specifying the behavioral aspects defined in the requirements. For this, we used also OCL<sup>3</sup> for specifying transition guards. The resulting model is considered as the *original*.
2. Generate the mutants, by applying the fault models (mutation operations) at all possible locations with additional selection of parameters. In MOGENTES, this was done while transforming the UML model to object-oriented action systems (OOAS, [6]) and subsequently into Back's action systems (AS, [5]). This transformation is done because the semantics of action systems is fully formally defined, which is not the case for UML models. Furthermore, actions systems are well suited for the next steps.

<sup>3</sup> Object Constraint Language, <http://www.omg.org/spec/OCL/2.0/>

3. For each mutant, a conformance check with the original model is performed. This is done by exploring the behaviors of the original and the mutated action systems. In this way, two Labeled Transition Systems (LTSs) are created. Then, the synchronous product modulo *ioco* of these two LTSs is calculated. If non-conformance is found, test case generation starts. The resulting test cases are able to detect whether the modeled fault (mutant) has been implemented. The used conformance relation is input output conformance (*ioco*, [18]). The result of this conformance check is a *product graph*. It contains a set of *fail states* representing non-conformance. For further details see [7].
4. To extract test cases from this product graph, a search for paths leading to non-conformance, i.e. a fail state, is performed. The resulting test case is again an LTS. Basically, it consists of *the inputs and outputs* along this path, together with the *final step from the original*. Several different test case extraction strategies have been implemented and compared. Some aim to reduce the resulting test suites. For further details on test case extraction see [1]. Within MOGENTES, test cases are output in a so-called *abstract test case* (ATC) format.
5. The selected test cases (e.g. minimal set finding all mutants) are transformed into a format suitable for the respective use case.

### 3.3 The MOGENTES UML/OOAS Tool Chain

To carry out the steps outlined before, following tools have been used or developed.

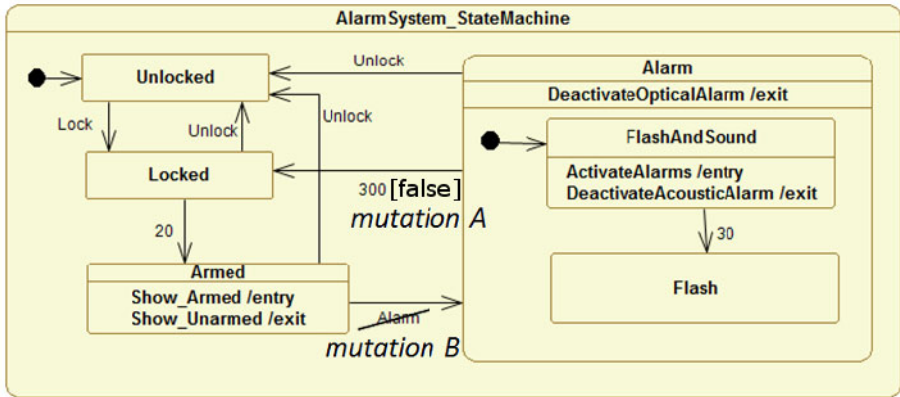
1. UML models are developed using *Papyrus* (v1.1).
2. For mutation and UML $\rightarrow$ OOAS transformation, the tool *Ummu* was developed using VIATRA2 [19] in the *Eclipse* framework.
3. For translating OOAS into non-object-oriented action systems, the tool *Argos* was implemented.
4. For exploring the action systems, check for their conformance, and test case extraction from the product graph, the tool *Ulysses* was developed.
5. To test and simulate at model level, several “model animators” were developed.
6. For each use case, a transformation tool from ATC format to the test case format used by the respective test environment was developed.

## 4 Example Use Case - Car Alarm System (CAS)

This use case from the industrial partner Ford Forschungszentrum Aachen (FFA) refers to a conventional intrusion and anti-theft alarm system for cars. The following three requirements describe the system’s behavior:

1. The system is armed 20 seconds after the vehicle is locked and the bonnet, luggage compartment and all doors are closed.
2. The alarm sounds for 30 seconds if an unauthorized person opens the door, the luggage compartment or the bonnet. The hazard flasher lights will flash for five minutes.
3. The anti-theft alarm system can be deactivated at any time – even when the alarm is sounding – by unlocking the vehicle from outside.

Of course, in practice more requirements have to be considered, e.g. for handling sensor faults, but the three requirements above specify the core functionality. Due to this small number of (though not atomic) requirements which can be well modeled by state machines, it was chosen as basic use case.



**Fig. 1.** State diagram of a simplified car alarm system, with two possible mutations: (A) setting the guard for transition *Alarm*→*Locked* to false and (B) removing the trigger on signal *Alarm* from transition *Armed*→*Alarm*

Modeling is an iterative process. This was also the case for the CAS. Figure 1 shows a late version of its state machine. Note the two example mutations: one blocking a transition (A) and one removing a signal trigger (B).

The test case for A (see Table 2) has several steps and traverses a large part of this state machine. Thereby it also kills mutations in several other places. To show that the

**Table 2.** Test cases for mutations A and B from Fig. 1. *ctr* denotes controllable actions i.e. inputs to the system, *obs* denotes observable actions i.e. system outputs; *obs delta* denotes quiescence

Step	Mutation A	Mutation B
1	ctr Lock	ctr Lock
2	wait 20	wait 20
3	obs Armed	obs Armed
4	ctr Alarm	obs delta
5	obs Unarmed	pass
6	obs OpticalAlarm_On	
7	obs AcousticAlarm_On	
8	wait 30	
9	obs AcousticAlarm_Off	
10	wait 270	
11	obs OpticalAlarm_Off	
12	Pass	



transition *Alarm*→*Locked* is actually available, the system is steered to this transition and the test case passes if the output *OpticalAlarm\_Off* resulting from this transition can be observed. The test case for B (see Table 2) on the other hand is rather short, but ends with a *delta* observation before the pass verdict. The *delta* observation represents quiescence, which means there is no further output from the system without a preceding input. To verify that the transition from *Armed* to *Alarm* does not take place without receiving an *Alarm* signal, the tester needs to check if there are no further observables. Obviously, in reality this can only be shown within a time bound which has to be defined for each application individually.

## 5 Results and Discussion

This chapter presents results of generating test cases for the CAS. It also reports some experiences with the other demonstrators of the MOGENTES project, in particular with respect to modeling and the selection of faults. Relevant points are illustrated by means of the CAS.

### 5.1 Test Case Generation

For an earlier, more complex CAS model shown in Fig. 2 with 11 states and 21 transitions, 111 mutants were generated, using 15 applicable fault models. For these 111 mutants, 152 test cases were generated, using the *minimal search depth* strategy; hence, for each mutant at least one test case was generated, but it was also checked whether a test case would find also other mutants. Following observations can be summarized:

- The strongest test case covers (detects) 75% of all mutants, while the weakest covers only one mutant.
- But this single test case is needed, because it detects a mutant not “killed” by any other test case! (It ends with quiescence, similar to mutation B above.)
- Some hierarchical relationships among mutation operators concerning coverage can be observed, but there is no strict containment relation (i.e. that the test cases from one mutation operator always kill all the mutants from another mutation operator in the same location).

Of course, all of these test cases together find all mutants, but establish a very inefficient test suite. It was therefore of interest to find a minimal test suite for all 111 mutants. However, several “minimality criteria” can be imagined, e.g. (a) smallest number of test cases, (b) minimum number of I/O steps, (c) shortest testing time as given by included time-outs. For (a), a set of 8 test cases could be found (of course containing that test case mentioned above, which finds only one mutant).

For the other (larger) use cases of the MOGENTES project, a similar coverage comparison between test cases generated for each mutant would have been computationally too expensive, both because of the TCG effort itself (as shortly addressed in section 6) and because of the effort of checking thousands of possible test cases against multiple hundreds of mutants. To cope with the TCG effort problem, the models were simplified or decomposed. For instance, for one use case, it was possible to

separate an initial device address negotiation phase from the operating mode simply by limiting the allowed input values. (This is supported by *ioco*, because it permits the use of partial specifications, which may accept fewer inputs than the implementation.) Another optimization step was to check whether already computed test cases detect a new mutant, and only if this was not the case a new test case was computed. This does not necessarily lead to an optimized set, but saves days or weeks of computing time. Following table illustrates the different levels of complexity of the models.

## 5.2 Source Code Coverage with Model-Based Test Case Generation

In general, black and white box testing complement each other – correctly applying both is usually better than using only one. Black box testing as done in this case aims at coverage of the requirements.

Evaluating source code coverage can be done by instrumenting the source code before applying the test cases. For this, an emulation of the system can be used; there is usually no need to check this on the target platform. Parts which remain uncovered should be scrutinized: when a sufficiently chosen set of mutation operators has been applied; there is a high probability that these uncovered parts realize non-required functionality.

In case a test case generator for achieving a certain level of source code coverage is available, the model can be used to verify the system outputs (test oracle). Since test case generation for classical code coverage tends to be cheaper than mutation-based methods, these test cases can also serve for seeding the mutation-based test case generation. Already killed mutants can be left out, thereby reducing the computation effort.

## 5.3 Modeling Experiences

As already argued, it was decided to develop models representing the use case requirements, in particular UML state diagrams, and to generate test cases which find faults in these models. This strategy has been chosen, because it allows a more direct assessment of conformance to requirements than deriving test cases from the source code (which is also not always available). Since a mutated model represents a correspondingly wrong interpretation (or implementation) of the affected requirement(s), it can be expected that the generated test cases detect corresponding deviations from a correct implementation of the requirements. It can be summarized that these expectations have been fulfilled, and therefore the taken approach has been proved as appropriate for the generation of strong and efficient test suites. However, there are a number of subtle aspects to consider in the specification of models for representing requirements. Some interesting ones are discussed in the following.

**Model Verification.** In most cases, requirements are given in natural language, which often leads to ambiguities and misinterpretation. Actually, verifying conformance of models with requirements can be expensive. In MOGENTES, manual inspection was considered to be sufficient, mainly because assuring model correctness was not a main concern. Further, the resulting test cases often uncover discrepancies between the requirements and the model.

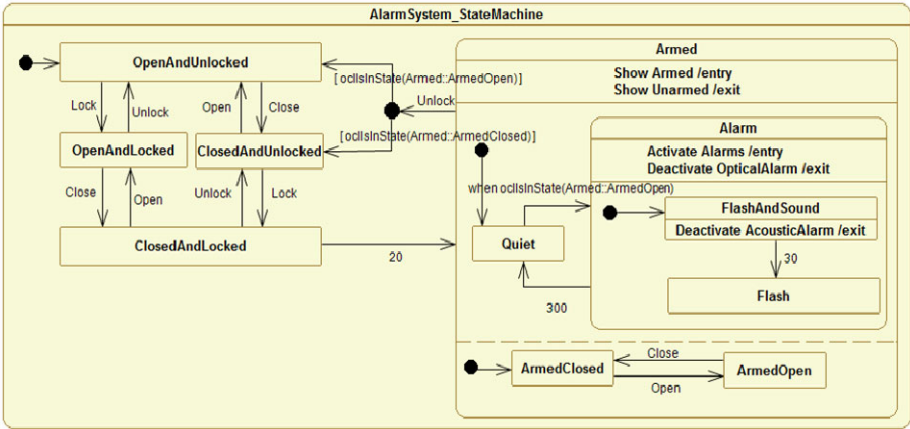


Fig. 2. Earlier CAS model with *Open/Close* and *Lock* as separate inputs

For instance, requirement 1 for the CAS was first interpreted such that “lock” and “close” were considered as two inputs, resulting in the model given in Fig. 2, i.e. it was first understood that the car can be open while being locked without raising an alarm. After clearing up this misunderstanding, the corrected model as shown in Fig. 1 resulted. (*Close* is now a precondition for *Lock*, treated outside of the model.)

*Conclusion:* deviations of (test) models from requirements can cause strange test cases which lead to detection of the modeling faults, but it does not replace conformance verification between model and requirements.

**Too Detailed Models / Underspecified Requirements.** A related situation occurs when the modeler decides on details not given in the requirements, either because they were forgotten to be specified, or by purpose, because the respective aspect is left open for implementation.

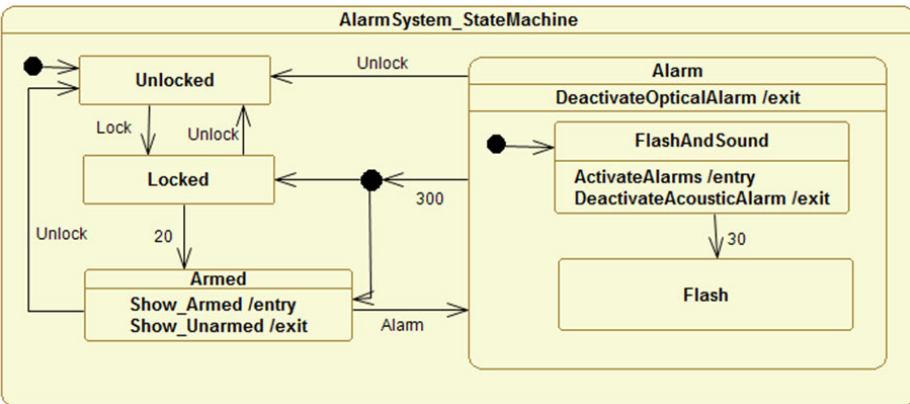


Fig. 3. CAS state diagram with non-deterministic state transition

For instance, from the CAS requirements it was unclear whether the AlarmArmed output stays high during the alarm or not, as well as whether after the alarms turned off, the system is in state armed or just locked. Making decisions on such aspects during modeling would either cause the model to deviate from the requirements (under-specification) or restrict the modeled behavior unnecessarily (implementation freedom).

In principle, non-determinism can be exploited for deliberately unspecified aspects. For instance, the model variant in Fig. 3 **Fehler! Verweisquelle konnte nicht gefunden werden.** contains a non-deterministic transition from Alarm exit (after 300 time units) to either Locked or Armed. (It should be noted that the used notation is not common but allowed by the UML 2.1 specification.)

An example for under-specification is that the requirements do not explicitly disallow moving from the *Unlocked* state to the *Alarm* state on an *Alarm* input signal, which would actually be *ioco* conform. To avoid this, an *Alarm* input to the state Unlocked can be added, as indicated in Fig. 4.

*Conclusion:* clarify under-specifications with the user. If they are by purpose, use non-determinism if possible to explicitly allow the behavior, otherwise, clarify and complete the requirements before continuing with the test model.

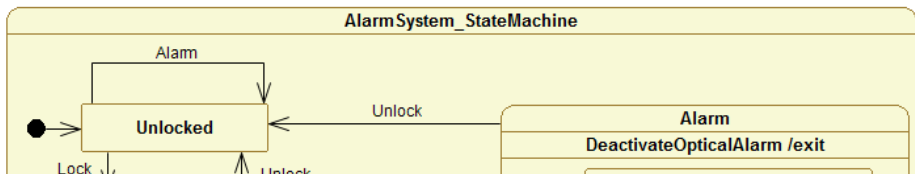


Fig. 4. Addition of “alarm” input to Unlocked state of CAS model

**“Too Elegant” Models (“Over-factoring”).** There is a risk that by using more sophisticated elements of the modeling language or simply by following good style, modelers, will make the models more “elegant” and compact than useful for mutation testing. good style. This often includes bringing information to one point in order to avoid redundancy (“factoring”).

In case of the alarm system, already the use of exit actions and nested states leads to an interesting situation: The exit action of state *Alarm*, i.e. deactivation of the optical alarm, occurs in three situations: when the optical alarm times out (transition *Alarm*→*Locked*), when the car is unlocked while being in state *FlashAndSound*, and when the car is unlocked while being in state *Flash*. An implementation for some reasons might handle the latter two with different code parts and miss to deactivate the optical alarm when leaving state *Flash*. But a mutation removing the exit action would be already found by unlocking the car in state *FlashAndSound*. In this case, there is no mutation forcing a test case for the third situation.

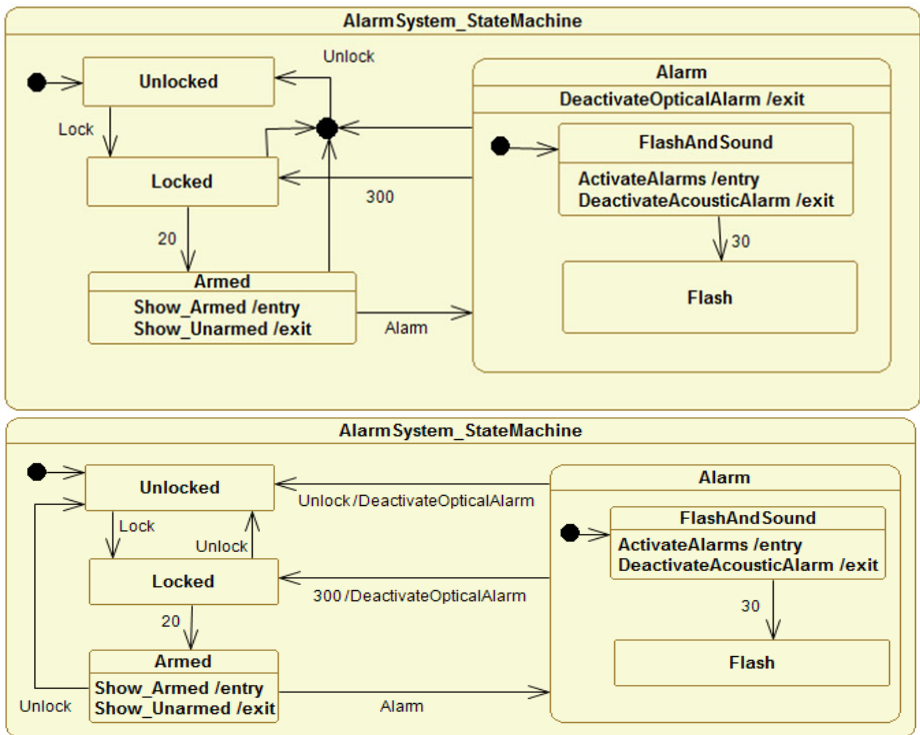
*Conclusion:* requirements should be specified as atomic as possible, and be represented 1:1 in models. Since forcing a modeler to avoid reuse and powerful modeling elements severely reduces the readability of models as well as other advantages of modeling itself, this is only a partial solution. Instead, de-factoring could either be integrated into the fault model application or the model could be transformed into a

canonical intermediate model, making sophisticated “short notations” explicit. In the first case, for instance, the fault model for an exit action mutation could include parameters selecting the outgoing edges where the mutation shall be applied. In the second case, constructs as exit and entry actions as well as transitions from nested states would be replaced.

### 5.4 Test Case Application - Adaptive Test Cases

Due to the inherently concurrent nature of UML, transitions in parallel regions of a state machine – in multiple instances of the same class or in instances of different classes – might occur in arbitrary sequences. Therefore, also the implementation should be allowed to do it one way or the other. This as well as above mentioned intentional non-determinism can result in adaptive test cases, where the next input depends on a former, non-deterministic, output of the system under test. However, in MOGENTES most test environments from the industrial partners were not able to cope with adaptive test cases without substantial modifications.

Only a part of this issue can be dealt with in the transformation of “abstract” test cases to the target notation (see step 6 in 3.3).



**Fig. 5.** Example for model (over-)factoring: (top) by combining transitions sharing the same trigger, and its avoidance, (bottom) by spreading an exit action to all exiting transitions (although in this case still not introducing separate transitions from states *FlashAndSound* and *Flash* which would solve the above mentioned problem)

*Conclusion:* be clear about the capabilities of the target test environment and address this either by extending the test environment or by, e.g., avoiding concurrency and non-determinism in the model.

## 6 Summary and Outlook

In this paper, an approach for model-based mutation testing has been presented, which uses UML-state diagrams for requirements modeling. Furthermore, experiences from its application to a specific use cases from the automotive domain have been discussed. Its main benefits are automatically generated test cases, which

- include oracles (i.e. expected results) according to requirements (which is difficult when deriving test cases from source code),
- can explicitly detect faults (requirements violations) in target systems,
- fulfill conventional coverage metrics such as states, transitions, equivalence classes, or border values,

Hence, the generated test cases not only are stronger with respect to finding faults than those developed with conventional coverage metrics, but also minimize the distance from the model to the final system. Of course, only modeled faults (more precisely: mutations) will be found, but at least our approach yields a known a priori coverage of faults.

Besides the research issues mentioned in the previous section, i.e.

- selection of fault models
- good (requirements/test) model design, in particular to optimize finding implementation faults

A major issue to be addressed in future is the state space explosion during test case generation, which is partially worsened by the problem of equivalent mutants (mutations which do not change the model's behavior), and which currently limits the application of this approach to rather small or sufficiently simplified models. Strategies to be investigated include symbolic computation, model decomposition, and to exploit known techniques for model checking like partial order reduction. Of course, it is also of interest to extend the approach to other modeling techniques, e.g., to UML activity diagrams or sequence charts.

**Acknowledgement.** Besides Bernhard Aichernig, Harald Brandl, and Willibald Krenn from Graz Univ. of Technology, who contributed significantly to the ground on which the reported work was built upon, we thank our MOGENTES partners from Ford, in particular Johannes Wiessalla, for the CAS example and fruitful discussions.

## References

1. Aichernig, B.K., Brandl, H., Jöbstl, E., Krenn, W.: Efficient Mutation Killers in Action. In: Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation, ICST (March 2011) (in press)

2. Aichernig, B.K., Krenn, W.: Model-Based Generation of Test-Cases (for Embedded Systems) – State of the Art Survey. Deliverable 1.2 of EU FP7 project MOGENTES, [http://www.mogentes.eu/public/deliverables/MOGENTES\\_1-19a\\_1.1r\\_D1.2\\_Survey\\_Part-a.pdf](http://www.mogentes.eu/public/deliverables/MOGENTES_1-19a_1.1r_D1.2_Survey_Part-a.pdf)
3. Aichernig, B.K., He, J.: Mutation testing in UTP. *Journal of Formal Aspects of Computing* 21(1-2) (February 2009)
4. Aichernig, B.K., Peischl, B., Weiglhofer, M., Wotawa, F.: Protocol conformance testing a SIP registrar: An industrial application of formal methods. In: *Proceedings of the 5th IEEE International Conference on Software Engineering and Formal Methods*, pp. 215–224. IEEE, London (2007)
5. Back, R.J.R., Kurki-Suonio, F.: Distributed cooperation with action systems. *ACM Trans. Program. Lang. Syst.* 10(4), 513–554 (1988) ISSN 0164-0925
6. Bonsangue, M.M., Kok, J.N., Sere, K.: An approach to object-orientation in action systems. In: *Jeuring, J. (ed.) MPC 1998. LNCS, vol. 1422*, pp. 68–95. Springer, Heidelberg (1998)
7. Brandl, H., Weiglhofer, M., Aichernig, B.K.: Automated Conformance Verification of Hybrid Systems. In: *Proceedings of the 2010 10th International Conference on Quality Software (QSIC 2010)*, pp. 3–12. IEEE Computer Society, Los Alamitos (2010) ISBN 978-0-7695-4131-0
8. Briand, L.C., Di Penta, M., Labiche, Y.: Assessing and improving state-based class testing: a series of experiments. *IEEE Trans. Software Eng.* 30, 770–783 (2004)
9. Chaki, S., Clarke, S., Groce, A., Jha, S., Veith, H.: Formal Verification of Software Components in C. *Trans. of SW Engineering* 30(6), 388–402 (2004)
10. Fabbri, S.C.P.F., Maldonado, J.C., Sugeta, T., Masiero, P.C.: Mutation testing applied to validate specifications based on statecharts. In: *Proceedings 10th International Symposium on Software Reliability Engineering (Cat. No.PR00443)*, Boca Raton, FL, USA, pp. S.210–219
11. Fröhlich, P., Link, J.: Automated test case generation from dynamic models. In: *Hwang, J. (ed.) ECOOP 2000. LNCS, vol. 1850*, pp. 472–492. Springer, Heidelberg (2000)
12. Gnesi, S., Latella, D., Massink, M., Moruzzi, V., Pisa, I.: Formal test-case generation for uml statecharts. In: *Proc. 9th IEEE Int. Conf. on Engineering of Complex Computer Systems 2004*, pp. 75–84 (2004)
13. Jia, Y., Harman, M.: An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering* PP(99), 1
14. Myers, G.J., Sandler, C.: *The Art of Software Testing*. John Wiley & Sons, Chichester (2004) ISBN 0471469122
15. Oster, N., Saglietti, F.: Automatic test data generation by multi-objective optimisation. In: *Górski, J. (ed.) SAFECOMP 2006. LNCS, vol. 4166*, pp. 426–438. Springer, Heidelberg (2006) ISBN 3-540-45762-3
16. Seifert, D., Helke, S., Santen, T.: Test Case Generation for UML Statecharts. In: *Broy, M., Zamulin, A.V. (eds.) PSI 2003. LNCS, vol. 2890*, pp. 462–468. Springer, Heidelberg (2004)
17. Shao, Z.: Certified Software. *Comm. ACM* 53(12), 56–66 (2010)
18. Tretmans, J.: Test generation with inputs, outputs, and quiescence. In: *Margaria, T., Stefan, B. (eds.) TACAS 1996. LNCS, vol. 1055*, pp. 127–146. Springer, Heidelberg (1996)
19. VIATRA, <http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/VIATRA2>

20. Walter, D., Täubig, H., Lüth, C.: Experiences in applying formal verification in robotics. In: Schoitsch, E. (ed.) SAFECOMP 2010. LNCS, vol. 6351, pp. 347–360. Springer, Heidelberg (2010)
21. Weißleder, S., Schlingloff, B.H.: Deriving input partitions from UML models for automatic test generation. In: Giese, H. (ed.) MODELS 2008. LNCS, vol. 5002, pp. 151–163. Springer, Heidelberg (2008)
22. Yang, Q., Ma, D., Zhao, Y., Li, Z.: Towards a Formal Verification Approach for Implementation of Web Services Specifications. In: Proc. of 2010 IEEE Asia-Pacific Services Computing Conf., pp. 269–276 (December 2010)



# ISO/IEC 15504-10: Motivations for Another Safety Standard

Giuseppe Lami, Fabrizio Fabbrini, and Mario Fusani

Consiglio Nazionale delle Ricerche, Istituto di Scienza e Tecnologie dell'Informazione  
via Moruzzi, 1 – I-56124 Pisa, Italy  
{giuseppe.lami, fabrizio.fabbrini, mario.fusani}@isti.cnr.it

**Abstract.** This paper presents the new standard ISO/IEC 15504 Part 10 Safety Extension. It has been developed to extend the well-known ISO/IEC 15504 standard for process assessment and improvement in order to make consistent judgment regarding process capability or improvement priorities for safety related systems development. In order to avoid misunderstanding and confute reluctance and worry related to such a new standard, its contents, purpose and intended are explained in this paper. Moreover, comparison between the ISO/IEC 15504 Part 10 and other existing safety standards for software is provided as well as a discussion on possible integrations and consequent benefits of its usage.

**Keywords:** Software Process Assessment, Software Safety Standards.

## 1 Introduction

Addressing software-related safety issues is an ever-growing need in the industry for several reasons; the most important of them is the increasing importance of software in safety-related systems.

Addressing the software process quality as a way to increase the confidence in the quality of the resulting software product has been a key issue for three decades.

Today two principal schemes exist to evaluate software process quality and guide its improvement: CMMI [1] (Capability Maturity Model Integration) and ISO/IEC 15504 [10] standard.

While some differences between CMMI (developed by the Software Engineering Institute of the Carnegie Mellon University, Pittsburgh, PA) and ISO/IEC 15504 exist in terms of structure and intended usage, they share some basic concepts, first of all the concept of Process Capability. Process Capability is defined as the likelihood a process achieves its defined goals.

Nevertheless neither CMMI nor ISO/IEC 15504 provide a sufficient basis to perform a process capability evaluation with respect to the development of complex safety critical systems.

For this reason in 2007 a Safety Extension to CMMI-DEV called +SAFE [2] has been released. The aim was to extend CMMI to provide an explicit and specific framework for functional safety with respect to the development of complex safety-critical products. +SAFE is not specific to any safety standard: any standard that defines safety principles, methods, techniques, work products, and product assessment methods may be used to satisfy the goal of the framework as appropriate. The +SAFE framework has been conceived to be used both for appraising safety-critical products suppliers and for improving an organization's capability in developing, sustaining, maintaining, and managing safety-critical products.

Also ISO/IEC JTC1/SC7 WG10 in 2009 launched a New Work Item in order to develop a new Technical Report (to be called ISO/IEC 15504 part 10 [3]) containing additional processes and guidance to support the use of the existing exemplar process assessment models for systems and software when applied to the assessment of safety related systems development in order to make consistent judgment regarding process capability and/or improvement priorities.

At the time of writing this paper ISO/IEC 15504-10 has finally reached a rather stable version over its standardization path. ISO/IEC 15504-10 is the main object of this paper.

While the scope and intended usage of ISO/IEC 15504-10 are well defined, in the software safety community such a new standard is being accompanied by a certain reluctance and worry because it is perceived as another standard against which compliance can be claimed and then as an additional source of requirements and constraints for the industry. In order to confute such a perception, in this paper the contents of ISO/IEC 15504-10 will be described as well as its scope, purpose and intended usage. Moreover, to add even more clarity, in this paper the mutual relationships between ISO/IEC 15504-10 and other existing safety standards will be discussed.

The paper is structured as follows: in section 2 a description of the structure and content of ISO/IEC 15504-10 is provided as well as its purpose and scope. In section 3 a comparative analysis between ISO/IEC 15504-10 and the principal existing safety standards addressing software is discussed. In section 4 the integration of ISO/IEC 15504-10 with the existing safety standards for software is discussed along with the critical points to be addressed when 15504-10 will be used in practice. Finally, in section 5 conclusions are given.

## **2 ISO/IEC 15504-10 Safety Extension**

ISO/IEC 15504 *Process assessment* is a well-known standard that provides a scheme for assessing the capability of system/software processes and a way to improve them. Process capability is defined as a characterization of the ability of a process to meet current or projected business goals. ISO/IEC 15504 provides a general framework in which assessments can take place.

However the ISO/IEC 15504 process assessment models for systems and software do not currently provide a sufficient basis for performing a process capability assessment of processes involved in the development of complex, safety critical systems. Even the Automotive SPICE standard [11, 12] (a standard developed as a tailored version of the ISO/IEC 15504 for the automotive domain) provides little support for its use in a safety-related context.

In this section, we first provide a description of the overall ISO/IEC 15504 structure as well as its underlying principal concepts, then describe in more detail the ISO/IEC 15504-10 structure and contents.

## 2.1 ISO/IEC 15504: Overview

It is not the aim of this section to extend our description to the different parts of the standard; our aim is to give the reader understand the basic concepts underlying the standard.

The purpose of ISO/IEC 15504 is to provide a scheme for evaluating the capability of the system/software process and a way to improve them.

The three fundamental concepts of the standard are: the Process Reference Model (PRM), the Process Assessment Model (PAM) and the Measurement Framework.

**PRM:** it is a model comprising the definition of the processes in a lifecycle described in terms of “process purpose” and “process outcomes”, together with an architecture describing the main relationships between processes. In other words, the PRM is the set of the descriptions of the processes to be assessed. The standard doesn’t include any specific PRM but it defines the requirements for defining a PRM. These requirements are described in ISO/IEC 15504 – Part 2.

**PAM:** it is a model suitable for the purpose of assessing process capability, based on one or more PRMs. The PAM provides a two-dimensional view of process capability: in one dimension, it describes a set of process entities that relate to the processes defined in the specific PRM (it is called Process Dimension); in the other dimension the PAM describes capabilities that relate to the process capability levels and process attributes according to the Measurement Framework defined in the standard.

**Measurement Framework:** it provides a schema for use in characterizing the capability of an implemented process with respect to the PAM. Capability is defined on a six-value ordinal scale. The scale represents increasing capability of the implemented process (starting from level 0: incomplete process, to level 5: optimizing process). In table 1 the description of the six Capability Levels within the ISO/IEC 15504 Measurement Framework is provided, along with the associated Process Attributes to be fulfilled to achieve a certain Capability Level. For the description of the meaning of each capability level, refer to [10]. The achievement of a certain capability level is established by the rating of specific Process Attributes (i.e. measurable characteristics of process capability applicable to any process). The measurement of the Process Attribute uses a four-value rating scale.

The three basic concepts described above along with their relationships are graphically shown in figure 1.

SPICE (Software Process Improvement and Capability dEtermination) is the acronym used to identify a major international initiative launched in early 90's to support the development of the new International Standard ISO/IEC 15504 for (Software) Process Assessment. The SPICE project was set up under the auspices of the Subcommittee 7 (Software Engineering) of Joint Technical Committee between ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) through its Working Group on Process Assessment (WG10).

The project had three principal goals:

- to develop a working draft for a standard for software process assessment,
- to conduct industry trials of the emerging standard,
- to promote the technology transfer of software process assessment into the software industry worldwide.

The first two goals having been satisfyingly accomplished [Rout 2007], the SPICE activity has continued with the implementation of more normative and informative parts (including Parts 5 and 6, that present exemplar PRMs referring to the ISO/IEC 12207 and ISO/IEC 15288 standards), up to the Part 10 that is discussed this paper.

**Table 1.** Capability Levels description

<b>Capability Level 0 Incomplete Process</b>	<i>The process is not implemented, or fails to achieve its process purpose.</i>	
<b>Capability Level 1 Performed Process</b>	<i>The implemented process achieves its process purpose</i>	
	<b>Process Attribute 1.1</b>	Process performance
<b>Capability Level 2 Managed Process</b>	<i>The previously described Performed process is now implemented in a managed fashion (planned, monitored and adjusted) and its work products are appropriately established, controlled and maintained.</i>	
	<b>Process Attribute 2.1</b>	Performance management
	<b>Process Attribute 2.2</b>	Work product management
<b>Capability Level 3 Established Process</b>	<i>The previously described Managed process is now implemented using a defined process that is capable of achieving its process outcomes.</i>	
	<b>Process Attribute 3.1</b>	Process definition
	<b>Process Attribute 3.2</b>	Process deployment
<b>Capability Level 4 Predictable Process</b>	<i>The previously described Established process now operates within defined limits to achieve its process outcomes</i>	
	<b>Process Attribute 4.1</b>	Process measurement
	<b>Process Attribute 4.2</b>	Process control
<b>Capability Level 5 Optimizing Process</b>	<i>The previously described Predictable process is continuously improved to meet relevant current and projected business goals</i>	
	<b>Process Attribute 5.1</b>	Process innovation
	<b>Process Attribute 5.2</b>	Continuous optimization

SPICE and ISO/IEC 15504 are often used as synonyms in the common understanding even though they are different.

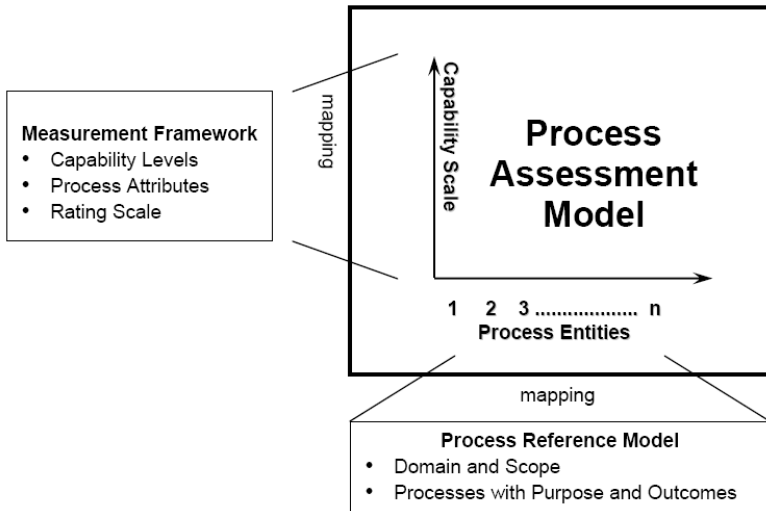


Fig. 1. ISO/IEC 15504: basic concepts

## 2.2 ISO/IEC 15504-10: Overview

In order to make consistent judgment regarding process capability or improvement priorities, additional guidance and processes are needed in any ISO/IEC 15504 PRM and PAM. With this intention, in 2009, a New Work Item proposal was submitted to ISO/IEC JTC1/SC7 Secretariat with the aim of developing a safety extension to ISO/IEC 15504.

At the date this paper is written, ISO/IEC 15504-10 *Safety Extension* (currently a DTR – Draft Technical Report) has finally reached a stable version over its standardization path.

The DTR is composed of two principal parts: Process definitions and Lifecycle guidance.

### 2.2.1 ISO/IEC 15504-10: Process Definitions

Three processes have been defined in clause 4 of the ISO/IEC 15504-10 DTR. These are the Safety Management process, the Safety Engineering process and the Safety Qualification process. The Process definition complies with the ISO/IEC 15504 standard style where each process definition is composed of:

- Process Name and Id.
- Process Purpose
- Process Outcomes

The processes defined in ISO/IEC 15504-10 are reported below in tabular format:

<b>Process ID</b>	SAF.1
<b>Process Name</b>	Safety Management
<b>Process Purpose</b>	The purpose of the Safety Management Process is to ensure that products, services and life cycle processes meet safety objectives.
<b>Process Outcomes</b>	As a result of the successful implementation of the Safety Management process: <ol style="list-style-type: none"> <li>1) Safety principles and safety criteria are established.</li> <li>2) The scope of the safety activities for the project is defined.</li> <li>3) Safety activities are planned and implemented.</li> <li>4) Tasks and resources necessary to complete the safety activities are sized and estimated.</li> <li>5) Safety organization structure (responsibilities, roles, reporting channels, interfaces with other projects or OUs ...) is established.</li> <li>6) Safety activities are monitored, safety-related incidents are reported, analyzed, and resolved.</li> <li>7) Agreement on safety policy and requirements for supplied products or services is achieved.</li> <li>8) Supplier's safety activities are monitored.</li> </ol>

<b>Process ID</b>	SAF.2
<b>Process Name</b>	Safety Engineering
<b>Process Purpose</b>	The purpose of the Safety Engineering process is to ensure that safety is adequately addressed throughout all stages of the engineering processes.
<b>Process Outcomes</b>	As a result of the successful implementation of the Safety Engineering process: <ol style="list-style-type: none"> <li>1) Hazards related to product are identified and analyzed.</li> <li>2) Hazard log is established and maintained.</li> <li>3) Safety demonstration for the product lifecycle is established and maintained.</li> <li>4) Safety requirements are defined.</li> <li>5) Safety integrity requirements are defined and allocated.</li> <li>6) Safety principles are applied to development processes.</li> <li>7) Impacts on safety of change requests are analyzed.</li> <li>8) Product is validated against safety requirements.</li> <li>9) Independent evaluations are performed.</li> </ol>

<b>Process ID</b>	SAF.3
<b>Process Name</b>	Safety Qualification
<b>Process Purpose</b>	The purpose of the Safety Qualification process is to assess the suitability of external resources when developing a safety-related software or system.
<b>Process Outcomes</b>	As a result of the successful implementation of the Safety Qualification process: <ol style="list-style-type: none"> <li>1) Safety qualification strategy for external resources is developed.</li> <li>2) Safety qualification plan is developed and executed.</li> <li>3) Safety qualification documentation is written.</li> <li>4) Safety qualification report is produced.</li> </ol>

### 2.2.2 ISO/IEC 15504-10 Safety Extension: Lifecycle Guidance

Clause 5 of ISO/IEC 15504-10 DTR provides guidance for assessors in considering specific safety-related aspects when a process assessment is performed in an environment where safety-related software/systems are developed. Guidance is provided in tabular format, and gives the assessors, for each process contained in ISO/IEC 15504-5 and ISO/IEC 15504-6 (i.e. those parts of the ISO/IEC 15504 standard where software and system related processes are respectively identified), an indication of the additional issues to be taken into account at assessment time. The issues are provided by means of sentences indicating specific relationships between the processes in ISO/IEC 15504-5 and ISO/IEC 15504-6 and those in ISO/IEC 15504-10. Moreover, relevant aspects to be considered to improve the completeness of the data-gathering phase of the assessment are highlighted.

## 3 Comparison of ISO/IEC 15504-10 with the IEC 61508 Family Standards

ISO/IEC 15504-10 *Safety Extension* differs from existing safety standards for software in terms of purpose, scope and intended usage. These differences are presented and discussed in this section.

Different safety standards for software-intensive systems exist. They differ from each other principally on the basis the intended application domain they are to be applied to.

The well-known standard IEC 61508 [4] addresses functional safety of electrical/electronic/programmable electronic safety-related systems. It provides general requirements for specific lifecycle phases of such systems, including those regarding the development of the software parts and their integration in the system. These requirements, mainly technical although not excluding management, come from the instantiation of engineering processes in a typical V-like system lifecycle [13, 14, 15]. The concept of processes as re-usable and improvable sets of practices, on which both SPICE and CMMI are grounded, is missing in IEC 61508 and in its derived standards. On the other hand, SPICE and CMMI have no concern on how

practices and techniques are implemented in a system lifecycle. In this sense, these two categories of standards - the processes-oriented one and the lifecycle-oriented one - are complementary to each other.

Yet, due to the fact that, in practice, the final product of lifecycle processes is a defined system, the lifecycle evidence to be compared against a specific SPICE PRM and those to be compliant with IEC 61508 or derivatives (for a product generated by executing the same processes) may partially overlap. This fact is causing a debate among the interested parties, but the complementarity cannot be denied and solutions to this problem are likely to be found. The issue involves economic aspects and is interesting indeed, but is out of the scope of this paper.

As mentioned, other standards have been derived from IEC 61508 according to the application domain (e.g. railways, automotive, medical) they refer to. In particular, the more recent of them is ISO 26262 [5] that addresses functional safety in the automotive domain. In this standard, basically another lifecycle-oriented one, some lifecycle processes have been defined, possibly partially mappable to the ones defined in ISO/IEC 12207 and ISO/IEC 15288,

IEC 61508 and its derived domain-specific safety standards are a representative sample of safety standards used in software development. For this reason, in this section, they are compared with ISO/IEC 15504-10 in order to point out the differences and possible commonalities.

For clarity reasons we consider the following issues to compare: *intended use (i.e. the way it should be used), purpose, scope, reference domain, clauses.*

*Intended use:* The IEC 61508 family standards, each of them in its application domains, are used as reference software life cycles as well as requirement schemes against which compliance can be claimed and required.

The context in which ISO/IEC 15504-10 should be used is different.

It has been developed in order to give the ISO/IEC 15504 assessors and improves the possibility to perform compliant assessment and improvement initiative also in contexts where safety-relevant systems are developed. In fact, because developing safety-critical systems requires specialized processes, techniques, skills and experience, process amplifications are needed in the area of safety management, safety engineering and the selection and qualification of software tools and libraries. Moreover, additional informative components concerning additional lifecycle verification activities related to the methods and techniques selected for the safety integrity levels are needed too.

The Safety Extension, developed as a standalone document, has been conceived to be used in conjunction with the Part 5 and/or Part 6 process assessment models by experienced assessors with minimal support from safety domain experts.

The Safety Extension has been developed independent of any specific safety standards that define safety principles, methods, techniques and work products, however elements of relevant safety standards will be able to be mapped to the Safety Extensions and the Safety Extensions will be extendable to be able to include specific safety standards requirements.

The Safety Extension does not include a glossary of new terms such as hazard, FMEA, safety argument, safety incident etc. Such terms will be defined with reference to existing source materials with commonly accepted terms and usage.



*Purpose:* The common characteristic of the IEC 61508 family standards is that they are conceived to provide a scheme that, on the basis of a defined integrity level of the system under development, set requirements for the development project and the organization that carries out it.

The purpose of the ISO/IEC 15504-10, on the contrary, is to provide a way to measure the capability of safety-related processes (Safety Management, Safety Engineering and Safety Qualification) as well as a scheme for their improvement. These facts are very important in order to avoid misinterpretations about the role of the ISO/IEC 15504-10.

*Scope:* The scope of application of the IEC 61508 family standards includes the technical and managerial activities of a development project. The scope of the ISO/IEC 15504-10 is instead limited to the SAF.1, SAF.2 and SAF.3 processes.

*Application domain:* The IEC 61508 is a generic standard not related to any specific application domain. The standards belonging to its family are typically developed with the aim to be specific for a particular application domain. In fact, for instance, the ISO 26262 [5] is for the automotive domain, the IEC 60880 [9, 16] for nuclear power, the EN 50128/9 [8] for rail transport, the IEC 60601 [6] for medical electrical equipment.

The ISO/IEC 15504-10 is not intended for any specific application domain.

*Clauses:* The clauses contained in the IEC 61508 family standards can be, according to the defined Integrity level, mandatory or recommended. They are then to be considered as technical constraints for a project under development.

On the contrary ISO/IEC 15504-10 does not prescribe any specific technique or method, it enlarges the scope in which the ISO/IEC 15504 can be used maintaining the original characteristic of considering the process (the “what”) and not requiring the adoption of any specific technique or method (the “how”).

## **4 Integrating ISO/IEC 15504-10 with Existing Safety Standards**

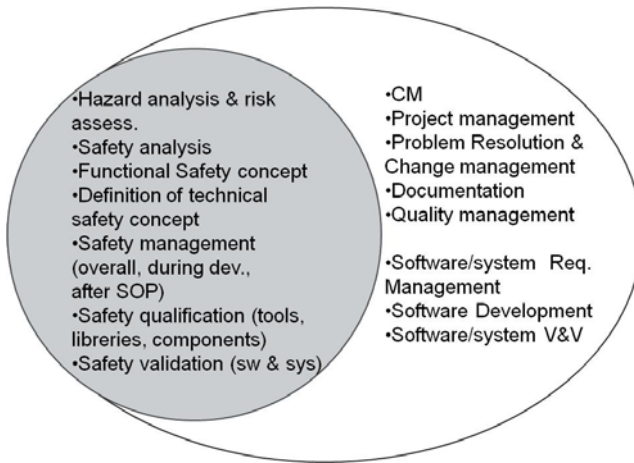
In this section the way ISO/IEC 15504-10 may be integrated with other existing domain-specific safety standards belonging to the IEC 61508 family is discussed.

The main activities in the scope of the IEC 61508 family of standards can be divided into two groups: Safety related activities, Technical and Project Management activities [4, 13].

Safety related activities are: Hazard analysis & risk assessment, Safety analysis, Functional Safety concept, Definition of technical safety concept, Safety management (overall, during development, after SOP), Safety qualification (tools, libraries, components), Safety validation (for software and system).

Technical and project management activities are: Configuration Management (CM), Project management, Problem Resolution & Change management, Documentation, Quality management, Software/System Requirements Management, Software Development, Software/system Validation and Verification.

As a first step the activities covered by both IEC 61508 and ISO/IEC 15504-10 are identified.



**Fig. 2.** Overlapping between IEC 61508 and ISO/IEC 15504-10 processes-related activities

In figure 2 the overlapping area is represented in the grey circle.

To complete the discussion about the integration between IEC 61508 and ISO/IEC 15504-10 we may refer to the two typical scenarios in which the two standards could be integrated.

*Scenario 1: Compliance with one of the IEC 61508 family standards achieved*

In this case, from the achievement of the compliance with an IEC 61508 family standard the achievement of Capability Level 2 (Managed) of the processes in ISO/IEC 15504-10 can be inferred.

In fact, the accomplishment of the clauses of the IEC 61508 family standard guarantees the performance of the three processes in ISO/IEC 15504-10 (then the fulfillment of the Process Attribute 1.1) as well as a sufficient degree of management of the project performance (Process Attribute 2.1) and Work Product management (Process Attribute 2.2) related to those three processes.

*Scenario 2: Capability level 2/3 achieved for the processes in ISO/IEC 15504-10*

In this case, achieving Capability Level 2 or 3 guarantees that the processes in ISO/IEC 15504-10 are performed, managed and (if Capability Level 3 is achieved) defined at the organizational level and applied (possibly with some tailoring). Unfortunately, since ISO/IEC 15504-10 addresses the processes (i.e. it says what to do but not how) there is no guarantee that the clauses of the IEC 61508 family standard are satisfied by the way the processes of ISO/IEC 15504-10 are performed.

Finally, in order to complete the analysis of the possible integration between the IEC 61508 family standards and ISO/IEC 15504-10, the following question should be answered: *Since compliance with a safety standard for software requires effort, why should my organization spend additional effort for ISO/IEC 15504-10?*

The answer should be based on the following considerations:

ISO/IEC 15504-10 allows safety processes to be assessed in terms of process capability. This offers a new opportunity to improve those processes. In fact, the

benefits derived from the application of ISO/IEC 15504 in terms of process improvement can be extended to safety processes. Improving capability means obtaining a more efficient and repeatable development process; reducing the risk of missing project goals (and then to maintain projects within schedule, cost and quality estimations), and identifying improvement areas in the organization. In other words, even though the compliance with a safety standard for software can be achieved independently of the capability level of processes, on the other hand improving the capability of processes allows an organization to enhance the way such a compliance can be obtained and maintained by reducing project risks and long term costs.

Moreover, adopting the ISO/IEC scheme allows for a measurement scheme for process capability that can be used to claim the achievement of a certain capability level for the key processes, as those dealing with safety. Such a possibility allows for the establishment of benchmarking mechanisms.

## 5 Conclusions

ISO/IEC 15504 is a well-known standard that provides a scheme for assessing the capability of the software processes and a way to improve them. Process capability is defined as a characterization of the ability of a process to meet current or projected business goals.

The published ISO/IEC 15504 process assessment models for systems and software do not currently provide a sufficient basis for performing a process capability assessment of processes with respect to the development of complex safety critical systems. For this reason a Safety Extension of the ISO/IEC 15504 is going to be issued under the name of ISO/IEC 15504 Part 10 Safety Extension (ISO/IEC 15504-10). ISO/IEC 15504-10 differs from the existing safety standards for software in terms of purpose, scope and intended usage. These differences have been presented and discussed in this paper. In particular, unlike the existing safety standards (for instance IEC 61508, IEC 60880, EN 50128 and ISO 26262) ISO/IEC 15504-10 Safety Extension does not provide a software life cycle and has not been conceived as a certification scheme with which compliance can be claimed. On the contrary, it does provide a way to measure the capability of safety-related processes (Safety Management, Safety Engineering and Safety Qualification) as well as a scheme for their improvement. These points are very important in order to avoid misinterpretations of the role of ISO/IEC 15504-10.

Moreover, in this paper, the relationships between ISO/IEC 15504-10 and IEC 61508 (and the standards derived from it) have been discussed. The advantages of the integration of these two standards have been identified as well as the mutual support they can provide to each other: in particular, their integration allows an organization aiming at being compliant with a IEC 61508 family standard to extend the possibility to assess and improve process capability also for safety related process.

## References

1. Chrissis, M.B., Konrad, M., Shrum, S.: CMMI Guidelines for Process Integration and Product Improvement. Addison-Wesley, Reading (2004)
2. Software Engineering Institute "+SAFE, V1.2 A Safety Extension to CMMI-DEV, V1.2. Technical Note CMU/SEI-2007-TN006 (2007)

3. International Organization for Standardization. ISO/IEC 15504 International Standard Information Technology – Software Process Assessment- Part 10: Safety Extension ISO/IEC DTR 15504-10 (2010)
4. International Electrotechnical Commission Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems (IEC 61508) (2005)
5. International Standardization Organization Road Vehicles – Functional Safety (ISO/DIS 26262) (2010)
6. International Electrotechnical Commission Medical Electrical Equipment – Part 1: General requirements for safety and essential performance (IEC 60601-1) (2005)
7. CENELEC EN 50128 Railway application – Communications, signaling and processing systems – Software for railway control and protection systems (2001)
8. CENELEC EN 50129 Railway application – Communications, signaling and processing systems – Safety related electronic systems for signaling (2003)
9. International Electrotechnical Commission Nuclear power plants – Instrumentation and control systems important to safety- software aspects for computer-based systems performing category A functions (IEC 60880) (2006)
10. International Organization for Standardization. ISO/IEC 15504 International Standard Information Technology – Software Process Assessment (2008)
11. Automotive SPICE, Process Assessment Model (PAM) v2.5 (2010)
12. Automotive SPICE, Process Reference Model (PRM) v4.5 (2010)
13. Smith, D.J., Simpson, K.G.L.: Functional Safety: A Straightforward Guide to Applying IEC 61508 and related Standards, 2nd edn. Elsevier, Butterworth Heinemann (2004)
14. Gall, H.: Functional Safety IEC 61508 / IEC 61511. The impact to Certification and the User. In: Proc. Of the IEEE/ACS International Conference on Computer Systems and Applications (2008)
15. Bilich, C.G., Zaijun, H.: Experiences with the Certification of a Generic Functional Safety Management Structure According to IEC 61508. In: Buth, B., Rabe, G., Seyfarth, T. (eds.) SAFECOMP 2009. LNCS, vol. 5775, pp. 103–117. Springer, Heidelberg (2009)
16. Lahtinen, J., Johansson, M., Ranta, J., Harju, H., Nevalainen, R.: Comparison between IEC 60880 and IEC 61508 for certification purposes in the nuclear domain. In: Schoitsch, E. (ed.) SAFECOMP 2010. LNCS, vol. 6351, pp. 55–67. Springer, Heidelberg (2010)

# Automatic Synthesis of SRN Models from System Operation Templates for Availability Analysis

Kumiko Tadano, Jiangwen Xiang, Masahiro Kawato, and Yoshiharu Maeno

Service Platforms Research Laboratories, NEC Corporation

Kawasaki 211-8666, Japan

{k-tadano@bq, j-xiang@ah, m-kawato@ap, y-maeno@aj}.jp.nec.com

**Abstract.** In order to cost-effectively verify whether system designs of information systems satisfy availability requirements, it is reasonable to utilize a model-based availability assessment of system design containing administrative operation procedures and a system configuration, because it does not require installing and testing in a real environment. However, since the model-based availability assessments typically require special expertise in mathematical modeling, it would be difficult for a practical system designer to build a correct availability model to assess his/her system design. Although there have been several methods to automatically synthesize the availability model from widely-used design description languages, the synthesized models do not capture impacts caused by operations in operation procedures on availability. To address this issue, this paper proposes a method to automatically synthesize an availability model in the form of stochastic reward net (SRN) from Systems Modeling Language (SysML) diagrams to specify operation procedures and system configurations. Modeling all the features of individual operations is impractical because the amount of required information in SysML diagrams input by system designers becomes larger as the number of features increases. To design the availability models with a smallest possible number of features, we classify typical availability-related features of operations into *operation templates*. The feasibility of the proposed method is studied by a case study based on a real system of a local government. We succeeded in synthesizing the availability models from the SysML diagrams based on an operation procedure and system configuration of the real system, and analyzing the synthesized availability models with an existing model analysis tool.

**Keywords:** availability model, stochastic reward Nets (SRNs), Petri nets, Systems Modeling Language (SysML), system operations.

## 1 Introduction

In order to verify whether system designs of information systems satisfy availability requirements, a system designer needs to quantitatively evaluate the system availability of the system. To evaluate system availability with limited budget, it is substantially reasonable to utilize a model-based availability assessment of system

design containing administrative operation procedures and a system configuration, because it does not require installing and testing in a real environment. Many model-based availability assessments of complex information systems have been performed successfully [1] [2] [3]. However, since the model-based availability assessments typically require special expertise in mathematical modeling, it would be difficult for a practical system designer to build a correct availability model to assess his/her system design.

An automated availability model synthesis method is necessary to support a model-based availability assessment of the system design. Several researchers proposed the methods to automatically derive stochastic models for availability/reliability/performance assessment such as Dynamic fault tree (DFT) [9], Timed Petri Net (TPN) [8], Stochastic Petri net (SPN) [10], Generalized Stochastic Petri Nets (GSPN) [11] [12], Deterministic and Stochastic Petri Nets (DSPN) [13], Stochastic Well-formed Net (SWN) [14] and Stochastic Reward Nets (SRN) [7] [15], from widely-used design description languages such as Unified Modeling Language (UML) [4], Systems Modeling Language (SysML) [5] and Architecture Analysis & Design Language (AADL) [6]. However, the models synthesized by these existing methods do not capture an impact caused by an operation procedure on system availability. Not only system configuration, but also administrative operation procedures impose a significant impact on the system availability. Quantitative assessment of the impact of various operations in the administrative operation procedures in a systematic manner is necessary.

To address this issue, this paper proposes a method to automatically synthesize an availability model in the form of stochastic reward net (SRN) [16] from SysML diagrams to specify operation procedures and system configurations. The synthesized SRN can be used to assess the impact of the operation procedure on the availability. Modeling all the features of operations is not practical for the following reasons: (1) the amount of information which the system designer needs to input increases as the number of features increases, and (2) it would take long time to calculate the system availability when the number of features is large, since the number of states in the models increases as the number of features increases. We choose the smallest number of features significant in modeling availability and classify them into five categories. This method is one of the essential functions of our case-based system assessment environment (CASSI) [17]. We apply the method to the information system for a local government as a case study, and analyze the availability model synthesized from SysML diagrams of operation procedure and system configuration.

## 2 Availability Model Synthesis Method

In this section, we propose a method to automatically synthesize an availability model in the form of SRN from SysML diagrams to specify operation procedures and system configurations.

### 2.1 Definition of System Designs

In this study, we define system design as follows. System design is created by a system designer. System design contains procedures of system administration

operations (henceforth, "operation procedures" for short) and a system configuration. The system configuration describes the system components such as operating systems, databases and applications which compose the system. The operation procedure is composed of system administration operations (henceforth, "operations" for short) such as status check, backup and change of configuration settings. The operations are performed by a system operator in accordance with the predetermined operation procedure.

### 2.2 Overview of Availability Assessment Process

Figure 1 shows the process to predict availability of the system as a target of assessment. At first, a system designer inputs SysML diagrams for the system design and values of parameters associated with portions of the SysML diagrams. The SysML diagrams consist of Activity Diagrams (AD) to describe system operation procedure, and Internal Block Diagrams (IBD) to describe system configuration.

Next, the availability model synthesis method automatically translates the input SysML diagrams to SRN with predefined SRN model modules stored in the *translation rules repository*. The availability model consists of a *control flow model* and *operation models* synthesized from AD, and *system configuration models* synthesized from IBD.

Finally, availability measures such as steady-state availability are calculated from the output availability models with such analysis tools as SPNP/SHARPE[18], [19]. This paper focuses on the availability model synthesis method which translates the operation procedure and system configuration to SRN automatically.

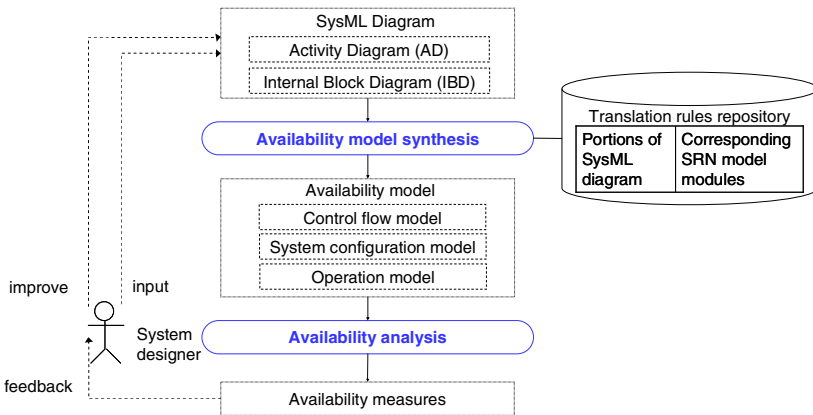


Fig. 1. Overview of availability assessment process

### 2.3 Approach for Availability Model Synthesis

It is not practical to model the all features of individual operations because the amount of information which the system designer needs to input increases as the number of features increases. In addition, it would take long to calculate the system availability when the number of features is large, since the number of states in the models increases as the number of features increases. To reduce the number of

necessary features in synthesizing the models, the operations are classified into the following five categories as the minimal set of features in modeling availability.

- (1) operations which do not influence availability (e.g. checking status)
- (2) operations which cause planned outage (e.g. scheduled maintenance)
- (3) operations whose failure may cause failure of the system component (e.g. making mistakes in configuration)
- (4) operations which recover the system from the planned outage (e.g. recovery from the scheduled maintenance)
- (5) operations which recover the system from unexpected errors (e.g. recovery from the system failure)

The operations (1) are not included in the availability models. Details of the operations (5) depend on the case-by-case decision of the human system operator or the service person called by the system operator, and cannot be determined in advance. Thus the operations of (5) are defined as a special transition to make the whole system recovered with an execution time much longer than other operations. The rest of the operations (2), (3), and (4) are represented by pre-defined operation templates described in SRN. Malicious operations which reduce availability are not within the scope of the paper (For security analysis, see [20]).

## 2.4 Availability Model Synthesis Process

The availability model synthesis method defines the translation rules between the input SysML diagrams and the output availability model. The availability model synthesis includes the following steps. Individual steps are detailed in the subsections.

- (Step 1) user input
- (Step 2) translation of operation procedure
- (Step 3) translation of individual operations
- (Step 4) translation of system configuration

### 2.4.1 User Input

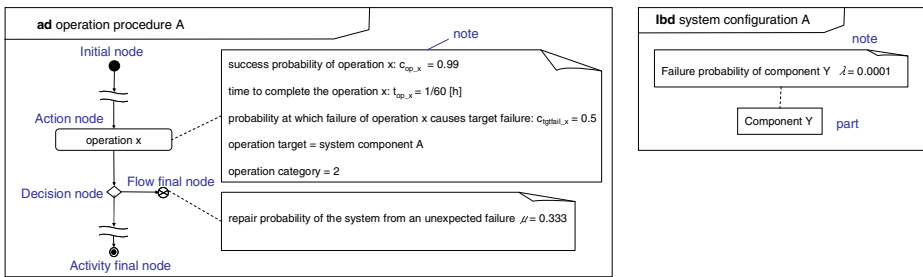
The proposed method supports Activity Diagrams (AD) and Internal Block Diagrams (IBD) as an input. AD represents operation procedures, and one of the elements in AD called *action* represents an operation to control the components in a system. IBD represent static system configurations, and one of the elements called *part* represents a system component like OS. Figure 2 shows examples of input IBD with a part and AD with an action. Parameter values associated with an action representing an operation are the success probability of the operation  $c_{op}$ , time to complete the operation  $t_{op}$ , the probability at which the failure of the operation causes the failure of the operation target  $c_{tgrfail}$ , the operation target, and the corresponding operation category for the operation described in Section 2.3. The operation category is determined according to three questions in Table 1. Depending on the answers to the questions, one of the categories (1) to (4) is selected. The parameter value associated with the flow final node is the repair probability of the system from an unexpected error  $\mu$ . These parameter values are input in a *note* associated with the action or the flow final node in AD. Regarding IBD, a parameter value associated with a part corresponding to a system component is the failure probability of the system component  $\lambda$ . These parameter values are utilized as transition probabilities of



synthesized availability models in the form of SRN. The parts corresponding to the operation targets are specified as the values of “operation target” in the notes associated with actions in AD by the system designer. Although SysML allocation also can be used to express various associations between elements, we use notes since SysML diagrams would be complicated if we use allocations when there are many operations and different operation targets. By inputting these parameter values in the notes in SysML diagrams before the synthesis of the SRN model, the system designer does not have to be aware of the underlying SRN models. When multiple operations are included in a single command such as reboot, multiples actions are used to describe the command in AD.

The system designer defines operation procedure as either scheduled or failure recovery. The scheduled operation procedure is carried out depending on the elapsed time. In this case, the system designer specifies the time. The failure recovery operation procedure is carried out when a failure happens. In this case, the system designer specifies the part of the IBD as a trigger to initiate the procedure.

The system designer also defines availability measures of interest. This paper focuses on the system availability and the probability at which the operation procedure is in execution as the measures of interest. To define the system availability, the system designer specifies under what conditions the system is regarded as available. To determine the conditions, the system designer selects the part(s) in IBD corresponding to the system component(s) which need(s) to be functioning. Based on the conditions, system availability is computed in analysis phase. The probability at which the operation procedure is in execution is automatically calculated.



**Fig. 2.** Examples of input SysML diagrams: an Activity Diagram (AD) and an Internal Block Diagram (IBD)

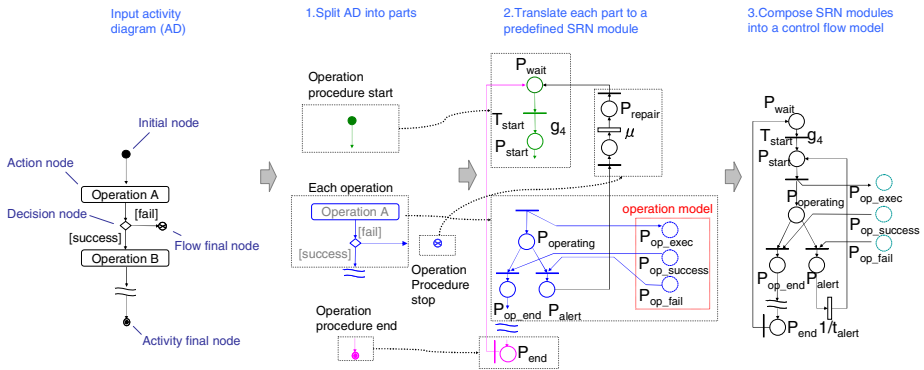
**Table 1.** Questions on the features of operations for the system designer

Questions	Answers	
	YES	No
(i) Does the operation stop its target component (i.e., planned outage) on purpose?	Select (2)	Go to Question (ii)
(ii) Can the operation failure cause unplanned outage (failure) of its target component accidentally?	Select (3)	Go to Question (iii)
(iii) Does the operation start up its target component from planned outage?	Select (4)	Select (1)

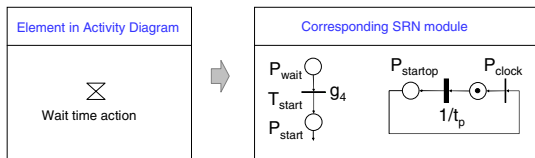
### 2.4.2 Translation of Operation Procedure

Figure 3 shows the process of translation of an Activity Diagram (AD). The AD is divided, and translated into pre-defined SRN modules stored in the translation rules repository. The SRN module to represent the conditional branch which depends on the status of the operation has places  $P_{op\_exec}$ ,  $P_{op\_success}$ , and  $P_{op\_fail}$ . The places are the interfaces to operation models. If the operation succeeds, a token moves to the place  $P_{end}$  and returns to the next operation. If all the operation succeeds, the token moves to the place  $P_{end}$  and returns to initial place  $P_{wait}$ . If the operation fails, the operation (5) is executed to recover from the unexpected error. The system recovers in the average time  $t_{repair}$  ( $=1/\mu$ ) and the token returns to the place  $P_{start}$ .

If the operation procedure is failure recovery, the transition  $T_{start}$  is enabled by guard function  $g_4$  when a certain system configuration model (see 2.4.4) is in the DOWN state. If the operation procedure is scheduled, an element of AD called “wait time action” is translated to the SRN module to represent clock shown in Figure 4, and the transition  $T_{start}$  is enabled by guard function  $g_4$  in regular time interval  $t_p$ . The SRN model is a collection of these translated SRN modules.



**Fig. 3.** Translation of an input Activity Diagram (AD) into a control flow model



**Fig. 4.** Translation of an element of an input Activity Diagram (AD) into a SRN module in the case that the operation procedure is performed in a certain time interval, i.e., triggered by the elapsed time

### 2.4.3 Translation of Individual Operation

The actions in AD are translated to operation models with the operation templates shown in Figure 5, which are stored in the translation rules repository. The operation templates are for the operation categories (2), (3), (4) in Section 2.3.

	User's answer to the question		
	(2) Operation stops the target	(3) Operation may cause the failure of the target	(4) Operation starts the target
Corresponding operation templates			
Parameters	$t_{op}$ : average time to complete the operation $C_{op}$ : success probability of the operation	$C_{tgt\_fail}$ : probability at which operation failure causes target failure	
Places	$P_{op\_exec}$ : the operation is being executed $P_{op\_fail}$ : the operation failed $P_{op\_success}$ : the operation succeeded $P_{stop}$ : the target is stopped as planned	$P_{tgt\_fail}$ : operation failure caused target failure $P_{tgt\_success}$ : operation failed but nothing happened	$P_{startup}$ : the target is started up from planned outage

Fig. 5. Operation templates used for the synthesis of operation models

### 2.4.4 Translation of System Configurations

The parts in IBD are translated to system configuration models. Each system configuration model includes UP and DOWN states. This simple system configuration model can be merged with SRN models representing complex system configurations such as redundant configurations including clustering and hot/warm/cold standby by the method proposed in Ref. [15].

Figure 6 shows the SRN modules used for the synthesis of system configuration models. The state transitions of SRN modules are as follows.

- (a) intrinsic (spontaneous) failure
- (b) unplanned (unexpected) outage when the token is in  $P_{tgt\_fail}$  in the operation template (3)
- (c) planned outage when the token is in  $P_{stop}$  in operation template (2)
- (d) recovery from the unplanned outage when the token is in  $P_{repair}$  in the control flow model
- (e) recovery from the planned outage when the token is in  $P_{startup}$  in the operation template (4)

The SRN modules (a) and (d) are mandatory. Table 2 shows the required SRN modules which depends on the selected operation templates. The same places of the required SRN modules are merged into one.

	Down			UP	
	(a) Intrinsic failure	(b) Unplanned outage caused by operation failure	(c) Planned outage by operation	(d) Recovery from failure	(e) Recovery from planned outage
Model modules					
Parameters	$\lambda$ : intrinsic failure probability				
Places	$P_{up}$ : the component is functioning	$P_{down}$ : the system component is down			
Guard functions		$g_1$ : Enabled if a token is in $P_{tgt\_fail}$	$g_2$ : Enabled if a token is in $P_{stop}$	$g_3$ : Enabled if a token is in $P_{repair}$	$g_4$ : Enabled if a token is in $P_{startup}$

Fig. 6. SRN modules used for synthesis of system configuration models

**Table 2.** SRN module required for synthesis which are determined by operation templates selected in 2.4.3

Selected operation templates	Required SRN modules for synthesis
(2)	(c)
(3)	(b)
(4)	(e)

### 3 Case Study

We applied the method to a system containing DataBase Management System (DBMS) and load balancer for a real local government shown in Ref. [22] as a case study. In the case study, first we classify the operations in the operation procedures in [22] according to the classification in 2.3. Then we show the synthesis of the availability model from SysML diagrams describing an operation procedure and system configuration by the proposed method, and the analysis of the synthesized availability model with SPNP.

#### 3.1 Classification of Operations

We classify the operations in the operation procedures in [22] according to the classification in 2.3. As for the operations corresponding to the operation category (5), we count the number of operations to call the service person when unexpected errors occur. Note that although such operations are not always explicitly stated in the operation procedures, in reality, the system designer needs to call the service person for unexpected errors of any operations. In Table 3, only operations which are explicitly stated in Ref [22] are included in the counting.

Table 3 shows the result of classification of the operation for this system. The total number of operations is 132. Regarding the features of operations, the numbers of operations are (1) 67, (2) 14, (3) 21, (4) 17, and (5) 13, respectively. Most are the operations (1) which are not included in the availability models. The number of operations in (2), (3), and (4) is similar. Most of the operations were successfully classified by the classification described in Section 2.3.

The following types of operations are difficult to handle by the current proposed method.

- When the operation does not impose any immediate bad impact, but may influence the availability in the long run (e.g. deletion of back-up files), human aid, rather than the automatic synthesis of the models, is necessary.
- When the specification of the operation is not clearly/completely described, the system designer needs to determine which operation category is suitable for the operation when it is input as AD.
- When the operation is optional (e.g. operation which may be carried out if the system operator has much time, file delete operation which is carried out only when sufficient amount of server disk is not available), whether or not the operation is carried out depends on the situation. The system designer needs to determine whether or not to include the operation in AD.

**Table 3.** The numbers of the operations included in a real operation procedure

System component	Sub procedure	The number of contained operations				
		(1) None	(2) Stop	(3) Failure	(4) Start from planned outage	(5) Repair from unplanned outage
Load balancer	Manual failover procedure	2	3	0	2	0
	Status check procedure	5	0	0	0	0
	Recovery procedure	8	0	1	4	5
Database server	Manual failover procedure	4	6	1	4	0
	Status check procedure	16	0	0	0	0
	Recovery procedure	32	5	19	7	8
<b>Total</b>		<b>67</b>	<b>14</b>	<b>21</b>	<b>17</b>	<b>13</b> <b>132</b>

### 3.2 Numerical Example

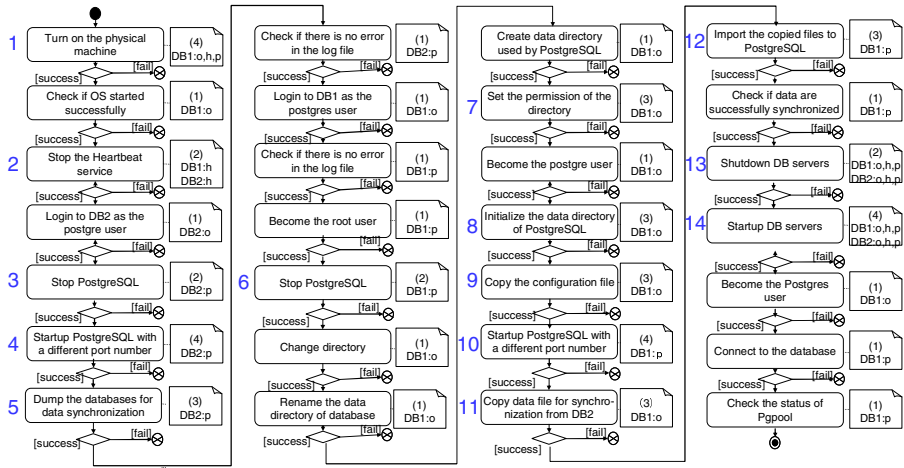
In this section, we analyze the availability model synthesized from SysML diagrams of an operation procedure and its target system components with SPNP [18].

#### 3.2.1 Target System

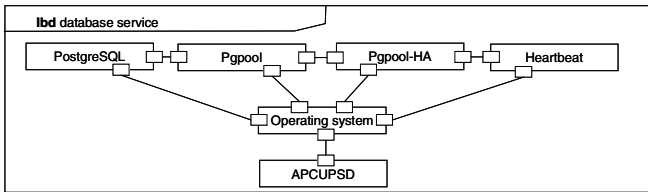
The system consists of a load balancer, two application (AP) servers, and two data base (DB) servers (DB1 and DB2). We assess the impact on availability of the system components by the operation procedure to recover from the failure of operating system (OS) of DB1. The target system components of the operation procedure are o:Operating system, h:Heartbeat, and p:PostgreSQL on the DB servers. This operation procedure includes complicated data synchronization. Heartbeat and Pgpool services are set to start automatically when the OS starts. Operations corresponding to the operation category (1) are not included in modeling as mentioned in Section 2.3, since the operations do not affect availability. In this case study, the operation procedure of recovery from the failure of Heartbeat or PostgreSQL is not included.

#### 3.2.2 Input SysML Diagrams

Figures 7 and 8 show the AD for the operation procedure and the IBD representing the system configuration of the DB servers, respectively. The notes associated with the actions in Figure 7 describe the values of the operation categories and the targets of operations only, since the space is limited. The rest of parameter values are described later. The parenthetic numbers in notes in the AD indicates the operation categories in Section 2.3. 14 operations in the figure 7 are classified to the operation templates (2), (3), or (4). These categories (2), (3), and (4) include 4, 6, and 4 operations, respectively. In this case study, the AD does not include optional operations in the operation procedure (e.g. file delete operation which is carried out



**Fig. 7.** Activity Diagram (AD) representing the operation procedure of the target system. The AD shows the repair procedure for a failure of an operating system of DB1.



**Fig. 8.** Internal Block Diagram (IBD) representing the system configuration of the target system

only when sufficient amount of server disk is not available). Although six components are included in the IBD, operation targets are OS, Heartbeat and Pgpool only.

### 3.2.3 Output Synthesized SRN Models

Figure 9 shows the operation models synthesized from the actions in AD. Figure 10 shows the system configuration models synthesized from the IBD. The system configuration model includes UP and DOWN states. Six SRN models represent OS, Heartbeat, and PostgreSQL running on DB1 and DB2. The availability measures are as follows. Steady-state availabilities of OS, Heartbeat, and PostgreSQL running on DB1 are calculated as the probabilities at which tokens are in the places  $P_{up\_1o}$ ,  $P_{up\_1h}$ , and  $P_{up\_1p}$  in the system configuration models of DB1. Table 4 shows the conditions which enable the transitions with the guard functions in the system configuration model in Figure 10. The probability at which the operation procedure is in execution is calculated. This can be calculated as the probability at which the token is not in  $P_{wait}$  in the control flow model. We do not show the synthesized control flow model in this Section. This is because the synthesized control flow model is the same as the

control flow model shown in Figure 2, except that the number of contained SRN modules representing the conditional branches is 14. Table 5 shows the parameter values of the synthesized availability model which consists of a control flow model, operation models and system configuration models.

	(2) Operation stops the target	(3) Operation may cause the failure of the target	(4) Operation which starts the target
Operation models			
Corresponding actions	(2), (3), (8),(16)	(5),(6),(7),(9),(10),(11),(13),(14), (15)	(1), (4),(12),(17)

Fig. 9. Operation models synthesized from actions of the Activity Diagram shown in Fig.7

	Operating system	Heartbeat	PostgreSQL
DB1			
DB2			

Fig. 10. System configuration models synthesized from the Internal Block Diagram shown in Fig.8

### 3.2.4 Analysis Results

Table 6 shows the results of the analysis of the synthesized availability model using SPNP. The steady-state availabilities of Heartbeat and PostgreSQL are very small. On the other hand, the steady-state availability of OS is very large. This is because the operation procedure of recovery of OS is included. The results indicate that recovery operation procedure has a big impact on availability of the target system component. Recovery operation procedures for Heartbeat and PostgreSQL are supposed to increase the availability of Heartbeat and PostgreSQL to the values similar to OS. In terms of the cost, if fewer operation procedures are included, the human labor cost decreases. Instead, the down time cost would increase. With the availability modeling and analysis, we can discover the most cost-effective operation procedures which minimize the total cost.

**Table 4.** Conditions under which guard functions used in system configuration models shown in Fig.10 enable transitions

Guard functions	Places which enable the transitions	Guard functions	Places which enable the transitions
$g_{1\_1o}$	$P_{tgt\_fail}$ of (7),(8),(9),(11)	$g_{2\_2o}$	$P_{stop}$ of (13)
$g_{2\_1o}$	$P_{stop}$ of (13)	$g_{3\_2o}$	$P_{repair}$
$g_{3\_1o}$	$P_{repair}$	$g_{4\_2o}$	$P_{startup}$ of (14)
$g_{4\_1o}$	$P_{startup}$ of (1),(14)	$g_{2\_2h}$	$P_{stop}$ of (2),(13)
$g_{2\_1h}$	$P_{stop}$ of (2),(13)	$g_{3\_2h}$	$P_{repair}$
$g_{3\_1h}$	$P_{repair}$	$g_{4\_2h}$	$P_{startup}$ of (14)
$g_{4\_1h}$	$P_{startup}$ of (1),(14)	$g_{1\_2p}$	$P_{tgt\_fail}$ of (5)
$g_{1\_1p}$	$P_{tgt\_fail}$ of (12)	$g_{2\_2p}$	$P_{stop}$ of (3),(13)
$g_{2\_1p}$	$P_{stop}$ of (6),(13)	$g_{3\_2p}$	$P_{repair}$
$g_{3\_1p}$	$P_{repair}$	$g_{4\_2p}$	$P_{startup}$ of (4),(14)
$g_{4\_1p}$	$P_{startup}$ of (1),(10),(14)		

**Table 5.** Parameter values of the synthesized availability model

Parameters	Description	Values
$t_{op\_x}$	Time to complete operation $x$ ( $x=1,2,..14$ )	1/60 [h]
$c_{op\_x}$	Success probability of operation $x$	0.99
$c_{tgtfai\_x}$	Probability at which failure of operation $x$ causes target failure	0.8
$\mu$	Probability of recovery from the unexpected error	1/3 [1/h]
$\lambda$	Failure probability of system components	1/1440 1/h]

**Table 6.** The results of the analysis of the synthesized availability model

Availability related measures	Values
Steady-state availability of operating system of DB1	0.99960
Steady-state availability of Heartbeat of DB1	0.34487
Steady-state availability of PostgreSQL of DB1	0.38182
The probability at which the operation procedure is in execution	0.00067
	(5.87 [hour/year])

## 4 Summary and Future Work

In this paper, we propose a method to synthesize SRN automatically from System Modeling Language [6] (SysML) diagrams to specify operation procedures and system configurations. The synthesized SRN can be used to assess the impact of the operations on the availability. We choose the smallest number of features significant in modeling availability and classify them into five categories. We applied the method to the information system for local governments as a case study, and analyze the availability model synthesized from SysML diagrams of operation procedure and system configuration.

For future works, we are going to improve the operation templates by deriving general laws regarding the features of operations, develop a method to obtain more accurate parameters (measurement and theory), and address general redundant



configurations by introducing the method in [15] to synthesize from the SysML diagrams to describe the system architecture.

## References

1. Trivedi, K.S., Wang, D., Hunt, D.J., Rindos, A., Smith, W.E., Vashaw, B.: Availability Modeling of SIP Protocol on IBM WebSphere. In: Proc. of PRDC 2008 (2008)
2. Smith, W.E., Trivedi, K.S., Tomek, L.A., Ackaret, J.: Availability analysis of blade server systems. *IBM System J.* 47(4) (2008)
3. Castelli, V., Harper, R.E., Heidelberger, P., Hunter, S.W., Trivedi, K.S., Vaidyanathan, K., Zeggert, W.P.: Proactive management of software aging. *IBM Journal of Research and Development* 45, 311–332 (2001)
4. OMG Unified Modeling Language (OMG UML), Superstructure Version 2.3, <http://www.omg.org/spec/UML/2.3/>
5. OMG Systems Modeling Language (OMG SysML) Version 1.2 (2010), <http://www.omg.org/spec/SysML/1.2/>
6. The SAE Architecture Analysis & Design Language (AADL) (2009), <http://standards.sae.org/as5506a/>
7. Huszerl, G., Majzik, I., Pataricza, A., Kosmidis, K., Dal Cin, M.: Quantitative Analysis of UML Statechart Models of Dependable Systems. *The Computer Journal* 45(3), 260–277 (2002)
8. Bondavalli, A., Maizik, I., Mura, I.: Automated Dependability Analysis of UML Designs. In: Proc. 2nd Int. Symp. on Objectoriented Real-time Distributed Computing, ISORC 1999 (1999)
9. Pai, G.J., Dugan, J.B.: Automatic synthesis of dynamic fault trees from UML system models. In: Proc. of the 13th Int. Symp. on Software Reliability Engineering (ISSRE 2002), pp. 243–254 (2002)
10. Khan, R.H., Heegaard, P.E.: Translation from UML to SPN model: A performance modeling framework for managing behavior of multiple collaborative sessions and instances. In: Proc. of Int. Conf. on Computer Design and Applications, ICCDA (2010)
11. Rugina, A.E., Kanoun, K., Kaâniche, M.: A System Dependability Modeling Framework Using AADL and GSPNs. In: DSN 2006 Workshops on Software Architectures for Dependable Systems (WADS 2006), pp. 14–38 (2006)
12. Rugina, A.E., Kanoun, K., Kaâniche, M.: The ADAPT Tool: From AADL Architectural Models to Stochastic Petri Nets through Model Transformation. In: EDCC 2008, pp. 85–90 (2008)
13. Bernardi, S., Merseguer, J., Petriu, D.C.: A Dependability profile within MARTE. *Journal of Software and Systems Modeling*, 1–14 (August 2009)
14. Bernardi, S., Merseguer, J.: Performance evaluation of UML design with Stochastic Well-formed Nets. *Journal of Systems and Software* 80(11), 1843–1865 (2007)
15. Machida, F., Kim, D.S., Trivedi, K.S.: Component-based Availability Modeling for Cloud Service Management. In: Proc. 21st Int. Symp. on Software Reliability Engineering, ISSRE 2010 (2010)
16. Trivedi, K.S.: Probability and Statistics with Reliability, Queuing, and Computer Science Applications. John Wiley, New York (2001)
17. Kimura, D., Osaki, T., Yanoo, K., Izukura, S., Sakaki, H., Kobayashi, A.: Evaluation of it systems considering characteristics as system of systems. In: Proc. of 6th IEEE international conference on System of Systems Engineering (SoSE 2011). IEEE, Los Alamitos (in press 2011)

18. Hirel, C., et al.: SPNP: Stochastic petri nets. Version 6.0. In: Haverkort, B.R., Bohnenkamp, H.C., Smith, C.U. (eds.) TOOLS 2000. LNCS, vol. 1786, pp. 354–357. Springer, Heidelberg (2000)
19. Trivedi, K.S., Sahner, R.: Sharpe at the age of twenty two. SIGMETRICS Perform. Eval. Rev. 36(4), 52–57 (2009)
20. Roy, A., Kim, D.S., Trivedi, K.S.: Cyber security analysis using attack countermeasure trees. In: Proc. the Sixth Annual Workshop on Cyber Security and Information Intelligence Research, CSIIRW 2010 (2010)
21. Swain, A.D., Guttman, H.E.: Handbook of human reliability analysis with emphasis on nuclear power plant applications. NUREG/CR-1278, USNRC (1983)
22. Operation procedure document Ver. 1.0 (2008),  
[http://www.bsnnet.co.jp/info/press/2007ipa/9\\_01.pdf](http://www.bsnnet.co.jp/info/press/2007ipa/9_01.pdf)

# A Collaborative Event Processing System for Protection of Critical Infrastructures from Cyber Attacks <sup>\*</sup>

Leonardo Aniello, Giuseppe Antonio Di Luna,  
Giorgia Lodi, and Roberto Baldoni

University of Rome “La Sapienza”  
Via Ariosto 25, 00185, Rome, Italy  
{aniello,lodi,diluna,baldoni}@dis.uniroma1.it

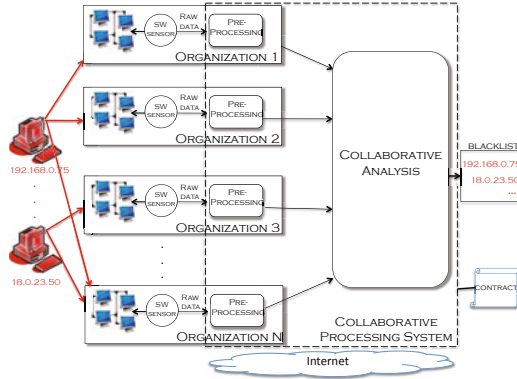
**Abstract.** We describe an Internet-based collaborative environment that protects geographically dispersed organizations of a critical infrastructure (e.g., financial institutions, telco providers) from coordinated cyber attacks. A specific instance of a collaborative environment for detecting malicious inter-domain port scans is introduced. This instance uses the open source Complex Event Processing (CEP) engine ESPER to correlate massive amounts of network traffic data exhibiting the evidence of those scans. The paper presents two inter-domain SYN port scan detection algorithms we designed, implemented in ESPER, and deployed on the collaborative environment; namely, Rank-based SYN (R-SYN) and Line Fitting. The paper shows the usefulness of the collaboration in terms of detection accuracy. Finally, it shows how Line Fitting can both achieve a higher detection accuracy with a smaller number of participants than R-SYN, and exhibit better detection latencies than R-SYN in the presence of low link bandwidths (i.e., less than 3Mbit/s) connecting the organizations to Esper.

## 1 Introduction

The seamless, ubiquitous, and scalable nature of the Internet has made it a convenient platform for critical infrastructures (e.g., telco, financial, power grids) as it allows them to benefit from reduced maintenance and management costs, and at the same time, offer a wider range of on-line and user-friendly services (such as on-line banking and e-commerce). The growing intersection between these critical infrastructures and Internet has however exposed them to a variety of security related risks, such as increasingly sophisticated cyber attacks aiming at capturing high value (or, otherwise, sensitive) information, or disrupting service operation for various purposes. Today’s cyber attacks result in both tangible and intangible economic losses due to the lack of service availability and infrastructural resilience, and the decreased level of trust on behalf of the customers<sup>1</sup>.

<sup>\*</sup> This research is partially funded by the EU project CoMiFin (Communication Middleware for Financial Critical Infrastructures [\[10\]](#)).

<sup>1</sup> Recent studies evaluate around 6 millions dollars per day the tangible loss for a utility company of a down of an e-service [\[14\]](#).



**Fig. 1.** Collaborative Event Processing System for inter-domain stealthy port scan

Hence, this economic argument pushes such organizations to collaborate in order to set more appropriate defense strategies.

We consider one of the most widespread mechanisms used by attackers for obtaining information on possible vulnerabilities of any target, i.e., *port scan*. Port scan is a preparatory action performed in several coordinated cyber attacks such as worm spreading, botnet formation and DDoS attacks. Single organizations use Intrusion Detection Systems (IDSs) to defend themselves from such actions. However, nowadays attackers attempt to perform their activities in a stealthy fashion in order to elude local IDSs. In particular, attackers distribute the port scans both in space and in time executing what we call an *inter-domain stealthy port scan*. In an inter-domain stealthy port scan a few ports of interest at different organizations are probed in order to circumvent configured thresholds (distribution in space), and single probes are delayed so as to bypass time window controls (distribution in time).

In this paper we propose a *collaborative* approach that allows us to address the general problem of protecting geographically dispersed organizations, belonging to different administrative domains, from cyber attacks. This is the typical scenario of organizations belonging to a critical infrastructure such as inter-utility of large scale power systems [17], networked telecommunication providers [24] or financial infrastructure [10]. In particular, we provide a novel collaborative event processing system for detecting inter-domain malicious port scan activities (see Figure 1). The system consists of two principal components: an *event engine* and a so-called *gateway*, collocated at each organization network. A gateway captures network packets and executes a pre-processing on those packets; that is, it filters out the packets that are not relevant with respect to the processing of the specific port scan detection algorithm (it might also aggregate the packets in order to reduce the overall computation to be performed at the event engine side). The pre-processed packets are sent to the event engine that correlates the data in order to discover spatial and/or temporal relationships among apparently uncorrelated data that would have been undetected by in-house IDSs.

The collaborative processing system is based on the Esper Complex Event Processing (CEP) engine [5]; through it we designed a novel port scan detection algorithm named *Line Fitting*. Line Fitting is implemented on the top of the collaborative system by means of a set of SQL-like queries that can be configured at run time. We compare Line Fitting with another algorithm, namely Rank-based SYN (R-SYN) algorithm which we developed in the context of an intra-organization intrusion detection system [13]. The use of Esper is motivated by both the low cost of ownership compared to other similar systems [9] and the ability of dynamically adapting the detection logic by integrating/removing SQL-queries for facing new threats that may arise.

We carried out an experimental evaluation in order to assess the detection and false positive rates of the two algorithms by using real network traces that include malicious port scans. The assessment aims to evaluate the impact of the collaboration on such metrics. Additionally, we computed the latency of the detection in both algorithms when 3, 6 and 9 organizations participate in the system. Results show that an increased number of collaborative organizations leads to a more accurate detection. At the same time, collaboration has a reasonable impact on the detection latency: in the presence of link bandwidths connecting the organizations to the engine in the range of [6.5Mbit/s, 3Mbit/s], the collaborative system exhibits detection latencies which are acceptable for the inter-domain port scan detection application. In general, we observe that Line Fitting achieves high levels of accuracy with a smaller number of organizations than R-SYN, and with low link bandwidths (less than 3Mbit/s) it also shows better detection latencies compared to R-SYN.

Finally, our collaborative processing system can manage (i) anonymized data (during the pre-processing) for privacy purposes, (ii) contract lifecycle (to join the collaborative processing system) and (iii) monitoring the adherence to the contract. Contracts and their monitoring are used to enforce trust among possibly distrusting participants in the system, thus fostering the collaboration. These topics are outside the scope of this paper; interested readers can refer to CoMiFin documents where they have been extensively investigated [22], [21], [10].

The rest of the paper is organized as follows. Section 2 introduces Line Fitting. Section 3 describes the architecture we designed and implemented of a collaborative processing system for inter-domain stealthy port scan detection based on Esper. Section 4 introduces the implementation of Line Fitting in Esper, and Section 5 discusses the principal experimental results we have obtained from a comparison between Line Fitting and the previously implemented R-SYN. Section 6 discusses principal related work and finally Section 7 concludes the paper.

## 2 Cyber Attacks: Distributed Stealthy Port Scan

We show the benefits of the collaborative approach in the case of inter-domain stealthy port scans detection. A scanner  $S$  sends a SYN packet to a target  $T$

on a specific port  $P$  and waits for a response. If a SYN-ACK packet is received,  $S$  can conclude that  $P$  is open and optionally reply with an RST packet to reset the connection. We call this kind of connections *incomplete connections*. In contrast, if an RST-ACK packet is received,  $S$  can consider  $P$  as closed. If no packet is received at all and  $S$  has some knowledge that  $T$  is reachable, then  $S$  can conclude that  $P$  is filtered. Otherwise, if  $S$  does not have any clue on the reachability status of  $T$ , it cannot assume anything about the state of  $P$ .

Not all the port scans can be considered malicious. For instance, there exist search engines that carry out port scanning activities in order to discover Web servers to index [19]. It becomes then crucial to distinguish accurately between actual malicious port scanning activities and benign ones.

**Line Fitting SYN Port Scan Detection Algorithm.** The underlying principle of Line Fitting concerns the observation that a scanner does not repeatedly perform the same operation towards specific hosts or ports: if the attempt fails on a  $T:P$  a scanner likely carries out a malicious port scan towards different targets. The rationale behind Line Fitting can be summarized as follows.

Let  $(ip, port)$  be the pair that identifies a destination host and a TCP port. Given a set of pairs  $C : IP \times Port$ , where  $IP$  is the set of IP addresses and  $Port$  is the set of TCP ports, the purpose of an inter-domain stealthy SYN port scan is to find out the subset  $A \subseteq C$  representing active TCP ports. A pair  $(ip_j, port_i)$  is active if and only if a service on port  $port_i$  is available at the destination IP  $ip_j$ . The standard behavior for a scanner is to issue few requests for each element in  $C$  in order to obtain the status of the pair  $(ip_j, port_i)$ .

Owing to these observations we define  $I = A \setminus C$  as the set of *inactive* pairs: every request issued to an element in  $I$  may lead to a failure. As failures are common during port scan activities, we can assume that  $I \neq \emptyset$  and that  $|I| \geq |A|$ .

The line fitting algorithm takes into account the set  $F_h$  which is a multiset of failures generated by the source host  $h$  (an element of  $I$  generated by  $h$  becomes an element of the set  $F_h$ ). We use the multiset since the multiplicity of any failure is crucial: we observe that in case of a normal failure (e.g., DNS crashes, service unavailability) the set  $F_h$  contains few elements with high multiplicity. In contrast, in case of a port scan the set includes many elements with low multiplicity. An ideal scanner issues few connections towards different (IP, Port) pairs exhibiting a “fitting curve” behavior; i.e., a horizontal line  $y = bx + q$  where  $b = 0$ , considering the pairs on x-axis and the multiplicities on y-axis. In contrast, a non ideal malicious port scan can emerge when  $b$  is close to 0.

Therefore, Line Fitting correlates data of the TCP three way handshake looking for patterns that are similar to a horizontal line representing few requests towards different (IP, port) pairs distributed over time. The patterns can be found by applying a linear fitting algorithm with the elements in  $F_h$ , checking then the similarity between the obtained fitting line and the ideal one. The algorithm we have designed and implemented can be described as follows.

---

**Algorithm 1.** Line Fitting algorithm

---

```

1.  $\forall x \in F_h$  if ( $x$  is inlier) List_h.add(x)
2. (b, q)=LinearFitting(List_h);
3. if( Match(b , q )){
4. portscanner(h); }

```

---

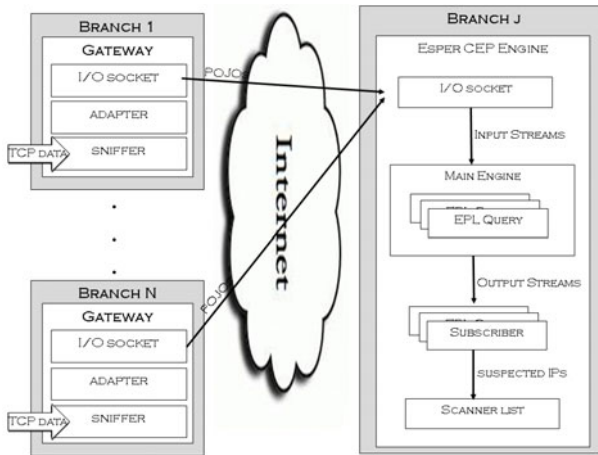
For all the elements  $x$  of type  $[destIP, port, multiplicity]$ , the check at line 1 of the algorithm “if  $x$  is inlier” is done using the mean and standard deviation of the series  $m(F_h)$  which is the list of multiplicities of all elements in  $F_h$ . If the multiplicity of  $x$  is in the interval  $[m - kd, m + kd]$  where  $m$  is the mean,  $d$  the standard deviation and  $k$  a constant value,  $x$  is considered inlier and is counted for the linear fitting (Line 2 of the algorithm). The linear fitting is realized through the least squares method and it produces two values of the fitting curve, namely,  $b$  and  $q$  which are then analyzed: if  $b$  and  $q$  are  $\leq$  than specific thresholds (the Match method in Line 3 of the algorithm; we set these thresholds to 1 and 6 respectively in our implementation) the source host  $h$  is considered a scanner and included in a blacklist (Line 4 of the algorithm).

**R-SYN Port Scan Detection Algorithm.** The Rank-based SYN (R-SYN) port scan detection algorithm adapts and combines three port scan detection techniques; namely (i) Half Open connections detection, (ii) Horizontal and Vertical port scans detection, and (iii) Entropy-based failed connections detection. The first technique aims at counting the number of incomplete connections. The second technique aims at identifying connection attempts to both a port across a range of IP addresses and a range of ports on a single destination host, and it uses a modified version of the Threshold Random Walk (TRW) mechanism introduced in [19]. The third technique aims at discriminating honest failures from malicious port scans. Finally, R-SYN employs a ranking mechanism that combines the results obtained from the three techniques in order to minimize the probability to miss a scanner which cheats by behaving apparently in a good way. The interested readers can refer to [13] for a detailed description of the R-SYN algorithm.

### 3 Collaborative Port Scan Detection System Architecture

Figure 2 illustrates the architecture of the collaborative processing system. The system consists of so-called Gateway components installed at each organization’s network participating in the collaborative system, and a single Esper [5] CEP engine instance used for processing purposes and deployed at any of the available organizations (the CEP engine could be hosted by the organization administering the processing system). These two components are described in detail below.

**Gateway.** Traffic data are captured from the monitored networks of organizations. The data are to be normalized and transformed in Plain Old Java Objects (POJOs) in order to be analyzed by the engine. To this end, the Gateway has



**Fig. 2.** Collaborative ESPER-based CEP architecture

been designed and implemented so as to (i) take as input the flows of network data (*TCP data* in Figure 2), (ii) filter them to maintain packets related to TCP three-way handshaking only, and, finally (iii) wrap each packet in a proper POJO to be sent to Esper.

We implemented *TCPPojo* for TCP packets. The POJO maps every field in the header of the protocol. POJOs are serialized and sent through Java sockets to Esper. When sending the POJOs our implementation maintains the order of the packets captured within the single organization, which is crucial when evaluating sequence operators in the Esper engine.

**Complex Event Processing (CEP).** The Esper CEP engine [5] receives POJOs that represent the events it has to analyze (input streams). The processing logic is specified in a high level SQL-like language named the Event Processing Language (EPL). In order to detect malicious port scanning activities a number of EPL queries are defined and executed by the engine, as shown in Figure 2. EPL queries run over a continuous stream of POJOs and produce output streams. When an EPL query finds a match against its clauses in its input stream, it generates a new tuple that is added to its output stream. A *Subscriber* is a Java object that can be subscribed to a particular output stream so that whenever the query outputs a new tuple, the *update()* method of the Subscriber is invoked using the tuple as argument.

We have implemented both algorithms as a set of EPL queries in Esper. In this paper we report the EPL implementation of the Line Fitting, only. The interested readers can refer to [13] for details on the implementation of R-SYN. Note that although the main implementation queries of R-SYN are unchanged since those discussed in [13], we have however modified the previous implementation in order to avoid the use of external data structures for the computation of the entropy-based failed connections. Our new implementation of R-SYN is fully realized through EPL queries, thus entirely exploiting the powerfulness of the language.



## 4 Line Fitting Implementation in Esper

For the implementation of the Line Fitting algorithm we first use general queries that filter specific packets of interest. In particular, filtering queries act on the TCPPojo input stream and filter both SYN packets and any packets involved in the TCP 3-way handshaking.

We then keep track, in the so-called *halfopen\_connection* output stream, of incomplete connections using the following query:

```
//Half Open (HO) connections
insert into halfopen_connection
  select ...
  from pattern [
    every a = syn_stream --> (
      ( b = syn_ack_stream(...) --> (
        (timer:interval(60 sec) or <c>) and not <d>
      ) where timer:within(61 sec) ) ) ]
```

We exploit the *pattern* construct of Esper to detect patterns of incomplete connections. In particular, *a* is the stream of SYN packets, *b* is the stream of SYN-ACK packets, *< c >* is the stream of RST packets and *< d >* is the stream of ACK packets, all obtained through the filtering queries previously mentioned. Such pattern matches if the involved packets are within a time window of 61 seconds.

In addition, we need to maintain the connections to unreachable hosts and closed ports. To this end, we use the query below for detecting unreachable hosts; it searches a data pattern in which a SYN packet is not followed by any packet matching the expression (*< b >* or *< c >*) within a time interval of 2 seconds. *< b >* represents the stream of SYN-ACK packets and *< c >* the RST-ACK packets stream. We also use the query for detecting connection attempts to closed ports for which we search patterns of SYN packets followed by RST-ACK packets within a time interval of 5 seconds.

```
//Connections to Hosts Unreachable(HU) //Connections to Closed Ports(CP)
insert into host_unreach                insert into closed_port
select ..., 0 as up, 1 as down          select ...
from pattern [                          from pattern[every a=syn_stream -->
  every a = syn_stream -->              <c> where timer:within(5 sec)
    timer:interval(2 sec) and          ]
    not (<b> or <c>)                    ]
]
```

Finally, Line Fitting needs to create the stream of events representing failed connections (*failures*); for this purpose, we use the following queries:

```
//Create failures stream from CP      //Create failures stream from HU
insert into failures                   insert into failures
select id,dst,1 as card                select id,dst,1 as card
from closed_port                       from host_unreach
where closed=1                          where down=1
```

```
//Create failures stream from HO
insert into failures
select id,dst,1 as card
from halfopen_connection
```

and for each couple (IP, Port) it returns the multiplicity of the multiset using the following query:

```
insert into multiset
select sourceIP,destIP,destPort,count(*) as N
from failures
group by sourceIP,destIP,destPort
```

Only one subscriber is associated with Line Fitting: it generates the list of scanner IP addresses waiting for 5 distinct events of type *failures* from the HO, HU, and CP streams and applies the least square method for the final computation.

## 5 Experimental Evaluation

We have carried out an experimental evaluation of the two algorithms. Such evaluation aims at assessing two metrics; namely the *detection accuracy* in recognizing distributed stealthy port scans and *detection latency*.

**Testbed.** For our evaluation we used a testbed consisting of a cluster of 10 Linux Virtual Machines (VMs), each of which equipped with 2GB of RAM and 40GB of disk space. The 10 VMs were hosted in a cluster of 4 quad core 2.8 Ghz dual processor physical machines equipped with 24GB of RAM. The physical machines are connected to a LAN of 10Gbit.

The layout of the components on the cluster consisted of one VM dedicated to host the Esper CEP engine. Each of the remaining 9 VMs represented the resources made available by 9 simulated organizations participating in the collaborative processing system. Each resource hosted the Gateway component. We emulated a large scale deployment environment so that all the VMs were connected with each other through an open source WAN emulator we have used for such a purpose. The emulator is called WANem [11] and allowed us to set specific physical link bandwidths in the communications among the VMs.

**Traces.** We used five intrusion traces. The first four were used in order to test the effectiveness of our algorithms in detecting malicious port scan activities whereas the latter has been used for computing the detection latency (see next paragraph). All traces include real network traffic of a network that has been monitored. The traces are obtained from the ITOC research web site [2], the LBNL/ICSI Enterprise Tracing Project [3] and the MIT DARPA Intrusion detection project [1]. The content of the traces is described in Table 1. In each trace, the first TCP packet of a scanner always corresponded to the first TCP packet of a real port scan activity.

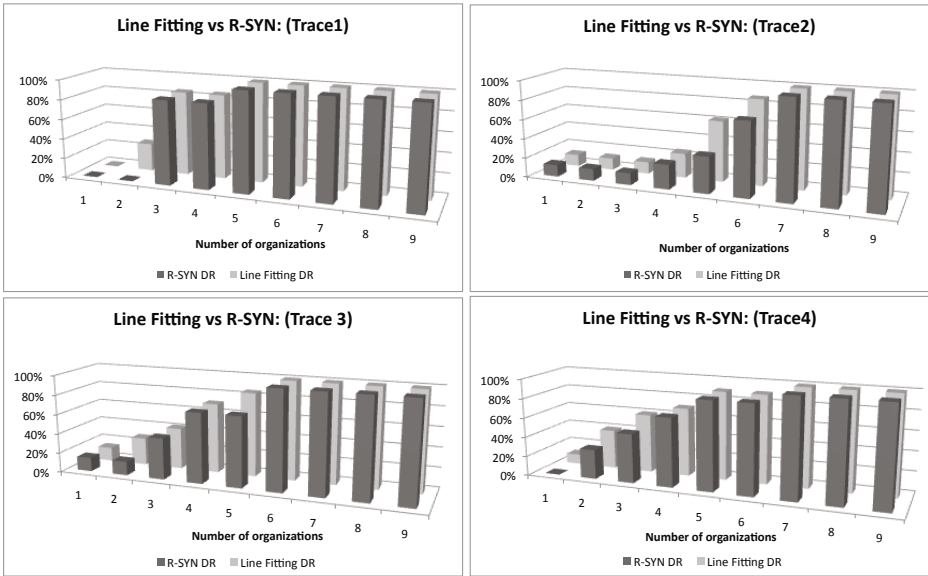
**Table 1.** Content of the traces

	trace1	trace2	trace3	trace4	trace5
size (MB)	3	5	85	156	287
number of source IPs	10	15	36	39	23
number of connections	1429	487	9749	413962	1126949
number of scanners	7	8	7	10	8
number of pkcts	18108	849816	394496	1128729	3462827
3way-handshake pkcts	5060	13484	136086	883500	3393087
length of the trace (sec.)	5302	601	11760	81577	600
3way-handshake pkct rate (p/s)	0.95	22.44	11.57	10.83	5655

**Detection Accuracy.** In order to assess the accuracy of R-SYN and Line Fitting, we partitioned the traces simulating the presence of 9 organizations participating in the collaborative processing system; the resulting sub-traces were injected to the available Gateways of each participant in order to observe what the two algorithms were able to detect. To this end, we ran a number of tests considering four accuracy metrics (following the assessment described in [27]): (i) *TP* (*True Positive*) which represents the number of suspicious hosts that are detected as scanners and are true scanners; (ii) *FP* (*False Positive*) which represents an error of the detection; that is, the number of honest source IP addresses considered as scanners; (iii) *TN* (*True Negative*) which represents the number of honest hosts that are not detected as scanners; (iv) *FN* (*False Negative*) which represents a number of hosts that are real scanners that the system does not detect. With these values we computed the *Detection Rate DR* and the *False Positive Rate FPR* as follows:  $DR = \frac{TP}{TP+FN}$ , and  $FPR = \frac{FP}{FP+TN}$ .

In all traces, with the exception of trace 4, we observed that none of the two algorithms introduced errors in the detection of port scanners; that is, in those cases the FPR was always 0% in our tests. In trace 4 of size 156MB, R-SYN exhibited a FPR equal to 3.4% against a FPR equal to 0% of Line Fitting; that is, R-SYN introduces 1 False Positive scanner.

Figure 3 shows the obtained results for the Detection Rate (DR). In this Figure, it emerges that the collaboration can be beneficial for sharpening the detection of port scanners. In both algorithms, augmenting the number of participants in the collaborative processing system (i.e., augmenting the volume of data to be correlated) leads to an increase of the detection rate as computed above. However, the behavior of the two algorithms is different: Line Fitting (light grey bars in Figure 3) converges more quickly to the highest detection rate compared to R-SYN (black bars in Figure 3); that is, in Line Fitting a smaller number of participants to the collaborative processing system and then a lower volume of data are required in order to achieve 100% of detection rate. This is principally due to a higher number of processing steps R-SYN executes and to R-SYN's subscribers that have to accumulate packets in order to carry out their TRW computation. In addition, R-SYN examines both good and malicious behaviors assigning a positive score to good ones. This implies that in



**Fig. 3.** Port scan DR vs number of organizations in the collaborative processing system for R-SYN and Line Fitting algorithms. Each organization contributes to the processing with a number of network packets that is on average  $1/9$  of the size of the trace.

some traces R-SYN has to wait more packets in order to effectively mark IP addresses as scanners.

**Detection Latency.** In the port scan attack scenario, the detection latency should be computed as the time elapsed between the first TCP packet of the port scan activity is sent by a certain IP address and the collaborative processing system marks that IP address as scanner (i.e., when it includes the address in the blacklist). Note that we cannot know precisely which TCP packet should be considered the first of a port scan, since that depends on the true aims of who sends such packet. As already said, in our traces the first TCP packet of a scanner corresponds to the first TCP packet of a real port scan activity so that we can compute the detection latency for a certain IP address  $x$  as the time elapsed between the sending of the first TCP packet by  $x$  and the detection of  $x$  as scanner.

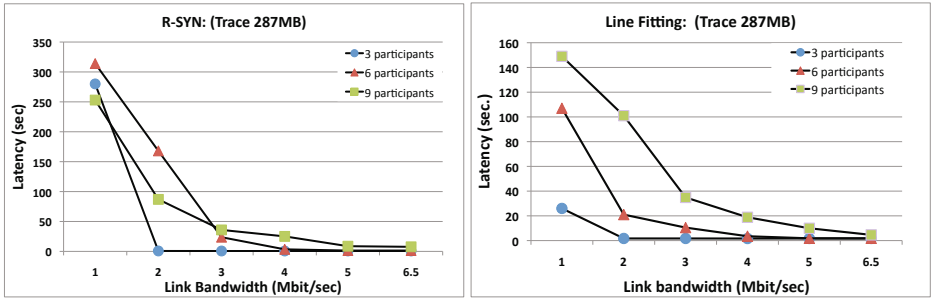
In doing so, we need the timestamps of the packets. For such a purpose we developed a simple Java application named `TimerDumping` which (i) takes a trace as input; (ii) sends the packets contained in the trace (according to the original packet rate) to the Gateway using a simple pipe; and (iii) maintains the timestamp of the first packet sent by each source IP address in the trace.

We deployed an instance of `TimerDumping` on each VM hosting the Gateway component. Each `TimerDumping` produces a list of pairs  $\langle ip\_address, ts \rangle$ , where  $ts$  is the timestamp of the first TCP packet sent by  $ip\_address$ . The timestamps

are then used as beginning events for detection latency computation. Since there are more `TimerDumping` instances, pairs with the same IP address but different timestamps may exist. In those cases, we consider the oldest timestamp.

Timestamps are generated using local clocks of the hosts of the cluster. In order to ensure an acceptable degree of synchronization, we configured all the clustered machines to use the same NTP server which has been installed in a host located at the same LAN. The offset between local clocks is in the order of 10 milliseconds which is accurate for our tests as latency measures are in the order of seconds.

For detection latency tests we used the trace of 287MB and changed the physical link bandwidths to the Esper in order to show in which setting one of the two algorithms can be preferable. Link bandwidth is controlled by the WANem emulator. We varied the physical link bandwidth using the WANem emulator with values ranging from 1Mbit/s up to 6.5Mbit/s. Figure 4 shows the average detection latency in seconds we have obtained in different runs of the two algorithms.



**Fig. 4.** R-SYN and Line Fitting detection latencies in the presence of 3, 6, and 9 participants in the collaborative processing system

As illustrated in this Figure, for reasonable link bandwidths of a large scale deployment scenario (between 3Mbit/s up to 6.5Mbit/s) both algorithms show a similar behavior with acceptable detection latencies for the inter-domain port scan application (latencies vary between 0.6 to 35 seconds). However, Line Fitting outperforms R-SYN in the presence of relatively low link bandwidths (looking at the left hand side of the curves, Line Fitting exhibits a detection latency of approximately 150 seconds when 9 participants are available against 250 seconds of R-SYN). In addition, in case of R-SYN, only, results show that when the collaborative system is formed by a higher number of participants (e.g., 9), detection latencies are better than those obtained with smaller collaborative systems. This is principally caused by the larger amount of data available when the number of participants increases: more data allow us to detect the scanners more quickly. In contrast, when 3 or 6 participants are available we need to wait more in order to achieve the final result of the computation. This behavior is not shown in case of Line Fitting for which an increased amount of information

is not sufficient to overcome the drawback related to the congestion on low link bandwidths (e.g., 1Mbit/sec).

## 6 Related Work

Many free IDSs exist that are deployed in enterprise settings. Snort [8] is an open source Network Intrusion Prevention/Detection System that performs real-time traffic analysis and packet logging on IP networks to detect probes or attacks. Bro [6] is an open-source Network IDS that passively monitors network traffic and searches suspicious activity. Its analysis includes detection of specific attacks using both defined signatures and events patterns, and unusual activities. In contrast to standalone IDSs, collaborative IDSs [4], [20], [25] significantly reduce time and improve efficiency of misuse detections by sharing information on attacks among the IDSs distributed at multiple organizations [26]. The main underlying principle of these approaches, namely the large-scale information sharing and collaborative detection, is similar to the ours. However, these systems are highly optimized for a specific type of attack whereas our Esper based architecture is a general-purpose system which can be effective against diverse attack scenarios.

CEP and Stream Processing (SP) systems play an important role in the IT technologies [9], [12], [23]. However, all these systems exhibit high cost-of-ownership. To this end, our solution employs open source CEP systems (e.g., JBoss Drools [7], Esper [5]).

The issue of using massive complex event processing among heterogeneous organizations forming a critical infrastructure for detecting network anomalies and failures has been suggested and evaluated in [18] and raised in [17]. Also the usefulness of collaboration and sharing information for telco operators with respect to discovering specific network attacks has been pointed out in [24]. In these works, it has been clearly highlighted that the main limitation of the collaboration approach concerns the confidentiality requirements. These requirements may be specified by the organizations that share data and can make the collaboration itself hardly possible as the organizations are typically not willing to disclose any private and sensitive information. This is also a critical issue in our collaborative system; however, in the context of the CoMiFIn project and of companion papers, we have deeply investigated how this architecture can be adapted to handle such issues [22], [21].

## 7 Concluding Remarks

It is well known that responsible information sharing among organizations that belong to the same economic infrastructure is a key factor for increasing their productivity (with consequent benefits for customers) such as improving competitiveness and cost reduction [16]. On the cyber security side, information sharing can facilitate the detection and prevention of cyber attacks.

The paper presented a collaborative processing system based on the Esper CEP engine. The system protects organizations willing to share specific network

data showing the evidence of distributed cyber attacks. The system has been instantiated for the detection of inter-domain port scanning. Two port scan detection algorithms have been designed and implemented, namely Line Fitting and R-SYN algorithms. Results show the effectiveness of the collaboration: augmenting the number of participating organizations, the detection accuracy increases. As for detection latencies, the collaboration has a reasonable impact: in the presence of link bandwidths in the range of [3Mbit/s, 6.5Mbit/s] the two algorithms exhibit acceptable detection latencies for our application. However, we note that Line Fitting outperforms R-SYN in terms of both detection accuracy and latency.

Future works include instrumenting the collaborative processing system to detect botnet-driven HTTP session hijacking attacks [15]. We are also investigating how to distribute the processing over a network of Esper sites in order to scale in terms of participating organizations. As shown in the performance results, the link bandwidth of Esper becomes a bottleneck when the number of organizations sending data increases. Thus, we wish to create a network of Esper sites able to distribute the load of the organizations' data and execute a first line of data aggregation and correlation.

## References

1. 2000 DARPA Intrusion Detection Scenario Specific Data Sets, <http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/2000data.html>
2. ITOC Research: CDX Datasets, <http://www.itoc.usma.edu/research/dataset/index.html>
3. LBNL/ICSI Enterprise Tracing Project, <http://www.icir.org/enterprise-tracing/>
4. DShield: Cooperative Network Security Community - Internet Security (2009), <http://www.dshield.org/indexd.html/>
5. Where Complex Event Processing meets Open Source: Esper and NEsper (2009), <http://esper.codehaus.org/>
6. Bro: an open source Unix based Network intrusion detection system, NIDS (2010), <http://www.bro-ids.org/>
7. JBoss Drools Fusion (2010), <http://www.jboss.org/drools/drools-fusion.html>
8. Snort: an open source network intrusion prevention and detection system, IDS/IPS (2010), <http://www.snort.org/>
9. System S. (2010), [http://domino.research.ibm.com/comm/research\\_projects.nsf/pages/esps.index.html](http://domino.research.ibm.com/comm/research_projects.nsf/pages/esps.index.html)
10. Communication Middleware for Monitoring Financial Critical Infrastructures (2011), <http://www.comifin.eu>
11. WANem The Wide Area Network emulator (2011), <http://wanem.sourceforge.net/>
12. Akdere, M., Çetintemel, U., Tatbul, N.: Plan-based complex event detection across distributed sources. PVLDB 1(1), 66–77 (2008)
13. Aniello, L., Lodi, G., Baldoni, R.: Inter-Domain Stealthy Port Scan Detection through Complex Event Processing. In: Proc. of 13th European Workshop on Dependable Computing, Pisa (May 11-12, 2011)

14. Baker, S., Waterman, S.: *In the Crossfire: Critical Infrastructure in the Age of Cyber War* (2010)
15. Bogk, A.: *Advisory: Weak PNG in PHP session ID generation leads to session hijacking* (March 2010)
16. Cate, F., Staten, M., Ivanov, G.: *The value of Information Sharing*. In: *Protecting Privacy in the New Millennium Series*, Council of Better Business Bureau (2000)
17. Hauser, C.H., Bakken, D.E., Dionysiou, I., Harald Gjermundrød, K., Irava, V.S., Helkey, J., Bose, A.: *Security, trust, and qos in next- generation control and communication for large power systems*. *IJCIS* 4(1/2), 3–16 (2008)
18. Huang, Y., Feamster, N., Lakhina, A., Xu, J.: *Diagnosing network disruptions with network-wide analysis*. In: *Proc. of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 61–72. ACM, New York (2007)
19. Jung, J., Paxson, V., Berger, A.W., Balakrishnan, H.: *Fast portscan detection using sequential hypothesis testing*. In: *Proc. of the IEEE Symposium on Security and Privacy* (2004)
20. Locasto, M.E., Parekh, J.J., Keromytis, A.D., Stolfo, S.J.: *Towards collaborative security and p2p intrusion detection*. In: *IEEE Workshop on Information Assurance and Security*, United States Military Academy, West Point, NY (June 15-17, 2005)
21. Lodi, G., Baldoni, R., Chockler, G., Dekel, E., Mulcahy, B.P., Martufi, G.: *A contract-based event driven model for collaborative security in financial information systems*. In: *Proc. of the 12th International Conference on Enterprise Information Systems*, Funchal, Madeira - Portugal (2010)
22. Lodi, G., Baldoni, R., Elshaafi, H., Mulcahy, B., Csertain, G., Gonczy, L.: *Trust Management in Monitoring Financial Critical Information Infrastructures*. In: *Proc. of the 2nd International Conference on Mobile Lightweight Wireless Systems - Critical Information Infrastructure Protection Track*, Barcelona (May 2010)
23. Tang, C., Steinder, M., Spreitzer, M., Pacifici, G.: *A Scalable Application Placement Controller for Enterprise Data Centers*. In: *16th International Conference on World Wide Web* (2007)
24. Xie, Y., Sekar, V., Reiter, M.K., Zhang, H.: *Forensic Analysis for Epidemic Attacks in Federated Networks*. In: *ICNP*, pp. 43–53 (2006)
25. Zhou, C.V., Karunasekera, S., Leckie, C.: *A peer-to-peer collaborative intrusion detection system*. In: *13th IEEE International Conference on Networks*, Kuala Lumpur, Malaysia (November 2005)
26. Zhou, C.V., Leckie, C., Karunasekera, S.: *A survey of coordinated attacks and collaborative intrusion detection*. *Computer and Security* 29, 124–140 (2009)
27. Zhou, C.V., Karunasekera, S., Leckie, C.: *Evaluation of a Decentralized Architecture for Large Scale Collaborative Intrusion Detection*. In: *Proc. of the 10th IFIP/IEEE International Symposium on Integrated Network Management* (2007)



# A Fault-Tolerant, Dynamically Scheduled Pipeline Structure for Chip Multiprocessors

Hananeh Aliee and Hamid Reza Zarandi

Department of Computer Engineering and Information Technology  
Amirkabir University of Technology (Tehran Polytechnic)  
{h.aliee,h\_zarandi}@aut.ac.ir

**Abstract.** This paper presents a dynamically scheduled pipeline structure for chip multiprocessors (CMPs). This technique exploits existing Simultaneous Multithreading (SMT), superscalar chip multiprocessors' redundancy to provide low-overhead, and broad coverage of faults at the cost of performance degradation for processors. This pipeline structure operates in two modes: 1) high-performance and 2) highly-reliable. In high-performance mode, each core works as a real SMT, superscalar processor. Whereas, the main contribution of the highly-reliable mode is: 1) To enhance the reliability of the system without adding extra redundancy strictly for fault tolerance, 2) To detect both transient and permanent faults, and 3) To recover existing faults. The experimental results show that the diagnosis mechanism quickly and accurately diagnoses faults. The fault detection latency for this technique is equal to the pipeline length of the processor, while it provides high fault detection coverage. Moreover, the reliable processor can function quite capably in the presence of both transient and permanent faults, despite of not using redundancy beyond which is already available in a modern microprocessor. Also, in the highly-reliable mode, the static and dynamic power consumption is declined by 25% and 36%, respectively.

**Keywords:** Reliability, Transient fault, Permanent fault, Fault tolerance, Pipeline structure, Chip multiprocessor, Superscalar processor.

## 1 Introduction

Technology scaling leads to widespread use of chip multiprocessors as a promising solution to use the large number of transistors [1-2]. Chip Multiprocessors (CMPs) provide higher levels of integration to achieve both high performance and reasonable power consumption within a packaged chip [3]. However, in forthcoming CMOS technology generations, this aggressive scaling poses critical reliability issues for various phenomena such as high energy particle strikes, voltage fluctuation, aging, lower supply voltage, and reduction in transistor size [4-6]. Due to being sporadic and unpredictable, transient-fault demands concurrent detecting and recovering. Therefore, fault tolerance is an area of major concern in recent designs.

To improve system reliability, there are various techniques. The three main commonly used techniques are mentioned here. The first one is information

redundancy techniques which involve adding extra information to existing information. These are mainly based on Error Detection/Correction Codes (ECC) or parity technique, which has been widely utilized to protect memory cells such as cache and register files [7]. The second one is hardware redundancy or spatial redundancy technique involving replication of hardware units, so that the same program will be executed by different pieces of hardware in parallel and their execution results could be compared to verify the correctness of the execution. Although it provides very high fault coverage, the hardware cost is also doubled [8]. The last one is categorized as time redundancy or temporal redundancy techniques which involve re-executing a program several times on the same pipeline [9]. The potential drawback of this scheme is that it might significantly reduce the overall performance.

As an example, to protect processors from soft errors, one approach is to execute two identical copies of a program simultaneously as independent threads [1]. The main challenges in these techniques are input replication and output comparison which requires interconnection buffers among the redundant copies. These extra buffers are shared among the copies and are fed by the main thread which makes them a single point of failure. Moreover, although they may require a few hardware because they use time redundancy in places where hardware redundancy is not critical, they are more time consuming than hardware-based approaches. In contrast, lockstep is used in the commercial fault-tolerant processors, such as IBM S/390 G5 [10] and Compaq Nonstop Himalaya [11]. Components are fully replicated and cycle-by-cycle synchronized to ensure that in each cycle, they perform same operation on same inputs and produce same outputs in the absence of faults. In fully replicated hardware components, fault detection coverage is more than previous methods because they do not have shared resources among redundant cores; however such systems are relatively expensive which precludes it from wide spread.

In this paper, we do not consider adding extra redundancy strictly for fault tolerance, because cost is such an important factor for commodity microprocessors. The key observation made by previous research [12-13] is that modern superscalar microprocessors, particularly simultaneously multithreaded (SMT) microprocessors, already contain significant amounts of redundancy for purposes of exploiting Instruction Level Parallelism (ILP) and enhancing performance. This is more when the number of cores in the combination of CMP/SMT increases. Superscalar processors exploit ILP by fetching and executing multiple instructions per cycle from a sequential instruction stream. The pipeline of the superscalar processor can handle more than one instruction per cycle, as long as each instruction occupies different pipeline stages [14]. SMT processors such as Intel Core i7, IBM POWER7, Sun Niagara and Rock, improve microprocessor utilization by sharing hardware resources across multiple active threads [15-17]. In this paper, we aim to tolerate faults by leveraging existing chip multiprocessors redundancy, at the cost of performance degradation for processors in the presence of faults.

To this end, we introduce a two-mode pipeline structure which can operate either in high-performance or highly-reliable mode. In high-performance mode, two distinct sets of programs (like two independent threads) can be run simultaneously in each core consisting two pipelines. So, the core can work with almost twice the speed of a typical core. While in the highly-reliable mode, the instructions of a program are

fetches and replicates in the pipelines. To ensure fault detection and recovery, the results of the pipelines are compared at the commit stage before presenting in the output. In the presence of a fault, the faulty instruction is re-executed. To reduce the cost of hardware replication, cores can be scheduled in different modes, so only the critical applications are run in the highly-reliable mode and other cores are operated in high-performance mode without performance degradation. This diagnosis mechanism quickly and accurately diagnoses faults. Moreover, the reliable microprocessor can function quite capably in the presence of both transient and permanent faults, despite of not using redundancy beyond which is already available in a modern microprocessor.

The experimental results on a MIPS-based superscalar processor with the ability of executing two instructions in parallel prove that the proposed technique in the highly-reliable mode decreases the static and dynamic power consumption by 25% and 36%, respectively. Also, the modifications to the base processor architecture do not affect the critical path delay, so the clock frequency is fixed. The fault detection latency for this technique is equal to the pipeline length of the processor, and it provides high fault detection coverage.

The rest of this paper is organized as follows. Section 2 describes the background on superscalar processors. In section 3, the proposed technique is discussed. Section 4 contains experimental results. Section 5 discusses the current solutions for fault detection and recovery. Finally, conclusions are presented in section 6 followed by references.

## 2 Background

This section discusses background on the main area we exploits in this paper: superscalar processors.

### 2.1 Superscalar Processors

A superscalar processor is one that is capable of sustaining an instruction-execution rate of more than one instruction per clock cycle. The pipeline of a superscalar

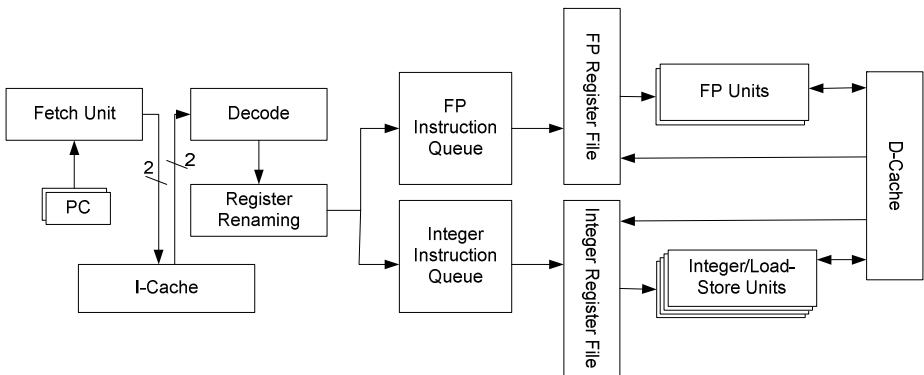


Fig. 1. Superscalar architecture [14]

processor can handle more than one instruction as long as it maps each instruction to a different pipeline stage. When the maximum capacity of a scalar processor is one instruction per cycle, the maximum capacity of a superscalar processor is more than one instruction depending on the level of parallelism supported in the processor. Fig. 1 shows a regular superscalar architecture [14] which fetches up to two instructions per cycle. Fetched instructions are then decoded and passed to the register renaming logic. Instructions are then placed in one of the instruction queues and waits until they are issued. Instructions are issued in functional units and after completing the execution, they are retired in order. This architecture provides some forms of hardware redundancy which is exploits in this paper to ensure reliable execution if correct execution is an essential feature of an application.

Adding simultaneous multithreading (SMT) to superscalar processors, which can be done with little overhead [14], overcomes underutilization of a superscalar processor due to missing instruction-level parallelism, where a processor can issue multiple instructions from multiple threads each cycle.

### 3 The Proposed Technique

In this work, we modify an SMT processor to both detect and recover faults using instruction-level redundancy by taking advantage of available resources in an SMT processor. This architecture operates either in highly-reliable or high-performance mode. Unlike a true SMT processor which is capable of handling  $n$  threads concurrently, this structure, in highly-reliable mode, supports  $n/2$  threads while the instructions of each thread are mapped to two distinct pipeline units in the same processor. The top view of this mode resembles an SMT processor with the set of  $n/2$  threads. The processor is scheduled, so each instruction can be executed twice. Redundant instructions are cycle-by-cycle synchronized and move concurrently through the pipeline stages. All replication and checking are performed transparently. In contrast, in high-performance mode, it acts as a real SMT processor without any overhead.

In this paper, we focus on extending an SMT, superscalar processor with two thread contexts and two pipeline structures to support our idea. Nevertheless, we can easily extend our design to support superscalar machines with more resources.

#### 3.1 Hardware Details

Without loss of generality, suppose a simplified MIPS-based processor pipeline [14] as a base processor. The processor contains two floating point units and two integer units. We assume that all functional units are completely pipelined. Each cycle, two instructions from two distinct threads are given control of the fetch unit. They are then decoded, issued, and executed. If they are memory operations (e.g. load and store instructions), they can access memory at memory stage to read from or write to memory. Finally, they are written back into register files in writeback stage. Data hazards are resolved by bypass unit. Also, interrupts and exceptions are taken into account by a system coprocessor. To handle branch hazards, branch prediction is provided by a history table per thread context which contains the last branch instruction results.

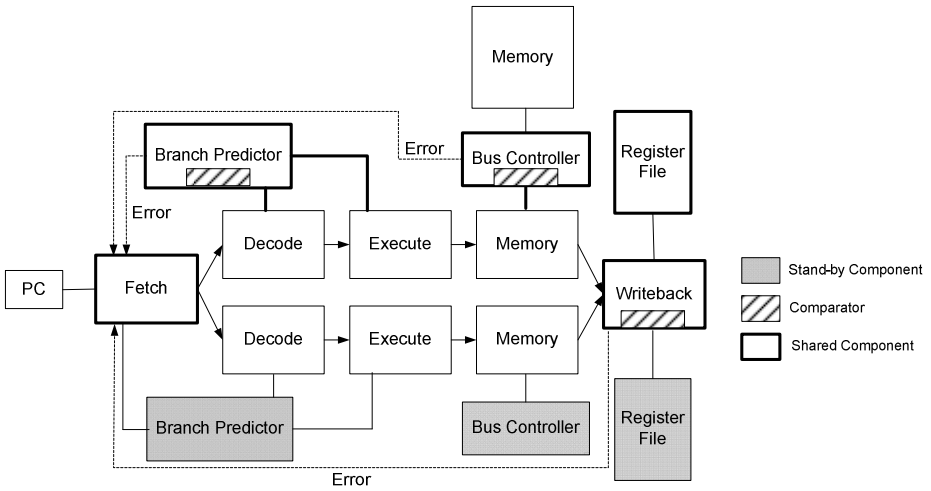


Fig. 2. The pipeline in the highly-reliable mode in the proposed technique

### 3.2 Highly-Reliable Mode

The main objective of the proposed technique is providing reliability for programs that require it and running other programs with the highest possible performance. To enhance the reliability of a superscalar processor, it is possible to take advantage of its available resources. Fig. 2 shows the pipeline of the proposed technique in the highly-reliable mode. In this mode, the instructions are fetched from one of the thread. After fetching an instruction, it is passed to two available pipelines in the processor. Output comparison involves checking register writeback values and memory store value. A mismatch in the outputs shows a fault in one of the components; which should be prevented from propagating into the system. The comparators in Fig. 2 are responsible for comparing the outputs and announcing the occurrence of an error in the presence of a fault in one of the outputs.

To reduce the overall power consumption of the system, components which are not employed can go to stand-by mode. In this figure, gray boxes refer to stand-by components. These components are idle because they are outside of sphere of replication [1]. The shared components are those which are accessed by both the main and the redundant instructions. These modules are required to either provide similar inputs (e.g. fetch unit) or compare the outputs (e.g. writeback unit and bus controller).

#### 3.2.1 Input Replication

Inputs to the pipelines should be handled carefully to ensure that both of them follow the same path and are cycle-by-cycle synchronized. Data from the outside of the sphere of replication should be replicated in the pipelines, so they receive same data values. Otherwise, the mismatch in their outputs is considered as a hardware fault. The inputs which need to be replicated are listed here:

**Fetch Instructions:** In this technique, in each cycle, both pipelines should execute same instruction. Fetch unit, reads instructions from memory, then pass them to

decode stages. The instructions go through the pipeline concurrently and reach memory and writeback stages at the same time which is essential for lockstep redundancy where in each cycle, the outputs are compared before propagating to the rest of the system out of the sphere of replication.

**Load Data:** Corresponding loads from replicated instructions must return the same values to each instruction. In temporal approaches, this is handled by employing some interconnection buffers which store loaded values of the leading thread for other trailing thread [1]. This is due to different access time of threads which may lead to inconsistency between their load values. However, in the suggested solution in this paper, this problem does not occur, because the pipelines are entirely synchronized and load instructions reach the memory stage at the same cycle; therefore a load value can be delivered to the both instructions without requiring extra buffer.

**Exceptions and Interrupts:** Interrupts and exceptions should be delivered to both set of instructions at precisely the same point in their execution. The exceptions and interrupts are managed in the coprocessor which is not shown in Fig. 2. If an interrupt or exception occurs, the fetch unit can easily redirect the program flow and start to fetch essential instructions. In this design, both pipelines receive identical instructions, so exceptions and interrupts are not concerned and this part of the processor can be left unmodified.

**Register Values:** Register files are not in sphere of replication. Hence, the outputs from this module must be replicated to ensure that both redundant instructions receive identical inputs. For that, it is enough to idle one of the register files and both pipelines work with a shared register file.

These concepts prove that in the proposed technique, the input replication can be easily managed with negligible overhead and effort.

### 3.2.2 Output Comparison

The sphere of replication boundary defines where outputs are compared. The outputs of the sphere of replication must be compared to guarantee that all data reaching out of this domain are reliable and correct. There are four types of values which necessitate comparing:

**Store Values:** Before forwarding a store value to the memory, both the address and the value of the store operations must be checked. Since redundant instructions reach the memory stage at the same clock, it is enough to verify their address and value at bus controller unit. The bus controller receives the address and the value of both memory stages. In highly-reliable mode, it first compares the values and the addresses, and then if there is no mismatch, it sends necessary signals to the memory to store the data. Otherwise, it sends an error signal to the fetch unit with the Program Counter (PC) value of the current store instructions for recovery phase, and no data is written in the memory. In thread-level redundancy, leading thread writes its store values in a store buffer and the second thread compares its address and data with the existing values in the buffer. So, in compared to the proposed technique, thread-level redundancy consumes more buffers.

**Load Address:** In the case of load operation, the load address of redundant instructions must be verified before reading from the memory. It is the responsibility

of the bus controller to get the load addresses from the both memory stages and compares them. A mismatch indicates a fault.

**Register Values:** In the writeback stage, the instructions in the pipeline write their register values in the register file. In this design, the register file is not in the sphere of replication, so the values written in this module must be checked to be fault free. Otherwise, faults are propagated to later instructions without being detected. In this case, the fault detection latency will be increased. Moreover, when a fault is detected, the origin of the fault is not recognizable.

**Branch Prediction Outputs:** Branch predictor determines the next PC to be fetched after a branch instruction. Due to lockstep execution, the predicted PC of redundant instructions must be verified before announcing to the fetch unit. In highly-reliable mode, one of the branch predictors is off and the other one is responsible for checking the branch outcomes.

### 3.2.3 Fault Recovery

Whenever a fault is detected in the system, an error signal is passed to the fetch unit. In addition to the error signal, the PC of the faulty instruction is also passed to this unit. In each clock, the fetch unit checks the error signal. If the signal is active, it starts to fetch from the PC indicating the faulty instruction and clears the other pipeline stages. Hence, if the fault is transient, it will be recovered by re-executing the instruction. In this condition, the fault recovery latency is 5 cycles maximum.

### 3.2.4 Permanent Fault Detection

The introduced technique can detect permanent faults as well as transient ones. However, it cannot distinguish permanent from transient faults. Permanent faults can be detected based on this fact that if over a period of time, more than a specific threshold of errors has been attributed to the system; it is very likely that this system has a permanent fault. To recognize permanent faults, a counter is specified to count the number of times an error has occurred in the system. When a mismatch is detected between the outputs of two redundant instructions, the counter is incremented by one and the execution is restarted from the faulty instruction (recovery phase). If the next execution is fault free, the counter is reset; otherwise if the counter exceeds the specific threshold, a permanent fault is reported. In the case of a permanent fault, the processor is marked as faulty and should stop its work.

## 3.3 High-Performance Mode

For some applications in the system, reliability may not be necessary. Therefore, these applications can be programmed in high-performance mode. In this mode, the processor operates as a true SMT, superscalar processor with the ability of performing two threads concurrently. The pipeline structure of the processor in this mode is similar to Fig. 2, however in this mode, all the units are active. The speed of a processor in high-performance mode is twice the speed of that processor in highly-reliable mode. The fetch unit can issue up to two instructions from two available threads. The instructions are then decoded and executed in decode and execution

units, respectively. In this mode, each instruction in the pipeline accesses the memory for load and store operations independently through separate bus controller. Moreover, each thread has its own register file and branch predictor.

### 3.4 Switching between High-Performance and Highly-Reliable Mode

In a CMP system, applications may have different features from reliability point of view. Some of them may have critical effect on the system, so they should be executed in highly-reliable mode to ensure that they produce correct outputs. To this end, cores in a CMP system can be scheduled in different modes based on the application running on them. The mode of each core can be programmed dynamically at run time with the help of the operating system.

## 4 Experimental Results

To implement this method, an instruction-level emulator in VHDL language is exploited; it borrows significantly from MIPS-I which is a MIPS-based emulator. This emulator is modified to provide the architecture mentioned in section 3. In this section, a brief comparison between the proposed techniques and two traditional schemes is addressed: the fully replicated hardware-based technique (FR) which duplicates a scalar processor with a single pipeline, and Simultaneously and Redundantly Threaded processors with Recovery (SRTR) [23]. It is assumed that each processor in FR technique has half the resources (functional units and data cache ports) of the SMT processor. In SRTR, two similar copies of a thread run on the SMT processor. One of the threads (the leading thread) goes ahead of the other redundant thread (the trailing thread). The instructions of the leading thread are committed only if they are checked by the trailing thread. A mismatch in comparison shows the occurrence of a fault in the system which is recovered by re-executing the faulty instruction. The parameters of the employed SRTR are presented in table 1 [23]. The SRTR is evaluated on an SMT processor similar to the architecture describes in the previous section.

**Table 1.** SRTR parameters

SRTR Parameters	
PredQ/LVQ/StB/RVQ	48/96/48/80 entries
Slack	32 instructions

**Table 2.** Comparison of the base superscalar processor versus the proposed technique

Architecture	# of Instructions per Clock	Clock Freq. (MHz)	Static Power (mW)	Dynamic Power (mW)	
Superscalar	2	30	1.094	1.676	
Proposed Technique	High-Performance	2	30	1.094	1.676
	Highly-Reliable	1	30	0.819	1.075



#### 4.1 Fault Model

In this paper, we target soft error which is a random event such as transient bit flips, that corrupts the value stored in a memory without damaging the cell itself [25]. The registers in processor's datapath from fetch to writeback stage are vulnerable to soft error. Other memory cells such as cache arrays are out of the sphere of replication and they are protected with information redundancy such as ECC and parity. However, these codes cannot be exploited for pipeline registers due to timing-critical nature of processor datapaths. The proposed technique in this paper protects these vulnerable memory cells to reduce the effect of soft errors.

#### 4.2 Results

Table 2 presents the number of instructions per clock, clock frequency, static power, and dynamic power of the base superscalar processor employed in this paper, and the proposed technique in both high-performance and highly-reliable mode. The superscalar processor architecture is described in previous section. The results are extracted using Synopsys tool chain [26] with the technology size of 65nm. The results prove that without considering stall, the optimistic number of committed instructions in each clock in the superscalar processor is similar to the proposed technique in high-performance mode, which is two times higher than the number of executed instructions per clock in the highly-reliable mode. This level of reduction is due to dual execution of each instruction in the highly-reliable mode for tolerating faults. The clock frequency of these three structures is equal which shows that the extra hardware which is employed for voting, has no effect on the critical path delay. In contrast, the static power and dynamic power have degraded 25% and 36%, respectively, in the highly-reliable mode in compared to two others. This reduction in power consumption is because of the components which are idle in the highly-reliable mode such as register file and bus controller. From this table, it can be concluded that in the high-performance mode, the processor behavior is similar to the base superscalar processor, while in the highly-reliable mode, the total performance is lower than the performance of the base processor, and however the power consumption has been decreased.

**Table 3.** Reliability comparison of FR, SRTR and the proposed technique

Technique	# of Memory Bits	# of Covered Memory Bits	Fault Detection Coverage (%)	Fault Detection Latency
FR	1723	1723	100	5
SRTR	10522	7450	100	Slack + 5
Proposed Technique	1818	1818	100	5

**Table 4.** Performance and area comparison of FR, SRTR and the proposed technique

Technique		Perf. Degradation (%)	Redundant Area
FR		32	One Processor + Comparators
SRTR		17	Comparators + Buffers
Proposed Technique	High-Performance	0	Comparators
	Highly-Reliable	13	Comparators

#### 4.2.1 Reliability Comparison

Table 3 estimates the fault coverage of the proposed techniques and compares that with FR and SRTR. The second column in this table estimates the number of datapath registers of the processor pipeline. The third column shows the number of registers which are fault-tolerant. Also, the fourth column presents the percentage of SEU fault detection coverage in these techniques. Finally, the last column shows the transient fault detection latency for these techniques. The total number of memory bits in the sphere of replication for FR is almost the same as the proposed technique. In SRTR, the interconnection buffers between the leading thread and the trailing thread have increased the number of memory bits noticeably, however any fault in these buffers are detected. FR, SRTR [19], and the proposed technique cover all the memory bits in the sphere; therefore the fault coverage is 100% for these structures.

When a fault takes place in one of the stages in the pipeline, the transient fault detection latency for FR and the proposed technique are approximately equal to the pipeline length (5 stages in this example). However for SRTR, this latency is equal to the slack value plus the pipeline length (37 in this example). SRTR benefits from slack to increase the performance of the trailing thread. In SRTR, the leading thread runs ahead of the trailing thread by the slack value. So, the trailing thread verifies the output of the leading thread after slack cycles. It can be concluded that FR and the proposed technique provide similar fault coverage, but they have less fault detection latency than SRTR.

#### 4.2.2 Performance and Area Comparison

Table 4 presents the performance degradation and the redundant area used for implementing each of the mentioned methods. The results in this table are in compared to our base SMT machine running one thread. The FR method degrades the performance for 32% [1], because each processor in this method has half the resources of the base SMT machine. In SRTR, two independent threads run concurrently and the processor can issue up to two instructions from the redundant threads each cycle which declines the performance of the system by 18%. Finally, to evaluate the performance of the proposed technique, it is implemented using Simplescalar tool set [27]. The set of benchmarks from SPEC2000 has been executed to extract the results. The results show that the proposed technique is 13% slower on average than the base machine. The high-performance mode of the proposed technique has no effect on the performance.

FR adds a scalar processor as a checker to checks the execution in the main processor. It then compares the external pins on each cycle. SRTR adds interconnection buffers and comparators to an SMT processor. Finally, the proposed technique adds only comparators to an SMT processor. FR and the proposed technique have almost equal area, while SRTR consumes more area than the proposed technique because of the interconnection buffers. Moreover, the proposed technique can be implemented with less modification to an SMT processor in compared to SRTR.

The main advantage of the proposed technique is that it provides higher performance with lower complexity than SRTR. More importantly, this technique can be scheduled dynamically in run time. Moreover, in a CMP system, each core can be programmed independently based on the application running in that core.

## 5 Related Work

Prior researches have employed variety of fault tolerance schemes. There have been several proposals to add extra redundancy to tolerate faults.

One of the most common and simple fault-tolerant solutions is duplicating hardware and compare the results [18]. Hardware-based fault tolerance solutions are transparent to programmers and system software, but require extra hardware. Compaq NonStop Himalaya detects soft errors by running identical copies of a program on two identical synchronized microprocessors. It locksteps the microprocessors and compares the external pins on each cycle. Also, IBM S/390 system replicates execution unit in the pipeline and execute identical instructions and data. [20] presents an approach to tolerate faults by utilizing instruction redundancy. It uses instruction reissue mechanism to tolerate transient faults accruing in the arithmetic and logical function by executing each committed instruction twice. The main disadvantage of this technique is that it only covers faults in functional unit not other components in a processor.

Mixed-Mode Multi-core (MMM) [21] enables one set of applications to run with high reliability in DMR mode, while another set, the performance applications, can avoid the penalty of DMR. The problems with this technique are its complexity as well as extra redundancy that should be considered for non-DMR applications to protect the integrity of reliable applications needing DMR.

To reduce the hardware cost of hardware-based fault tolerance approaches, several temporal techniques have been proposed which are more flexible and cheaper in terms of physical resources. Active Stream/Redundant Simultaneous Multithreading (AR-SMT) [22] is the first to use SMT processors to execute copies of the same program. In AR-SMT, two explicit copies of the program run concurrently on the same processor resources. Simultaneous and Redundant Threads (SRT) [1] processor improves on AR-SMT via the two optimizations of slack fetch and checking only stores. Later, Simultaneously and Redundantly Threaded processors with Recovery (SRTR) [23] enhances SRT to support recovery. It does not allow any leading instruction to commit before checking occurs. The recovery is done by re-executing the faulty instruction. Chip-Level Redundantly Threaded multiprocessor (CRT) [24] applies SRT's detection to CMP in which the leading and trailing threads are executed on different processors to reduce the probability of a fault corrupting both threads.

[19] proposes two semi-complementary techniques called Partial Explicit Redundancy (PER) and Implicit Redundancy Through Reuse (IRTR). This solution achieves better trade-off between fault coverage and performance degradation, however it increases the number of inter-thread communication buffers which makes it more complex than SRT. All these techniques rely on specialized hardware extensions.

## 6 Conclusion

This paper presents a fault-tolerant pipeline structure which is implemented with negligible modification to an SMT processor. SMT processors already contain significant amounts of redundancy for purposes of exploiting ILP and enhancing

performance. The proposed technique in this paper utilizes the available redundancy in these architectures to tolerate faults if it is essential. This technique operates in two modes: high-performance and highly-reliable. In high-performance mode, the processor works as a regular SMT processor with no performance and power overhead. In highly-reliable mode, the processor pipeline is scheduled so that each fetched instruction is mapped to two distinct pipelines in the base SMT processor. The redundant instructions execute concurrently through the pipelines. The lockstep feature of this technique makes input replication and output comparison easy to implement which are two real concerns in temporal redundancy solutions. When their results are ready to be presented in the output, the results are first verified to ensure that no fault will be propagated out of the sphere of replication. When a fault is detected, the fault is recovered by re-executing the faulty instruction.

The experimental results on a MIPS-based superscalar processor prove that the proposed technique in the highly-reliable mode decreases the static and dynamic power consumption by 25% and 36%, respectively. Finally, the fault detection latency for this technique is equal to the pipeline length of the processor, and it provides high fault detection coverage.

## References

1. Reinhardt, S.K., Mukherjee, S.S.: Transient-Fault Detection via Simultaneous Multithreading. In: *The Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA 2000)*, Canada, pp. 25–36 (June 2000)
2. Gibson, D., Wood, D.A.: Forward flow: a Scalable Core for Power-Constrained CMPs. In: *The Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA 2010)*, USA, pp. 1–12 (June 2010)
3. Bhattacharjee, A., Martonosi, M.: Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors. In: *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA 2009)*, USA, pp. 290–301 (June 2009)
4. Sanchez, D., Aragon, J.L., Garcia, J.M.: Extending SRT for Parallel Applications in Tiled-CMP Architecture. In: *The Proceedings of the 23rd IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2009)*, USA, pp. 1–8 (July 2009)
5. Prvulovic, M., Zhang, Z., Torrellas, J.: ReVive: Cost-Effective Architectural Support for Rollback Recovery in Shared-Memory Multiprocessors. In: *The Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA 2002)*, USA, pp. 111–122 (May 2002)
6. Aggrawal, N., Smiths, J.E., Saluja, K.K., Jouppi, N.P., Ranganathan, P.: Implementing High Availability Memory with a Duplication Cache. In: *The Proceedings of the 41st IEEE/ACM International Symposium on Microarchitecture (MICRO 2008)*, USA, pp. 71–82 (November 2008)
7. Zarandi, H.R., Miremadi, S.G.: A Highly Fault Detectable Cache Architecture for Dependable Computing. In: Heisel, M., Liggesmeyer, P., Wittmann, S. (eds.) *SAFECOMP 2004*. LNCS, vol. 3219, pp. 45–59. Springer, Heidelberg (2004)
8. Vadlamani, R., Zhao, J., Burleson, W., Tessier, R.: Multicore Soft Error Rate Stabilization Using Adaptive Dual Modular Redundancy. In: *The Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2010)*, Germany, pp. 27–32 (March 2010)

9. Kumar, S., Hari, S., Li, M., Ramachandran, P., Choi, B., Adve, S.V.: mSWAT: Low-Cost Hardware Fault Detection and Diagnosis for Multicore Systems. In: The Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2009), USA, pp. 122–132 (December 2009)
10. Siegel, T.J., et al.: IBM's S/390 G5 Microprocessor Design. *IEEE Micro* 19(2), 12–23 (1999)
11. Compaq Computer Corporation, Data Integrity for Compaq Nonstop Himalaya Servers (1999), <http://nonstop.compaq.com>
12. Bower, F.A., Sorin, D.J., Ozev, S.: Online Diagnosis of Hard Faults in Microprocessors. *ACM Transactions on Architecture and Code Optimization (TACO)* 4(2), article 8 (June 2007)
13. Srinivasan, J., Adve, S.V., Bose, P., Rivers, J.A.: Exploiting Structural Duplication for Lifetime Reliability Enhancement. In: The Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA 2005), USA, pp. 520–531 (June 2005)
14. Tullsen, D.M., et al.: Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor. In: The Proceedings of the 23rd Annual International Symposium on Computer Architecture (ISCA 1996), USA, pp. 191–202 (June 1996)
15. Eyerman, S., Eeckhout, L.: Probabilistic Job Symbiosis Modeling for SMT Processor Scheduling. In: The Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2010), USA, pp. 91–102 (March 2010)
16. Ramirez, T., Pajuelo, A., Santana, O.J., Valero, M.: Run ahead Threads to Improve SMT Performance. In: The Proceedings of the 14th International Symposium on High Performance Computer Architecture (HPCA 2008), UT, pp. 149–158 (February 2008)
17. Eyerman, S., Eeckhout, L.: Per-Thread Cycle Accounting. *IEEE Micro* 30(1), 71–80 (2010)
18. Timor, A., Mendelson, A., Birk, Y., Suri, N.: Using Underutilize CPU Resources to Enhance Its Reliability. *IEEE Transactions on Dependable and Secure Computing* 7(1), 94–109 (2010)
19. Gomaa, M.A., Vijaykumar, T.N.: Opportunistic Transient-Fault Detection. In: The Proceedings of the 32nd International Symposium on Computer Architecture (ISCA 2005), pp. 172–183 (June 2005)
20. Sato, T.: Exploiting Instruction Redundancy for Transient Fault Tolerance. In: The Proceedings of the 18th International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2003), USA, pp. 547–555 (November 2003)
21. Wells, P.M., Chakraborty, K., Sohi, G.S.: Mixed-Mode Multicore Reliability. In: The Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2009), USA, pp. 169–180 (March 2009)
22. Rotenburg, E.: AR-SMT a Microarchitectural Approach to Fault Tolerance in Microprocessors. In: The Proceedings of 29th Annual International Symposium on Fault-Tolerant Computing Systems (FTCS 1999), USA, pp. 84–91 (June 1999)
23. Vijaykumar, T.N., Pomeranz, I., Cheng, K.: Transient-Fault Recovery Using Simultaneous Multithreading. In: The Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA 2002), USA, pp. 87–98 (May 2002)

24. Mukherjee, S.S., Kontz, M., Reinhardt, S.K.: Detailed Design and Evaluation of Redundant Multithreading Alternatives. In: The Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA 2002), USA, pp. 99–110 (May 2002)
25. Aggarwal, N., Ranganathan, P., Jouppi, N.P., Smith, J.E.: Configurable Isolation: Building High Availability Systems with Commodity Multi-Core Processors. In: The Proceedings of the 34th International Symposium on Computer Architecture (ISCA 2007), USA, pp. 340–347 (June 2007)
26. Ragel, R., Ambrose, A., Peddersen, J., Parameswaran, S.: RACE: A Rapid, Architectural Simulation and Synthesis Framework for Embedded Processors. In: Hinchey, M., Kleinjohann, B., Kleinjohann, L., Lindsay, P.A., Rammig, F.J., Timmis, J., Wolf, M. (eds.) DIPES 2010. IFIP AICT, vol. 329, pp. 137–144. Springer, Heidelberg (2010)
27. Burger, D.A., Austin, T.M.: The SimpleScalar Tool Set, Version 2.0. Technical report #1342, University of Wisconsin-Madison, Computer Science Department (June 1997)

# FloGuard: Cost-Aware Systemwide Intrusion Defense via Online Forensics and On-Demand IDS Deployment

Saman Aliari Zonouz<sup>1</sup>, Kaustubh R. Joshi<sup>2</sup>, and William H. Sanders<sup>1</sup>

<sup>1</sup> University of Illinois

<sup>2</sup> AT&T Labs Research

**Abstract.** Detecting intrusions early enough can be a challenging and expensive endeavor. While intrusion detection techniques exist for many types of vulnerabilities, deploying them all to catch the small number of vulnerability exploitations that might actually exist for a given system is not cost-effective. In this paper, we present FloGuard, an on-line intrusion forensics and on-demand detector selection framework that provides systems with the ability to deploy the right detectors dynamically in a cost-effective manner when the system is threatened by an exploit. FloGuard relies on often easy-to-detect symptoms of attacks, e.g., participation in a botnet, and works backwards by iteratively deploying off-the-shelf detectors closer to the initial attack vector. The experiments using the EggDrop bot and systems with real vulnerabilities show that FloGuard can efficiently localize the attack origins even for unknown vulnerabilities, and can judiciously choose appropriate detectors to prevent them from being exploited in the future.

## 1 Introduction

Automatic response to security attacks that exploit previously unknown vulnerabilities can help the majority of computer systems that are not supported by dedicated security teams. If an attack's initial infection point can be isolated to an individual process or file, response techniques such as online attack signature generation and filtering, e.g., [10,31] can be effective. However, the usefulness of such approaches for unknown "zero-day" attacks is often hampered by lack of early and accurate detection of unknown vulnerability exploitations. There are approaches in the literature for detecting many different types of vulnerability exploitations such as buffer overflows [7], SQL injections and other format string attacks [30], and brute-force attacks [5]. Nevertheless, many computer systems today run with very little protection against unknown attacks, and often the first sign of compromise happens too late, either by users noticing degraded system performance, or ISPs detecting that the system is part of a BotNet or DDoS attack [23]. If such a range of detection options are available in the literature, why are they not used?

We hypothesize that there are two main challenges to early but effective attack detection: cost and precision. Many of the detection mechanisms cited earlier are simply too expensive to be continuously deployed. For example, bounds checking techniques such as CRED have overheads of as much as 300% [28], while taint-tracking can add

as much as 20X (Section 7). As one broadens the range of vulnerability types to be detected and the number of system components to be protected, the overheads add up, and push the cost threshold beyond which a technique becomes infeasible even lower. The move to mobile devices with limited compute power and battery life further exacerbates this problem. Detection mechanisms such as anomaly detectors or syscall sequence detectors that have low precision (i.e., high false positive rates) are rarely used even if their computational costs are low. On the other hand, cheap detectors do exist, e.g., change detectors for critical files [32], or anomaly detectors [5], but they often only detect the consequences of an attack, not the actual vulnerability that was exploited.

In this paper, we introduce FloGuard, which extends our previous work [34], an online forensics and backtracking framework that takes a system-wide cost-sensitive approach to attack detection and tracing. It uses the observation that although it may be difficult to notice the immediate effects of an exploit inexpensively, the ultimate consequences of attacks are often easier to detect. For example, inexpensive in-network scanning techniques such as BotGrep [23] can detect participation in a botnet or DDoS attack. Malware scanners such as ClamAV [17] can detect previously known payloads even if the attack vector is unknown, and anomaly detectors can detect deviations in a system's performance or modifications to its critical files. Once an attack is detected in this manner, FloGuard can roll the system back to a clean checkpoint<sup>1</sup>, determine possible paths the attack could have taken to reach its detection point using an online forensics algorithm, and deploy additional security detectors to catch and detect the attack at an earlier stage the next time that it is attempted. By iteratively repeating this process, it can deploy detectors successively closer to the initial attack vector until such a time that the attack can be stopped by automatic prevention techniques, such as input signature generation or quarantining.

To determine the set of detectors to enable and disable in every iteration, FloGuard uses an *Attack-Graph-Template* (AGT), which is an extended attack graph that enumerates *possible* attack and privilege escalation paths in the system. AGTs include possible paths, not ones known to be in the particular implementation being protected. E.g., an AGT for a server written in C can include privilege escalation using a buffer-overflow exploit, even though there may be no such known vulnerability. Therefore, AGTs can be constructed automatically by using system call monitoring to track *all* information and control flow paths between a system's privilege levels via processes, sockets, and files. During the forensics phase, FloGuard solves an optimization problem based on the the deployed detectors' outputs, the cost and coverages of the unused detectors, and the paths encoded by the AGT to determine which detectors to deploy for the next round.

We believe that FloGuard is one of the first frameworks to effectively balance the cost of security detection mechanisms against their coverage. By invoking mechanisms “on-demand” only when they are needed to forensically evaluate an attack that is already known to exist for the target system, FloGuard can utilize expensive mechanisms such as buffer bounds checking and taint tracking that are known to have good coverage characteristics. Furthermore, since FloGuard is a whole-system tool, it can initiate its detection and forensics analysis based on a wide range of attack consequences that may

---

<sup>1</sup> We define a clean snapshot as the last system snapshot before all potential attack entry points, e.g., a socket connection, that could have influenced the detection point.



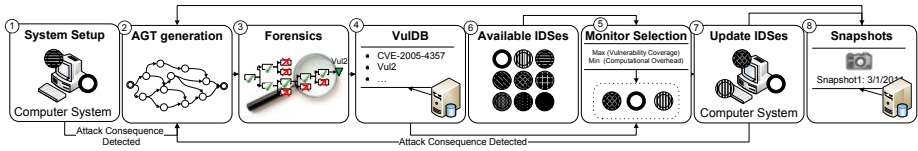


Fig. 1. FloGuard Architecture

be many processes and files away from the initial attack vector. Finally, the models we use are designed so that new security mechanisms can be easily integrated into its decision-making, making FloGuard a flexible and extensible detection tool that can be practically used for a wide variety of attack types.

The basic premise behind FloGuard is not that one cannot statically determine the necessary IDSes to install for “known” vulnerability (e.g., using CERT); instead, we content that statically deploying all the necessary IDSes or countermeasures to protect against the entire universe of “unknown” vulnerabilities that may be present in a system is not feasible from a performance and/or usability standpoint. We currently focus on scripted zero-day worms and malware exploiting both known and unknown vulnerabilities. These usually generate easy to detect consequences such as network scanning, and participation in botnets. Our focus is not on stealthy attacks (e.g., Stuxnet).

We demonstrate FloGuard’s capabilities against a modified real-world bot, namely Eggdrop [20], and several attacks against multiple real applications with a number of actual vulnerabilities in them. We show how FloGuard can integrate several off-the-shelf IDSes to detect a range of attacks that lie beyond any single tool’s capabilities.

## 2 Architecture

We begin by describing FloGuard’s architecture and its overall operation. Figure 1 shows a bird’s-eye view of different components in FloGuard. FloGuard assumes a virtual machine environment and requires the target system to operate as a guest VM<sup>2</sup>. FloGuard itself operates in the hypervisor/host OS of the VM environment to protect itself from attack and facilitate secure snapshotting facilities. Our prototype makes use of the Qemu [8] system emulator. The main inputs that FloGuard requires from a system administrator are a Vulnerability-Detector database (VulnDB) and an initial set of “attack consequence detectors.”

Attack consequence detectors are lightweight detectors that can operate continuously to detect the eventual symptoms of an intrusion, and cannot be disabled by an attack. Examples include in-network botnet/DDoS tracking done by ISPs and hypervisor-based file-integrity checkers. The output of an attack consequence detector is a process, port, or file that exhibits the symptoms of an attack. The VulnDB encodes information about the kinds of vulnerabilities that FloGuard is to guard against and the intrusion detection systems available to it. The database does not require knowledge of the specific vulnerabilities that may actually be present in the target system (which are unknown), but just

<sup>2</sup> In fact, VMs are not strictly needed, but they make incremental snapshots more efficient.

the high-level types, e.g., buffer overflow or SQL injection, that are possible and for which detection mechanisms are available.

During normal operation, FloGuard turns on the consequence detectors and periodically collects incremental snapshots of the system. It also keeps an append-only log of all system calls that are sent to a secure backend through a unidirectional communication link. When the attack consequence detector produces an alert (i.e., the detection point), FloGuard parses the syslogs, and determines the set of all potential attack sources (entry points) ((similar to [16, 14]). The last system snapshot taken before all the potential attack sources is marked as the last clean snapshot. FloGuard produces an Attack Graph Template (AGT), which by design consists of a superset of actual attack paths using the syslogs from the clean snapshot to the detection point<sup>3</sup>.

Through an iterative forensics analysis process, FloGuard invokes intrusion detectors, during each iteration (attack repetition), to refine the AGT from possible attack paths into an actual path. For each iteration, FloGuard selects a new set of detectors, rolls the system back to a past clean snapshot, deploys the detectors, and waits for a repeat attempt of the attack. After the iteration, FloGuard uses the outputs of the intrusion detectors to determine which paths in the AGT can be implicated or eliminated, and produces a refined AGT that is a subset of the original one. Eventually, once the actual attack path gets identified, a mitigation mechanism can then be used to block similar attacks in future. Thus, only the detectors related to vulnerabilities that are present in the system and for which an exploit actually exists need to be deployed in the process. Because FloGuard makes use of securely logged syscalls, it does not rely on any knowledge of what the attacker may do in the future. Additionally, because FloGuard works based on logged past activities within the system, it can work against social engineering attacks by tracing the attack path back to the executable which was downloaded during the social engineering phase of the attack.

### 3 Vulnerability-Detector DB

The vulnerability-detector database encodes all the domain knowledge about vulnerability types, detection mechanisms, and the applications in the system that are used by the forensics engine to make its detection decisions.

**Vulnerability Types.** In general, vulnerabilities are software flaws that are used by an adversary in the penetration step of an attack to obtain a privilege domain on a machine. The *vulnerability types* set in the VulDB includes all “possible” types of vulnerabilities that could potentially exist in different parts of the target system. A vulnerability type represents general vulnerability classes, e.g., buffer overflow, that encompass all target systems, without mentioning their context. It does not represent a *specific* vulnerability in the system, e.g., the vulnerable application’s name and the exact location in the application’s code. In addition, the VulDB also contains a target-system-specific mapping of

<sup>3</sup> It is important to mention that FloGuard also addresses kernel vulnerabilities. The last possible exploitation by the attacker, which would give him or her the highest privilege, is the root escalation. And that last exploitation (i.e., a consequence) is also logged, because logs are sent real-time to a backend server, and are later used for forensics analysis.

vulnerabilities to processes in the system. The mapping can be positive or negative (e.g., the eVision application [4] could be vulnerable to a SQL injection attack while the sshd daemon could not). Since the system’s actual vulnerabilities are unknown, these mappings represent *potential* vulnerabilities, not known ones. The mappings are optional, and FloGuard assumes that every process can be vulnerable to every vulnerability type if the mapping is missing.

**Detection Mechanisms.** The second element in the VulDB is the set of intrusion detection systems (IDSes) that are available to FloGuard to monitor different parts of the target system. Different kinds of IDSes that together cover as many different vulnerability types as possible are preferable. For each detection mechanism, FloGuard also requires a relative cost measure, e.g., CPU overhead, associated with the detector when it is deployed. The cost measure is only used to compare one intrusion detector against another; so as long as a uniform measure is used for all the detectors, it is not necessary for the cost to represent any specific performance or overhead measure. Ultimately, FloGuard’s main objective is to protect a system from attack once its corresponding vulnerability gets identified. While each detector can be converted into a rudimentary intrusion mitigator (by restarting the target process once the detector detects an intrusion), specialized mitigation mechanisms that may not detect attacks but can block them can also be included in the VulDB database. For example, a “disable account” action cannot detect attacks per se, but can block password attacks.

**Detector-Capability Matrix.** The detector-capability matrix indicates the ability of a given IDS to detect various vulnerability types. The matrix is defined over the Cartesian product of the vulnerability type set and the set of IDSes. Each matrix element shows how likely it is that each IDS, due to false positives and negatives, could detect an exploitation of a specific vulnerability type. In our experiments, we have used discrete N, L, M, H and C notations to represent no, low, medium, high and complete coverages, respectively. The detection capability matrix is later employed by the forensics and detector selection algorithms in deciding on the minimum-cost set of intrusion detection systems with maximum exploit detection capability.

## 4 Attack Graph Templates

Generally, every cyber attack scenario (path) consists of a number of subsequent exploits. In other words, the adversary, with initially no access to the system, can subsequently exploit various vulnerabilities to achieve the privilege required for his or her malicious goal, e.g., modifying a sensitive system file. Throughout this paper, exploits are represented as (process, vulnerability type) pairs in which the first and second elements denote, respectively, the vulnerable application, e.g., eVision, and the vulnerability type, e.g., buffer overflow, in the application.

Traditionally, an attack graph for a computer system represents a collection of known penetration scenarios according to a set of known vulnerabilities in the system [29]. In this section, we present the attack graph template (AGT), i.e., an extended attack graph, which is intended to cover all “possible” *attack types*. As an example, the attack graph template for a web server addresses the possibility that the server application might be

vulnerable to buffer overflow attacks, even if there is no such known vulnerability in the application. The attack graph template is a state-based graph in which each state is defined to be the set of the attacker's privileges in that state. State transitions in AGT (each mapped to a vulnerability exploitation) represent privilege escalations. Formally, the AGT encodes all possible attack paths from the initial state, in which the attacker has no privilege, to the goal state, in which the attacker has achieved the required privilege to accomplish his or her malicious goals.

In general, unknown vulnerability exploitations in a given application could be in any part of the application code; however, almost all of them are of known *types*, such as buffer overflow or SQL injection. Furthermore, as each IDS can detect certain *types* of exploits, generating AGTs that consider all possible vulnerability exploitations in applications allows FloGuard to determine which state transitions in an AGT get detected if a particular set of IDSes are deployed. Additionally, as the AGT is designed to consider all exploit types that constitute a finite set, the order (#states) and size (#state transitions) of AGT are finite, and often manageable in practice (see Section 7).

**Automatic AGT Generation.** FloGuard is particularly interested in the syscalls that cause data dependencies among the OS-level objects<sup>4</sup>. A dependency relationship is specified by three things: a source object, a sink object, and a timestamp<sup>5</sup>. For example, the reading of a file by a process causes that process (sink) to depend on that file (source). Given a detection point and the syscall logs, the dependency graph is generated using an algorithm similar to BackTracker [16]. Syslogs are parsed and analyzed line by line from the beginning to the detection point; their corresponding source and sink objects are created or updated; and a directed edge, labeled with the event's occurrence time, is created between those nodes.

We run two optimizations on the dependency graph before converting it to AGT. First, using time-sensitive backward reachability analysis, we eliminate irrelevant vertices/edges that do not causally affect the state of the detection point [19]. Second, by calculating transitive closure of the graph, we get rid of all the non-process nodes; any pair of processes get connected if there is a causal directed path [19] between them consisting of only non-process nodes. Finally, the refined dependency graph is used to generate the AGT. The idea is to consider any dependency graph edge connecting two nodes from different privileges (security contexts), a potential vulnerability exploitation by the attacker to obtain the privilege of the process to which the edge directs.

To convert the dependency graph to AGT, we traverse the dependency graph and concurrently update the AGT. First, the initial state of AGT with an empty privilege set is created. Starting from each entry point node in the graph, we run a causal depth-first search (DFS), i.e., with increasing time tags on the edges of the paths being traversed. While DFS is recursively traversing the dependency graph, it keeps track of the current state in the AGT, i.e., the set of privileges already gained through the path traversed so

<sup>4</sup> Throughout the paper, we use the term *OS-level objects* for processes, regular files, directories, symbolic links, character devices, block devices, FIFO pipes, and all thirty-five types of sockets, including internet sockets, i.e., `AF_INET`, interchangeably.

<sup>5</sup> We also log the security context of the objects. For instance, on a SE-Linux system, the web server directory is associated with the security type `httpd_sys_content_t`.

far by DFS. When DFS traverses a dependency graph edge that crosses over privilege domains, a state transition in AGT happens if the current state in AGT does not include the privilege domain of the process to which the edge directs. The transition is between the current state and the state that includes exactly the same privilege set as the current state plus the privilege of the process directed by the dependent graph edge. In fact, more than one transition edge might be created, depending on how many vulnerability types could potentially exist in the process, according to the VulDB.

Once the depth-first search is over, AGT is generated such that all its terminal states include the privilege domain of the process that had caused the detection point event during the attack. The last step would be to add a *goal state* to the AGT and connect all the terminal states to it using NOP (No-Operation) edges, meaning that no action is needed to make the transition. Once the AGT is generated, the forensics analysis (Section 5) tries to identify the exact path traversed by the attacker.

## 5 Intrusion Forensics

The forensics analysis by FloGuard requires two logging subroutines. First, we assume that an incremental snapshot of the system is taken periodically, e.g., once a day; therefore, we could go back to any time instant in the past that coincides with one of the snapshot times. Second, syscalls are being logged and stored in a secure back-end storage device while the system is operating. That enables FloGuard to generate the AGT for any past time interval.

FloGuard employs an iterative forensics algorithm; it restores a clean system snapshot and waits for attackers to launch similar attacks (exploiting the same vulnerability) several times. During each iteration, it deploys a different IDS to gather further evidence regarding the attack. The deployed IDSeS are chosen based on their cost, detection capabilities, and the generated AGT. In particular, FloGuard chooses the intrusion detector  $d^*$  for each forensics iteration using the following equation:  $d^* \leftarrow \arg \max_{d \in D} \{ \text{Coverage}(AGT, d) / \text{Cost}(d) \}$ , where  $D$  is the set of available IDSeS;  $\text{Coverage}(AGT, d)$  denotes the expected number of already-unmonitored transitions in  $AGT$  that are monitored by  $d$ . Using VulDB, FloGuard knows what vulnerability exploitations (state transitions) each IDS can detect; therefore, after each iteration, it can prune the AGT based on the deployed IDS and its alerts during the attack. More specifically, the detected vulnerability exploitations are marked, and the rest (the vulnerabilities whose exploitations did not get reported, while being monitored, by the deployed IDS) can safely be removed from the AGT. The refined AGT is used to choose the IDS for the next iteration (attack repetition). FloGuard continues the forensics iterations until the refined AGT consists of one marked path from its initial state to the goal state. The marked path is the actual path traversed during the attack.

In practice, detection systems are not always accurate, and sometimes either produce false positives or miss some misbehaviors (false negatives). FloGuard takes such inaccuracies into account by using their corresponding rates provided in VulDB<sup>6</sup> (otherwise,

<sup>6</sup> Before the calculations, the qualitative values in VulDB are mapped to their corresponding crisp values as follows.  $N$  and  $C$  are mapped to 0 and 1, respectively.  $L$ ,  $M$ , and  $H$  are, respectively, mapped to 0.25, 0.5, and 0.75.

a default value is used) and defining the edge weights  $w^e$  (which are all initially set to 1). In particular, if an IDS  $d$  reports a vulnerability exploitation  $e$  during a forensics iteration  $i$ , the corresponding edge (state transition) in AGT is not completely removed; instead, its weight is updated using the equation  $w_i^e \leftarrow w_{i-1}^e \times [1 - \text{FPR}(d, e)]$ , according to its previous weight and the false positive rate (FPR) of the IDS  $d$  in monitoring the exploit  $e$ . Similarly, in case no incident is reported, the weight is updated based on the IDS's false negative rate using the equation  $w_i^e \leftarrow w_{i-1}^e \times \text{FNR}(d, e)$ .

Essentially, to provide a precise automated forensics analysis, FloGuard must traverse the time dimension back and forth. FloGuard's implementation provides two different solutions, which are conceptually identical. 1) If the infrastructure supports system-wide restore/replay, FloGuard uses it to restore the past snapshot and replay the whole system several times, running the same forensics analysis as mentioned above, until the exact attack path in AGT is identified. 2) If the system-wide restore/replay is not supported, as described in this section, instead of making use of a restore/replay mechanism, FloGuard waits for the attacker to repeat the attack in the future. As our main target is scripted attacks, worms, and malware threats, so the repetition assumption is reasonable. In the unlikely scenario that an unprotected exploit is never re-exploited (i.e., an attack never repeats), forensics may not even be required - a simple rollback to pristine state will suffice. In this paper, we focus on the second situation due to the space limit; however, the main concept is the same for both solutions.

## 6 Monitor Selection

Once the attack path in the AGT is identified, FloGuard chooses the cost-optimal set of IDSes, as mitigation mechanisms (unless it is statically defined in VulDB) to deploy in the system permanently until the administrators install suitable patches. FloGuard selects and deploys a subset of IDSes that cooperatively detect and block exploitations of the *known* vulnerabilities represented by the refined AGT after the forensics analysis.

Formally, FloGuard decides upon the subset of IDSes to handle known vulnerabilities using  $D_k^* = \arg \min_{D_i \subseteq D} [\sum_{d \in D_i} \text{Cost}(d) \times \arg \max_{AP \in \text{AGT}} C(AP, D_i)]$ <sup>7</sup>, where  $AP$  represents an attack path (sequence of exploits) from the initial state to the goal state in  $AGT$ .  $D$  is the set of available IDSes. The  $C(AP, D_i)$  function denotes the overall cost if the system operates with IDSes  $d \in D_i$  turned on. The overall cost is determined through consideration of two factors: 1) detection cost (performance penalty); and 2) damage cost by the attacker trying to get from the initial to the goal state through the attack path  $AP$ . Depending on the exploits in  $AP$  and the detection capability of the IDSes in  $D_i$ ,  $AP$  might, but would not necessarily, be cut at some point between the initial and goal state by one of the detectors. Formally, the  $C$  function is defined as follows:

$$C(AP, D_i) = \sum_{e \in AP} \left[ \text{Depth}(AP_e) \cdot \prod_{e' \in AP_e} [w^{e'} \cdot \min_{d \in D_i} \text{FNR}(d, e')] \right], \quad (1)$$

which formulates the damage by the attacker before he or she gets caught by any of the deployed IDSes  $D_i$ . Briefly, we used the detection latency from the initial exploit

<sup>7</sup> In effect, the equation picks the subset of IDSes, that minimize the maximum possible cost that would result (according to AGT) if the system operated with that IDS subset deployed.

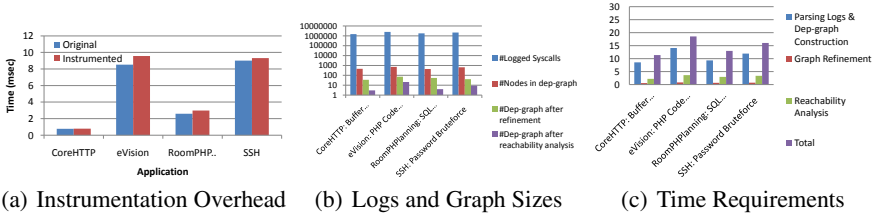


Fig. 2. Automated AGT Generation for Four Attack Scenarios

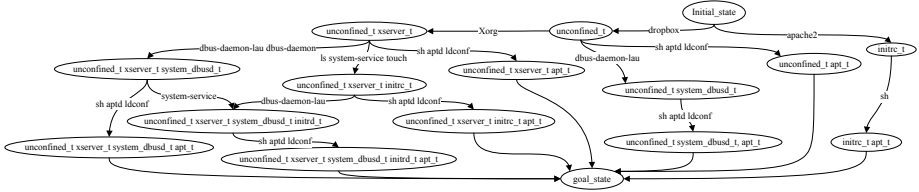
node to the detection point as the damage cost, since it determines how much rollback (and data loss) is needed. More formally, Equation (1) formulates the expected depth of penetration (number of subsequent vulnerability exploitations) the attacker can cause following the attack path  $AP$  without being detected by any of the deployed intrusion detectors  $D_i$ ; to do so, the algorithm considers the penetration depth of each subattack  $AP_e$  (defined as the subpath of  $AP$  from the initial state to  $e$ ) and the likelihood of it not being detected. The penetration depth for a subattack  $AP_e$ , denoted by  $\text{Depth}$ , is defined as the length of the path from the initial state to  $e$  through the attack path  $AP$ . The probability that the attack at a specific step  $AP_e$  is not yet detected is calculated by considering 1) the weights on each exploit  $e'$  in the subpath  $AP_e$  that have been updated through the forensics iterations; and 2) the false negative rates (denoted by FNR) of the deployed IDSes (more specifically, the IDS with the lowest false negative rate) regarding each exploit  $e'$  in the subpath  $AP_e$ . Consequently, the  $D_k^*$  equation above solves the tradeoff and selects the IDS set that minimizes the overall cost according to the AGT's structure. In practice, AGTs of actual attacks are small enough to permit a brute-force optimization of Equation (1).

## 7 Evaluations

We implemented FloGuard and evaluated it on a real botnet worm and different attack scenarios against four applications, each with a specific vulnerability. The vulnerability exploitations included buffer overflow exploitation, a SQL injection vulnerability, PHP remote code execution, and password attacks.

**Experimentation Setup.** The experiments were conducted on a system with 2.20 GHz AMD Athlon™64 Processor 3700+ CPU, 1 MB of cache, and 2.0 GB of RAM. The host and guest OSes running on the machine were Ubuntu 9.04 with Linux 2.6.22 kernels. The production system included a web server with several PHP applications, including eVision content management system [4], and the RoomPHPPlanning [3] scheduling application. Furthermore, the applications could connect to a MySQL database, and the trusted remote clients made use of SSH to obtain access to the system.

We used a set of IDSes that fall into the following categories. To block malicious use of library functions and malformed network packets, we used LibSafe [7] and Snort [27], respectively. To detect viruses and malicious actions on file system objects, we employed ClamAV [17] and Samhain [32], respectively. We employed Zabbix [5] and Memcheck in Valgrind [24] to detect anomalous activities, such as DoS or brute-force



**Fig. 3.** An Automatically Generated AGT for Remote PHP Code Execution

attacks, and general memory access violations, respectively. We used the TEMU [30] system-wide taint-tracking engine, which runs on the host OS. More specifically, using TEMU, one can mark some input data, such as network interfaces, as tainted, and then TEMU will track the information flow and store the executed instructions in a trace file on the host OS. To actually make use of TEMU, we had to improve its capability to produce higher-level information (not only instruction-level) regarding file-system objects, such as names of the files that are being dynamically tainted. We improved its implementation by using The Sleuth Kit (TSK) [9] to read the file system in the virtual machine’s image file; this enabled us to dynamically translate disk-level tainted addresses, which are generated by TEMU, to file system object names, such as file names and their absolute addresses.

**Services and Vulnerabilities.** Next, we describe the vulnerabilities and the affected services in our experiments. *SQL injection:* According to CVE-2009-4669, multiple SQL injection vulnerabilities in RoomPHPPlanning 1.6 allow attackers to execute arbitrary SQL commands. *Buffer overflow:* Based on CVE-2007-4060, an off-by-one error in the CoreHTTP 0.5.3.1 web server allows remote attackers to execute arbitrary code via an intelligently handcrafted HTTP request. *PHP remote code execution:* According to CVE-2008-6551, multiple directory traversal vulnerabilities in e-Vision 2.0 allow attackers to include and execute arbitrary PHP files. *The weak password:* Our production system includes accounts with weak passwords that open up the opportunity to make use of password-cracking software tools, e.g., John the Ripper [2]. We implemented a Perl password brute-force script that reads a file storing a large number of passwords and tries them against the target system.

**Instrumentation Overhead.** We implemented the syscall interception as a loadable kernel module; however, recently, other approaches like [18] have proposed interception of system calls from within the hypervisor using the previously generated signatures of the process memory images. However, that approach also assumes that the root domain is tamper-proof, since the attacker with root access can modify kernel data structures and consequently make the signatures useless. We configured the `/etc/syslog.conf` file such that all the syslogs are directly sent through a uni-directional link to a secure backend machine; therefore, all the system calls logged before the attacker gets access to the root domain are trusted.

The syscall interception module is always loaded while the system is operating. We measured the performance overhead by FloGuard’s syscall interception module (see Figure 2(a)). For the first three applications, the figure shows the average response time



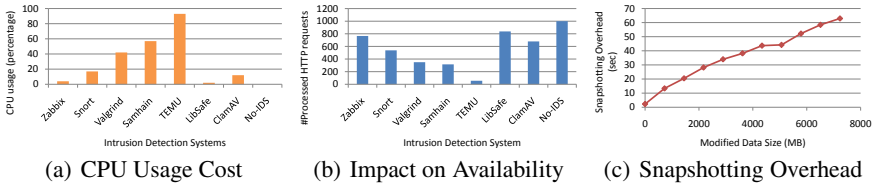


Fig. 4. Cost Evaluation of Individual Intrusion Detection Systems and the Snapshotting Procedure

for subsequent HTML requests for the CoreHTTP server, and PHP page requests for the eVision and RoomPHPlanning applications. The SSH results show the time required to transfer 100 KB of data from the server. In fact, our logging requirements are very modest. We only log a subset of syscalls that indicate data flow between processes/files (15 out of 350 call types) and only the first interaction when multiple syscalls are present between the same nodes. We ignore all syscall arguments except source or destination process/file IDs, and our encoding requires 10 bytes per syscall. Our 4 attack scenarios produced 40k records per minute (i.e., 576MB/day). Furthermore, syscall types repeat frequently, and after compression, each trace required an average of 10MB per day.

**Attack-Graph Template Generation.** We have collected the intercepted syscalls for the four abovementioned attacks. Figure 2(b) shows the number of syscalls for each attack on the system (the first columns). For each attack, once the attack consequence detector reports the detection point event, which was a sensitive file modification for all the attacks, the parser started reading the syslogs line by line (approximately 300K lines per second) and automatically creates the directed dependency graph. The second columns show the number of nodes in the generated dependency graph for each attack. The third columns illustrate the number of nodes in the generated dependency graph after non-process nodes are removed. The last phase (fourth columns) before generation of the AGTs is to further prune the dependency graph using reachability analysis.

Figure 2(c) shows how long (seconds) each of the abovementioned steps took. Because in our implementations, the syslogs parsing and the initial dependency graph creation are done concurrently, we report the time they took as a single result (first columns in Figure 2(c)). The second and third columns report the results for the graph refinement steps. Most of the total time is spent on parsing the syslogs and producing the initial dependency graph. As the reachability analysis significantly prunes the dependency graph, the time required to convert the resulting graph to AGT was less than 1 second in our experiments. Figure 3 shows the automatically generated AGT (using the Graphviz-dot tool) for the remote PHP code execution attack scenario. The AGT is represented using the SE-Linux privilege domains; however, for systems with traditional two-level discretionary access control, i.e., user and root, the nodes in AGT will have those two privilege levels only.

**IDS Computational Cost.** We measured the CPU overhead for individual IDSes. As illustrated in Figure 4(a), the TEMU engine puts the highest load on the system. Second, we deployed all of the IDSes and evaluated their impact on the system’s overall

throughput (see Figure 4(b)), which is defined as the maximum number of client requests, generated by HTTPTrafficGen [11], that could be processed within a fixed amount of time.

**Periodic Snapshots.** We measured the average performance overhead of the incremental system-wide snapshotting. Figure 4(c) illustrates the time needed for the engine to snapshot the whole system given the amount of data modified since the previous snapshot. As a case in point, if 2 GB of data are modified, e.g., downloaded, in the virtual machine between two successive snapshots, i.e., 30 minutes in our experiments, it takes about 13.4 seconds on average to pause the system and take and store a complete system-wide snapshot<sup>8</sup>. In our experiments, the snapshot restoration process was done quite fast, i.e., 3.2 seconds on average.

**Intrusion Forensics.** Finally, we present iterative intrusion forensics analysis results for six different attack scenarios. As the consequence detector, Samhain was configured to check the marked sensitive files and directories against its database and fire an alert upon identifying a modification or access.

First, we start with the buffer overflow attack scenario. While the CoreHTTP web server was operating after the snapshot, we remotely launched a buffer overflow exploit, which we had created manually using GDB, and got shell access to the machine. We then modified the web server's configuration file, i.e., `chttp.conf`, which had been marked to be monitored by Samhain. Upon receiving the Samhain alert, FloGuard started its forensics analysis by parsing the `syscall` logs from the last snapshot to the detection point, and generating the AGT. As shown in Table 1, the initial AGT consisted of 6 possible attack paths based on the intercepted `syscalls` during the attack. Having employed the monitor selection algorithm, FloGuard picked Valgrind as the first detector, as it maximized the coverage/cost measure, to deploy to monitor the web server. Consequently, the past clean snapshot was retrieved, Valgrind was deployed, and the system started its normal operation while FloGuard was waiting for the next repetition of the attack. We then relaunched the attack. Valgrind did not detect the buffer overflow in CoreHTTP, since it does not perform bounds checking on static arrays (allocated on the stack). After the first iteration, AGT was pruned, and the resulting AGT consisted of 3.7 expected number of attack paths (the fractional number is due to IDS uncertainties). Using the refined AGT and the detector-capability matrix, FloGuard chose the next detector, i.e., Valgrind on the `sh` process, and the iterative forensics continued until Libsafe was picked to monitor the web server that successfully detected the buffer overflow in CoreHTTP. Consequently, LibSafe was permanently turned on to detect and block similar attacks until the administrator manually patches the system.

The second attack scenario was the PHP remote code execution in the eVision-2.0 CMS application. We launched the attack using the Perl exploit from <http://www.exploit-db.com> against eVision-2.0 that enabled us to upload an arbitrary file using the local file inclusion. Consequently, we could remotely execute any arbitrary command on the server. As shown in Table 1, we modified the `/etc/passwd` file, which was marked to be monitored by Samhain. The first detector to turn on for

<sup>8</sup> Using the VirtualBox framework, taking live snapshots, w/o pausing the system, is possible.

**Table 1.** Iterative Intrusion Forensics Analysis

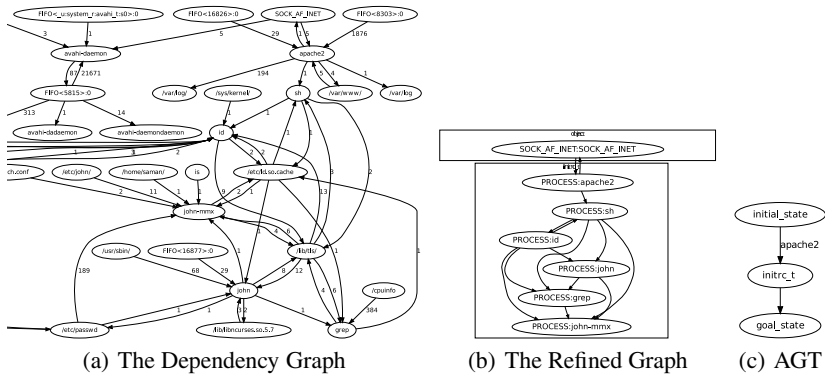
Attacks	CoreHTTP: Buffer Overflow				eVision: Remote Code Execution				SSH: Password Brute-force				RoomPHPlanning: SQL Injection			
DP	/var/www/chtcp.conf				/etc/passwd				/var/log/auth.log				/var/lib/mysql/RMP/rpresa.MYD			
	#Paths	IDS	Overhead	Dctd?	#Paths	IDS	Overhead	Dctd?	#Paths	IDS	Overhead	Dctd?	#Paths	IDS	Overhead	Dctd?
It.1	6	V(corehttp)	1.8X	No	53	TEMU(sh)	16.8X	No	4	V(ssh)	0.1X	No	5	V(mysql)	1.2	No
It.2	3.7	V(sh)	1.3X	No	9	V(apache2)	1.7X	No	1.7	Zabbix	0.3X	No	2.6	Zabbix	0.3	No
It.3	1.4	LS(corehttp)	0.2X	Yes	6.8	V(sh)	0.2X	No	0.9	LS(ssh)	0.1X	No	1.7	Snort	0.8	Yes
It.4	0.9				4.4	Zabbix	0.3X	No	0.4	Snort	0.3X	Yes	0.9			
It.5					3.4	ClamAV	0.5X	Yes	0.3							
It.6					2.4											

the forensics analysis was TEMU, because of its capability in tracing back the data from the process sh that caused the detection point event (see Figure 3). The sh process was then traced by TEMU’s tracebyname command, and the actual data source, i.e., the apache2 process (see Figure 3), was identified via the list\_tainted\_files command, which we had implemented by translating disk-level addresses to filenames. TEMU helped FloGuard to prune the AGT to include only the paths exploiting possible vulnerabilities in the apache2 process. Finally, the ClamAV detection system detected the uploaded file during the attack, and FloGuard, using the VulDB, decided to turn off the magic\_quotes (even though this might have affected other applications).

The third attack, i.e., SSH password brute-force, was remotely launched using a Perl password trial script. Subsequent password trials made Samhain fire an alert upon the /var/log/auth.log file modification. In forensics analysis, Snort finally detected the password brute-force attack. The next attack scenario was to modify a sensitive database file through the exploitation of a SQL injection. Upon receiving the Samhain alert, FloGuard, as shown in Table 1, selected Valgrind and Zabbix to be deployed in the first and second iterations, respectively; however, neither detected any misbehavior in the system. Finally, in the last iteration, FloGuard picked Snort, which successfully detected the SQL injection by identifying SQL meta-characters in the incoming data.

Both Snort rules picked by FloGuard in attacks #3 and #4 are non-standard rules (one written by us and one disabled by default) that cannot be permanently deployed because they have a high likelihood of false positives. On the other hand, FloGuard correctly identifies the specific rule that must be enabled, and which port to enable it on only when an attack is detected, thus eliminating false positives. Furthermore, because FloGuard can keep multiple detectors for the same type of attack on standby, it degrades gracefully. E.g., If the Snort rule had missed the MySQL injection attack #4 because HTTPS was used, then FloGuard would have picked the much more expensive TEMU as the detection mechanism. Such graceful degradation cannot be achieved by static deployment. For attack #2, the ClamAV detector checks only for the presence of a commonly used PHP payload. Since the actual exploit is assumed to be zero-day without a patch, and the mitigation action turns off PHP magic quotes. Doing so permanently can impair system functionality. Finally, other “detectors” such as disabling an account or quarantining a process are even more disruptive and can never be deployed permanently. But FloGuard can use them. We will add these clarifications in the prose.

We also experimented with FloGuard on a multi-step attack scenario. We deployed the Zabbix consequence detector in the target machine. First, having exploited the eVision vulnerability, we got shell access on the target system. Then, to maintain control over the system, through reboots and software patches, we tried to get the administrative



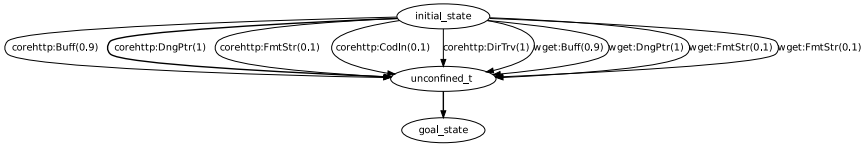
**Fig. 5.** The Generated Graphs for the Multi-step Attack

password of the system using the John the Ripper password cracker. The tool started subsequent password trials that over-consumed the system’s computational resources (96% CPU usage on average); this caused the Zabbix anomaly detector to fire an alert, making FloGuard start the forensics analysis. Due to space limits, only a portion of the generated dependency graph (which had 286 vertices) is shown in Figure 5(a). john-mmx was marked as the process causing the detection point. Figure 5(b) shows the graph, after the refinement procedures, which had a total of 7 vertices. Figure 5(c) shows the generated AGT, in which the apache2 process is the only possible entry point (initial vulnerability exploitation) for the attack. FloGuard went through the iterative forensics analysis and picked Valgrind for the first detector that was unable to detect any misbehavior; however, the second chosen detector, i.e., ClamAV, detected the downloaded file osirys.txt.gif during the PHP code execution. Consequently, FloGuard then turned off the magic\_quotes in the target machine.

Finally, we evaluated FloGuard against an updated real and well-known IRC botnet worm, namely Eggdrop [20]. We remotely launched the worm, hitting the target system as one of its victims. The worm accomplished several actions: it exploited the buffer-overflow vulnerability in the CoreHTTP web server; it downloaded, installed, and launched the Eggdrop package; the bot got connected to the EFNet IRC channel and then started listening on the tcp port 3355 to commands received through telnet from the remote attackers; later, the bot scanned the network to find the next victim machine to compromise; finally, it received remote commands trying to access some sensitive files in the system. The Samhain file integrity checker detected the file access and marked the event as the detection point. Figure 6 shows the generated AGT for the attack. On the third forensics iteration, FloGuard chose to deploy LibSafe, which successfully detected the buffer overflow violation. Consequently, LibSafe was deployed permanently to block similar attacks in the future.

## 8 Related Work

Several papers in the literature propose parts of what FloGuard achieves. However, we are not aware of any other approach that can perform on-demand, cost-aware intrusion



**Fig. 6.** The Generated AGT for Eggdrop Botnet Worm

detector deployment to defend against multi-stage attacks affecting multiple parts of a system. Several approaches perform alert correlation across multiple IDSes like [12]. Also, BotHunter [15] introduce a aggregation algorithm to recognize successful bot infections. However, both methods assume that all the appropriate intrusion detectors have already been chosen and deployed.

The concept of on-demand re-execution in a modified environment has also received some attention. Rx [26] and First-Aid [13] re-execute applications in a different execution environment, but they are not targeted towards security exploits. Bouncer [10], Vigilante [11], and Sweeper [31] combine an IDS along with program slicing or symbolic execution to trace-back from the detection point and produce input filters that can block the exploit packets. FLIPS [21] employs re-execution with instruction set randomization to detect root vulnerabilities. Shadow Honeypots [6] use re-execution in space (i.e., another machine) instead of time. However, most of these techniques only support detection of a single type of vulnerability (usually memory errors), and rely on the ability to detect attacks within the same process as the exploit entry-point. They cannot trace multi-step attacks that are detected in other parts of the system. Moreover, they do not consider multiple types of detectors and associated cost factors. FloGuard complements systems such as Sweeper by tracing detection points across multiple processes.

Attack graphs [35] have been extensively used to document known system vulnerabilities and attack paths. Two main drawbacks of the current approaches are 1) their inability to address unknown attacks, e.g., zero-day attacks, and 2) to improve scalability, their logic-based state notion does not represent system-level detailed information, significantly limiting their practical usage.

There has also been work on intrusion forensics analysis. Mukkamala et al. [22] use neural networks to discover sources of information breaches. Carrier [9] presents file-system-based forensics techniques to determine the source of security breaches by investigating their effects on the file-objects. Taser [14], BackTracker [16] and Panorama [33] aid off-line forensic analysis by producing taint-traces of file and process connections that led to a detected security breach. Because these forensics tools are based on passive data collection, they are either very pessimistic, marking most activities as malicious, or optimistic, thus missing many malicious activities that occur during an attack. In comparison, because FloGuard can actively deploy additional detection mechanisms to validate or refine its suspected attack paths, it can support much more realistic analysis. Nevertheless, pessimistic techniques that automatically produce system-level taint-graphs can be used to automatically produce initial AGTs for FloGuard.

Intrusion prevention solutions (IPS) [35] have mainly focused on how to recover from attacks after the system is compromised. Zonouz et al. [35] introduce RRE, a

game-theoretic IPS, whose goal is to take cost-optimal responsive actions against the adversary. EMERALD [25], a dynamic cooperative IPS, introduces a layered approach to correlate monitor reports through different abstract layers of the network. Unlike FloGuard, IPS solutions assume that a complete set of monitors are already deployed, and their main objective is not to identify previously unknown system vulnerabilities and exploitations to avoid identical attacks in the future.

## 9 Conclusion

We presented FloGuard, a cost-aware intrusion forensics system that uses online forensics and on-demand IDS deployment. FloGuard enables systems to defend against attacks that exploit various classes of previously unknown vulnerabilities. Our experiments show that FloGuard can deploy off-the-shelf IDSes only when they are needed and help protect systems against previously unknown vulnerabilities with minimal snapshotting overheads during normal operation.

## References

1. HTTPTrafficGen (2008), <http://www.nsauditor.com/>
2. John the Ripper (2008), <http://www.openwall.com/john/>
3. RoomPHPanning (2008), <http://www.beaussier.com/>
4. e-Vision (2009), <http://sourceforge.net/projects/e-vision/>
5. Zabbix (2010), <http://www.zabbix.org/>
6. Anagnostakis, K., Sidiroglou, S., Akritidis, P., Xinidis, K., Markatos, E., Keromytis, A.: Detecting targeted attacks using shadow honeypots. In: USENIX-Security, p. 9 (2005)
7. Baratloo, A., Singh, N., Tsai, T.: Transparent run-time defense against stack smashing attacks. In: USENIX-ATC, pp. 251–262 (2000)
8. Bellard, F.: Qemu, a fast and portable dynamic translator. In: USENIX-ATC, p. 41 (2005)
9. Carrier, B.: File System Forensic Analysis. Addison-Wesley Prof., Reading (2005)
10. Costa, M., Castro, M., Zhou, L., Zhang, L., Peinado, M.: Bouncer: Securing software by blocking bad input. In: SOSP, pp. 117–130 (2007)
11. Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou, L., Zhang, L., Barham, P.: Vigilante: End-to-end containment of internet worms. In: SOSP, pp. 133–147 (2005)
12. Debar, H., Wespi, A.: Aggregation and correlation of intrusion-detection alerts. In: Lee, W., Mé, L., Wespi, A. (eds.) RAID 2001. LNCS, vol. 2212, pp. 85–103. Springer, Heidelberg (2001)
13. Gao, Q., Zhang, W., Tang, Y., Qin, F.: First-aid: Surviving and preventing memory management bugs during production runs. In: EuroSys, pp. 159–172 (2009)
14. Goel, A., Po, K., Farhadi, K., Li, Z., de Lara, E.: The taser intrusion recovery system. In: SOSP, pp. 163–176 (2005)
15. Gu, G., Porras, P., Yegneswaran, V., Fong, M., Lee, W.: BotHunter: Detecting malware infection through IDS-driven dialog correlation. In: USENIX-Security, pp. 1–16 (2007)
16. King, S.T., Chen, P.M.: Backtracking intrusions. In: SOSP, vol. 37(5), pp. 223–236 (2003)
17. Kojm, T.: ClamAV (2009), <http://www.clamav.net/>
18. Krishnan, S., Snow, K.Z., Monrose, F.: Trail of bytes: Efficient support for forensic analysis. In: CCS, pp. 50–60. ACM, New York (2010)
19. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. ACM-Comm. 21(7), 558–565 (1978)

20. Li, C., Jiang, W., Zou, X.: Botnet: Survey and case study. In: ICICIC, pp. 1184–1187 (2009)
21. Locasto, M., Wang, K., Keromytis, A.D., Stolfo, S.J.: FLIPS: Hybrid adaptive intrusion prevention. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 82–101. Springer, Heidelberg (2006)
22. Mukkamala, S., Sung, A.H.: Identifying significant features for network forensic analysis using artificial intelligent techniques. *IJDE*, 1 (2003)
23. Nagaraja, S., Mittal, P., Yao Hong, C., Caesar, M., Borisov, N.: BotGrep: Finding P2P bots with structured graph analysis
24. Nethercote, N., Seward, J.: Valgrind: A program supervision framework. In: Runtime-Verification WS (2003)
25. Porras, P., Neumann, P.: EMERALD: Event monitoring enabling responses to anomalous live disturbances. In: Proc. of the Info. Systems Security Conf., pp. 353–365 (1997)
26. Qin, F., Tucek, J., Sundaresan, J., Zhou, Y.: Rx: Treating bugs as allergies: A safe method to survive software failures. In: SOSP, pp. 235–248 (2005)
27. Roesch, M.: Snort: Lightweight intrusion detection for networks. In: USENIX-LISA, pp. 229–238 (1999)
28. Ruwase, O., Lam, M.S.: A practical dynamic buffer overflow detector. In: NDSS, pp. 159–169 (2004)
29. Schneier, B.: Attack trees. *Dr. Dobbs's Journal* (1999)
30. Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M.G., Liang, Z., Newsome, J., Poosankam, P., Saxena, P.: BitBlaze: A new approach to computer security via binary analysis. In: Sekar, R., Pujari, A.K. (eds.) ICISS 2008. LNCS, vol. 5352, pp. 1–25. Springer, Heidelberg (2008)
31. Tucek, J., Newsome, J., Lu, S., Huang, C., Xanthos, S., Brumley, D., Zhou, Y., Song, D.: Sweeper: A lightweight end-to-end system for defending against fast worms. *EuroSys* 41(3), 115–128 (2007)
32. Wotring, B., Potter, B., Ranum, M., Wichmann, R.: Host Integrity Monitoring Using Osiris and Samhain. Syngress Publishing (2005)
33. Yin, H., Song, D., Egele, M., Kruegel, C., Kirda, E.: Panorama: Capturing system-wide information flow for malware detection and analysis. In: CCS, pp. 116–127 (2007)
34. Zonouz, S.A., Joshi, K.R., Sanders, W.H.: Cost-aware systemwide intrusion defense via on-line forensics and on-demand detector deployment. In: CCS-SafeConfig, pp. 71–74 (2010)
35. Zonouz, S.A., Khurana, H., Sanders, W.H., Yardley, T.M.: RRE: A game-theoretic intrusion Response and Recovery Engine. In: DSN, pp. 439–448 (2009)

# Reducing Complexity of Data Flow Testing in the Verification of a IEC-62304 Flexible Workflow System

Federico Cruciani<sup>1</sup> and Enrico Vicario<sup>2</sup>

<sup>1</sup> I+ s.r.l., Piazza Puccini ,26 - 50144 Florence, Italy  
f.cruciani@i-piu.it

<sup>2</sup> Università degli Studi di Firenze

**Abstract.** In the development of SW applications, the workflow abstraction gives primary relevance to the way how some process can be accomplished through a sequence of connected steps. This largely conditions analysis, implementation architecture, and verification. In particular, testing activities are naturally oriented towards a data flow approach, which effectively exercises dependencies among steps. In several application scenarios, the workflow model cannot completely determine the sequencing of actions and it must rather leave space to variability. While easily encompassed both in the analysis and implementation stages, this comprises a major hurdle for the testing stage due to the explosion in the number of allowed execution orders and paths.

We address the problem reporting on the verification of the control software of a Computer Assisted Surgery system. In this case, the workflow abstraction captures the constraints of a medical protocol, and variability in the order of steps reflects dynamic adaptation of the course of actions to the specific characteristics of each patient. This largely increases the testing effort needed to accomplish the prescriptions of the IEC-62304 certification standard. To cope with the problem, we show how data flow analysis can be used to identify an appropriate set of constraints that can be exploited in the verification stage, so as to reduce the test suite while preserving coverage.

**Keywords:** Workflow architecture, Workflow verification, Data Flow testing.

## 1 Introduction

Workflow applications are commonly used to automate processes that require a sequence of steps to be performed in a certain order to complete a task[2]. Typical areas of application include business process management, manufacturing and supply management, healthcare protocols. In general, a workflow application can be conveniently implemented by letting an executable specification of a process be enacted by an operational engine. This enables consistent reuse of the engine among multiple applications, facilitates evolutionary maintenance



of processes, and centers the overall SW life cycle on a process-oriented model that can be effectively agreed among software engineers, stakeholders, and users. In so doing, the overall development can be accompanied by effective tools including UML Activity diagrams supporting representation [3][4], formal models providing theoretical foundation [5], and frameworks providing a basis for effective enactment [6].

Various testing strategies for workflow applications have been proposed, mainly relying on the automated derivation of a Control Flow Graph (CFG) abstraction [6][7] that drives test case selection or coverage analysis. CFGs can be derived directly from workflow specifications such as UML Activity diagrams [4][3]. A data flow model can be built from the requirements and this model can conveniently be exploited also in formal verification, following a kind of model-based approach to design develop, and test the system. Model verification of workflows allows indeed the early detection of sequencing problems such as deadlock or non terminating behaviors [8].

In the transition stage of the SW life cycle, and in particular in the acceptance step, the process model provides a native abstraction enabling application of the consolidated theory of data flow testing in functional perspective [9]. In particular, according to all-uses criterion, the workflow is operated so as to exercise at least one path from each step where some relevant variable is modified to each the next step where the same variable is used. Though not prescribed by certification standards, this criterion has proven to effectively increase fault coverage capability with respect to branch coverage [10] while maintaining the testing effort in the range of polynomial complexity.

However, when testing comes to input generation and test execution, full coverage of the all-uses criterion still remains a complex task, especially when execution involves user interfaces and physical system devices. The complexity is further exacerbated whenever the automated process is not completely determined and rather enables multiple orders of execution determined during the run-time, resulting in a so-called flexible workflow [11][12]. This case is relevant for any procedure where some of prescribed actions can be executed in different orders without compromising the integrity of the overall process. In the end, this kind of flexibility often serves to smooth the stiff mechanism of workflow applications, which work finely in setting rules more than accommodating exceptions. This takes a specific relevance in the healthcare context, where a crucial role is played by dynamic adaption of the execution order of a protocol according to the course of actions applied to a specific patient.

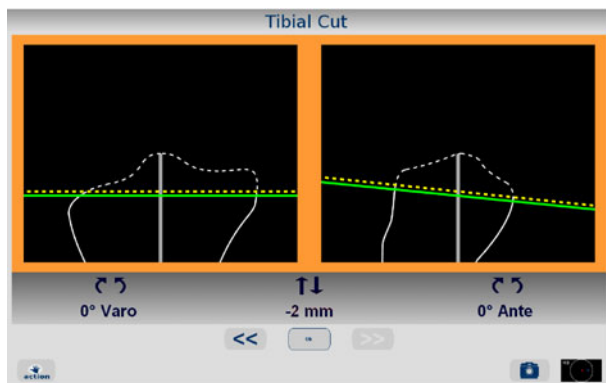
In this paper, we report on testing activities performed in the verification of the control software of a Computer Assisted Surgery system subject to ISO-62304 standard for medical software [1]. The SW under test is organized as a flexible workflow application that accompanies the steps of a surgical protocol. During the certification process, in order to compensate lacks in the structural coverage of component items, integration testing was planned and executed so as to attain all-uses coverage of the workflow specification. In doing so, the complexity of the test plan could yet be limited through the assumption of design choices

that statically guarantee equivalence conditions among different paths and thus permit a substantial simplification of the test suite while preserving its coverage capability.

The rest of the paper is organized as follows. The characteristics of the application case and its testing requirements are described in Sect. 2. In section 3, we introduce a model abstracting data flow behavior of a workflow in a high-level Directed Acyclic Graph (DAG) showing how this can be extracted from a specification accepting multiple execution orders, and how this reduces testing complexity in the application case. Conclusions are drawn in Section 4.

## 2 Testing Requirements for a Computer Assisted Surgery System

Miró<sup>1</sup> is a workflow application for Computer Assisted (image guided) Surgery applied to knee arthroplasty. The software is based on BLU-IGS, an ad-hoc workflow engine optimized for orthopedical surgical procedures that supports a product line of softwares for hip arthroplasty, kinematic analysis, knee prosthesis implant and revisioning.



**Fig. 1.** A screenshot of Miró during the navigated execution of tibial cut. The dotted line indicates the planned cutting plane while the continuous line represent the actual position of the cutting guide.

Miró integrates an infrared optical localizer of surgical tools, used to track position of fiducials markers placed to the patient's bone, so as to provide a reference system through which the anatomy of the patient limb is then reconstructed by means of registration of anatomical points through a pointer tool. Data acquired through the pointer are then used to build the anatomical reference systems for tibia and femur allowing an intra-operative planning of the

<sup>1</sup> Miró is developed by I+ s.r.l. as part of the BLU-IGS system distributed by Orthokey LLC: <http://www.orthokey.com/index.php/totalknee>.

final prosthesis implant. Surgical tools are then referred with respect to these anatomical reference systems and navigated during the operation to verify the correct positioning of the tibial and femoral components.

The control software accompanies the surgeon along a workflow protocol composed by *phases* or *steps* including tibial and femoral registration, implant positioning planning, navigation of femoral tibial cut. Many of these phases determine choices that condition the subsequent steps (e.g. the selection of the type of prosthesis to implant). The process is composed by data acquisition and operational steps strictly bound to the conventional surgical protocol. As in most existing competitors products, the current version of Miró follows a fixed execution order, forcing the surgeon to adapt his way of performing the operation to the built-in procedure. This lack of flexibility now appears to be a major hurdle for the adoption of Computer Assisted Surgery, despite this has been demonstrated to improve the quality of results, in particular in restoring the neutral alignment of the leg [13] [14] [15]. To cope with this issue, the new BLU-IGS version is moving towards a more flexible workflow engine, allowing the surgeon to vary the execution order, either to best fit his way of operating, or according to the patient anatomy. Some of the phases in the procedure are constraining, e.g. the type of prosthesis to implant. In the fixed workflow solution, this choice has to be performed at the initial stage of the operation, thus determining the behavior of some of the subsequent steps. Flexibility in this case would allow to postpone constraining choices to a latter phase of the operation, when the anatomy of the patient is more clear and the choice can leverage on a more complete understanding of the case.

## 2.1 Testing Requirements for IEC 62304

The international standard IEC 62304 specifies life cycle requirements for the development of medical software and software within medical devices [1]. A Crucial element of the standard is the concept of design for patient safety, for which a fundamental role is played by risk analysis (including hazard identification), evaluation, and control.

In the case of Miró, the attainment of this objective had to face the infamous (yet common in real practice) problem of components classified as Software Of Unknown Provenance (SOUP). Specifically in this case, these are software items integrated in the overall 20000 lines of C++ code of Miró, that had been already developed and generally available but that had not been developed for the purpose of being incorporated into a medical device subject to certification requirements. In order to compensate the presence of SOUP components with non-compliant structural coverage of unit tests, a higher responsibility and effort was charged on integration testing. Specifically, a grey-box testing approach was planned (and agreed in the certification process), with the goal of covering all the interactions among components for which unit testing coverage had not been attained. To this end, the test plan was targeted to attain def-use coverage (for each definition of any variable of global scope, cover *at least* one path reaching each subsequent use of that variable) and all du-paths (for each definition of any

variable of global scope, cover *each* path reaching each subsequent use of that variable) [9].

Disciplined reasoning on UML activity diagrams of the basic process identifies 3 paths that are sufficient to cover the def-use criterion and 9 sufficient to ensure all-du paths. However, the addition of flexible choices to the basic process dramatically increases the set of feasible executions and the set of combinations among definitions and uses of data coupling different steps of the protocol. In principle, since the process consists of 13 steps, this results in  $13!$  possible orders of execution. Fortunately enough, the protocol includes dependencies between states that constrain feasible executions (e.g., the navigation of tibial cutting guide can not be executed before the registration of tibial reference system). Despite the reduction, this still results in 96 test cases, which subtends a huge testing and documentation effort due to at least two major factor exacerbating complexity. On the one hand, functional tests on the integrated system must be manually performed by simulating a complete intervention for each path, with a significant effort also in the generation of input data for each path. On the other hand, due to accuracy requirements on measurements taken by the system during the surgical protocol, the oracle verdict on the results of each single test requires that device measurements and numerical processing be shown accurate up to 1mm and  $1^\circ$  to get benefits compared to conventional procedure.

This type of complexities suggested that the code be partially refactored so as to implement a few basic design-for-testability principles that could permit a significant reduction in the complexity of the Test Plan. In this particular scenario, each variable, either a cutting plan, an anatomic reference system or a point acquired during registration, is bound to be defined only in a single step where its value is acquired or calculated. Its value will then be used by some other subsequent steps, but, the only portion of code where the value can be modified remains the step where the value is assigned. We are interested in testing the IUT with all-uses coverage [9] of the behavior model specified in functional requirements. In so doing, we guarantee that behaviors allowed by the functional specification are tested so as to cover all-nodes, all-edges and also a relevant subset of all paths. Specifically, paths are covered according so as to guarantee that for each variable  $x$  defined in some step  $X$  and later used in some step  $Y$  without any intermediate side-effect on  $x$ , at least one test is performed that reaches  $Y$  from  $X$  without ever modifying  $x$ . We will show that in this case, the data flow testing applied in functional perspective, not only detects an effective set of paths to test, but also helps, adding some appropriate constraints, to keep under control the variability introduced by the execution of the process in a flexible workflow.

### 3 Abstraction and Problem Formulation

It is worth in this context, to exactly define and classify the type of workflow we are working on. From a data flow perspective each variable has an only point in which is defined and this can be verified by static inspection of the code.

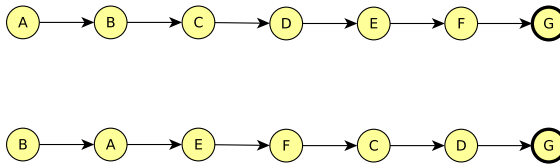
There are two kinds of variables involved: some variables are used to calculate or measure quantitative data, and some variables that correspond to choices, for instance the type of implant the surgeon decide to use. This second set of variables, in other words, affects a subset of steps where based on their values one of several branches is chosen within the execution of the step itself, i.e. some steps implement different behaviors according to some pre determined condition. As for the variables we have thus two kind of steps: some steps can be viewed as a basic block while some other steps hide execution branches. However, we can consider each step in the procedure as an *extended basic block*, i.e. a sequence of consecutive instructions always executed from start to finish that may contain branches.

We also assume that a set of dependency between states can be derived by static analysis of the process.

The surgical process is composed by a set of steps  $S$  that must all be performed but can be serialized in any way satisfying a given partial order  $\prec \subseteq S \times S$  reflecting the constraints of the surgical protocol.

Our workflow can thus be defined as  $W = \langle S, \prec \rangle$  where  $S$  is the set of possible phases that can be executed in different combination in the process, and  $\prec$  is a set of high-level dependencies among steps.

As an example, be  $S$  the workflow composed by the set of steps  $S = \{A, B, C, D, E, F, G\}$ . The control flow across nodes is conditioned by a set of variables  $V = \{a, b, c, d, e, f, g\}$  with global scope, i.e. variables that are defined and used in different activities.



**Fig. 2.** Two possible execution order of the process

The dependency between steps implies that some process constraints have to be introduced to avoid sequencing problems i.e. the use in a particular step of a variable that has not yet been assigned.

### 3.1 Data Flow Perspective

In order to define the set of constraints we need to analyze all steps in the workflow model specification. We need *order constraints* [16] [12] to avoid the execution of a step  $N$  which uses a variable that is defined in step  $M$  prior to the execution of  $M$ . This kind of constraints indicates that two steps have to be executed in an exact order but could have other independent workflow steps in between. This *dependency* relation between two steps can be naturally defined by using the data flow perspective.

As in [9], it is here convenient to distinguish among c-uses and p-uses, i.e. references to variable that condition the value assigned to some other variable (computational use) or the the result of a decision determining the flow of actions in the control flow graph (predicate use), respectively. In particular, in our setting, a p-use can determine the choice among different ways how some activity is performed. For instance, the selection of a type of prosthesis in some activity  $A$  might define a global variable  $a$ , which in turn is later p-used in some activity  $F$  to select the way how some measure is taken. This results in different modes of execution for  $F$ , say  $\{F_1, F_2, F_3, \dots\}$ , which may define different variables, say  $\{f_1, f_2, f_3, \dots\}$ . It may also happen that in some subsequent activity  $G$ , the same variable  $a$  is p-used again to select among different modes  $\{G_1, G_2, G_3, \dots\}$  each of which uses in respective manner the variables  $\{f_1, f_2, f_3, \dots\}$  defined in  $F$ .

This setting directly reflects explicit needs of the context of use, and it is quite easily implemented in a workflow oriented SW architecture. However, a major hurdle for its practical realization arises in the testing stage. In fact, attaining all-def coverage for this variety of behaviors is by far beyond the limits of a feasibility. And, relaxation of the aim from all-uses to all-edges does not substantially change the nature of the problem. In fact, tests are here performed at the system level, and each of them requires manual application of a sequence of physical steps, which basically reproduce those of a surgical operation. Just to give an idea of the order of complexity that can be reasonably afforded, in the certification of the first release of the product which did not include workflow flexibilities, the test suite specified in the test plan was made by 9 cases.

In order to complete the specification of this flexible workflow we need to introduce some definitions.

**Definition 1.** Let  $M, N$  be two steps of the procedure and  $adu(M), adu(N)$  the corresponding sets of all definitions and uses of variables. We will say that  $N$  depends on  $M$  if  $\exists$  any variable  $x$  thus that  $def(x) \in adu(M)$  and  $p-use(x) | c-use(x) \in adu(N)$ .

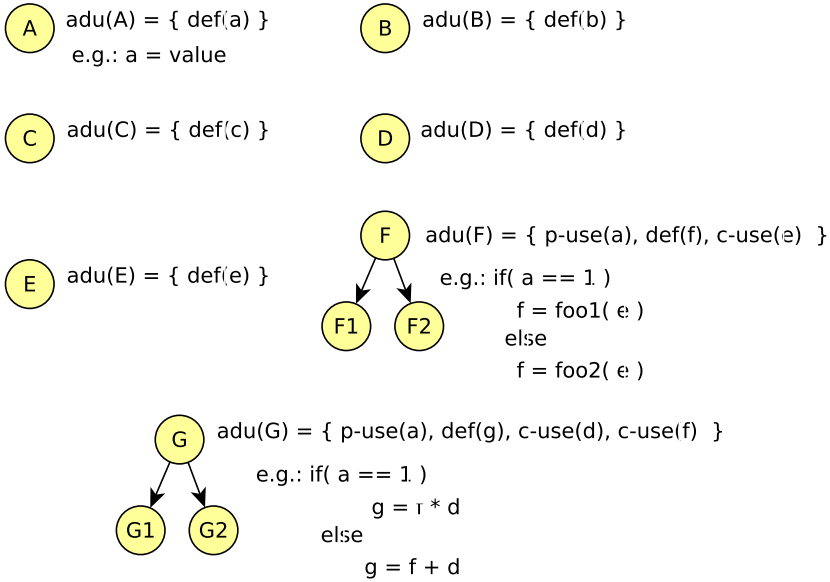
**Definition 2.** Let  $M, N$  be two steps and  $adu(M), adu(N)$  the correspondents sets of definitions and uses of variables and  $V(N)$  and  $V(M)$  the corresponding set of used or defined variables, we will say that  $N$  and  $M$  are independent if  $V(M) \cap V(N) = \emptyset$ .

**Definition 3.** A Dependency Graph is a pair  $DG = \langle S, \Gamma \rangle$  were  $S$  is a set of vertexes each of them representing a step, and  $\Gamma$  is a set of pairs  $(x, y) \in S^2$  called set of directed edges between two vertexes each of them representing a dependency between the two steps.

Looking at this problem in data flow perspective we can define for each possible state which variables are involved and how [6].

We are going to define for each state  $X$  the set of all *defs* and *uses* denoted as  $adu(X)$  and we will consider each step, from a data flow perspective, as a *basic block* [9] [17].

Predicate uses within a step cause that state to be split in parts:  $F$  and  $G$  have been split in  $F, F1, F2$  and  $G, G1, G2$ . This is based on the concept of *extended*



**Fig. 3.** The adu sets for the example workflow. For each step the set of all definitions and uses of variable is reported.

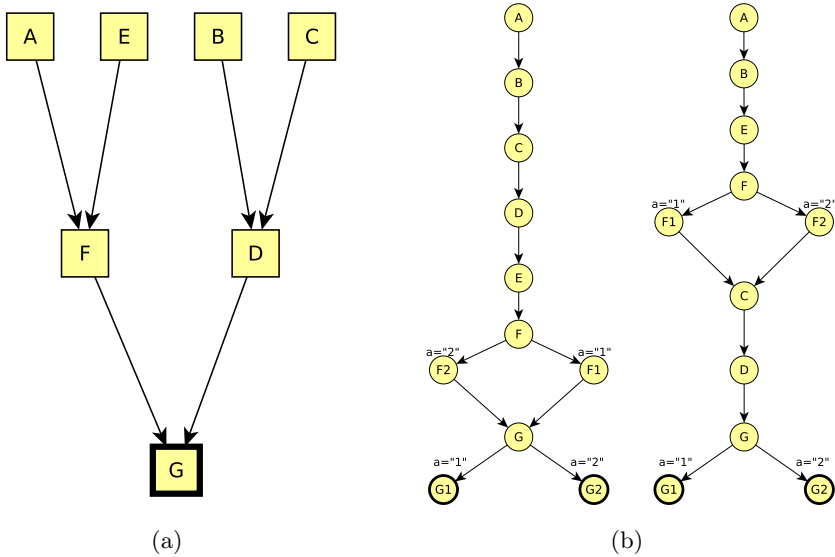
*basic block.* Note that execution of  $F1$  or  $F2$  within the step  $F$  is implicitly determined in the definition of variable  $a$  and does not correspond to a surgeon choice during the step  $F$ . The only choice here would be the execution of  $F$  rather than any other eligible step.

These two steps in the example behave differently according to the value of  $a$  that, in this example can have two values 1 or 2, anyway both the steps can be considered as extended basic block. In fact the execution flow within the block is deterministic.

We can represent the set of order constraints in a DG as in fig 4(a). Any possible order of execution can be obtained by picking nodes from the DG respecting all dependencies from other step which have not yet been executed. We show two possible CFGs in fig 4(b). For each graph we can see that there are two feasible paths. Both these CFGs respect the set of order constraints explicitly defined by the data flow analysis of each step.

### 3.2 Reducing Complexity through Design for Testability

To reduce the testing effort, development was inspired to general principles of design for testability [18], and in particular to the usage of design patterns that support of effective and efficient verification of functional assumptions through static inspection of code architecture [19] [20]. Note that, in so doing, the structure of implementation is conditioned to functional testing objectives.



**Fig. 4.** In (a) is represented the dependency DAG based on the set of constraints generated by data flow analysis. In (b) there is an example of two among the many possible control flow graphs that can be obtained respecting the order constraints. Note that the control flow graphs represent two possible run time execution of the process in which state F and G have been split considering the run time value of variables involved in the corresponding *p - use* in F and G. For each CFG, or in other words for any legal execution order would require at least two test cases executing the two possible path on the graph.

In particular three major assumptions were supported through adequate and verifiable choices in the implementation structure.

- The structure of implementation of the workflow model was implemented using the BLU-IGS engine. In so doing, the workflow is explicitly encoded into a set of states and a set of dependency rules, whose consistency with the expected specification can be supported by static code inspection.
- Each variable  $a \in V$  was restrained to be defined within a single activity of the process:

$$\forall A, B \in S, \forall a \in V, a \in def(A) \wedge a \in def(B) \rightarrow A = B$$

This constraint was enforced at design level, by making each activity be a class and each global variable be a private attribute of the class were it is defined with public get methods and private set method.

- Consistency in subsequent choices subordinated to the p-use of the same global variable is guaranteed through the verification phase since any inconsistency would result on a test failure caused by missing data or incorrect values.



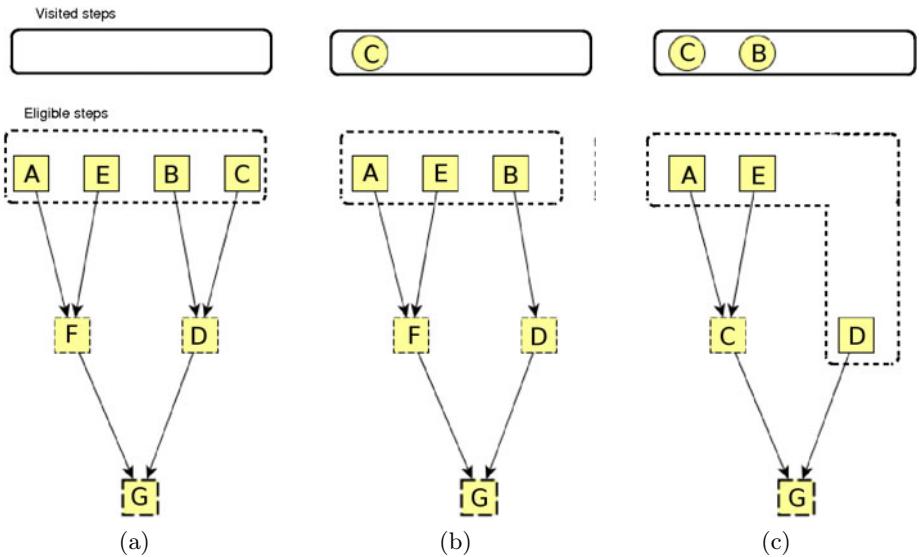
- The choice of a particular step during the procedure is constraining, i.e. it is not possible to re-execute an already visited operational phase. This assumption is not restrictive in this case, since for instance it has no sense to re-acquire a registration point after the cut has been executed.

### 3.3 Reducing Complexity through Test Equivalence

Under the assumptions enforced at design level, the test suite that guarantees all-uses coverage can be drastically reduced as most test cases turn out to be equivalent.

The dependency DAG defines a partial order between states implicitly defining an equivalence class of CFGs. Based on the example dependency DAG there are 76 possible graphs, or in other words, 76 ways to complete the process without breaking any order constraint. Considering the data flow analysis that would make  $76 \times 2$  possible du-paths.

Based on this partial specification, any order in which we complete the task without breaking any constraint can be obtained by picking a node at a time from the dependency DAG following the rule that we can pick any node that does not depends on other node in the DAG. Explored nodes are removed from the dependency DAG as showed in fig 5.



**Fig. 5.** An example of execution of three steps (a),(b) and (c) on the dependency DAG. In the first step node C is selected, followed by selection of B and C. Once the third step is complete there are three eligible nodes as next step: A, E and D. Steps F and G are not eligible until their dependencies are verified. Note that dependency DAG defines no relation between nodes in the eligible set.

The workflow engine must simply enable the choice of a subset of steps that are eligibles based on a queue of already explored steps and the set of all dependencies.

Any choice made is thus guaranteed to be a valid workflow since all dependencies are respected. Plus all the possible ways of choice order are equivalent since, at any decision, only a subset of states is eligible. All nodes in that subset are *independent* guaranteeing that the order in which steps are performed is equivalent. A similar concept is the one applied in the case Out-Of-Order Execution, where data flow information is used in order to optimize the CPU resources usage, in any case the data flow analysis detect different execution order that do not affect the final result of computation.

Let be  $S_1 \rightsquigarrow S_n$  the sequence of explored states, and be  $ES = S_{i1}, S_{i2} \dots, S_{im}$  the set of eligible states, any sequence  $\forall S_{ix} \in ES$  the sequence  $S_1 \rightsquigarrow S_n \rightarrow S_{ix}$  is equivalent.

Expanding this concept in terms of equivalence between du-paths, let be  $S_i$  the step that contains the definition of a variable  $i$ , and let be  $S_j$  a subsequent step where the value of  $i$  is used. Let be  $S_i \rightsquigarrow S_x \rightsquigarrow S_j$  an execution order that execute the du-path between  $S_i$  and  $S_j$  with respect to variable  $i$ . For the hypothesis of non-interference,  $S_i$  is the only portion of code where the value of  $i$  is modified, meaning that any step  $S_x$  executed between  $S_i$  and  $S_j$ , either is *independent* from  $S_i$ , or contains a use of  $i$ , but for sure will not modify its value. Therefore, a test case executed in this path, would produce the same result as in any other possible execution path  $S_i \rightsquigarrow S_y \rightsquigarrow S_j$ .

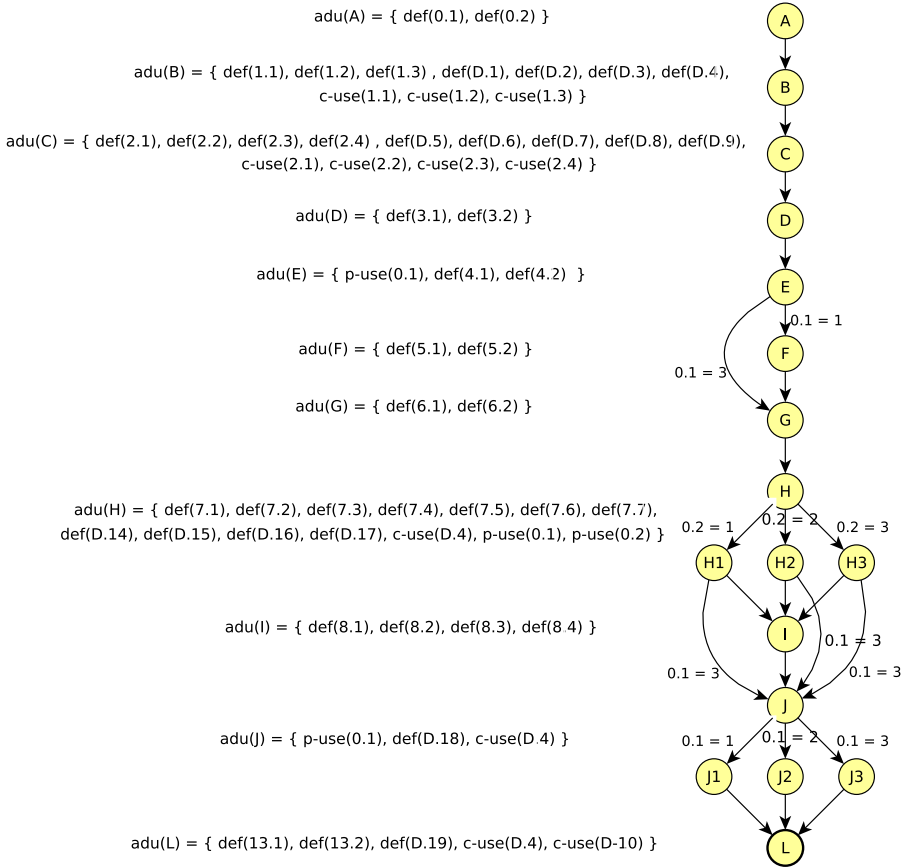
In conclusion, given any of the CGFs that respects all data dependencies, it is possible to build a test suite covering all-du paths, that produce the same coverage of all-du paths in any other order of execution.

### 3.4 Application to Our Case Study

In the application to the case of the Miró system, the equivalence between the possible CFGs allows to build an equivalent test suite based on one on the many possible orders of execution. In this case the results is far more relevant, since it means that the same test suite, and also the corresponding data set for verification, built for a fixed-workflow scenario is still valid. Fig. 6 illustrates the abstraction applied to Mirò.

Analyzing the CFG, there are only two variables for which there is at least a p-use in some step. This variables are indicated as 0.1 and 0.2 and consist respectively on, the choice of the prosthesis, and the type of acquisition to use as a reference to evaluate the correct positioning of the femoral prosthesis component. Both the variables can assume 3 values, leading to a minimal test suite built on 3 path for all-uses and 9 paths for all-du paths using the two variable in all the possible combinations. The use of this test suite ensures the execution of all possible behaviors also on SOUP items where unit test level does not provide any form of verification.

Even though, the use of computer assisted surgery is increasing, it still remains a lower percentage compared to traditional technique, and that due also to its



**Fig. 6.** Miró Integration Test Plan

lack of flexibility. The Total Knee arthroplasty surgical procedure comprise the execution of cuts both to the tibia and the femur, surgeons are used to perform these two phases following one particular order better than the other. A flexible procedure supports surgeons with this choice, making them more confident when passing from the conventional technique to a computer assisted procedure, without the need of modifying their way of conducting the operation.

## 4 Conclusions

In our case, it has been possible, following this approach, to reuse the integration test suite detected in the previous software version. In this way, any increase in terms of cost for integration test execution has been avoided, adding on the same time the flexibility feature. This advantage is particularly relevant, when considering that integration test phase is the most complex verification phase in this context.

Most of the effort has been put instead, on verifying the aforementioned assumptions about non-interference of procedure steps on the variables, while at the level of unit test a test suite has been introduced to test the new workflow engine, testing the engine on accepting or reject all the possible execution orders.

The last point to verify was the assumption that we can consider a single step as a basic block. This assumption is indeed a big limitation in the case of the registration step, where the set of anatomical points are acquired to build the anatomical reference system. At this level, flexibility allows the surgeon to re-acquire any of the registration points without the need of re-acquiring all values. The use of design patterns oriented for testability can help also to remove the assumption that within the same step a variable can not be re-defined. In other words this would imply the redefinition of a variable  $x$ , that can have been already c-used in the same step to compute a variable  $y$ , causing the risk of this second variable to be not correctly updated. The information about the dependency between data related to each step, can be used to automatically update all dependent data. This automatic update, has been guaranteed by using an extension of the observer pattern [21], where all variables are encapsulated in a data class, able to notify changes on its internal state and to observe notification from other data object in the same step.

Even though the assumptions, valid in this particular study case, appear to be restrictive, we believe that this kind of approach can be extended to cover more general workflow specifications. Flexibility is a common issue in many kind of workflow applications [11], and this approach can easily be applied to any domain in which a similar workflow modeling is suitable.

In conclusion, this case study reports on how, combining the right integration testing approach, in our case the data flow testing, with some elements oriented to design for testability it is possible to implement major changes on a certified software minimizing the cost of the verification process.

## References

1. International Electrotechnical Commission: Medical device software - Software life-cycle processes, IEC62304:2006 (2006)
2. Georgakopoulos, D., Hornick, M., Sheth, A.: An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases* 3(2), 119–153 (1995)
3. Fowler, M., Scott, K.: *UML distilled* (3rd ed.): a brief guide to the standard object modeling language, 3rd edn. Addison-Wesley Professional, Reading (September 25, 2003) ISBN:978-0321193681
4. Dumas, M., ter Hofstede, A.H.M.: UML activity diagrams as a workflow specification language. In: Gogolla, M., Kobryn, C. (eds.) *UML 2001*. LNCS, vol. 2185, pp. 76–90. Springer, Heidelberg (2001)
5. Van Der Aalst, W.M., Ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: *Workflow Patterns*. *Distributed and Parallel Databases* 14(1), 5–51 (2003)
6. Mei, L., Chan, W.K., Tse, T.H.: Data flow testing of service oriented workflow applications. In: *ICSE 2008 Proceedings of the 30th International Conference on Software Engineering* (2008)

7. Mei, L., Chan, W.K., Tse, T.H.: An empirical study of the use of Frankl-Weyuker data flow testing criteria to test BPEL Web services. In: 33rd Annual IEEE International Computer Software and Applications Conference (2009)
8. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
9. Rapps, S., Weyuker, E.J.: Selecting test data using data flow information. *IEEE Transactions on Software Engineering* SE-11(4) (April 1985)
10. Frankl, P.G., Weiss, S.N.: An experimental comparison of the effectiveness of branch testing and data flow testing. *IEEE Transactions on Software Engineering* 19(8) (August 1993)
11. Sadiq, S.W., Orlowska, M.E., Sadiq, W.: Specification and validation of process constraints for flexible workflows. *Information Systems* 30(5), 349–378 (2005), doi:10.1016/j.is.2004.05.002, ISSN 0306-4379
12. Sadiq, S.W., Orlowska, M.E., Lin, J., Sadiq, W.: Quality of Service in flexible workflows through process constraints. In: *Enterprise Information Systems*, vol. VII, part 3, pp. 187–195 (2006)
13. Schep, N.W.L., Broeders, I.A.M.J., van der Werken, C.: Computer assisted orthopaedic and trauma surgery: State of the art and future perspectives. *Original Research Article Injury* 34(4), 299–306 (2003)
14. Saragaglia, D., Picard, F., Chaussard, C., Montbarbon, E., Leitner, F., Cinquin, P.: Computer-assisted knee arthroplasty: comparison with a conventional procedure. Results of 50 cases in a prospective randomized study. *Rev Chir Orthop Reparatrice Appar Mot.* 87(1), 18–28 (2001)
15. Bathis, H., Perlick, L., Tingart, M., Luring, C., Zurakowski, D., Grifka, J.: Alignment in total knee arthroplasty, A Comparison of Computer-Assisted Surgery with the Conventional Technique. *Journal of Bone and Joint Surgery - British* 86-B(5), 682–687
16. Sadiq, S.W., Orlowska, M.E., Sadiq, W., Foulger, C.: Data Flow and Validation in Workflow Modelling. In: *ADC 2004, Proceedings of the 15th Australasian Database Conference*, vol. 27 (2004)
17. Binkley, D., Gallagher, K.B.: *Program Slicing, Advances in Computers*, vol. 43, pp. 1–50. Academic Press, London (1996)
18. Binder, R.V.: Design for testability in object-oriented systems. *Commun. ACM* 37(9), 87–101 (1994), R 10.1145/182987.184077
19. Baudry, B., Le Sunyé, Y., Jézéquel, J.-M.: Towards a 'Safe' Use of Design Patterns to Improve OO Software Testability. In: *Proceeding ISSRE 2001 Proceedings of the 12th International Symposium on Software Reliability Engineering*. IEEE Computer Society, Washington, DC, USA (2001), table of contents ISBN:0-7695-1306-9
20. Baudry, B., Le Traon, Y., Sunyé, G.: Testability Analysis of a UML Class Diagram Software Metrics. In: *IEEE International Symposium on Eighth IEEE International Symposium on Software Metrics (METRICS 2002)*, p. 54 (2002)
21. Gamma, Helm, Johnson, Vlissides: *Design Patterns, Element of Reusable Object-Oriented Software*, 1st edn. Addison-Wesley, Reading (1995)

# Improvement of Processes and Methods in Testing Activities for Safety-Critical Embedded Systems

Giuseppe Bonifacio, Pietro Marmo, Antonio Orazio,  
Ida Petrone, Luigi Velardi, and Alessio Venticinquè

AnsaldoSTS, via Argine 425, 80147 Napoli, Italy  
{Giuseppe.Bonifacio.interim,Pietro.Marmo,  
Antonio.Orazio,Ida.Petrone.Prof423,Luigi.Velardi,  
Alessio.Venticinquè.Prof202}@ansaldo-sts.com

**Abstract.** In order to sustain competitiveness in transport domain, especially in automotive, aerospace and rail, it is extremely important to control and optimize the entire development process of complex safety-critical embedded systems. In this context, the ARTEMIS EU-project CESAR<sup>1</sup> (Cost-Efficient methods and processes for SAFETY Relevant embedded systems) aims to boost cost efficiency of embedded systems development, safety and certification processes by an order of magnitude. We want to achieve the above target in the railway domain with particular emphasis on the Verification and Validation (V&V) process where activities to be performed, due to their complexity, require a significant amount of economical resources. Starting from an industrial use case (the On-Board Unit of the European Railway Traffic Management System Level 1, ERTMS L1) we provide a methodology that overcomes some weaknesses in testing processes. It supports requirements analysis and automatic test cases generation, avoiding a computational explosion.

**Keywords:** Testing, Safety, Requirements engineering, Ontology, V&V.

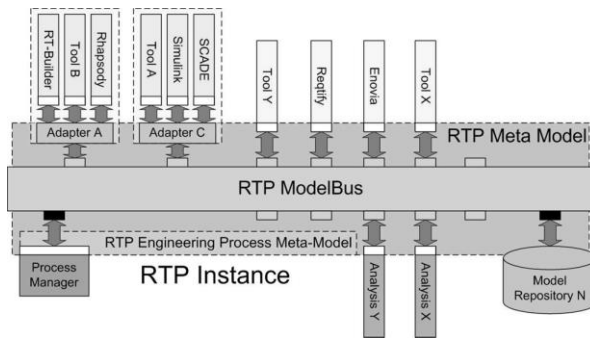
## 1 Introduction

The embedded safety-critical systems design and development industry is facing increasing complexity and variety of systems and devices, coupled with increasing regulatory constraints while costs, performances and time to market are constantly challenged. This has led to a profusion of enablers (new processes, methods and tools), which are neither integrated nor interoperable because they have been developed more or less independently, addressing only a part of the complexity issue, such as safety. The absence of internationally recognized open standards is a limiting factor in terms of industrial performance when companies have to select among these enablers. The EU-project CESAR will bring significant and conclusive innovations in the two most improvable systems engineering disciplines such as Requirements Engineering [1], in particular through formalization of multi viewpoint, multi criteria

---

<sup>1</sup> <http://cesarproject.eu/>

and multi level requirements and Component Based Engineering applied to design space exploration comprising multi-view, multi-criteria and multi level architecture trade-offs. CESAR intends to provide industrial companies with a breakthrough in system development by deploying a customizable systems engineering Reference Technology Platform (RTP) making it possible to integrate or interoperate existing or emerging available technologies. The RTP is composed by various tools integrated in the RTP bus in order to provide a complete environment that covers the phases of design system, from the system conception and requirement capturing to the system realization. The RTP architecture is shown in Fig. 1.



**Fig. 1.** RTP architecture

This will be a significant step forward in terms of industrial performance improvement that will help to establish de-facto standards and contribute to the standardization effort from a European perspective. Relying on use-cases and scenarios from Aerospace, Automotive, Automation and Railway, CESAR is strongly industry driven. Benefiting from this multi-domains point of view, CESAR addresses safety aspects of transportation and other societal mobility and environmental demands. Some key needs identified by AnsaldoSTS (from now on, ASTS) driving CESAR calls for an innovation boost in particular related to Verification and Validation (V&V) activities, regulated by international standards (see [2]-[8]), starting from requirements specifications expressed by system stakeholders, adapted from standard documents or re-used from previous projects. In this work we present the ASTS use case that will contribute to the CESAR objectives providing the assessment of RTP applicability to perform the functional testing activities. This use case, even if it reflects the industrial “state of the practice” of these tests in the rail domain, aims to reveal technical gaps that should be closed by one or more technical innovations and its applicability can be easily extended to other domains.

### 1.1 ERTMS Level 1 and ASTS Pilot Application

ERTMS, the European Railway Traffic Management System, has been designed by the European railways and the supply industry supported by the European Commission to meet the future needs of the European Railways. The deployment of

ERTMS will enable the creation of a seamless European railway system and increase European railway's competitiveness. ETCS (European Train Control System) is an ERTMS basic component: it is an Automatic Train Protection system (ATP) to replace the existing national ATP-systems. UNISIG is an industrial consortium which was created to develop the ERTMS/ETCS technical specifications and actively contributes to the activities of the European Railway Agency in order to assure the "interoperability" that is the main driver for ERTMS in the context of the European Railway Network. The meaning of "interoperability" is two-fold: on the one hand, it refers to a "geographical interoperability" between countries and projects (a train fitted with ERTMS may run on any other ERTMS-equipped line); on the other hand, it also refers to a technical notion of "interoperability between suppliers" (a train fitted by a given supplier will be able to run on any other trackside infrastructure installed by another supplier). This opens the supply market and increases competition within the industry. The ERTMS/ETCS application has three "levels" that define different uses of ERTMS as a train control system, ranging from track to train communications (Level 1) to continuous communications between the train and the Radio Block Centre (Level 2). Level 3, which is in a conceptual phase, will further increase ERTMS' potential by introducing a "moving block" technology. This Pilot Application (PA) deals with ERTMS Level 1.

ERTMS Level 1 is designed as an add-on to or overlays a conventional line already equipped with line side signals and train detection equipment which locates the train.

ERTMS Level 1 has two main sub-systems:

- Ground sub-system: collects and transmits track data (speed limitations, signal-status, etc.) to the on-board sub-system;
- On-board sub-system: analyzes data received from the ground and elaborates a safe speed profile.

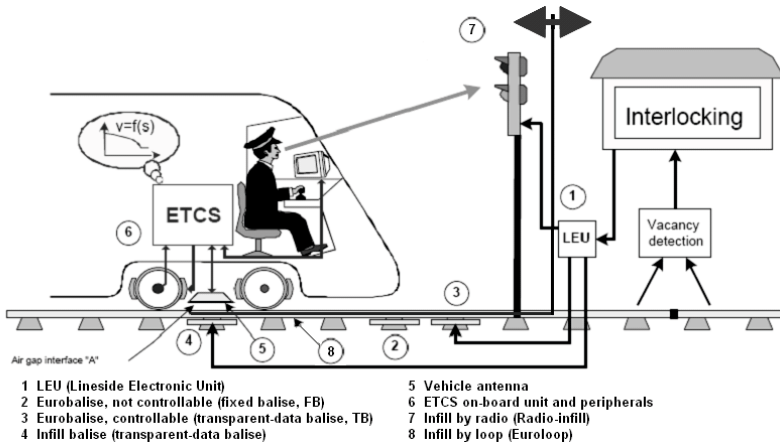
Communication between the tracks and the train are ensured by dedicated balises (known as "Eurobalises") located on the trackside adjacent to the line side signals at required intervals, and connected to the train control centre (Fig. 2). The balises contain pre-programmed track data. The train detection equipment sends the position of the train to the control centre. The control centre, which receives the position of all trains on the line, determines the new movement authority (MA) and sends it to the balise. Train passes over the balise, receiving the new movement authority and track data. The on-board computer then calculates its speed profile from the movement authority and the next braking point. This information is displayed to the driver.

The industrial use case is a specific function of the On-Board SubSystem: the Linking Function. As described above, balises transmit track data (speed limitations, signal-status, etc.) to the on-board sub-system.

The aim of Linking Function is:

- to determine whether a balise group has been missed or not found within the expectation window and take the appropriate action;
- to assign a co-ordinate system to balise groups consisting of a single balise;





**Fig. 2.** On-Board Unit (OBU) ERTMS Level 1

- to correct the confidence interval due to odometer inaccuracy. It is not possible avoiding the odometer error, also for the best technologies (e.g., due to sliding). The balise position, instead, is a reference system for train position that resets the odometric error each time the train passes over a balise.

The linking information transmitted by balise group to the OBU (On-Board Unit) is composed of:

- a) the identity of the linked balise group (the list of expected balises group)
- b) where the location reference of the group has to be found
- c) the accuracy of this location
- d) the direction whereby the linked balise group will be passed over (nominal or reverse)
- e) the reaction required if a data consistency problem occurs with the expected balise group.

## 2 State of Art in Testing Activities

The use case reflects the industrial “state of the practice” of the testing activities in the rail domain and aims to reveal technical gaps that should be closed by one or more technical innovations. In the testing process it is possible to identify the following activities: test definition, test execution, test report analysis and test report document drawing up (a document filled on the basis of test report analysis). A workshop carried out among experienced test engineers in ASTS, suggested that the above mentioned activities could be significantly improved. In particular, the technical gaps identified in each activity and the amount of effort spent to develop each activity exploiting current techniques and methods are illustrated in the Table 1. The amount

of effort put on each testing activity is indicated in percentage points. The sum of percentage points should be 100%, as it is assumed that 100% represents the current overall effort spent in the testing process.

**Table 1.** Effort evaluation in testing activities

<b>Activity</b>	<b>Technical Gaps</b>	<b>% of effort actually spent</b>
Test Definition	Starting from system requirements, tests are manually defined and recorded in test cards. Then, the test cards are used for a manual execution or translated in test scripts, in a proprietary data format, for an automatic execution.	25%
Test Execution	In the “current practice”, where an automatic test execution environment is available, there are interoperability problems due to different proprietary data formats from heterogeneous providers. In particular, test data and test logs are expressed in a proprietary format, usually different for each provider.	15%
Test Report Analysis	Most of the efforts spent in this phase concerns the test report analysis that is manually performed.	50%
Test Report document Drawing Up	Most of the efforts spent in this phase are due to the test report document that is manually drawn up.	10%

Actually, the average efforts, using the current ASTS test equipments for a typical industrial project (more than 2000 test cases generated from almost 1000 requirements), concerning test definition and test script translation phase, is shown in Table 2.

**Table 2.** Average evaluation of the effort for Test Definition activity

<b>Activity</b>	<b>Average rate</b>	<b>Time consumption (1 person)</b>
Test Cards definition from requirements in Natural Language	40 test/month	50 months
Test Cards translation in Test Script	300 test/month	6,7 months

### 3 Technical Innovation Needed in Testing Activities

The proposed actions to be taken, in order to improve the above mentioned phases, are illustrated below. The percentage points below represent the new effort requested for each activity of the testing process when the technical innovations indicated will be implemented. As shown in Table 3, technical innovations could reduce ASTS testing effort of almost the 50%. This represents, indeed, the overall effort spent in the testing process if the methods indicated for each phase are implemented. Note that the technical innovations proposed for the Test Execution phase are necessary to allow different subsystems from heterogeneous providers to work together.

**Table 3.** New effort evaluation in testing activities

Activity	Technical Innovation of CESAR	% of new efforts spent with RTP
Test Definition	<p>It is necessary to implement methods that allow the definition of automatic tests:</p> <ul style="list-style-type: none"> <li>• Automatic test specification from the SRS.</li> <li>• Customizable test generation.</li> </ul> <p>The preliminary phase of requirements specification in formal language allows to analyze these requirements respect to completeness, consistency and ambiguity. In this way the test definition phase starts from well defined requirements.</p>	5%
Test Execution	<p>It is necessary to follow the approach proposed by UNISIG (Subsets 110, 111 and 112), that allows converting:</p> <ul style="list-style-type: none"> <li>• Test data input expressed in a proprietary data format.</li> <li>• Test data output expressed in a standard log format that is common to all providers.</li> </ul>	15%
Test Report Analysis	It is necessary to automatically analyze the results, by comparing the report with the expected output.	25%
Test Report document Drawing Up	It is necessary to automatically generate the test report document.	5%

The reduction of any efforts is obtained by an analysis, made by a domain expert, on a simulation of a testing activity in which the Test Validator is supported in his work by this technical innovation. The results of this process simulation rely on:

- **Test Definition:** the RTP supports the Test Validator in the automatic generation of test scenarios from formal requirements. Moreover, the Test Validator has to define rules to generate a customized set of tests, if needed.
- **Text Execution:** is independent from this approach and no improvement are present.
- **Test Report Analysis:** test oracle has permitted an automatic success report, with a decrease of log that user must analyze. The effort of Test Validator is only to analyze failed tests.
- **Test Report document Drawing Up:** the automatic test report generation is a RTP specific function that allow to have a draft documentation with test specification and results of each test. The user must only complete this test report with his analysis.

### 3.1 Proposed Solution: ASTS Pilot Application

The scheme in Fig. 3 shows the activities that should be performed to develop the ASTS PA. The scheme evidences the activities that could be performed in each CESAR industrial domain (and also in a generic industrial domain), those ones that should be performed only in the rail domain or specific for a particular company.

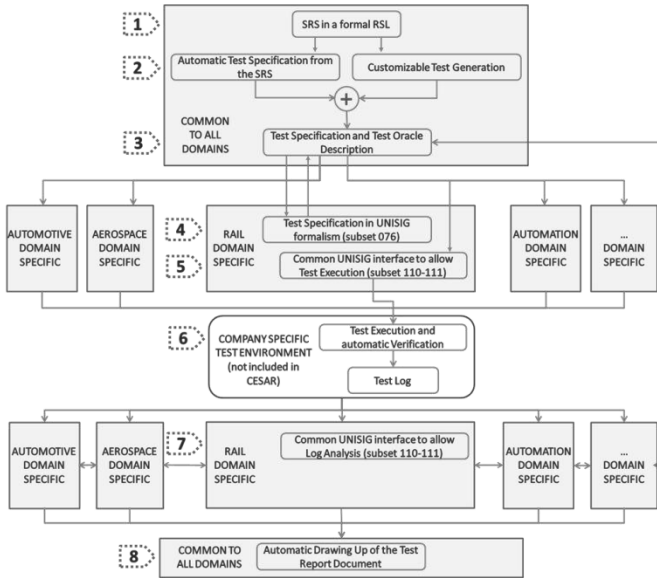


Fig. 3. Development scheme of ASTS Pilot Application

In the scheme the following activities are identified:

1. **System requirements specification (SRS):**

System requirements should be specified in a formal requirements specification language (RSL). The coherence and completeness of the requirements should be automatically checked by means of RTP functionality.

2. **Test definition:**

Two complementary approaches should be taken into account when defining the test:

a) **Automatic test specification from the SRS:**

For each system requirement a test should be automatically specified, in order to determine whether the system satisfies the requirement.

b) **Customizable test generation:**

It should be possible to define rules, which allow the user to generate a customized set of tests (see [9], [10]).

3. **Test specification and test oracle description:**

The scripts of the identified tests should be expressed using a flexible standard formalism, for example xml. The test oracle should be automatically defined: it should define the expected outputs of the test, in order to detect if the actual outputs obtained by the test are correct or not.

4. **Test specification in UNISIG formalism:**

The scripts of the identified tests, expressed in a standard formalism, should be translated to the formalism proposed by UNISIG (Subset 076 [11]) in order to make the test specifications easily understandable by everybody involved in rail applications (suppliers, customers, assessors). It should also be possible to translate the test scripts from the UNISIG formalism to standard one proposed by CESAR.

**5. Common UNISIG interface to allow test execution:**

It is necessary to follow the approach proposed by UNISIG (Subsets 110 [12], 111 [13], 112 [14]), that allows us to convert the test scripts from a standard format (that is common to all providers) to a proprietary format. Indeed, as UNISIG has a strategic importance for the rail market, CESAR should be compliant to all its future disposals in this domain.

**6. Test execution and automatic verification:**

Test should be executed and the results automatically analyzed, comparing the actual outputs with the expected ones, defined in the activity 3. This activity is not common to all domains but depends on the particular application and on the specific test environment, so it is specific for each company. Each industrial partner should perform this activity internally, using its own test environment.

**7. Common UNISIG interface to allow log analysis:**

It is necessary to follow the approach proposed by UNISIG (Subsets 110, 111 and 112) that allows converting the test log from a proprietary format into a standard format, common to all providers. As in the activity 5, the common interface should be developed in the research area of CESAR and be exploited only in rail domain.

**8. Automatic drawing up of the test report document:**

It is necessary to automatically generate the test report document containing the test specification and the result of each test. A specific RTP function should pick up all the elements requested by the user in the customized template from the test log and use them to automatically generate the test report document.

The efforts spent to perform testing activities with the use of the RTP is expected to be 50% lower than the ones spent with the use of the current ASTS test equipments. At present, the technical innovations provided by CESAR project are related to preliminary phases of ASTS PA. Therefore we expect a reduction of both the efforts for system requirements specification activity and efforts test definition activity.

## **4 Instance of RTP with ASTS Use Case**

The current version of CESAR RTP is composed by various tools that allow to the users to manage, analyze, check system requirements, modeling the system, auto code generation and test cases generation. All these tools are COTS ones and they are integrated in the RTP bus in order to provide a complete environment that covers the phases of design system, from the system conception and requirement capturing to the system realization. The ASTS demonstrator is related to the Requirements Analysis (DODT, Domain Ontology Design Tool [15]) and Automatic Test Cases Generation (ATG) by means the connection of two tools to the RTP, as shown in Fig. 4. The first tool is for requirements specification and completeness, consistency and ambiguity check. The second tool is for the automatic test cases generation out of formalized, consistent and not ambiguous requirements. In this phase the project covers the steps 1) and 2) of ASTS Pilot Application. It is planned that the total workflow of previous section will be developed with next versions of CESAR RTP.

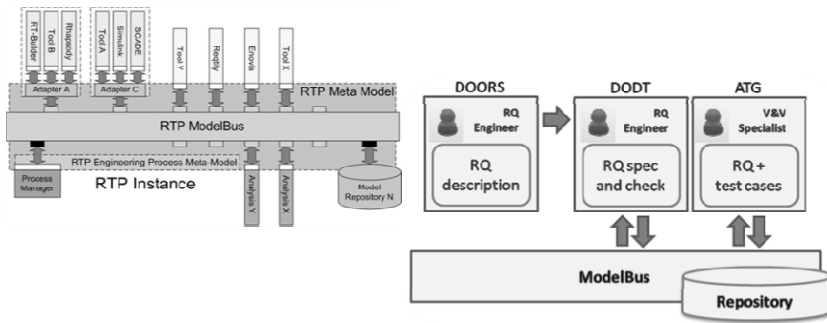


Fig. 4. RTP instance with Ansaldo STS demonstrator

#### 4.1 DODT (Domain Ontology Design Tool)

The DODT is a tool that implements the conversion, in a semi-automatic way, from natural language (NL) requirements into a semi-formal representation called boilerplates. Hull, Jackson and Dick [16] first suggest this approach to requirements elicitation thought of as using semi-formal requirements that are parameterized to suit a particular context. Using fixed syntax and variable parts, called attributes predefined templates could be created. Users can specify requirement attributes as stakeholder or capability objects and events involved in the system, as well as performance characteristics etc. ... Some examples of templates could be “<System> shall / shall be able to <action>” or “If <state>/<event>, the <System> shall / shall be able to <action>”. In this templates <System>, <action>, <state> and <event> are named attributes and **shall / shall be able to / if / the** are named fixed syntax element. The use of predefined structures allows reducing spelling mistakes, ambiguity, etc., thus facilitating understandability, categorization and identification of the requirements. Additionally, boilerplate RSL acts as an input mechanism for capturing the underlying system-related semantics behind requirements i.e. the stakeholders involved, the system capabilities and so forth.

The boilerplate requirements method requires a domain ontology. Stålhane, Omoronyia and Reichenbach [17] extended boilerplates with a domain ontology by linking attribute values to ontology concepts. The domain ontology is expected to contain the following entities:

**Concepts:** Concepts are things (both physical and abstract) which exist in the domain, e.g. Balise Group. In addition to its name a concept also contains a textual definition, e.g. a short paragraph explaining the meaning of the term “Balise Group”.

**Relations:** Relations are direct connections between two concepts with an attached label. Relations represent knowledge in the form “subject verb object”, e.g., “<Balise Group> <send> <message>”.

**Axioms:** Axioms express certain formal relations between concepts, e.g. equality or subclass relations between concepts. Equivalence information can be used in requirements analysis, e.g. to suggest replacing an occurrence with an equivalent to improve consistency in the requirements. Subclass information can be used to suggest more specific concepts instead of too generic ones.

The domain ontology for the On-Board Unit Level 1 was created manually, directly writing in the DODT ontology editor. The Requirement Engineer (RE) selects one or more boilerplate-templates and the tool provides suggestions gained from domain ontology information for the attribute values. Thus the DODT tool is able to reduce the amount of mechanical work required to the RE compared with a manual transformation [18].

By means this tool will be executed the following workflow:

1. Requirements are described in DOORS (Dynamic Object Oriented Requirements Systems) file by RE
2. Requirements are imported in DODT
3. Ontology is created in DODT editor by RE
4. Requirements are specified in boilerplates by RE with support of DODT and Ontology
5. RE checks completeness, consistency and ambiguity of requirements with DODT functionality
6. RE stores requirements in a repository of RTP.

After these steps, the V&V Specialist imports these formalized requirements in the ATG tool that generates a set of test cases requirements-based. In the last part of the present article the ATG functionalities will be described.

#### 4.2 Example of Requirements Formalization

For the evaluation a set of 40 requirements was chosen. The creation of boilerplates and formulation of requirements is an interwoven process. The RE determines if appropriate boilerplates for a requirement exist; if they are missing, he needs to create new boilerplates before instantiating boilerplate requirements. The task to create boilerplates is mostly a one-time job with small updates later on, since boilerplates are to a high degree independent of the domain and can be reused well. Exceptions to this are the usage of domain-specific boilerplate attributes (the attributes used in this evaluation are system, entity, action, capability, operational condition and event; all of them are generic and can be reused) and domain- or project-specific guidelines to formulate requirements. The Table 4 contains the On-Board Unit Level 1 ontology domains.

**Table 4.** Linking Function Ontology contents

Contents		Contents	
<b>12</b>	<b>Object Properties</b>	<b>50</b>	<b>Axiom</b>
<b>62</b>	<b>Class</b>	24	Subclass of Axiom
15	Parameter	17	Equivalent Class
8	State	9	Contain
5	System	<b>6</b>	<b>Failure Mode</b>
3	Verb		
5	Object		
10	Entity		
7	Operational Mode		
6	Action		
3	Capability		

Some examples of boilerplate requirements, translated from original requirements, are presented in the Table 5.

**Table 5.** Examples of system requirements specified using boilerplates formalism

	REQ specified in Natural Language	REQ specified in Formal Language
REQ_23	The ERTMS/ETCS on-board equipment shall ignore (i.e. it will not consider as LRBG) a balise group found with its location reference outside its expectation window.	<b>The</b> <ERTMS/ETCS on-board equipment> <b>shall not</b> <consider> <b>the</b> <balise group> <b>if</b> <balise group> <b>have</b> <location reference> <b>unequal to</b> <inside expectation window>.
REQ_42	If a message has been received containing the information “default balise information”, the driver shall be informed.	<b>The</b> <SSB> <b>shall be able to</b> <inform> <b>the</b> <driver> <b>if</b> <balise group> <b>with</b> <message> <b>equal to</b> <default balise information>.

### 4.3 ATG (Automatic Test Generator)

The ATG module must be able to produce test scripts set starting from formal requirements in a standard format. In this first version, ATG allows a static generation of test scripts, setting conditions expressed in the requirement and verifying the consequences. A more complex version will be available in future, implementing a methodology allowing a dynamic generation of test scripts, based on influences variables and on specific rules defined by the Test Designer [19].

The ATG is composed by the following components:

- Test Data Generator: generates the test case from Requirements Analysis.
- Oracle: calculates the value of the expected output.
- Test Manager: coordinates the different modules of ATG.
- File comparator: compares the outputs of the test execution with the Oracle prediction.
- Report Generator: implements test traceability and result of test campaign (i.e. log error, test report).

Here below is shown an example of the static test generation that summarizes the algorithm implemented by the ATG. This example is based on the requirement REQ\_23 (Table 5) of the linking function.

The input of the ATG is the formal requirement, according to the following structure:

REQ_23-BP	<b>If</b> <balise group> <b>have</b> <location reference> <b>unequal to</b> <inside expectation window>, <b>the</b> <ERTMS/ETCS on-board equipment> <b>shall not</b> <consider> <b>the</b> <balise group>
-----------	---

The ATG tool is able to implement the following algorithm:

- Takes in input the set of formal requirements.
- Recognizes the structure of the requirement in terms of:
  - PREFIX: pre-conditions to be set in the scenario.
  - MAIN: conditions to be verified to be compliant with the requirement.



- Recognizes, starting from pre-configured structures, the PREFIX and MAIN parts:

PREFIX	<entity> <b>have</b> <parameter> <b>unequal to</b> <state>
MAIN	<b>the</b> <system> <b>shall not</b> <verb> <b>the</b> <entity>

- Instantiates a test script template as the following:

```

Test ID
BEGIN
VERIFY "initial conditions"
SET SCENARIO: <PREFIX>
START "operation"
VERIFY: <MAIN>
RESTORE "initial conditions"
END TEST
    
```

In this template, the statement “initial conditions” and “operation” are predefined macro, aiming the first one to identify a well defined state, used as initial conditions for the test and to restore the same initial conditions after the perturbations introduced by the test, and, the second one, to initiate the operation required by the test (e.g., train run).

- Instantiates the correct values of each indeterminate part of the PREFIX and MAIN (i.e. “parameter”, “state”, “system”, “verb”, “entity”).
- Defines the test script using the test script templates and the correct values:

```

Test Req_ID-BP
BEGIN
VERIFY "initial conditions"
SET SCENARIO:
    <balise group> have <location reference> unequal to <inside expectation window>
START "train run"
VERIFY: the <ERTMS/ETCS on-board equipment> shall not <consider> the <balise group>
RESTORE "initial conditions"
END TEST
    
```

It appears clear that this level of test scripts is very general and doesn’t require any information related to the owner system. Anyway, at this level, it is necessary, as shown in Figure 3, to cover last step toward the owner execution environment, in order to resolve the gap between the “standard level” shown above and the “owner level” of the real system. It should be noted that all the information required for this last step are present in the standard test script.

## 5 Results

The partial results obtained from CESAR project cover two of the most critical parts of the ASTS PA. Formalization of the requirements and the static automatic generation of functional test cases are two fundamental milestones, also if applied to other domain or industrial cases. In this section we present a comparative analysis between efforts using the current ASTS test equipments for a typical industrial project

and efforts using the ASTS demonstrator for CESAR project. Also if based on few cases, the results obtained are sufficient for a good comparison, due to the linear growth of the complexity related to the requirements and tests number. From a comparative analysis between Table 1 and Table 3, we can presume, against the whole Pilot Application, a possible time reduction of about 50% for test activities.

More precisely, referring to the real use case shown in this article and a typical project based on almost 1000 requirements and 2000 tests, the comparison between the old approach and the new one, in terms of time consumption, is the following:

### **Old approach**

In the old approach there were two main steps:

- 1) Manual definition of Test Cards starting from the requirements in Natural Language.
- 2) Manual translation of Test Cards in to the Test Scripts.

The Estimated time to perform these activities for one person is around 50 months.

### **New approach**

In the new approach there are three main steps:

- 1) Refinement of requirements defined in Formal Language
- 2) Automatic generation of Test Cards/Scripts from the requirements in Formal Language
- 3) Definition of rules to allow the user to generate a customized set of tests.

The Estimated time to perform these activities for one person is around 5 months. It should be noted that almost this time is related to the step 3.

As a consequence of the above considerations, the time consumption reduction is around 90% and it is related only to the phase 1) and 2) of the PA.

## **5 Conclusions and Way Forward**

In future works, the RTP will support, by means a new version of the ATG, a more exhaustive tests definition, obtained combining all possible system inputs and allowing the user to significantly reduce the number of tests generated, by defining a set of reduction rules, depending on the domain, specific configuration and other factors related to the project and the tester designer experience. The real challenge is to automatically extract the “system inputs” from the requirements. It is expected that this last innovation will contribute not only to speed up the test activities but also, and mainly, to improve the quality in terms of reduction of errors in test definition and completeness of the functional test set, in order to improve the competitiveness without jeopardizing the safety.

## **References**

1. Kotonya, G., Sommerville, I.: Requirements Engineering. John Wiley & Sons, Chichester (1998)
2. IEEE Recommended Practice for Software Requirements Specification. IEEE Std 830-1998 (1998)

3. CENELEC EN 50126: Railway applications - The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS) (2001)
4. CENELEC EN 50128: Railway Applications - Communication, signalling and processing systems - Software for railway control and protection systems (2001)
5. CENELEC EN 50129: Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling (2003)
6. CENELEC EN 50159-1: Railway applications - Communication, signalling and processing systems – Part 1: Safety-related communication in closed transmission systems (2001)
7. CENELEC EN 50159-2: Railway applications - Communication, signalling and processing systems – Part 2: Safety-related communication in open transmission systems (2001)
8. CENELEC Home Page, <https://www.cenelec.org>
9. De Nicola, G., di Tommaso, P., Esposito, R., Flammini, F., Marmo, P., Orazio, A.: A Grey-Box Approach to the Functional Testing of Complex Automatic Train Protection Systems. In: Dal Cin, M., Kaâniche, M., Pataricza, A. (eds.) EDCC 2005. LNCS, vol. 3463, pp. 305–317. Springer, Heidelberg (2005)
10. De Nicola, G., di Tommaso, P., Esposito, R., Flammini, F., Marmo, P., Orazio, A.: ERTMS/ETCS: Working Principles and Validation. In: Proc. International Conference on Ship Propulsion and Railway Traction Systems, SPRTS 2005, Bologna, Italy, pp. 59–68 (2005)
11. UNISIG, ERTMS/ETCS – Class 1, Scope of Test Specifications, Subset-076-7, issue 1.0.2 (2009)
12. UNISIG, ERTMS/ETCS – Class 1, Interoperability Test Guidelines, Subset-110, issue 1.0.0 (2009)
13. UNISIG, ERTMS/ETCS – Class 1, Interoperability Test Environment Definition (General), Subset-111-1, issue 1.0.0 (2009)
14. UNISIG, ERTMS/ETCS – Class 1, Rules for Interoperability Test Scenarios, Subset-112, issue 0.1.4 (2008)
15. Farfeleder, S., Moser, T., Krall, A., Stålhane, T., Zojer, H., Panis, C.: DODT: Increasing Requirements Formalism using Domain Ontologies for Improved Embedded System Development. In: 14th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, Germany (2011)
16. Hull, E., Jackson, K., Dick, J.: Requirements Engineering. Springer, Heidelberg (2005)
17. Stålhane, T., Omoronyia, I., Reichenbach, F.: Ontology-guided requirements and safety analysis. In: Proceedings of 6<sup>th</sup> International Conference on Safety of Industrial Automated Systems, SIAS 2010 (2010)
18. Omoronyia, I., Sindre, G., Stålhane, T., Biffel, S., Moser, T., Sunindyo, W.: A Domain Ontology Building Process for Guiding Requirements Elicitation. In: Wieringa, R., Persson, A. (eds.) REFSQ 2010. LNCS, vol. 6182, pp. 188–202. Springer, Heidelberg (2010)
19. De Nicola, G., di Tommaso, P., Esposito, R., Flammini, F., Marmo, P., Orazio, A.: An experience in validating train control systems by a grey-box testing approach. In: The Second International Conference on Complex, Intelligent and Software Intensive System, Technical University of Catalonia Barcelona, Spain (2008)

# On the Adoption of Model Checking in Safety-Related Software Industry

Alessandro Fantechi<sup>1,2</sup> and Stefania Gnesi<sup>1</sup>

<sup>1</sup> Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, Pisa, Italy  
`stefania.gnesi@isti.cnr.it`

<sup>2</sup> Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Italy  
`fantechi@dsi.unifi.it`

**Abstract.** In the last fifteen years, model checking has been applied successfully in the design and verification of many safety related software systems. However, it is not yet routinely adopted in the industry of safety-critical systems. In this paper we introduce the model checking technique and its relations to safety; then we survey the sensible areas of research related to the current and potential industrial application of this technique, exploring the current trends, that in our opinion will bring to a wider adoption of model checking in the next years.

## 1 Introduction

Due to the possibility offered by model checking to give a definite result on the satisfaction of a property by a system, model checking has been considered as a very interesting technique in the realm of critical systems, where safety could be put at stake by software errors. Indeed, a naive view of model checking makes immediately evident its potential:

- safety properties have in general a simple expression in temporal logic as  $AG(\sim \textit{badstate})$ : “in all states, a system is never in an unsafe state”;
- proving such property of a system by model checking means to exhaustively explore the system state space: a positive answer means that the whole state space has been verified (100% coverage of states and transitions);
- model checking is a pushbutton technique: in principle once the property is expressed, there is no need of effort in formal reasoning about the system, and this makes the technique extremely attractive in the industrial development process
- in case a safety property is not verified model checkers provide a counterexample that explains the reason why the property is not verified, therefore pointing to a safety flaw.

This potential is however impaired by several problems that have so far limited the actual application of the technique to safe software development. The scalability of verification techniques to the huge number of states of complex real systems is still the main issue, notwithstanding the advances in techniques that

address the state space explosion. The definition of (safety) properties by means of temporal logic or other formalism is not always a simple exercise. Enforced guidelines for the development of Safety Critical Systems are not always in line with recent advances of verification technology and tend to favour traditional verification techniques, namely testing.

This paper surveys the sensible areas of research on, and industrial application of, model checking applied to safety critical systems. We will focus first on the characteristics of safety, also in connection with enforced safety guidelines, and we will see how modelling and evaluating system safety can benefit from this technique. Then, we will consider how software safety is strictly related to software correctness: in this direction the application of formal verification techniques, and in particular model checking, within the development of software systems is following two different trends: on one side, a verification activity on models of a system, within a model based development cycle; on the other side, verification conducted directly on the code. These two trends will be discussed, especially from the point of view of actual industrial applications.

## 2 Background: Model Checking and Temporal Logic

Model checking is an automated technique that, given a finite-state model of a system and a property stated in some appropriate logical formalism (such as temporal logic), checks the validity of this property on the model [8].

Several temporal logics have been defined for expressing interesting properties. A temporal logic is an extension of the classical propositional logic in which the interpretation structure is made of a succession of states at different time instants. We consider here the popular CTL (Computation Tree Logic), a *branching time* temporal logic, whose interpretation structure (also called *Kripke structure*) is the *computation tree* that encodes all the computations departing from the initial states. The CTL syntax is defined on top of the propositional connectives, adding temporal connectives: for the purpose of this paper we consider only these basic temporal operators, with their informal semantics:

- $EX\phi$ : there exist a computation in which in the next state  $\phi$  is true
- $AX\phi$ : in all computations, in the next state  $\phi$  is true
- $EF\phi$ : there exists a computation in which in a future state  $\phi$  is true
- $AF\phi$ : in all computations, there exists a future state in which  $\phi$  is true
- $EG\phi$ : there exist a computation in all states of which  $\phi$  is true
- $AG\phi$ : in all states of all computations  $\phi$  is true

Formal verification by means of model checking consists in verifying that a transition system  $M$ , modelling the behaviour of a system, satisfies a temporal logic formula  $\phi$ , expressing a desired property for  $M$ .

A first simple algorithm to implement model checking works by labelling each state of  $M$  with the subformulae of  $\phi$  that hold in that state, starting with the ones having length 0, that is with atomic propositions, then to subformulae of length 1, where a logic operator is used to connect atomic propositions, then to

subformulae of length 2, and so on. This algorithm requires a navigation of the state space, and can be designed to show a linear complexity with respect to the number of states of  $M$ .

### 3 Safety Guidelines

Safety guidelines for the production of software have been issued in several safety critical systems domains; among them we cite IEC 61508 for embedded systems, CENELEC EN 50128 for railway signalling, RTCA DO-178B for avionics software, MoD DEF-STAN 00-55 for defense equipment. Their issue dating back at the nineties (when model checking was hardly leaving the research labs to the software industry), and the fact that only mature technologies are considered in guidelines for safety-critical systems cannot make a surprise the fact that model checking is never mentioned. But formal methods are mentioned and even recommended.

Although with different wordings and shades, guidelines basically agree to define safety as the capability of a system not to cause, or contribute to, catastrophic consequences in case of a failure [3].

Actually, we need to distinguish between two different views of safety: Absolute Safety, when the total absence of safety is sought, or Probabilistic Safety, in which a residual probability of unsafe behaviour is admitted. The former requires that all causes of threats to safety are removed, and is a conceptual goal to be pursued when designing a system. Probabilistic safety acknowledges the existence of possible residual unsafe events, although with less than a required maximum probability of occurrence: this is a more realistic view, that however in general requires more sophisticated tools to measure this residual probability and to guarantee that the expected threshold is respected.

This distinction finds a parallel in the nature of faults. Some faults are random, and a probability of occurrence can be estimated on the basis of previous failure experience. Some are systematic, and have their root in some design error; all design errors should be removed, and it is indeed difficult, or impractical, to estimate a residual probability of occurrence of a systematic error. Very roughly speaking, we can say that hardware exhibits random failures, while software exhibits systematic failures<sup>1</sup>.

Hence, hardware is mostly subject to quantitative, probabilistic analysis of safety; on the opposite side, safe software should be just correct. This view is enforced by the DEF-STAN 00-55 guidelines, that say: Where safety is dependent on the safety related software (SRS) fully meeting its requirements, demonstrating safety is equivalent to demonstrating correctness with respect to the Software Requirement. Model checking can directly contribute to this demonstration.

---

<sup>1</sup> Although random software faults are commonly reported, they can be in most cases traced to the random occurrence of events or situations that activate a design fault. Whether considering these kind of bugs deterministic or random depends ultimately on the kind of analysis that is considered most effective.

### 3.1 Safety Integrity Level

Expectations on the safety degree of a system are summarized in the *Safety Integrity Level* (SIL), a number ranging typically from 0 to 4, where 4 indicates the higher criticality, 0 gives no safety concern (in DO178B the term used is Development Assurance Level, ranging from A – possible catastrophic effect of a failure of the system – to E – no safety effect of a failure of the system). The SIL is actually a property of the system, related to the damage a failure of the system can produce, and is usually apportioned to subsystems and functions at system level in the preliminary safety assessment; also software functions are associated a level (Software SIL, Software DAL) in the system safety assessment.

Recommendations given by guidelines are usually graduated along the SIL: in particular, SIL apportionment allows software developers to concentrate the verification effort on those functions that are assigned a higher SIL. Accordingly, model checking could be used to address correctness of higher SIL components, hence addressing the complexity in a divide and conquer fashion. However, SIL apportionment to software components is made difficult since it requires independence of components (the failure of one should not affect the correct functioning of the other ones), which is hard to prove, and a typical situation is that all software modules of an equipment inherit the same (high) SIL.

### 3.2 Revision of Current Guidelines

Safety guidelines are undergoing periodic reviews; in particular, the revisions of EN50128 and DO178 deserve a mention for the expectations they have generated in the respective domains. Indeed, the revised EN50128, due to appear in these days, marks the first appearance of model checking as one of the formal verification techniques for safety critical software. On the other side, while DO-178B only mentioned Formal Methods among the Additional Considerations, the revision DO-178C, expected to be released in the first half of 2011, will, for the first time, officially recognize the validity of using Formal Methods within the avionics software development process: formal methods will be possibly used to augment or replace verification steps that DO-178B assigns to testing, and will be allowed to verify requirements correctness, consistency, and augment reviews. Also, DO-178C will allow formal methods to verify or replace test cases used to check low level requirements and replace some forms of testing via formal methods based reviews. There is much space for Model Checking there, although we are not currently aware of the final text of the norm.

### 3.3 Tool Qualification

When using a model checker for the certification of a safety-related software, one of the issues that is often raised is: Is the model checker itself bug-free? That is, can I trust the model checker tool when it says that a system is safe? DO178B addresses this issue by *Tool Qualification*, classifying software tools as:

- Software development tools: Tools whose output is part of the developed software and thus can introduce errors (this type includes compilers)
- Software verification tools: Tools that cannot introduce errors, but may fail to detect them: this type may include model checkers.

According to DO178B the qualification criteria for software verification tools should be achieved by demonstration “that the tool complies with its Tool Operational Requirements under normal operational conditions”; this demonstration requires comprehensive testing and the establishment of a controlled development process. To our knowledge, no model checker up to now has been qualified.

An alternative to qualification can be found in the *proven in use* concept from EN50128: a tool that has a long record of usage within similar projects with no known failure. Since we are at the beginning of industrial application of model checking to software code correctness, this is a hardly justifiable alternative.

Another alternative is to adopt diverse redundancy: duplicating the validation effort by repeating the verification session over two different independent model checkers, and then compare the result for equality.

## 4 Safety Properties

If we want to express safety properties by means of temporal logic formulae, it is natural to resort to the  $AG$  operator: Subclasses of  $AG$  formulae are often used; for example, system safety admits that a system may fail, but with a non-critical failure. Typical is the adoption of safety nets mechanism that avoid critical failures, and the correct working of a safety net could be expressed by the formula  $AG(fault \implies AX\ failsafe)$ : whenever a fault occurs, the next state of the system is a fail safe state ( $AGAX$  form). If model checking this kind of formula returns a counterexample, it represents a computation path leading to the unsafe state.

As a further example, taken from the railway signaling sector, is the no-derailing property: while a train is crossing a point, the point shall not change its position [15]. This typical system level requirement can be represented in the  $AGAX$  form:

$$AG((occupied(tc_i) \wedge setting(p_i) = value) \implies AX(setting(p_i) = value))$$

whenever the track circuit  $tc_i$  associated to a point  $p_i$  is occupied, and the point has the proper setting value, this setting shall remain the same on the next state.

Safety properties are usually confronted with *liveness* properties, which ask to a system to be alive, that is, to have at least the possibility to make some action in the future, and can be exemplified by the  $EF$  operator. We will not discuss further this class of properties, that can be interesting for safety as well in some cases.

### 4.1 Bounded Model Checking of Safety Properties

We have already noted that model checking is an exhaustive technique: the simplest model checking algorithms therefore needs to explore the entire state



space, incurring in the so called *exponential state space explosion*, since the state space often has a size exponential in the number of independent variables of the system. Many techniques have been developed to attack this problem: among them, two approaches are the most prominent and most widely adopted. The first one is based on a symbolic encoding of the state space by means of boolean functions, compactly represented by Binary Decision Diagrams (BDD) [7]. The second approach considers only a part of the state space that is sufficient to verify the formula, and within this approach we can distinguish local model checking and bounded model checking.

If we concentrate on safety properties, that we have seen are often in an *AG* form, we note that they however require the exploration of the complete state space. Hence they can be proved by BDD-based model checking, but they cannot by the approaches that partially explore the state space. Indeed, such approaches may succeed in falsifying them: if a violation to an *AG* property is found in the partially explored state space, then a negative result can be returned and a counterexample produced. That is, verification of *AG* properties is hard, but falsification is not; or, in other words, model checking used for verification is more expensive than model checking used for “bug hunting”.

But let us look more in detail to Bounded Model Checking. This technique aims to prove properties by exploring only a finite depth of the computation tree of the model. One very efficient way to do this is that all computation paths from the initial state are analysed to a depth  $k$ , by encoding them in a boolean formula. Suppose we want to check a property of the form *AGp* on a path of length  $k$ . We can write a boolean formula expressing that at least one state of the path does not satisfy  $p$ :

$$Init(x_0) \wedge \bigwedge_{i=0}^{k-1} T(x_i, x_{i+1}) \wedge \bigvee_{i=0}^k \sim p(x_i) \quad (1)$$

where  $x_i$  are state bit vectors (that is, boolean encoding of state enumeration), *Init* is a predicate that holds for the initial state,  $p$  is the predicate saying that  $p$  holds in that state,  $T$  is the transition relation (a boolean function that is true if a transition exists among the two states). If a *satisfying assignments*, that is, a set of boolean values for all the variables that makes the formula (1) true, is found, then the path does not satisfy *AGp* and is actually a counterexample. Finding a satisfying assignment is a NP-complete problem, but efficient SAT solvers exist that can deal with a huge number of variables in a reasonable time in most cases: efficient bounded model checkers like the one included in NuSMV embed such SAT solvers [5].

On the other side, if no satisfying assignment is found, for all the  $k$ -long paths starting from the initial states, we can state that  $p$  is true in all states up to a depth  $k$ : but we still don't know about longer paths, hence we cannot state that *AGp* is satisfied. In order to verify the property we would need to look for longer counterexamples by incrementing the bound  $k$ , until the *diameter* (that is, the maximum length of a cycle) is reached. However, the diameter might be

very large, and it is not easy to compute it in advance. This task may be eased in the case of those embedded system that exhibit a cyclic control structure.

## 4.2 Adoption of Model Checking

These considerations reinforce the observation that *falsification* is easier than verification of safety properties, that is, model checking is much more capable at finding bugs than at certifying the absence of bugs.

This has practical consequences for the adoption of model checking as evidence in front of an assessor. Another issue to be considered at this regard is that one must show to the assessor that the proved safety properties are complete, that is, that no safety violation can slip in because of a “forgotten” property. This is not an easy task, and should be responsibility of the safety assessment phase, which should produce in advance a “complete” list of properties to be checked on the system.

Considering all the observations we have done about safety, the ultimate problems addressable by model checking can be classified into two main, but overlapping, categories: verification and evaluation of safety of a system, and verification of software code correctness.

In the first category fall all the cases in which safety properties are verified over a (quite abstract) model of the system, a model that is typically available when working at the system design level. Model checking can help to consolidate the correctness of the model. A particular case is when a modelling of the timing behaviour at run-time is given (e.g. by means of timed automata) and timing properties are proven over such model, by proper model checking tools (such as UPPAAL or KRONOS). Another case is the application of probabilistic model checking to evaluate quantitative safety properties: in a proper temporal logic it is possible to express for example the property “after a fault, there is a 99% probability that the system does not run in an unsafe state”, and a probabilistic model checker (such as PRISM) can be used to verify it.

The category of verification of software correctness hosts two main approaches, namely Model Based Development and Code Model Checking (also known as Software Model Checking).

We will not deal any further with real-time properties, nor with probabilistic model checking, which would require a more extended discussion. In the following sections, we discuss instead with some detail those approaches falling in the second category.

## 5 Model Checking within Model Based Design

In a large part of the safety-critical systems industry, the Model Based Design approach has emerged as the main paradigm for the development of software. In this paradigm, early models are refined to obtain detailed models from which code can be derived: one option, more and more popular, is that code is automatically generated, so that the code preserves the behaviour of the detailed

model (modulo correctness of the translator, which in a safety critical regulated domain is not a trivial issue). Hence, verification is conducted on the detailed models, either by extensive simulation (which ultimately corresponds to testing the code) with realistic simulation scenarios, which in some cases are pushed to interface the model with the hardware (the so called hardware-in-the-loop), or by formal verification. In the following we briefly discuss three examples of industrial application of model checking within Model Based Design, taken from the literature. Indeed, many case studies and pilot projects have been presented in the literature, but only a few reports, such as those cited in the following, give interesting data on the overall adoption of model checking in the development process, also due to confidentiality.

One example from the railway signalling domain is the model based development cycle defined at General Electric Transportation Systems (GETS) within a collaboration with the University of Florence [4][14]. The production process for Automatic Train Protection (ATP) Systems is based on modeling by means of Simulink/Stateflow descriptions. Extensive simulation of Stateflow diagrams, automatic code generation from the diagrams, and back-to-back model/code testing are employed in the process. When the process was already established, GETS decided to perform a systematic experimentation with formal verification by means of Simulink Design Verifier, a test generation and property proving engine based on Prover Technology [1], that uses a proprietary algorithm based on bounded model checking with SAT-solvers.

Verification through Design Verifier is performed by translating the property that one wishes to verify into a formula expressed in the Simulink language. In the GETS process, the properties are the unit requirements obtained through the system functional requirements decomposition. The formula expressing the property has the form of a graphical circuit where the variables observed by the property are connected by Simulink blocks implementing logical, arithmetic and time delay operators. The engine verifies that the formula is globally true for every execution path of the Simulink/Stateflow model. The property is interpreted as if the *AG* operator would be prefixed to it. If the property is violated, a counterexample showing a failing execution is given in the form of a test case for the model.

The experience by GETS has produced a domain-dependent classification of unit requirements: two classes, amounting to more than two thirds of the requirements, are shown to be verifiable with this approach, with half the effort of that required to verify them by testing. For the remaining classes, either translating the requirement into Simulink formulae is too expensive, or verification fails to terminate. A traditional, and hence not exhaustive, verification by testing is more suitable for those classes.

The experience at Airbus [6] has many points in common with the one of GETS; in this case however the choice of Model Based development environment has favoured SCADE by Esterel Technologies. Although the adopted V-shaped software development cycle is highly depending on testing, several studies have been conducted on the application of model checking techniques to validate

SCADE models. As in the case of Mathworks Stateflow Design Verifier, also Esterel Technologies SCADE Design Verifier is built on top of the proprietary very efficient SAT solver by Prover Technology. Also in this case, an *observer* based approach to property expression is adopted, this time using the SCADE synchronous logic blocks and gates.

Formal verification proved effective in the early discover of violations in models of flight control functions: such violations could not be detected by early tests performed by simulation runs on the SCADE models, since these early tests do not consider highly dynamic aspects, e.g, unusual aircraft trajectories, which are usually taken into account only by later test on a flight simulator; however, the earlier violations are detected, the cheaper is their correction. A challenge to formal verification was posed by some models which employed temporal counters, which produce enormous state spaces: one of these function was solved in 48 computation hours. Counterexamples (that is, detected erroneous executions) have been reported to be 50 to 160 cycles long, witnessing the intricacy of the conditions that bring to the error.

Another prominent experience in the avionics domain is the one by Rockwell Collins [17], where both Simulink/Stateflow and SCADE were used as Model Based development environments, and several translators have been developed to apply different model checkers, among which SMV, NuSMV and, again, Prover, to the models coming from different sources. The first phases in this experience have confirmed that BDD-based SMV and NuSMV are capable of dealing with very large state spaces, and that model checking is much more effective than testing in finding errors: this has been actually proved by a parallel verification conducted by two independent verification teams: while the model checking team found 12 errors, the testing team was not able to find any error, although spending half of the time spent by the model checking tool.

A more advanced phase has addressed systems that make extensive use of floating point numbers. Floating point numbers pose a big challenge because of the complexity of their arithmetics and the implied huge size of the state space. Abstraction has been attempted by converting floating point to fixed point, through proper scaling, and converting fixed point into integers, by shift to preserve the order of magnitude. Resorting to satisfiability modulo theories (SMT) solvers provided by Prover has been necessary. Due to the resulting loss of precision, model checking once again has demonstrated itself still very valuable for debugging, but not at demonstrating correctness.

All the reported experiences witness a convergence towards a few commercial MBD development environments, and related formal verification engines. Other common observations were:

- the expression of properties by means of the same (graphical) formalisms in which the models are designed is a feature particularly appreciated by designers;
- verification is actually far from being a single push-button experiment. It is rather an iterative process;
- the tools give however not sufficient support for this iterative process;

- interpretation of counterexamples requires most effort;
- the inherent inability of the tools to supply more than a single counterexample does not help.

Hiding the technicalities of temporal logics to the user is sometimes pushed a step further. VisualSTATE, by IAR Systems, is another Model Based design tool that allows the user to describe the behaviour of a system by means of statecharts. Its verification engine allows only for “built-in” properties, such as absence of deadlock, or absence of nondeterminism, to be proved.

## 6 Software Model Checking

The first decade of model checking has seen its major applications in the hardware verification; meanwhile, applications to software have been made at system level, or at early software design. Later, applications within the model-based development have instead considered models at a lower level of design, closer to implementation. But such an approach requires an established process, while in many cases software is written directly from requirements. Or, software is received from third parties, who do not disclose their development process. In such cases direct verification of code correctness is therefore a must for safety-related systems: testing is the usual choice, but we know that testing cannot guarantee exhaustiveness. Direct application of model checking to code is however still a challenge, because the correspondence between a piece of code and a finite state model on which temporal logic formulae can be proved is not immediate: in many cases software has, at least theoretically, an infinite number of states, or at best, the state space is just huge.

Pioneering work on direct application of model checking to code (also known as *Software Model Checking*) has been made at NASA since the late nineties by adopting in the time two strategies: first, by translating code into the input language of an existing model checker – in particular, translating into PROMELA, the input language for SPIN. Second, by developing ad hoc model checkers that directly deal with programs as input. In both cases, there is the need to extract a finite state abstract model from the code, with the aim of cutting the state space size to a manageable size.

### 6.1 Abstraction

Abstraction is therefore the key to Software Model Checking. Abstraction eliminates details irrelevant to the property, but is likely to introduce loss of precision, by producing false positives or false negatives.

Actually, we can distinguish three kinds of abstraction:

- *Over-approximation*, i.e. more behaviors are added to the abstracted system than are present in the original;
- *Under-approximations*, i.e. less behaviors are present in the abstracted system than are present in the original;

- *Precise abstractions*, i.e. the same behaviors are present in the abstracted and original program.

These kinds of abstraction are useful to classify effective abstraction techniques such as:

- Limit input values to 0..5 rather than all integer values, limit size of buffers to 3 instead of unbounded, etc. (under-approximation);
- Property-directed program slicing: only the parts of the program that influence variables referred to within properties are maintained (precise approximation);
- Abstract Interpretation: Maps sets of states in the concrete program to one state in the abstract program, Reduces the number of states, but increases the number of possible transitions from each state, and hence the number of behaviors [12] (over-approximation);
- Predicate abstraction: abstracts data by considering only certain predicates on the data. Each predicate is represented by a Boolean variable in the abstract program [10] (over-approximation).

It can be shown that over-approximation preserves formulae from the universal fragment of CTL (which do not use the existential quantifier  $E$ ), that is, if a formula of this fragment is true on the over-approximated abstract systems, then it is also true on the concrete system. Dually, under-approximation preserves formulae from the existential fragment of CTL. Since we are mainly interested in  $AG$  formulae, we consider from now on only over-approximation.

If in checking an  $AG$  formula a violation is found in the abstract program, we cannot say whether it is also an error in the original program. This is not always the case: a counterexample referring to a violation that is not related to an error in the original program is called *spurious*.

## 6.2 Counterexample-Guided Abstraction Refinement

The presence of a spurious counterexample is an indication that the abstraction is too coarse. A technique that allows for an automated abstraction refinement is the Counterexample Guided Abstraction Refinement paradigm (CEGAR) [9], that can be summarized by the following steps:

1. An over-approximated finite state model is extracted from the code;
2. model checking is run to check if the abstract model satisfies  $\phi$ ;
  - If yes, then  $\phi$  is reported to be satisfied by the examined code and CEGAR terminates;
  - Otherwise, the model checker reports a counterexample, and the process go on to the next step;
3. It is determined if the counterexample is spurious. This is done by simulating the (concrete) program using the abstract counterexample as a guide, to find out if the counterexample represents an actual program behavior;
  - If the counterexample is not spurious, then  $\phi$  is reported to be not satisfied by the examined code and CEGAR terminates;
  - Otherwise, go on with the next step;

4. The abstraction is refined in order to eliminate the detected spurious counterexample. The CEGAR loop goes back to step 2.

This process aims to produce the coarsest abstraction (therefore, with the minimum state space) that is able to effectively verify or falsify the formula.

### 6.3 Software Model Checkers

The first attempts to apply model checking to verification of code have been conducted mainly, as we have already said, by translation to established model checker such as SPIN. Later on, a number of model checkers taking code as input have been developed. Some common characteristics of such tools are:

- the use in most case of an explicit state space representation, which is developed on-the-fly along the principles of *local model checking*, where only the portion of the state space necessary to prove the property is computed and kept in memory;
- the adoption of a CEGAR abstraction engine;
- they do not require the verifier to express formulae in a temporal logic: normally assertions in a syntax close to the one of the programming language have to be defined, and the model checker verifies that such assertions are verified in every interested computation: again, the assertions are interpreted as if the AG operator would be prefixed to them.
- hiding the formality to the user is even pushed to provide “built-in” default properties to be proven, such as absence of division by zero, safe usage of pointers, safe array bounds, etc. On this ground, such tools are in competition with tools based on Abstract Interpretation such as Polyspace [13].

Most notorious among software model checkers is JavaPathFinder [18], a project developed by Nasa to verify Java programs. JavaPathFinder includes the Bandera translator from Java Bytecode to a number of popular model checkers. Indeed, the tool provides a Java Virtual Machine that does not execute the bytecode, but checks it over the given assertions. JavaPathFinder has been used to verify software deployed on space probes; in particular the detection and correction during the flight of a bug inside software on board of the Deep Space probe DS-1 has been reported [16].

An interesting application of model checking is for counterexample guided automatic test generation (ATG). The basic technique is simple: if testing has to cover some goal, a temporal logic formula expressing the property “the uncovered goal can never be reached” is checked over the model. If the property is false, a counterexample is produced, showing an execution that exercises the uncovered structure. A test case exercising the coverage goal can then be extracted from the counterexample. A recent implementation of this technique in the railway signalling domain uses a software model checker, CBMC, for the generation of test cases for the code under test [2]: the goal in this case is to achieve the 100% decision coverage required for the code of the ERTMS train control system. Ansaldo STS reports that this approach allows for the generation of the double of tests w.r.t. manual generation, with an effort a magnitude order less.

**Table 1.** Main free model checkers

Model Checker	Model specification	Property specification
SMV (CMU)	Network of automata communicating by shared variables	CTL
<i><a href="http://www.cs.cmu.edu/modelcheck/smv.html">http://www.cs.cmu.edu/modelcheck/smv.html</a></i>		
SMV (Cadence)	Network of automata communicating by shared variables	CTL
<i><a href="http://www-cad.eecs.berkeley.edu/kenmcmil/smv">http://www-cad.eecs.berkeley.edu/kenmcmil/smv</a></i>		
NuSMV	Network of automata communicating by shared variables	CTL (LTL)
<i><a href="http://nusmv.fbk.eu">http://nusmv.fbk.eu</a></i>		
SPIN	PROMELA	LTL
<i><a href="http://spinroot.com/spin/whatispin.html">http://spinroot.com/spin/whatispin.html</a></i>		
<b>Software Model checkers</b>		
BLAST	C programs	C annotations
<i><a href="http://mtc.epfl.ch/software-tools/blast/index-epfl.php">http://mtc.epfl.ch/software-tools/blast/index-epfl.php</a></i>		
CPAChecker	C programs	C annotations
<i><a href="http://cpachecker.sosy-lab.org/">http://cpachecker.sosy-lab.org/</a></i>		
CBMC	C,C++ programs	C assertions
<i><a href="http://www.cprover.org/cbmc/">http://www.cprover.org/cbmc/</a></i>		
JavaPathFinder	Java Bytecode	JPF annotations, verification API
<i><a href="http://javapathfinder.sourceforge.net/">http://javapathfinder.sourceforge.net/</a></i>		
SLAM	C programs	C assertions
<i><a href="http://research.microsoft.com/en-us/projects/slam/">http://research.microsoft.com/en-us/projects/slam/</a></i>		

Table 1 lists the most known free (software) model checkers; in particular we notice SLAM, developed by Microsoft, used in-house to check that Windows device drivers obey API conventions.

## 7 Conclusions

Model Checking advantages are more and more recognized in several safety-critical systems domains. Model checking is slowly slipping in safety critical systems development guidelines, and anyway, just mentioning formal methods has apparently favoured more penetration of this technique in regulated domains, w.r.t. unregulated one, such as automotive.

However, model checking shows itself very valuable for debugging, but not at demonstrating safety (especially in front of an assessor). Still problems of complexity, scalability, tool support make this technique at best appear as a side validation possibility to achieve more confidence on what is developed.

Although much advancement is still needed, the next decade will most probably see a fast growth in Model Checking application to industrial safety critical



systems, including adoption of timed models and of probabilistic model checking, which we have left out of our discussion.

## References

1. Abdulla, P.A., Deneux, J., Stålmarmark, G., Ågren, H., Åkerlund, O.: Designing safe, reliable systems using scade. In: Margaria, T., Steffen, B. (eds.) *ISO/LA 2004*. LNCS, vol. 4313, pp. 115–129. Springer, Heidelberg (2006)
2. Angeletti, D., Giunchiglia, E., Narizzano, M., Puddu, A., Sabina, S.: Using Bounded Model Checking for Coverage Analysis of Safety-Critical Software in an Industrial Setting. *J. Autom. Reason.* 45(4) (2010)
3. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.E.: Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. Dependable Sec. Comput.* 1(1), 11–33 (2004)
4. Bacherini, S., Fantechi, A., Tempestini, M., Zingoni, N.: A story about formal methods adoption by a railway signaling manufacturer. In: Misra, J., Nipkow, T., Karakostas, G. (eds.) *FM 2006*. LNCS, vol. 4085, pp. 179–189. Springer, Heidelberg (2006)
5. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without bDDs. In: Cleaveland, W.R. (ed.) *TACAS 1999*. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
6. Bochet, T., Virelizier, P., Waeselynck, H., Wiels, V.: Model checking flight control systems: The Airbus experience. In: *ICSE Companion 2009*, pp. 18–27 (2009)
7. Bryant, R.: Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* C-35(8), 677–691 (1986)
8. Clarke, E., Grumberg, O., Peled, D.: *Model Checking*. MIT Press, Cambridge (1999)
9. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) *CAV 2000*. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
10. Clarke, E., Kroening, D., Sharygina, N., Yorav, K.: Predicate Abstraction of ANSI-C Programs Using SAT. *Form. Methods Syst. Des.* 25, 105–127 (2004)
11. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. *Form. Methods Syst. Des.* 19, 7–34 (2001)
12. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: *Proceedings of the 4th POPL*, pp. 238–252. ACM, Los Angeles (1977)
13. Deutsch, A.: *Static verification of dynamic properties - Polyspace white paper* (2004)
14. Ferrari, A., Grasso, D., Magnani, G., Fantechi, A., Tempestini, M.: The metro ATP case study. In: Kowalewski, S., Roveri, M. (eds.) *FMICS 2010*. LNCS, vol. 6371, pp. 1–16. Springer, Heidelberg (2010)
15. Ferrari, A., Grasso, D., Magnani, G., Fantechi, A.: Model Checking Interlocking Control Tables. In: *FORMS/FORMAT 2010*, Braunschweig, Germany (December 2-3, 2010)
16. Havelund, K., Lowry, M., Park, S.J., Pecheur, C., Penix, J., Visser, W., White, J.L.: Formal Analysis of the Remote Agent Before and After Flight. In: *5th NASA Langley Formal Methods Workshop*, Williamsburg, Virginia (June 13-15, 2000)
17. Miller, S.P., Whalen, M.W., Cofer, D.D.: Software model checking takes off. *Commun. ACM* 53(2), 58–64 (2010)
18. Visser, W., Havelund, K., Brat, G., Park, S.J., Lerda, F.: Model Checking Programs. *Automated Software Engineering* 10(2), 203–232 (2003)

# Equivalence Checking between Function Block Diagrams and C Programs Using HW-CBMC

Dong-Ah Lee<sup>1</sup>, Junbeom Yoo<sup>1</sup>, and Jang-Soo Lee<sup>2</sup>

<sup>1</sup> Division of Computer Science and Engineering, Konkuk University  
1 Hwayang-dong, Gwangjin-gu, Seoul, 143-701, Republic of Korea  
{ldalove, jbyoo}@konkuk.ac.kr  
<http://dslab.konkuk.ac.kr>

<sup>2</sup> Korea Atomic Energy Research Institute, 150 Deokjin, Yuseong  
Daejeon, 305-335, Republic of Korea  
jslee@kaeri.re.kr  
<http://www.kaeri.re.kr>

**Abstract.** Controllers in safety critical systems such as nuclear power plants often use Function Block Diagrams (FBDs) to design embedded software. The design program are translated into programming languages such as C to compile it into machine code for particular target hardware. It is required to verify equivalence between the design and the implementation, because the implemented program should have same behavior with the design. This paper introduces a technique about verifying equivalence between a design written in FBDs and its implementation written in C language using HW-CBMC. To demonstrate the effectiveness of our proposal, as a case study, we used one of 18 shutdown logics in a prototype of Advanced Power Reactor's (APR-1400) Reactor Protection System (RPS) in Korea. Our approach is effective to check equivalence between FBDs and ANSI-C programs if the automatically generated Verilog program is translated into appropriate one of the HW-CBMC.

**Keywords:** Equivalence Checking, Behavioral Consistency, FBDs, Verilog, ANSI-C, HW-CBMC.

## 1 Introduction

Controllers in safety critical systems such as nuclear power plants use Function Block Diagrams (FBDs) to design embedded software. The design is implemented using programming languages such as C to compile it into a particular target hardware. The implementation must have the same behavior with the design's one and it should be verified explicitly. For example, Korea Nuclear Instrumentation & Control System R&D Center (KNICS) [1] has developed a loader software, POSAFE-Q Software Engineering Tool (pSET) [2], to program POSAFE-Q Programmable Logic Controller (PLC). It provides Integrated Development Environment (IDE) including editor, compiler, downloader, simulator and monitor/debugger. It uses FBD, Ladder Diagram (LD) and Sequential Function Chart (SFC) to design a program of PLC software. The pSET translates

FBDs program into ANSI-C program to compile it into machine code for PLC. The ANSI-C program must have same behavior with the FBDs program.

Language difference between the design and the implementation (i.e., FBDs and ANSI-C) makes preserving behavioral consistency difficult. There are several studies and products to guarantee the behavioral consistency. Mathematical proof or verification of compiler, including code generator and translator, can help guarantee the behavioral consistency between two programs written in different languages. Those techniques have weaknesses, which are high expenditure [3] and repetitive fulfillment whenever the translator is modified. On the other hand, RETRANS [4] which is a verification tool of automatically generated source code, doesn't consider transformation rules of a specific translator. It analyzes only the generated source code to reconstruct its inherent functionality and compares it with its underlying specification to demonstrate functional equivalence between both. This approach requires additional analysis to reconstruct useful information from the generated source code. Our approach is verification of equivalence between design program and its implementation program without additional analysis or transformation of the implementation program.

This paper introduces a technique verifying equivalence between design program written in FBDs and its implementation program written in ANSI-C using HW-CBMC [5]. The HW-CBMC is formal verification tool, verifying equivalence between hardware and software description. It requires two inputs for checking equivalence, Verilog for hardware and ANSI-C for software. We used it for verifying equivalence between design program and its implementation program. We first translated the design program written in FBDs into semantically equivalent Verilog program [6][7] to use as an input program of HW-CBMC, and verified equivalence between the Verilog program and the ANSI-C implementation of the FBD designs. We performed a case study to demonstrate its feasibility with one of 18 shutdown logics in a prototype of Advanced Power Reactor's (APR-1400) Reactor Protection System (RPS) in Korea. We translated FBDs program of the shutdown logic program into Verilog program. The ANSI-C implementation of the shutdown logic was implemented manually, because it hadn't prepared yet. We found specific features of the HW-CBMC that input program should follow specific rules related with naming variables or function calls of Verilog program. We, therefore, should modify the Verilog program to be appropriate of the rules, and verified equivalence between the design and implementation of the shutdown logic. We founded that our approach is feasible to verify equivalence between design program written in FBD and its implementation program written in ANSI-C using ANSI-C.

The remainder of the paper is organized as follows: Section 2 explains equivalence checking, and translation rules from FBDs into Verilog briefly. Section 3 describes our technique to verify equivalence between design program and its implementation program. Section 4 explains a case study which verify equivalence between one of 18 shutdown logics in RPS of APR-1400 and its implementation. Section 4 also explains modification for the Verilog program and results of the checking. We conclude the paper at Section 5.

## 2 Related Work

### 2.1 Equivalence Checking

Equivalence checking is a technique to check behavioral consistency between two programs. The VIS (Verification Interacting with Synthesis) [9] is a widely used tool for formal verification, synthesis, and simulation of finite state systems. It uses Verilog as a front-end, and also provides combinational and sequential equivalence checking of two Verilog programs. The combinational equivalence of the VIS provides a sanity check when re-synthesizing portions of a network, and its sequential verification is done by building the product finite state machine. The checking whether a state where the values of two corresponding outputs differ, can be reached from the set of initial states of the product machine. On the other hand, there is a study for equivalence checking between two different descriptions. [8] presents a formal definition of equivalence between Transaction Level Modeling (TLM) and Register Transfer Level (RTL) is presented. The TLM is the reference modeling style for hardware/software design and verification of digital systems, and the RTL is a level of abstraction used in describing the operation of a synchronous digital circuit. The definition is based on events, and it shows how such a definition can be used for proving the equivalence between both.

### 2.2 Function Block Diagram

An FBD (Function Block Diagram) consists of an arbitrary number of function blocks, 'wired' together in a manner similar to a circuit diagram. The international standard IEC 61131-3 [11] defined 10 categories and all function blocks as depicted in Fig.1. For example, the function block ADD performs arithmetic

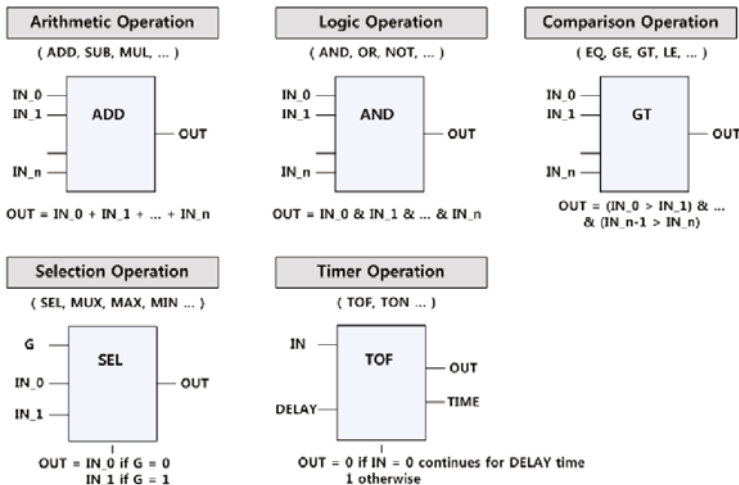


Fig. 1. A part of function blocks and categories defined in IEC 61131-3

addition of  $n+1$  IN values and stores the result in OUT variable. Others are interpreted in a similar way. The FBD described in Fig. 4 which is our case study consists of a set of interconnection.

### 2.3 Transformation from FBDs into Verilog

Our approach of equivalence checking using HW-CBMC first translates FBDs into Verilog. Translation rules were proposed in [6]. The rule consists of three parts, corresponding to unit, component and system FBDs respectively. It defines the IEC 61131-3 [11] FBDs as state transition systems.

First part of the rules describes how a unit of FBD is translated into a function in Verilog language. It first determines Verilog function type, and each input and its type are declared. Behavioral description of the function is then followed, such as arithmetic, logic or selection operations. Second part explains translation rules for component FBDs. The component FBD is a logical block of independent function blocks which a number of function blocks are interconnected with to generate meaningful outputs. The rules of the second part declare a name of component FBD, ports and register type variables. If an output variable is also used as input, it is declared as reg type as its value is to be used in the next cycle. Every Verilog function is called if there is a function according to its execution order to generate outputs of the component FBD. Every function block is separately translated as a Verilog function and included in the definition of module for the component FBD. The last part describes how a system FBD is translated into a Verilog program. A system FBD contains a number of component FBDs and their sequential interconnections. While translation rules for system FBDs look similar to the rules of component FBDs, it calls Verilog Modules instead of Verilog function. Verilog modules are instantiated and called according to their execution order with outputs communicated.

### 2.4 HW-CBMC

The HW-CBMC [5] is a testing and debugging tool for verifying behavioral consistency between two implementations of the same design: one written in ANSI-C, which is written for simulation, and one written in register transfer level HDL, which is the actual product. Motivation of the HW-CBMC is to reduce additional time for debugging and testig of the HDL implementation in order to produce the chip as soon as possible. The HW-CBMC reduces cost by providing automated way of establishing the consistency of HDL implementation using the ANSI-C implementation as a reference, because debugging and testing cost of the ANSI-C implementation is usually lower.

The HW-CBMC verifies the consistency of the HDL implementation written in Verilog using the ANSI-C implementations as a reference. The data in the Verilog modules is available to the C program by means of global variables, and the Verilog model makes a transition once the function *next\_timeframe()* is called in C program. The HW-CBMC provides counterexample when a condition of the

function *assert(condition)* in C program is not satisfied with two trace: One for the C program and a separate trace for the Verilog module. The values of the registers in the Verilog module are also shown in the C trace as part of the initial state.

### 3 Equivalence Checking

#### 3.1 Equivalence Checking Process

This section introduces how the equivalence checking works in a software development process for nuclear power plant’s reactor protection system. A part of existing software development process for KNICS’s ARP-1400 RPS is described in a upper part of Fig. 2 devided by dotted line. Each development phase has verification or testing techniques to guarantee its correctness. The design phase uses model checking techniques to verify it against important design properties. The design written in FBDs is translated into an input language of specific model checker such as SMV [10] and the model checker verifies that the design program satisfies its properties. After the model checking, the code generator generates an implementation written in ANSI-C from the design program. The ANSI-C program is compiled into executable machine code for PLCs, and the testing of the executable machine code is performed after being loaded on PLCs. The code

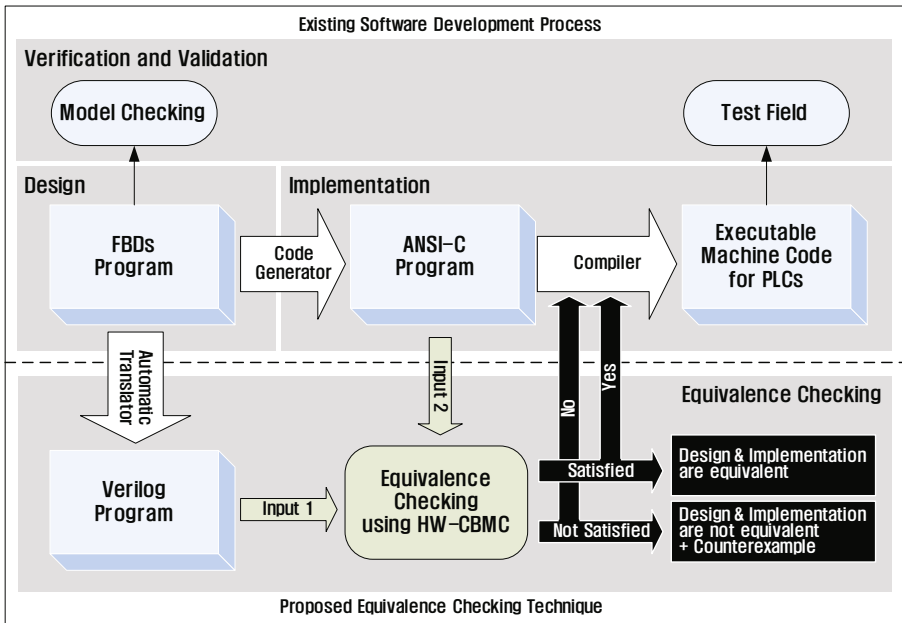


Fig. 2. A part of proposed software development process using equivalence checking with POSAFE-Q Software Engineering Tool

generator must guarantee the correctness of its behavior, since it has to translate all of behavior and feature of FBDs program to ANSI-C program precisely. It, therefore, is necessary to use a certified code generator or prove the correctness of a code generator mathematically. The mathematical proof is difficult to apply and requires high expenditure. It also requires additional proofs whenever the code generator is upgraded.

Our approach of verifying consistency between design and implementation programs includes the equivalence checking using the HW-CBMC, described in a lower part of Fig. 2, without considering the code generator. The HW-CBMC for equivalence checking requires two input programs, Verilog and ANSI-C programs. The language of implementation program, the ANSI-C program, is the same with one of input programs of the HW-CBMC, while the design program is different. We, therefore, should translate the design program written in FBDs into semantically equivalent Verilog program. If the equivalence checking is satisfied, then the ANSI-C program will be compiled into executable machine code for PLCs. If the equivalence checking, on the other hand, is not satisfied, then we will have to look into the code generator in depth.

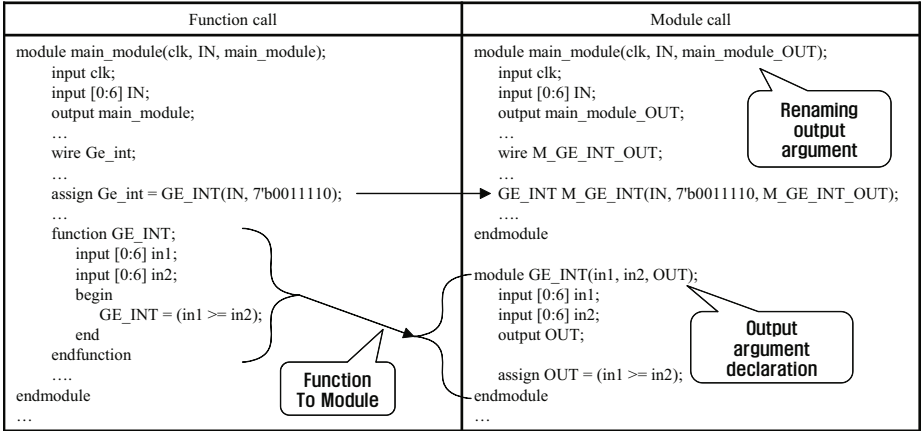
### 3.2 Verilog Program for HW-CBMC

We found that the HW-CBMC has specific rules for input programs. Verilog program as an input language of the HW-CBMC should keep the rules as following:

- A variable's name should be different from the module's name which defines and uses it.
- Function calls are not allowed.

Our research team has developed automatic *FBDtoVerilog* translators [6][12], and the current version uses the rule that a module name should be the same as that of its output variable name, in accordance with the commonly accepted usage of FBDs. The HW-CBMC maps variables which are defined as global variables in ANSI-C program onto the data in Verilog modules. If there is a variable which has same name with its module, the HW-CBMC maps variables incorrectly. The variables of module, therefore, must have a different name with the name of its module.

The *FBDtoVerilog* translates each function block in FBDs into a Verilog function, and call it according to the execution order in the Verilog module definition. However, we found that the HW-CBMC does not allow to use function calls. In order to allow Verilog program as one input of HW-CBMC, we must translate a function into a module and the effect of the modification should be analyzed further. Fig. 3 shows an example of the translation. We first translated a function *GE\_INT* into a module *GE\_INT()*, moved it out from *main\_module()*. Next, we declared input arguments and an additional output argument. We replaced the function call, *GE\_INT(IN, 7'b0011110)*, with the module call, *GE\_INT M\_GE\_INT(IN, 7'b0011110, M\_GE\_INT\_OUT)*. The module calls must follow order of function blocks of FBDs in order to make behavior



**Fig. 3.** An example of translation from a function to a module

of the Verilog program same with the FBDs. Fig. 3 also shows renaming output variable of *main\_module*.

## 4 Case Study

We applied the proposed equivalence checking approach to one of 18 shutdown logic programs, named *th\_X\_Pretrip*, in ARP-1400 RPS developed in Korea. We first describe the FBD of *th\_X\_Pretrip* and Verilog program which was automatically generated by the *FBDtoVerilog 1.0* in Subsection 4.1. Then we introduce the desirable modifications required to use the HW-CBMC. Subsection 4.2 shows the details of C programs generated by the C code generator. Subsection 4.3 shows the equivalence checking of the Verilog and C program in details.

### 4.1 *th\_X\_Pretrip* Program

The *th\_X\_Pretrip* logic consists of 8 Function Blocks as depicted in Fig. 4. It creates a warning signal, *th\_X\_Pretrip* (name of logic and output could be same), when the pretrip condition (e.g., reactor shutdown) remains true for *k\_Trip\_Delay* time units as implemented in the TOF function block. The number in parenthesis above each function block denotes its execution order. The output *th\_Prev\_X\_Pretrip* from MOVE stores current value of *th\_X\_Pretrip* for using in the next execution cycle. A large number of FBD is assembled hierarchically and executed according to predefined sequential execution order.

As described in Fig. 5, the Verilog program for the *th\_X\_Pretrip* logic has two inputs, *clk* and *f\_x*, and one output, *th\_X\_Pretrip*. 7 functions and one module match with function blocks of the *th\_X\_Pretrip* FBDs. The output, *th\_X\_Pretrip*, will become 0 when the trip condition remains true for 5 time units which TOF module counts. We modified automatically generated Verilog program. We first



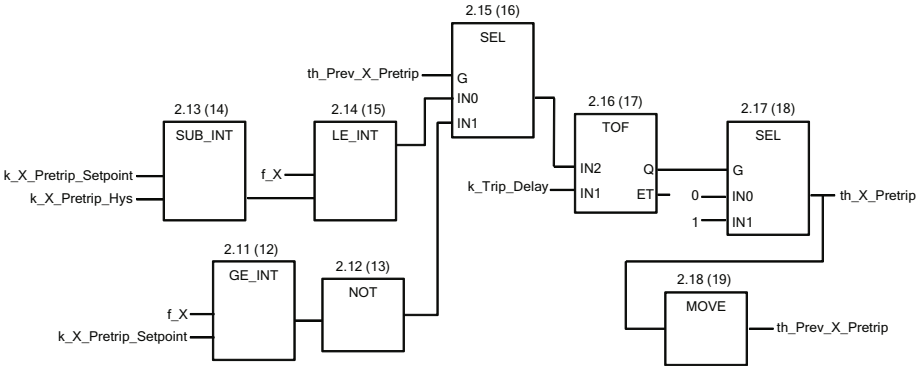


Fig. 4. Function block diagrams of *th\_X\_Pretrip*

Automatically generated Verilog program	Modified Verilog program
<pre> module th_X_Pretrip(clk, f_X, th_X_Pretrip); input clk; input [0:6] f_X; output th_X_Pretrip; ... wire Ge_int; ... reg th_prev_X_Pretrip; initial th_prev_X_Pretrip = 1; assign Ge_int = GE_INT(f_X, 7'b0011110); ... TOF M1(clk, Sel1, 7'b0000101, tof_out); assign th_X_Pretrip = MOVE(Sel2); ... always @(posedge clk) begin th_prev_X_Pretrip = th_X_Pretrip; end  function GE_INT; input [0:6] in1; input [0:6] in2; begin GE_INT = (in1 &gt;= in2); end endfunction  function NOT; ... endmodule  module TOF(clk, IN, DELAY, OUT); ... endmodule </pre>	<pre> module th_X_Pretrip(clk, f_X, th_X_Pretrip_OUT); input clk; input [0:6] f_X; output th_X_Pretrip_OUT; ... wire M_GE_INT_OUT; ... reg th_prev_X_Pretrip; initial th_prev_X_Pretrip = 1; ... GE_INT M_GE_INT (f_X, 7'b0011110, M_GE_INT_OUT); ... TOF M_TOF(clk, M11OUT, 7'b0000101, M_TOF_OUT); ... assign th_X_Pretrip_OUT = M_MOVE_OUT;  always @(posedge clk) begin th_prev_X_Pretrip = th_X_Pretrip; end endmodule  module GE_INT(in1, in2, OUT); input [0:6] in1; input [0:6] in2; output OUT;  assign OUT = (in1 &gt;= in2); endmodule  module NOT(in1, OUT); ... endmodule  module TOF(clk, IN, DELAY, OUT); ... endmodule </pre>
<p>Call modules in execution order of FBDS</p>	<p>Renaming output argument</p>
<p>Function To Module</p>	<p>Output argument declaration</p>

Fig. 5. The automatically generated Verilog program from *th\_X\_Pretrip* FBDs and its modified one

translated functions into modules. Next we changed function call to module call. The order of module call must follow the execution order of FBDs. Fig. 5 shows difference between the two Verilog programs.

## 4.2 Implementation of ANSI-C Program

We implemented ANSI-C program which has same behavior with *th\_X\_Pretrip* logic (Fig. 6). The program includes input value generation, synchronization with the Verilog program and equivalence checking property. The implemented ANSI-C program works according to the following steps:

- Step 1** generate input value nondeterministically,  $f_X = \text{nondet.int}()$ , and synchronize inputs with Verilog program using *set\_input()* statement.
- Step 2** update conditions,  $\text{Cond}_{\{a,b,c\}_1}$  (corresponding FB from 2.11 to 2.15), and a output signal, *th\_X\_Pretrip\_OUT*.
- Step 3** count time unit (corresponding FB 2.16), *timer*, where the input variable is over the limitation named *k\_Pretrip\_Setpoint*.
- Step 4** control state of the program, *state*, by checking the conditions.
- Step 5** check equivalence of output of ANSI-C and Verilog programs using *assert(th\_X\_Pretrip.th\_X\_Pretrip\_OUT == th\_X\_Pretrip\_OUT)* statement and make a transition of the Verilog program using *next\_timeframe()* function.

One loop of for statement means one transition of the Verilog program, because the for loop statement has one *next\_timeframe()* function. We, therefore, could check equivalence between output of Verilog and variable of C program every transitions. The *assert(th\_X\_Pretrip.th\_X\_Pretrip\_OUT == th\_X\_Pretrip\_OUT)* statement means that the checking will stop if the output of Verilog program is not same with the variable of ANSI-C program. If the condition is not satisfied, then the HW-CBMC will make two counterexamples of ANSI-C and Verilog program.

## 4.3 Euqivalence Checking

In order to verify equivalence between the Verilog and ANSI-C programs using HW-CBMC, we executed following statement in Visual Studio command prompt:

```
>hw-cbcm.exe th_X_Pretrip.v th_X_Pretrip.c --module Pretrip
--bound 20
```

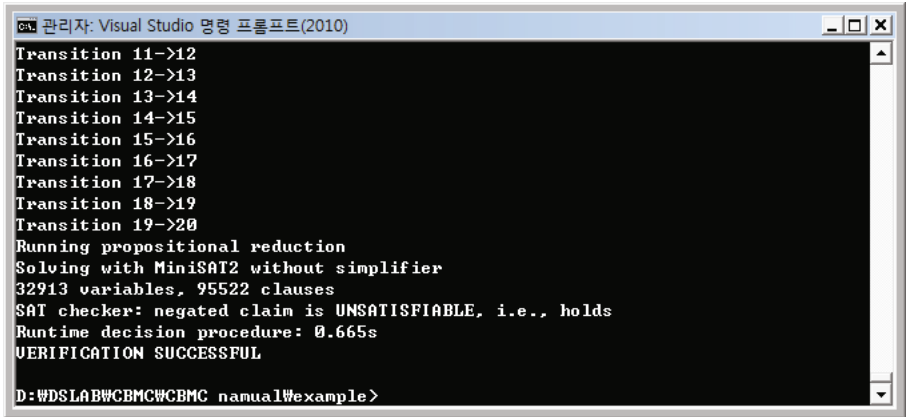
The *th\_X\_Pretrip.v* is the modified Verilog program of the automatically generated Verilog from FBDs. The *th\_X\_Pretrip.c* is the ANSI-C program which is implemented. The main module of the Verilog program is declared using the option *module*. The option *bound* specifies the number of times the transition relation of the module *Pretrip*. HW-CBMC result shows "VERIFICATION SUCCESSFUL" message which means that the Verilog and ANSI-C programs

```

1  ...
2  struct module_Pretrip {
3      unsigned int f_X;
4      unsigned int th_X_Pretrip_OUT;
5  };//definition of the variables that holds the value of the Verilog module
6  extern struct module_Pretrip th_X_Pretrip;
7  int main()
8  {
9      unsigned int f_X=0;
10     ... //variable declaration
11
12     for(cycle=0; cycle<bound; cycle++)
13     {
14         //synchronizing Inputs
15         f_X = nondet_int(); th_X_Pretrip.f_X = f_X; set_inputs();
16
17         Cond_a_1 = (f_X >= k_Pretrip_Setpoint);
18         Cond_b_1 = ((f_X >= k_Pretrip_Setpoint) && (timer == 5));
19         Cond_c_1 = (f_X <= k_Pretrip_Setpoint - k_X_Pretrip_Hys);
20         th_X_Pretrip_OUT = (state==0 && Cond_a_1)?th_Prev_X_Pretrip:
21             (state==0 && !Cond_a_1)?th_Prev_X_Pretrip:
22             (state==1 && !Cond_a_1 && !Cond_b_1)?th_Prev_X_Pretrip:
23             (state==1 && Cond_a_1 && !Cond_b_1)?th_Prev_X_Pretrip:
24             (state==1 && Cond_b_1)?0:
25             (state==2 && Cond_c_1)?1:th_Prev_X_Pretrip;
26         if(f_X >= k_Pretrip_Setpoint) //count the time unit
27             if(timer == 5) timer == 5;
28             else timer++;
29         else
30             timer=0;
31         //assertion statement
32         assert(th_X_Pretrip.th_X_Pretrip_OUT == th_X_Pretrip_OUT);
33
34         th_Prev_X_Pretrip = th_X_Pretrip_OUT;
35         switch(state) //control a state by conditions
36         {
37             case 0:
38                 if(Cond_a_1) state = 1;
39                 else state = 0;
40             break;
41             case 1:
42                 ...
43         }
44         next_timeframe();
45     }
46 }

```

**Fig. 6.** Implemented ANSI-C program



```
관리자: Visual Studio 명령 프롬프트(2010)
Transition 11->12
Transition 12->13
Transition 13->14
Transition 14->15
Transition 15->16
Transition 16->17
Transition 17->18
Transition 18->19
Transition 19->20
Running propositional reduction
Solving with MiniSAT2 without simplifier
32913 variables, 95522 clauses
SAT checker: negated claim is UNSATISFIABLE, i.e., holds
Runtime decision procedure: 0.665s
VERIFICATION SUCCESSFUL
D:#DSLAB\CBMC\CBMC namual\example>
```

**Fig. 7.** A screen dump of equivalence checking result between *th\_X\_Pretrip.v* and *th\_X\_Pretrip.c* program

produced same output against the same input generated randomly. Fig. 7 shows a screen dump of the equivalence checking result.

As a result of this case study, we concluded that the HW-CBMC is effective for our proposal. Although there is gap between automatically generated Verilog program and an input program of the HW-CBMC, we could check equivalence between both if the Verilog program modified successfully.

## 5 Conclusion

In this paper, we have proposed equivalence checking approach between design written in FBDs and its implementation program written in ANSI-C using HW-CBMC. The FBDs should be translated into Verilog language in order to make the FBDs into an input of the HW-CBMC. The automatically translated Verilog program, however, was not exactly appropriate to be the input. We modified some features of the Verilog program, and made it into the input of HW-CBMC. As a result of the case study, the equivalence checking between FBDs and ANSI-C programs using HW-CBMC is effective.

We are planning to verify equivalence between FBDs and its automatically generated ANSI-C program by pSET. We are also planning to modify translation rules from FBDs to Verilog in order to make automatically generated Verilog program appropriate to the HW-CBMC and develop a translator for automation of this process.

**Acknowledgments.** This research was partially supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)” (NIPA-2011-(C1090-1131-0008)) and the KETEP

(Korea Institute of Energy Technology Evaluation And Planning)”(KETEP-2010-T1001-01038), the Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2010-0002566) and the IT R&D Program of MKE/KEIT [10035708, ”The Development of CPS(Cyber-Physical Systems) Core Technologies for High Confidential Autonomic Control Software”].

## References

1. Korea Nuclear Instrumentation & Control System R&D Center, <http://www.knics.re.kr/>
2. Cho, S., Koo, K., You, B., Kim, T.-W., Shim, T., Lee, J.S.: Development of the loader software for PLC programming. In: Proceedings of Conference of the Institute of Electronics Engineers of Korea, vol. 30(1), pp. 595–960 (2007)
3. Hoare, T.: The Verifying Compiler: A Grand Challenge for Computing Research. *Journal of the ACM* 50, 63–69 (2003)
4. RETRANS, Institute for Safety Technology (ISTec), [http://www.istec.grs.de/en/produkte/leittechnik/retrans.html?pe\\_id=54](http://www.istec.grs.de/en/produkte/leittechnik/retrans.html?pe_id=54)
5. Clarke, E., Kroening, D.: Hardware verification using ANSI-C programs as a reference. In: Proceedings of the 2003 Asia and South Pacific Design Automation Conference, pp. 308–311 (2003)
6. Yoo, J., Cha, S., Jee, E.: Verification of PLC programs written in FBD with VIS. *Nuclear Engineering and Technology* 41(1), 79–90 (2009)
7. IEEE: IEEE standard hardware description language based on the Verilog hardware description language. (IEEE Std. 1364-2001) (2001)
8. Bombieri, N., Fummi, F., Pravadelli, G., Marques-Silva, J.: Towards Equivalence Checking Between TLM and RTL Models. In: 5th IEEE/ACM International Conference on Formal Methods and Models for Codesign, MEMOCODE 2007, pp. 113–122 (2007)
9. Sangiovanni-Vincentelli, A., Aziz, A., Cheng, S.-T., Edwards, S., Khatri, S., Kukimoto, Y., Qadeer, S., Shiple, T.R., Swamy, G., Hachtel, G.D., Somenzi, F., Pardo, A., Ranjan, R.K., Brayton, R.K.: VIS: A System for Verification and Synthesis. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 428–432. Springer, Heidelberg (1996)
10. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers, Dordrecht (1993)
11. IEC (International standard for programmable controllers): Programming languages 61131- Part 3 (1993)
12. Jee, E., Jeon, S., Cha, S., Koh, K., Yoo, J., Park, G., Seong, P.: FBDVerifier: Interactive and Visual Analysis of Counterexample in Formal Verification of Function Block Diagram. *Journal of Research and Practice in Information Technology* 42(3), 255–272 (2010)

# A Framework for Simulation and Symbolic State Space Analysis of Non-Markovian Models

Laura Carnevali, Lorenzo Ridi, and Enrico Vicario

Dipartimento di Sistemi e Informatica - Università di Firenze  
{laura.carnevali,lorenzo.ridi,enrico.vicario}@unifi.it

**Abstract.** Formal methods supporting development of safety-critical systems require tools that can be integrated within composed environments. Sirio is a framework for simulation and analysis of various timed extensions of Petri Nets, supporting correctness verification and quantitative evaluation of timed concurrent systems. As a characterizing trait, Sirio is expressly designed to support reuse and to facilitate extensions such as the definition of new reward measures, new variants of the analysis, and new models with a different semantics. We describe here the functional responsibilities and the SW architecture of the framework.

**Keywords:** Correctness verification, quantitative evaluation, preemptive Time Petri Net, non-Markovian Stochastic Petri Net, stochastic Time Petri Net, symbolic state space analysis, steady state evaluation, transient evaluation.

## 1 Introduction

In safety-critical systems, failures can lead to environmental harm, human property damage, injury or even loss of human life. This characterizes a wide and increasing spectrum of application areas including industrial automation, robotics, railway and flight control, military and space missions. Formal methods can largely help the development of safety-critical systems by supporting formal specification of requirements, early verification and evaluation of design choices, automated code derivation, test planning and execution. The adoption of formal methods is explicitly recommended by certification standards [29], [13], [25] as a means to achieve high Safety Integrity Levels (SILs) in the development of SW with concurrency control, synchronization mechanisms, and distributed processing. The integration of formal methods in the SW life cycle has been actually practiced in several Model Driven Development (MDD) frameworks [11], [31], [1], [21]. To this end, a number of tools have been developed that support correctness verification and/or quantitative evaluation of concurrent timed models. While the former is targeted to the identification of the set of feasible behaviors, the latter is aimed at providing a measure of their probability.

Various tools for correctness verification develop upon the formalisms of Timed Automata (TA) [28], [5] and Time Petri Nets (TPNs) [26], [2], [16], [18], [33], [10], [22], [9]. In the area of TA, Uppaal [5], [4] is the most well-established tool

for verification of real-time systems modeled as networks of TA enriched with features that extend their expressivity. In particular, the tool supports schedulability analysis of preemptive task-sets with asynchronous and dense release times, under the assumption that the model does not include at the same time both nondeterministic Execution Times and dependencies among release and completion times of tasks [17]. In the area of TPNs, the Oris Tool [8] implements symbolic state space analysis of preemptive Time Petri Nets (pTPNs) [18], [11], an extension of TPNs that encompasses a mechanism of suspension and resume in the advancement of clocks, enabling schedulability analysis of real-time systems running under priority preemptive scheduling. Tina [3] and Romeo [19] support the construction of various abstract state space representations and the model-checking of reachability properties. In particular, Romeo also supports approximate and exact state space enumeration of Scheduling-TPNs, an extension of TPNs with an expressivity comparable with pTPNs.

Several tools for quantitative evaluation support the derivation of performance and dependability rewards. SHARPE [32] is a SW tool for specification and analysis of performance and reliability models including including Fault Trees (FTs) and Fault Trees with Repeated Events (FTREs), reliability graphs, series-parallel acyclic directed graphs, product-form queuing networks, Markov and semi-Markov chains, Generalized Stochastic Petri Nets (GSPNs), and Markov Regenerative Processes (MRPs) [24]. In particular, it facilitates the hierarchical combination of different model types. DEEM [7] is a dependability modeling and evaluation tool specifically tailored for Multiple Phased Systems, which implements the solution procedure of Deterministic Stochastic Petri Nets (DSNPs) underlying a MRP. TimeNet [35] mainly supports modeling and evaluation of extended Deterministic Stochastic Petri Nets (eDSNPs), which include transitions with either immediate, deterministic, exponentially distributed, or expolynomially distributed firing time [30], [14]. In particular, the analysis is performed under the assumption that at most one generally distributed (GEN) transition is enabled in any reachable tangible marking (enabling restriction). The Web-SPN Tool [6] implements analysis of models with multiple concurrently enabled timers through a discrete abstraction of time, encompassing any kind of GEN distribution and different memory policies. The Oris Tool [8] supports simulation and analysis of stochastic Time Petri Nets (sTPNs) [33], [10], [22], [9], a variant of non-Markovian SPNs that extends TPNs with a stochastic characterization of time distributions and choices. As a characterizing trait, the tool implements symbolic state space enumeration of sTPN models that may include multiple concurrent GEN timers with possibly overlapping activity cycles, provided that timers are associated with an expolynomial distribution, enabling derivation of steady-state and transient probabilities of the underlying Generalized Semi-Markov Process (GSMP) [27].

The ever increasing variety of modeling formalisms and solution techniques gives relevance to integration frameworks and reusable components that can support multi-formalism modeling and multi-solution evaluation of complex and heterogeneous systems. Möbius [15] is a multi-paradigm multi-solution framework

which provides an extensible environment for dependability, security, and performance modeling of large-scale discrete-event systems. In particular, it supports numerical and analytical solution techniques for specific Markovian models as well as discrete event simulation of a more general class of models. The OsMoSys Multi-solution Framework [34] comprises a SW environment for the analysis of multi-formalism models, which provides a strong separation between model representation and analysis algorithms. This guarantees a high flexibility both in modeling and solution phases, and permits to combine different formalisms and to concurrently apply multiple solvers. *SIMTHESys* [23] is an open framework supporting compositional modeling of complex systems based on formalisms such as Stochastic Petri Nets (SPNs), Queuing Networks (QNs), Bayesian Networks (BNs), and FTs. *DrawNet* [20] is a customizable tool for design and solution of models expressed in any graph based formalism, including Fluid Stochastic Petri Nets (FSPs), Dynamic Parametric Fault Trees (DPFTs), and BNs.

In this paper, we present a new framework named *Sirio* for modeling, simulation, and analysis of various timed extensions of Petri Nets, which notably include pTPNs and sTPNs. As a characterizing feature, the SW architecture of *Sirio* is purposely designed to guarantee high reusability of the code and to support the implementation of extensions that add new functionalities or modify the existing ones. This largely eases SW maintenance, helps the validation of new theoretical developments, and facilitates application within an existing tool such as *Oris* [8] or integration within a composed environment such as *Möbius* [15] or *OsMoSys* [34]. The SW architecture is organized in: *i*) three base libraries supporting representation of the structural elements of Petri Net models, manipulation of expolynomial functions, and generation of samples from expolynomial distributions; *ii*) two component tools implementing the semantics of Petri Net models and their simulation/analysis. *Sirio* is developed by the Software Technologies Laboratory of the University of Florence (<http://www.stlab.dsi.unifi.it>) and is available for experimentation.

The rest of the paper is organized as follows. Section 2 describes functional responsibilities of the *Sirio* framework; Sections 3 and 4 present the SW architecture of *Sirio* base libraries and tools, respectively; Section 5 finally draws conclusions.

## 2 Sirio Functional Responsibilities

The *Sirio* framework includes three base libraries and two component tools, as shown in Fig. 1. The libraries provide basic functionalities for model representation and manipulation; the tools support model simulation and analysis.

### 2.1 Sirio Base Libraries

The Petri Net Library provides support to the representation of various kinds of Petri Nets models including:



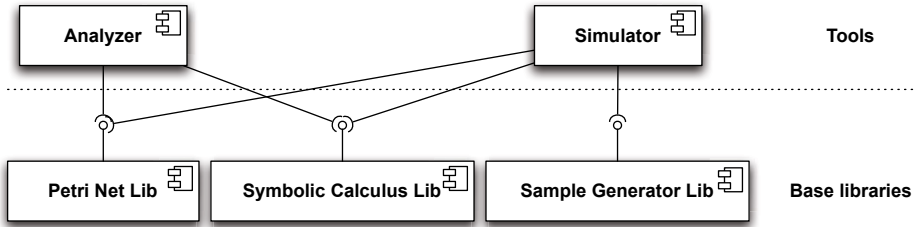


Fig. 1. The SW architecture of the Sirio framework

- *Time Petri Nets* (TPNs) [16], which enable reachability and timeliness analysis of densely-timed models;
- *preemptive Time Petri Nets* (pTPNs) [18], which support schedulability analysis of real-time systems running under priority preemptive scheduling;
- *stochastic Time Petri Nets* (sTPNs) [33], [10], [22], which enable derivation of steady-state and transient probabilities of models with multiple concurrently enabled GEN timers that underlie a GSMP [27], provided that timers are associated with an expolynomial distribution [30], [36], [14];
- *stochastic preemptive Time Petri Nets* (spTPNs) [9], which support steady-state evaluation of densely-timed preemptive systems with non-Markovian temporal parameters, under the assumption that timers have expolynomial distribution with non-pointlike support and those with unbounded support are all distributed over  $[0, \infty)$  with negative exponential distribution.

The Symbolic Calculus Library supports representation and manipulation of mathematical expressions and functions supported over polyhedral and Difference Bound Matrix (DBM) domains. In particular, the library supports manipulation of expolynomials, which constitute a fairly general class of expressions including constants, exponentials, and polynomials. An expolynomial is an expression of the type:  $\sum_{h=1}^H c_h \prod_{n=1}^N x_n^{\alpha_n} e^{-\lambda_n x_n}$ .

The Sample Generator Library supports the generation of pseudo-random samples from an expolynomial probability distribution function  $F(x)$ , using different methods depending on the form of the distribution. More specifically, if  $F$  is invertible with inverse function  $F^{-1} = f$ , samples can be generated through the method of *Symbolic Inversion*: given a sample  $y$  from the uniform distribution over  $[0, 1]$ , a new sample from  $F$  is obtained as  $f(y)$ . Otherwise, if  $F$  is not invertible, samples can be generated through:

- the method of *Numeric Inversion*: given a sample  $y$  from the uniform distribution over  $[0, 1]$ , a new sample from  $F$  is obtained as the unique root of the monotonic and differentiable function  $F(x) - y$ . The root is estimated through the method of Newton: starting from an initial guess reasonably close to the unknown root value, root approximations are iteratively derived by computing the x-intercept of the tangent line that passes from the function point with x-coordinate equal to the current approximation.

- the method of *Acceptance-Rejection*: given an envelope distribution  $G(x)$  such that  $F(x) < K \cdot G(x) \forall x$  for some constant  $K > 1$  and a sample  $y$  from the uniform distribution over  $[0, 1]$ , a new sample from  $F$  is obtained as a sample  $\bar{x}$  from  $G$  that satisfies  $\bar{x} < F(x)/(K \cdot G(x))$
- the algorithm of *Metropolis-Hastings*, a Markov Chain Monte Carlo method which generates a Markov Chain in which each state  $x_k$  depends on only on the previous state  $x_{k-1}$ : given a proposal probability distribution  $G(x, x_k)$  depending on the current state  $x_k$  and given a sample  $y$  from the uniform distribution over  $[0, 1]$ , a new sample from  $F$  is obtained as a sample  $\bar{x}$  from  $G$  that satisfies  $y < (F(\bar{x})G(x_{k-1})) / (F(x_{k-1})G(\bar{x}))$ .

## 2.2 Sirio Tools

The Simulation Tool performs stochastic simulation of Petri Net models and supports the evaluation of *transient* and *steady-state* rewards, both in *discrete time* and in *continuous time*, i.e., the probability of a marking and the mean time between the firings of two transitions.

The Analysis Tool enables exhaustive state space enumeration of Petri Net models. Since the Petri Net library supports the representation of a large number of model types, the Analysis Tool implements various kinds of analysis, allowing selection of analysis-dependent parameters. In particular, the tool supports both steady state and transient analysis.

## 3 Sirio SW Architecture: Base Libraries

### 3.1 Petri Net Library

Fig. 2 shows the SW architecture of the Petri Net Library, which reflects the actual syntax of a Petri Net: the `PetriNet` class aggregates places, transitions, precondition and postcondition arcs, represented by the `Place`, `Transition`, `Precondition`, and `Postcondition` classes, respectively. Each of these classes is only responsible for maintaining a unique identifier for the associated component together with a list of *features* enriching the behavior of the component, i.e., the `PlaceFeature`, `TransitionFeature`, `PreconditionFeature`, and `PostconditionFeature` interfaces. As a relevant example, we describe here some of the features associated with the `Transition` class, which are classes that implement the `TransitionFeature` interface:

- `TimeTransitionFeature` encodes the temporal information of a timed transition, as defined by the syntax of TPNs [26], [2], [16], i.e., a time interval  $[EFT, LFT] \in \mathbb{R}_0^+ \times \mathbb{R}_0^+ \cup \{\infty\}$ ;
- `StochasticTransitionFeature` represents the probability density function associated with a transition in sTPN models [33], [10];
- `PreemptiveTransitionFeature` enriches the associated transition with a list of resources and a priority value, so as to make it suitable for the analysis in the preemptive setting defined by the theory of pTPNs [18].

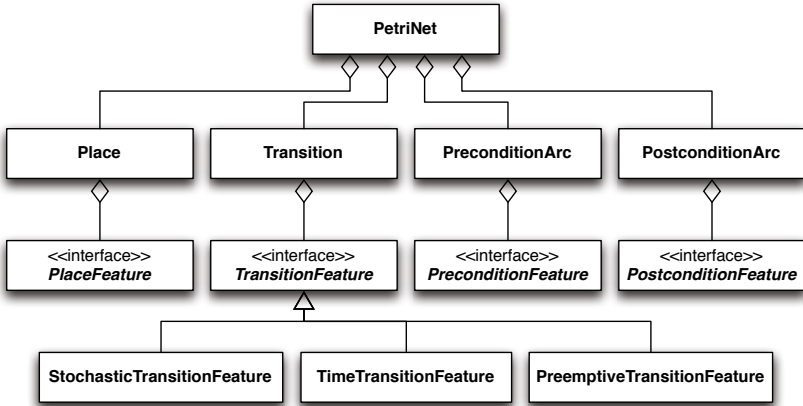


Fig. 2. A portion of the SW architecture of the Petri Net library

As a characterizing trait, the variety of Petri Net models is not *explicitly* represented, but *implicitly* defined through the features associated with structural elements of the net. This provides a strong modeling flexibility, supporting the representation of hybrid models such as *partially stochastic Time Petri Nets* (pTPNs) [12] which include both non-deterministic and stochastic transitions, and guarantees high reusability of the code, since the implementation of new model types amounts to the implementation of one or more features for structural net components.

### 3.2 Symbolic Calculus Library

Fig. 3 shows the SW architecture of the Symbolic Calculus Library. An exponential expression [30], [36], [14] is represented by the `Expolynomial` class, which implements the `Expression` interface and aggregates instances of the `Expmonomial` class representing terms of the form  $c_h \prod_{n=1}^N x_n^{\alpha_n} e^{-\lambda_n x_n}$ . In turn, an `Expmonomial` aggregates atomic terms which can be instances of the `MonomialTerm` class or the `ExponentialTerm` class. The `Domain` interface represents a multi-dimensional domain and it is specialized into `PolyhedralDomain` and `DBMDomain`. A function is an expression over a domain and is represented by the `Function` class, which maintains a reference to the `Domain` interface and the `Expression` interface.

### 3.3 Sample Generator Library

Fig. 4 shows the SW architecture of the Sample Generator Library. Each supported sample generation technique is implemented by a concrete class that realizes the `Sampler` interface, i.e., `SymbolicInversion`, `NumericInversion`, `AcceptanceRejection`, and `MetropolisHastings`.

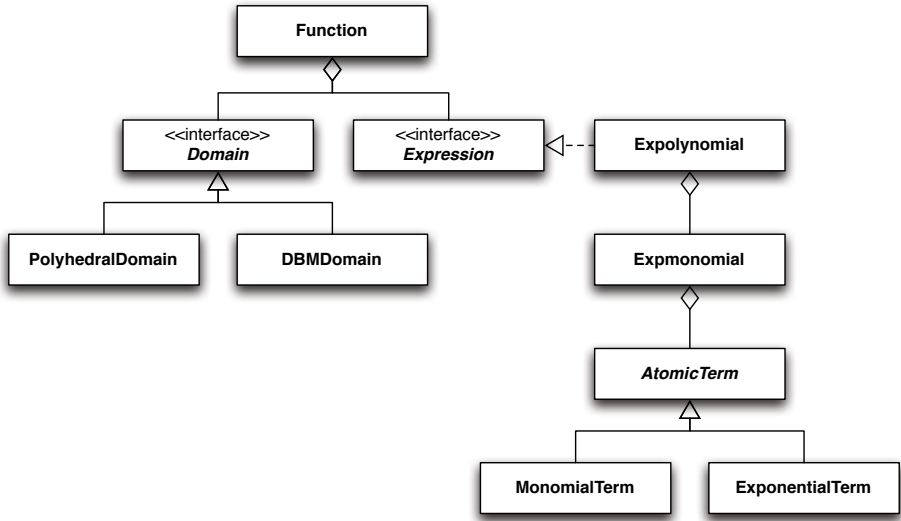


Fig. 3. A portion of the SW architecture of the Symbolic Calculus library

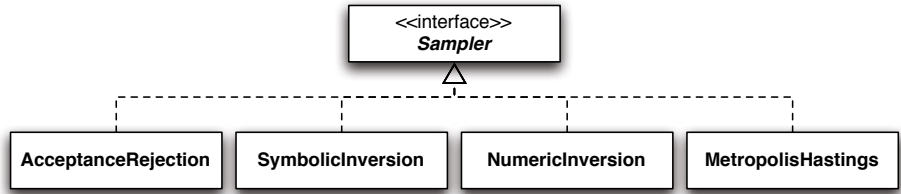
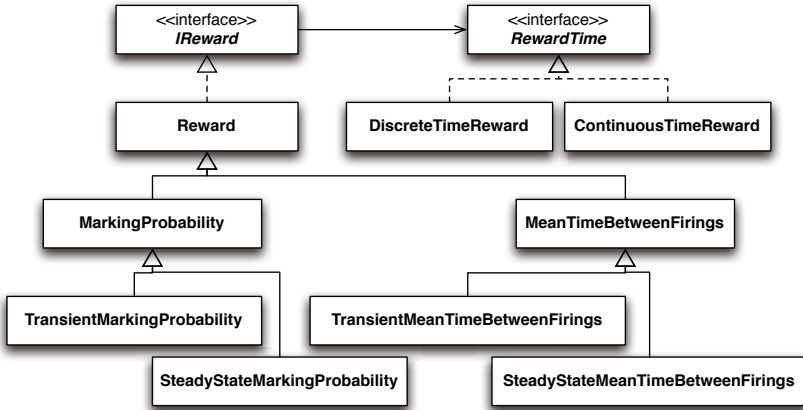


Fig. 4. The architecture of the Sample Generator library

## 4 Sirio SW Architecture: Tools

### 4.1 Simulation Tool

Fig. 5 shows a portion of the SW architecture of the Simulation Tool concerning the evaluation of reward measures. The `Reward` abstract class implements the `IReward` interface and performs operations that are common to the evaluation of all supported reward types, each represented by a class that extends `Reward`. In particular, the `MarkingProbability` and `MeanTimeBetweenFirings` abstract classes model the two reward types supported by the tool and each one is specialized in two different concrete classes for transient and steady-state regime, respectively. The tool uses composition instead of inheritance to decouple the reward interface from its implementation in continuous/discrete time, allowing agile definition of new reward types. According to this, basic operations that are implemented in a different manner depending on the domain of time are abstracted by the `RewardTime` interface, which is implemented by the `DiscreteTimeReward` and `ContinuousTimeReward` concrete classes.



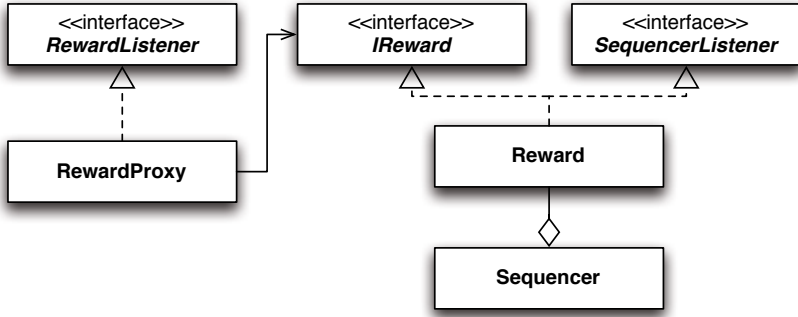
**Fig. 5.** A portion of the SW architecture of the Simulation Tool, concerning the evaluation of rewards

Fig. 6 shows a portion of the SW architecture of the tool concerning the implementation of the simulation process. The **Sequencer** class is responsible for the evolution of the model and maintains a reference to the **SuccessorEvaluator** interface, which performs the derivation of the successor state; in turn, a state is represented by the **SimulatorState** interface (see also Fig. 9). The simulation process is performed according to the following algorithm:

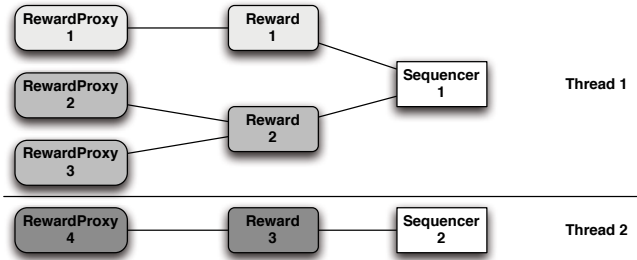
- compute the initial state  $s_0$ ;
- at the  $n$ -th step, until a stop condition is false:
  - compute the set  $T^f(s_{n-1})$  of transitions that are fireable in  $s_{n-1}$ ;
  - select a transition  $t$  in  $T^f(s_{n-1})$ ;
  - compute the successor  $s_n$  of  $s_{n-1}$  through the firing of  $t$ .

The **Sequencer** class is responsible for notifying the **Reward** class of each executed step of simulation; in turn, the **Reward** class updates reward measures at each simulation step and notifies the **RewardProxy** class which is responsible for maintaining the result of reward evaluation. The simulation process is iterated until an assigned Execution Time has elapsed or until the model has executed an assigned number of runs with an assigned number of steps.

Fig. 7 illustrates the allocation of the evaluation of reward measures to different threads. Each reward to be evaluated is associated with an object of the **RewardProxy** class. In particular, rewards of the same type that have the same domain of time are grouped together and the corresponding **RewardProxy** objects are associated with the same **Reward** object. In turn, **Reward** objects that share the same Petri Net model, the same sampling generation techniques for the selection of times-to-fire of model transitions, and the same regime are grouped together and associated with the same **Sequencer** object. Finally, the simulation process managed by each **Sequencer** object is allocated exclusively to a different thread. In so doing, the evaluation of rewards is parallelized and the number of



**Fig. 6.** A portion of the SW architecture of the Simulation Tool, concerning the implementation of the simulation process



**Fig. 7.** A schema illustrating the allocation of the evaluation of reward measures to different threads

simulation steps is notably reduced. For instance, the sequential evaluation of the same reward for an increasing number of firings equal to 10,000, 20,000, ..., and 100,000 would require 550,000 simulation steps, while the Simulation Tool completes all rewards evaluation in 100,000 steps. Moreover, the simultaneous evaluation of different rewards along the same simulation runs permits to derive statistically correlated measures.

**Implementation of New Reward Types:** amounts to the definition of an abstract class that extends `Reward`, two concrete subclasses of this class for transient and steady-state regime, and, if necessary, the addition of methods to `RewardTime` and their concrete implementation in `DiscreteTimeReward` and `ContinuousTimeReward`.

**Implementation of the Simulation Process for New Petri Net Models:** amounts to the definition of a class that implements the `SuccessorEvaluator` interface, and, in case the state of the model has to carry new additional information, the definition of a class that implements the `State` interface.

## 4.2 Analysis Tool

Fig. 8 shows a portion of the SW architecture of the Analysis Tool concerning the structure of the enumeration algorithm. The tool uses composition instead of inheritance to customize the behavior of the following general enumeration algorithm implemented by the `Analyzer` class:

- generate the initial state  $n_0$  and insert it into a queue  $Q$ ;
- until  $Q$  is empty:
  - take a new state  $n$  from queue  $Q$ ;
  - enumerate all its possible successor transitions;
  - for each possible successor transitions:
    - \* evaluate the successor state  $n'$ ;
    - \* insert  $n'$  into  $Q$ ;

The implementation of operations involved in the enumeration algorithm is delegated to objects complying with specific interfaces. The `EnumerationPolicy` interface manages the insertion and removal of states from queue  $Q$ . It is unaware of the actual type of states and only knows their order inside  $Q$ , so that different enumeration policies (i.e., LIFO, FIFO, Priority) can be implemented independently of the specific analysis type. The creation of the initial state is delegated to the `InitialClassBuilder` interface, whose implementations generate different initial states depending on the kind of analysis. In a similar manner, the `SuccessorEvaluator` interface is responsible for the evaluation of the successor state.

Composition can be further applied to each enumeration step in order to achieve higher modularity: for instance, all the objects that implement the

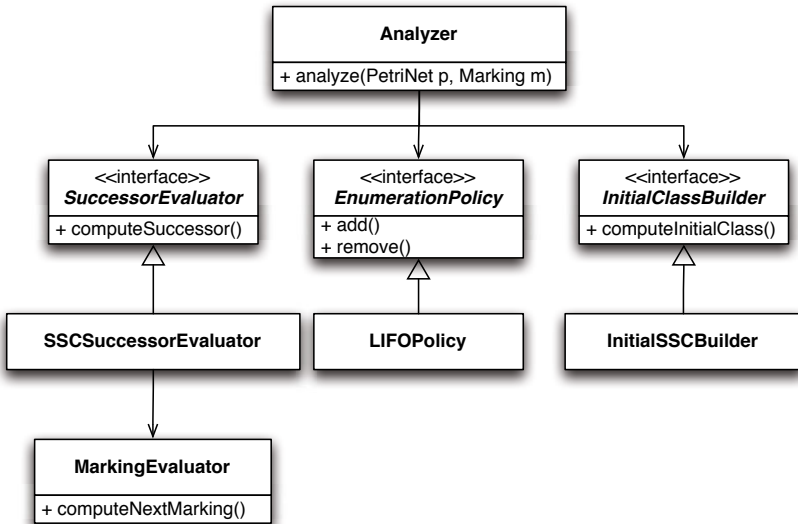


Fig. 8. A portion of the SW architecture of the Analysis Tool, concerning the structure of the enumeration algorithm

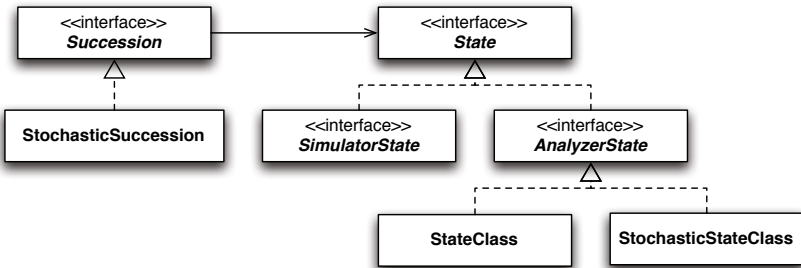
**SuccessorEvaluator** interface share the portion of code that implements the update of the marking of a Petri Net model after the firing of a transition. To guarantee a reasonable reuse of the code, the **MarkingEvaluator** class encapsulates the *token game* logic and makes it available to every class that could make use of it.

The enumeration procedure uses several structures to encode states, succession relations between states, and the overall generated state space (usually a labeled directed graph). Fig. 9 shows the portion of the SW architecture of the tool that concerns the representation of the graph of state-classes: **AnalyzerState** constitutes the common interface for all the possible kinds of states that are generated by the analyzer; the **Succession** interface represents a succession between two states, carrying additional information depending on the specific kind of analysis. Composition is also used to provide additional functionalities:

- the **SuccessionProcessor** class is responsible for processing nodes prior to the evaluation of their successors or prior to their insertion in queue *Q*: the feature is useful, for instance, when the analysis requires some kind of approximation to limit the complexity of evaluated states;
- the **StopCriterion** class supports the implementation of criteria to interrupt the enumeration process at a global level, i.e., no node is further enumerated, and at the local level, i.e., successors of a specific node are not enumerated;
- the **Logger** class is responsible for performing logging operations.

Classes implementing stop criteria and logging operations inherit some of their features from the **AnalyzerObserver** interface, so as to receive events notification during the analysis process.

**Implementation of New Types of Analysis:** amounts to the definition of appropriate delegate components of the **Analyzer** class and, if necessary, the definition of the corresponding data structure for the encoding of the state space elements. To exemplify the concept, we describe here the additions that are needed to implement the theory of transient analysis of sTPN models described in [22]. *Transient stochastic classes* extend the concept of stochastic



**Fig. 9.** A portion of the SW architecture of the Analysis Tool, concerning enumeration and representation of the graph of state-classes



state classes through the introduction of an *age clock* that accumulates the time elapsed since the initial state, enabling the evaluation of transient rewards for the underlying non-Markovian process. To encode and manipulate this additional information in the *Sirio* framework, a new `TransientStateClass` is inherited from `StochasticStateClass`. Thus, new `InitialTSCBuilder` and `TSCSuccessorEvaluator` classes are implemented in order to manage creation and evolution of this new type of state class.

## 5 Conclusions

The development of safety-critical systems may largely benefit from the application of formal methods. This requires tool support, which in turn requires integration platforms and components. *Sirio* is a framework that implements simulation and symbolic state space analysis of preemptive and stochastic extensions of TPNs, supporting an integrated approach to correctness verification and quantitative evaluation of concurrent timed systems. As characterizing features, the SW architecture of *Sirio* is easily extensible and guarantees high reusability of the code. This facilitates SW maintenance operations, provides a suitable environment where new theoretical results can be experimented, and also eases the application within an existing toolchain such as *Oris* [8] or the integration within a composed environment such as *Möbius* [15] or *OsMoSys* [34]. Most relevant elements of the theory implemented in *Siro* as well as case studies illustrating the level of affordable complexity can be found in [18], [33], [10], [22], [9].

## References

1. Alur, R., Lee, I., Sokolsky, O.: Compositional refinement for hierarchical hybrid systems. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 33–48. Springer, Heidelberg (2001)
2. Berthomieu, B., Diaz, M.: Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. on SW Eng.* 17(3), 259–273 (1991)
3. Berthomieu, B., Ribet, P.-O., Vernadat, F.: The tool TINA – Construction of Abstract State Spaces for Petri Nets and Time Petri Nets. *International Journal of Production Research* 42(14), 2741–2756 (2004)
4. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
5. Bengtsson, J., Larsen, K.G., Larsson, F., Petterson, P., Yi, W.: UPPAAL: a Tool-Suite for Automatic Verification of Real-Time Systems. In: Alur, R., Sontag, E.D., Henzinger, T.A. (eds.) HS 1995. LNCS, vol. 1066. Springer, Heidelberg (1996)
6. Bobbio, A., Puliafito, A., Scarpa, M., Telek, M.: WebSPN: A WEB-Accessible Petri Net Tool. In: Proc. Conf. on Web-based Modeling and Simulation (1998)
7. Bondavalli, A., Mura, I., Chiaradonna, S., Filippini, R., Poli, S., Sandrini, F.: DEEM: a tool for the dependability modeling and evaluation of multiple phased systems. In: *IEEE Int. Conf. on Dependable Systems and Networks, DSN* (June 2000)

8. Bucci, G., Carnevali, L., Ridi, L., Vicario, E.: Oris: a Tool for Modeling, Verification and Evaluation of Real-Time Systems. *Int. Journal of Software Tools for Technology Transfer* 12(5), 391–403 (2010)
9. Carnevali, L., Giuntini, J., Vicario, E.: A symbolic approach to quantitative analysis of preemptive real-time systems with non-markovian temporal parameters. In: *VALUETOOLS* (May 2011)
10. Carnevali, L., Grassi, L., Vicario, E.: State-Density Functions over DBM Domains in the Analysis of Non-Markovian Models. *IEEE Trans. on SW Eng.* 35(2), 178–194 (2009)
11. Carnevali, L., Ridi, L., Vicario, E.: Putting preemptive Time Petri Nets to work in a V-model SW life cycle. *IEEE Trans. on SW Eng.*, (accepted for publication)
12. Carnevali, L., Ridi, L., Vicario, E.: Partial stochastic characterization of timed runs over DBM domains. In: *Proc. of the 9th International Workshop on Performability Modeling of Computer and Communication Systems* (September 2009)
13. CENELEC. EN 50128 - Railway applications: SW for railway control and protection systems (1997)
14. Ciardo, G., German, R., Lindemann, C.: A characterization of the stochastic process underlying a stochastic Petri net. *IEEE Trans. On SW Eng.* 20(7), 506–515 (1994)
15. Courtney, T., Gaonkar, S., Keefe, K., Rozier, E., Sanders, W.H.: Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models. In: *IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, pp. 353–358 (2009)
16. Vicario, E.: Static Analysis and Dynamic Steering of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. on SW Eng.* 27(1), 728–748 (2001)
17. Fersman, E., Krcal, P., Pettersson, P., Yi, W.: Task automata: Schedulability, decidability and undecidability. *Inf. Comput.* 205(8), 1149–1172 (2007)
18. Bucci, G., Fedeli, A., Sassoli, L., Vicario, E.: Timed State Space Analysis of Real Time Preemptive Systems. *IEEE Trans. SW Eng.* 30(2), 97–111 (2004)
19. Gardey, G., Lime, D., Magnin, M., Roux, O.: Romeo: A Tool for Analyzing Time Petri Nets. In: Etessami, K., Rajamani, S.K. (eds.) *CAV 2005*. LNCS, vol. 3576, pp. 418–423. Springer, Heidelberg (2005)
20. Gribaudo, M., Codetta-Raiteri, D., Franceschinis, G.: Draw-net, a customizable multi-formalism, multi-solution tool for the quantitative evaluation of systems. In: *Int. Conf. on the Quantitative Evaluation of Systems* (2005)
21. Henzinger, T.A., Horowitz, B., Kirsch, C.M.: Giotto: A time-triggered language for embedded programming. In: *Proc. of the IEEE*, pp. 84–99. IEEE, Los Alamitos (2003)
22. Horvath, A., Ridi, L., Vicario, E.: Transient analysis of generalised semi-markov processes using transient stochastic state classes. In: *Proc. of the Int. Conf. on Quant. Eval. of Systems, QEST 2010* (2010)
23. Iacono, M., Gribaudo, M.: Element based semantics in multi formalism performance models. In: *IEEE Int. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 413–416 (2010)
24. Kulkarni, V.G.: *Modeling and analysis of stochastic systems*. Chapman & Hall, Ltd., London (1995)
25. Jordan, P.: IEC 62304 International Standard Edition 1.0 Medical device software - Software life cycle processes. In: *The Institution of Engineering and Technology Seminar on Software for Medical Devices 2006* (2006)
26. Merlin, P., Farber, D.J.: Recoverability of Communication Protocols. *IEEE Trans. on Comm.* 24(9), 1036–1043 (1976)

27. Glynn, P.W.: A GSMP formalism for discrete-event systems. *Proceedings of the IEEE* 77, 14–23 (1989)
28. Alur, R., Dill, D.L.: Automata for Modeling Real-Time Systems. In: Paterson, M. (ed.) *ICALP 1990*. LNCS, vol. 443, pp. 322–335. Springer, Heidelberg (1990)
29. Radio Technical Commission for Aeronautics. DO-178B, *Software Considerations in Airborne Systems and Equipment Certification* (1992)
30. Sahnerand, R.A., Trivedi, K.S.: Reliability Modeling Using SHARPE. *IEEE Trans. on Reliability* 36(2), 186–193 (1987)
31. The Mathworks. Simulink, <http://www.mathworks.com/products/simulink>
32. Trivedi, K.S., Sahner, R.A.: Sharpe at the age of twenty two. *ACM SIGMETRICS Perf. Eval. Review* 36(4), 52–57 (2009)
33. Vicario, E., Sassoli, L., Carnevali, L.: Using Stochastic State Classes in Quantitative Evaluation of Dense-Time Reactive Systems. *IEEE Trans. on SW Eng.* 35(5), 703–719 (2009)
34. Vittorini, V., Iacono, M., Mazzocca, N., Franceschinis, G.: The OsMoSys approach to multi-formalism modeling of systems. *Software and Systems Modeling* 3, 68–81 (2004)
35. Zimmermann, A.: Dependability evaluation of complex systems with timenet. In: *Proc. Int. Workshop on Dynamic Aspects in Dependability Models for Fault-Tolerant Systems, DYADEM-FTS 2010* (2010)
36. Zimmermann, A., Freiheit, J., German, R., Hommel, G.: Petri Net Modelling and Performability Evaluation with TimeNET 3.0. In: Haverkort, B.R., Bohnenkamp, H.C., Smith, C.U. (eds.) *TOOLS 2000*. LNCS, vol. 1786, pp. 188–202. Springer, Heidelberg (2000)

# Model-Based Multi-objective Safety Optimization

Matthias Guedemann and Frank Ortmeier

Computer Systems in Engineering  
Otto-von-Guericke University of Magdeburg  
{matthias.guedemann, frank.ortmeier}@ovgu.de

**Abstract.** It is well-known that in many safety critical applications safety goals are antagonistic to other design goals or even antagonistic to each other. This is a big challenge for the system designers who have to find the best compromises between different goals.

In this paper, we show how model-based safety analysis can be combined with multi-objective optimization to balance a safety critical system wrt. different goals. In general the presented approach may be combined with almost any type of (quantitative) safety analysis technique. For additional goal functions, both analytic and black-box functions are possible, derivative information about the functions is not necessary. As an example, we use our quantitative model-based safety analysis in combination with analytical functions describing different other design goals. The result of the approach is a set of *best compromises* of possible system variants.

Technically, the approach relies on genetic algorithms for the optimization. To improve efficiency and scalability to complex systems, elaborate estimation models based on artificial neural networks are used which speed up convergence. The whole approach is illustrated and evaluated on a real world case study from the railroad domain.

## 1 Introduction

In virtually all engineering domains two common trends can be identified. Firstly, system complexity is rising steadily and ever more functionality is provided by software controls. The second trend is a steady rise of criticality of system failure. A derailling of a train in the 1950s was much less severe than the same accident with a modern high-speed train which may cause numerous casualties. As a consequence, safety analysis has become more difficult *and* more important.

During the last decade model-based safety analysis methods have become prominent [27, 13, 23, 10] which allow for very precise and reliable safety analysis. The common idea is to *calculate* safety assessments (automatically) from a model of the system. Newest developments also make quantitative, model-based safety analysis possible, which computes hazard probabilities much more accurate than traditional methods [2, 11, 5, 8].

But in real-world applications minimal hazard probabilities are not the only design goal. Most often other antagonistic requirements have to be met as well.

One example is of course the cost of a system, but also functionality and availability must be taken into account. These goals are often negatively affected by focusing solely on safety aspects, raising the question how “best compromises” between antagonistic goals can be found.

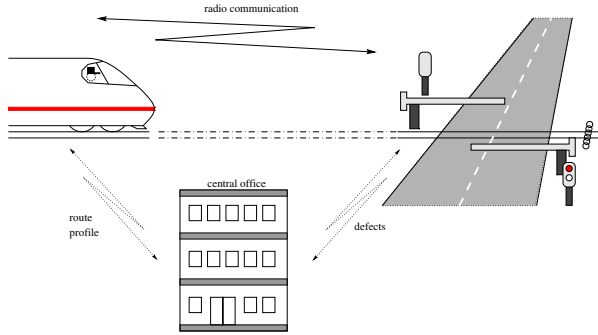
The approach described in this paper is one way to answer this question. It is based on the mathematical theory of Pareto optimization adapted to safety critical systems. In particular, *quantitative* safety analysis using state-of-the-art stochastic model checkers is combined with neural networks for efficiency, scalability and to speed up the necessary computation times. Direct integration of *qualitative* model-based safety analysis techniques further increases performance and quality of the results. In practice, this means that design variants not fulfilling required qualitative safety properties are automatically identified and discarded. This combination leads to an extremely precise characterization and computation of the best design variants of the analyzed system, which are difficult to find with traditional methods.

The rest of the paper is structured as follows: Sect. 2 introduces an illustrative case study which is used throughout the whole paper. In Sect. 3 basics of model-based safety analysis are briefly discussed and the techniques used in the paper are introduced. The main scientific contribution is in Sect. 4 which describes and explains our approach for multi-objective safety optimization and the underlying algorithms. Finally, the approach is applied to the running example. Related work is discussed in Sect. 5, while Sect. 6 summarizes the results and gives an outlook to future work.

## 2 Case Study

The following case study of a radio-based railroad control was the reference case study in the priority research program 1064 “Integrating software specifications techniques for engineering applications” of the German Research Foundation (DFG). It was supplied by the German railway organization, Deutsche Bahn, and addresses a novel technique for controlling railroad crossings. This technique aims at medium speed routes, i.e. routes with maximum speed of  $160 \frac{km}{h}$ . The main difference between this technology and the traditional control of railroad crossings is that signals and sensors on the route are replaced by radio communication and software computations in the train and railroad crossing. This offers cheaper and more flexible solutions, but also shifts safety critical functionality from hardware to software. The system works as follows:

Trains are assumed to continuously monitor their position. When a train approaches a crossing, it broadcasts a *secure*-request to the crossing. When a railroad crossing receives this *secure*-request, it switches on the traffic lights, first the *yellow* light, then the *red* light, and finally closes the barriers. Once they are closed, the railroad crossing is *secured* for a certain period of time. Shortly before the train reaches the *latest braking point* (latest point, where it is possible for the train to stop in front of the crossing), it requests the status of the railroad crossing. If the crossing is *secured*, it responds with a *release* signal which



**Fig. 1.** Radio-based railroad crossing

indicates, that the train may pass the crossing. If the train receives no confirmation of the status of the crossing, it will initiate an emergency brake. Behind the crossing, a sensor detects that the train has passed and an *open* signal is sent to the crossing. A schematic view of the case study is depicted in Fig. 1.

One special requirement (from Deutsche Bahn) is, that once the barrier is closed, it shall reopen automatically after 5 minutes even if no train has passed. This requirement might seem counter-intuitive, but the background is that people waiting at the crossing for a long time without any train in sight, are very likely to get impatient and cross anyway. But if the crossing is closed, they will probably drive very slowly which increases the risk of a collision. This case study was first introduced in [17] and [13] contains a modelling including quantitative aspects..

### 3 Model-Based Safety Analysis

One method to examine the safety of such a system is a model-based safety analysis. It basically consists of three steps: (1) construction of a formal system model, (2) qualitative safety analysis and (3) quantitative safety analysis. There exist numerous variants of for model-based analysis, but they all share this principle. Although some techniques merge some steps into one (e.g. failure injection) or only focus on qualitative *or* quantitative aspects.

#### 3.1 Formal Model Construction

The first step for model-based safety analysis is the construction of a formal model that captures the nominal behavior of the system. Such a model contains for example the movement and acceleration/deceleration of the train, the behavior of the barrier of the crossing and the control system. Such a model already allows to verify the correctness of the system according to its intended behavior.

But in safety analysis, it must also be examined what happens if one or more components do not work as expected. As a consequence, failure mode behavior must be integrated into the system model. Which failure modes are relevant is application specific. For standard components catalogs of failure modes exist. For non-standard system components, there exist structured approaches like HaZop [16] or failure sensitive specification [25] to identify possibly relevant failure modes.

For the safety analysis of the case-study, the following six relevant failure modes were identified: *error\_passed* (which means that the sensor misdetects that the train has already passed the crossing), *error\_odo* (which models a deviation of its measured velocity), *error\_comm* (which models a communication failure), *error\_close* (which models that the crossing wrongly signals that it is closed although it is not), *error\_actuator* (which models that the barrier gets stuck when closing), and *error\_brake* (which models failing brakes of the train).

These failure modes are integrated into the nominal system model to form the *extended system model*. This integration is done in such a way that the functional behavior is still contained as a real subset (in terms of possible traces). More details on sound integration of failure mode behavior may be found in [26]. Finally, probabilistic estimations on environment and failure modes must be integrated. Convenient modeling frameworks for stochastic models are SAML [11] or SLIM [4].

### 3.2 Qualitative Safety Analysis

Using the extended system model, qualitative safety analysis can be applied to compute all combinations of failure modes that can lead to a hazard. In this case this means the computation of all failure mode combinations that can cause the train to pass the crossing although the barrier is not closed. This can for example be computed with deductive cause-consequence analysis (DCCA). The results of DCCA are the combinations of failure modes that can lead to a system hazard which are called the minimal critical sets. In the case study, the following sets of failure modes can cause the hazard, i.e. be the reason that the train enters the crossing but the barrier is not closed [23]:

- $\Gamma_1 := \{error\_passed\}$
- $\Gamma_2 := \{error\_odo\}$
- $\Gamma_3 := \{error\_comm, error\_close\}$
- $\Gamma_4 := \{error\_comm, error\_brake\}$
- $\Gamma_5 := \{error\_close, error\_actuator\}$
- $\Gamma_6 := \{error\_actuator, error\_brake\}$

The advantage of DCCA compared to (formal) fault-tree analysis [31,27] is that DCCA is provably correct and complete. This means that for each computed critical combination of failure modes, there actually exists a system run on which it causes the hazard and there exist no other (inclusion-minimal) combinations of failure modes that can cause the hazard.

### 3.3 Quantitative Safety Analysis

For certification of a safety critical system according to different norms and standards, it is most often mandatory to prove that the occurrence probability of a hazard is below a specified threshold whose amount depends on the application domain. So, in addition to qualitative information about the possible causes of a hazard, its occurrence probability is also very important.

In many approaches such a probability is computed in an a-posteriori way. Most often the probability that failure modes cause the hazard is estimated based on assumptions like stochastic independence of the failure modes. A method for accurate computation of hazard probabilities using directly an extended system model – i.e. without assumptions about stochastic independence etc. – is probabilistic DCCA (pDCCA) [12].

For the quantitative analysis of the case study *error\_comm* was modeled as a per-demand failure mode with an occurrence probability of  $3.5 \cdot 10^{-5}$ . The other failure modes were modeled as per-time failure modes with a failure rate of  $7 \cdot 10^{-5} \frac{1}{s}$  for *error\_odo* and  $7 \cdot 10^{-9} \frac{1}{s}$  for the other per-time failure modes. More details on the combination of failure rates and failure probabilities can be found in [12], a more detailed description of the quantitative formal modeling of the railroad case-study can be found in [13]. The computed hazard probability for the example with pDCCA to an extended system model (described in the SAML framework) is shown in Eq. (1):

$$P(H) := P[true \mathbf{U} H] = 5.6286 \cdot 10^{-7} \quad (1)$$

It computes the probability that the hazard  $H$  occurs (the train passes the crossing but the barrier is open) expressed in the probabilistic temporal logic PCTL [14]. It is computed using state of the art stochastic model-checking tools like PRISM [18] or MRMC [15]. Note, that for actual computation of this number a lot of assumptions (besides failure rates) had to be made. For example, in the calculation of the latest breaking point no extra safety margin had been added and a maximum allowed speed of  $115 \frac{km}{h}$  had been assumed. Both are obviously free parameters of the system and variation of these parameters will affect both safety and availability (i.e. time delay) of the system.

### 3.4 Summary

These three steps basically mark the procedure for a qualitative and quantitative model-based safety analysis. A convenient way to do this is to construct the extended system model in SAML. It allows for the combination of per-demand and per-time failure modes. It is also tool-independent as it allows for the analysis of the system with different state of the art verification tools. The necessary transformations of a SAML model into the input specification of different verification tools are semantically funded which guarantees that the qualitative and quantitative safety analysis are conducted on equivalent models.

Nevertheless, whether a computed hazard occurrence probability is sufficient or not depends on external requirements. In the case study, it will be defined for



example by the EN 50128 standard for safety in the railway domain. If it is too high, the system must be augmented with risk reducing measures to reach the required threshold. But even if the probability is acceptable, often other aspects cannot be disregarded: Is this the best compromise for other antagonistic goals like costs? Is it possible to achieve a lower but acceptable safety threshold while getting an disproportional larger advantage wrt. antagonistic goals?

For example, reducing the allowed speed a lot will most probably increase safety but will also increase time delays. Choosing such free parameters is typically solely based on the experience/intuition of system designers. The approach presented in the following section is an attempt to help them make this decision.

## 4 Model-Based Safety Optimization

One way to answer these questions is to use an additional analytic mathematical model and optimize it to find better system designs [22]. The problem with such an approach is that it does not reflect whether qualitative safety properties – in particular critical failure mode combinations – are still valid in a changed system design. Another problem is the usage of a-posteriori estimation methods which are often based on unrealistic assumptions and are therefore not very accurate.

Because of these problems and challenges, we propose a different approach which is based on mathematical optimization, but uses model-based techniques to compute hazard probabilities and only evaluates system designs for which the desired qualitative safety properties hold. The approach is completely automatic and can cope with multiple antagonistic goals. The result is a set of possible system variants which give the best compromises between the desired goals.

### 4.1 Multi-objective Optimization

Central to multi-objective optimization is the notion of Pareto sets and Pareto optimality. Informally, a system is Pareto optimal if there is no way to change it in such a way that it becomes better wrt. one objective function, without becoming worse wrt. another one. A Pareto set contains those elements which are Pareto optimal. As there are multiple objective functions, it is not possible to define a total ordering, but only a partial ordering. If two elements are in Pareto order, the worse one is *dominated* by the better one. For a more detailed description see for example [19].

In order to be optimizable, variable parameters must exist in a system. This can be actually free parameters that can be instantiated, e.g. maximal speed of the train, or varying failure rates for different qualities of system components. It is also possible to vary complete system components that have the same nominal behavior but have different other properties, e.g. use redundant sensors instead of a single sensor. In general, a system will be described as the fixed part, a set of possible parameter values and a set of possible system component substitutions. A design variant of the system is then an instantiation of the free parameters and a selection of the variable system components.

For the case study, three parameters were identified that can be adjusted to get an optimized system design. The first is the accuracy of the odometer (i.e. using odometer components of different quality and costs). The quality is measured as the failure rate  $\lambda_{odo}$  of the deviation from the real velocity. This clearly has an influence on the safety of the system, as the breaking point calculation is based on the reported value of the odometer. The second parameter is a safety margin  $z$  which is added to the calculated breaking point of the train to compensate for some wrong sensor data of the odometer or variants in braking coefficients. This safety margin basically adds a buffer to the distance at which the train initiates the communication for the closing of the barrier. The third parameter  $v_{allowed}$  is the allowed maximum speed of trains on the track when approaching the crossing. This velocity directly influences the calculation of the activation point. Both safety margin and allowed speed directly influence the safety of the system as the calculation of the activation point of the radio communication is dependent on these parameters. For the example, we used the interval  $[0, 1]$  for failure rates of the odometer<sup>1</sup>, possible safety margins between 0 and 200m and allowed maximum velocities between 1 and 120  $\frac{km}{h}$ . The (antagonistic) objective functions considered in the example are:

$$\begin{aligned}
 f_1(\lambda_{odo}, z, v_{allowed}) &:= P[true \mathbf{U} H] \\
 f_2(\lambda_{odo}, z, v_{allowed}) &:= cost(\lambda_{odo}) \\
 f_3(\lambda_{odo}, z, v_{allowed}) &:= \frac{z + x_{brake}(v_{allowed})}{v_{allowed}} - \frac{z + x_{brake}(v_{allowed})}{v_{max}}
 \end{aligned}$$

The first function  $f_1$  is the occurrence probability of the hazard  $H$ , i.e. that the train enters the crossing while the barrier is not closed. It is computed using stochastic model-checking of the extended system model instantiated with the concrete parameters. The second function  $f_2$  describes the cost of more accurate components (i.e. odometer). It is modeled in such a way that the decrease in failure rate results in an exponential rise in cost (proportional to the negative logarithm of  $\lambda_{odo}$ ). The third objective function  $f_3$  describes the time delay caused by lowering the allowed speed of the train at a single crossing (compared to normal travel time on a track without crossing at a speed  $v_{max} = 160 \frac{km}{h}$ ).

For multi-objective optimization, every design variant of the system can then be represented as an element of  $x \in [0, 1] \times [0, 200] \times [1, 120]$  and may be assessed by computing the values of the objective vector function  $f(x) = (f_1(x), f_2(x), f_3(x))$ . The Pareto set for this problem describes all "best" compromises. Computing the Pareto set for arbitrary functions is a computationally hard problem, even more so for non-analytic functions. In the special situation of model-based quantitative safety analysis, one objective function is the

---

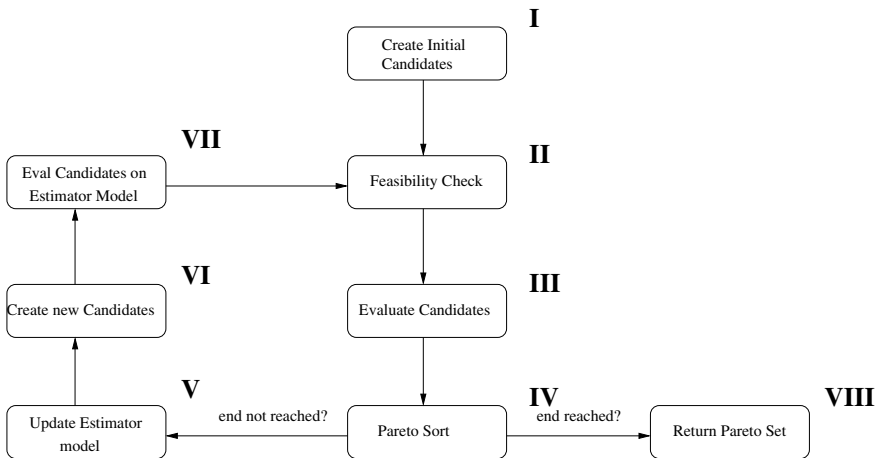
<sup>1</sup> For simplicity, a continuous value is used here. In practice, a fixed number of different odometers and respective failure rates will be more realistic. Such a situation would of course be also possible with this approach (even with lower computational effort).

evaluation of the model with a stochastic model checker. This is treated as a black-box function for which no information about derivatives or mathematical properties is known. Therefore, the minimization problem is solved with an optimization scheme based on genetic algorithms which do not rely on derivative information.

Perhaps the biggest challenge for the safety optimization is that evaluation of  $f_1$  (the quantitative safety assessment with pDCCA) needs much longer than the computation of the analytic functions or even a qualitative safety analysis, the average running time of one single pDCCA for the example is  $11.5min$ . Normally genetic algorithms rely on a large number of function evaluations. This problem is tackled by using adaptive estimators which allow for an a-priori identification of potentially good system variants. The costly operations are then only applied on promising candidates.

## 4.2 Safety Optimization

Our approach for model-based multi-objective safety optimization is based on the non-dominating sorting genetic algorithm (NSGA-II) [30]. It is one of the most widely used multi-objective genetic algorithms. It has a very elaborate strategy to assure diversity in the system variants and has successfully been applied to many different optimization problems. Combination of genetic algorithms with estimation models to increase the convergence has already been done by [6]. A NSGA-II variant using artificial neural network estimators to reduce the number of actual function evaluations is introduced in [20]. This algorithm has been adapted to allow for the model-based multi-objective optimization of safety critical systems. The complete approach for the model-based optimization is shown in Fig. 2.



**Fig. 2.** Schematic View of Model-Based Multi-Objective Safety Optimization

- I At first an **Initial Set of Candidates** for the system design is created. For system components where different variants with the same functional properties are available, one specific implementation is selected. For parameters, an initial value is chosen in a pseudo-random fashion. To ensure that a representative choice is made, values are chosen with Latin-hypercube sampling, either from a uniform distribution over an interval or from a logarithmic partitioning of an interval (this is desirable e.g. for probabilities where very small probabilities would be left out on an only uniform distribution).
- II In the **Feasibility Check**, the candidates are checked for admissibility by verifying qualitative properties, i.e. qualitative safety properties like minimal critical sets (DCCA). Only candidates that meet the qualitative requirements are evaluated with quantitative methods. If no admissible candidates are found in the initial sampling the optimization process is terminated.
- III In the **Evaluation** step, all candidates are evaluated for all objective goal functions and the results are stored. For safety optimization one objective function will always be the hazard occurrence probability (we are using pDCCA for this).
- IV The evaluation results are used in the **Pareto Sort** step to rank all candidates and compute the current Pareto set. The dominating candidates in the current population are identified in this step. NSGA-II uses the “crowding distance” [30] which balances function values and distance of candidates to increase diversity in the population.
- V The results of the evaluation are used to **Update the Estimation Model**. In the current implementation this is realized as an artificial neural network which estimates the values of all objective goal functions for a system variant. It is based on the *fast artificial neural network* library libfann [21].
- VI In the next step, **Creation of New Candidates** is done by combining the most promising existing candidates and mutating in a random fashion. To ensure diversity and prevent early overfitting, the mutation rate is adjusted with the number of generations, and a roulette selection with fixed probabilities is used to select candidates for combination.
- VII The generated new candidates are then **Evaluated on the Estimation Model** and the best ones – according to the estimation model – are chosen for the next iteration. This selects the most promising candidates for concrete quantitative assessment, thus saving the evaluation time from less-promising ones. The selected candidates are then checked for qualitative feasibility (step II) and the algorithm continues.
- VIII The algorithm terminates once either the maximal time is reached or the planned number of generations has been reached and the planned number of candidates has been evaluated.

### 4.3 Optimization of the Case Study

It is relatively obvious, that decreasing the failure rate for the odometer directly increases the cost and that decreasing the allowed speed increases the time delay. These objective functions are analytic and the effect of the different variables can

be studied using calculus. Still, finding the best compromise, i.e. an optimum for both at the same time which is not necessarily an optimum for every objective function considered in isolation, is difficult.

For the non-analytic “black-box” function  $f_1$ , the effect of the parameters is unclear. From the modeling, an increase in the safety margin should decrease the hazard probability. It also seems obvious that the effect of the allowed speed on the hazard probability is also directly proportional and that a lower allowed speed means a decreased risk.

Yet, the “obvious” effect of decreased safety with increased speed does not really hold. Under closer examination one finds that, although for larger values for the allowed speed the increase in hazard probability clearly shows (even rather dramatically for values over  $17 \frac{m}{s}$ ), the minimum is actually reached at a value of  $4 \frac{m}{s}$  and slower speed increase the hazard probability! This effect is not expected and would probably lead to wrong assumptions if not checked on the quantitative model. The reason for this is the requirement that the crossing reopens after a while even if the train had not passed the crossing (see Sect. 2).

The optimization of the case study was computed with two parallel runs on a 8 core Xeon CPU with 2.66 Ghz and 16G RAM<sup>2</sup>. An additional advantage of using the proposed approach for optimization is its trivial parallelization for further speedup. The population size was 25 and 20 generations were created in total. For each new generation, 250 candidates were created and evaluated on the estimation model. So in total 1050 ( $2 \cdot 25$  initial candidates and  $2 \cdot 20 \cdot 25$  evolved) function evaluations were conducted which required 4 days and 5 hours. PRISM [18] was used as verification tool, the model representations were sparse matrices.

The results of the parallel runs were then combined into a single result file to further increase diversity. The Pareto set was then computed on the combined results. Note that in contrast to the published original NSGA-II, *all* evaluated pDCCA results are stored and used for the final Pareto set computation, not only the candidates in the last generation of the genetic algorithm. The total number of elements in the Pareto set is 258. As there are 3 objective functions, it is difficult to visualize the complete Pareto set. For illustration purposes, a two-dimensional projection of the Pareto set for the functions  $f_1$  and  $f_2$  is shown in Fig. 3.

The figure clearly shows, that the initial “guess” (the point marked with the arrow) is not even in the Pareto set. This means there are system variants which are better than the original system design with respect to these two objective functions. This holds true for all other combinations of two objective functions, which are omitted here because of space restrictions. Analyzing the complete three-dimensional Pareto set, it turned out that exactly one system variant  $x^*$  was found which is better for all three aspects than the initial system variant  $x^{initial}$  as shown in the following:

$$f(x^*) = f \left( \begin{pmatrix} 9.04 \cdot 10^{-5} \frac{1}{s} \\ 4.97m \\ 67.08 \frac{km}{h} \end{pmatrix} \right) = \begin{pmatrix} 4.90 \cdot 10^{-7} \\ 7.70 \\ 2.76s \end{pmatrix} < \begin{pmatrix} 5.63 \cdot 10^{-7} \\ 7.96 \\ 2.80s \end{pmatrix} = f(x^{initial})$$

<sup>2</sup> For the experimental software please contact the authors.

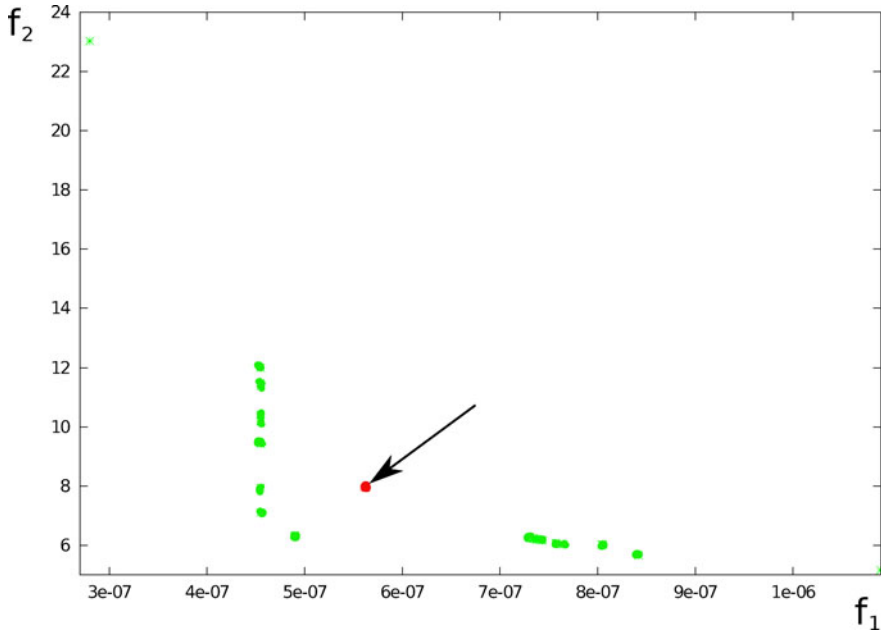


Fig. 3. Projection of the Pareto set onto  $f_1$  and  $f_2$

#### 4.4 Selection of System Design

In this example there was a single element in the Pareto set which was better in all aspects than the original reference system. In general, there will be several dominating candidates. Therefore one of them has to be selected. This is a well-known problem in multi-criteria optimization. Different criteria are possible to decide which to choose for general systems [7]. For safety critical systems, hazard probability must typically be kept below a given threshold. This means that all systems variants that have a hazard probability above a given threshold must be discarded. A possible decision strategy might then be to choose the most cost effective system variant which adheres to the hazard probability threshold required by a standard. More elaborate approaches are of course possible, but are not in the scope of this paper.

## 5 Related Work

A first approach to use hazard probabilities in an optimization approach has already been described in [24]. It uses analytical mathematical models in an a-posteriori optimization. This approach is computationally much more efficient. However, it can not cope with stochastic dependencies and relies on separate models for quantitative analysis. In addition, quantitative approximations are much coarser. This approach might be useful alternative to the current estimator model.

In [9], a framework for early quality prediction of component-based systems is proposed. It aims in particular at systems where reliability, safety, availability and security are important. But also allows consideration of performance of the system. Although it mainly considers software systems, the framework could be extended to also include software-intensive safety critical systems and additional non-functional requirements like costs as described in this paper. This would allow the described model-based multi-objective safety optimization to be integrated into the larger framework proposed in [9].

In [28], the Hip-Hops [29] methodology is used as the basis for optimization of safety critical systems. Hip-Hops is a structural approach to safety analysis, where components with known properties can be combined into a larger system model and a failure propagation model is used to describe how the system reacts in case of failure occurrence. Such a structural description is then used to evolve better system designs with similar properties wrt. functionality, but better failure tolerance and/or lower costs. For this optimization a variant of NSGA-II is used.

The advantage of our model-based approach is the accompanying increase in accuracy. On the other hand, as the whole system must be analyzed, it may suffer from the state-explosion problem, which is less severe for an approach as [28]. A combination of the two approaches would be interesting, e.g. using the Hip-Hop approach for the structural description of the system, but analyzing the single components with the more accurate model-based safety analysis. The optimization could then be applied on either structural level – exchanging equivalent components – or the parameter level of the components. This could be an interesting compromise between accuracy and scalability.

## 6 Conclusion

The paper presented a method to optimize safety-critical systems with respect to multiple goal functions. It will automatically find (all) best compromises between feasible safety, affordable cost and functional properties. The approach itself is generic. It may be combined with any model-based automatic safety analysis method. In this paper DCCA was used for feasibility checking and pDCCA for quantitative assessment of the system variants.

The approach guarantees that any specified set of qualitative (safety) properties holds for each proposed optimized system variant. Combination with very precise model-based qualitative and quantitative safety analysis methods make quantitative estimations very accurate and avoid coarse over estimations. Although computational costs are high, a smart estimator model allowed to apply the approach to a medium-sized case study with a single work station. Possible parallelization is straight forward, so that it can easily be distributed on a cluster of normal PCs and run over night. Further increases in performance seem possible through combination with other methods for estimating system safety.

Besides these performance improvement, an interesting area for further research is comparing different multi-objective optimization algorithms in the context of safety optimization. Another very interesting question is the effect of

different estimation methods on the quality of the optimization. The method of estimation is crucial for both convergence speed and quality of the results. Our experiments with the optimization of the case study described in this paper showed that both the absolute and relative estimation error of the neural network for the estimation of the hazard probability decreased considerably with the number of available real evaluation data. An interesting question for further research is how much of the effect comes from generalization and how much from concentrating the search in similar directions due to the estimation model. Furthermore different estimation methods and also different (adaptive?) strategies for using the estimations methods will be compared in further research.

**Acknowledgment.** Matthias GÜdemann is funded by the German Ministry of Education and Science (BMBF) within the ViERforES project (no. 01IM08003C).

## References

1. Abdulla, P.A., Deneux, J., Stålmarch, G., Ågren, H., Åkerlund, O.: Designing safe, reliable systems using Scade. In: Margaria, T., Steffen, B. (eds.) *ISoLA 2004*. LNCS, vol. 4313, pp. 115–129. Springer, Heidelberg (2004)
2. Böde, E., Peikenkamp, T., Rakow, J., Wischmeyer, S.: Model based importance analysis for minimal cut sets. In: Cha, S(S.), Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) *ATVA 2008*. LNCS, vol. 5311, pp. 303–317. Springer, Heidelberg (2008)
3. Bozzano, M., Villaforita, A.: Improving system reliability via model checking: the FSAP/NuSMV-SA safety analysis platform. In: Anderson, S., Felici, M., Littlewood, B. (eds.) *SAFECOMP 2003*. LNCS, vol. 2788, pp. 49–62. Springer, Heidelberg (2003)
4. Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V.Y., Noll, T., Roveri, M.: Model-based codesign of critical embedded systems. In: *Proceedings of ACES-MB*, vol. 507, pp. 87–91. CEUR Workshop Proceedings (2009)
5. Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V.Y., Noll, T., Roveri, M.: Safety, dependability, and performance analysis of extended AADL models. *The Computer Journal* (2010)
6. Branke, J., Schmidt, C.: Faster convergence by means of fitness estimation. *Soft Computing - A Fusion of Foundations, Methodologies and Applications* 9, 13–20 (2005)
7. Branke, J., Deb, K., Dierolf, H., Osswald, M.: Finding knees in multi-objective optimization. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiño, P., Kabán, A., Schwefel, H.-P. (eds.) *PPSN 2004*. LNCS, vol. 3242, pp. 722–731. Springer, Heidelberg (2004)
8. Grunske, L., Colvin, R., Winter, K.: Probabilistic model-checking support for FMEA. In: *Proceedings of the QEST*. IEEE, Los Alamitos (2007)
9. Grunske, L.: Early quality prediction of component-based systems - a generic framework. *Journal of Systems and Software* 80(5), 678–686 (2007); *Component-Based Software Engineering of Trustworthy Embedded Systems*
10. GÜdemann, M., Ortmeier, F., Reif, W.: Computing ordered minimal critical sets. In: *Proceedings of FORMS / FORMAT* (2008)



11. Güdemann, M., Ortmeier, F.: A framework for qualitative and quantitative model-based safety analysis. In: Proceedings of HASE 2010 (2010)
12. Güdemann, M., Ortmeier, F.: Probabilistic model-based safety analysis. In: Proceedings of QAPL. EPTCS (2010)
13. Güdemann, M., Ortmeier, F.: Quantitative model-based safety analysis: A case study. In: Proceedings of SICHERHEIT. LNI (2010)
14. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6, 102–111 (1994)
15. Katoen, J.-P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation, Corrected Proof*. 167–176 (2010) (in press)
16. Kletz, T.A.: Hazop and HAZAN notes on the identification and assessment of hazards. Technical report, Inst. of Chemical Engineers, Rugby, England (1986)
17. Klose, J., Thums, A.: The STATEMATE reference model of the reference case study ‘Verkehrslleittechnik’. Technical Report 2002-01, Universität Augsburg (2002)
18. Kwiatkowska, M., Norman, G., Parker, D.: Prism: Probabilistic symbolic model checker, pp. 200–204. Springer, Heidelberg (2002)
19. Miettinen, K.: Some methods for nonlinear multi-objective optimization. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) EMO 2001. LNCS, vol. 1993, pp. 1–20. Springer, Heidelberg (2001)
20. Nain, P.K.S., Deb, K.: Computationally effective search and optimization procedure using coarse to fine approximations. In: Proceedings of CEC (2003)
21. Nissen, S.: Implementation of a fast artificial neural network library (fann). Technical report, Department of Computer Science University of Copenhagen, DIKU (2003), <http://fann.sf.net>
22. Ortmeier, F., Reif, W.: Safety optimization: A combination of fault tree analysis and optimization techniques. In: Proceedings of DSN, Florence. IEEE Computer Society, Los Alamitos (2004)
23. Ortmeier, F., Reif, W., Schellhorn, G.: Formal safety analysis of a radio-based railroad crossing using deductive cause-consequence analysis (DCCA). In: Dal Cin, M., Kaàniche, M., Pataricza, A. (eds.) EDCC 2005. LNCS, vol. 3463, pp. 210–224. Springer, Heidelberg (2005)
24. Ortmeier, F., Schellhorn, G., Reif, W.: Safety optimization of a radio-based railroad crossing. In: Proceedings of FORMS / FORMAT (2004)
25. Ortmeier, F.: Formale Sicherheitsanalyse. Logos Verlag, Berlin (2006)
26. Ortmeier, F., Güdemann, M., Reif, W.: Formal failure models. In: Proceedings of DCDS. Elsevier, Amsterdam (2007)
27. Ortmeier, F., Schellhorn, G.: Formal Fault Tree Analysis - Practical Experiences. In: Proceedings of AVoCS (2006)
28. Papadopoulos, Y., Walker, M., Parker, D., Rùde, E., Hamann, R., Uhlig, A., Grätz, U., Lie, R.: Engineering failure analysis and design optimisation with hip-hops. *Engineering Failure Analysis* (2010)
29. Pasquini, A., Papadopoulos, Y., McDermid, J.: Hierarchically performed hazard origin and propagation studies. In: Felici, M., Kanoun, K., Pasquini, A. (eds.) SAFECOMP 1999. LNCS, vol. 1698, pp. 139–152. Springer, Heidelberg (1999)
30. Deb, K., Pratap, A., Agarwal, S., Meyarivan T.: A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transaction on Evolutionary Computation*, 181–197 (2002)
31. Vesley, W., Dugan, J., Fragole, J., Minarik II, J., Railsback, J.: *Fault Tree Handbook with Aerospace Applications*. NASA Office of Safety and Mission Assurance (August 2002)

# Tradeoff Exploration between Reliability, Power Consumption, and Execution Time

Ismail Assayad<sup>1</sup>, Alain Girault<sup>2</sup>, and Hamoudi Kalla<sup>3</sup>

<sup>1</sup> ENSEM (RTSE team), University Hassan II of Casablanca, Morocco

<sup>2</sup> INRIA and Grenoble University (POP ART team and LIG lab), France

<sup>3</sup> University of Batna (SECOS team), Algeria

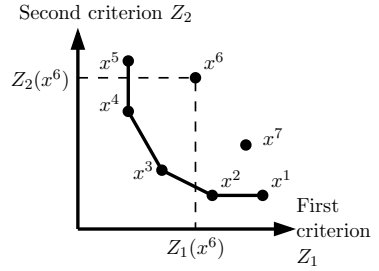
**Abstract.** We propose an off-line scheduling heuristics which, from a given software application graph and a given multiprocessor architecture (homogeneous and fully connected), produces a static multiprocessor schedule that optimizes three criteria: its *length* (crucial for real-time systems), its *reliability* (crucial for dependable systems), and its *power consumption* (crucial for autonomous systems). Our tricriteria scheduling heuristics, TSH, uses the *active replication* of the operations and the data-dependencies to increase the reliability, and uses *dynamic voltage and frequency scaling* to lower the power consumption.

## 1 Introduction

For autonomous critical real-time embedded systems (e.g., satellite), guaranteeing a very high level of reliability is as important as keeping the power consumption as low as possible. We present an off-line scheduling heuristics that, from a given software application graph and a given multiprocessor architecture, produces a static multiprocessor schedule that optimizes three criteria: its *length* (crucial for real-time systems), its *reliability* (crucial for dependable systems), and its *power consumption* (crucial for autonomous systems). We target homogeneous distributed architecture, such as multicore processors. Our tricriteria scheduling heuristics uses the *active replication* of the operations and the data-dependencies to increase the reliability, and uses *dynamic voltage and frequency scaling (DVFS)* to lower the power consumption. However, DVFS has an impact of the failure rate of processors, because lower voltage leads to smaller critical energy, hence the system becomes sensitive to lower energy particles. As a result, the failure probability increases. The two criteria *length* and *reliability* are thus *antagonistic* with each other and with the schedule length, which makes this problem all the more difficult.

Let us address the issues raised by multicriteria optimization. Figure 1 illustrates the particular case of two criteria to be minimized. Each point  $x^1$  to  $x^7$  represents a solution, that is, a different tradeoff between the  $Z_1$  and  $Z_2$  criteria: the points  $x^1$ ,  $x^2$ ,  $x^3$ ,  $x^4$ , and  $x^5$  are *Pareto optima* [17]; the points  $x^2$ ,  $x^3$ , and  $x^4$  are *strong optima* while the points  $x^1$  and  $x^5$  are *weak optima*. The set of all Pareto optima is called the *Pareto front*.

It is fundamental to understand that no single solution among the points  $x^2$ ,  $x^3$ , and  $x^4$  (the strong Pareto optima) can be said, a priori, to be the best one. Indeed, those three solutions are *non-comparable*, so choosing among them can only be done by the user, depending on the precise requirements of his/her application. This is why we advocate producing, for a given problem instance, the Pareto front rather than a single solution. Since we have three criteria, it will be a *surface* in the 3D space (length, reliability, power).

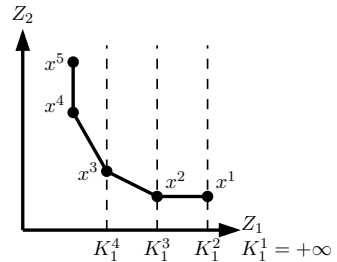


**Fig. 1.** Pareto front for a bicriteria minimization problem

The main contribution of this paper is TSH, the *first* tricriteria scheduling heuristics able to produce a Pareto front in the space (length, reliability, power), and taking into account the impact of voltage on the failure probability. Thanks to the use of active replication, TSH is able to provide any required level of reliability. TSH is an extension of our previous bicriteria (length, reliability) heuristics called BSH [6]. The tricriteria extension presented in this paper is necessary because of the crucial impact of the voltage on the failure probability.

## 2 Principle of the Method and Overview

To produce a Pareto front, the usual method involves transforming all the criteria except one into *constraints*, and then minimizing the last remaining criterion iteratively [17]. Figure 2 illustrates the particular case of two criteria  $Z_1$  and  $Z_2$ . To obtain the Pareto front,  $Z_1$  is transformed into a constraint, with its first value set to  $K_1^1 = +\infty$ . The first run involves minimizing  $Z_2$  under the constraint  $Z_1 < +\infty$ , which produces the Pareto point  $x^1$ . For the second run, the constraint is set to the value of  $x_1$ , that is  $K_1^2 = Z_1(x_1)$ : we therefore minimize  $Z_2$  under the constraint  $Z_1 < K_1^2$ , which produces the Pareto point  $x^2$ , and so on. Another way is to slice the interval  $[0, +\infty)$  into a finite number of contiguous sub-intervals of the form  $[K_1^i, K_1^{i+1}]$ .



**Fig. 2.** Transformation method to produce the Pareto front

The application algorithm graphs we are dealing with are large (tens to hundreds of operations, each operation being a software block), thereby making infeasible exact scheduling methods, or even approximated methods with backtracking, such as branch-and-bound. We therefore have to use *list scheduling heuristics*, which have demonstrated their good performances in the past [10]. We propose in this paper a suitable list scheduling heuristics, adapted from [6].

Using list scheduling to minimize a criterion  $Z_2$  under the constraint that another criterion  $Z_1$  remains below some threshold  $K_1$  (as in Figure 2), requires

that  $Z_1$  be an *invariant measure*, not a varying one. For instance, the energy is a strictly increasing function of the schedule: if  $S'$  is a prefix schedule of  $S$ , then the energy consumed by  $S$  is strictly greater than the energy consumed by  $S'$ . Hence, the energy *is not* an invariant measure. As a consequence, if we attempt to use the energy as a constraint (i.e.,  $Z_1=E$ ) and the schedule length as a criteria to be minimized (i.e.,  $Z_2=L$ ), then we are bound to fail. Indeed, the fact that all the scheduling decisions made at the stage of any intermediary schedule  $S'$  meet the constraint  $E(S') < K_1$  cannot guarantee that the final schedule  $S$  will meet the constraint  $E(S) < K_1$ . In contrast, the power consumption *is* an invariant measure (being the energy divided by the time), and this is why we take the power consumption as a criterion instead of the energy consumption (see Section 3.5).

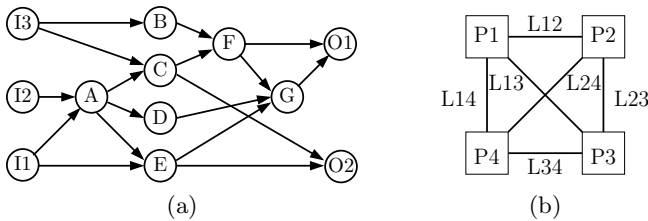
The reliability too is *not* an invariant measure: it is neither an increasing nor a decreasing function of the schedule. So the same reasoning applies if the reliability is taken as a constraint. This is why we take instead, as a criterion, the *global system failure rate per time unit* (GSFR), first defined in [6]. By construction, the GSFR is an invariant measure of the schedule's reliability (see Section 3.7).

For these reasons, each run of our tricriteria scheduling heuristics TSH minimizes the schedule length under the double constraint that the power consumption and the GSFR remain below some thresholds, respectively  $P_{obj}$  and  $\Lambda_{obj}$ . By running TSH with decreasing values of  $P_{obj}$  and  $\Lambda_{obj}$ , starting with  $+\infty$  and  $+\infty$ , we are able to produce the Pareto front in the 3D space (length,GSFR,power). This Pareto front shows the existing tradeoffs between the three criteria, allowing the user to choose the solution that best meets his/her application needs. Finally, our method for producing a Pareto front could work with any other scheduling heuristics minimizing the schedule length under the constraints of both the reliability and the power.

### 3 Models

#### 3.1 Application Algorithm Graph

Most embedded real-time systems are reactive, and therefore consist of some algorithm executed periodically, triggered by a periodic execution clock. Our



**Fig. 3.** (a) An example of algorithm graph  $\mathcal{Alg}$ :  $I_1, I_2,$  and  $I_3$  are input operations,  $O_1$  and  $O_2$  are output operations,  $A-G$  are regular operations; (b) An example of an architecture graph  $\mathcal{Arc}$  with four processors,  $P_1$  to  $P_4$ , and six communication links

model is therefore that of an application algorithm graph  $\mathcal{Alg}$  which is repeated infinitely.  $\mathcal{Alg}$  is an *acyclic oriented graph*  $(\mathcal{X}, \mathcal{D})$  (See Figure 3(a)). Its nodes (the set  $\mathcal{X}$ ) are software blocks called *operations*. Each arc of  $\mathcal{Alg}$  (the set  $\mathcal{D}$ ) is a *data-dependency* between two operations. If  $X \triangleright Y$  is a data-dependency, then  $X$  is a *predecessor* of  $Y$ , while  $Y$  is a *successor* of  $X$ . Operations with no predecessor (resp. successor) are called *input* operations (resp. *output*). Operations do not have any side effect, except for input/output operations: an input operation (resp. output) is a call to a sensor driver (resp. actuator).

The  $\mathcal{Alg}$  graph is acyclic but it is infinitely repeated in order to take into account the reactivity of the modeled system, that is, its reaction to external stimuli produced by its environment.

### 3.2 Architecture Model

We assume that the architecture is an *homogeneous and fully connected multi-processor* one. It is represented by an architecture graph  $\mathcal{Arc}$ , which is a *non-oriented bipartite graph*  $(\mathcal{P}, \mathcal{L}, \mathcal{A})$  whose set of nodes is  $\mathcal{P} \cup \mathcal{L}$  and whose set of edges is  $\mathcal{A}$  (see Figure 3(b)).  $\mathcal{P}$  is the set of processors and  $\mathcal{L}$  is the set of communication links. A *processor* is composed of a computing unit, to execute operations, and one or more communication units, to send or receive data to/from communication links. A *point-to-point communication link* is composed of a sequential memory that allows it to transmit data from one processor to another. Each edge of  $\mathcal{Arc}$  (the set  $\mathcal{A}$ ) always connects one processor and one communication link. Here we assume that the  $\mathcal{Arc}$  graph is *complete*.

### 3.3 Execution Characteristics

Along with the algorithm graph  $\mathcal{Alg}$  and the architecture graph  $\mathcal{Arc}$ , we are also given a function  $\mathit{Exe} : (\mathcal{X} \times \mathcal{P}) \cup (\mathcal{D} \times \mathcal{L}) \mapsto \mathbb{R}^+$  giving the *worst-case execution time* (WCET) of each operation onto each processor and the *worst-case communication time* (WCCT) of each data-dependency onto each communication link. An intra-processor communication takes no time to execute. Since the architecture is homogeneous, the WCET of a given operation is identical on all processors (similarly for the WCCT of a given data-dependency).

The WCET analysis is the topic of much work [18]. Knowing the execution characteristics is not a critical assumption since WCET analysis has been applied with success to real-life processors actually used in embedded systems, with branch prediction, caches, and pipelines. In particular, it has been applied to one of the most critical embedded system that exists, the AIRBUS A380 avionics software [16].

### 3.4 Static Schedules

The graphs  $\mathcal{Alg}$  and  $\mathcal{Arc}$  are the *specification* of the system. Its *implementation* involves finding a multiprocessor schedule of  $\mathcal{Alg}$  onto  $\mathcal{Arc}$ . It consists of two functions: the *spatial allocation function*  $\Omega$  gives, for each operation of  $\mathcal{Alg}$  (resp. for each data-dependency), the subset of processors of  $\mathcal{Arc}$  (resp. the subset of

communication links) that will execute it; and the *temporal allocation function*  $\Theta$  gives the starting date of each operation (resp. each data-dependency) on its processor (resp. its communication link):  $\Omega : \mathcal{X} \mapsto 2^{\mathcal{P}}$  and  $\Theta : \mathcal{X} \times \mathcal{P} \mapsto \mathbb{R}^+$ .

In this work we only deal with *static* schedules, for which the function  $\Theta$  is static, and our schedules are computed *off-line*; i.e., the start time of each operation (resp. each data-dependency) on its processor (resp. its communication link) is statically known. A static schedule is *without replication* if for each operation  $X$  (and for each data-dependency),  $|\Omega(X)|=1$ . In contrast, a schedule is *with (active) replication* if for some operation  $X$  (or some data-dependency),  $|\Omega(X)| \geq 2$ . The number  $|\Omega(X)|$  is called the *replication factor* of  $X$ . A schedule is *partial* if not all the operations of  $\mathcal{Alg}$  have been scheduled, but all the operations that are scheduled are such that all their predecessors are also scheduled. Finally, the *length* of a schedule is the max of the termination times of the last operation scheduled on each of the processors of  $\mathcal{Arc}$ . For a schedule  $S$ , we note it  $L(S)$ .

### 3.5 Voltage, Frequency, and Power Consumption

The maximum supply voltage is noted  $V_{max}$  and the corresponding highest operating frequency is noted  $f_{max}$ . For each operation, its WCET assumes that the processor operates at  $f_{max}$  and  $V_{max}$  (and similarly for the WCCT of the data-dependencies). Because the circuit delay is almost linearly related to  $1/V$  [3], there is a linear relationship between the supply voltage  $V$  and the operating frequency  $f$ . From now on, we will assume that the operating frequencies are *normalized*, that is,  $f_{max}=1$  and any other frequency  $f$  is in the interval  $(0, 1)$ . Accordingly, the execution time of the operation or data-dependency  $X$  placed onto the hardware component  $C$ , be it a processor or a communication link, which is running at frequency  $f$  (taken as a scaling factor) is:

$$\mathcal{E}xe(X, C, f) = \mathcal{E}xe(X, C)/f \quad (1)$$

The power consumption  $P$  of a single operation placed on a single processor is computed according to the classical model of Zhu et al. [19]:

$$P = P_s + h(P_{ind} + P_d) \quad P_d = C_{ef}V^2f \quad (2)$$

where  $P_s$  is the static power (power to maintain basic circuits and to keep the clock running),  $h$  is equal to 1 when the circuit is active and 0 when it is inactive,  $P_{ind}$  is the frequency independent active power (the power portion that is independent of the voltage and the frequency; it becomes 0 when the system is put to sleep, but the cost of doing so is very expensive [5]),  $P_d$  is the frequency dependent active power (the processor dynamic power and any power that depends on the voltage or the frequency),  $C_{ef}$  is the switch capacitance,  $V$  is the supply voltage, and  $f$  is the operating frequency.  $C_{ef}$  is assumed to be constant for all operations, which is a simplifying assumption, since one would normally need to take into account the actual switching activity of each operation to compute accurately the consumed energy. However, such an accurate computation is infeasible for the application sizes we consider here.

For a multiprocessor schedule  $S$ , we cannot apply directly Eq (2). Instead, we must compute the total energy  $E(S)$  consumed by  $S$ , and then divide by the schedule length  $L(S)$ :

$$P(S) = E(S)/L(S) \quad (3)$$

We compute  $E(S)$  by summing the contribution of each processor, depending on the voltage and frequency of each operation placed onto it. On the processor  $p_i$ , the energy consumed by each operation is the product of the active power  $P_{ind}^i + P_d^i$  by its execution time. As a conclusion, the total consumed energy is:

$$E(S) = \sum_{i=1}^{|\mathcal{P}|} \left( \sum_{o_j \in p_i} (P_{ind}^i + P_d^i) \cdot \mathcal{E}xe(o_j, p_i) \right) \quad (4)$$

### 3.6 Failure Hypothesis

Both processors and communication links can fail, and they are *fail-silent* (a behavior which can be achieved at a reasonable cost [11]). Classically, we adopt the failure model of Shatz and Wang [15]: failures are *transient* and the maximal duration of a failure is such that it affects only the current operation executing onto the faulty processor; this is the “hot” failure model. The occurrence of failures on a processor (same for a communication link) follows a Poisson law with a constant parameter  $\lambda$ , called its *failure rate per time unit*. Modern fail-silent processors can have a failure rate around  $10^{-6}/hr$  [1].

Failures are *transient*. Those are the most common failures in modern embedded systems, all the more when processor voltage is lowered to reduce the energy consumption, because even very low energy particles are likely to create a critical charge leading to a transient failure [19]. Besides, failure occurrences are assumed to be *statistically independent events*. For hardware faults, this hypothesis is reasonable, but this would not be the case for software faults [9].

The *reliability* of a system is defined as the probability that it operates correctly during a given time interval. According to our model, the reliability of the processor  $P$  (resp. the communication link  $L$ ) during the duration  $d$  is  $R=e^{-\lambda d}$ . Hence, the reliability of the operation or data-dependency  $X$  placed onto the hardware component  $C$  (be it a processor or a communication link) is:

$$R(X, C) = e^{-\lambda_C \mathcal{E}xe(X, C)} \quad (5)$$

From now on, the function  $R$  will either be used with two variables as in Eq (5), or with only one variable to denote the reliability of a schedule (or a part of a schedule).

Since the architecture is homogeneous, the failure rate per time unit is identical for each processor (noted  $\lambda_p$ ) and similarly for each communication link (noted  $\lambda_\ell$ ).

### 3.7 Global System Failure Rate (GSFR)

As we have demonstrated in Section 2, we must use the *global system failure rate (GSFR)* instead of the system’s reliability as a criterion. The GSFR is the

failure rate per time unit of the obtained multiprocessor schedule, seen as if it were a *single* operation scheduled onto a *single* processor [6]. The GSFR of a static schedule  $S$ , noted  $\Lambda(S)$ , is computed by Eq (6):

$$\Lambda(S) = \frac{-\log R(S)}{U(S)} \text{ with } R(S) = \prod_{(o_i, p_j) \in S} R(o_i, p_j) \text{ and } U(S) = \sum_{(o_i, p_j) \in S} \mathcal{E}xe(o_i, p_j) \quad (6)$$

Eq (6) uses the reliability  $R(S)$ , which, in the case of a static schedule  $S$  without replication, is simply the product of the reliability of each operation of  $S$  (by definition of the reliability, Section 3.6). Eq (6) also uses the total processor utilization  $U(S)$  instead of the schedule length  $L(S)$ , so that the GSFR can be computed *compositionally*. According to Eq (6), the GSFR is *invariant*: for any schedules  $S_1$  and  $S_2$  such that  $S = S_1 \circ S_2$ , where “ $\circ$ ” is the concatenation of schedules, if  $\Lambda(S_1) \leq K$  and  $\Lambda(S_2) \leq K$ , then  $\Lambda(S) \leq K$  (Proposition 1 in [6]).

Finally, it is very easy to translate a reliability objective  $R_{obj}$  into a GSFR objective  $\Lambda_{obj}$ : one just needs to apply the formula  $\Lambda_{obj} = -\log R_{obj}/D$ , where  $D$  is the mission duration. This shows that the GSFR criterion is usable in practice.

## 4 The Tricriteria Scheduling Algorithm TSH

### 4.1 Decreasing the Power Consumption

Two operation parameters of a chip can be modified to lower the power consumption: the frequency and the voltage. We assume that each processor can be operated with a *finite set of supply voltages*, noted  $\mathcal{V}$ . We thus have  $\mathcal{V} = \{V_0, V_1, \dots, V_{max}\}$ . To each supply voltage  $V$  corresponds an operating frequency  $f$ . We choose not to modify the operating frequency and the supply voltage of the communication links.

We assume that the cache size is adapted to the application, therefore ensuring that the execution time of an application is linearly related to the frequency [12] (i.e., the execution time is doubled when frequency is halved).

To lower the energy consumption of a chip, we use *Dynamic Voltage and Frequency Scaling (DVFS)*, which lowers the voltage and increases proportionally the cycle period. However, DVFS has an impact of the failure rate [19]. Indeed, lower voltage leads to smaller critical energy, hence the system becomes sensitive to lower energy particles. As a result, the fault probability increases both due to the longer execution time and to the lower energy: the voltage-dependent failure rate  $\lambda(f)$  is:

$$\lambda(f) = \lambda_0 \cdot 10^{\frac{b(1-f)}{1-f_{min}}} \quad (7)$$

where  $\lambda_0$  is the nominal failure rate per time unit,  $b > 0$  is a constant,  $f$  it the frequency scaling factor, and  $f_{min}$  is the lowest operating frequency. At  $f_{min}$  and  $V_{min}$ , the failure rate is maximal:  $\lambda_{max} = \lambda(f_{min}) = \lambda_0 \cdot 10^b$ .

We apply DVFS to the processors and we assume that the voltage switch time can be neglected compared to the WCET of the operations. To take into account the voltage in the schedule, we modify the spatial allocation function  $\Omega$  to give



the supply voltage of the processor for each operation:  $\Omega : \mathcal{X} \mapsto \mathcal{Q}$ , where  $\mathcal{Q}$  is the domain of the sets of pairs  $\langle p, v \rangle \in \mathcal{P} \times \mathcal{V}$ .

Figure 4 shows a simple schedule  $S$  where operations  $X$  and  $Z$  are placed onto  $P_1$ , operation  $Y$  onto processor  $P_2$ , and the data-dependency  $X \triangleright Y$  is placed onto the link  $L_{12}$ . Since we do not apply DVFS to the communication links, we only compute the energy consumed by the processors (see Eq (4)):

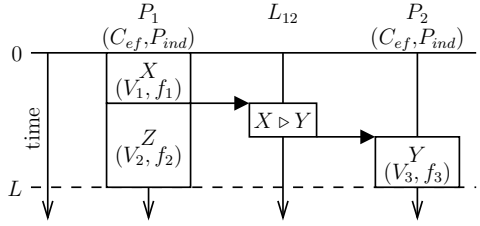


Fig. 4. A simple schedule of length  $L$

- On  $P_1$ :  $E(P_1) = P_{ind}L + C_{ef}V_1^2f_1 \mathcal{E}xe(X, P_1, f_1) + C_{ef}V_2^2f_2 \mathcal{E}xe(Z, P_1, f_2)$ .
- On  $P_2$ :  $E(P_2) = P_{ind}L + C_{ef}V_3^2f_3 \mathcal{E}xe(Y, P_1, f_3)$ .

By applying Eqs (1) and (3), we thus obtain:

$$P(S) = \frac{E(P_1) + E(P_2)}{L} = 2P_{ind} + \frac{C_{ef}}{L} \cdot (V_1^2 \mathcal{E}xe(X, P_1) + V_2^2 \mathcal{E}xe(Z, P_1) + V_3^2 \mathcal{E}xe(Y, P_2))$$

The general formula for a schedule  $S$  is therefore:

$$P(S) = |\mathcal{P}| \cdot P_{ind} + \frac{C_{ef}}{L(S)} \sum_{i=1}^{|\mathcal{P}|} \left( \sum_{o_j \in p_i} V(o_j)^2 \cdot \mathcal{E}xe(o_j, p_i) \right) \quad (8)$$

## 4.2 Decreasing the GSFR

According to Eq (6), decreasing the GSFR is equivalent to increasing the reliability. Several techniques can be used to increase the reliability of a system. Their common point is to include some form of redundancy (this is because the target architecture  $\mathcal{A}rc$ , with the failure rates of its components, is fixed). We have chosen the *active replication* of the operations and the data-dependencies, which consists in executing several copies of a same operation onto as many distinct processors (resp. data-dependencies onto communication links).

To compute the GSFR of a static schedule with replication, we use *Reliability Block-Diagrams (RBD)* [11]. An RBD is an *acyclic oriented graph*  $(N, E)$ , where each node of  $N$  is a *block* representing an element of the system, and each arc of  $E$  is a *causality link* between two blocks. Two particular connection points are its *source*  $S$  and its *destination*  $D$ . An RBD is *operational* if and only if there exists at least one operational path from  $S$  to  $D$ . A path is operational if and only if all the blocks in this path are operational. The probability that a block be operational is its reliability. By construction, the probability that an RBD be operational is thus the reliability of the system it represents.

In our case, the system is the multiprocessor static schedule, possibly partial, of  $\mathcal{A}lg$  onto  $\mathcal{A}rc$ . Each block represents an operation  $X$  placed onto a processor  $P$  or a data-dependency  $X \triangleright Y$  placed onto a communication link  $L$ . The reliability of a block is therefore computed according to Eq (5).

Computing the reliability in this way requires the occurrences of the failures to be *statistically independent events*. Without this hypothesis, the fact that some blocks belong to several paths from  $S$  to  $D$  makes the reliability computation infeasible. At each step of the scheduling heuristics, we compute the RBD of the partial schedule obtained so far, then we compute the reliability based on this RBD, and finally we compute the GSFR of the partial schedule with Eq (6).

Finally, computing the reliability of an RBD with replications is, in general, exponential in the size of the schedule. To avoid this problem, we insert *routing operations* so that the RBD of any partial schedule is always *serial-parallel* (i.e., a sequence of parallel macro-blocks), hence making the GSFR computation *linear* [6]. The idea is that, for each data dependency  $X \triangleright Y$  such that it has been decided to replicate  $X$   $k$  times and  $Y$   $\ell$  times, a routing operation  $R$  will collect all the data sent by the  $k$  replicas of  $X$  and send it to the  $\ell$  replicas of  $Y$  (see Figure 5).

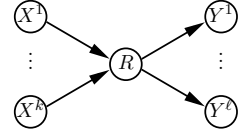


Fig. 5. A routing operation

### 4.3 Scheduling Heuristics

To obtain the Pareto front in the space (length,GSFR,power), we pre-define a virtual grid in the objective plane (GSFR,power), and for each cell of the grid we solve one different single-objective problem constrained to this cell, by using scheduling heuristics TSH presented below.

TSH is a *greedy list scheduling heuristic*. It takes as input an algorithm graph  $Alg$ , a homogeneous architecture graph  $Arc$ , the function  $\mathcal{E}xe$  giving the WCETs and WCCTs, and two constraints  $\Lambda_{obj}$  and  $P_{obj}$ . It produces as output a static multiprocessor schedule  $S$  of  $Alg$  onto  $Arc$ , such that the GSFR of  $S$  is smaller than  $\Lambda_{obj}$ , the power consumption is smaller than  $P_{obj}$ , and such that its length is as small as possible. TSH uses *active replication of operations* to meet the  $\Lambda_{obj}$  constraint, *dynamic voltage scaling* to meet the  $P_{obj}$  constraint, and the *power-efficient schedule pressure* as a cost function to minimize the schedule length.

TSH works with two lists of operations of  $Alg$ : the candidate operations  $\mathcal{O}_{cand}^{(n)}$  and the already scheduled operations  $\mathcal{O}_{sched}^{(n)}$ . The superscript  $(n)$  denotes the current iteration of the scheduling algorithm. One operation is scheduled at each iteration. Initially,  $\mathcal{O}_{sched}^{(0)}$  is empty while  $\mathcal{O}_{cand}^{(0)}$  contains the input operations of  $Alg$ . At any iteration  $(n)$ , all the operations in  $\mathcal{O}_{cand}^{(n)}$  are such that all their predecessors are in  $\mathcal{O}_{sched}^{(n)}$ .

The power-efficient schedule pressure is a variant of the schedule pressure cost function [8], which tries to minimize the length of the critical path of the algorithm graph by exploiting the scheduling margin of each operation. The *schedule pressure*  $\sigma$  is computed for each operation  $o_i$ , and each processor  $p_j$  as:

$$\sigma^{(n)}(o_i, p_j) = ETS^{(n)}(o_i, p_j) + LTE^{(n)}(o_i) - CPL^{(n-1)} \quad (9)$$

where  $CPL^{(n-1)}$  is the critical path length of the partial schedule composed of the already scheduled operations,  $ETS^{(n)}(o_i, p_j)$  is the earliest time at which the operation  $o_i$  can start its execution on the processor  $p_j$ , and  $LTE^{(n)}(o_i)$  is the latest start time from end of  $o_i$ , defined to be the length of the longest path from  $o_i$  to  $Alg$ 's output operations; this path contains the ‘‘future’’ operations of  $o_i$ . When computing  $LTE^{(n)}(o_i)$ , since the future operations of  $o_i$  are not scheduled yet, we do not know their actual voltage, and therefore neither what their execution time will be (this will only be known when these future operations will be actually scheduled). Hence, for each future operation, we compute its average WCET for all existing supply voltages.

First, we generalize the schedule pressure  $\sigma$  to a set of processors:

$$\sigma^{(n)}(o_i, \mathcal{P}_k) = ETS^{(n)}(o_i, \mathcal{P}_k) + LTE^{(n)}(o_i) - CPL^{(n-1)} \tag{10}$$

where  $ETS^{(n)}(o_i, \mathcal{P}_k) = \max_{p_j \in \mathcal{P}_k} ETS^{(n)}(o_i, p_j)$ .

Then, we consider the schedule length as a criterion to be minimized, and the GSF $R$  and the power as two constraints to be met: for each candidate operation  $o_i \in \mathcal{O}_{cand}^{(n)}$ , we compute the best subset of pairs  $\langle \text{processor, voltage} \rangle$  to execute  $o_i$  with the *power-efficient schedule pressure* of Eq (III):  $\mathcal{Q}_{best}^{(n)}(o_i) = \mathcal{Q}_j$  s.t.:

$$\sigma^{(n)}(o_i, \mathcal{Q}_j) = \min_{\mathcal{Q}_k \in \mathcal{Q}} \left\{ \sigma^{(n)}(o_i, \mathcal{Q}_k) \mid \Lambda^{(n)}(o_i, \mathcal{Q}_k) \leq \Lambda_{obj} \wedge P^{(n)}(o_i, \mathcal{Q}_k) \leq P_{obj} \right\} \tag{11}$$

where  $\mathcal{Q}$  is the set of all subsets of pairs  $\langle p, v \rangle$  such that  $p \in \mathcal{P}$  and  $v \in \mathcal{V}$  (see Section A.II), and  $\Lambda^{(n)}(o_i, \mathcal{Q}_k)$  (resp.  $P^{(n)}(o_i, \mathcal{Q}_k)$ ) is the GSF $R$  (resp. the power consumption) of the partial schedule after replicating and scheduling  $o_i$  on all the processors of  $\mathcal{Q}_k$  with their respective specified voltages. When computing  $\Lambda^{(n)}(o_i, \mathcal{Q}_k)$ , the failure rate of each processor is computed by Eq (7) according to its voltage in  $\mathcal{Q}_k$ . Finally,  $P^{(n)}(o_i, \mathcal{Q}_k)$  is computed by Eq (8).

To guarantee that the constraint  $\Lambda^{(n)}(o_i, \mathcal{Q}_k) \leq \Lambda_{obj}$  is met, the subset  $\mathcal{Q}_k$  is selected such that the GSF $R$  of the parallel macro-block that contains the replicas of  $o_i$  on the processors of  $\mathcal{Q}_k$  is less than  $\Lambda_{obj}$ . If this last macro-block  $B$  is such that  $\Lambda(B) \leq \Lambda_{obj}$  and if  $\Lambda^{(n-1)} \leq \Lambda_{obj}$ , then  $\Lambda^{(n)} \leq \Lambda_{obj}$  (thanks to the invariance property of the GSF $R$ ).

Similarly, the subset  $\mathcal{Q}_k$  is selected such that the power constraint  $P^{(n)}(o_i, \mathcal{Q}_k) \leq P_{obj}$  is met. There can exist several valid possibilities for the subset  $\mathcal{Q}_k$  (valid in the sense that the power constraint is met). However, some of them may lead to the impossibility of finding a valid schedule for the *next* scheduled operation, during step  $n+1$ . In particular, this is the case when the next scheduled operation *does not* increase the schedule length, because it fits in a slack of the previous schedule:  $L^{(n+1)} = L^{(n)}$ . At the same time, the total energy increases strictly because of the newly scheduled operation:  $E^{(n+1)} > E^{(n)}$ . By hypothesis, we have  $P^{(n)} = E^{(n)} / L^{(n)} \leq P_{obj}$ , but it follows that  $P^{(n+1)} = E^{(n+1)} / L^{(n+1)} = E^{(n+1)} / L^{(n)} > E^{(n)} / L^{(n)} = P^{(n)}$ , so even though  $P^{(n)} \leq P_{obj}$ , it may very well be the case that  $P^{(n+1)} > P_{obj}$ . To prevent this and guarantee the invariance property of  $P$ , we *over-estimate* the power consumption, by computing the consumed energy as if all the ending slacks were ‘‘filled’’

by an operation executed at  $P_{max}$ .  $P_{max}$  is the computed power under the highest frequency  $f_m$  such that  $P_{ind} + P_{max} = P_{ind} + C_{ef}V^2f \leq P_{obj}/N$ , where  $N$  is the processors number. If the consumed power with  $f_m$  exceeds  $P_{obj}$ , then the next highest operating frequency  $f \leq f_m$  is selected, and so on. Thanks to this over-estimation, even if the next scheduled operation fits in a slack and does not increase the length, we are sure that it will not increase the power-consumption either. This is illustrated in Figure 6. For lack of space, we do not study in this paper the impact of this over-estimation on the total schedule length.

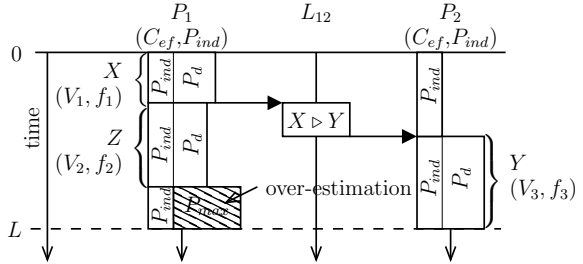


Fig. 6. Over-estimation of the energy consumption

Once we have computed, for each candidate operation  $o_i$  of  $\mathcal{O}_{cand}^{(n)}$ , the best subset of pairs (processor, voltage) to execute  $o_i$ , with the power-efficient schedule pressure of Eq (11), we compute the *most urgent* of these operations by:

$$o_{urg} = o_i \in \mathcal{O}_{cand}^{(n)} \text{ s.t. } \sigma^{(n)}(o_i, \mathcal{Q}_{best}^{(n)}(o_i)) = \max_{o_j \in \mathcal{O}_{cand}^{(n)}} \left\{ \sigma^{(n)}(o_j, \mathcal{Q}_{best}^{(n)}(o_j)) \right\} \quad (12)$$

Finally, we schedule this most urgent operation  $o_{urg}$  on the processors of  $\mathcal{Q}_{best}^{(n)}(o_j)$ , and we finish the current iteration ( $n$ ) by updating the lists of scheduled and candidate operations:  $\mathcal{O}_{sched}^{(n)} := \mathcal{O}_{sched}^{(n-1)} \cup \{o_{urg}\}$  and  $\mathcal{O}_{cand}^{(n+1)} := \mathcal{O}_{cand}^{(n)} - \{o_{urg}\} \cup \{t' \in succ(o_{urg}) \mid pred(t') \subseteq \mathcal{O}_{sched}^{(n)}\}$ .

## 5 Simulation Results

We perform two kinds of simulations. Firstly, Figure 7 shows the Pareto fronts produced by TSH for a randomly generated *Alg* graph of 30 operations, and a fully connected and homogeneous *Arc* graph of respectively 3 and 4 processors; we have used the same random graph generator as in [6]. The nominal failure rate per time unit of all the processors is  $\lambda_p = 10^{-5}$ ; the nominal failure rate per time unit of all the links is  $\lambda_\ell = 5.10^{-4}$ ; these values are reasonable for modern fail-silent processors [1]; the set of supply voltages is  $\mathcal{V} = \{0.25, 0.50, 0.75, 1.0\}$  (scaling factor).

The virtual grid of the Pareto front is defined such that both high and small values of  $P_{obj}$  and  $A_{obj}$  are covered within a reasonable grid size. Hence, the decreasing values of  $P_{obj}$  and  $A_{obj}$ , starting with  $+\infty$  and  $+\infty$ , are selected from two sets of values:  $A_{obj} \in \{\alpha \cdot 10^{-\beta}\}$  where  $\alpha \in \{4, 8\}$  and  $\beta \in \{1, 2, \dots, 20\}$ , and  $P_{obj} \in \{0.8, 0.6, 0.4, 0.2\}$ . TSH being a heuristics, changing the parameters of this grid could change locally some points of the Pareto front, but not its overall shape.

The two figures connect the set of non-dominated Pareto optima (the surface obtained in this way is only depicted for a better visual understanding; by no means do we assume that points interpolated in this way are themselves Pareto optima, only the computed dots are). The figures show an increase of the schedule length for points with decreasing power consumptions and/or failure rates. The “cuts” observed at the top and the left of the plots are due to low power constraints and/or low failure rates constraints.

Figure 7 exposes to the designer a choice of several tradeoffs between the execution time, the power consumption, and the reliability level. For instance in Figure 7 (right), we see that, to obtain a GSFR of  $10^{-10}$  with a power consumption of 1.5 V, then we must accept a schedule three times longer than if we impose no constraint on the GSFR nor the power. We also see that, by providing a 4 processor architecture, we can obtain schedules with a shorter execution length even though we impose identical constraints to the GSFR and the power.

Secondly, Figure 8 shows how the schedule length varies, respectively in function of the required power consumption (left) or of the required GSFR (right). Both curves are averaged over 30 randomly generated *Alg* graphs. We can see that the average schedule length increases when the constraint  $P_{obj}$  on the power

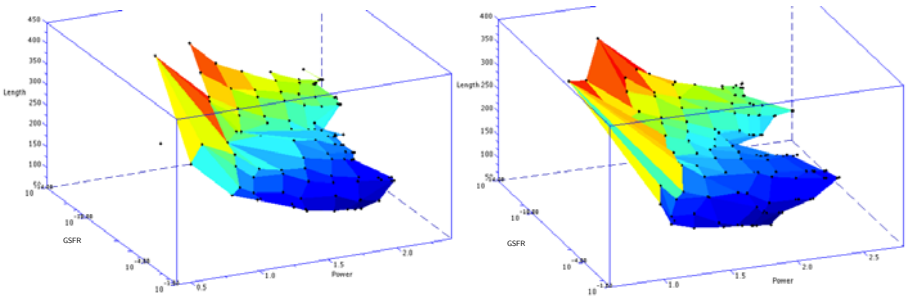


Fig. 7. Pareto front generated for a random graph of 30 operations on 3 processors (left) or 4 processors (right)

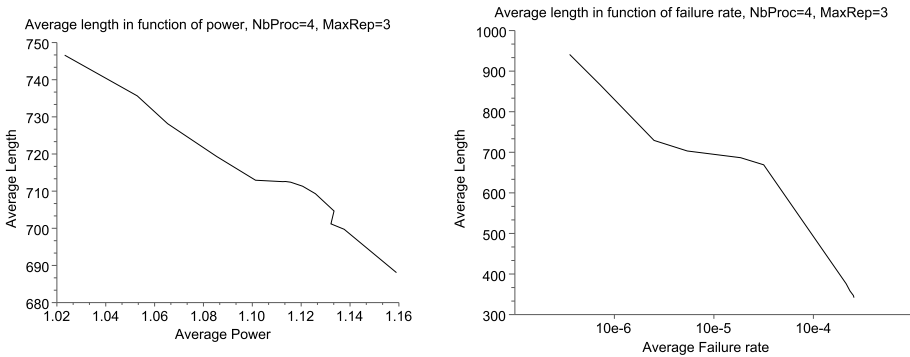


Fig. 8. Average schedule length in function of the power (left) or the GSFR (right)

consumption decreases. This was expected since the two criteria, schedule length and power consumption, are antagonistic. Similarly, the average schedule length increases when the constraint  $\Lambda_{obj}$  on the GSFR decreases. Again, the two criteria, schedule length and GSFR, are antagonistic.

## 6 Related Work

Many solutions exist in the literature to optimize the schedule length and the energy consumption (e.g., [13]), or to optimize the schedule length and the reliability (e.g., [4,7,2]), but very few tackle the problem of optimizing the *three* criteria (length,reliability,energy). The closest to our work are [19,14].

Zhu et al. have studied the impact of the supply voltage on the failure rate [19], in a passive redundancy framework (primary backup approach). They use DVFS to lower the energy consumption and they study the tradeoff between the energy consumption and the performability (defined as the probability of finishing the application correctly within its deadline in the presence of faults). A lower frequency implies a higher execution time and therefore less slack time for scheduling backup replicas, meaning a lower performability. However, their input problem is not a multiprocessor scheduling one since they study the system as a single monolithic operation executed on a single processor. Thanks to this simpler setting, they are able to provide an analytical solution based on the probability of failure, the WCET, the voltage, and the frequency.

Pop et al. have addressed the (length,reliability,energy) tricriteria optimization problem on an heterogeneous architecture [14]. Both length and reliability are taken as a constraint. These two criteria are *not* invariant measures, and we have demonstrated in Section 2 that such a method *cannot always guarantee* that the constraints are met. Indeed, their experimental results show that the reliability decreases with the number of processors, therefore making it impossible to meet an arbitrary reliability constraint. Secondly, they assume that the user will specify the number of processor failures to be tolerated in order to satisfy the desired reliability constraint. Thirdly, they assume that all the communications take place through a reliable bus. For these three reasons, it is not possible to compare TSH with their method.

## 7 Conclusion

We have presented a new off-line tricriteria scheduling heuristics, called TSH, to minimize the schedule length, its global system failure rate (GSFR), and its power consumption. TSH uses the *active replication* of the operations and the data-dependencies to increase the reliability, and uses *dynamic voltage and frequency scaling* to lower the power consumption. Both the power and the GSFR are taken as *constraints*, so TSH attempts to minimize the schedule length while satisfying these constraints. By running TSH with several values of these constraints, we are able to produce a set of non-dominated Pareto solutions, which is a surface in the 3D space (length,GSFR,power). This surface exposes the

existing tradeoffs between the three antagonistic criteria, allowing the user to choose the solution that best meets his/her application needs. TSH is an extension of our previous bicriteria (length,reliability) heuristics BSH [6]. The tricriteria extension is necessary because of the crucial impact of the voltage on the failure probability.

To the best of our knowledge, this is the *first* reported method that allows the user to produce the Pareto front in the 3D space (length,GSFR,power). This advance comes at the price of several assumptions: the architecture is assumed to be homogeneous and fully connected, the processors are assumed to be fail-silent and their failures are assumed to be statistically independent, the power switching time is neglected, and the failure model is assumed to be exponential.

## References

1. Baleani, M., Ferrari, A., Mangeruca, L., Peri, M., Pezzini, S., Sangiovanni-Vincentelli, A.: Fault-tolerant platforms for automotive safety-critical applications. In: International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES 2003, San Jose (CA), USA. ACM, New-York (2003)
2. Benoit, A., Dufossé, F., Girault, A., Robert, Y.: Reliability and performance optimization of pipelined real-time systems. In: International Conference on Parallel Processing, ICPP 2010, San Diego (CA), USA (September 2010)
3. Burd, T.D., Brodersen, R.W.: Energy efficient CMOS micro-processor design. In: Hawaii International Conference on System Sciences, HICSS 1995, Honolulu (HI), USA. IEEE, Los Alamitos (1995)
4. Dogan, A., Özgüner, F.: Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. IEEE Trans. Parallel and Distributed Systems 13(3), 308–323 (2002)
5. Elnozahy, E., Kistler, M., Rajamony, R.: Energy-efficient server clusters. In: Workshop on Power-Aware Computing Systems, WPACS 2002, Cambridge (MA), USA, pp. 179–196 (February 2002)
6. Girault, A., Kalla, H.: A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. IEEE Trans. Dependable Secure Comput. 6(4), 241–254 (2009)
7. Girault, A., Saule, E., Trystram, D.: Reliability versus performance for critical applications. J. of Parallel and Distributed Computing 69(3), 326–336 (2009)
8. Grandpierre, T., Lavarenne, C., Sorel, Y.: Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In: International Workshop on Hardware/Software Co-Design, CODES 1999, Rome, Italy. ACM, New York (1999)
9. Knight, J.C., Leveson, N.G.: An experimental evaluation of the assumption of independence in multi-version programming. IEEE Trans. Software Engin. 12(1), 96–109 (1986)
10. Leung, J.Y.-T. (ed.): Handbook of Scheduling. Algorithms: Models, and Performance Analysis. Chapman & Hall/CRC Press (2004)
11. Lloyd, D., Lipow, M.: Reliability: Management, Methods, and Mathematics. ch.9. Prentice-Hall, Englewood Cliffs (1962)
12. Melhem, R., Mossé, D., Elnozahy, E.N.: The interplay of power management and fault recovery in real-time systems. IEEE Trans. Comput. 53(2), 217–231 (2004)

13. Pering, T., Burd, T.D., Brodersen, R.W.: The simulation and evaluation of dynamic voltage scaling algorithms. In: International Symposium on Low Power Electronics and Design, ISLPED 1998, Monterey (CA), USA, pp. 76–81. ACM, New York (August 1998)
14. Pop, P., Poulsen, K., Izosimov, V.: Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In: International Conference on Hardware-Software Codesign and System Synthesis, CODES+ISSS 2007, Salzburg, Austria. ACM, New York (October 2007)
15. Shatz, S.M., Wang, J.-P.: Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems. *IEEE Trans. Reliability* 38(1), 16–26 (1989)
16. Souyris, J., Pavec, E.L., Himbert, G., Jégu, V., Borios, G., Heckmann, R.: Computing the worst case execution time of an avionics program by abstract interpretation. In: International Workshop on Worst-case Execution Time, WCET 2005, Mallorca, Spain, pp. 21–24 (July 2005)
17. T'kindt, V., Billaut, J.-C.: *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer, Heidelberg (2006)
18. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The determination of worst-case execution times — overview of the methods and survey of tools. *ACM Trans. Embedd. Comput. Syst.* 7(3) (April 2008)
19. Zhu, D., Melhem, R., Mossé, D.: The effects of energy management on reliability in real-time embedded systems. In: International Conference on Computer Aided Design, ICCAD 2004, San Jose (CA), USA, pp. 35–40 (November 2004)



# Criticality-Driven Component Integration in Complex Software Systems

Antonio Pecchia, Roberto Pietrantuono, and Stefano Russo

Dipartimento di Informatica e Sistemistica,  
Università degli Studi di Napoli Federico II,  
Via Claudio 21, 80125, Naples, Italy  
{antonio.pecchia,roberto.pietrantuono,stefano.russo}@unina.it

**Abstract.** Complex software systems are commonly developed by integrating multiple, occasionally Off-The-Shelf (OTS), components. This process results into a more modular design and reduces development costs; however, it raises new dependability challenges in case of safety critical systems. Testing activities conducted during the development of the individual components might be not enough to ensure a proper safety level after the integration. The failures of the components and their impact on the overall system safety have to be assessed in critical scenarios. This paper proposes a method to support component integration in complex software systems. The method uses (i) the knowledge of the architectural dependencies among the system components, and (ii) the results of failure-modes emulation experiments, to assess both error propagation phenomena within the system and the criticality of the components in the system architecture. This information is valuable to design effective error-mitigation means and, when needed, to select the most suitable OTS item if multiple equivalent options are available. The method is applied to a real world Air Traffic Control system, developed in the context of an academic-industrial collaboration.

**Keywords:** Integration, Criticality Assessment, Failure-modes Emulation, Air Traffic Control.

## 1 Introduction

The development of complex software systems increasingly relies on the integration of existing services and components (built-in-house and, occasionally, Off-The-Shelf (OTS)), rather than on items built entirely *from scratch*. This process is commonly adopted also in critical domains, because it results, in principle, into a more modular and reliable design and it allows reducing development costs and time-to-market [1]. However, this process might raise dependability issues that are related to the integration and interactions among components [2, 3], especially in critical contexts, such as avionic and railway systems.

Software items that are reused in the design of a system are often developed either referring to a *different context*, i.e., developed for another system, or without any *specific context* in mind, i.e., the items have been developed

with the precise goal of being reused. As a result, testing activities conducted during the development of these items might be not enough to ensure a proper service during operation, because of unforeseeable interactions with the system integrating them, and the execution environment. The failures of the individual components and their impact on the overall system safety have to be carefully assessed in critical scenarios. Furthermore, the cost of the integration activities (e.g., components selection, adapters development, integration testing, design of fault tolerance mechanisms), might be even higher than the cost for developing components from scratch, especially in large and complex systems.

In this paper we present a method to support component integration in complex software systems. The method uses a model of the system at a high level of abstraction. The model encompasses the architectural components of the system, and formalizes the dependencies among them. The model is then used to drive failure-modes emulation experiments that aim at investigating error propagation phenomena within the system. Based on the error propagation paths and the error mitigation means observed in the system, the method determines a *criticality level* for each component. A criticality level represents the impact that the failures of the component have on the overall system, i.e., *the impact of its integration*. Criticalities allow engineers (i) identifying the components whose integration is potentially dangerous (these components require either more integration testing, or the design of proper fault tolerance mechanisms), (ii) comparing different equivalent components, possibly OTSs, with respect to the impact they have on the overall system safety. This information supports decisions regarding fault tolerance mechanisms, allocation of integration testing efforts, and comparison/selection of (OTS) components.

The proposed method is applied to a real world Air Traffic Control (ATC) system, developed in the context of an academic-industrial collaboration involving a world leading company, SELEX-SI, and academic partners in the COSMIC<sup>1</sup> project. We assess the criticality of the components integrated in the ATC system. Furthermore, because of the need of the company's system developers to select and to integrate a suitable Data Distribution Service (DDS) in the system we compare two functionally equivalent DDS platforms from the dependability perspective. Obtained results show that the experiments conducted according to the proposed method, allow identifying architectural dependencies and resources of the execution environment that impact the overall system safety, thus driving the choices taken by the project team.

The rest of the paper is organized as follows. Section 2 surveys related work in the area of dependability assessment of critical systems. Section 3 describes the proposed method and the algorithm implementing it. Section 4 and 5 describe our experience with the ATC system, and provide the results of the experimental campaign. Section 6 discusses the implications of the results on design choices.

---

<sup>1</sup> COSMIC is a three-year Italian research project aiming to create a research laboratory for the development of an open source middleware for mission critical systems.

## 2 Related Work

Issues related to the development by integration, such as the difficulties in controlling/testing complex interactions among components [2], [3], make dependability a significant challenge in critical scenarios. Several organizations defined standards and methodologies, such as [4], [5], [6], to support the development of dependable systems. These standards define a set of tasks and evidence to produce during the phases of the software development cycle. However, these tasks may be *time consuming*, thus neglecting the needs of current software industry. Testing and validation efforts can be driven with a preliminary knowledge of the system, in terms of criticality of its components. The standards suggest adopting *hazard analysis* and *risk assessment* techniques, such as failure modes and effects analysis (FMEA), hazard and operability (HAZOP), event tree analysis (ETA), and fault tree analysis (FTA) [8]. For example, in [9] and [10] the authors describe the hazard analysis methodology used in railway dependable systems. In [11] safety assessment processes for ATM systems have been proposed.

Several works describe approaches based on a dynamic flow graph methodology (DFM) [12], [13] to generate timed fault trees, for assessing the risk associated with dynamic behaviors. Additionally, methodologies and/or technologies for the safety assessment of real complex infrastructures and operations have been proposed. Authors in [14] present a case study to apply a goal-oriented method for car security-related hazard analysis. In [15], it has been proposed a model based on a network representation, where objects represent concepts and links represent relations. Nevertheless, this type of works do not consider the impact of the system architecture on dependability attributes.

Some issues might compromise the effectiveness of existing approaches in industrial scenarios. For example, the DFM analysis does not provide mechanisms to cope with the computational complexity of large-scale software systems. Furthermore, risk assessment is often performed by examining only faults at the interface level without considering the mitigation means included in the architecture of the system. To overcome these limitations, the proposed method adopts a system model, whose *gain* is decided by the analyst; this allows lowering the complexity of the assessment task. Furthermore, the failure-modes emulation experiments highlight error mitigation means implemented by the system.

## 3 Integration Strategy

In the following we describe the integration strategy. Section 3.1 provides background definitions. The assumptions for the failure-modes emulation experiments are described in Section 3.2. Section 3.3 formalizes the integration algorithm.

### 3.1 Background: System Model and Criticality Levels

A software system is assumed to be made by a set of *software elements*, which interact to implement the services provided by the system. Elements consist of

*entities* and *resources*. An **entity**, is a stateful *active* element, which interacts with resources and/or other entities. A **resource**, is stateful *passive* element, i.e., it does not interact with any other system element. Interactions can induce a state modification in the target element (*stateful* interactions), or they can leave the state of the element unchanged (*stateless* interactions).

We focus on the *dependencies* among the system elements to take into account propagation phenomena. More specifically, we assume the existence of (i) a **control dependency** between two entities A and B (or between an entity A and a resource R), if there is a direct interaction between them, and (ii) a **state dependency** between two entities A and B, if there is an interaction between A and a resource on which B performs a *stateful* interaction.

A **service** provided by the system is implemented via a sequence of interactions. Let  $s$  be such a service, with  $s = 1 \dots N$  ( $N$  is the total number of services). We associate two matrix to each service  $s$ , as follows. To represent *control dependencies*, we define a matrix  $C_s$ ,  $(n + m) * (n + m)$ , with  $n$  denoting the number of entities and  $m$  the number of resources. The element  $C_{s_{ij}}$  is 1 if an interaction exists between the entity  $E_i$  and the entity (resource)  $E_j$  ( $R_j$ ). Fig 2 shows an example of this type of matrix with reference to the case study. To obtain the *state dependencies* between the elements implementing a service  $s$ , we calculate a matrix  $S_s$ , as follows: (i) for each  $C_s$ , we extract a matrix  $Cr_s$ , with a value 1 only for each *entity-resource* stateful interaction, (ii) then, we sum each obtained  $Cr_s$  matrix, and transpose the resulting matrix, obtaining  $C_T$ , (iii) finally, for each service  $s$ ,  $C_s * C_T$  (rows \* columns) returns  $S_s$ .

The criticality of a system element is quantified via the notion of **criticality level** (CL). The value of a CL is related to the *severity* of the effects of the element failures. Several standards for mission and safety critical systems, such as [4], [5], and [7], provide specific CL rankings. In the context of this work, without loss of generality, we consider a *generic ranking* encompassing four CLs, i.e.,  $1 \dots 4$ , with 1 denoting the highest CL. We adopt a reverse ranking (higher the criticality, lower the CL) as in the avionic standards, because of the nature of proposed case study; however, any other choice would have been equivalent. Adopted CLs, which, again, represent the risk associated with a system function, and, in turn, with the software system element(s) implementing the function, are: HIGH (1), i.e., software whose failure causes or contributes to the occurrence of a catastrophic condition, MEDIUM (2), for software whose failure results in major failure conditions, LOW (3), for software whose failure results in minor failure conditions, NO criticality (4), for software whose failure has no effect.

### 3.2 Assumptions

The proposed method adopts failure-modes emulation experiments to assess the integration risk. We assume a set of failure-modes representing *how* a system element (either entity or resource) can fail [16]. In the case of *entities* we consider (i) **crash**, i.e., the entity stops providing service due to unexpected failure; (ii) **passive hang**, i.e., the entity waits indefinitely for a resource which will never be released (e.g. deadlocks) or for signals which will never be generated; (iii) **active**

**hang**, i.e., the entity indefinitely halts, but it keeps the system resources busy. Failures related to *return* values are not considered in this study: we assume that an interaction with correct parameters does not modify the state inconsistently and returns a correct value, i.e. a value within the expected value domain [16]. Also, we assume that the underlying network does not alter returned values (reliable channels assumption). As for *resources*, we consider the following failure-modes (i) **access denied**: the resource becomes unavailable; (ii) **read denied**: the resource is accessed, but it can not satisfy a reading request; (iii) **write denied**: the resource is accessed, but it can not satisfy a writing request; (iv) **corruption**: the content of the resource is altered.

The entity-resource model has to be tailored to the system under analysis and to the chosen grain and level of abstraction. For example, application components and OS processes may be regarded as entities and resources might be OS resources or databases. *Entities and resources have to be identified before applying the algorithm*. Furthermore, we assume that the proposed method is applied after a preliminary hazard assessment step: for each *service* the potential hazards have been identified and assigned a criticality level.

### 3.3 Algorithm

The integration strategy is formalized as a novel algorithm that, expanding the set of depending entities *recursively*, assigns a criticality level to the system elements involved in the provisioning of a given service. Criticality levels are represented by i) the **CL** ranking defined above, for entities, and ii) **labels** for resources indicating if they are critical or not for the system. The algorithm outputs the integration risk for each entity as a result of i) individual criticality, and ii) failure propagation paths (and intrinsic mitigation means). We report a C-like version of the algorithm, whose key steps are detailed in the following.

```

1. void RLAssignment( system Sys ){
2. EntitySet E = all the entities
3. EntitySet BorderE = border entities set;
4. ENTITY e, reader, writer, ToExpand;
5. MATRIX C,S; //dependency matrices
6. int N,M; //Number of entities and resources
7. for(e ∈ BorderE ){
8.   InitialCL = getHaResult(e); //STEP 1: Initial definition of CLs
9.   //STEP 2: Control-CL assignment
10.  ToExpand = e;
11.  Expand (ToExpand); //function defined below
12.  //STEP 3: State-CL adjustment
13.  //For each pair of entities(ei,ej) set CLs according to Table 1
14.  for (Sij! = 0 ∈ S){ //Sij value is the name of resource causing the dependency
15.    readerEntity = getReaders(Sij);
16.    writerEntity = getWriters(Sij);
17.    if (writerEntity ! ∈ (R.isNotRobust)){//if it is not robust to R failures
18.      AssignCL(writer)=min(reader,writer); //it pushes a CL value onto
19.      //the writer CL stack
20.      SetRobustEntities(Sij,writerEntity) //set the Sij isNotRobust vector
21.    } //value for "writerEntity" entry
22.  }
23. for (e ∈ E){ //STEP 4: Final adjustment of the CL value
24.   FinalCL(e) = min(e.CLs);
25. }

```

```

1. voidExpand (Entity ToExpand) {
2. RESOURCE dcR; ENTITY dcE;
3. EntitySet DCE; ResourceSet DCR;
4. //building depending entities and resources sets
5. int i = getRow(ToExpand)//get the C row corresponding to "ToExpand"
6. for (int j = 0 to N+M){
7.     if (Cij != 0) {
8.         if (j <= N)
9.             DCE = DCE + Ej;
10.        else
11.            DCR = DCR + R(j-N);
12.    }
13. //evaluating robustness of "ToExpand" entity with respect to dcR failures
14. for( dcR ∈ DCR) {
15.     if (ToExpand ∈ dcR.isNotRobust)
16.     //add ToExpand entity to isNotRobust vector resource dcR
17.         SetRobustEntities(dcR,ToExpand);
18. }
19. //evaluating robustness of "ToExpand" entity with respect to dcE failures
20. for(dcE ∈ DCE){
21.     if (ToExpand! ∈ dcE.isNotRobust)
22.         AssignCL(dcE)= getCL(ToExpand)+1; //lower its risk level
23.     else//the same risk level
24.         AssignCL(dcE)=getCL(ToExpand);
25.     ToExpand = dcE;
26.     Expand(ToExpand); //recursive call
27. }}

```

The main types in the algorithm are: ENTITY, RESOURCE and EntitySet, i.e. a set of "Entity". Each ENTITY type has associated a stack of CL values, named "CLs", because of possible involvement in more than one service and therefore in more than one CL assignment. Both RESOURCE and ENTITY types have associated a vector of entity names, named "*isNotRobust*". For each resource R (entity E), this vector contains all the entities that are experienced as not robust to the resource R (entity E) failures itself. A set of auxiliary functions is explained by comments.

The goal is to verify (by diving into the system elements tree) if a specific element implements mitigation means to tolerate, if not, to stop, the propagation of a failure induced by other depending elements. In this case, the criticality level of the failing element can be lowered, or the element can be labeled as non-critical resource, because its failures are tolerated; otherwise, the criticality level remains unchanged. The main subsequent steps follow:

**STEP 1.** A starting CL is assigned to the entities directly responsible for providing a given service, i.e. the so-called *border entities*: for all identified hazards related to the given service (again, it is assumed that a preliminary hazard assessment has been performed (Section 3.2) ), the minimum observed CL is assigned to the border entity.

**STEP 2.** Given an entity  $E_i$ , with an assigned CL value, this step aims to (i) assign a CL to the entities which it depends on (by a *control dependency*) and (ii) assess the robustness of  $E_i$  with respect to the failure of resources which it has a *control dependency* (due to reading or writing operations). The dependency relation causes a mutual effect among involved entities, so that a failure in one of them can impact the behavior of depending ones. If we denote with  $E_i$  the entity with an assigned CL, and with  $E_j$  a depending entity, in order to assign a

**Table 1.** CL assignment due to state dependency.  $s = \min(\text{CL}[E_i], \text{CL}[E_j])$ 

$E_j$ (write) $E_i$ (read)	Not robust	Robust
Not Robust	$\text{CL}[E_j] = s$ reading/writing critical resource	<b>CL unchanged</b> reading critical resource
Robust	$\text{CL}[E_j] = s$ writing critical resource	<b>CL unchanged</b> non critical resource

CL to  $E_j$ , we consider the behavior of  $E_i$  once a failure of  $E_j$  occurs. According to the failure-modes  $E_i$  can tolerate the failure of  $E_j$ , mitigate it, or it can not be able to tolerate such a failure. If the failure of  $E_j$  is mitigated, we can lower the criticality level of  $E_j$  (i.e. increasing its ranking). Otherwise, we set  $\text{CL}[E_j] = \text{CL}[E_i]$ , because  $E_j$  has to be considered as critical at least as  $E_i$ . In other words, if the failure of an entity is mitigated, this entity can be considered less critical for the overall system. In practice, we evaluate the robustness of an entity to other entities failures, through failure injections campaigns, according to the adopted failure-modes. As for *resources*, we are interested in figuring out if a resource is risky for the system, i.e., if its failures are not tolerated by the entities accessing it.

**STEP 3.** In the third step, we modify CLs according to the *state dependencies*. Two entities, e.g.,  $E_i$  and  $E_j$ , might depend on each other through a resource  $R$  by either reading or writing access. If both entities read from  $R$ , there is no dependency between them, because they do not alter the state of the resource. Similarly, if both the entities only write on  $R$  (thus no one reads the changed state) we say that there is no dependency. A dependency exists if one of the entities writes and the other one reads the resource. In this case, if both entities have been assigned a CL, there are four possibilities shown in Table 1. A CL can be modified according to the robustness of the entities to the failures of  $R$ . If the writing entity is robust to the failures of  $R$  (e.g., if it can detect and recover from a failure by exploiting temporal and/or spatial redundancy), we do not modify the CL assigned in the previous step. Otherwise, if it does not tolerate the failures of  $R$ , i.e., some writings can be lost and this can compromise the *reading* entity, we have to consider  $\text{CL}[E_j]$  at least critical as the reading entity (i.e.,  $\text{CL}[E_j] \leq \text{CL}[E_i]$ ).

**STEP 4.** In the final step, we *adjust* the CL values. Since the algorithm analyzes the system through each of the provided services individually, an entity  $E_i$  might be involved in more than one service and thus assigned more than one CL. The final CL for  $E_i$  is the minimum of observed CLs.

The output of the algorithm is a set of labels: for the entities, they indicate a CL value; as for resources, each label points out if it is a critical resource for the system or not. Furthermore, the algorithm allows achieving insights on the failure propagation paths and intrinsic mitigation means of the system.

## 4 Case Study

The proposed integration strategy is applied to an Air Traffic Control (ATC) system developed in the context of an academic-industrial collaboration. A preliminary experience with this system is presented in [19]. The components and the middleware layers composing the system are described in Section 4.1. In Section 4.2 we present how the proposed strategy has been applied to the system.

### 4.1 ATC System

The reference case study consists of a real-world ATC system. In particular, we consider a **Flight data Plan (FPL) Processor**, developed atop an open-source middleware platform, named CARDAMOM<sup>2</sup>. A FPL provides information, such as, the flight route, the expected trajectory of the airplane, airplane-related information, and meteorological data. The ATC system uses (i) services of the CARDAMOM platform, such as, the Load Balancer (LB), Replication (R), and System Management (SMG), and (ii) an OMG-compliant<sup>3</sup> Data Distribution Service (DDS) [17]. The DDS allows the components of the application to transmit the FPL instances. This is done by means of the `read` and `write` facilities provided by the DDS API, which allows to retrieve and to publish a FPL instance, respectively.

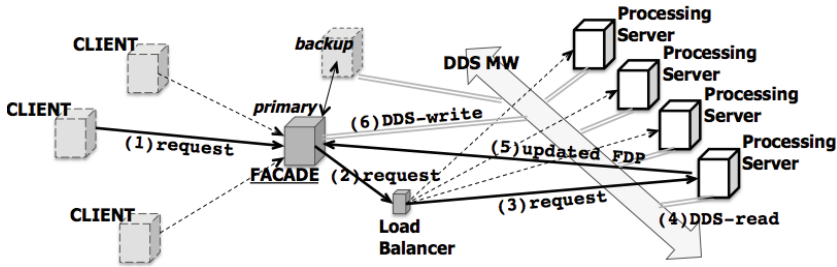


Fig. 1. Overview of the target system: FPL Processor

Fig. 1 depicts the FPL Processor. It is implemented as a CORBA-based distributed objects system. The FPL Processor is composed by the **Facade** object and a pool of **Processing Servers** managed via the LB service. The system components interact as follows: the Facade object accepts FPL processing requests (i.e., insert, delete, update) supplied by external **Clients** and guarantees the data consistency by means of mutual exclusion among requests accessing the same FPL instance. The Facade redirects each allowed request to 1 out of N Processing Server, according to the *round robin* service policy. The selected

<sup>2</sup> CARDAMOM is a CORBA-based middleware platform providing services to support the development of software architectures for safety and mission critical systems (<http://forge.objectweb.org/projects/cardamom>)

<sup>3</sup> OMG specification for the Data Distribution Service, <http://www.omg.org>



server (i) retrieves the specified FPL instance from the DDS middleware (e.g., DDS MW in Fig.1) by means of the `read` facility (ii) executes request-specific computations, and (iii) returns the updated FPL instance to the Facade. The latter publishes the updated FPL instance by means of the DDS `write` facility and finalizes the request. Machines composing the testbed (Intel Pentium 4 3.2 GHz, 4 GB RAM, 1,000 Mb/s Network Interface equipped) run a RedHat Linux Enterprise 4. An Ethernet LAN interconnects these machines. As normal operation profile, Client objects invoke the services provided by the Facade with an average frequency of 50 requests per second. About 4,000 FPLs instances, each of them of 77,812 bytes, are shared with the DDS MW.

### 4.2 Integration Strategy

The proposed strategy is applied to the ATC system as described in Section 3. In the context of the analysis conducted in this paper, we assume the components of the ATC application (Client, Facade and processing Servers) to be *entities* and the DDS a *resource*. However, it should be noted that the grain of the model is defined by the analyst; thus, alternative models might have been chosen without the need to modify the proposed algorithm.

The **interaction matrix** is built by taking into account the interactions among the system components, as described in Section 4.1. The matrix is shown in Fig.2 (A); it has to be noted that all the services of the system exhibit the same matrix. The interaction between the Facade and the DDS is **stateful**, as the Facade writes the updated version of the FPL instance to the DDS. As a result, a *state* dependence exists between each processing Server and the Facade.

The iterative algorithm is applied as follows. The Client, i.e., the *border* entity of the reference system, is the entry point of the algorithm (Fig.2 (B) - step 1). We assume HIGH to be a suitable criticality level for this entity; the choice is

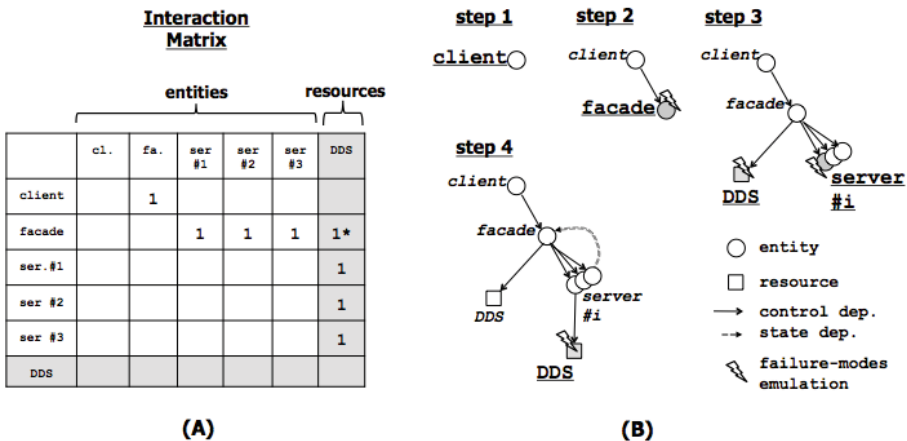


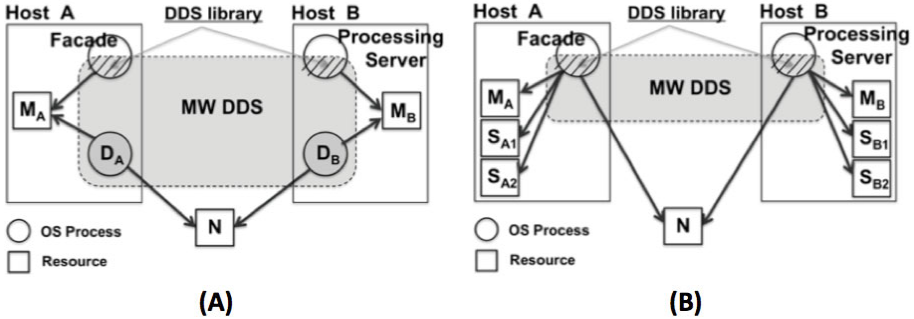
Fig. 2. Experimentation: (A) interaction matrix; (B) steps of the algorithm

**Table 2.** Failure-modes emulation: entities (ENT.), resources (RES.)

ENT.	<u>ATC component (CORBA object)</u>		
crash	the process is terminated by means of bad manipulations of an uninitialized pointer		
active hang	triggering of an infinite loop in the code		
passive hang	infinite wait on a locked semaphore		
RES.	<u>shared memory</u>	<u>semaphore</u>	<u>network</u>
access denied	the <code>vm_area_struct</code> related to the target shared memory is deleted from the addressing space of the process	target semaphore is deleted with <code>ipcrm</code> command-line util	network is made unavailable in two different ways (i) via the <code>ifconfig eth0 down</code> command (ii) network cable disconnection
read denied	bits storing the memory access policies are modified by interacting with the OS paging sub-system [18]	access permissions are modified with <code>semctl</code> ; 200 is set as new value	<i>not meaningful in the case study</i>
write denied	bits storing the memory access policies are modified by interacting with the OS paging sub-system [18]	access permissions are modified with <code>semctl</code> ; 400 is set as new value	<i>not meaningful in the case study</i>
corrupt.	bit-flip technique. We perform experiments by flipping a <i>single bit</i> or a <i>bit sequence</i> of increasing sizes {10, 100, 1,000, 10,000, 100,000}	target semaphore content is modified with <code>semctl</code> and the <code>SETVAL</code> flag	negligible for the case study. A dedicated LAN environment interconnects testbed machines.

reasonable in the case study since the Client represents the most external point where the service is delivered. We investigate how the failures emulated in the Facade object impact the Client in the first iteration of the algorithm (Fig. 2 (B) - step 2); obtained results allow allocating a proper criticality level to the Facade object. We subsequently analyze how the Facade object behaves in case of failures emulated in the components it depends on, i.e., the processing Server and the DDS (Fig. 2 (B) - step 3). Again, a criticality level is allocated to the Server and we label the DDS (at the Facade, i.e., **writer**, side) as *critical* or *non-critical* resource, according to the results of the failure-modes analysis. In the last iteration (Fig. 2 (B) - step 4), we investigate how failures emulated in the DDS impact the processing Server; the DDS (at the Server, i.e., **reader**, side) is labelled as *critical* or *non-critical* resource. Failures in the Facade and Server object are emulated by triggering a faulty piece of code when a request is invoked by the Client. The adopted emulation mechanisms are described in Table 2 (ENT.). Experiment results are described in 5.1

We analyze two OTS alternatives, coming from different vendors, as DDS middleware. The DDS middleware plays a key role in the described ATC system, both because (i) of the criticality of the domain and (ii) of the workload. For reasons of confidentiality we do not disclose the actual names of the vendors; we refer with **DDS\_1** and **DDS\_2** to the two DDS implementations. Both DDSs are integrated in the ATC system as follows: a shared library (`*.so`, shared object) is linked to the component that wants to use the DDS. The library relies on



**Fig. 3.** Internal architecture of the DDS: (A) DDS\_1, (B) DDS\_2

external resources, e.g., shared memories and/or semaphores, used as support data structures (Fig.3). To assess the criticality of the DDS, we investigate how both the Facade and Server behave in case of failures occurring in the resources used by the DDS library. Table 2 (RES.) shows how these failures are emulated in the context of the Linux OS. Failures have been injected during the system operational time with kernel modules. The analysis of the criticality of the DDS is reported in Section 5.2 and 5.3, for each implementation, respectively.

## 5 Failure-Modes Emulation Campaign

We conduct the failure-modes emulation campaign as described in 4.2. Each emulation experiment has been repeated 10 times, in order to ensure that the results were not distorted by transitory phenomena.

### 5.1 ATC Components: Facade and Processing Server

The *entry point* of the integration algorithm is the Client object, whose criticality level has been assumed to be **HIGH**. We analyze the interaction **Client/Facade** beforehand; results of the failure-modes emulation experiments allow allocating a proper criticality level to the Facade. **Crash** failures of the Facade object causes the interruption of the service invoked by the Client. For example, an update request for a FPL instance does not succeed and the new data are lost, raising an exception. The emulation of **hang** failures (both **active** and **passive**) causes the interruption of the invoked service; in this case no exception is raised at the Client side. However, if other services are invoked after a hang failure, the Facade is able to process the new requests, since it is implemented as a multithreaded CORBA object. As a result, we conclude that, even if hangs are less critical than crash failures (the Facade encapsulates specific error mitigation mechanisms), a service is never able to succeed in case of failures in the Facade object. *The Facade is as critical as the Client. We assign HIGH as criticality level.*

We analyze the interaction **Facade/Server**. **Crash** failures of the Server object cause the request forwarded by the Facade to be lost. However, processing

Servers are organized as a load-balancing pool. For this reason, when the request is lost the Facade re-forwards the request to another server, until it is correctly executed. The emulation of **hang** failures (both **active** and **passive**) in the Server causes the interruption of the invoked service. Again, no exception is raised at the Facade side, however, the Server will be able to execute new incoming requests because of the multithreaded implementation. We conclude that the Facade is robust to crashes emulated in the Server; hangs are partially tolerated. *The processing Server is less critical than the Facade, thus it is assigned MEDIUM as criticality level.*

## 5.2 Analysis of the DDS\_1

The DDS\_1 consists of a shared library to be linked to the application, and internal middleware processes. Applicative processes (i.e., Facade and Processing Server, respectively) communicate with the DDS internal ones ( $D_A$  and  $D_B$ , *networking* processes) by means of a shared memory (Fig. 3 (A)). Middleware processes are responsible for the communication among the computing nodes of a domain. Facade and processing Server interact with *shared memories* (named  $M_A$  and  $M_B$ , respectively) on both the nodes. We investigate how the failures emulated in these resources impact the correct behavior of the system.

As depicted in Fig. 3 (A) the Facade object interacts with  $M_A$ . As expected, the Facade crashes with a “**segmentation fault**” message in case of an **access, read, or write denied**. This is due to the nature of the Linux OS paging subsystem. The **corruption** of  $M_A$  has different consequences, depending on the modified bits. In particular, the Facade enters a hang state if the corruption affects lowest  $M_A$  bits, a crash one, otherwise. The Processing Server interacts with  $M_B$ . It crashes with a “**segmentation fault**” message in case of an **access, read, or write denied**. Apparently, the **corruption** of  $M_B$  does not make the Processing Server to hang or crash; however, after the emulation experiment, the updated versions of FPL instances are not delivered to the server anymore. Furthermore, no error notifications are returned.

We conclude that *failures of  $M_A$  and  $M_B$  always compromise the mission of the ATC system*.  $M_A$  and  $M_B$  are *critical* resources at the *writer* and *reader* side, respectively, since the DDS library, integrated in the ATC components, does not encapsulate suitable mitigation means to tolerate injected failures.

## 5.3 Analysis of the DDS\_2

The DDS\_2 exhibits a different architecture (Fig. 3 (B)). All the code of the DDS is mapped into the application processes in the form of a shared library. As a result, Facade and processing Servers interact *directly* with *shared memories*, *semaphores* and the *network*. Let  $M_A$ ,  $S_{A1}$ ,  $S_{A2}$  and  $M_B$ ,  $S_{B1}$ ,  $S_{B2}$  be shared memories and semaphores at Facade and Processing Server side, respectively. Let  $N$  be the *network*. Fig. 3 (B) depicts the interactions to transmit data, i.e., FPLs, between two computing nodes.

The Facade process relies on  $M_A$ ,  $S_{A1}$  and  $S_{A2}$ . As we expected, the Facade object crashes with a “**segmentation fault**” message in case of an **access,**

read, or write denied. The corruption of  $M_A$  does not compromise both the behavior of the Facade and data transmission (i.e., each subsequent DDS write invocation correctly succeeds). Furthermore, the improper modification of the shared memory content is notified with the following message:

*\*\_Transport\_Shmem\_attach\_writer: incompatible shared memory segment found. All applications using \* must use compatible shared memory protocols*". This warning message is triggered by the DDS library every 10 seconds and it is printed on the console of the Facade. We can conclude that failures of  $M_A$  do not necessarily compromise the mission of the ATC system. In this case, the library of the DDS\_2 encapsulates mitigation mechanisms that tolerate the corruption and notify the faulty state of the resource.  $S_{A1}$  and  $S_{A2}$  access/read denied, and corruption do not affect Facade operations. The write denied emulated on both the semaphores is notified with the following messages: *\* Mutex\_lock: OS semop() failure error OXD. \*\_send: !take semaphore*", and: *\* Mutex\_live: OS semctl() failure error OXD. \*\_send: !give semaphore.*", respectively. The warning messages are triggered by the DDS library every 10 seconds and are printed on the Facade console. We conclude that failures emulated on  $S_{A1}$  and  $S_{A2}$  do not compromise the mission of the ATC system. DDS\_2 tolerates and notifies emulated failures: thus, semaphores are not critical resources.

Processing Server uses  $M_B$ ,  $S_{B1}$  and  $S_{B2}$ . Failure emulation provides findings similar to the Facade. DDS\_2 tolerates, and occasionally notifies, emulated failures. Facade and processing Server communicate through  $N$ . When we emulate network unavailability with any of the proposed mechanisms, updated FPL instances are lost. However, both processes do not exhibit any explicit notification. Communication between the nodes is restored when the network is resumed.  $N$  unavailability compromises the mission of the system, however DDS\_2 is robust to transient failures of the network.

## 6 Design Implications and Lessons Learnt

Experiments show that the Facade is the most critical object in the ATC system. Failures of different types occurring in the Facade are not tolerated by the Client; furthermore, a crash of the Facade object compromises the mission of the system *as a whole*. On the other hand, the processing Server is not particularly critical. The proposed approach highlights that the system encapsulates error mitigation means to tolerate the failures occurring in the processing Server. As a result, most of the testing/validation efforts should be devoted to the Facade object. Alternatively, additional mitigation means might be included in the system to reduce the criticality of the Facade, considering the observed failure-modes. In general, *the overall safety level of a system depends on the nature of the dependencies among its components*. Thus, such dependencies should always be analyzed as showed. Indeed, the architecture of the system and design choices affect the ability of the system to react and to mitigate the failures.

The analysis of the two DDSs reveals that different implementations of the same service, i.e., the data distribution, affect the ability of tolerating failures. We observed that both DDSs use **shared memories**. In DDS\_1 they have a *critical* role. Communications occur via shared memory both at the Facade and Processing Server. In other words, it is a *single point of failure*, since each emulated failure compromises data transmission. DDS\_2 uses shared memories too. Anyway, in this case they provide only support facilities. Their corruption does not compromise data transmission. DDS\_1 does not use **semaphores**, thus avoiding the introduction of new potential failure sources. However, they do not represent an actual dependability threat in DDS\_2. Even if they are used to access resources, their failures do not compromise the service completion. Again, the same resource, e.g., the shared memory provided by the Linux OS, is critical for the first DDS implementation, but not for the other. This enforces that, in general, *distinct OTS implementations might result in different fault tolerance features with respect to the execution environment; these features have therefore to be assessed, in order to make convenient choices* (e.g., choosing a different OTS, or implement fault tolerance mechanisms).

Despite the different features of the DDS, the choice of a less dependable implementation does not affect the criticality levels of the components in the proposed case study. As discussed, a *state dependence* exists between the processing Server and the Facade. A state dependence might modify the criticality level of the writer entity, when it is not robust to the failures of the resource (see Table II). However, the Facade, i.e., the writer, is already more critical than the processing Server, thus the criticality of these two components does not change due to the dependence. We observe that, in general, *both (i) the fault tolerance capabilities of the OTS, and (ii) the criticality of the components of the system that use the OTS are worth to be assessed, since the choice of a specific OTS component depends on both these aspects*.

In the future, we will extend the integration approach to other domains and applications. We aim to compare in terms of dependability, other than different architectures, different execution environments. Moreover, we will extend the approach by considering different kinds of OS/middleware resources, as well as other failure-modes, in order to increase the accuracy of the experimentation. One of our goals is to investigate further failure emulation techniques to deal with software, coming from different suppliers, for which vendor does not share the code and does not provide practical support for the integration (i.e., a failure emulation completely transparent to the OTS internals). This will help establishing a common procedure, *independently* from the OTS suppliers.

**Acknowledgment.** This work has been partially supported by the project “CRITICAL Software Technology for an Evolutionary Partnership” (CRITICAL-STEP, <http://www.criticalstep.eu>), Marie Curie Industry-Academia Partnerships and Pathways (IAPP) number 230672, within the context of the EU Seventh Framework Programme (FP7) and by the Italian Ministry for Education, University, and Research (MIUR) in the framework of the Project of National Research Interest (PRIN) “DOTS-LCCI: Dependable Off-The-Shelf based middleware systems for Large-scale Complex Critical Infrastructures”.

## References

1. Hammet, R.: Flight-Critical Distributed Systems: Design Considerations. *IEEE AESS Systems Magazines*, 30–36 (2003)
2. Weyuker, E.J.: Testing Component-Based Software: A Cautionary Tale. *IEEE Software* 15(5), 54–59 (1998)
3. Moraes, R.L.O., Durães, J., Barbosa, R., Martins, E., Madeira, H.: Experimental Risk Assessment and Comparison Using Software
4. CENELEC: EN 50126 Railways Applications. The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS)
5. DO-178B/ED12B Software consideration in airborne systems and equipment certification. RTCA and EUROCAE (December 1992)
6. SAF.ET1.ST03.1000-MAN-01. Air Navigation System Safety Assessment Methodology (v2-0). EUROCONTROL EATMP Safety Management (April 2004)
7. Functional safety and IEC 61508. Functional safety of electrical/electronic/programmable electronic safety-related systems. Produced by IEC/SC65A/WG14, The working group responsible for guidance on IEC 61508 (September 2005)
8. Storey, N.: Safety-Critical Computer Systems. Pearson and Prentice Hall (1996)
9. Hassami, A.G., Foord, A.G.: Systems safety-a real example (European rail traffic management system, ERTMS). In: Proc. of the Second IEEE International Conference on Human Interfaces in Control Rooms, Cockpits and Command Centres, pp. 327–334 (2001)
10. Pasquale, T., Rosaria, E., Pietro, M., Antonio, O.: Hazard analysis of complex distributed railway systems. In: Proc. of the 22nd IEEE International Symposium on Reliable Distributed Systems (SRDS 2003), pp. 283–292 (October 2003)
11. Mana, P., De Redet, J.M., Fowler, D.: Assurance Levels for ATM elements: Human (HAL), Operational Procedure (PAL), Software (SWAL). In: Proc. of the 2nd IEEE Int. Conference on Institution of Engineering and Technology, pp. 13–19 (October 2007)
12. Garrett, C., Apostolakis, G.: Automated hazard analysis of digital control systems. *Reliability Engineering and System Safety* 77, 1–17 (2002)
13. Garrett, C., Guarro, S., Apostolakis, G.: The Dynamic Flowgraph Methodology for Assessing the Dependability of Embedded Software Systems. *IEEE Trans. on Syst., Man, and Cybern.* 25(5), 824–840 (1995)
14. Supakkul, S., Lawrence, C.: Applying a Goal-Oriented Method for Hazard Analysis: A Case Study. In: Proc. of the 4th International Conference on Software Engineering Research, Management and Applications (SERA 2006), pp. 22–30 (August 2006)
15. Hewett, R.: Assessment of Software Risks with Model-Based Reasoning. In: Proc. of IEEE Inter. Conf. on Systems, Man and Cybernetics, vol. 4, pp. 3238–3243 (2005)
16. Powell, D.: Failure Mode Assumptions and Assumption Coverage. In: Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing, FTCS 1992 (1992)
17. Pardo-Castellote, G.: OMG data-distribution service: Architectural overview. In: Proc. of the IEEE ICDCS Workshops, pp. 200–206 (2003)
18. Rubini, A., Corbet, J.: *Linux Device Drivers*, 2nd edn. O'Reilly, Sebastopol (2001)
19. Cotroneo, D., Pecchia, A., Pietrantuono, R., Russo, S.: A failure analysis of data distribution middleware in a mission-critical system for air traffic control. In: Proc. of the 4th ACM Int'l Workshop on Middleware for Service Oriented Computing, pp. 25–30 (2009)

# On the Use of Semantic Technologies to Model and Control Security, Privacy and Dependability in Complex Systems

Andrea Fiaschetti<sup>1,\*</sup>, Francesco Lavorato<sup>1</sup>, Vincenzo Suraci<sup>1</sup>,  
Andi Palo<sup>1</sup>, Andrea Tagliatela<sup>2</sup>, Andrea Morgagni<sup>3</sup>,  
Renato Baldelli<sup>3</sup>, and Francesco Flammini<sup>4</sup>

<sup>1</sup> University of Rome “La Sapienza” - Via Ariosto 25, 00185 Roma, Italy  
{fiaschetti, andi, suraci}@dis.uniroma1.it,  
francesco.lavorato@yahoo.it

<sup>2</sup> TRS SpA. - Via Circumvallazione Esterna - 80014 Giugliano in Campania (Napoli), Italy  
andrea.tagliatela@trs.it

<sup>3</sup> Elsag Datamat S.p.a. - Via Laurentina 760, 00143 Roma, Italy  
{andrea.morgagni, renato.baldelli}@elsagdatamat.com

<sup>4</sup> Ansaldo STS S.p.a. - Via Argine 425, 80147 Napoli, Italy  
francesco.flammini@ansaldo-sts.com

**Abstract.** In this paper a semantic approach is presented to model and control Security, Privacy and Dependability (SPD) in complex interconnected environment composed by heterogeneous Embedded Systems.

Usually, only the individual properties are locally considered to obtain desired functionalities and this could result in sub-optimal solutions. With the use of modern semantic technologies (like OWL or reasoning engines) it is possible to model not only the individual parameters but also the relations between the different (and dynamically changing) parts of the systems, thus providing enriched knowledge and more useful information that could feed control algorithms.

The model presented in this paper is based on the results obtained during the first phase of the pSHIELD<sup>1</sup> project (conceived and lead by Finmeccanica) and it is focused on a concrete application coming from a critical scenario in railway environment: the monitoring of freight trains transporting hazardous material.

**Keywords:** Ontology, Security, Privacy, Dependability, Model, Common Criteria.

## 1 Introduction

The diffusion of Embedded Technologies in everyday life has increased in recent years. Dozens of components surround us and the intrinsic modularity and connectivity

---

\* Manuscript received on March 21, 2011. Andrea Fiaschetti is the corresponding author (phone: +39-0677274039; fax: +39-0677274033).

<sup>1</sup> ARTEMIS Call 2009 – SP6100204.



capabilities allow them to compose together and to provide enriched and more complex services and platforms. The composition allows the creation, starting from elementary devices, of a more complex system with enhanced functionalities: but what about the Security, Dependability and Privacy of the resulting system? Current methodologies, like for example the Common Criteria approach [1], can easily address Security and Dependability assessment for each individual component or algorithm, but are not able to provide a global view of the complex systems starting from its elementary components: the system should be assessed as a whole ex-novo. This leads into a waste of resources (in terms of time and money) and excessive engineering effort to validate the integrated platforms.

For that reason, the pSHIELD project, funded by ARTEMIS (the Technology Platform on Embedded System), is proposing an innovative methodology to model and assess the Security, Privacy and Dependability properties of Embedded System by means of semantic technologies. Semantic should allow the abstraction of the system and its functionalities and the extrapolation of SPD relevant information that can be used by supervisors or controlled to perform action on the individual system's components; for that reason, this methodology could be very useful if applied to safety-critical applications, where the identification and control of Security and Dependability attributes is a challenging task.

The paper is structured as follows: in Section 2 a generic overview of standard semantic technology approach is provided; then in Section 3 the solution adopted for the pSHIELD project is presented as well as the methodology that has lead to its development. Finally Section 4 will explain how these technologies could be integrated in a real application scenario and finally in Section 5 conclusions are drawn.

## 2 Overview on Semantic Technologies

Since the purpose of this research is to propose a sound framework for an effective exploitation of semantic technologies in Embedded Systems context in general, and in particular for Security, Privacy and Dependability issues, we should provide at first a formal overview on the "State of the Art" methodology to build, verify and maintain ontology. This State of the Art methodology that we have chosen as reference comes out from recent valuable work performed by IASI - LEKS (Institute of Informatics and Systems Analysis - Laboratory for Enterprise Knowledge and Systems) in the scope of the ONTOMAN, SPEED, COOPER and SITMAR research projects, from which the following paragraphs are derived.

As a matter of fact, in order to make available large-scale, high quality domain ontology, effective and usable methodologies are needed to facilitate the process of ontology building. Ontology building is a task that pertains to the ontology engineers that we classify as knowledge engineers (KE) and domain experts (DE). Even though automatic ontology learning methods (such as text mining) significantly support ontology engineers, speeding up their task, there is still the need of a significant manual effort, in the integration and validation of the automatically generated ontology.

Existing ontology building methods only partly are built capitalizing the large experience that can be drawn from widely used standards in other areas, like software engineering and knowledge representation. For our purposes, we shall embrace a methodology for ontology building derived from a well-established and widely used software engineering process, the Unified Software Development Process.

This is a novel approach to large-scale ontology building that takes advantage of the Unified Process (UP) and the Unified Modeling Language (UML). This choice makes ontology building an easier task for modelers familiar with these techniques: each phase of the method fits in the UP, providing a number of consolidated steps that guide the process of ontology development. UML has been already shown to be suitable to this end, confirming its nature of rich and extensible language. What distinguishes the UP and the ontology building methodology from the other methodologies, respectively for software and ontology engineering, is their use-case driven, iterative and incremental nature.

The methodology is use-case driven since it does not aim at building generic domain ontology, but its goal is the production of ontology that serve its users, both humans and automated systems (e.g. semantic web services, intelligent agents, etc.), in a well defined application area. Use cases are the first diagrams that drive the exploration of the application area, at the beginning of the ontology building process.

The nature of the process is iterative since each interaction allows the designer to concentrate on part of the ontology being developed, but also incremental, since at each cycle the ontology is further detailed and extended.

Following the UP, in the methodology we have cycles, phases, iterations and workflows. Each cycle consists of four phases (inception, elaboration, construction and transition) and results in the release of a new version of the ontology. Each phase is further subdivided into iterations. During each iteration, five workflows (described in the next subsections) take place: **requirements, analysis, design, implementation and test**. Workflows and phases are *orthogonal* in that the contribution of each workflow to an iteration of a phase can be more or less significant: early phases are mostly concerned with establishing the requirements (identifying the domain, scoping the ontology, etc.), whereas later iterations result in additive increments that eventually bring to the final release of the ontology (Fig.1). Notice that, as illustrated in the figure, more than one iteration may be required to complete each of the four phases. This scheme follows faithfully the Unified Process.

The first iterations (inception phase) are mostly concerned with capturing requirements and partly performing some conceptual analysis. Neither implementation nor test is performed. During subsequent iterations (belonging to the elaboration phase) analysis is performed and the fundamental concepts are identified and loosely structured. This may require some design effort and it is also possible that the modelers provide a preliminary implementation in order to have a small skeletal blueprint of the ontology, but most of the design and implementation workflows pervade iterations in the construction phase. Here some additional analysis could be still required aiming at identifying concepts to be further added to the ontology. During the final iterations (transition phase), testing is heavily performed and the ontology is eventually released. In parallel, the material necessary to start the new cycle, that will produce the next version of the ontology, is collected. As shown in Fig. 1, and detailed in the next sections, at each iteration different workflows come

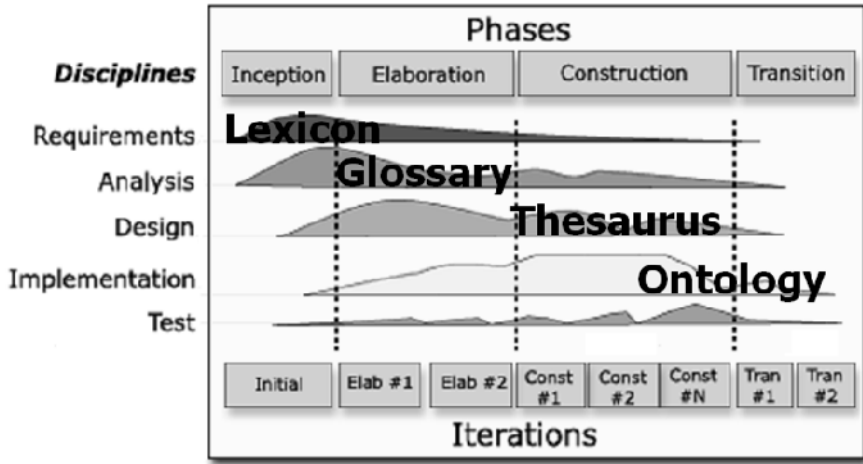


Fig. 1. Mapping onto workflows and phases of UP

into play and a richer and more complete version of the target ontology is produced. The incremental nature of the methodology requires first the identification of the relevant terms in the domain, gathered in a lexicon; then the latter is progressively enriched with definitions, yielding a glossary; adding to it the basic ontological relationships allows a thesaurus to be produced, until further enrichments and a final formalization produces the sought reference ontology.

In the following subsections each ontology building workflow is described in detail.

### 2.1 The Requirements Workflow

Requirements capture is the process of specifying the semantic needs and the knowledge to be encoded in the ontology. The essential purpose of this workflow is to reach an agreement between the modelers, the knowledge engineers, and the final users, represented by the domain experts. During the first meetings, knowledge engineers and domain experts establish the guidelines for building the ontology. The first goal is the identification of the objectives of the ontology users. To this end, it is necessary to:

- determining the domain of interest and the scope, and
- defining the purpose.

These objectives are achieved by:

- writing one or more storyboards
- creating an application lexicon
- identifying the competency questions, and
- the related use cases.

## 2.2 The Analysis Workflow

### 2.2.1 The Conceptual Analysis Consists of the Refinement and Structuring of the Ontology Requirements Identified in Previous Section

The ontological commitments derived from the definition of scope are extended, by reusing existing resources and through concept refinement. The application lexicon will be enriched through the definition of a more general domain lexicon, then definitions will be added to produce the *Reference Glossary*.

### 2.2.2 Considering Reuse of Existing Resources: Identification of the Domain Lexicon

The domain lexicon is defined as the terminology used in the domain of interest, extracted by analyzing a corpus of existing resources. The analysis is mainly based on external resources, such as documents, standards, glossaries, thesauri, legacy computational lexicons and available ontology. This task, like in the case of the application lexicon, can be supported by automatic tools. The description of this activity adheres to the view of linguistic ontology in which concepts, at least the lower and intermediate levels, are anchored to texts, i.e. they have a counterpart in natural language.

A statistical analysis shall be done in a corpus of documents of reference to identify frequently used terms to be included in the domain lexicon. The domain experts shall decide to include, in this lexicon, all the terms present in, for instance, at least two standards. Some other terms, present in only one resource, can be included after approval from a wider panel of experts. After this activity, the domain lexicon shall contain a number of terms (including synonyms).

### 2.2.3 Modeling the Application Scenario Using UML Diagrams

The goal of this activity is to model the application scenario and better specify the Use Case Diagrams, drawn in the requirement workflow, with the aid of Activity and Class Diagrams. UML diagrams represent a model of the application and will be used for the validation of the ontology. In principle, all the classes, actors, and activities modeled in UML must have a corresponding concept in the ontology.

### 2.2.4 Building the Glossary

A first version of a glossary of the domain of interest has to be built merging the application lexicon and the domain lexicon. During the merge of the two lexicons we can organize all the concepts in two major areas: the intersection area and the disjoint area. Then we use the following “inclusion policy”: the glossary should include all the concepts coming from the intersection area and, after the domain experts approval, some concept belonging to the disjoint area. The output is a reference lexicon that will grow into a glossary by associating one or more definitions to each term. The definitions should be selected from knowledgeable sources and agreed among domain experts.

## 2.3 The Design Workflow

The main goal of this workflow is to give an ontological structure to the set of terms gathered in the Glossary. The refinement of entities, actors and processes identified in

the analysis workflow, as well as the identification of their relationships, is performed during the design workflow.

### 2.3.1 Categorizing the Concepts

Each concept is categorized by associating a “kind” to it. Such *kinds* should include the major ontological categories, according to proposals of *upper ontology*, or *meta-ontology*,

### 2.3.2 Refining the Concepts and Their Relations

At this stage, concepts are organized by introducing formal relations among them. Between sets of synonyms identified in the previous phase. A first step consists in organizing the concepts in a taxonomic hierarchy through the generalization (i.e., kind-of or is-a) relation. To this end, three main approaches are known in the literature:

- top-down (from general to particular)
- bottom-up (from particular to general)
- middle-out (or combined), which consists in finding the salient concepts and then generalizing and specializing them. This approach is considered to be the most effective because concepts “in the middle” tend to be more informative about the domain.

The resulting taxonomy can be extended with other relations, i.e., part-of and association. The outcome of this step is Thesaurus, structured according the UML class diagram relations: generalization (IsA), aggregation (Part-Of) and association. In parallel, the actual UML diagrams can be built.

## 2.4 The Implementation Workflow

The purpose of this workflow is to perform the final building step, by formalize defining the actual ontology in a formal language. The structure of the ontology will be the one given in the enriched Thesaurus, but here the different elements will be formally represented. To this end, the Ontology Web Language (OWL) proposed by the W3C shall be adopted.

The outcome of this workflow is the implementation model, i.e., a reference ontology encoded in OWL.

## 2.5 TheTest Workflow

The test workflow allows to verify that the ontology correctly implements the requirements produced in the first workflow. We envisage two kinds of test.

The first concerns the coverage of the ontology with respect to the application domain. In particular, the domain experts are asked to semantically annotate the UML diagrams, representing the application scenario, with the ontology concepts. (This test is particularly relevant for ontology used in ontology-based reconciliation of messages).

The second kind of test concerns the competency questions and the possibility to answer them by using concepts in the ontology. Such questions will trigger a traversal

of the ontology that will produce proper concepts. Competency questions represent a good test for ontology to be used in search and discovery of resources.

### 3 pSHIELD Methodology

The described methodology is the most valuable chain to produce ontology and meta-models for a specific scenario. The novelty proposed in this paper is one step before of this procedure and is about the choice of the “domains” and “scenarios” that will feed the five step procedure.

So the problem is: given a clear procedure on how to build ontology, what are we supposed to describe in it? Starting from that, we can affirm that the context is the one of Embedded Systems; in particular the more specific contest are the SPD functionalities provided by their interaction/composition.

The main objective of our approach with semantic models are:

1. the *abstraction* of the *real word* from a technology-dependent perspective into a technology-independent representation.
2. the *representation* of *functional properties* by means of ontology as well
3. the identification of the relations between real/structural and functional world.

So, as depicted in Fig. 2, the problem of modeling SPD in the context of ES is reduced to the formulation of three different meta-models describing: i) structure, ii) functions, iii) a relations between structure and functions.

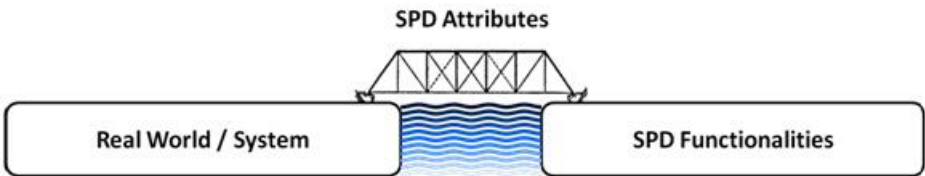


Fig. 2. Proposed approach to model SPD for ES

The bridge has been built thanks to the introduction of a third metamodel taking into account the atomic attributes that are impacted in this context and to map them in these two worlds. This concept is depicted in picture Fig. 2.

For each of the three models, a justification is provided below:

#### 3.1 Structural Ontology

The structural ontology (on the left side of the picture) is the easier to model, because it is a simple description of the Embedded System component. It contains the hardware components and basic functionalities provided by the individual element and the related attribute, all in an SPD relevant environment. For example a node, in a first simplification, is composed by a memory a CPU, a battery and a transmission

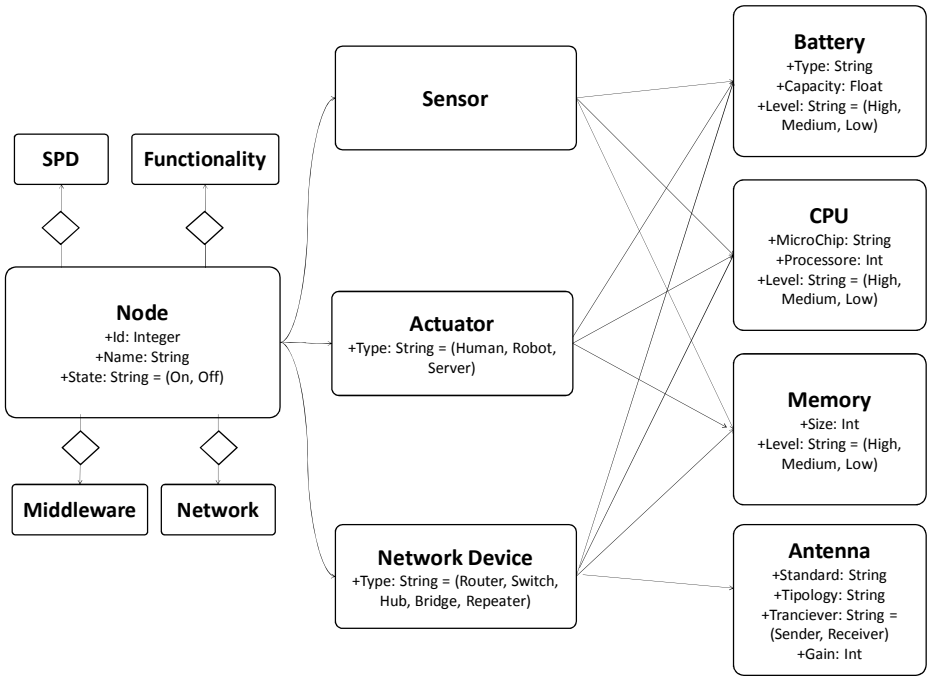


Fig. 3. Structural Ontology for a node

antenna; furthermore the CPU is characterized by the frequency and bit length and, in SPD relevant context, the possibility of performing hardware cryptography. By doing so, all the components constituting a complex system can be represented. In Fig. 3 an example of Node model is provided.

### 3.2 Functional Ontology

In order to identify the functionalities provided by the systems components, a preminent approach based on the Security Certification Process (Common Criteria [1]) has been chosen. This approach is based on the five-steps reported in the following:

- **Step 1 – What is the overall system?**

The overall System is a set of interacting and interconnected Embedded Systems with specific composability and SPF Functionalities.

- **Step 2 – What is the role of the System?**

The system, in SPD relevant context, should assure SPD for a certain asset or goods in a specific scenario. In the following, for convenience, we will replace the expression “assure SPD” with the generic expression “protect”, even if its real meaning is different. In Fig. 4 this concept is represented: the green boxes represent the interconnected ESs and the gray box is the addressed asset/goods (the SPD functionalities are still missing from this graphical representation because they are identified in the following steps).

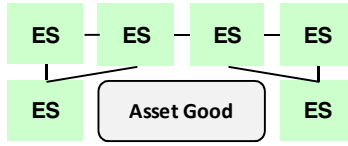


Fig. 4. Step 1 and 2 representation

• **Step 3 – What are the asset/goods and the scenario addressed by the System?**

The selected scenario (for the purpose of this work) is railways transportation and the asset/goods is the secure and dependable monitoring of freight trains transporting hazardous materials. Moreover, since the System should protect itself, it is an asset/good as well.

• **Step 4 – What are the possible menace and/or attack that could affect the protected asset/goods.**

Once the assets and goods, as well as the application scenarios are clearly identified, it is easy to enumerate all the possible menaces and attacks that could affect the level of Security, Privacy and Dependability of the system. The output of this activity is a fundamental input for next step. The logical step is given in Fig. 5.

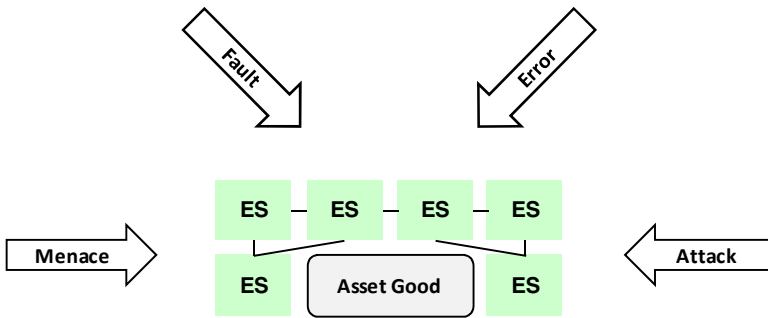


Fig. 5. System attacks and menaces

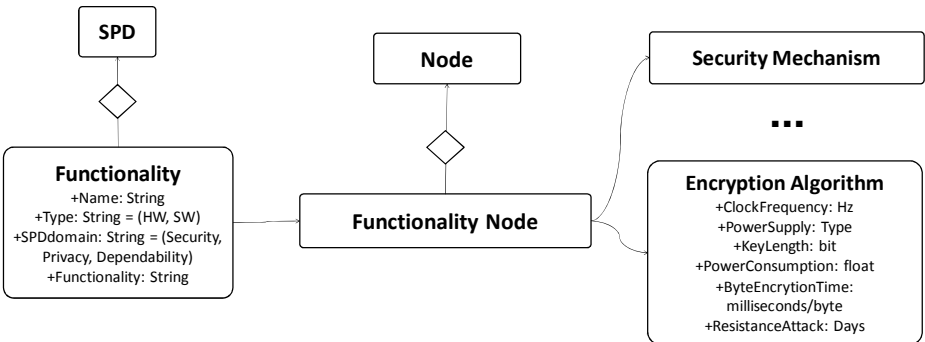


Fig. 6. Functional Ontology



- **Step 5 – What are the SPD functionalities that can prevent or minimize the effect of the previously identified menace and/or attack?**

Starting from the identified menaces and attacks, a set of SPD Functionalities is identified that are able to prevent or mitigate them. The functionalities are the ones that we have to represent in our SPD relevant ontology. An example is provided in Fig. 6. Of course, the node model has relation with the functional ontology.

### 3.2 SPD Ontology

The last model is given by the SPD attributes that allow the link between the structural word and the functional word. This is the most simple and, at the same time, significant ontology. For the purpose of our work we have choose to describe all Dependability, Security (and Privacy) issues by means of six attributes: *availability*, *reliability*, *safety*, *confidentiality*, *integrity*, *maintainability* (see Fig. 7).

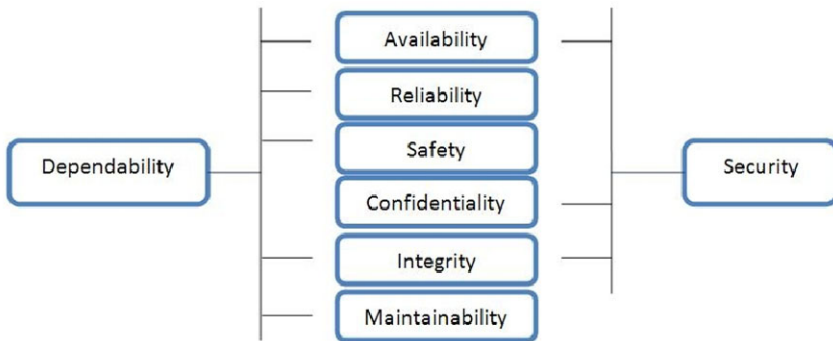


Fig. 7. SPD attributes

Each structural and functional component is associated to one of more of the following attributes: for example the cryptography could be related to Confidentiality and the Node's CPU is related to cryptography. Then we know that if we want to address the confidentiality issues of our system we have to take into account the cryptographic algorithms as well as the CPU characteristics.

The example is trivial, but in more complex system the advantage is disruptive: we have an high level representation by means of ontology that, thanks to inferential engines, can provide us a list of all relations and components relevant to our SPD issues and objective.

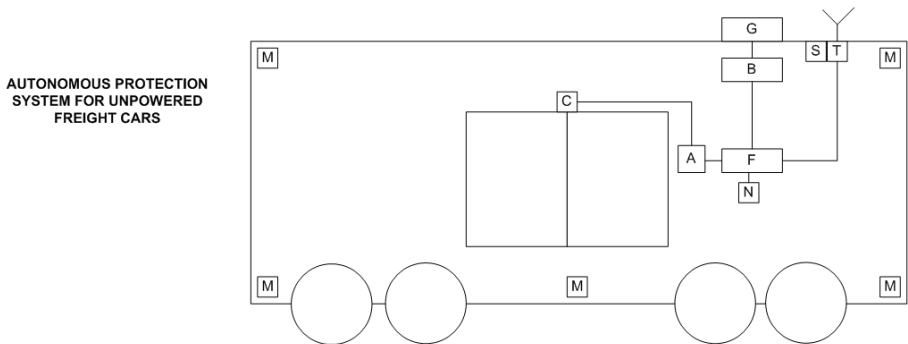
## 4 Application Scenario

This approach is currently under investigation in the pSHIELD project, where the considered system is a monitoring platform for wagons transporting hazardous materials (see Fig. 8).

The procedure could be exploited easily as follows:

1. The components equipped in the wagon are modeled with the structural ontology, with their specification and supported functionalities
2. The SPD functionalities are quickly given by the scenario: dependable monitoring of goods, confidentiality in collected data, reliability and integrity of transmission of information.
3. A one-by-one mapping with these functions and the components (sensors, GPS receiver, GSM antenna, battery pack, ...) is a trivial task.

The technology chosen to represent the ontology is OWL DL ([6][7]), since it is the best compromise between expressiveness and performance. Then obtained models can be put in the central control unit of the wagon and, both offline and online, inferential engine (like Jena or Pellet) will monitor the status of the components involved in the SPD tasks, providing to the operator only a small, but relevant, subset of information. A further step is the automatic information analysis, so that the system is able to discriminate, without human intervention, whether an event is relevant for SPD purposes or not and then take the adequate corrective actions. But this is out of scope of this work and will be at the bases of future research.



**Legend**

- A: Access control device powered on-demand (e.g. numeric keypad featuring "ON" button)  
 B: Long lasting battery pack  
 C: Magnetic contact or other very-low consumption intrusion detection device  
 F: FPGA-based central control unit featuring embedded CPU and OS  
 G: Low size power generator (e.g. eolic, solar panel, piezoelectric pad) - optional  
 M: Self-powered smart wireless sensor measuring vibrations, temperature, humidity, light, sound  
 N: Gateway node for the wireless sensor network  
 S: Satellite positioning antenna  
 T: Wide area wireless transceiver (GSM, GPRS, UMTS, EDGE)

**Basic working logic**

**Case of abnormal environmental or operational conditions:**

Whenever an abnormal event is detected by M, its transmission unit is activated and data is received by N. F validates data and if the anomaly is confirmed it activates S to achieve the current position and T to send an appropriate warning message to the control center.

**Case of intrusions**

Whenever C detects an opening while the alarm system is activated (by A), F activates S to achieve the current position and T to send an appropriate alarm message to the control center.

**Fig. 8.** Application Scenario

## 5 Conclusion and Future Works

In this paper a methodology to develop semantic representation of Embedded System and SPD functionalities has been provided. This methodology consists of three steps, each one in charge of producing a single ontology: the structural one, the functional one and the SPD attributes. The decoupling of these context eases the development of the meta-model but, at the same time, the presence of common attributes allows to easily re-establish relations between components, functions and SPD issues.

Following a well consolidated process, starting from the presented methodology the three OWL files are produced and can be used in a plenty of applications, especially in the design (and running) phases of safety critical systems. The design phase, for example, allows a quicker certification of system's components, because the description is standardized and clear, while in the running phase it is possible, by means of human or automatic reaction, to control the system evolution towards specific objectives.

The next step is to develop the instances of the OWL file and to store them in the scenario's component and to perform simulations to check possibilities and performance of a system enabled with semantic capabilities. Then the methodology can be applied in other context, like the ones foreseen by the follow up of the pSHIELD project (named nSHIELD), the most challenging being the assurance of SPD in avionic context (UAV control and navigation payload).

**Acknowledgments.** The research is performed in the context of the pSHIELD ARTEMIS project. Some of the work performed for the evaluation of semantic technologies has been carried out in the scope of the EU-FP7 TASS Project.

An acknowledgement goes to the IASI LEKS for having enriched the literature and the state of the art with a valuable procedure.

## References

- [1] Common Criteria for Information Technology Security Evaluation, v3.1 (July 2009)
- [2] Ding, L., Kolari, P., Ding, Z., Avancha, S.: Using Ontologies in the Semantic Web: A Survey. *Ontologies* 14, 79–113 (2007)
- [3] Mascardi, V., Locoro, A., Rosso, P.: Automatic Ontology Matching via Upper Ontologies: A Systematic Evaluation. *IEEE Transactions on Knowledge and Data Engineering* 22(5), 609–623 (2010)
- [4] Henkel, J., Narayanan, V., Parameswaran, S., Ragel, R.: Security and Dependability of Embedded Systems: A Computer Architects' Perspective". In: 22nd International Conference on VLSI Design 2009, pp. 30–33 (2009), doi:10.1109/VLSI.Design.2009.114
- [5] pSHIELD Technical Annex (June 2010)
- [6] Dean, M., Schreiber, G.: OWL Web Ontology Language Reference, <http://www.w3.org/TR/2004/REC-owl-ref-20040210/>
- [7] McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language Overview, <http://www.w3.org/TR/owl-features/>

- [8] Compton, M., Henson, C., Lefort, L., Neuhaus, H., Sheth, A.: A Survey of the Semantic Specication of Sensors. In: Proc. Semantic Sensor Networks 2009, pp. 17–32 (2009)
- [9] Gaines, B., Shaw, M.: Integrated knowledge acquisition architectures. *Journal of Intelligent Information Systems* 1(1), 9–34 (1992)
- [10] Grosso, E., Eriksson, H., Fergerson, R., Tu, S., Musen, M.: Knowledge modeling at the millennium — the design and evolution of Protégé-2000. In: Proceedings of KAW 1999, Banff, Canada (1999)
- [11] Staab, S., Schnurr, H.-P., Studer, R., Sure, Y.: Knowledge processes and ontologies. *IEEE Intelligent Systems* 16(1) (2001)

# Author Index

- Alexander, Rob 185  
Aliee, Hananeh 324  
Aniello, Leonardo 310  
Armengaud, Eric 57, 113  
Assayad, Ismail 437
- Baldelli, Renato 467  
Baldoni, Roberto 310  
Baufreton, Philippe 57  
Bernardi, Simona 15  
Bieber, Pierre 43  
Blanquart, Jean-Paul 57  
Bondavalli, Andrea 128  
Bonichon, Richard 85  
Bonifacio, Giuseppe 369  
Bourrouilh, Quentin 57, 113  
Bovenzi, Antonio 128  
Brancati, Francesco 128  
Bürklen, Susanne 29
- Cameron, Neil 228  
Canet, Géraud 85  
Carnevali, Laura 409  
Clegg, Kester 185  
Coppolino, Luigi 143, 199  
Correnson, Loïc 85  
Cotroneo, Domenico 213  
Cruciani, Federico 355
- D’Acierno, Luca 171  
D’Antonio, Salvatore 143, 199  
Delmas, Rémi 43  
Di Leo, Domenico 213  
Di Luna, Giuseppe A. 310
- Elia, Ivano Alessandro 143
- Fabbrini, Fabrizio 284  
Fantechi, Alessandro 383  
Felici, Massimo 99  
Fiaschetti, Andrea 467  
Fisher, Michael 228  
Flammini, Francesco 15, 467  
Formicola, Valerio 199  
Fusani, Mario 284
- Girault, Alain 437  
Gnesi, Stefania 383  
Goubault, Eric 85  
Griessnig, Gerhard 57  
Grießnig, Gerhard 113  
Güdemann, Matthias 423
- Haucourt, Emmanuel 85  
Hawkins, Richard 185  
Herzner, Wolfgang 270  
Hillebrand, Joachim 257  
Hirschowitz, Michel 85  
Höflinger, Jens 29
- Jöbstl, Elisabeth 270  
Joshi, Kaustubh R. 338  
Jump, Mike 228
- Kaâniche, Mohamed 157  
Kalla, Hamoudi 437  
Kang, Eun-Young 243  
Kanoun, Karama 157  
Kawato, Masahiro 296  
Kelly, Tim 185  
Knoop, Michael 29  
Krammer, Martin 57  
Kreiner, Christian 113  
Kuntz, Matthias 71
- Labbé, Sébastien 85  
Lamberti, Immacolata 171  
Lami, Giuseppe 284  
Laurent, Odile 57  
Lavorato, Francesco 467  
Lee, Dong-Ah 397  
Lee, Jang-Soo 397  
Leitner, Andrea 113  
Leitner-Fischer, Florian 71  
Leue, Stefan 71  
Lodi, Giorgia 310
- Machrouh, Joseph 57  
Mader, Roland 113  
Maeno, Yoshiharu 296  
Mandic, Irenka 257  
Manno, Gabriele 1

- Marmo, Pietro 171, 369  
 Marrone, Stefano 15  
 Mazzeo, Antonino 171  
 Mazzocca, Nicola 171  
 Meduri, Valentino 99  
 Merseguer, José 15  
 Mimram, Samuel 85  
 Montella, Bruno 171  
 Morgagni, Andrea 467  
  
 Nardone, Roberto 171  
 Natella, Roberto 213  
  
 Orazzo, Antonio 369  
 Ortmeier, Frank 423  
  
 Palo, Andi 467  
 Papa, Camilla 15  
 Papadopoulos, Chris 157  
 Pecchia, Antonio 452  
 Peer, Christian 257  
 Peikenkamp, Thomas 57  
 Petrone, Ida 369  
 Pettersson, Paul 243  
 Pietrantuono, Roberto 213, 452  
 Popov, Peter 1  
 Punzo, Vincenzo 171  
  
 Quaglietta, Egidio 171  
  
 Reichenpfader, Peter 257  
 Ridi, Lorenzo 409  
 Romano, Luigi 143, 199  
 Russo, Stefano 128, 452  
  
 Sanders, William H. 338  
 Schindler, Cecile 57  
 Schlick, Rupert 270  
 Schobbens, Pierre-Yves 243  
 Seguin, Christel 43, 157  
 Siegl, Hannes 257  
 Solhaug, Bjørnar 99  
 Steger, Christian 113  
 Suraci, Vincenzo 467  
  
 Tadano, Kumiko 296  
 Tagliatela, Andrea 467  
 Tedeschi, Alessandra 99  
 Tiassou, Kossi 157  
 Trapp, Mario 29  
  
 Velardi, Luigi 369  
 Venticinque, Alessio 369  
 Vicario, Enrico 355, 409  
 Vittorini, Valeria 15  
  
 Webster, Matt 228  
 Weiß, Reinhold 113  
 Wien, Tormod 57  
  
 Xiang, Jiangwen 296  
  
 Yoo, Junbeom 397  
  
 Zarandi, Hamid Reza 324  
 Zimmer, Bastian 29  
 Zonouz, Saman Aliari 338