

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

Erwin Schoitsch (Ed.)

Computer Safety, Reliability, and Security

29th International Conference, SAFECOMP 2010
Vienna, Austria, September 14-17, 2010
Proceedings

Volume Editor

Erwin Schoitsch

AIT Austrian Institute of Technology GmbH, Safety and Security Department

Donau-City-Str. 1, 1220 Vienna, Austria

E-mail: erwin.schoitsch@ait.ac.at

Library of Congress Control Number: 2010933788

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

ISSN 0302-9743

ISBN-10 3-642-15650-9 Springer Berlin Heidelberg New York

ISBN-13 978-3-642-15650-2 Springer Berlin Heidelberg New York

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.

springer.com

© Springer-Verlag Berlin Heidelberg 2010

Printed in Germany

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

Preface

Computers and microprocessors are indispensable in modern technical systems, their deployment spanning the domains automotive, railway, aerospace, and transportation, security, energy supply, telecommunication, critical infrastructures and process industries. They perform tasks that a few decades ago were very difficult if not impossible. As they perform these tasks with increasing efficiency, more and more tasks are shifted from hardware to software, which means that the dependability of computer systems becomes crucial for the safety, security and reliability of technical systems.

With the so-called “embedded systems” (becoming more and more intelligent, networked and co-operating with each other, with humans and the environment) computers have invaded all aspects of daily life. New paradigms have arisen, like ubiquitous computing, systems-of-systems, energy and resource awareness, enormous complexity issues and the like, requiring a more holistic systems view as well.

So, after 31 years of SAFECOMP, the emphasis of the 29th event is on critical embedded systems, which are almost omnipresent. Their impact on our lives, risks and challenges are often not well understood (underestimated or exaggerated). The primary issue is to cope with complexity, new failure modes and resource management, due to shrinking feature size, multi-core systems and management of multiple variants, while maintaining dependability properties and robustness.

This book addresses system analysis, safety cases and certification, fault tolerance and error detection, validation and verification, testing, the interplay and interdependencies of security and safety, and various aspects of safety engineering. Many domains and applications are covered, including automotive, aerospace and critical infrastructures (smart grid). The book reproduces the 34 papers from 12 countries presented at the 29th International Conference on Computer Safety, Reliability and Security, SAFECOMP 2010, held in Vienna, Austria, September 14–17, 2010. SAFECOMP 2010 continued the successful series of SAFECOMP conferences, started by EWICS TC7, Computer Safety, Reliability and Security, in 1979.

Therefore, this international conference reflects not only the state of the art, experience and new trends in the technology of dependable systems, but also the increasing strength of Europe in this area, in competition with the USA and Japan, which is due to the efforts of the European Commission, the EU member and associated states, academia and industry. Besides the Framework Programmes, the EC has created European Technology Platforms and industry-driven Joint Technology Initiatives like ARTEMIS (Advanced Research and Technology for Embedded Intelligence and Systems) and ENIAC (European Nanoelectronics Initiative Advisory Council) being most relevant for progress in the areas addressed by embedded systems. Austria, for example, has its own supporting funding programmes like FIT-IT for embedded systems, and is a member of the European Joint Technology Initiatives, under the auspices of the Federal Ministry for Transport, Innovation and Technology, managed by FFG, the Federal Funding Agency. In Germany, UK, Italy and the other European

countries represented in the papers and SAFECOMP committees, we will find similar conditions. Many papers present work from European research projects. SAFECOMP 2010 takes into account the work done in EWICS TC7, a group of experts from Europe (and some overseas) from all major sectors of industry, research, assessment organizations and authorities.

As Chairman of the International Programme Committee (IPC) I would like to thank all authors who submitted their work, the keynote speakers for their valuable contribution, all other speakers, the members of the IPC and the Local Organizing Committee (Austrian Computer Society - OCG), the session chairmen and the sponsors, co-sponsors and exhibitors, for their efforts and support. I would also like to thank the Austrian Federal Ministry for Transport, Innovation and Technology, the City of Vienna and AIT, the Austrian Institute of Technology, for their contribution to this conference.

I hope that all participants benefited from SAFECOMP 2010 and enjoyed the conference and accompanying programme, that they as well as the readers of the proceedings, will join us at future SAFECOMP conferences!

July 2010

Erwin Schoitsch
Gerald Futschek

Organization

Conference Chair

Erwin Schoitsch

AIT, Austria

EWICS Chair

Francesca Saglietti

University of Erlangen, Germany

Local Chair

Gerald Futschek

OCG and Vienna University of Technology,
Austria

Local Organizing Committee (OCG, Austria)

Eugen Mühlvenzl
Elisabeth Waldbauer
Karin Hiebler
Christine Haas

International Programme Committee

Anderson, S. (UK)	Genser, R. (Austria)	Kelly, T. (UK)
Anderson, T. (UK)	Gerstinger, A. (Austria)	Knight, J.C. (USA)
Ata, B. (Turkey)	Gorski, J. (Poland)	Koomneef, F. (The Netherlands)
Bloomfield, R. (UK)	Gran, B.A. (Norway)	Kopacek, P. (Austria)
Bologna, S. (Italy)	Halang, W. (Germany)	Ladkin, P. (Germany)
Braband, J. (Germany)	Heisel, M. (Germany)	Lehmann, T. (Germany)
Buth, B. (Germany)	Haxthausen, A. (Denmark)	Lindskov Hansen, S. (Denmark)
Daniel, P. (UK)	Heitmeyer, C. (USA)	Littlewood, B. (UK)
Ehrenberger, W. (Germany)	Hübner, M. (Germany)	McDermid, J. (UK)
Emmet, L. (UK)	Johnson, C. (UK)	Nordland, O. (NO)
Felici, M. (UK)	Kaâniche, M. (France)	Pareigis, S. (Germany)
Flammini, F. (Italy)	Kanoun, K. (France)	Peleska, J. (Germany)

Pfeiffenberger, T. (Austria)	Schulze, S.-O. (Germany)	van der Meulen, M. (The Netherlands)
Pozzi, S. (Italy)	Seyfarth, T. (Germany)	Weinert, A. (Germany)
Rabe, G. (Germany)	Skavhaug, A. (Norway)	Wittmann, S. (Belgium)
Reich, S. (Austria)	Strigini, L. (UK)	Yüceer, R.E. (Turkey)
Saglietti, F. (Germany)	Sujan, M. (UK)	Zurakowski, Z. (Poland)
Schedl, G. (Austria)	Traverse, P. (France)	
Schoitsch, E. (Austria)	Trienekens, J. (The Netherlands)	

Scientific Sponsors and Co-sponsors

EWICS TC7, European Workshop on Industrial Computer Systems, Technical Committee 7, Safety, Reliability and Security
AIT Austrian Institute of Technology
OCG, Austrian Computer Society

AARIT, Austrian Association for Research in IT
ARTEMIS Austria, Advanced Research and Technology for Embedded Intelligence and Systems
ENCRESS, European Network of Clubs for Reliability and Safety of Software and Systems
ERCIM, European Research Consortium for Informatics and Mathematics
GfSE, System Engineering Association
GI, Gesellschaft für Informatik
IFAC, International Federation of Automatic Control
IFIP, International Federation for Information Processing
OVE, Austrian Electrotechnical Association

City of Vienna
Austrian Federal Ministry of Transport, Innovation and Technology

Table of Contents

Session 1

System Analysis

Reliability Analysis of Safety-Related Communication Architectures <i>Oliver Schulz and Jan Peleska</i>	1
A Novel HAZOP Study Approach in the RAMS Analysis of a Therapeutic Robot for Disabled Children <i>Petr Böhm and Thomas Gruber</i>	15
Variability Management of Safety and Reliability Models: An Intermediate Model towards Systematic Reuse of Component Fault Trees <i>Carolina Gómez, Peter Liggesmeyer, and Ariane Sutor</i>	28
QoS Analysis of Weighted Multi-state Probabilistic Networks via Decision Diagrams <i>Roberta Terruggia and Andrea Bobbio</i>	41

Session 2

Safety Cases and Certification

Comparison between IEC 60880 and IEC 61508 for Certification Purposes in the Nuclear Domain <i>Jussi Lahtinen, Mika Johansson, Jukka Ranta, Hannu Harju, and Risto Nevalainen</i>	55
Deriving Safety Cases for Hierarchical Structure in Model-Based Development <i>Nurlida Basir, Ewen Denney, and Bernd Fischer</i>	68
Assurance of Automotive Safety – A Safety Case Approach <i>Robert Palín and Ibrahim Habli</i>	82
How to “Survive” a Safety Case According to ISO 26262 <i>Torsten Dittel and Hans-Jörg Argus</i>	97

Session 3

Aerospace

Benchmarking Software Requirements Documentation for Space Application	112
<i>Paulo C. V́eras, Emilia Villani, Ana Maria Ambrósio, Rodrigo P. Pontes, Marco Vieira, and Henrique Madeira</i>	
Verifying Mode Consistency for On-Board Satellite Software	126
<i>Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky, Kimmo Varpaaniemi, Pauli Väisänen, Dubravka Ilic, and Timo Latvala</i>	
Computational Concerns in the Integration of Unmanned Airborne Systems into Controlled Airspace	142
<i>Christopher W. Johnson</i>	

Session 4

Error Detection

Residual Error Probability of Embedded CRC by Stochastic Automata	155
<i>Frank Schiller and Tina Mattes</i>	
ANB- and ANBDMem-Encoding: Detecting Hardware Errors in Software	169
<i>Ute Schiffl, André Schmitt, Martin Süßkraut, and Christof Fetzer</i>	

Session 5

Validation and Verification

Field Test Methods for a Co-operative Integrated Traffic Management System	183
<i>Thomas Gruber, Egbert Althammer, and Erwin Schoitsch</i>	
100% Coverage for Safety-Critical Software – Efficient Testing by Static Analysis	196
<i>Daniel Kästner, Reinhold Heckmann, and Christian Ferdinand</i>	
MODIFI: A MODEL-Implemented Fault Injection Tool	210
<i>Rickard Svenningsson, Jonny Vinter, Henrik Eriksson, and Martin Törngren</i>	

Automated Test Coverage Measurement for Reactor Protection System Software Implemented in Function Block Diagram	223
<i>Eunkyoung Jee, Suin Kim, Sungdeok Cha, and Insup Lee</i>	

Session 6

Testing

Overcoming Non-determinism in Testing Smart Devices: A Case Study	237
<i>Peter Bishop and Lukasz Cyra</i>	
Software Testing by People with Autism	251
<i>Suzanne Haanappel and Sjaak Brinkkemper</i>	

Session 7

Critical Infrastructure - Smart Grid

Information Flow Analysis of Energy Management in a Smart Grid	263
<i>Ravi Akella and Bruce M. McMillin</i>	
Integrated Cyber-Physical Fault Injection for Reliability Analysis of the Smart Grid	277
<i>Ayman Faza, Sahra Sedigh, and Bruce McMillin</i>	
A Metrics for Measuring the Strength of Inter-dependencies	291
<i>Silvia Ruzzante, Elisa Castorini, Elena Marchei, and Vincenzo Fioriti</i>	

Session 8

Security and Safety

Security Analysis of Open Building Automation Systems	303
<i>Wolfgang Granzer and Wolfgang Kastner</i>	
A UML Profile for Requirements Analysis of Dependable Software	317
<i>Denis Hatebur and Maritta Heisel</i>	

Session 9

Safety Engineering (1)

Model-Based Safety Engineering of Interdependent Functions in Automotive Vehicles Using EAST-ADL2	332
<i>Anders Sandberg, DeJiu Chen, Henrik Lönn, Rolf Johansson, Lei Feng, Martin Törngren, Sandra Torchiaro, Ramin Tavakoli-Kolagari, and Andreas Abele</i>	

Experiences in Applying Formal Verification in Robotics	347
<i>Dennis Walter, Holger Täubig, and Christoph Lüth</i>	
Evolving a Safe System Design Iteratively	361
<i>Alexandre Mota, Joabe Jesus, Adriano Gomes, Felipe Ferri, and Edson Watanabe</i>	
An Approach to Using Non Safety-Assured Programmable Components in Modest Integrity Systems	375
<i>Peter Bishop, Kostas Tourlas, and Nick Chozos</i>	

Session 10

Safety Engineering (2)

Development of High-Integrity Software Product Lines Using Model Transformation	389
<i>Stuart Hutchesson and John McDermid</i>	
On the Safety Implications of E-Governance: Assessing the Hazards of Enterprise Information Architectures in Safety-Critical Applications . . .	402
<i>Christopher W. Johnson and Stefan Raue</i>	
The Right Degree of Configurability for Safety-Critical Embedded Software in Variable Message Signs	418
<i>Thomas Novak and Christoph Stoegerer</i>	
INDEXYS, a Logical Step beyond GENESYS: INDustrial EXploitation of the genesYS cross-domain architecture	431
<i>Andreas Eckel, Paul Milbredt, Zaid Al-Ars, Stefan Schneelee, Bart Vermeulen, György Csertán, Christoph Scheerer, Neeraj Suri, Abdelmajid Khelil, Gerhard Fohler, Roman Obermaisser, and Christian Fidi</i>	

Session 11

System Modelling and Fault Tolerance

Integrating System Modelling with Safety Activities	452
<i>Bernhard Kaiser, Vanessa Klaas, Stefan Schulz, Christian Herbst, and Peter Lascych</i>	
Aspect-Oriented Implementation of Fault Tolerance: An Assessment of Overhead	466
<i>Ruben Alexandersson, Peter Öhman, and Johan Karlsson</i>	

Invited Talks (Keynote Abstracts)

System of Systems Challenges	480
<i>Hermann Kopetz</i>	
Murphy Was an Optimist	481
<i>Kevin R. Driscoll</i>	
Process Control Security: Go Dutch! (United, Shared, Lean and Mean)	483
<i>Eric Luijff</i>	
Author Index	485

Reliability Analysis of Safety-Related Communication Architectures

Oliver Schulz and Jan Peleska

University of Bremen, 28359 Bremen, Germany
{oschulz,jp}@informatik.uni-bremen.de
<http://www.informatik.uni-bremen.de/agbs>

Abstract. In this paper we describe a novel concept for reliability analysis of communication architectures in safety-critical systems. This concept has been motivated by applications in the railway control systems domain, where transitions into stable safe state are usually considered as undesired events because they cause a severe deterioration of the service reliability expected by end users. We introduce a domain-specific language for modelling communication architectures, the protocols involved and the fault hypotheses about anticipated deviations of communication channels and possibly other components from expected behaviour. From such model, a generator creates mutant models associated with probability formulae expressing each mutant’s probability of occurrence. Each mutant is analysed with respect to its unreliability, that is, whether it contains paths leading into stable safe state. Then the system reliability can be conservatively estimated by calculating an upper bound of the probability for the system to perform a transition into stable safe state within a given operational period. Our approach deliberately refrains from utilising probabilistic model checking, in order to avoid the state space explosions typically occurring when considering all possible erroneous behaviours within a single model. Instead, we analyse many different models, each only containing a restricted variant of deviations, which leads to faster evaluation times. In addition, several models can be evaluated in parallel in a distributed multi-core environment.

1 Introduction

1.1 Background: Safety versus Reliability in Communicating Railway Control Systems

In safety related communication domains there are two important characteristics of communication architectures: Safety and reliability. In the railway domain the standard EN 50159-2 defines a basic design of communication architectures for safety related equipment. In general the standard splits the architecture into two parts: A safety layer, which must fulfil a specific safety integrity level (SIL) and a “grey channel” without any safety responsibility (see Fig. 1 and 2). Safety layers have to detect six different types of message errors to grant functional safety. The standard EN 50159-2 defines a defence matrix against these threats (Table 1,

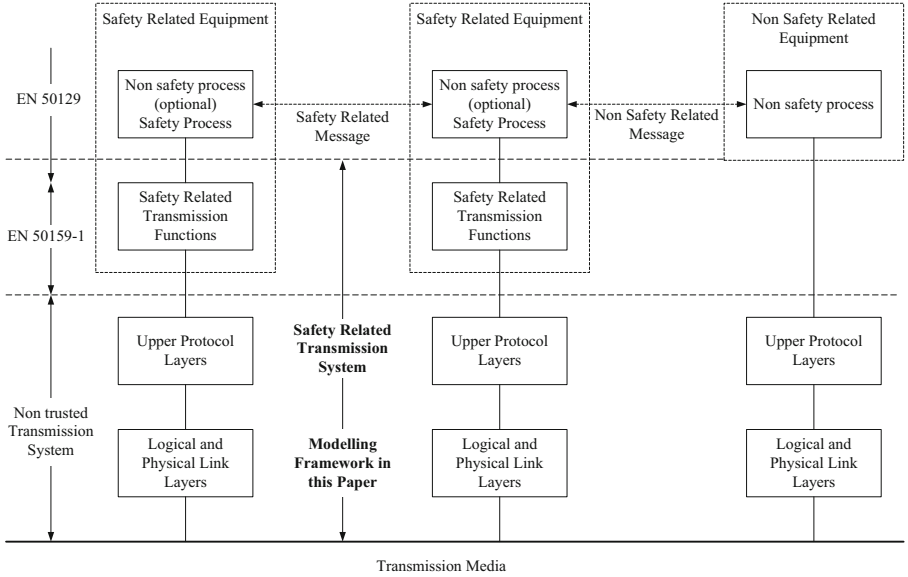


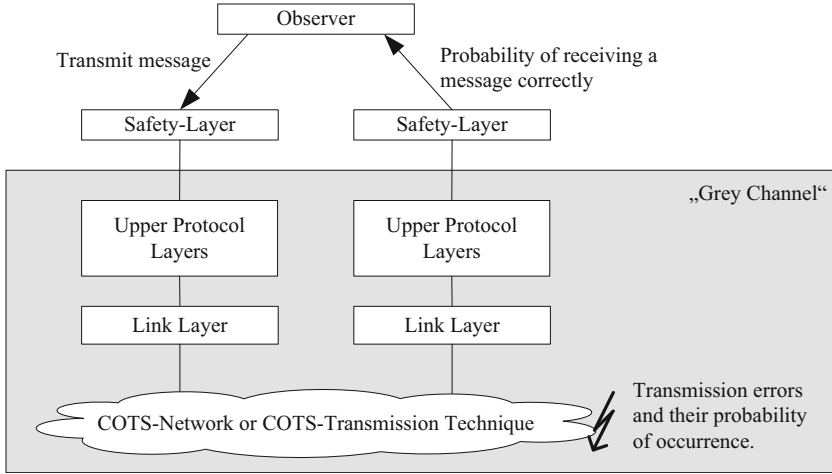
Fig. 1. Structure of safety-related communication architecture (from [2]). The term “Non Safety Process (optional)” in the *Safety Related Equipment* block indicates that also processes without safety-relevance can be deployed in the safety-critical equipment.

[1,2]). The safety reaction on such errors must be a safe state, which usually stops the communication service until the system is reinitialised or reset by an operator. Therefore a safe communication reduces the fault tolerance against arbitrary transmission errors and lowers the reliability of the communication architecture. To improve the fault tolerance against message errors it is necessary to use a reliable message transmission service (e.g. ARQ, Automatic Repeat Request) before the safety check is executed. A reliable transmission service can be included in the safety layer, in the upper protocol layer of the grey channel or in both layers (Fig. 2).

A “naive” combination of fault-tolerance mechanisms in the grey channel and safety layers will not necessarily increase the overall fault-tolerance: if, for example, lost messages in the grey channel lead to re-transmissions after timeouts, the message eventually passed to the receiving safety layer may be out-dated and therefore has to be discarded. As a consequence, it is necessary to perform analyses whether – given a trustworthy estimate for the occurrence of basic transmission faults as classified in Table 1 – the fault-tolerance mechanisms deployed in the grey channel will really increase the overall reliability of the distributed safety-critical control system.

1.2 Objectives and Contributions

In this paper we present a novel method for reliability analysis of safety-related communication architectures structured into safety layers and grey channels as

**Fig. 2.** General Modelling Architecture**Table 1.** Threats

Defences Threat	Sequence number	Time stamp	Time out	Src. and dst. ID	Feed-back message	Identification procedure	Safety code
Repetition	x	x					
Deletion	x						
Insertion	x			x	x	x	
Resequencing	x	x					
Corruption							x
Delay		x	x				

described in the previous section. In this context reliability is defined as the probability that the overall system will perform its (deterministic) safety-related services in a given operational time period $[t_1, t_2]$ without interruption and resulting transition into stable safe state, though transmission faults may occur in the grey channel with a given probability (see IEC 60050(191) [3] for the general definition).

Our analysis approach uses a domain-specific modelling language (DSL) developed by the authors. This DSL facilitates modelling communication architectures and protocols, together with the fault hypotheses concerning the probabilistic occurrence of the basic faults listed in Table 1. These communication models are used to create *mutants*, that is, derived models showing erroneous behaviour resulting from one or more basic faults occurring in compliance with the fault hypotheses at various places in the communication architecture. For each mutant the probability of its occurrence can be calculated. Since the mutants themselves show deterministic (erroneous) behaviour, conventional non-probabilistic model

checkers can be used to analyse whether the safety-related services will still operate properly in presence of the behaviour specified by the mutant. Time constraints play an important rôle in the behaviour of the system layers involved; therefore we have chosen Timed Automata [4] for modelling the mutant behaviour and use the UPPAAL tool to perform the associated analyses. UPPAAL verifies or falsifies TCTL (Timed Computation Tree Logic) statements like $A\Box\phi^1$ or $E\Diamond\phi^2$ on a given UPPAAL model [5]. The verification goal $A\Box(\text{SAFE} \wedge \neg\phi)$ is to show that the safety layer will always satisfy its safety-specification **SAFE** and never transit into stable safe state ϕ , despite of the faults occurring in the grey channel according to the mutant model under investigation. If a combination of faults on the grey channel leads to a violation of $A\Box\text{SAFE}$ the design has to be changed in any case, since a design-intrinsic safety violation that can be provoked by erroneous grey channel behaviour is not to be tolerated, regardless of the probability of its occurrence. If all mutants satisfy $A\Box\text{SAFE}$, they are classified by their occurrence probability, and according to their satisfaction or violation of $A\Box(\neg\phi)$. Then the resulting reliability of the overall system is calculated as the probability that only correct behaviour or mutants satisfying $A\Box(\neg\phi)$ occur during the given operational time period.

Our modelling approach requires *transaction-oriented* processing of safety-related communication functions: it is assumed that each activity consists of a bounded number of communication and processing steps, such that (1) the success or failure of the activity can be clearly determined after this sequence, and, (2) the success of the actual transaction is stochastically independent on the success of preceding actions. In the context of safety-related communication architectures this restriction is not a severe one: applications usually proceed according to different protocol phases like system setup, connection request, transmission of one application-specific datagram, and going through each of these phases corresponds to processing transactions of a specific type $T_\ell, \ell = 1, \dots, q$. A minor limitation is discussed in Section 5.

We have developed an integrated tool chain starting with the modelling phase supported by the MetaEdit+ meta case tool [6] which was also used to design the DSL. A model-to-text generator creates an internal representation of the DSL model. A mutation generator creates the mutants from this model and calculates their occurrence probability. Each mutant is expressed by an XML text representation conforming to the internal input format for UPPAAL models.

Our main contributions consist in the design of the DSL, the automated generation of the mutants and the calculation of their occurrence probability. Furthermore, our approach avoids the occurrence of state space explosions arising when all possible faulty behaviours are simultaneously considered in one probabilistic model (see further comments in Section 1.3). Finally, the different mutants can be analysed independently; therefore our analysis tool distributes the UPPAAL

¹ “Always globally ϕ ”: in every computation possible according to the model, and in every state of such a computation, predicate ϕ holds.

² “Exists finally ϕ ”: there exists a model computation where finally a state satisfying ϕ is reached.

model checking tasks over several computers and CPU cores, so that model checking of different mutants can be performed simultaneously.

1.3 Related Work

Model-checking has been widely used for the verification of communication protocols and also for checking safety-properties of systems, see [7,8,9] and the references given there for related work in the railway domain. Reliability aspects have mostly been approached by means of probabilistic model checking, see, for example, [10,11].

Our solution differs from the latter in that we deliberately do not use probabilistic model checking for these reliability aspects: extensive experiments performed by our group with the PRISM tool [11] showed that (1) the lack of real-time modelling capabilities enforces abstractions which either oversimplify the real communication behaviour or leads to unnecessarily complex constructions involving clock tick counters or similar devices, and (2) the incorporation of *all* possible faulty behaviours in one model lead to unacceptable checking times and even state explosions for the more sophisticated models. Indeed, since the probability that all possible faults occur while processing one transaction is so low that it can be neglected anyway, such a model would contain many computations of no practical relevance. Finally, (3) tools like PRISM only handle numeric probability values, but do not allow to investigate symbolic ones. As a consequence, parameter-dependent analyses require to re-run the time-consuming model checks for every parameter value to be considered.

Our approach tackles the combinatorial problem by checking many models instead of a single one and profit from the smaller size of each model: the complexity of evaluating one (probabilistic) model incorporating all possible faults is considerably higher than checking many simpler models, in particular, if the simpler models can be checked in parallel. Additionally, we calculate algebraic representations of occurrence probabilities. As a consequence, parameter-dependent analyses can be made by just inserting concrete probability values into the parameters of the formula.

1.4 Overview

In Section 2 we sketch the work flow supporting reliability analysis and the tool components involved. Section 3 introduces the DSL CAMoLa, our description formalism for communication architectures. In Section 4 the principles of mutation generation and the reliability calculation based on mutant evaluation are described. Section 5 contains a discussion of results and prospects for future work.

2 Workflow and Tool Chain

The reliability analysis workflow starts with modelling a communication architecture in the domain-specific *Communication Architecture Modelling Language*

(*CAMoLa*), using the informal communication architecture specification with associated protocol descriptions as input (Fig. 3). Next, *CAMoLa*'s model-to-text generator transforms the *CAMoLa* model into an UPPAAL model, enriched with syntactic markers for the so-called *behaviour switches* which are part of the *CAMoLa* formalism and used to model possible deviations from normal behaviour (see Section 3 below). Now the mutation generator tool inserts *behaviour-vectors* (Section 3) into the UPPAAL model to create mutations with different message transmission behaviour. Intuitively speaking, each vector specifies which deviations from normal behaviour are applied to message sequences passing at specific locations in the model, and each model location where faulty behaviour is anticipated is associated with such a vector. The mutation generator records the algebraic formula for each mutation's occurrence probability in a table. Each formula is an arithmetic expression over the occurrence probability parameters associated with each fault type (see Table 1) possibly occurring in some part of the model when processing a message. Then the UPPAAL tool is activated to verify the reliability property on the mutation; this process is parallelised over several CPU cores and computers to increase performance. For each mutant, it is recorded in the table whether it shows reliable behaviour or leads to a transition into stable safe state.

3 The Communication Architecture Modelling Language *CAMoLa*

CAMoLa was designed for modelling communication architectures and associated protocol behaviours. Each model is derived from the informal specification of the architecture and consists of synchronised processes representing protocol components, transmission channels or additional components simulating environment behaviour or acting as observers in the verification process. *CAMoLa* and its model-to-text generator were designed with the tool *Metaedit+* [12], which is a meta-modelling and modelling-workbench [6]. The DSL supports two hierarchical views on communication architectures: A view on all components with their interactions (Fig. 4) and a process view on each component behaviour in timed automata notation (Fig. 5).

CAMoLa extends the usual timed automata notation by the notion of *behaviour-switches* bs , representing controlled normal and exceptional behaviour transitions between locations (see Fig. 5). Each possible controlled transition is identified by a marker from set $o_{bs} = \{0, \dots, n, stop\}$. The transition connected to one distinguished switch position (position 1 in Fig. 5) is associated with normal behaviour at this model location, so the error-free timed automata model can be extracted from the *CAMoLa* model by deleting at every behaviour switch all outgoing transitions but the one associated with normal behaviour. Each other switch position gives rise to a type of mutated behaviour.

In order to reflect the possibility of different types of transient errors occurring at a specific model location, mutant models are not simply generated from the *CAMoLa* model by fixing switch positions, but by associating each behaviour

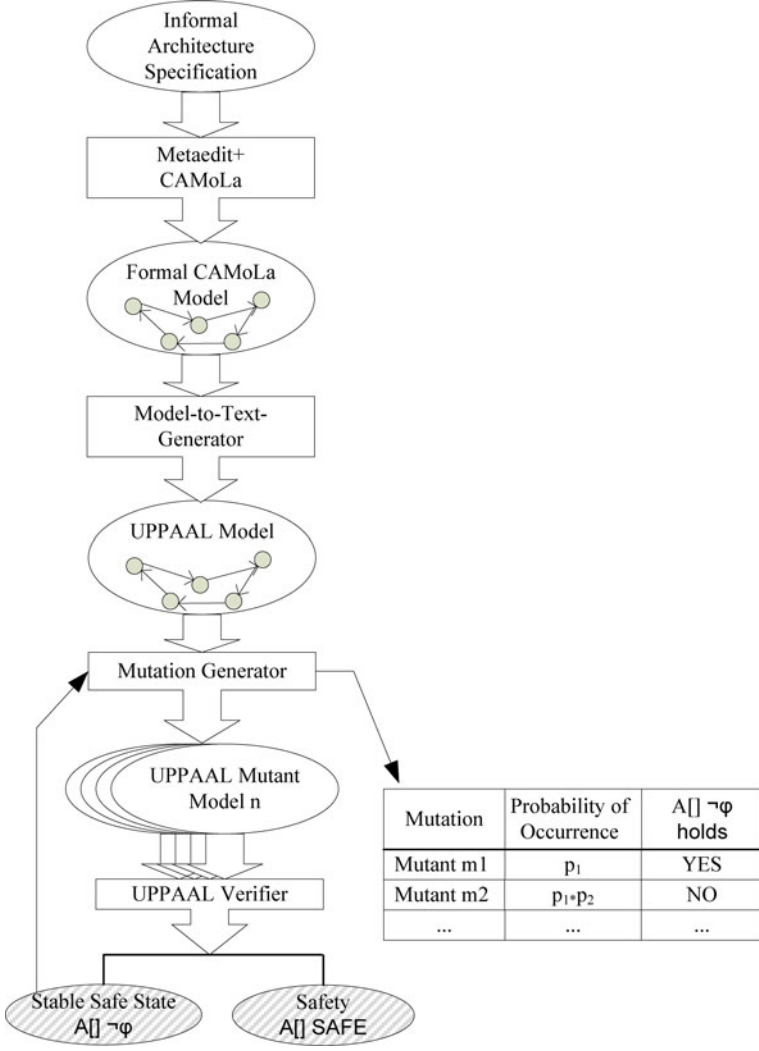


Fig. 3. Workflow of the presented Framework

switch with *behaviour-vectors* v^d : if $o_{bs} = \{0, \dots, n, stop\}$, then $v^d \in \{0, \dots, n\}^d$, and it specifies that the first d messages $m_1, \dots, m_d$ passing along the model location controlled by bs trigger transitions $v^d(1), \dots, v^d(d) \in \{0, \dots, n\}$, respectively (Fig. 6).

The semantics of this construction is defined by translating the CAMoLa process containing the pair bs, v^d into an ordinary timed automaton utilising an additional auxiliary variable j counting the number of messages passing along the behaviour switch, that is, the number of outgoing transitions of bs which have been triggered so far, and an auxiliary location l_{stop} : suppose that bs is

located at source location l and that the switch controls outgoing transitions with identifiers $0, \dots, n$, leading to target locations $l_0, \dots, l_n$. Then the associated timed automaton has outgoing transitions

$$\begin{aligned}
 l &\xrightarrow{j < d \wedge v^d(j)=0/j:=j+1;} l_0 \\
 l &\xrightarrow{j < d \wedge v^d(j)=1/j:=j+1;} l_1 \\
 &\vdots \\
 l &\xrightarrow{j < d \wedge v^d(j)=n/j:=j+1;} l_n \\
 l &\xrightarrow{j \geq d} l_{stop}
 \end{aligned}$$

at location l (j is initialised to 0 when the automaton is initialised).

While the designers specify the behaviour-switches and model the possible deviations from normal behaviour, behaviour-vectors are generated automatically by the mutation generator (Section 4). In order to control this generation process, each behaviour-switch position carries an upper bound indicating up to how many times the transition can be taken. The bound can be taken from the set $\mathbb{N}_0 \cup \{*\}$ (in the sample state machine of Fig. 5 only 0 and * are used): symbol * indicates that the mutant generator can select this transition an unbounded number of times when generating behaviour-vectors; a bound $b_e \in \mathbb{N}_0$ associated with transition e constrains the behaviour-vector generation in such a way that e occurs at most b_e times in the vector: $\text{num}(v, e) \leq b_e$. A bound b_e reduces the amount of model-mutations but leads to an under-approximation in the reliability calculation (see further comments in Section 4).

Observe that all locations introduced on behalf of the behaviour-switch are urgent, since the switch is only a selector of normal or mutated behaviour, and does not consume processing time in the real world.

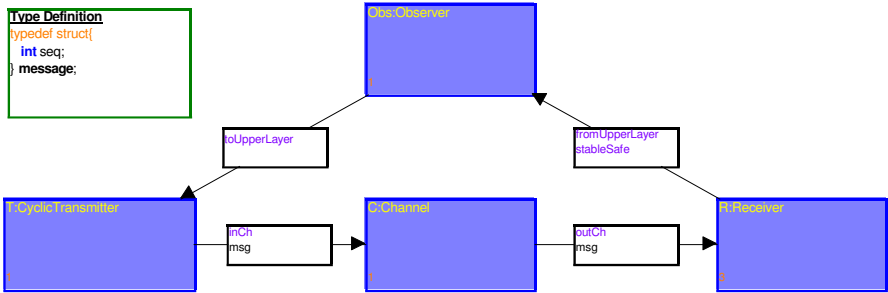


Fig. 4. Simple Architecture, System View

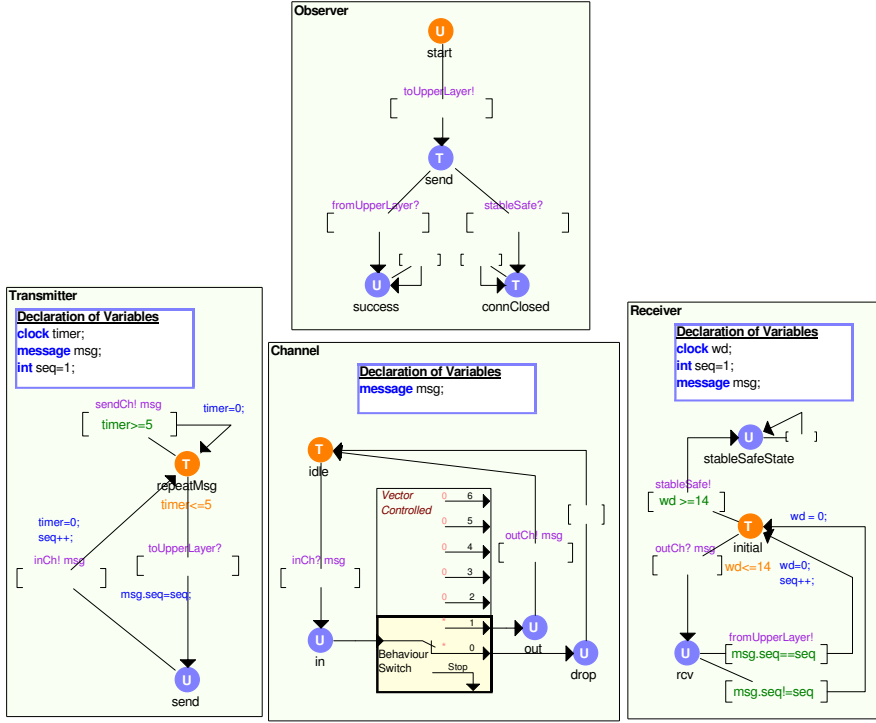


Fig. 5. Simple Architecture, Process View

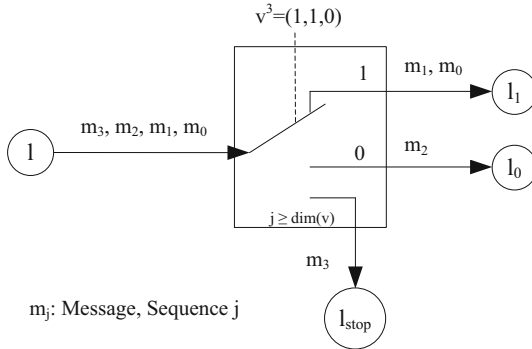


Fig. 6. Vector controlled Behaviour-Switch

4 Mutation Generation and Reliability Calculation

Generation Concept. Suppose we have created a CAMoLa model for each transaction type T_ℓ occurring in our communication architecture. The mutation generator creates concrete mutants as timed automata models, where all

nondeterminism regarding fault occurrences has been eliminated. This is achieved by means of the behaviour-vectors: let $\{bs_1, \dots, bs_k\}$ the behaviour-switches in the CAMoLa model associated with transaction type T_ℓ . Given a bound $max \in \mathbb{N}$, the mutation generator creates tuples of behaviour vectors $V = (v_1^d, \dots, v_k^d)$, such that each behaviour switch bs_i is associated with one behaviour vector v_i^d of dimension d , and the following conditions are fulfilled: (1) $d \leq max$, (2) each vector component $v_i^d(j), i = 1, \dots, k, j = 0, \dots, d-1$ is in range $\{0, \dots, n_i\}$, such that an outgoing transition with identifier $v_i^d(j)$ exists at behaviour switch bs_i , (3) the mutants $\mathcal{M}(V)$ associated with V satisfy $A \Box (\neg \phi)^3$ (we call them *reliable mutants*), and, (4) reducing the dimension of any v_i^d by one will result in an unreliable mutant satisfying $E \Diamond \phi$. Conditions (3) and (4) are checked by means of the UPPAAL model checker.

Calculation of Overall Reliability. It is our objective to calculate an approximation of the communication architecture's expected reliability which is *conservative* in the sense that the real reliability is equal to or better than the calculated estimate. The calculations performed below are based on the assumptions that (1) no other faults occur in the communication system than the anticipated ones that have been represented in the CAMoLa model by means of behaviour-switches, (2) all faults occur in a stochastically independent manner, and, (3) the safety-related services are performed in a transaction-oriented manner as explained in Section 1.2, so that the outcome of transactions is again stochastically independent.

If these hypotheses are satisfied it is possible to approximate the reliable system operation $R(t_0, t_1)$ over a time period $[t_0, t_1]$ by means of the reliability of single transactions: suppose that R_{T_ℓ} is the probability that execution of transactions of type T_ℓ will not transit into stable safe state, but perform the specified service, and that different transaction types $T_\ell, \ell = 1, \dots, q$ have to be considered. For each transaction type T_ℓ let $c_\ell^{max} \in \mathbb{N}$ the maximal number of T_ℓ -transactions which are possible per time interval $[t_0, t_1]$, and $\delta_\ell > 0$ the minimal duration of such a transaction. Then the overall reliability $R(t_0, t_1)$ can be approximated conservatively by

$$R(t_0, t_1) \geq \min \left\{ \prod_{\ell=1}^q (R_{T_\ell})^{c_\ell} \mid 0 \leq c_\ell \leq c_\ell^{max} \wedge t_1 - t_0 \leq \sum_{\ell=1}^q c_\ell \cdot \delta_\ell \leq t_1 - t_0 + \varepsilon \right\}$$

for ε satisfying $0 \leq \varepsilon < \max\{\delta_\ell \mid 1 \leq \ell \leq q\}$. The right-hand side of the above formula represents the worst-case situation, where a maximal number of transactions is performed during time interval $[t_0, t_1]$, and the combination of transactions performed in this interval is technically still possible, but represents the least reliable combination which may occur. It remains to determine the reliability of each transaction type T_ℓ . To this end, we observe that the occurrence probability of a reliable mutant $\mathcal{M}(V)$ is

³ Recall that ϕ denotes the property that the system is in stable safe state, that is, still safe, but no longer operable.

$$P_V = \prod_{i=1}^k \prod_{j=0}^{d-1} p_{v_i^d(j)}^i$$

where p_e^i denotes the occurrence probability of the basic fault (or normal behaviour) associated with outgoing transition number e at behaviour switch bs_i (so $\sum_{e=0}^{n_i} p_e^i = 1$ for all $i = 1, \dots, k$). The probability that a transaction of type T_ℓ will terminate successfully without transition into stable safe state is

$$R_{T_\ell} = \sum_{\{V \mid \mathcal{M}(V) \models A \Box (\neg \phi)\}} P_V + \sum_{\{\pi, V \mid \mathcal{M}(V) \models E \Diamond \phi \wedge \pi \models \Box (\neg \phi)\}} P_\pi \cdot P_V$$

where π denotes a computation of mutant $\mathcal{M}(V)$ and P_π the probability of π 's occurrence: R_{T_ℓ} is the sum of all occurrence probabilities of reliable mutants plus the occurrence probabilities of paths in unreliable mutants leading to successful completion of the transaction. If we neglect the occurrence probability of reliable computations π in unreliable mutants and only consider reliable mutants whose behaviour vectors v are of dimension $\dim(v) \leq \max$, and each transition e emanating from a behaviour switch occurs at most $b_e \in \mathbb{N}_0$ times in v , this results in the conservative approximation

$$R_{T_\ell} \geq \sum_{\{V \mid \mathcal{M}(V) \models A \Box (\neg \phi) \wedge \forall v, e: \dim(v) \leq \max \wedge \text{num}(v, e) \leq b_e\}} P_V$$

The right-hand side of this inequation can be computed by the mutant generator in combination with the model checker.

Example. As an example we demonstrate the calculation of the reliability of the example architecture in Fig. 4. This architecture consists of a transmitter, channel, receiver and observer, transmitter and receiver being allocated in the safety-layer. The observer performs a safety-related transaction which completes successfully in terminal state `success` if a message sent on channel `toUpperLayer` is finally received on channel `fromUpperLayer` (see Fig. 5). The transmitter sends messages in fixed cycles of 5 time units. It repeats a message with the same sequence number until a next message has to be transmitted. The receiver removes duplicated messages indicated by identical sequence numbers. It also monitors the operability of the transmission channel: at least one message within 14 time units is expected (regardless of the sequence number). If no message is received within 14 time units, the receiver transits into `stableSafeState`, so we are interested in the probability that the complete system satisfies $A \Box \neg \text{Receiver.stableSafeState}$, or, equivalently, $RQ \equiv A \Diamond \text{Observer.success}$ (“ RQ ” standing for *Reliability Query*). The communication channel includes a behaviour-switch bs_1 with the set of outgoing transitions identified by $\{0, 1\}$. The outgoing transition number 0 models the message-loss-error and transition 1 transmits the message correctly. The *-character in the behaviour switch denotes that the transition can be take arbitrarily many times, so there are no restrictions regarding the creation of behaviour-vectors for bs_1 . The

mutation generator generates the initial vectors $v_{1,1}^1 = (1)$, $v_{1,2}^1 = (0)$ and starts the model-checking processes to verify RQ on each mutation. The model mutation induced by $v_{1,1}^1 = (1)$ satisfies RQ , but the mutation induced by $v_{1,2}^1 = (0)$ violates RQ , because the mutant derived from $v_{1,2}^1$ will drop the first message and block as soon as the second message arrives. In the next generation step, the mutation generator extends all vectors which are not satisfying RQ by all possible outgoing transitions of the behaviour-switch – this results in $v_{1,1}^2 = (0, 1)$, $v_{1,2}^2 = (0, 0)$ – and resumes the verification process. The tool iterates until the dimension of the vectors have reached a predefined limit (in the example we set the limit to 4, because we know that RQ can never be satisfied in presence of more than 3 message-drops). In Fig. 7 the whole set of generated behaviour-vectors is shown, each inducing one mutant model.

All behaviour-vectors whose mutants satisfy RQ represent reliable computations of the communication architecture: each transmission where only fault-combinations still ensuring $A \Box \neg \text{Receiver.stableSafeState}$ occur is still reliable. The probability of transmitting a sequence of messages specified in a behaviour-vector is calculated due to the known probability for an error-type to occur. In our example there is a probability to drop (p_0) or to transmit ($1 - p_0$) a message. The probability that a sequence of controlled transitions occurs is the product of each transition probability in a behaviour-vector (e.g. $v_{1,1}^3 = (0, 0, 1)$, probability of occurrence: $p(v_{1,1}^3) = p_0 \cdot p_0 \cdot (1 - p_0)$). We assume that all events are stochastically independent. Now the reliability of a communication model is the sum of all mutation-occurrence probabilities satisfying RQ . For the example system this results in the reliability formula $R_{Ex} = (1 - p_0) + p_0 \cdot (1 - p_0) + p_0^2 \cdot (1 - p_0)$ which can be reduced to $R_{Ex} = 1 - p_0^3$. $\square$

5 Discussion and Future Work

The reliability analysis of communication architectures according to the concepts introduced in this article allows users to compare different architectural designs and fault-tolerance mechanisms of communication protocols in safety-related domains. Furthermore, the analysis results induce requirements on message error probabilities. These probabilities represent decision criteria whether specific transmission techniques like WLAN, IP-Networks or xDSL should be allowed or forbidden in safety-related communication architectures with high reliability requirements. We have successfully analysed the reliability of the safety protocol SAHARA over UDP [13], a proprietary session-layer over the HDLC (High-Level Data Link Control) protocol, PROFIsafe over PROFINET and PROFINET DCP (Basic Discovery and Configuration Protocol). The results of these analyses imply maximal error probabilities and properties like maximal latencies of transmission techniques which are still acceptable in presence of the high levels of overall reliability required. Additionally the knowledge about the communication behaviour in presence of errors and error combinations has led to improvements of protocol specifications. Due to the divide-and-conquer approach the availability of an array of computers and multiple CPU cores makes

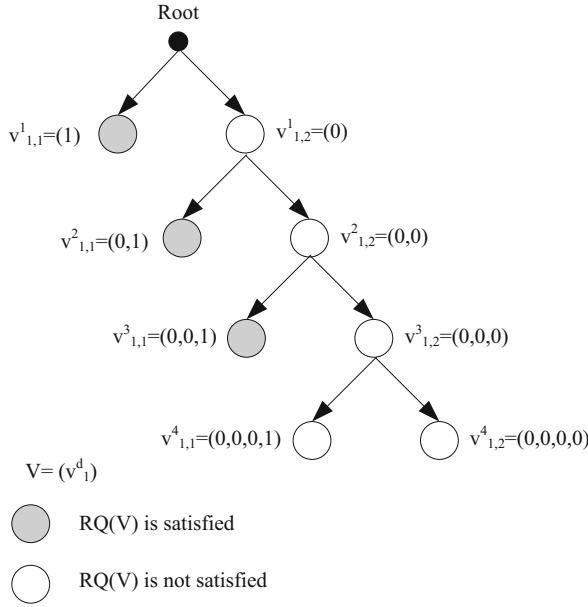


Fig. 7. Generated Vector Tree

model checking feasible on a large amount of error combinations (i. e., mutants). We have successfully analysed an architecture with about 34 million error combinations which takes about 60 hours with an array of 25 computers (each 3 GHz).

In the future, we will analyse further architecture specifications, especially with reliable transport protocols like TCP and SCTP. Furthermore, we will improve the DSL CAMoLa for modelling communication architectures in a more generic way, such that pre-defined error behaviours applicable in specific communication domains can be re-used by means of building blocks from libraries. Additionally, it is planned to allow deviations from the transaction-oriented approach, in the sense that some system variables will be allowed to evolve across sequences of transactions. This will be helpful if, for example, fault counters are introduced in the system and incremented across transitions, so that shutdowns can be enforced if the fault rate is considered to be too high: in such a situation the success of a transaction also depends on the probability that the fault counter has reached its admissible limit before start of transaction.

Acknowledgements. The second author has been supported by Siemens AG in the context of the Graduate School on Embedded Systems GESy at the University of Bremen (<http://www.informatik.uni-bremen.de/gesy>).

References

1. CENELEC: En 50159-1. railway applications -communication, signalling and processing systems part 1: Safety-related communication in closed transmission systems (2001)
2. CENELEC: En 50159-2. railway applications -communication, signalling and processing systems part 2: Safety related communication in open transmission systems (2001)
3. IEC: Iec 60050-191-am1 ed1.0 amendment 1 - international electrotechnical vocabulary. ch. 191, Dependability and quality of service (1999)
4. Alur, R., Dill, D.: A Theory of Timed Automata. *Theoretical Computer Science* (126), 183–235 (1994)
5. 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)
6. Kelly, S., Lyytinen, K., Rossi, M.: Metaedit+ a fully configurable multi-user and multi-tool case and came environment. In: Constantopoulos, P., Vassiliou, Y., Mylopoulos, J. (eds.) *CAiSE 1996*. LNCS, vol. 1080, pp. 1–21. Springer, Heidelberg (1996)
7. Esposito, R., Sanseviero, A., Lazzaro, A., Marmo, P.: Formal verification of ertms euroradio safety critical protocol. In: *Proceedings of FORMS 2003*, Budapest, Hungary, May 15–16 (2003)
8. Peleska, J., Große, D., Haxthausen, A.E., Drechsler, R.: Automated verification for train control systems. In: Schnieder, E., Tarnai, G. (eds.) *Proceedings of the FORMS/FORMAT 2004 - Formal Methods for Automation and Safety in Railway and Automotive Systems*, Technical University of Braunschweig, pp. 252–265 (December 2004) ISBN 3-9803363-8-7
9. Schlingloff, F.: Barthel: Verifikation und test des profisafe-sicherheitsprofils (2007)
10. Maxemchuk, N.F., Sabnani, K.K.: Probabilistic verification of communication protocols. In: *PSTV*, pp. 307–320 (1987)
11. Dufлот, M., Fribourg, L., Hérault, T., Lassaigne, R., Magniette, F., Messika, S., Peyronnet, S., Picaronny, C.: Probabilistic model checking of the CSMA/CD protocol using PRISM and APMC. In: *Proc. 4th Workshop on Automated Verification of Critical Systems (AVoCS 2004)*. *Electronic Notes in Theoretical Computer Science*, vol. 128(6), pp. 195–214. Elsevier Science, Amsterdam (2004)
12. metacase.com: Metaedit+ workbench (2009)
13. Kähloer, M.: The european train control system in thales signalling solutions. *Mechanics Transport Communications* 3, VIII–8–VIII–12 (2008)
14. Baier, C., Katoen, J.P.: *Principles of Model Checking*. MIT Press, Cambridge (2008)

A Novel HAZOP Study Approach in the RAMS Analysis of a Therapeutic Robot for Disabled Children

Petr Böhm and Thomas Gruber

AIT Austrian Institute of Technology GmbH,
Donau-City-Str.1, 1220 Vienna, Austria
{petr.boehm,thomas.gruber}@ait.ac.at
<http://www.ait.ac.at>

Abstract. In the EU project IROME (Interactive RObotic social MEdiators as Companions) a consortium of eight multidisciplinary partners has developed a new therapeutic robotic system for children. It is composed of a mobile base platform with obstacle detection sensors and a so-called application module with a head and movable arms. The embedded controller is programmed for various play scenarios like following a child or dancing. The system is intended to help children with minor motor disabilities or communication deficiencies like for instance autism, who are thereby expected to overcome their shortcomings. It is evident that an autonomous robot represents a potential hazard to its surrounding, in particular to handicapped children who cannot be assumed to react properly in the presence of an autonomously moving robot. Therefore, a RAMS analysis with emphasis on safety issues was performed with special respect to this specific therapeutic situation. This paper describes the methods used and the results found by applying a holistic HAZOP study with a novel two-fold approach to this specific case of a robotic system.

Keywords: RAMS analysis, HAZOP study, therapeutic robot, autonomous system, disabled children, dependability, safety.

1 Introduction

State of the art research emphasizes the important role of play in child development as a crucial vehicle for learning about the physical and social environment and for developing social relationships. IROME¹ [1] targets children who are prevented from playing, either due to cognitive, developmental or physical impairments which affect their playing skills, leading to general impairments in their learning potential and more specifically resulting in isolation from the social environment. Another effect is expected in the play therapy with autistic children whose blockade against communication with humans is expected to be released by communication with the robot in a play scenario.

¹ STREP IROME, Contract no.: 045 356, was co-funded by the European Commission within the RTD activities of the Strategic Objective SO 2.6.1 “Advanced Robotics” of the FP6.

The robot is preset to a predefined game by the operator and performs the play scenario autonomously while interacting with the patients in various ways. By the robot's interactive way of playing, children are strongly encouraged to participate in the game. As a result, they strengthen their motor abilities or overcome their communication deficiency, respectively.

Children in general and especially those with disabilities tend to unexpected behaviour. In particular, we cannot assume that they behave in a proper way to avoid critical situations like collisions or to realize hazards and react adequately. Therefore a detailed hazard analysis had to be performed. We decided to use the HAZOP (HAZard and OPerability) analysis method, which had already proven useful in another unconventional context (cf. [2], [3] and [4]), though it had originally been developed for the safety analysis of chemical process plants. Unlike in COOPERS, we performed the analysis not only with respect to the component structure of the system, but we used in addition an approach based on the play scenarios and the specific situations which are expected during a game.

After an introduction to the robotic system and an overview on the play scenarios in section 2, the third chapter explains what we investigated with the RAMS analysis; then it gives information on the hazards we expected in the robot, discusses applicable safety standards and presents our decision for choosing the HAZOP method. Details how we applied the analysis method are given in section 4, and section 5 contains a brief outline of the HAZOP results and a discussion about the particular analysis method, where the benefit of the two-fold approach becomes obvious. Finally, section 6 presents conclusions and gives an outlook on future work.

2 The IROMEC Robotic System and the Therapeutic Scenarios

In this section, the IROMEC robot and the play scenarios are briefly described.

2.1 Robot Description

The basic configuration of the IROMEC robot has a three component structure that includes a *mobile platform*, an *interaction module* and several *control buttons*.

The *mobile platform* contains all the technological components for managing the robot movement, including wheels and a collision avoidance system with a laser scanner and ultrasonic actuators and sensors.

The *interaction module*, also called *application module*, consists of

- a *body* whose skin can display different visual effects by way of a projection, thus supporting identity, expression and feedback,
- a movable *head*, which can be shaken and nodded, with a digital display for both expression and orientation, and
- *arms* for basic manipulation features.

Some *add-ons* like hair, hat and a coloured skin coating for the surface provide for personalization and customization of the robot.

The *control buttons* are used by the children in the course of the play scenarios for triggering certain functions like for instance stopping the robot when it is moving.

The therapeutic personnel controls the robot via wireless communication by means of an application on the parental personal computer.

The robot can be used in a vertical (fixed) and a horizontal (mobile) configuration, cf. figure 1.



Fig. 1. IROMEC robot design in horizontal and vertical configuration

In the horizontal position, the interaction module is attached to the mobile base platform by a mechanically locking mechanism and an electrical connector. In this configuration, the head is bent back so that the face looks ahead when the robot is moving forward.

In the vertical position, the robot is used without base module, thus unable to move, but the arms can make gestures. The head is used upright with the face looking straight forward showing expression by means of a built-in flat screen. Moreover, the robot has the capability to nod and to shake his head. In addition, the mentioned add-ons like hair and hat can be used to make the robot more attractive for the children.

A structured list in section 4.1 shows the system structure, detailed in subsystems and components. More information on the robot requirements and design can be found in the publication [5].

2.2 Play Scenarios

An essential part of the IROMEC project is the development of play scenarios and their execution in appropriate trial setups. These scenarios are supported by programmed behaviour of the robot, including obstacle detection and interaction with the child in the play context.

Ten play scenarios (cf. 6) were defined, implemented and eventually evaluated:

- Turn taking with a mobile robot
- Turn taking for sensory reward
- Make it move

- Imitation
- Follow me
- Dance with me
- Build the tower
- Bring me the ball
- Get in contact
- Pretending to be a character

Each of the scenarios comes with a basic version and one or several specific variations. The following example gives an idea how a therapeutic play scenario is typically constructed:

Scenario Example "Make it move". The game consists of clapping hands in order to make the IROMEC robot move within the room. Sitting on the floor, the child produces a sound by clapping the hands one time and the robot moves forward. To stop it the child has to clap again. To change the direction of the robot the child has to clap his hands twice or three times in order to make the robot turn left or right.

As the example illustrates, the scenarios are fairly complex sequences of events and activities which repeatedly require interactive responses of the child. All these scenarios target children with various types of disabilities, in particular with autism, severe motor impairments, or moderate mental retardation, and the scenarios vary according to the specific impairments.

3 RAMS Analysis of the IROMEC Robotic System

3.1 Term Definitions

RAMS is an acronym for the following quality attributes of dependable systems:

- **Reliability.** The ability of a system or component to perform its required functions under stated conditions for a specified period of time [8].
- **Availability.** The degree to which a system or component is operational and accessible when required for use [8].
- **Maintainability.** The ability of an item, under stated conditions of use, to be retained in, or restored to, a state in which it can perform its required functions, when maintenance is performed under stated conditions and using prescribed procedures and resources [10].
- **Safety.** Freedom from those conditions that can cause death, injury, occupational illness, or damage or loss of equipment or property [9].

Some authors include security as additional quality attribute in "RAMSS":

- **Security.** Protection from unauthorized access or uncontrolled losses or effect [11].

3.2 Quality Attributes Considered for the IROMEC Robot

The quality attribute we had to analyse primarily was *safety*. This pertains to the hazards caused by electrical, mechanical, thermal as well as chemical energy. For the

IROMEC robot, which is operated in a therapeutic environment with handicapped children, we had to assume unexpected spontaneous reactions like intentional collisions or stepping on the robot as well as dismounting or even sucking parts of it. The above mentioned specific conditions did not require a modification of the method, but the judgement on the degree (if at all) to which we could rely on a safeguard had to be adapted to the increased unreliability of children.

For functional safety of PES (programmable electronic systems), the generic international standard IEC 61508 [7] or an equivalent domain-specific standard has to be applied. As in many cases, functional safety of electronic equipment and control software depends on its *reliability*; therefore the latter quality attribute had to be treated as well.

Availability of the robot is of interest for successfully performing the experiments and for efficiently evaluating the effectiveness of the therapy with the robot. But for a safety analysis this quality attribute is only of interest as far as the availability of safety functions is concerned. More on which requirements apply for safety functions can be found in [7].

Maintainability is of high value when adaptations of the robotic system are necessary in the course of the project. But the resulting robot demonstrator is not yet a production sample, which should expose maintainability features as expected by the customers from a mature product. For this reason we excluded this property from our analyses.

Although the first editions of the generic functional safety standard IEC 61508 [7] neglected *security* it can play an important role in safety-critical systems. In the IROMEC RAMS analysis we were aware of security but it was not the main focus. The most vulnerable part with respect to security issues is the wireless connection between the parental PC and the robot. Here we can rely on security features like authentication and encryption, which are offered by the standard components used. Further security aspects are malicious incorporation of dangerous features during robot development or abuse during operation. All these security threats were not treated further as we excluded malicious intent and rely on standard components. The only remaining issues are abuse and intentional damage of the robot by the playing children - these threats had to be included in the analyses.

3.3 Applicable Safety Standards for IROMEC

The long-term goal of the IROMEC project is to bring play robots to the market and to support thereby the therapy of children with certain disabilities. Any potentially hazardous product on the market must conform to the respective normative safety standards. In the absence of a directly applicable domain-specific standard, the generic functional safety standard IEC 61508 [7] has to be used as a default.

The IROMEC robot is something between a toy and an electro-medical appliance. The general safety requirements for toys are described in EN 71 "Safety of toys" [12]. This standard covers all aspects applicable to toys which are used by children, typically also when they are alone without supervision by adults. So this standard covers many safety aspects of IROMEC. But on the other hand IROMEC is a therapeutic appliance. So, to some extent medical standards can be appropriate.

The standard EN / ISO 14971:2009 [14] gives a framework for hazard analyses and is a source to get useful hints. The safety standard IEC 60601-1 [13] gives directives for the design and implementation of medical electrical devices, but of course not for therapeutic play robots. So, for our uncommon and entirely new case, a specifically targeted analysis was required.

3.4 Goal of the IROMEC RAMS Analysis

At the first glance a play robot seems widely harmless as we are all used to see battery-driven toy cars in toy shops. But when analyzing a play robot system more in-depth, one gets aware of many different hazard types, which to be treated in a safety analysis: Mechanical, electrical and chemical hazards, but also heat, pressure, radiation, and - more specifically - suffocation.

As described in section 2.1 of this paper, the IROMEC robot is able to go around, to nod and shake its head and to move its arms. Depending on the mechanical parameters of these movements like speed, force and torque, the robot can hurt the children during the play scenarios. Manipulations by the supervisory personnel before, during and after the play have to be considered, too. Moreover the robot contains potentially hazardous substances: There are accumulators with etching chemicals, surfaces which possibly evaporate poisonous substances, especially when they become hot. Heat itself represents a danger, and there are even parts which can explode like accumulators and electrolytic capacitors.

Also a passive safety perspective has to be taken into consideration: Children, in particular mentally disabled ones, may tend to suck attractive things or even to swallow small parts if they can loosen them from the robot.

Summarizing we have to conclude that there are a lot of different hazards in a play robot, even more in the context of therapeutic plays with mentally or physically handicapped children. The goal of the IROMEC RAMS analysis was to identify these risks covering all different kinds of hazards, all thinkable situations and all technical components contained in the robot.

Traditional hazard analysis methods like HAZOP (HAZard and OPerability) study or FMECA (Failure Modes, Effects and Criticality Analysis) usually start from a block diagram of the system. This allows good coverage of all components of the system but lacks a systematic approach for granting good coverage of all kinds of actions occurring during play scenarios or other manipulations. Therefore we took a two-fold approach as will be detailed further below in section 4.1 .

3.5 Method for the RAMS Analysis

For conventional software-intensive systems a number of proven state of the art methods (including appropriate tools) for dependability analyses (both of components and systems) are available, such as HAZOP study, FMECA, FTA (Fault Tree Analysis) or ETA (Event Tree Analysis). In the first step the suitability of these methods for the robot system designed for IROMEC was assessed.

For our purpose, we decided to apply a HAZOP study to the IROMEC robot design. The HAZOP analysis, which was originally developed for the chemical process industry, is specifically appropriate for safety analyses of new systems. It allows - due

to its highly systematic approach - to deliver results with a very good coverage of all hazards. We applied it for the specific case of a therapeutic robot, which is somewhat different in detail compared to the original domain of process industry. But we already have experience with HAZOPs in unconventional cases, cf. [2]. The following chapter explains how a HAZOP analysis works in our specific case.

4 Performing the HAZOP

4.1 Scope of the Analysis and Assumptions

For the IROMEC analysis we treated a *components* and an *operations* view. Thereby, some overlap was expected, but the structured method helped to avoid leaving undetected gaps and overlooking risky abnormal situations.

The HAZOP procedure applied in our case can be summarized as follows:

- Define the objects of the HAZOP study based on components & scenarios
- Define the parameter sets to be analysed
- Use the general set of guidewords

For the analysis from a components point of view, we need first a system structure containing all electrical parts in the robot. The following list gives an overview of the electronic equipment from the components point of view.

Technical System Structure According to the Components View

- C1. Robot base
 - C1.0. Base body
 - C1.1. LiPo (lithium polymer) accumulators
 - C1.2. Motors
 - C1.3. Wheels
 - C1.4. Obstacle detection system
 - C1.4.1. IR (infra red) sensors
 - C1.4.2. US (ultra sonic) transmitter and sensors
 - C1.4.3. Laser scanner
- C2. Application module
 - C2.0. Application module body (without display and skin)
 - C2.1. LiPo Accumulator
 - C2.2. Arms
 - C2.3. Head
 - C2.4. Skin / Display / Button slots
 - C2.5. Electronic robot control units, comprising APC (Application module PC) and AµC (Additional MicroController) in the application module plus PPC (Platform PC) in the robot base platform

The components listed below were considered potentially hazardous to a very low degree. Therefore we considered their conformance with toy standards, especially [12], but they were not subject to the more specific HAZOP study.

- Spongy rubber ball
- Play cubes
- Coloured tag
- Gear box: Parts being ejected are considered not hazardous as they will remain within the robot base housing. Hot lubricant is excluded due to the low speed of the robot. It is expected that long hair cannot enter the gearbox in any case. The only remaining risk of blocking the motor is handled in the "motor" table.
- Skin

Operations View. The operations view comprised actions which are part of the play scenarios as well as preparatory activities performed by the therapeutic personnel or supervisors.

- O1. Robot rotates
 - O1.1. with arms extended
 - O1.2. with arms closed
 - O1.3. without arms
- O2. Robot moves
 - O2.1. with arms extended
 - O2.2. with arms closed
 - O2.3. without arms
- O3. Robot moves arm
- O4. Robot moves head while standing on the table
- O5. Robot closes arms (picks object)
- O6. Robot dances
- O7. Robot destroys tower of cubes
- O8. Robot performs "Follow me"
 - O8.1. without tag
 - O8.2. with tag
- O9. Robot reacts
 - O9.1. on clap
 - O9.2. on human voice
 - O9.3. on artificial sound
- O10. Robot stops in front of an obstacle
- O11. Robot rolls over long hair of a child lying on the floor

The following operations were considered as potentially hazardous to a very low degree; therefore they were not further subject to the HAZOP analysis.

- Adult puts robot on table
- Adult takes robot from table
- Robot is moved and connected to charging station
- Mount left/right arm
- Dismount left/right arm
- Patient composes hand-held remote control
- Patient composes button set on robot's back

The following Operation was identified as hazardous but - after a detailed discussion - not subjected to the HAZOP analysis as the risk was quite clear and we could not expect that applying parameters and guidewords would bring additional information.

- Robot standing on a table without base tilts and falls from the table
- Robot with base is put on a table, moves and falls down

Summarizing the hazards associated with components and those related to operations it comes out clearly that the commonly used components view alone would not have yielded results with sufficient coverage of the hazards, and that - vice versa - an analysis based on the operations perspective only would neglect many hazards inherent to certain components.

As an example, the above mentioned hazard of a robot falling from the table would not have been identified by a merely component-based approach. On the other hand, in an analysis focused on operations only the risk of an exploding lithium polymer accumulator would remain undiscovered.

Our two-fold analysis promised to result in a very high coverage of hazards, but some risks were detected by both approaches. This is no problem for sufficient coverage but causes some extra work and makes the results less concise. However, this drawback is minor compared to the clear advantage of maximized hazard coverage. And it is easy to tackle with it avoiding double entries in the analysis results: After discovering the double entry of a hazard we discussed from which viewpoint the issue could be analyzed better and made a decision in which part of the analysis - component centred or operations based - the hazard should be described. For consistency reasons, a reference to the description at the respective location in the other analysis part was of course added.

4.2 Parameters and Guide Words

The parameters we had to consider were entirely different for the two approaches, while the guide words were the same.

Parameters for Components View. After the structure of the robot had been analyzed as depicted above, all parameters which are reasonably applicable to these components were defined. Parameters are, for instance, voltage, strain, temperature, or speed. Not all parameters were applicable to all functions or components, if not they were marked in the analyses as "NA" (not applicable). The following list shows the parameters used for the components view.

- a. Electric current
- b. Voltage
- c. Power
- d. Temperature
- e. Pressure
- f. Stored chemical energy
- g. Emission of chemicals
- h. Time

- i. Turning rate
- j. Torque
- k. Force
- l. Light intensity
- m. Sound volume
- n. Electromagnetic field strength
- o. Separation (removing parts, parts being ejected or falling down)
- p. Integrity (of robot body, head, arms, buttons, button case, different charging stations, parental remote control: nothing broken, splintered, torn off, ejected, fallen off, polluted)

Parameters for Operations View. For the operational approach, we used the following very small set of parameters.

- A. Turning rate / Torque
- B. Speed / Force
- C. Distance to obstacle

Some remarks must be made to explain the selection of parameters given above, in particular regarding the very short list used in the operations view.

- For the arms, the moment of inertia was not considered as a parameter based on the assumption that the mass of the arms is much smaller than the one of the robot's body. Instead we used the maximum moment of inertia of the entire robot with extended arms.
- Stored mechanical energy was considered only in connection with operations, and there it was eventually substituted with "turning rate/torque" and "speed/force".
- We treated stored electrical energy only in connection with capacitors; the stored energy in accumulators was subsumed under chemical energy.

Guide Words. For HAZOP analyses, the following set of *Guide Words* is applied to all combinations of processes and parameters: No, More, Less, As well as, Part of, Reverse, Other than, Early, Late, Before, and After.

Although this set of guide words was defined for the original target of HAZOP analyses - chemical process plants - we used the same set for the programmable electronic system in IROMEC, like we had already done it in the COOPERS RAMS analysis (cf. [2], [3], and [4]). At the first glance this may seem remarkable because of the evident difference between chemical plants and mechatronic appliances for therapeutic purposes. But the difference lies in the parameter sets: We included, for instance, electric current or torque as parameters, whereas in chemical industry flow, concentration or fluid level plays a role.

After all this had been defined in the preparatory phase, the analysis itself began. The HAZOP method avoids any unwanted omissions by its inherently systematic approach: Each of the guide words is applied to all reasonable pairs of a parameter and a function or component. If the combination makes sense, the analysis team discusses the meaning of the combination and identifies hazards and causes, and, if

necessary, existing and - otherwise - proposed countermeasures to mitigate the hazard. The results are eventually presented to the design team with the request to make the necessary modifications.

5 Results of the Analysis

Our HAZOP analysis aimed at pointing out to possible risks and at minimizing them. As described above, we took a two-fold approach for analyzing the safety of the robot. The component-centred part allowed reliable coverage of the entire system while the operational approach helped to cover hazards which became evident only when we took the situation of the playing children with their impairment into account. Due to the two-fold approach some double findings had to be treated. But on the other hand the method helped essentially to avoid missing certain risks.

The components approach would have revealed risks associated to a rapidly moving robot colliding with a child; so the reliability of the collision avoidance system seemed critical. But we concluded from analyzing the scenarios that we had to assume that collisions between robot and children during playing are not only normal and unavoidable, but that they also occur when the robot is standing still.

The complete analysis results were written in the form of HAZOP tables; in the following, examples of HAZOP-sheets are shown. Table 1 shows an excerpt from the component-wise analysis, Table 2 stems from the operational approach.

Table 1. HAZOP sheet from the component-wise analysis

Process: C1.0 Base body				Function: Housing of the robot base			
Item Nr	Deviation	Causes	Detection	Consequences	Safe-guards	Action	Comment
C1.0.a	XXX electric current						N.A.
C1.0.b	NO voltage						is safe
	MORE voltage	Contacts with hazardous voltage accessible from outside	Electric shock	Electric shock		Verify that no contacts with hazardous voltage can be touched (EN 71)	if switched off "yes": 12/16V; battery current limitation, => severe violation improbable
	PART OF voltage						N.A.
							N.A.

The item number (first column) is the combination of a component (e.g. "C1.0") and a parameter (e.g. "b"). XXX instead of a guide word refers to *all* guide words and was applied where none of them was applicable.

Table 2. HAZOP sheet from the operational approach

Process: O10 Robot stops in front of an obstacle				Function: Is part of play scenario.			
Item-Nr.	Deviation	Causes	Detection	Consequences	Safe-guards	Action	Comment
O10.A	Xxx turning rate / torque						NA
O10.B	Xxx speed / force						NA
O10.C	NO distance to obstacle	obstacle detection and/or collision avoiding system malfunction	visible	collision, child possibly hurt, object damaged	diverse collision detection system failure detection?	UM1; rounded bottom edge! Test necessary!	special risk: hand of child on floor -> invisible for laser scanner => hurt (7mm distance)
	LESS distance to obstacle	obstacle detection and/or collision avoiding system malfunction	visible	failure deteriorates, later collisions may occur		UM2	
	MORE distance to obstacle	obstacle detection and/or collision avoiding system malfunction		play does not work properly		UM2	not safety-critical
	PART OF distance to obstacle						NA
							

In column Action, UM1 and UM2 refer to requested user manual entries:

- UM1..... "Stop playing with robot and call maintenance"
- UM2..... "If this occurs repeatedly call maintenance"

The risk detected in item O10.C (Guide word "no" applied to Operation "O10" combined with parameter "C") is a typical example how beneficial the operations view was. We would not have derived the following recommendation from a mere components point of view:

- A hand of a child sitting on the floor could be probably invisible for the sensors and the collision avoiding system, a child could be hurt => rounded bottom edges are recommended, a test is necessary.

On the other hand, voltage-related hazards like in table 1 would not be found in an operational approach. High coverage was attained only by the two-fold analysis.

Several further recommendations for the robot-design and play-scenarios were made during and after the HAZOP analysis:

- Robot arms should be designed soft and without sharp edges
- All used material should be conformant to EN 71
- Over-current and over-temperature detection should be implemented
- Make failure detection of collision avoiding system (sensors) fail-safe

6 Conclusions and Further Work

In our HAZOP analysis we decided to take a two-fold approach: On the one hand a component-centred view of the robot system and on the other a operational one. Both views have shown several interesting results; some of them need further study, others indicate potential for optimization of the play robot.

Of interest is the fact that we partly encountered double coverage of hazards; on the other hand, several risks were found only in one of the approaches. This was expected for hazards of electronic parts which clearly belonged to the components view. But several aspects could be treated correctly only on the basis of the scenarios approach. This finding is promising for future research on unconventional application of hazard-evaluation techniques; we plan to use similar approaches in related areas.

References

1. <http://www.iromec.eu>
2. Selhofer, A., Gruber, T.: COOPERS RAMS Analysis. In: Saglietti, F., Oster, N. (eds.) SAFECOMP 2007. LNCS, vol. 4680, pp. 87–92. Springer, Heidelberg (2007)
3. Gruber, T., Althammer, E.: Sicherheitsanforderungen und Validierung eines kooperativen integrierten Verkehrsmanagementsystems, OVE Schriftenreihe Nr.50, Tagungsband zur Informationstagung Mikroelektronik 08, Vienna, Austria, pp. 320–326
4. Gruber, T., Althammer, E., Schoitsch, E.: Safety Requirements for a Cooperative Traffic Management System: The Human Interface Perspective. In: Buth, B., Rabe, G., Seyfarth, T. (eds.) SAFECOMP 2009. LNCS, vol. 5775, pp. 159–172. Springer, Heidelberg (2009)
5. Marti, P., Moderini, C., Giusti, L., Pollini, A.: A Robotic Toy for Children with special needs: From requirements to Design. In: IEEE 11th International Conference on Rehabilitation Robotics, Kyoto, Japan, pp. 918–923 (2009)
6. Robins, B., Ferrari, E., Dautenhahn, K.: Developing Scenarios for Robot Assisted Play. In: Proceedings of the 17th IEEE International Symposium on Robot and Human Interactive Communication, August 1–3, Technische Universität München, Munich (2008)
7. EN ISO/IEC 61508, Functional Safety of Electrical/Electronic/Programmable Electronic Systems, Part 1 – Part 7 (1998–2001)
8. IEEE 90 – Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York (1990)
9. MIL-STD-882B Military Standard - System Safety Program Requirements
10. BS 4778-3: Quality vocabulary Part 3: Availability, reliability and maintainability terms (1991)
11. ECSS P-001B Glossary of terms, European cooperation for space standardization (July 14, 2004)
12. EN 71 - Part 1-11 Safety of toys (2005-2010)
13. IEC 60601-1:2005 Medical electrical equipment - Part 1: General requirements for safety, International Electrotechnical Commission (2005)
14. EN ISO 14971: Medical devices. Application of risk management to medical devices (2009)

Variability Management of Safety and Reliability Models: An Intermediate Model towards Systematic Reuse of Component Fault Trees

Carolina Gómez¹, Peter Liggesmeyer¹, and Ariane Sutor²

¹ Technical University of Kaiserslautern, Dept. Software Engineering: Dependability
Building 32, Paul-Ehrlich-Straße, 67663 Kaiserslautern, Germany
{gomez, liggesmeyer}@cs.uni-kl.de

² Siemens Corporate Technology, Reliability and Availability
Otto-Hahn-Ring 6, 81739 Munich, Germany
ariane.sutor@siemens.com

Abstract. Reuse of fault trees helps in reducing costs and effort when conducting Fault Tree Analyses (FTAs) for a set of similar systems. Some approaches have been proposed for the systematic reuse of fault trees along with the development of a product line of systems. Nevertheless, these approaches are not longer effective when FTAs are performed after systems have been put into operation. This is mainly due to the lack of product line information required to make fault trees reusable. The model proposed in this paper is a step towards systematically reusing fault trees in the aforementioned context. It acts as an intermediate model between the specification of a system and its corresponding Component Fault Tree (CFT). In particular, it abstracts from the implementation details of a CFT, allowing the integration of variability inherent of product line systems as well as the one obtained from performing fault tree analyses incrementally over time. The model is part of a systematic reuse approach.

Keywords: Component Fault Trees, Variability Management, Fault Tree Analysis, Product Line Engineering, Safety and Reliability.

1 Introduction

Fault Tree Analysis (FTA) is an accepted technique to estimate the Safety and Reliability of Systems. It has been widely used for a variety of domains, including automotive, avionics, power generation and railway systems. Most systems in those domains are developed based on product lines, in order to reduce production costs and to improve end user quality. A possibility to conduct fault tree analyses for such systems, is to do it in parallel with their development (in the early phases), once product line requirements and possible hazards/failures of the system are identified. This allows fault trees to be systematically reused, since product line information is directly mapped to the failure events of the fault tree. However, fault tree analyses can be also conducted after product line systems have been developed. For example, when analyzing causes that lead to an accident [9] or to do preventive/corrective maintenance. At Siemens, we have identified

the need for reusing fault trees in a systematic way. Results of an early survey have shown us that fault trees are reused in an ad-hoc manner, based on the experience of the engineers and on previous FTAs done to a similar system. This type of reuse is not suitable to share the knowledge acquired by performing such analyses. The knowledge mostly remains private and tacit of the individuals. In some cases it becomes public but mostly not explicit. Therefore, there is a lack of guidance on what, how and from where to reuse. The main idea of the model proposed in this paper is to provide a basis for solving this problem.

This paper is structured as follows: in Section 2, related approaches for the systematic reuse of fault trees following the principles of product line engineering are presented. Section 3 gives a description of our model. Section 4 provides an example of the usage of the model as well as a description of the systematic reuse approach. Section 5 introduces a summary of the results of our work and also future work (pending research questions).

2 Related Work

Nowadays, some Embedded Systems (ES) are developed by following the principles of Component Based System Engineering (CBSE) and Product Line Engineering (PLE). To analyze quality properties of those systems, such as Safety and Reliability, fault trees are used. A first attempt to facilitate the analysis of complex CBSE-based systems through FTA was Component Fault Trees (CFTs) [7]. In contrast to regular fault trees, CFTs are built by considering the decomposition of a system by its architectural components. Besides reducing the complexity of FTA, the CFT concept opened up the possibility for fault trees to be reused. This possibility was further explored by several approaches that took advantage of the potential for reuse naturally present in PLE-based systems. Some of them focus on the derivation and evaluation of product line safety related requirements supported by fault trees [5,14] whereas others focus on the derivation of product line fault trees [4,10,15]. Safety requirements are derived and evaluated in [14] by conducting a safety assessment process to identify the hazards of a family of systems and by building scenarios¹ to identify the most vulnerable points of variation. Fault trees are used here to graphically visualize the most vulnerable points of variation in terms of failure events that lead to the occurrence of a specific hazard. Product line fault trees are derived in [4] based on Commonality and Variability Analysis (CVA) and Preliminary Hazard Analysis (PHA). Commonalities and variabilities as well as a list of hazards are identified for a product line. A product line fault tree is then constructed to visualize at which extent each single commonality/variability contribute to the occurrence of an identified hazard for the product line. Similarly, an approach proposed by Noda et al. [13] uses fault trees that contain product line information to create exception handling features². Additional approaches are mainly dedicated to the reuse of dependability information and not to the reuse fault trees. For instance, an approach proposed by Kelly and McDermid [8] uses safety case patterns to be applied in specific

¹ An scenario is built for each variant of the system.

² Exception Handling Features are components and/or services of the system that support the realization of corrective actions.

circumstances, e.g. when constructing a particular safety argument. These patterns are described by using a goal structuring notation. In conclusion, the aforementioned approaches are useful when conducting FTA in parallel with the development of a product line of systems. Concretely, our work differs from the aforementioned approaches in the following points:

- We consider the construction of fault trees not in the development phases of an ES but afterwards.
- Product line information that is required to make fault trees reusable is not known beforehand, we capture it incrementally over time.
- There is no single standard ES specification from which we obtain the information needed to derive fault trees, within our model we propose a standard one.
- We consider not only variability of product line systems, but also of FTAs.

Nevertheless, our approach is similar to [8] in which we use dependability information as a criterion for reusing fault trees.

3 Intermediate Model

The Intermediate Model (IM) was created as an attempt to systematically reuse CFTs, provided that:

CFTs are constructed after systems have been developed, which usually happens when an external entity has to prove how safe/reliable a system is after being put into operation.

FTAs are done to a set of similar systems over time and thus complete documentation of a product line is not provided, just the one related to single systems. Therefore, product line information that is required to make CFTs reusable has to be collected insofar as FTAs are conducted.

In general, since documentation of systems is presented in a variety of ways, we needed to define a representation of a system that exactly contains the information sufficient to build CFTs. The IM along with a FTA specification form an abstract representation of a CFT. In particular, the IM serves as a filter for extracting the information needed of an embedded system specification to construct a fault tree. Additionally, since we are not only considering product line information of systems as the only criteria to reuse CFTs, but also dependability and FTA-related information, we decided not to model variability directly in the CFTs but in the IM. This allows us to better capture and integrate variability over time. In short, instead of directly reusing CFTs we propose the IM as the product line artifact to be reused when constructing CFTs.

Figure 1 depicts a conceptual diagram of the IM. Given the specification of an embedded system, we filter the information that is relevant for us to conduct a FTA and we document it in the IM of a system. We define the IM of a system, as the set of IMs of its single components. The IM is divided into two parts. The first part, Component Information, describes the contents, structure, behavior and functionality of a component, as proposed in the Kobra [1] and MARMOT [3] approaches for product

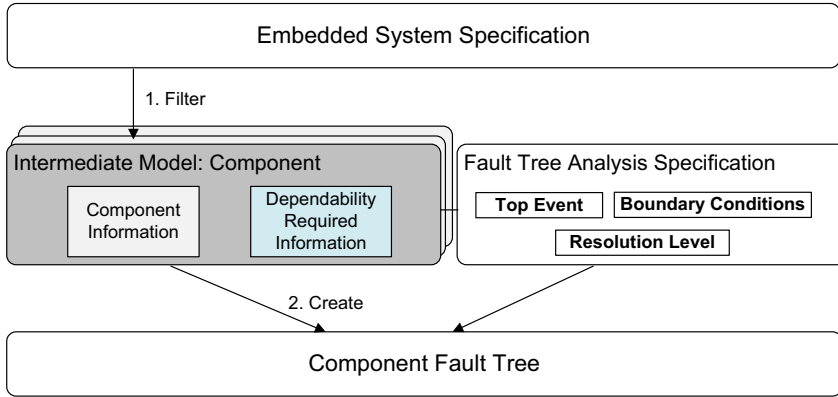


Fig. 1. Intermediate Model Conceptual Diagram

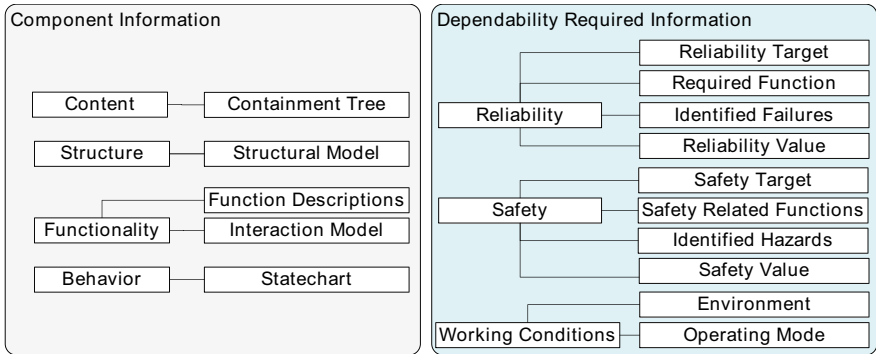


Fig. 2. Intermediate Model Structure Diagram

line and embedded component design.³ The second part, Dependability Information, includes relevant safety and reliability information, as well as the working conditions of a component. The IM together with a FTA Specification, will be the basis for the semi-automatic generation of Component Fault Trees (CFTs.)

Figure 2 shows the structure of the Component and Dependability Information parts of the IM. For the Component Information part we consider the models for component specification of the Kobra and MARMOT approaches.⁴ The Content of a component is specified by using a containment tree, similar to the containment hierarchy defined in the MARMOT approach. An ES is composed out of hardware components (at the

³ Kobra is a component based product line development approach and MARMOT is an extension of Kobra dedicated to Embedded Systems.

⁴ Component design by using Kobra and MARMOT defines two phases: Specification and Realization. Each phase includes different models to describe a component from different perspectives. Once the first phase is completed, the models for Specification are refined into their corresponding ones for Realization.

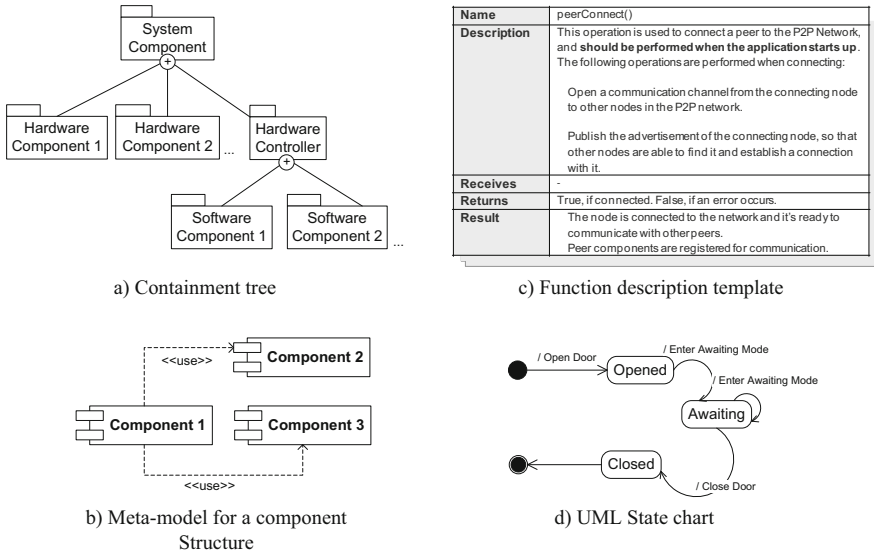


Fig. 3. Component Information Example

first level of the tree) and whenever a software controller is identified for the hardware device, its containment structure is depicted (at the second level of the tree), as shown in Figure 3 a). An abstract representation of the Structure of a component includes the list of functions that the component provides as well as its interaction with other components. This interaction can be, for instance of type, "uses", which states that a component requires to use other component(s) in order to provide its intended functionality. A meta-model for component structure is depicted in Figure 3 b). Moreover, the Functionality of a component is represented by a list of a description of its functions, which can be documented by using the template presented in Figure 3 c). Finally, Behavior of a component is described by using an UML statechart as shown in Figure 3 d).

Dependability information of a component is required to conduct a FTA, and also to use it as a criterion for reusing IMs. The Dependability Information part of the IM defines a structure for documenting information about safety, reliability and working conditions of a component, see Figure 4. At first, we need to specify a safety or reliability target depending on the focus of the FTA . This target can be described informally by using text, or formally, e.g., by using temporal or boolean logic formulas. In case a reliability analysis is performed, we need to know the required function of a component. This not only includes a textual description of the functionality of a component but also a mission profile⁵. As opposed to reliability analysis, a safety analysis requires

⁵ According to [2] a mission profile defines the execution of a component's functionality under stated working conditions as a function of time.

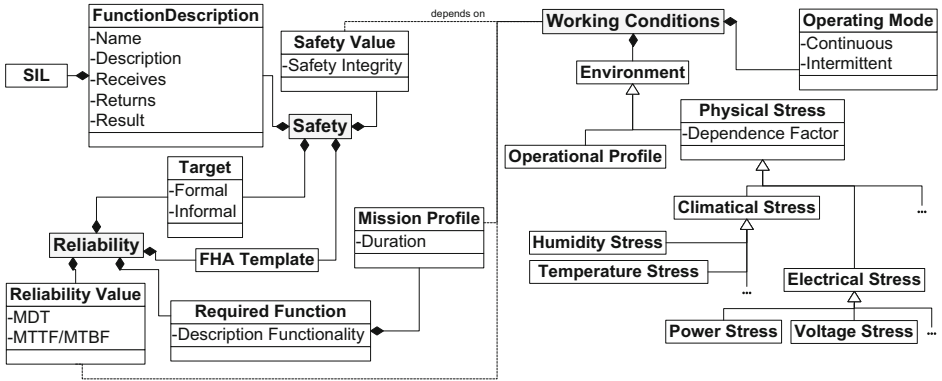


Fig. 4. Intermediate Model Dependability Information

the knowledge of safety related functions. These are documented by adding a Safety Integrity Level (SIL) to each of the component's function descriptions. We also require to document the identified failures or hazards of a component. Since both hazards and failures might have common elements to be documented, we have chosen a Functional Hazard Analysis (FHA) template for this purpose. Moreover, quantitative values for safety and reliability of a component can be estimated and therefore they have to be documented. For instance, representative reliability values are Mean Time To Failure (MTTF), Mean Time Between Failure (MTBF) or Mean Down Time (MDT), whereas safety can be represented by using the Safety Integrity⁶ value.

The estimation of these values is influenced by the working conditions of a component. These conditions might vary depending on the environment and the mode in which a component operates. We define the environment of an ES component in terms of its surrounding physical stresses as well as the profile of its input data. Physical stresses are related to heat, radiation, salt, sand, dust, noise, vibration, shock, fall and acceleration [2] and quantified by using dependence factors. A dependence factor is a mathematical equation that describes the empirical model of the physical factor dependence with respect to a failure/hazard rate. Moreover, the profile of the input data affects the behavior of the software controller. For instance, we expect that a software component executing in a business/industrial environment will deliver a different behavior from the one delivered in a home/domestic environment. For defining this type of environment we use the operational profile. One way of defining the operational profile according to Musa et al. [11], is to define the run types⁷ that a software function executes together with the probabilities in which they occur in a given time interval. Additionally, the mode of operation of a component refers to the continuity in time at which a component executes i.e. continuous or intermittent.

⁶ Safety Integrity is the probability of a component satisfactorily performing the required safety functions under all stated conditions within a stated period of time.

⁷ The type of a run is characterized by its input states. An input state is defined by the set of values of the input variables of a software function along with their probability of occurrence.

3.1 Fault Tree Analysis Specification

In order to obtain a CFT for the system, we require to document information about the Fault Tree Analysis that is being performed. For this purpose, we use a Fault Tree Analysis (FTA) specification. This specification not only allows us to semi-automatically derive CFTs, but also to collect and model variability of several FTAs conducted across the time. The FTA specification includes information about the top event to be analyzed, the boundary conditions of the FTA as well as the resolution level to be taken into account. Generally, the top event is a particular failure or hazard event of a component. In the context of CFTs, one might define more than one top event [7]. Similar to a safety or reliability target, a top event is documented by using a textual description and its representing propositional logic formula. In addition to that, we also document the type of hazard/failure event that is handled. Moreover, the depth and wide of a component to be considered in an FTA is represented by the resolution level and boundary conditions, respectively. The resolution level represents the depth at which a component will be modeled in the fault tree. This depth varies from 0 to n , being n the height of the component's containment tree. A resolution level equals to 0 means that no contents of a component will be explored during the analysis, whereas a resolution level between 1 and n means that at least one part of its first containment level will be explored. Boundary conditions represent the scope of a component. This is defined as the set of units belonging to a component together with the set of interfaces by which a component interacts with others.

3.2 Variability Modeling for the IM

In order to make the IM reusable based on the principles of PLE, we need to identify the potential sources of product line information and also, we need to know how to document it. Product line information in a set of related systems refers to commonalities, common information for all the systems of the set, and variabilities, information that varies from one system to another. Within the IM we handle two different levels of variability:

Level I Variability inherent of product line systems.

Level II Variability present in the dependability conditions of a system.

Figure 5 gives an overview about variability levels and how they are related to the parts of the IM. At first, we take into account variability inherent of product line systems in the Component Information part. We consider that there are several variants of a component regarding its contents, structure, functionality and behavior. Once we resolve this level of variability, i.e. we know the specific component that will be analyzed, we pass to the next variability level (II). At this level, we assume that dependability conditions might be different for a single component. These are collected and documented by conducting FTAs incrementally over time in the Dependability Information part, in terms of safety and reliability information as well as working conditions. In order to resolve this level, a set of dependability information has to be chosen.⁸ In particular, this

⁸ A set of dependability information is chosen, when a FTA specification has been defined.

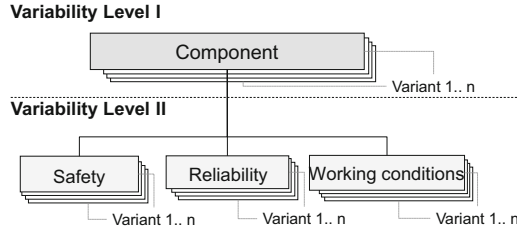


Fig. 5. Variability Levels of the Intermediate Model

set can be of safety or reliability type. For instance, in a set of reliability information we include a reliability target and its related failures, the required function and its mission profile as well as a reliability value. Analogously, in a set of safety information, we include the safety target and its related hazards, the safety related functions to be taken into account as well as a safety value. For both types of sets we also include working conditions.

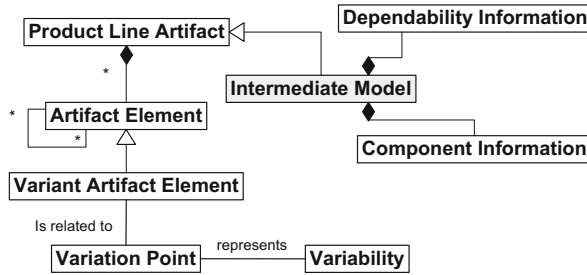


Fig. 6. Product Line Metamodel and the Intermediate Model

Moreover, variability present at both levels is modeled in the IM based on the product line information meta-model proposed in [12]. This meta-model describes how product line information can be explicitly documented in an artifact so that it becomes reusable. In terms of the model, a reusable artifact is called product line artifact. In our approach, the IM will be a product line artifact and the Component and Dependability Information parts along with their elements will be the corresponding artifact elements, as shown in Figure 6. Each artifact element can be defined as common or variable. By definition, variant artifact elements relate to a variation point, whereas common artifact elements do not relate to any source of variation. Specifically, a variation point represents one place at the IM where variability occurs.

4 Example

A reliability analysis has to be made for a Ventilation System (VS) component of the Siemens Gas Turbine SGT 400. In particular, the objective of the analysis is to estimate

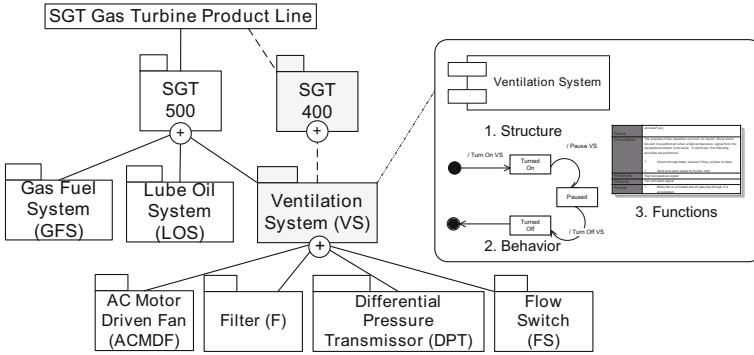


Fig. 7. Containment tree for the SGT Family

the reliability of the VS by calculating the Mean Time Between Failure (MTBF) under expected working conditions. We know that a previous reliability analysis was made to a Gas Turbine of the same family (SGT 500). Therefore, the idea is to reuse from this information as much as possible. Since our IM is part of a systematic reuse approach,⁹ the aforementioned scenario is built upon the following four steps defined therein:

- 1. Define system/component to be analyzed:** The information of the system or component to be analyzed through FTA is defined and documented by using the IM. There are two possibilities for defining the information: from scratch or by reusing from previously defined information. In the latter case, it is possible to select information from a system or component that matches exactly or partially our needs.
- 2. Define fault tree analysis information:** In order to define a FTA specification, the information captured in the IM is used. Firstly, the information about one or more top events and their type is defined. Secondly, the information about the boundary conditions and the resolution level to be considered during the analysis is specified.
- 3. Derive fault tree regarding system and FTA information:** Here, a component fault tree is derived based on the on the IM and the FTA specification.
- 4. Collect and package experiences for future analyses:** The new information acquired about the system and its FTA is packaged and integrated into a product line and application specific artifact bases. The product line artifact base, contains information that can be reused in future analyses whereas the application specific artifact base includes specific information of FTAs that cannot be reused anymore.

The next sections will describe in detail the course of events of the scenario based on the aforementioned steps.

4.1 Define System/Component To Be Analyzed

In this step, the information of a system is documented by creating a new IM model. The idea is to reuse the information of previously defined IMs. In the scenario provided

⁹ The systematic reuse approach is a tool-supported approach currently being developed in co-operation with Siemens Corporate Technology.

here, we know that we can reuse the IM of a system of the same family of Gas Turbines (SGT 500). To do this, it is required to have a look at the containment tree of the SGT 500 and to check if it contains the VS component. Figure 7, shows the containment tree of the SGT 500 embedded in an application domain tree¹⁰. Here, we can observe that the SGT 500 contains a Ventilation System (VS), a Lube Oil System (LOS) and a Gas Fuel System (GFS). In order to verify that the VS of the SGT 500 corresponds to the VS of the SGT 400, one has to compare their contents along with their structure, behavior and function specifications. If it is proven that the components are the same, the next step will be to create a new IM for the SGT 400. Furthermore, to complete the Dependability Information part of the newly created IM, it is required to check the dependability conditions of the VS component. This is done to select dependability information that can be reused for the current analysis. In the context of this example, we realize that there is information defined for safety and working conditions, but not for reliability. Thus, reliability information has to be defined:

- *Target*: MDT of the VS.
- *Required Function*: Maintain the gas turbine cool under expected working conditions during 5 years.
- *Failures*: DPT Down, FS Down, F Down and ACMDF Down.
- *Value*: To be defined through FTA.

Successively, one has to select information from working conditions. In this case, only operating mode (Continuous) is selected. The environment is not considered, because it is assumed that the VS is properly designed and constructed for the expected environmental influences.

4.2 Define FTA Information

Here, the information of the IM defined in the step above is mapped to the FTA specification. Firstly, the textual description of the reliability target is used as the informal description of the top event. In this case, a formal description as well as a type of failure for the top event are not provided. Secondly, the resolution level is set to 1. This means that we will consider the contents at the 1st level of the VS's containment tree (See Figure 7). Thirdly, the boundary conditions for the VS in the FTA are defined. The boundary contents are selected from the 1st level of the containment tree: Filter (F), AC Motor Driven Fan (ACMDF) and Flow Switch (FS). Moreover, the boundary interfaces are obtained from the structure model. As we observe in Figure 7, the VS has no interactions with other components. Thus, there are no interfaces to be included. Finally, the boundary failures correspond to the failures of the boundary contents: Filter Down, AC Motor Driver Fan Down and Flow Switch Down.

4.3 Derive CFT

The CFT for the VS component is semi-automatically derived from the IM and the FTA specification. To do this, the elements of the FTA specification are mapped to

¹⁰ The application domain tree is used to keep track of the SGT family of systems.

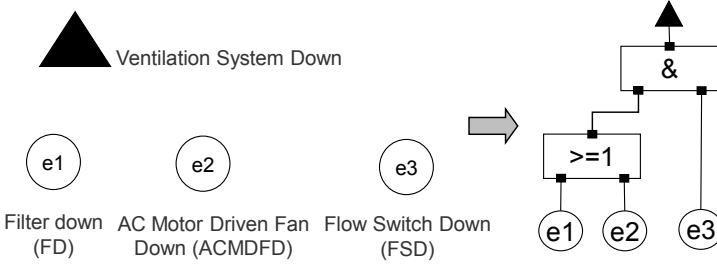


Fig. 8. Semi-automatic derivation of CFT

CFT elements. Figure 8 depicts the elements that we obtain through semi-automatic generation and how they can be connected according to the propositional logic formula defined in equation 1. Moreover, the MTBF of the VS can be calculated by using the MTBF values of the Filter, AC Motor Driven Fan and Flow Switch components.

$$Te = (e1 \vee e2) \wedge e3 \quad (1)$$

4.4 Collect and Package Experiences for Future Analyses

At the end of the process, product line information to be integrated in the product line artifact base is obtained. At the Component Information level we realize that the Ventilation System (VS) component is common to both the SGT 400 and SGT 500 systems. Since these are currently the only members of the SGT family, the VS component becomes a commonality of the product line. At the Dependability Information level, information for the reliability of the VS component, which did not exist previously, was defined. Thus, this reliability information is integrated as commonality. This remains like that, as far as different reliability information is defined for the VS. Furthermore, the IM and the FTA specification used for this particular reliability analysis are packaged and stored into an application specific artifact base.

5 Conclusions and Outlook

Systematic reuse of fault trees contributes to reduce time and development costs, as well as to have a well founded safety and reliability knowledge base. There are two alternatives to implement systematic reuse of fault trees based on Product Line Engineering (PLE) principles. The first alternative is to develop product line fault trees along with the development of product line systems and the second one is to collect product line information needed to make fault trees reusable after product line systems have been developed. Our Intermediate Model (IM) builds on the second alternative. It contributes to the systematic reuse of CFTs in the following ways:

The IM abstracts from the implementation details of a CFT acting as a filter to extract the information of Embedded Systems (ES) specifications that is required to

perform a FTA. This is useful when system specifications are presented in a variety of forms and product line information about a system and its dependability conditions has to be captured incrementally over time.

The IM considers two levels of variability allowing the information required to derive CFTs being reused based on two different criteria: similarity of system components and similarity of the dependability conditions of a component.

Besides the aforementioned contributions, this research work left a set of open questions. For instance, the question how human influences and organizational issues can be captured in the intermediate model. Approaches like the ones proposed by Leveson [9] and Kaiser [6] tackle the problem of considering organization decisions and human errors in the safety analysis of a system. Another question regarding the semi-automatic derivation of CFTs is, which derivation approach will be best suitable for our purposes. Until now, elements from the IM and the FTA specification are automatically mapped to CFT elements, but their connection is done manually. Moreover, it is to clarify how domain concepts of FTAs can be defined. This will allow FTA specifications to be reused based on Product Line Engineering principles.

References

1. Atkinson, C., Bayer, J., Muthig, D.: Component-Based Product Line Development: The Kobra Approach. In: *The First Software Product Lines Conference SPLC*, vol. 1, pp. 289–309. Kluwer Academic Publishers, Norwell (2000)
2. Birolini, A.: *Reliability Engineering: Theory and Practice*. Springer, Heidelberg (2007)
3. Bunse, C., Gross, H.G., Peper, C.: Applying a Model-based Approach for Embedded System Development. In: *33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, pp. 121–128. IEEE Computer Society, Los Alamitos (2007)
4. Dehlinger, J., Lutz, R.: Software Fault Tree Analysis for Product Lines. In: *8th IEEE International Symposium on High Assurance Systems Engineering*, pp. 12–21. IEEE Press, New York (2004)
5. Giese, H., Tichy, M.: Component-Based Hazard Analysis: Optimal Designs, Product Lines, and Online-Reconfiguration. In: Górski, J. (ed.) *SAFECOMP 2006*. LNCS, vol. 4166, pp. 156–169. Springer, Heidelberg (2006)
6. Kaiser, B.: *State/Event Fault Trees: A Safety and Reliability Analysis Technique for Software-Controlled Systems*. Ph.D. thesis, TU Kaiserslautern (2005)
7. Kaiser, B., Liggesmeyer, P., Mäkel, O.: A New Component Concept for Fault Trees. In: *8th Australian workshop on Safety critical systems and software*, pp. 37–46. Australian Computer Society, Inc., Darlinghurst (2003)
8. Kelly, T., McDermid, J.: Safety Case Construction and Reuse Using Patterns. In: Daniel, P. (ed.) *SAFECOMP 1997*. LNCS, pp. 55–69. Springer, Heidelberg (1997)
9. Leveson, N.: *System Safety Engineering: Back to the Future*. Massachusetts Institute of Technology (2002)
10. Lu, D., Lutz, R.R.: Fault Contribution Trees for Product Families. In: *13th International Symposium on Software Reliability Engineering*, pp. 231–242. IEEE, Washington (2002)
11. Musa, J., Iannino, A., Okumoto, K.: *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, New York (1987)
12. Muthig, D.: *A Light-weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines*. Ph.D. thesis (2002)

13. Noda, A., Nakanishi, T., Kitasuka, T., Fukuda, A.: Introducing Fault Tree Analysis into Product Line Software Engineering for Exception Handling Feature Exploitation. In: 25th conference on IASTED International Multi-Conference, pp. 229–234. ACTA Press, Anaheim (2007)
14. Stephenson, Z., de Souza, S., McDermid, J., Ward, A.: Product Line Analysis and the System Safety Process. In: 22th International System Safety Conference, pp. 790–799. The System Safety Society (2004)
15. Sun, H., Hauptman, M., Lutz, R.: Integrating Product-Line Fault Tree Analysis into AADL Models. In: 10th IEEE High Assurance Systems Engineering Symposium (HASE 2007), pp. 15–22. IEEE Computer Society, Washington (2007)

QoS Analysis of Weighted Multi-state Probabilistic Networks via Decision Diagrams

Roberta Terruggia and Andrea Bobbio

Dipartimento di Informatica, Università del Piemonte Orientale,
15121 Alessandria, Italy

{[roberta.terruggia](mailto:roberta.terruggia@mf.n.unipmn.it),[bobbio](mailto:bobbio@mf.n.unipmn.it)}@mf.n.unipmn.it

Abstract. Network reliability analysis is usually carried out under the simplified hypothesis that the elements of the network are binary entities that can be in one of two mutually exclusive states, perfect functioning or failed. The present paper enlarges this view from two points of view. The elements of the networks are described by multiple states that can represent a variety of different situations, like degradation levels or multiple failure modes. Furthermore, in order to increase the description power of the model, we assign to each state a weight describing a performance attribute of the element in that state. The weights may assume different physical meanings so that different Quality of Service (QoS) indicators may be evaluated. We show that the QoS analysis of a multi-state weighted probabilistic network can be performed by resorting to data structures called Multi-valued Decision Diagrams. Several examples illustrate the methodology.

1 Introduction

Most works in reliability theory are based on the traditional binary concept of reliability models allowing only two possible states for a system and its components, perfect functionality or complete failure. A variety of different algorithms for system reliability evaluation of binary systems have been presented in the literature, but in the last years, methods based on Binary Decision Diagrams (BDD) have become predominant [7,1].

However, many real-world systems are composed of multi-state components, with different performance levels and several failure modes with various effects on the system's entire performance. Such systems are called Multi-State Systems (MSS). Under multi-state assumption, the relationship between component states and system states is much more complicated and the analysis becomes more complex [11,19]. Even the term "reliability" becomes inappropriate, since the combination of different states for the components may provide different levels of performance for the whole system. In this case, a more appropriate and meaningful measure is to evaluate the level of the Quality of Service (QoS) provided by the system as a combination of the states of its components. The definition and formalization of MSS can be traced long back in the reliability literature [8] but only recently, various extensions of Decision Diagrams have

been applied [20,18]. In this paper, we concentrate on the analysis of weighted multi-state probabilistic networks (WMPN). WMPN are networks whose elements, both nodes and arcs, can have different mutually exclusive states with an associated probability of being in that state. Moreover, a performance attribute, called weight, is assigned to each state of each element to characterize a property of the element in that state. In previous studies [20,18], the performance attribute was assumed to be the capacity or bandwidth of the arcs, whose value decreases with the degradation of the arc. We show that the weights can have different physical meanings so that different QoS properties can be investigated. In particular we define a performance attribute, called cost, that is increasing with the arc degradation and may represent a property, like the resistance or the time to traverse the arc.

In many real-life situations, multi-state models provide a more realistic description of the actual behavior of the system with respect to binary models, and the theory presented in this paper opens new developments at least in three directions.

- i) - Degradable systems.* Components or subsystems show different degradation levels with reduced performance capabilities. This is the case considered in the present paper and possible areas of applications are: power networks, communication and Wireless Sensor networks, pipeline networks, aqueducts.
- ii) - Multi-mode failures.* Components or subsystems may manifest multi-mode failures, typically stuck-open and stuck-close, like valves or interrupting devices. In safety studies the failure mode influences the fail safe or fail danger mode of operation of the system.
- iii) - Optimal system design.* For each component the technology or the market make available different alternatives with different costs, performances and reliabilities [12]. The optimal choice can be translated in a multi-state weighted problem.

In the present paper we limit the assignment of weights to arcs, only. We show that when the weights are assumed to be the arc capacities the QoS function represents the total flow of the network and is computed by a function that is additive over the cuts of the networks. When the weights are interpreted as costs (or traversal times) the QoS is the total time to traverse the network, and it is computed by a function that is additive over the paths. In both interpretations of the weights the analysis is performed by resorting to Multi-valued Decision Diagrams (MDD) [10,4] and by defining suitable arithmetic/logic operations on the basic MDD. Several examples and comparisons with previous literature results illustrate the capabilities of the approach.

2 Weighted Multi-state Probabilistic Networks

A weighted multi-state probabilistic network (WMPN) can be defined as a tuple $N = (G, M, P, W)$ where $G = (V, E)$ is a network with a set V of nodes

and a set E of arcs. In this paper we consider that only arcs are multi-state with probabilities and weights assigned to them. Extension to nodes is, however, straightforward. Given that n is the number of arcs, $M = \{M_1, M_2, \dots, M_n\}$ is the set of states of the arcs. $M_i = \{0, 1, \dots, m_i\}$ is the set of the exhaustive and mutually exclusive states of arc i , where state 0 is complete failure and state m_i perfect working.

P is the probability function that assigns to each arc i in state j the corresponding probability value $p_{i,j}$. Since states are exhaustive and mutually exclusive, the following holds ($\sum_{j=0}^{m_i} p_{i,j} = 1$).

W is the weight function that assigns a real value $w_{i,j}$ representing a performance attribute of the arc i in state j . We consider two cases:

- weight is a cost: by this we intend an attribute which increases as the arc degrades (like electrical resistance or traversal time). The state representing the arc perfectly working takes the lowest value.
- weight is a capacity. In this case the state representing the arc perfectly working takes the highest value.

Given a source node s and a terminal node t , we define a function $\Psi_{s,t}$ that evaluates the QoS of the network between s and t and depends on the structure of the graph G , on the multi-states M , on the probability function P and on the weight function W . The computation of $\Psi_{s,t}$ is related to the interpretation of the weight function W .

Weight as Cost - The cost is additive with respect to the traversed arcs. Given a path connecting s to t the cost of the path is the sum of the costs of its arcs. The function $\Psi_{s,t}$ is the minimum of the costs over all the possible paths connecting s to t . In a WMPN the QoS problem can be formulated as:

Problem 1 - Given $N = (G, M, P, W)$, a weighted multi-state probabilistic network, compute the probability that the cost $\Psi_{s,t}$ between s and t is below an assigned threshold φ_{max} .

To solve *Problem 1* we need some definitions.

Definition 1 - Given a network $N = (G, M, P, W)$ and a source node s and a terminal node t , the cost of a path $H(s, t)$ is the sum of the costs w of the arcs forming the path. *Corollary 1 Min-cost* - If a network $N = (G, M, P, W)$ has ℓ minpaths $H_1, H_2, \dots, H_\ell$ the minimum cost between s and t is equal to the minimal cost of all its minpaths considering the different states of the components.

The min-cost corollary says that the minimal cost between any two nodes cannot be less than the minimal cost of all its paths.

Weight as flow - We identify with flow the nominal capacity or the bandwidth that the arc is able to carry. Networks with this attribute are usually called flow networks [9,16]. In this case the function $\Psi_{s,t}$ is the maximum flow that can be transmitted from s to t and can be computed by summing the weights over the cuts. The maximum flow problem has received a great attention even in the

recent literature [2,13,5,6,3], also in the contest of multi-state systems [14,18]. In WMPN the problem can be reformulated as a QoS problem:

Problem 2 - Given $N = (G, M, P, W)$, a weighted multi-state probabilistic network, compute the probability that the flow $\Psi_{s,t}$ guaranteed between s and t exceeds a minimum threshold φ_{min} .

Definition 2 - Given a network $N = (G, M, P, W)$, and a source node s and a sink node t , the capacity of a cut $K(s, t)$ is the sum of the capacities w of all the arcs forming the cut.

Corollary 2 Max-flow Min-cut- If a network $N = (G, M, P, W)$ has ℓ mincuts $K_1, K_2, \dots, K_\ell$ the maximum flow between s and t is equal to the minimal capacity of all its mincuts taking in account the different states of the arcs.

The max-flow min-cut theorem says that the value of the maximum flow is equal to the minimal capacity carried by a mincut.

3 Multivalued Decision Diagrams

Multi-valued decision diagrams (MDDs) [10,4] are directed, acyclic graphs used to represent n -variable functions of the form

$$f : \{0, \dots, m_1\} \times \dots \times \{0, \dots, m_n\} \rightarrow \{0, \dots, S-1\}$$

Nodes in the MDD are either terminal or non-terminal. The terminal nodes correspond to the return values of the function and are labeled with a value $0, \dots, S-1$. Nonterminal nodes are labeled with a variable y_i , and contain $m_i + 1$ pointers to other nodes.

These pointers correspond to the cofactors of f , where a cofactor is defined as

$$f_{y_i=c} \equiv f(y_n, \dots, y_{i+1}, c, y_{i-1}, \dots, y_1)$$

for variable y_i and constant c . A non-terminal node representing function f is then written as the $(m_i + 1)$ -tuple $(y_i, f_{y_i=0}, \dots, f_{y_i=m_i})$.

As for BDD, the paths in an ordered MDD (OMDD) visit non-terminal nodes according to some total ordering on the variables $y_n \prec, \dots, \prec y_1$.

A reduced OMDD (ROMDD) has the following additional properties.

- There are no duplicate terminal nodes. That is, at most one terminal node is labeled with a given value.
- There are no duplicate non-terminal nodes. That is, given two non-terminal nodes $(y_i, f_{y_i=0}, \dots, f_{y_i=m_i})$ and $(y_j, g_{y_j=0}, \dots, g_{y_j=m_i})$, we must have either $y_i \neq y_j$ or $f_{y_i=k} \neq g_{y_i=k}$ for some $k \in 0, \dots, m_i$.
- All non-terminal nodes depend on the value of their variable. That is, given a non-terminal node $(y_i, f_{y_i=0}, \dots, f_{y_i=m_i})$, we must have $f_{y_i=k} \neq f_{y_i=w}$ for some $k, w \in 0, \dots, m_i$.

It has been shown that ROMDDs are a canonical structure: given any integer function and a variable ordering, there is exactly one ROMDD representation for that function.

BDDs are a special case of MDDs applied to binary logic functions. The size of ROMDD depends heavily, as in the BDD case, on the input variable ordering used to build the ROMDD. In the present case, we assume as the n variables the arcs of the graph and the n -variable evaluated function $\Psi_{s,t}$ is defined according to *Corollary 1* or *Corollary 2* depending on the definition of the weights.

- If the weights are cost functions the terminal leaves of the MDD provide all the possible values of the costs lower than the maximum threshold φ_{max} computed along the minpaths that connect s to t (*Corollary 1*).
- If the weights are capacity functions the terminal leaves of the MDD provide all the possible values of the flows greater than the minimum threshold φ_{min} that can be transmitted from s to t , computed from the mincuts following *Corollary 2*.
- The single terminal leaf labeled 0 is reached by the combination of variables for which the graph is disconnected or the function $\Psi_{s,t}$ does not respect the constraints.

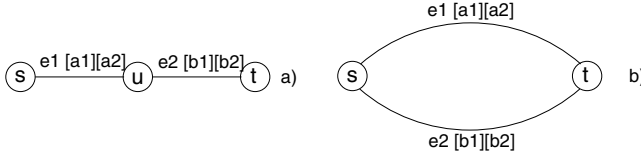


Fig. 1. Network with two arcs: a) in series; b) in parallel

3.1 Basic Operations for MDD Manipulation

In order to compute the QoS function $\Psi_{s,t}$, we need to define new operators for manipulating and constructing the MDDs. We know that the cost function is additive with respect to the minpaths while the flow function is additive with respect to the mincuts. To introduce the basic MDD operations we consider the two elementary series/parallel networks of Figure 1. The two arcs e_1 and e_2 are supposed to have three states numbered (0, 1, 2) with weights (0, a_1 , a_2) and (0, b_1 , b_2), respectively.

Weights interpreted as Costs - The network of Figure 1a) has a single minpath $H_1 = e_1 \wedge e_2$, the connectivity function is $C_{s,t} = H_1$ and when the network is connected ($C_{s,t} = 1$) the cost function is $\Psi_{s,t} = a_i + b_j$, where i and j are the states of e_1 and e_2 respectively, otherwise $\Psi_{s,t} = 0$. In the computation of the cost function the $\wedge$ operator corresponds to a sum in the costs. We call this operation *AndSum*.

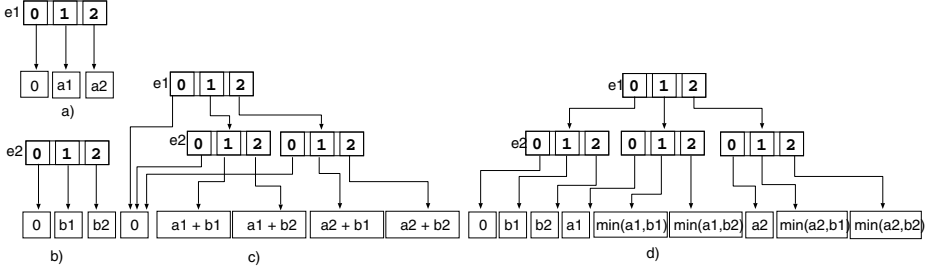


Fig. 2. Basic MDD operations: a) MDD for arc e_1 ; b) MDD for arc e_2 ; c) e_1 *AndSum* e_2 ; d) e_1 *OrMin* e_2

The network of Figure 1b) has two minpaths $H_1 = e_1$ and $H_2 = e_2$, the connectivity function is $C_{s,t} = H_1 \vee H_2$. When the network is connected ($C_{s,t} = 1$) through arc e_1 in state i the cost is $\Psi_{s,t} = a_i$, when it is connected through arc e_2 in state j the cost is $\Psi_{s,t} = b_j$, when both arcs are up the cost is $\Psi_{s,t} = \min(a_i, b_j)$. We call *OrMin* the MDD operation corresponding to the Boolean $\vee$.

Hence the generation of the cost function defined in Corollary 1 requires the implementation of the *AndSum* and *OrMin* operators. We summarize the definition of these operations in the truth Table 1 and the corresponding MDD construction in Figure 2 .

Table 1. Truth table of *AndSum* and *OrMin*

Arc e_1	Arc e_2	$C_{s,t}e_1 \wedge e_2$	<i>AndSum</i>	$C_{s,t}e_1 \vee e_2$	<i>OrMin</i>
0	0	0	0	0	0
0	1	0	0	1	b_1
0	2	0	0	1	b_2
1	0	0	0	1	a_1
1	1	1	$a_1 + b_1$	1	$\min(a_1, b_1)$
1	2	1	$a_1 + b_2$	1	$\min(a_1, b_2)$
2	0	0	0	1	a_2
2	1	1	$a_2 + b_1$	1	$\min(a_2, b_1)$
2	2	1	$a_2 + b_2$	1	$\min(a_2, b_2)$

Weights interpreted as flows - The network of Figure 1a) has two mincuts $K_1 = e_1$ and $K_2 = e_2$, the connectivity function is $C_{s,t} = K_1 \wedge K_2$. When the network is connected ($C_{s,t} = 1$) the flow function is $\Psi_{s,t} = \min(a_i, b_j)$, where i and j are the states of e_1 and e_2 respectively, otherwise $\Psi_{s,t} = 0$. In the computation of the flow function the $\wedge$ operator corresponds to a *min* in the flows. We call this operation *AndMin*.

The network of Figure 1b) has a single mincut $K_1 = e_1 \vee e_2$ and the connectivity function is $C_{s,t} = K_1$. When the network is connected ($C_{s,t} = 1$) through arc e_1 in state i the flow is $\Psi_{s,t} = a_i$, when it is connected through arc e_2 in

state j the flow is $\Psi_{s,t} = b_j$, when both arcs are up the flow is $\Psi_{s,t} = a_i + b_j$. We call this operation *OrSum*.

Hence the generation of the flow function defined in Corollary 2 requires the implementation of the *AndMin* and *OrSum* operators. We summarize the definition of these operations in the truth Table 2 and the corresponding MDD construction in Figure 3.

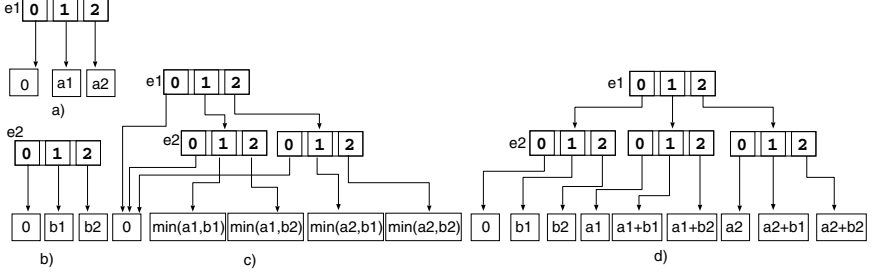


Fig. 3. Basic MDD operations: a) MDD for arc e_1 ; b) MDD for arc e_2 ; c) e_1 *AndMin* e_2 ; d) e_1 *OrSum* e_2

Table 2. Truth table of *AndMin* and *OrSum*

Arc e_1	Arc e_2	$C_{s,t}e_1 \wedge e_2$	<i>AndMin</i>	$C_{s,t}e_1 \vee e_2$	<i>OrSum</i>
0	0	0	0	0	0
0	1	0	0	1	b_1
0	2	0	0	1	b_2
1	0	0	0	1	a_i
1	1	1	$\min(a_1, b_1)$	1	$a_1 + b_1$
1	2	1	$\min(a_1, b_2)$	1	$a_1 + b_2$
2	0	0	0	1	a_2
2	1	1	$\min(a_2, b_1)$	1	$a_2 + b_1$
2	2	1	$\min(a_2, b_2)$	1	$a_2 + b_2$

3.2 Algorithm Implementation and Probability Evaluation

The implementation of the QoS evaluation algorithm is based on the construction of the MDD using the Iowa State University MEDDLY library [17]. When the QoS function is defined as a cost according to Corollary 1, we start from the list of the minpaths $H_1, H_2, \dots, H_\ell$ and we find all the possible outcomes of the function QoS by applying the operators *AndSum* and *OrMin* defined in Table 1 and Figure 2.

When the QoS function is defined as a flow according to Corollary 2, we start from the list of all the mincuts $K_1, K_2, \dots, K_\ell$ and we find all the possible outcomes of the function QoS by applying the operators *AndMin* and *OrSum* defined in Table 2 and Figure 3.

The terminal nodes of the MDD represent all the possible outcomes of the QoS function and the goal of the analysis is to find the probability associated to these outcomes. The probability of the MDD terminal nodes can be computed in a top down fashion, resorting to a breadth first search (BFS) on the MDD. The algorithm starts from the root and computes the probabilities of the MDD nodes at the successive levels until the terminal nodes are reached. Given a node d in the MDD, the probability of d depends only on the parent nodes and on the connecting edges, according to the following basic rule:

$$Pr(node\_d) = \sum_{k \in \text{set parent nodes of } d} Pr(node\_k) \cdot p(edge(k, d)) \quad (1)$$

Where $p(edge(k, d))$ is the probability of the edge connecting $(node\_k)$ with $(node\_d)$ and is given by the probability of the state of the arc represented by $(node\_k)$. For example if we consider the MDD in Figure 3c), the probability associated to terminal node 0 is computed as $P_0 = p_{1,0} + p_{1,1} \cdot p_{2,0} + p_{1,2} \cdot p_{2,0}$ where $p_{i,j}$ is the probability of the arc i in state j .

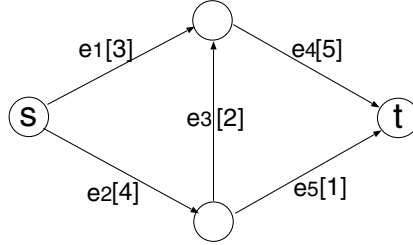


Fig. 4. Bridge network. The numbers in brackets represent the link cost/capacity as defined in Table 3

Table 3. State probability of elements

State	Probability	Link cost					Link capacity				
		e_1	e_2	e_3	e_4	e_5	e_1	e_2	e_3	e_4	e_5
perfectly working (state 1)	0.65	3	4	2	5	1	3	4	2	5	1
degradated (state 2)	0.25	6	8	4	10	2	1.5	2	1	2.5	0.5
fail (state 0)	0.1	0	0	0	0	0	0	0	0	0	0

3.3 Implementation Example

In order to illustrate the algorithmic implementation, we consider the bridge network depicted in Figure 4. The values in square brackets are the nominal weights of the arcs that can be interpreted either as costs or as flows as defined in Table 3.

Weights as costs - We assume that the arcs have three states and that the weights are increasing with the degradation according to the following specification:

- **perfectly working** : the cost of arc i is the nominal cost w_i and the probability $p_i = 0.65$;
- **degraded**: the cost of arc i is $2 w_i$ and the probability is $p_i = 0.25$;
- **failed**: the cost of arc i is 0 and the probability is $p_i = 0.1$.

The weights and probabilities used in the computation are reported in Table 3. The network has three minpaths:

$$H_1 = \{e_1 e_4\} \quad H_2 = \{e_2 e_3 e_4\} \quad H_3 = \{e_2 e_5\} \quad (2)$$

and the connectivity function is:

$$C_{(s,t)} = e_1 e_4 \vee e_2 e_3 e_4 \vee e_2 e_5 \quad (3)$$

The cost of each minpath is obtained by the *AndSum* operator while the MDDs representing the minpaths are combined by the *OrMin* operator. Assuming as an arbitrary variable ordering the sequence $e_1 \prec e_2 \prec e_3 \prec e_4 \prec e_5$, the final MDD is displayed in Figure 5.

The terminal leaves of the MDD provide all the possible QoS outcomes, and are reported, together with the associated probabilities, in Table 4a). The value 0 in the table, provides the probability that s and t are not connected.

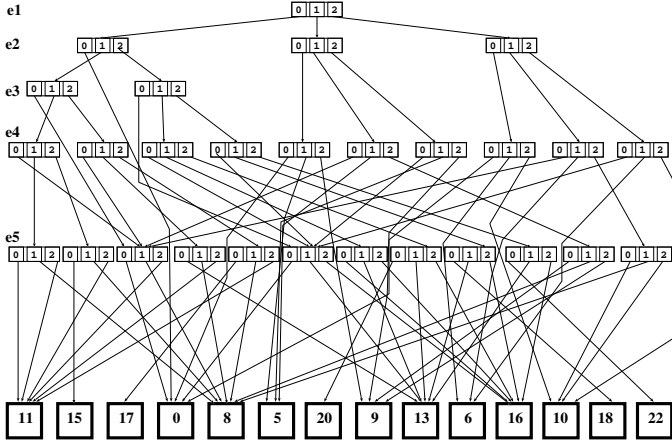


Fig. 5. MDD of bridge network (cost)

Weights as Flows - We assume that the arcs have three states and that the weights are decreasing with the degradation according to the following specification:

- **perfectly working** : the flow of arc i is the nominal value w_i and the probability is $p_i = 0.65$;

- **degraded**: the flow of arc i is $0.5 w_i$ and the probability is $p_i = 0.25$;
- **fail**: the flow of arc i is 0 and the probability is $p_i = 0.1$.

The weights and probabilities used in the computation are reported in Table 3. In this case we start from the list of the mincuts:

$$K_1 = \{e_1 e_2\} \quad K_2 = \{e_2 e_4\} \quad K_3 = \{e_4 e_5\} \quad K_4 = \{e_1 e_3 e_5\} \quad (4)$$

The operator *OrSum* is used in order to evaluate the flow of each single mincut. The MDDs representing each mincut are combined with the *AndMin* operator. Assuming the same variable ordering and building the MDD, the QoS outcomes and their probabilities are reported in Table 4b).

Table 4. Probability of the MDD terminal values

(s-t) Cost	QoS
5	0.4225
6	0.1625
8	0.175338
9	0.0938438
10	0.0360938
11	0.0336213
13	0.0319313
15	0.00105625
16	0.0129313
17	0.00040625
18	0.00040625
20	0.00040625
22	0.00015625
0	0.02881

(a) Weight as cost

(s-t) Max Flow	QoS
0.5	0.024525
1	0.06579
1.5	0.0295875
2	0.0366525
2.5	0.0622625
3	0.129128
3.5	0.194513
4	0.0513866
4.5	0.0683922
5	0.148298
5.5	0.0446266
6	0.116029
0	0.02881

(b) Weight as flow

In order to check the validity of our approach, we have compared our results with those obtained in [15] on a bridge network similar to the one of Figure 4 but with edge e_3 assumed to be bidirectional. The number of states per arc and the respective weights are reported in Table 5a). The weights have the meaning of flows and the assigned threshold is a flow between s and t equal to 3. Reference [15] considers four cases with different probability values. Table 5a) reports in the second column the probabilities of case 1, while in Table 5b) the results for the four cases are compared. We can observe a perfect fit between our approach and the one in [15].

4 Example

In order to verify the potentialities of the proposed method we have taken the benchmark network considered in [16] and reported in Figure 6a). The weights

Table 5. State probability and results of bridge benchmark network

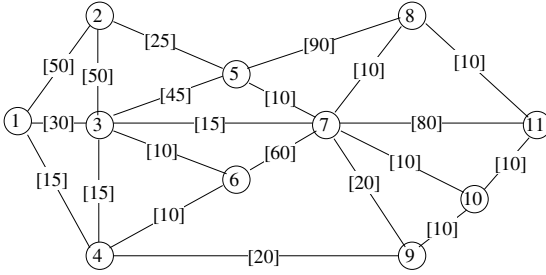
Link	States				Probability			
e_1	0	1	2	3	0.05	0.025	0.025	0.9
e_2	0	1	-	-	0.02	0.98	-	-
e_3	0	1	-	-	0.05	0.95	-	-
e_4	0	1	2	-	0.025	0.025	0.95	-
e_5	0	1	2	-	0.075	0.025	0.9	-

(a) State probability of elements of bridge benchmark network

Case	Our method	[15]
Case 1	0.830989	0.83098994
Case 2	0.677599	0.67759875
Case 3	0.553735	0.55373516
Case 4	0.49512	0.49512

(b) Results for bridge benchmark network

are reported in square brackets along the arcs, and are interpreted as flows. In [16] the standard binary behaviour (up or down) for the arcs is assumed. We have enriched the example by considering multi-state arcs with three states in Case 1 and with four states in Case 2.



(a) Benchmark network

# states	MDD nodes	MDD peak
2	9730	9732
3	143498	143498
4	621813	2904516

(b) MDD nodes

Fig. 6. Benchmark network

In Case 1 we suppose that each edge can be in one of three possible states with the following specification:

- **perfectly working** : the flow of arc i is the nominal value w_i and the probability is $p_i = 0.65$;
- **degraded**: the flow of arc i is $0.5 w_i$ and the probability is $p_i = 0.25$;
- **fail**: the flow of arc i is 0 and the probability is $p_i = 0.1$.

Table 6 shows the results. The first, third and fifth column report all the possible values of the flow that can be transmitted between s and t . Each value gives origin to a different terminal leaf in the constructed MDD. The second, fourth and sixth column report the corresponding probabilities value. Since the network has 21 arcs the state space has 3^{21} states, the final MDD contains 143498 nodes as displayed in Table 6b).

Table 6. Probability of the MDD terminal values (max flow)(3 states)

Max Flow	QoS	Max Flow	QoS	Max Flow	QoS
5	0.0034	35	0.0108	65	0.0682
7.5	0.0029	37.5	0.0232	67.5	0.0221
10	0.0100	40	0.0417	70	0.1052
12.5	0.0026	42.5	0.0204	72.5	0.0249
15	0.0200	45	0.0523	75	0.0134
17.5	0.0055	47.5	0.0459	77.5	0.0342
20	0.0210	50	0.0375	80	0.0199
22.5	0.0199	52.5	0.0167	82.5	0.0010
25	0.0255	55	0.1080	85	0.0188
27.5	0.0058	57.5	0.0455	0	0.0023
30	0.0430	60	0.0276		
32.5	0.0146	62.5	0.0864		

Table 7. Probability of the MDD terminal values (max flow) (4 states)

Max Flow	QoS	Max Flow	QoS	Max Flow	QoS	Max Flow	QoS
2.5	0.0034	23.75	0.0190	45	0.0360	66.25	0.0074
3.75	0.0029	25	0.0268	46.25	0.0223	67.5	0.0037
5	0.0038	26.25	0.0069	47.5	0.0371	68.75	0.0031
6.25	0.0029	27.5	0.0360	48.75	0.0086	70	0.0087
7.5	0.0083	28.75	0.0276	50	0.0169	71.25	0.0009
8.75	0.0058	30	0.0303	51.25	0.0218	72.5	0.0036
10	0.0100	31.25	0.0260	52.5	0.0153	73.75	0.0032
11.25	0.0109	32.5	0.0414	53.75	0.0103	75	0.0011
12.5	0.0105	33.75	0.0253	55	0.0272	76.25	0.0003
13.75	0.0035	35	0.0467	56.25	0.0058	77.5	0.0025
15	0.0237	36.25	0.0315	57.5	0.0257	78.75	0.0002
16.25	0.0171	37.5	0.0314	58.75	0.0159	80	0.0009
17.5	0.0135	38.75	0.0234	60	0.0069	81.25	0.0001
18.75	0.0073	40	0.0457	61.25	0.0086	82.5	0.0003
20	0.0182	41.25	0.0108	62.5	0.0151	83.75	0.0003
21.25	0.0106	42.5	0.0333	63.75	0.0038	85	0.0008
22.5	0.0340	43.75	0.0281	65	0.0070	0	0.0023

In Case 2 we suppose that each edge can be in one of four possible states with the following specification:

- **perfectly working** : the flow of arc i is the nominal value w_i and the probability is $p_i = 0.5$;
- **level 1 of degradation**: the flow of arc i is $0.5 w_i$ and the probability is $p_i = 0.15$;

- **level 2 of degradation:** the flow of arc i is $0.25 w_i$ and the probability is $p_i = 0.25$;
- **fail:** the flow of arc i is 0 and the probability is $p_i = 0.1$.

Table 7 shows the results. The first, third and fifth column report all the possible values of the flow that can be transmitted between s and t . Each value gives origin to a different terminal leaf in the constructed MDD. The second, fourth and sixth column report the corresponding probabilities value. Since the network has 21 arcs the state space has 4^{21} states, the final MDD encoding the network contains 621813 nodes with a peak of 2904516 nodes as displayed in Table 6b).

5 Conclusion

This paper has shown that MDD can provide a suitable, compact and flexible data structure to handle the problem of the quantitative evaluation of the QoS in weighted multi state probabilistic networks. We have enlarged previous literature in the field by showing that various physical interpretations for the performance attributes assigned to the network elements can be accommodated in the model and the corresponding QoS function can be computed on the network by defining new arithmetic/logic operations for manipulating the MDD. We have illustrated the capabilities of our approach on various benchmarks of rather substantial dimensions. For instance the network of Figure 6a) with four states per arc spans over a state space of 4^{21} states. This approach can be applied to systems of different technological areas like power networks, telecommunication systems, computer systems, where the multi-states can represent different levels of performance.

Acknowledgments

This work has been partially supported by MIUR fund through the PRIN 2007 project under grant 2007J4SKYP.

References

1. Bobbio, A., Terruggia, R.: Binary decision diagram in network reliability analysis. In: 1st IFAC Workshop on Dependable Control of Discrete Systems (DCDS 2007), pp. 57–62 (2007)
2. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26(9), 1124–1137 (2004)
3. Chaturvedi, S.K.: Irredundant subset cut generation to compute capacity related reliability. *Int. J. of Performability Engineering* 3, 243–256 (2007)
4. Ciardo, G., Luttgen, G., Miner, A.S.: Exploiting interleaving semantics in symbolic state-space generation. *Formal Methods in System Design* (2007)

5. Gan, D., Luo, X., Bourcier, D.V., Thomas, R.J.: Min-max transfer capabilities of transmission interfaces. *Int. J. of Electrical Power & Energy Systems* 25(5), 347–353 (2003)
6. Hajiaghayi, M.T., Leighton, T.: On the max-flow min-cut ratio for directed multi-commodity flows. *Theoretical Computer Science* 352(1-3), 318–321 (2006)
7. Hardy, G., Lucet, C., Limnios, N.: K-terminal network reliability measures with binary decision diagrams. *IEEE Transactions on Reliability* 56, 506–515 (2007)
8. Hudson, J.C., Kapur, K.C.: Reliability analysis of multistate systems with multistate components. *IIE Transactions* 15, 127–135 (1983)
9. Jane, C.C., Yuan, J.: A sum of disjoint products algorithm for reliability evaluation flow of flow networks. *European J. of Op. Research* 127(3), 664–675 (2001)
10. Kam, T., Villa, T., Braytonand, R., Sangiovanni-Vincentelli, A.: Multi-valued decision diagrams: Theory and applications. *Multiple-Valued Logic* 4(1), 9–62 (1998)
11. Kolowrocki, K.: On limit reliability functions of large multi-state systems with ageing components. *Appl. Math. Comput.* 121(2-3), 313–361 (2001)
12. Levitin, G., Lisnianski, A.: A new approach to solving problems of multi-state system reliability optimization. *Quality and Reliability Engineering International* 17, 93–104 (2001)
13. Nagamochi, H., Ibaraki, T.: On max-flow min-cut and integral flow properties for multicommodity flows in directed networks. *Information Processing Letters* 31(6), 279–285 (1989)
14. Ramirez-Marquez, J.E., Coit, D.W.: A Monte-Carlo simulation approach for approximating multi-state two terminal reliability. *Reliability Engineering and System Safety* 87(2), 253–264 (2005)
15. Shrestha, A., Xing, L., Dai, Y.: Decision diagram based methods and complexity analysis for multi-state systems. *IEEE Trans. on Reliability* 59(1), 145–161 (2010)
16. Soh, S., Rai, S.: An efficient cutset approach for evaluating communication-network reliability with heterogeneous link-capacities. *IEEE Transactions on Reliability* 54(1), 133–144 (2005)
17. Iowa State University Ames (IA) USA. Meddly decision diagram library, <http://sourceforge.net/projects/meddly/>
18. Xing, L., Dai, Y.: A new decision diagram based method for efficient analysis on multi-state systems. *IEEE Transactions on Dependable and Secure Computing* 6(3), 161–174 (2009)
19. Zaitseva, E., Levashenko, V.: Investigation multi-state system reliability by structure function. In: *DEPCOS-RELCOMEX 2007: Proceedings of the 2nd International Conference on Dependability of Computer Systems*, Washington, DC, USA, pp. 81–90. IEEE Computer Society, Los Alamitos (2007)
20. Zang, X., Wang, D., Sun, H., Trivedi, K.: A BDD-based algorithm for analysis of multistate systems with multistate components. *IEEE Transactions on Computers* 52(12), 1608–1618 (2003)

Comparison between IEC 60880 and IEC 61508 for Certification Purposes in the Nuclear Domain

Jussi Lahtinen¹, Mika Johansson², Jukka Ranta¹, Hannu Harju¹, and Risto Nevalainen²

¹ VTT Technical Research Centre of Finland, P.O. Box 1000, FI-02044, Finland

² Tampereen Teknillinen Yliopisto, Porin yksikkö, P.O. Box 300, FI-28101 Pori, Finland

jussi.lahtinen@vtt.fi, mika.johansson@fisma.fi,
jukka.ranta@vtt.fi, hannu.harju@vtt.fi,
risto.nevalainen@falconleader.fi

Abstract. In the nuclear domain, regulators have strict requirements for safety-critical software. In this paper requirements in three documents (two software standards and the Common Position of nuclear domain regulators) were compared. The aim of the work was to find out how these requirements compare to each other in terms of strictness and scope, and to evaluate the usefulness of the documents for certification purposes. Another goal was to determine whether it is possible to choose only one of the standards as the basis of software certification. The nuclear domain software standard IEC 60880 provides requirements for the purpose of achieving highly reliable software. The standard is similar to the part 3 of IEC 61508 standard in the sense that it covers requirements for all software lifecycle activities. The Common Position document "Licensing of safety critical software for nuclear reactors" states the requirements from the perspective of European nuclear regulators. The comparison was twofold. First, the absolute 'shall' requirements of a few key themes were extracted from all three documents. The strictness of these requirements was analyzed against each other. Second, to evaluate the documents' usefulness for certification, the extent in which these themes were covered by each document was analyzed by expert judgment. The main result was that the use of IEC 60880 alone is not sufficient for software certification.

Keywords: IEC 61508, IEC 60880, nuclear domain, safety requirements, safety standards, safety-critical software.

1 Introduction

Certification of software products by independent evaluation has been practiced in the software industry since early 1990s, especially in Europe and later in the United States. In Finland, a type acceptance certificate is required mainly in highest safety classes of I&C equipments and systems in NPP, and recommended in lowest safety classes. In the research project "Certification facilities for software (CERFAS)", the objective is to develop facilities for flexible, supported, commercially exploitable, high quality Software Certification Service, SCS, able to certificate safety critical software for the demands in Finnish nuclear area.

Most important nuclear specific requirements are standards, which include requirements for safety critical systems and software. The most relevant for software safety is IEC 60880 [2]. The other main reference is the generic functional safety standard IEC 61508 [1], whose part 3 discusses software. Nuclear domain requirements can also be found in other documents. One example is the Common Position of nuclear regulators [11].

The relationship between these documents has not yet matured, and needs to be defined. It is unclear how these standards compare to each other in terms of strictness. In the article [14], Smith and Simpson claim that it is generally accepted that the category A discussed in IEC 60880 corresponds to SIL 3 of IEC 61508. In draft versions of IEC 60880 it was even said that IEC 60880 corresponds to minimum of SIL 4 defined in IEC 61508. Yet it is not evident whether IEC 60880 is at all comparable to the safety integrity levels of IEC 61508. The first release of IEC 60880 in 1986 used to be considered a very strict standard. Nowadays, almost any embedded critical software satisfies its normative annexes A and B.

Another uncertainty is the relation of the standards (IEC 61508 and IEC 60880) to the regulatory requirements and interpretation of strictness in these standards. Additionally, each country may have national guides or standards of their own. For example, The Radiation and Nuclear Safety Authority (STUK) in Finland has YVL Guides. Can the standards fulfill the regulator common position requirements individually / together? Can IEC 60880 alone be used for software certification in the nuclear domain? Can the standards be applied alone to specify the strictest requirements such as diversity?

To answer these questions a comparison of these three documents [1, 2, 11] was conducted. The comparison focused completely on software requirements. Not all software requirements were covered in the comparison, but requirements from a few key themes concerning software were selected. The comparison was twofold. The first phase of the comparison examined only the absolute 'shall' requirements in each document. Each requirement was mapped to requirements in the other documents. The level of compliance of the similar requirements was judged using terms fully compliant, partially compliant, and non-compliant. The non-compliant and partially compliant clauses require something the other documents do not require. Based on these requirements it was determined whether one document was in this sense significantly stricter than the others, and whether one of the standards encompasses the other.

The second phase of the comparison was more subjective. Strictly looking at the absolute 'shall' requirements tells little about applicability of the requirements. This is why the usefulness for certification of the requirements provided by each document was analyzed.

The annexes of the standards are so different that a full comparison is not meaningful. Thus, our comparison mainly focuses on the body texts of the standards, and does not include some of the strict requirements, which are in annexes only.

Section 2 presents the IEC standards and the Common Position of nuclear regulators. Section 3 discusses the usefulness of requirements for certification purposes. The comparison method is discussed in section 4. Results are given in section 5, and conclusions are presented in section 6.

2 Nuclear Domain Requirements

There are several IEC, IEEE and IAEA standards that are relevant in the nuclear domain. The primary users of IAEA standards are regulatory bodies and they are also used by organizations that design, manufacture and operate nuclear facilities. The IEC standards relevant to this work are illustrated in Figure 1. The safety categorization of I&C functions and systems is given in IEC 61226 [3]. The document guides the classification of functions into categories (A, B, C, not important to safety) that designate the importance to safety of that function.

IEC 61513 [4] is the system level standard for nuclear power plants I&C systems. The standard covers I&C systems performing functions in categories A, B, or C as defined in IEC 61226. The standard covers implementation of the systems based on hard-wired and computer-based equipment. Requirements for both the total architecture of the I&C systems and the individual systems are given. The standard should be used in conjunction with other IEC standards so that IEC 60880 is used with category A software, IEC 62138 [5] is used with software in categories B and C, and IEC 60987 [6] is used with hardware.

Categorisation: IEC 61226	
Systems: IEC 61513	
Software	Hardware
Category A: IEC 60880	IEC 60987
Categories B and C: IEC 62138	

Fig. 1. Main IEC nuclear standards

2.1 IEC 60880

IEC 60880 provides requirements for the software of computer-based I&C systems of nuclear power plants performing category A functions as defined in IEC 61226. The standard provides requirements for the purpose of achieving highly reliable software. The standard is similar to the IEC 61508 standard in the sense that it covers requirements for all software lifecycle activities.

2.2 The General IEC 61508 Standard

IEC 61508 is an umbrella standard concerning basic functional safety issues in all kinds of industry. The standard sets out a generic approach for all safety lifecycle activities of electrical/electronic/programmable electronic safety-related systems. The part 3 of IEC 61508 covers the software requirements of such safety-related systems. Requirements for all software lifecycle activities are considered. In addition, requirements for support tools such as development and design tools, translators, testing tools and configuration management tools are provided. IEC 61508 also defines various recommendations of techniques for software design, analysis or testing.

2.3 The Common Position of Nuclear Regulators

The Common Position of nuclear regulators [11] is a result of the work of a group of regulator and safety authorities' experts. The document identifies the consensus and common technical positions on several important licensing issues in the design and operation of computer based safety systems in nuclear power plants. The document offers guidance to regulators and manufacturers, and is intended to be useful as a reference in safety cases and demonstrations of safety of software based systems.

2.4 Comparability of IEC 61508, IEC 60880 and the Common Position

The documents' scopes are quite uniform. Because of this it is quite easy to locate corresponding requirements in each document. However, the assumptions and approaches in the standards are different. IEC 60880 often has an emphasis on the required matters, or tasks, whereas IEC 61508 discusses the methods that can be used to meet these requirements. For instance, where IEC 60880 requires a specific property to be verified, IEC 61508 requires the use of a specific verification method. In addition, it is characteristic for IEC 61508 to use rather generic phrasing. It typically requires completeness, correctness and consistency. On the other hand, IEC 60880 focuses more on how these attributes are attained.

3 Usefulness of Requirements

In the safety domain, explicit safety cases are increasingly required by law, regulations and standards. Safety cases are also used in certification. The safety case methodology [15] was established in industries dealing with functional safety of computerized automation.

Several different types of safety cases exist. The University of York (Adelard) has been active in safety case research. Adelard defines a safety case as a document body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in given environment [15]. A typical safety case is defined in terms of three elements:

- Claims about properties of the system
- Evidence used as the basis of the safety argument
- Argument that links the evidence to the claims via a series of inference rules

Claims are statements or objectives that need justification or demonstration, and they provide the major structure of the safety case.

In order to be useful for certification, and to the demonstration of safety, a requirement has to be compatible with the safety case methodology. In this sense a good requirement is such that it can be used (with little modification) as a claim in the safety case. For certification purposes it is useful if the requirement of a standard also demands detailed evidence that can be looked for.

Another way for a standard to contribute to certification is to provide some valuable information that is otherwise useful in the construction of a safety case. This includes details that need to be paid attention to, available techniques, acceptable practices et cetera. This kind of information is necessarily not required, but is usually an annex of the standard document.

There is no objective way to evaluate the usefulness of a requirement. In this comparison the subjective judgment is based on the issues of this section.

4 Comparison of Software Safety Requirements

The main objectives of the comparison were to identify the differences in the documents' strictness and scope, and to evaluate their usefulness to certification. To attain these goals a comparison schema presented in Figure 2 was created that visualizes how each document was compared against the others. The clauses in the common position document were compared against both IEC 60880 and IEC 61508, and IEC 60880 and IEC 61508 were compared against each other. The standards' requirements were not compared against the Common Position requirements because of the nature of the Common Position, i.e., the Common Position document is not a standard.

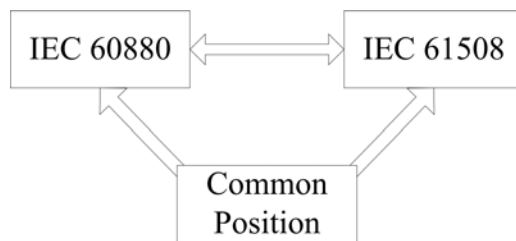


Fig. 2. The comparison schema

4.1 Selection of Themes

Because of the extent of the three documents, only requirements from a few key themes were selected as the focus areas of the comparison. The themes were chosen so that the requirements could be examined from several important viewpoints. The comparison was limited to software. The part 2 of IEC 61508, for example, was not

examined. The themes include life-cycle phases, software specific issues, technical issues, and generic system-level issues. The selected themes were:

1. Requirements specification (Life-cycle phase)
2. Design (Life-cycle phase)
3. Verification (Life-cycle phase)
4. Traceability (System)
5. Independent assessment: Independence of safety assessment and V&V (System)
6. Pre-developed software (Software)
7. Tool selection (Software)
8. Formal methods (Technical)
9. Software diversity and software common cause failure (Technical)

The software life-cycle is rather well-established, especially in the safety-critical context. The life-cycle in the three documents is on fundamental parts identical. Requirements in themes 1, 2 and 3 are essential in the development of safety-critical software.

In addition to the life cycle phases, the Common Position document identifies generic licensing issues that are not specific to a certain life cycle phase. Themes 5, 6, 8 and 9 represent some of these issues.

The compared documents focus mainly on safety-critical software. Thus, the specific requirements for safety-critical software were emphasized. Themes specifically related to safety-critical software are 4, 6, 7, 8 and 9.

4.2 The Comparison Method

The comparison had two separate phases. Both phases examined only requirements relevant to the themes chosen in section 4.1. The purpose of the first phase was to identify the differences in the documents' strictness and scope. The second phase aimed to evaluate the usefulness of the contents to certification.

The intention of the comparison was to compare the three documents especially in terms of strictness. This is why, in the first phase of the comparison, only absolute 'shall' clauses were examined. The selected themes were gone through one by one, and all relevant 'shall' clauses were collected from each document. The collected requirements were then compared against requirements in the other two documents, and requirements with a similar or equivalent meaning were mapped to each other. The level of similarity of the found requirements was graded using three categories: fully compliant, partially compliant, and non-compliant. Using this method, the coverage of the standards was estimated. Additionally, the percentage of one document's requirements covered by another document was calculated.

In the second phase of the comparison the objective was to judge each document's capability to provide a basis to certification. The usefulness of each requirement for certification can be judged subjectively. A requirement can be considered useful if it clearly requires some evidence of its fulfillment that the certification authority could

check. Also clauses providing some advantageous information can be considered useful to certification. The contents and meaningfulness of requirements was not evaluated because of the certification perspective of the comparison.

4.3 Other Similar Comparisons

Fusani [10] analyses and compares several international standards for safety-critical systems depending on software. Standards related to functional safety of software are selected from various domains. In addition, software engineering standards are selected. The standards, including IEC 61508 and IEC 60880 are compared against each other with respect to several criteria.

Pre-developed software, or SOUP (software of uncertain pedigree) in IEC 61508-compliant safety-related applications is examined in [13]. The approach to SOUP software is compared against approaches between IEC 61508, IEC 60880, FDA 1252 [7], and Def Stan 00-55 [8] and 00-56 [9]. The approach of the Common Position is also discussed, and the requirements for SOUP software are listed.

Safety lifecycles are compared in [12]. The paper compares the safety lifecycles of several software safety standards including IEEE standards, IEC 61513, IEC 61508 and IEC 60880. Aside from safety lifecycles, individual requirements are not examined.

5 Results

Section 5.1 lists the differences of the three documents. The differences in every theme are gone through. Section 5.2 discusses the coverage of absolute ‘shall’ clauses in each document. The numbers are derived from the number of clauses not having an equivalent clause in the other document. In this section, the content of the clauses itself is not discussed. Finally, in section 5.3 the usefulness for certification of each document is evaluated subjectively.

5.1 Differences between the Documents

During the comparison it became evident that the documents emphasize different aspects, and not all requirements overlap. The main differences are elaborated in Table 1.

5.2 Coverage of Strict Requirements

Figure 3 presents the percentage of Common Position ‘shall’ requirements covered in IEC 61508. Similarly, the percentage of Common Position ‘shall’ requirements covered in IEC 60880 are shown in Figure 4. The coverage is given both for full and partial compliance. The coverage was calculated for each theme. The figures show for example that IEC 60880 fully covers 81% of the Common Position requirements for common cause failure and diversity (Theme 9), while IEC 61508 fully covers only 31% of the same requirements.

Table 1. Main differences between the documents

	IEC 60880	IEC 61508	The Common Position
Requirements specification	Requires that what the software must not do is identified.	Explicitly requires specific safety aspects	No significant differences.
Design	Requirements for code generators. Restricts operational software use.	Explicitly discusses software architecture design, and separation and independence of safety functions.	More specific requirements for separation of safety functions, and fault tolerance.
Verification	Independent verification team. Detailed fault resolution procedures.	Traceability of verification. Testing and verification methods.	Role of the regulator. Test coverage requirements.
Traceability	Traceability is required, but the requirements are written in an implicit way. Tests traceability is not as strong as in 61508.	Well-defined explicit traceability chain required from safety needs to validation.	Emphasis traceability between requirements and tests, and between requirements and implementation. Additional requirements as "should" statements.
Independence of safety assessment and V&V	Explicitly requires independence for verification and validation	Functional safety assessment performed by an independent party (varies from person to organization).	Requirements for independent functional safety assessment and independent V&V.
Pre-developed software	Operational experience can be only used to compensate weaknesses found in software during the evaluation process.	Many ways to evaluate pre-developed software products.	Follows, and also directly references, IEC 60880 in many ways.
Tool selection	Covers all but T1 of IEC 61508 off-line tools such as text editors. More analysis and information on tools.	Requirements for all tools, for example: -Online tools -T1: text editors -T2: testing tools -T3: compilers	Tool experience feedback must be maintained and updated by the vendor
Formal methods	Appropriate techniques (may include formal methods) shall be selected.	Appropriate formal / semi-formal methods shall be selected. Introduction to the use of the methods.	Detailed requirements for formal methods use.
Software diversity	Emphasizes analysis. More requirements and information in general.	Techniques not mentioned in IEC 60880.	Requirements similar to IEC 60880.

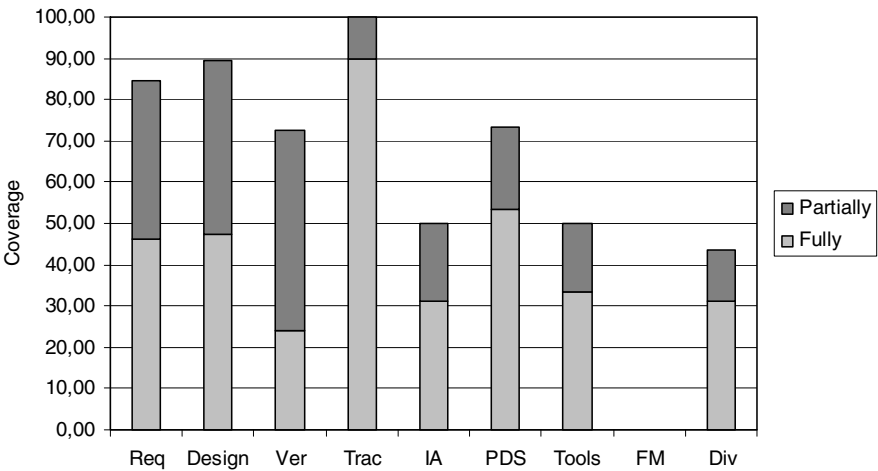


Fig. 3. Percentage of Common Position requirements covered in IEC 61508

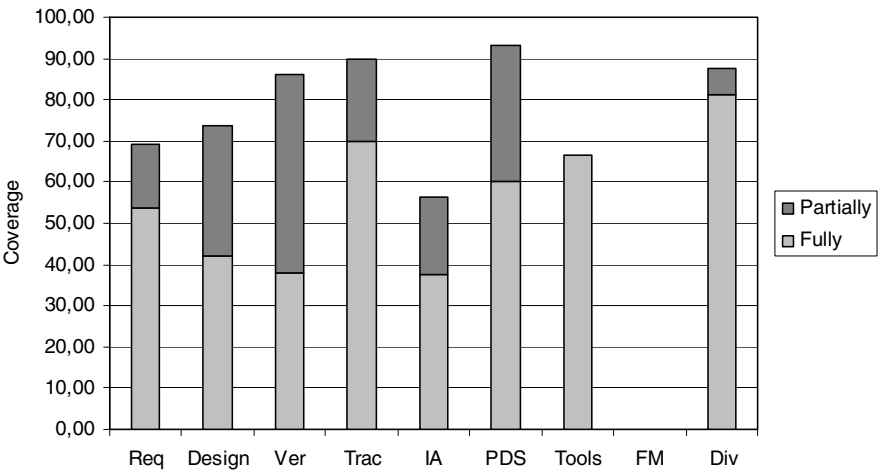


Fig. 4. Percentage of Common Position requirements covered in IEC 60880

Figure 5 illustrates the percentage of Common Position requirements covered in by both standards combined, i.e., the union of requirements.

Coverage of shall requirements for each theme was estimated for both standards. The coverage numbers are in Figure 6. The coverage presented in the figure was calculated so that full coverage is the minimal set of requirements in all three documents that covers all requirements. The coverage estimate gives all requirements the same weight, i.e., the contents of the requirements are not given any importance. The coverage estimate was calculated for each theme.

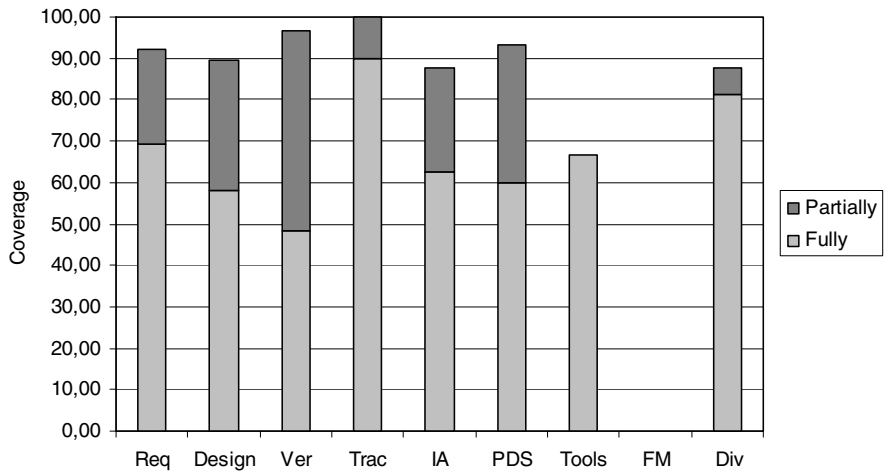


Fig. 5. Percentage of Common Position requirements covered in IEC 60880 and IEC 61508 combined

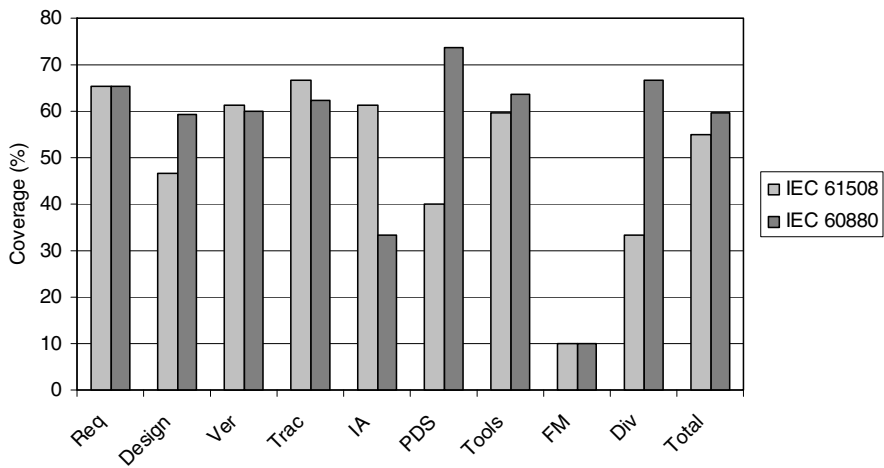


Fig. 6. Coverage of shall requirements in IEC 61508 and IEC 60880

5.3 Usefulness for Certification Purposes

Requirements of each theme were gone through and their usefulness for certification purposes was evaluated. Figure 7 shows the number of clauses in each document contributing to certification.

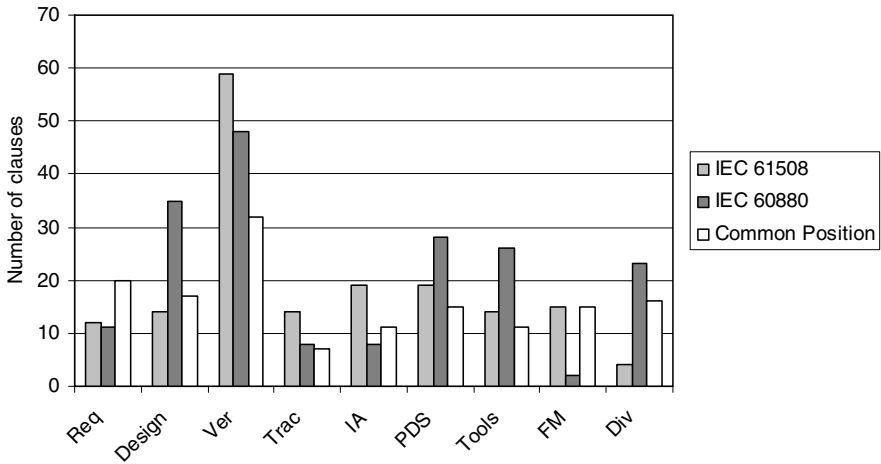


Fig. 7. Contribution to certification of IEC 61508, IEC 60880 and the Common Position, for each theme

6 Conclusions

Our work compares the requirements in three documents relevant to I&C safety in the nuclear domain. The strictness and usefulness of requirements could be evaluated because of the similarities in the documents' scopes and structure. However, their comparison is difficult even with respect to some fundamental concepts. First of all, the safety integrity levels of the IEC 61508 can not be compared against IEC 60880 in terms of strictness. For software, the SIL level is mostly based on the selection of techniques and measures used. IEC 60880 does not have a similar list of recommended techniques, although some techniques are mentioned. The correspondence between IEC 60880 and SIL 4 could be determined by going through the annexes of both standards.

It must be noted that only requirements stated in the examined documents were compared. Often standards reference other documents and standards, and some standards should be used together with other standards. For example, some relevant requirements missing from IEC 60880 can be found from IEC 61513. The Common Position also references other documents including IEC 60880. Referenced documents were generally not part of the comparison.

Some engineering concepts and life-cycle phases (e.g. requirements specification, use of formal methods) have greater relevance on achievement of safety than others. Thus, use of weights in an overall comparison would lead to more meaningful results from the perspective of practical safety. Our interest, however, was a direct comparison from a certification perspective. Thus, the themes of the comparison were considered separately in our work.

When requirements are looked at in a strict way, the requirements of IEC 61508 and IEC 60880 cover each other only partly. In fact, the standards overlap quite

equally. Both cover ca. 63% of the other standards' requirements. In most themes neither of the documents covers all of the requirements in the other document.

IEC 61508 coverage is greater than IEC 60880 coverage in the areas of requirements specification, verification, traceability and independent assessment. IEC 60880 requirements cover the areas of design, tool selection, pre-developed software, and software diversity more rigorously.

When the requirements of the standards are compared against the Common Position requirements, IEC 60880 is better for requirements in software diversity, tool selection, pre-developed software and verification. IEC 61508 on the other hand excels when requirements for requirements specification, design, and traceability are considered. When standards are compared to each other, it seems that IEC 60880 covers design requirements, and IEC 61508 covers verification requirements. Comparison against the Common Position suggests the contrary. Hence, the Common Position requirements probably resemble the IEC 61508 requirements for design, and IEC 60880 requirements for verification.

When the Common Position is compared against the union of the two standards' requirements the coverage of the standards is much greater. For most themes, a considerable number of the Common Position requirements can be covered. The results suggest that in order to better satisfy regulatory requirements the standards should be used in conjunction. However, the requirements for formal methods are non-existent in the standards. The Common Position has several strict requirements for their use.

The second phase of the comparison mostly backs up the observations of the first phase. IEC 61508 is most useful in the areas of verification, traceability, independent assessment, and formal methods. IEC 60880 provides more useful information on design, tool selection, pre-developed software and software diversity issues. The results indicate that for certification purposes, the Common Position is especially useful in the areas of requirements specification, formal methods, and software diversity.

When our results in the second phase of the comparison (Figure 7) are compared against results from a similar study [10], it can be seen that the results are alike for requirements in independent assessment, pre-developed software, tool selection, and diversity. Our results would, however, indicate IEC 61508 preference in verification, traceability and formal methods, when it comes to usefulness for certification. The similar study, however, did not assess the usefulness for certification, but the quantity and content of information.

All in all, IEC 60880 is superior when it comes to requirements for software diversity, pre-developed software and tool selection, while IEC 61508 is better suited for requirements specification, traceability, formal methods, and information on testing and verification. To conform to the Common Position, independent assessment requirements should be collected from both standards. Special consideration is required for formal methods use, since the area is not adequately covered in IEC 61508 or IEC 60880.

The comparison suggests that while IEC 60880 and IEC 61508 are fairly similar, both should be used to achieve regulatory satisfaction. For certification purposes, the use of IEC 60880 alone is not sufficient.

References

1. International Electrotechnical Commission, IEC 61508, Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems – Part 3: Software requirements, 65A/550/FDIS, IEC (2009)
2. International Electrotechnical Commission, IEC 60880, Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions, IEC (2006)
3. International Electrotechnical Commission, IEC 61226, Nuclear power plants – Instrumentation and control systems important to safety – Classification of instrumentation and control functions, IEC (2005)
4. International Electrotechnical Commission, IEC 61513, Nuclear power plants – Instrumentation and control for systems important to safety – General requirements for systems, IEC (2001)
5. International Electrotechnical Commission, IEC 62138, Nuclear power plants - Instrumentation and control important for safety - Software aspects for computer-based systems performing category B or C functions, IEC (2004)
6. International Electrotechnical Commission, IEC 60987, Nuclear power plants - Instrumentation and control important to safety - Hardware design requirements for computer-based systems, IEC (2007)
7. FDA 1252, Off-The-Shelf Software Use in Medical Devices, US Department of Health and Human Services, Food and Drug Administration, Center for Devices and Radiological Health (September 9, 1999)
8. DS 00-55, Requirements for Safety Related Software in Defence Equipment, UK Defence Standard 00-55, Parts 1 and 2, Issue 2 (August 1, 1997)
9. DS 00-56, Safety Management Requirements for Defence Systems, UK Defence Standard 00-56, Parts 1 and 2, Issue 2 (December 13, 1996)
10. Fusani, M.: Examining Software Engineering Requirements in Safety-Related Standards. In: Proceedings of DeSSerT (Dependable Systems, Services and Technologies) 2009, Kirovograd, Ukraine (April 22-25, 2009)
11. European Commission's Advisory Experts Group, Nuclear Regulators Working Group, Licensing of safety critical software for nuclear reactors - Common Position of seven European nuclear regulators and authorized technical support organizations, Revision 2010 (2010)
12. Lee, J.-S., Lindner, A., Choi, J.-G., Miedl, H., Kwon, K.-C.: Software Safety Lifecycles and the Methods of a Programmable Electronic Safety System for a Nuclear Power Plant. In: Górski, J. (ed.) SAFECOMP 2006. LNCS, vol. 4166, pp. 85–98. Springer, Heidelberg (2006)
13. UK Health and Safety Executive (HSE) commissioned research. Methods for assessing the safety integrity of safety-related software of uncertain pedigree (SOUP). Report No: CRR337 HSE Books 2001 (2001) ISBN 0 7176 2011 5, http://www.hse.gov.uk/research/crr_pdf/2001/crr01337.pdf
14. 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)
15. Bishop, P.G., Bloomfield, R.E.: A methodology for safety case development. In: Safety-Critical Systems Symposium, Birmingham, UK (February 1998)

Deriving Safety Cases for Hierarchical Structure in Model-Based Development

Nurlida Basir¹, Ewen Denney², and Bernd Fischer¹

¹ ECS, University of Southampton, Southampton, SO17 1BJ, UK
(nb206r,b.fischer)@ecs.soton.ac.uk

² SGT / NASA Ames Research Center
Moffett Field, CA 94035, USA
Ewen.W.Denney@nasa.gov

Abstract. Model-based development and automated code generation are increasingly used for actual production code, in particular in mathematical and engineering domains. However, since code generators are typically not qualified, there is no guarantee that their output satisfies the system requirements, or is even safe. Here we present an approach to systematically derive safety cases that argue along the hierarchical structure in model-based development. The safety cases are constructed mechanically using a formal analysis, based on automated theorem proving, of the automatically generated code. The analysis recovers the model structure and component hierarchy from the code, providing independent assurance of both code and model. It identifies how the given system safety requirements are broken down into component requirements, and where they are ultimately established, thus establishing a hierarchy of requirements that is aligned with the hierarchical model structure. The derived safety cases reflect the results of the analysis, and provide a high-level argument that traces the requirements on the model via the inferred model structure to the code. We illustrate our approach on flight code generated from hierarchical Simulink models by Real-Time Workshop.

Keywords: Model-based software development, automated code generation, formal proofs, formal analysis, safety case, automated theorem proving.

1 Introduction

Model-based development and automated code generation are increasingly used for actual production code, in particular in mathematical and engineering domains. For example, NASA's Project Constellation uses Real-Time Workshop (RTW) for its Guidance, Navigation, and Control (GN&C) systems and subsystems. However, since code generators are typically not qualified, there is no guarantee that their output is correct or even safe, and additional evidence of its safety is required. In previous work [5], we have thus constructed safety cases [19] from information collected during a formal verification of the generated code. We also have constructed safety cases that correspond to the formal proofs found by automated theorem provers of the verification conditions, and reveal the underlying proof argumentation structure and top-level assumptions [6].

This paper is a continuation of our previous work, but here we systematically derive safety cases that argue along the hierarchical structure in model-based development.

A safety case is a structured argument, supported by a body of evidence, which provides a convincing and valid justification that a system is acceptably safe for a given application in a given operating environment [19]. In the Goal Structuring Notation (GSN) [13], which we use as technique to explicitly represent the logical flow of a safety argument, the main construction elements of a safety case are goals (which are the safety claims to be met by the system), strategies (which describe how a claim is addressed by evidence or further subgoals), evidence, and assumptions. In our work, the safety cases are constructed mechanically using a formal analysis, based on automated theorem proving, of the automatically generated code. Goals are thus given by the formal safety requirements on the model, which express as logical formulas the properties that the (software sub-) system's output signals must satisfy for the (overall) system to be safe. Strategies are the high-level steps of the formal analysis (e.g., decomposing the set of requirements, or decomposing the system into components) while the evidence comes from the low-level proofs of the verification conditions. Assumptions are logical formulas which express the properties that the input signals must satisfy for the (overall) argument to be valid; they are thus dual to requirements and, hence, goals.

We illustrate our work using the verification of two safety requirements for a spacecraft navigation system that was generated from a Simulink model by Real-Time Workshop [3]. The requirements determine the interface between the software system safety cases (where they are the root elements) and the subsystem safety case (where they are leaf nodes). Each requirement induces a verified abstraction or *slice* of the system architecture. The formal analysis recovers the hierarchical structure of these slices from the code and identifies requirements that rely on any externally given assumptions. This enables us to identify how the system safety requirements are broken down into low-level component requirements and distributed over the system components, and thus also to identify where the requirements are ultimately established, resulting in a hierarchy of requirements that is aligned with the hierarchy of the components.

We use safety cases to reflect the results of the program analysis, and provide a high-level argument that explains how the system slices establish the corresponding safety requirements. The safety cases help tracing the safety requirements from the model via the inferred system structure to the code, thus providing independent assurance of both model and code. They also provide a traceable safety argument that shows in particular where the code, subsystem, and system depend on any internal and external assumptions. We believe they highlight the claims, key safety requirements, and evidence that are required to understand and trust generated code, which is essential for the use of code generators in safety-critical applications.

2 Background

2.1 Model-Based Software Development

Model-based software development comprises a number of techniques that focus on creating and transforming domain-specific abstractions or *models* rather than algorithms or even code. In *model-based design* [3,18], mathematical or, more often, visual methods are used to create an initial model of the system design. It is commonly

used in the control systems domain, where block diagrams provide an accepted notation. Blocks can represent arbitrary computations and can be nested hierarchically, which helps counter system complexity. They are connected by wires that represent the flow of signals through the system. A number of academic and commercial tools support model-based design in this domain. We focus on MathWorks Simulink [3], which is used by many NASA projects for at least some of their modeling and code development, particularly for GN&C problems. Simulink comes with a large library of standard modeling blocks that provide mathematical operations and signal routing suitable for control systems and complex operations.

Model-based code generation [16,18] complements model-based design, and translates specifications in the form of a model into a program in a high-level programming language such as C or ADA. The translation process can be organized as a sequence of model transformations, where the last model is equivalent to the program. The final source code generation can then be realized with a simple template engine. Here we focus on a commercial generator, MathWorks Real-Time Workshop Embedded Coder [3]. Real-Time Workshop generates ANSI/ISO compliant C and C++ code from MathWorks Simulink and Stateflow models. Embedded Coder adds various features, such as optimization, which are useful for generating C code tuned for embedded devices.

2.2 Formal Program Analysis Using AutoCert

The techniques described here are based on the AUTOCERT code analysis tool [9], which takes a set of requirements, and formally verifies that the code satisfies them. AUTOCERT can verify execution-safety requirements (e.g., array bounds), as well as individual mathematically specified requirements. AUTOCERT thus supports certification by formally verifying that auto-generated code is free of certain safety violations and complies with domain-specific safety requirements as those mentioned.

AUTOCERT follows the Hoare logic approach to verification, which needs *annotations*, i.e., logical assertions of program properties, at key locations in the code. These annotations are constructed automatically by a post-generation inference phase that exploits the idiomatic nature of auto-generated code and is driven by a generator- and domain-specific set of idioms. The inference algorithm builds an abstracted control-flow graph (CFG), collapsing the code idioms into single nodes. It then traverses the CFG from *use* nodes (where a requirement must hold) backwards to all corresponding *definitions* (where the relevant properties are established) and annotates the statements along the paths as required [9]. The definitions typically correspond to model blocks (more precisely, to parts of the code implementing a block), which can use assumptions on the properties of their input signal to establish the requirement. Hence, the inference algorithm must recurse over the variables corresponding to the input signals, derive the assumptions, and establish them as new requirements. This chain of requirements on variables and their definitions constitutes the backbone of our safety argument. As byproduct, the inference derives the component interfaces (i.e., the requirements placed on them, and the assumptions made by them) as well as the system's overall assumptions, which need to be established by its context. A verification condition generator (VCG) processes the annotated code, feeding a set of verification conditions (VCs) into an automated theorem prover (ATP); their proofs guarantee that the code satisfies the

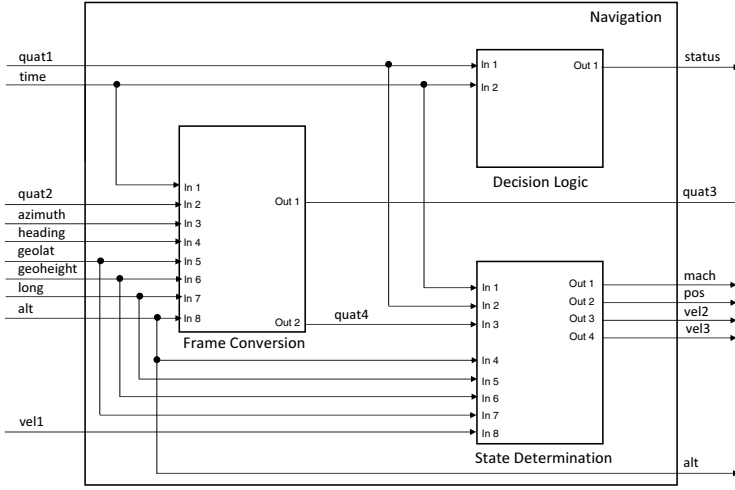


Fig. 1. High-level Architecture of Navigation System

requirements and also validate the definitions identified by the analysis, and thus the derived architecture. In the safety case, the proofs serve as evidence.

During the course of analysis, AUTOCERT records various facts, such as the locations of uses and definitions, which are later used as input to the safety case generation process. Here, we extended the existing mechanism to record additional information from which we can reconstruct the system architecture slices.

2.3 Guidance, Navigation, and Control Systems

Spacecraft are typically decomposed into a number of different systems such as the power, thermal protection, or guidance, navigation, and control (GN&C) systems [22]. The GN&C system is a necessary element of every spacecraft. Here, we focus on the Navigation (sub-) system within the GN&C system. It is used to determine a spacecraft's orientation and position, which is challenging from a safety perspective, due to its complex and mathematical nature. We give a brief, simplified description of the system where we also have changed the names of components and signals from the original.

Navigation (see Fig. 1 for its architecture) takes several input signals, representing various physical quantities, and computes output signals representing other quantities, such as Mach number, angular velocity, position in a specified frame of reference, and so on. Signals are generally represented as floating point numbers or as quaternions and have an associated physical unit and/or frame which correctness are critical to the safety of the system. However, the units and frames are usually not made explicit in the model, and instead are expressed informally in comments and identifiers.

Navigation is comprised of three components, a decision logic that computes a status value irrelevant to the requirements we consider here, a frame conversion, and a state determination. **Frame Conversion** first converts the frames of the incoming signals from a vehicle-based coordinate system to an earth-based coordinate system. The transformations of the coordinate systems are done by converting quaternions to direction cosine matrices (DCMs), applying some matrix algebra, and then converting them back to quaternions [20]. **State Determination** then performs the calculations to determine the vehicle state (e.g., position, attitude, and attitude rate) from these signals. It is defined in terms of the relevant physical equations. Note that there are no individual blocks within Navigation, but only within the components and thus all computation happens there.

3 Deriving Safety Cases from the Formal Analysis of Hierarchical Structure

While Leveson et al. [14] rightly argue that a formal verification of software against its requirements does not guarantee safety, it is important to note that the safety requirements are not the same as the software requirements specification, even if many requirements from a software requirements specification do impact safety. Instead, in this work, we assume that the safety requirements have been established independently, for example by a hazard analysis of the overall system, and so take them as given.

Here we use requirements on the GN&C as driving example, since the GN&C is clearly safety-critical, and maintenance of the correct navigation state is therefore safety-critical. In particular, we require that the navigation state be represented in the correct coordinate frames, where “correct” has been independently determined.

3.1 Formalization of the Requirements

We illustrate our work using the results of the formal verification of two safety requirements for the code generated from the Simulink model of the above navigation system:

- (1) The system shall compute a quaternion representing a transformation from the Earth-Centered Inertial (ECI) frame to the body fixed frame in signal $quat_3$, and
- (2) The system shall compute a velocity in the ECI frame in signal vel_2 .

Since we are working with a formal, logic-based analysis framework, we need to formalize these requirements using a domain theory:

- (1') $quat_3 :: quat(ECI, Body)$.
- (2') $vel_2 :: vel(ECI)$.

Here, ECI and Body are constants denoting the respective frames, quat and vel are functions denoting transformations of or quantities in those frames, and $::$ is a predicate that asserts that the signal represents a transformation between (resp. quantity in) the required frame(s).

Obviously, the actual formalization of the safety requirements themselves is safety-relevant: a wrong formalization can invalidate the assurance provided by the proofs [4,15]. It thus needs to be called out and justified in the safety case.

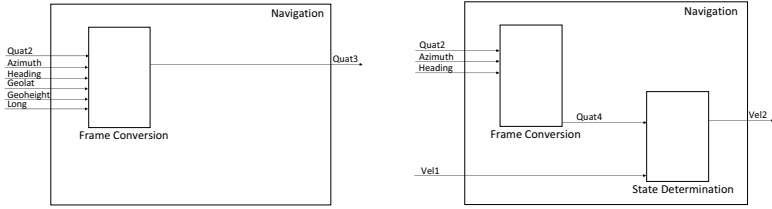


Fig. 2. Architecture Slices Recovered for Example Requirements

3.2 Architecture Recovery

In order to certify the requirements on a system, and to build a comprehensible safety case, we need to know where in the system they are established, and which parts of the system contribute to them. In the system architecture (Fig. 1) we can see that the first requirement should be established by Frame Conversion, since the signal $quat_3$ comes straight out of that component (and similarly for vel_2 and State Determination in the case of the second requirement). However, this view is too simplistic. First, without looking inside the component models it is not clear whether the requirement is indeed established within a component, or simply passed through (cf. for example alt in Navigation), and which of the component's input signals (if any), or more precisely which assumptions on them, are used in establishing the requirement. However, simply expanding the component models destroys the hierarchical structure of the system. More importantly, the safety of the system ultimately depends on the safety of the code rather than the model, but because we cannot trust the code generator to translate the model correctly we cannot derive any trust from the model.

Instead, we analyze the code and recover the slice of the system architecture that is relevant to a given safety requirement. We record when the analysis enters resp. leaves a component (implemented by RTW as a parameter-free procedure), and then remove the part of the requirements-definition chain that is contained within the component. The key to obtaining precise architecture slices is to identify situations in which the control flow just passes through a component, without encountering a definition. In these cases, we can ignore the component altogether. We then assemble the slices from the signals involved in the recorded requirements-definitions chains and from the retained component.

Fig. 2 shows the architecture slices recovered for both requirements. In both cases, the irrelevant Decision Logic component has been removed by the analysis. For the first requirement, it has further identified that $Quat_3$ is unaffected by the call to the State Determination procedure, and consequently removed that component as well. For the second requirement, the analysis has identified $Quat_4$ as the (global) variable through which the two components communicate. In addition, although not shown in Fig. 2, it has derived the property placed as an assumption on this variable by State Determination, i.e., $Quat_4 :: quat(NED, Body)$. This becomes a subordinate requirement to the

original safety requirement, reflecting the hierarchical model structure. The requirements hierarchy is completed by the assumptions placed on the variables Vel_1 and $Quat_2$ corresponding to the components' input signals.

The property derived for $Quat_4$ also becomes part of the interfaces of both components that are connected through this link, as assumption on the State Determination and as safety requirement on Frame Conversion. By regrouping the analysis results by component rather than by original safety requirement, we thus obtain full component interfaces. They give a complete functional specification of the component, including all assumptions, as far as it is required to satisfy the given system-level safety requirements. The interfaces also serve as starting point for verifying the components independently, hence allowing a compositional (and therefore scalable) verification.

The recovered system architecture and requirements hierarchy already constitute a core safety argument: Navigation satisfies the safety requirement (2') if the components Frame Conversion and State Determination satisfy their respective interfaces, and the requirements for Vel_1 , $Quat_2$, and $Quat_4$ hold. This argument can serve as blueprint for a full-fledged safety case. In addition, the derived component interfaces serve as starting points for the construction of independent safety cases for the components, yielding a hierarchy of safety cases that is aligned with the system's hierarchy of models.

3.3 Arguing from System-Level Safety Requirements to Component-Level Safety Requirements

The upper part of the safety case argues the safety of the method of formal reasoning that we use but also points out the important provisos that we abstract away from real-time, and numerical issues. This is a straightforward modification of our previous work on programs without hierarchical system structure (see Fig. 3 Tier I: Explaining the Safety Notion in [5]). Here, we thus focus on the lower part of the safety case that explains that, and how, the generated source code Nav.cpp satisfies the given safety requirements by providing formal proofs as evidence (see Fig. 3).

The key argument strategy here is to argue over each individual requirement that contributes to the program safety. The additional information that is required for the strategy to be understood and valid is identified and explained. This concerns the independent validity of the safety requirements and the logical consistency of the assumptions. We thus assume that no safety requirement is available for use as a (logical) assumption in the safety proofs, which prevents vacuous proofs based on mutually recursive dependencies between requirements and assumptions. We further assume that the given and derived assumptions together are consistent, again to prevent vacuous proofs. Each assumption is justified by a valid justification (e.g., the consistency can be checked by theorem prover).

As a result of this strategy we get as many subgoals as there are safety requirements given. Here we focus on the goal (R2) corresponding to the second requirement, i.e., that the system shall compute a velocity in the ECI frame in signal vel_2 . Context nodes with hyperlinks outline additional evidence in the form of documents, containing, for example, a detailed description of the system and requirement, and also the result of the hazard analysis.

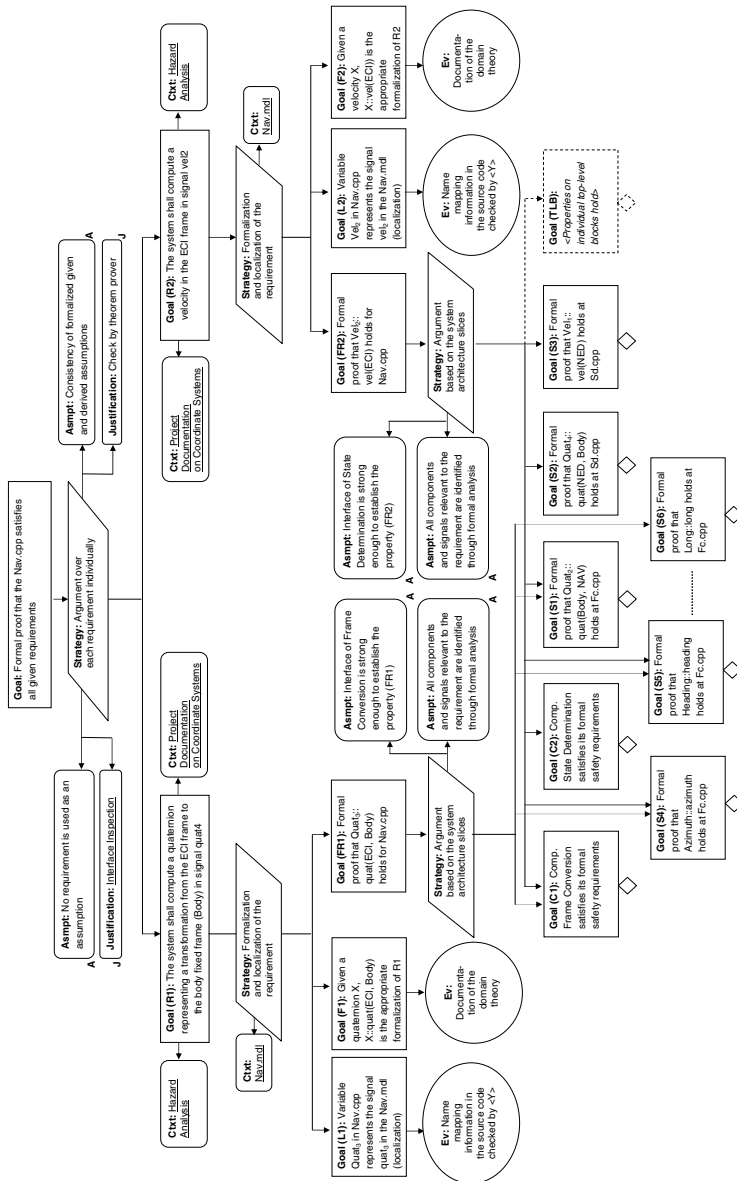


Fig. 3. Arguing from System-Level Requirements to Component-Level Requirements

The next step of the argument transitions from the informal level to a formalized safety requirement. This step helps in showing that the formal verification runs over the correct requirement, based on the right formula and variable, and thus provides a relevant proof of the program. We use an explicit strategy to describe this transition, which spawns three subgoals. As already discussed in Section 3.1, the first subgoal (F2)

demonstrates that the formal proof is based on an appropriate formalization of the requirement, and the safety case points to the documentation of the logical domain theory as evidence of this. The second subgoal (L2) “glues together” model and code levels, which allows us to build a safety case for the model based on the analysis of the code. In particular, as discussed in Section 3.2, we need to show the mapping between the signal names used in the model and the corresponding variable names used in the source code, which cannot be recovered by our analysis but must be given externally. Here, the safety case points to the mapping information given in the source code, and that it has been checked by a reviewer, as evidence. In addition, at this goal we also have to show the mapping between the model and code files, and in particular, in which code file the property formalized in (F2) has to be shown. In our example, this is straightforward, but for larger systems the localization needs more evidence.

With the results of (F2) and (L2) we can now construct the final subgoal (FR2) of our strategy, which shows that the fully formalized safety requirement $Vel_2:: vel$ (EC1) holds after execution of the code in `Nav.cpp`. This requirement eventually needs to be proven formally. However, at this level of abstraction, the safety case does not use an argument based on the full formal proofs. Instead, we use an argument based on the system architecture, or more precisely, on the recovered system architecture slices. It shows how the system level requirements are broken down into the component level requirements i.e., properties of the part of the system that is relevant to satisfy the requirement (FR2). The strategy is based on the assumption that the formal analysis has identified all relevant components and signals. We thus reduce (FR2) to a number of (delayed) subgoals for the components and signals in the architecture slice. For each component, we need to show that it satisfies the safety requirements specified in its interface (i.e., subgoals (C1) and (C2)). This induces a further assumption on the strategy, namely that the interface is strong enough to show the requirement (FR2). Delaying the subgoals allows us to reuse the component-level safety cases. This way, we achieve a hierarchical structure for the system safety case that mirrors the hierarchy embedded in the system architecture. If the system contains top-level blocks in addition to the components (which is not the case in our example), we need to reason about their properties as well. This is indicated by the dashed subgoal (TLB). For each variable representing a signal, we need to show that it satisfies the safety requirements derived by the analysis (i.e., subgoals (S1) to (S5)). This guarantees that the components’ assumptions are met. These subgoals are delayed here as well, to keep the safety case compact. Their expanded structure again follows the lines of our previous work [5], and uses the argumentation shown in Fig. 5 (Tier III) of the safety case there with small modifications; in particular, the notion of safety condition needs to be replaced by that of safety requirement. Note that we make no distinction at this level between subgoals that are established by the components (S2) and those that are reduced to assumptions about the system’s input signals and thus have trivial formal proofs, e.g., (S4).

3.4 Arguing from Component-Level Safety Requirements to Source Code

In the next step of our hierarchical development, we argue about the safety of the components wrt. their identified interfaces. The component-level safety cases also argue about a set of requirements, but there are two significant differences to the system-level

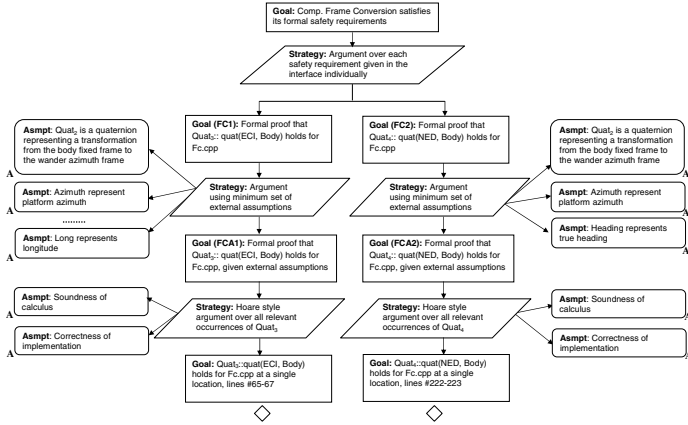


Fig. 4. Component-level safety case for Frame Conversion

safety cases. First, the component-level requirements are already formalized, due to the use of the formal analysis, so that we do not need to argue about the safety of the formalization and localization any more. Second, the argument will generally go down to the level of the generated code, with the proofs of the VCs as evidence; obviously, however, another layer of hierarchy is introduced if a component contains further components.

Fig 4 shows the safety case for the Frame Conversion component. For each component, the strategy is to argue over each individual safety requirement stated in its interface. Here, we have two requirements, (FC1) which is used to discharge the (essentially identical) system-level goal (FR1) via (C1), and (FC2), which is used to discharge the signal subgoal (S2). Even though they serve different purposes in the system-level safety case we treat them the same at the component level. We focus on (FC2) here.

The component interfaces also list the assumptions that the component itself makes about the environment. However, not all assumptions are used for all requirements, so we use an explicit strategy to argue only using the minimal set of external (i.e., on the system's input signals) assumptions. Note that the use of internal assumptions (e.g., on $Quat_4$), which have been identified as subgoals in the system-level safety case (i.e., (S2) in Fig. 3) will be made explicit further down in the component-level safety case.

The next strategy finally transitions from the safety argument to a program correctness proof, using a Hoare-style argument over all relevant occurrences of the variable. The structure of this Hoare-style argument is determined by the structure of the program. In this case, it leads to a single subgoal, proving that the safety requirement holds at the given source location. This is predicated on the assumptions that the applied Hoare-calculus is sound, and that the VCG is implemented correctly, which need to be justified elsewhere. Since the rest of the safety case is constructed as described in our previous work [5], we do not expand it here any further.

Showing the safety of the component is thus reduced to formally showing the validity of the VCs associated with each requirement in the interface. If (and only if) proofs for all corresponding VCs can be found, then the property holds for the entire program. The construction of safety cases from the proofs is described in our previous work [6].

3.5 Combining System-Level and Component-Level Safety Cases

Splitting the argument into system-level and component-level makes it easier to follow and allows common sub-arguments to be factored out, but in order to obtain a complete argument we need to combine the system- and component-level safety cases. However, simply attaching the entire component-level safety cases to the corresponding component goals would introduce redundancies. Clearly, not every safety requirement on the system level relies on the full set of requirements established by the components, for example, (FR2) only uses the requirement derived for $Quat_4$ (i.e., goal (FC2) in Fig 4).

We thus replace each component goal only by the “branches” of the component-level safety case that are required; this information is provided by the program analysis. For component goals that are shared between different requirements this will lead to an “unsharing”. For example, (C1) will be replaced by the branch rooted in (FC1) below (FR1) and by the one rooted in (FC2) below (FR2). However, common subgoals at the level of the Hoare-style argument, which are based on computations contributing to different requirements, can remain shared.

Additional changes occur elsewhere in the system-level safety case. The assumptions to the architecture-based strategy solving (FR1) and (FR2) can be removed because the detailed argumentation in the component-level safety case provides the necessary evidence. Further the subgoals associated with the system’s input signals (i.e., (S1) and (S3)–(S6)) can be removed because corresponding subgoals still appear as leafs in the component-level safety case, where they are discharged by the assumptions. The subgoals on the connecting signals (here only (S2)) will be replaced by the root goals of the corresponding branches in the component-level safety case (i.e., (FC2)) at the appropriate position in the Hoare-style argument for the client component (i.e., State Determination).

4 Safety Case Construction

The safety cases described here quickly become too large for manual development. Fortunately, the bulk of the argument is based on information provided by AUTOCERT’s formal program analysis, and the argument structure follows the program and analysis structure, so that a largely automated safety case construction is possible. However, some information cannot be produced by the program analysis, such as environment constraints, external assumptions, list of related documents, or model names. This information must be specified externally by a safety engineer. This also applies to the formalization of the top-level safety requirements that drive AUTOCERT’s analysis and their integration with the system-wide hazard analysis and safety case. Even though the constructed safety cases quickly become too large, an abstraction mechanisms can be used to highlight different aspects of the safety case. In particular, we can derive safety cases that are restricted to specific requirements, or to specific subsystems which is thus construct minimal but consistent safety case slices representing specific categories of information that help in manual safety case assessment.

In order to support the automated safety case construction, we integrate AUTOCERT’s formal program analysis with an existing commercial safety case tool, Adalard’s ASCE v3.5 tool [1]. We extended AUTOCERT to extract the manually specified

information from its own input and to structure this together with all information derived by the analysis into an XML format. The XML file records all the relevant information needed for the safety case construction. Subsequently, an XSLT program is used to transform this into a second XML format that logically represents the structure of the safety case as defined by safety case templates underlying the examples shown above. Here, the templates were designed so that the same argument structure can easily be adapted to other programs and systems. Finally, we use a custom Java program to present the safety case using GSN. The Java program helps to set the position of the nodes in the safety case which involved some mathematical calculations and to represent the argument to follow the standard Adelard ASCE file format. This architecture avoids a tight integration of the analysis (i.e., AUTOCERT) and presentation (i.e., ASCE) tools, and provides enough flexibility to change the latter with little effort.

The integration is largely completed; in particular, we have already fully automated the construction of the component-level safety cases that argue down to the code structure, and make up the overwhelming fraction of the combined safety case. However, the print quality of these large safety cases is insufficient for presentation, so we choose to recreate them in Microsoft Word here. The integration of system-level and component-level safety cases, as described in Section 3.5, requires further implementation work.

5 Related Work

The development and acceptance of a safety argument or safety case is a key element of safety regulation in most safety-critical sectors [19]. For example, Weaver [21] in his thesis presents arguments that reflect the contribution of software to a safety-critical system. Audsley et al. [4] present an argument based on correctness of the specification mapping, i.e., translation from the system specifications into a model and subsequently into code. Our work in contrast focuses on deriving a safety case that argues along the hierarchical structure of systems in model-based development and traces the safety requirements on the model via the inferred system structure to the code.

With the increased use of model-based development in safety-critical applications, the integration of safety cases into such approaches has become an important research topic. For example, Chen et al. [7] introduce an integration of model-based engineering with safety analysis and safety cases to help in assessing decisions in system design of automotive embedded systems. Hause and Thom [12] describe how SysML and UML can be used to model system requirements and how the safety requirements and other system elements identified in system design were used to construct the safety case. However, the focus in these papers is typically on extending various modelling frameworks to simply represent safety cases. Rushby [17] also uses automated theorem proving technology (based on the Yices SMT solver) to make a safety argument, but does not construct a detailed safety case. Moreover, his analysis starts with a manually constructed logic-based model of the system, whose connection to the underlying code remains unclear. In contrast, we focus on showing safety of the system on the code level and recover the slices of the system architecture to identify where in the system the safety requirements are established.

Most safety cases (see for example [11]) are constructed manually, as no advanced tools are available to support the automatic safety case construction. However, a manual

safety case construction [8] is far from satisfactory as it is a time-consuming and error-prone process. Most existing safety case construction tools only provide basic drawing support à la “boxes and arrows”. For example, GSN: ASCE v3.5 from Adelard [1], the University of York Freeware Visio Add-on and GSNCaseMaker [2] are graphical tools for creating a safety case by means of a drag and drop interface based on a commercial drawing tool. Obviously, tools supported by automated analyzers such as AUTOCERT are needed to produce the complex safety arguments for software. In our work, we integrate formal analysis with a commercial safety case tool (i.e., Adelard’s ASCE tool [1]) to automatically construct the safety case. Parallel to the work on safety cases described here, we have also used the same underlying information to create safety explanations in a textual form suitable for code reviews [10]. However, this work does not yet extend to the model-based reasoning level described here.

6 Conclusions and Future Work

We have described an approach where the hierarchical structure of systems in model-based development drives the construction of a hierarchical safety case for the generated code. Here, assurance is not implied by the trust in the generator but follows from a formal analysis of the code. The analysis is based on a set of formal safety requirements and provides formal proofs for use as evidence. We believe greater confidence in the assurance claims can be placed if the rationale behind the validity of the translation from the model to the program can be shown. We thus make explicit reference to the correct translation from the model level representation to the source level representation, including an argument over the formalization of the requirement. We show how the external assumptions on the systems input signals are used in establishing the safety of the program wrt. the given safety requirement. Like Rushby [17], we believe that “a safe design will have ensured that the assumptions are valid”. Moreover, Littlewood et al. [15] explain why there is a very low probability of a claim that has been shown by a formal proof, actually being false, when the assumptions and evidence are valid. We thus believe that formal methods can provide the highest level of assurance when they are combined with explicit safety arguments such as the ones we derived here.

The work described here is still in progress, and we are currently completing the automatic construction of the safety cases. So far, we only consider nominal component behavior, but our approach could also be applied to the off-nominal case, provided that appropriate safety requirements for the off-nominal modes can be identified. We have applied our technique only to flight code generated by Real-Time Workshop from hierarchical Simulink models but we are confident that the same approach can be applied to other modelling systems and generators as well. Future work will focus on complementary safety cases that argue the safety of the certification framework itself, in particular the safety of the underlying safety logic (the language semantics and the safety policy). We believe that the result of our research will clearly communicate the safety claims, key safety requirements, and evidence required to trust the generated code.

Acknowledgements. This material is based upon work supported by NASA under awards NCC2-1426 and NNA07BB97C. The first author is funded by the Malaysian Government and USIM.

References

1. ASCE home page (2007), <http://www.adelard.com/web/hnav/ASCE>
2. CET GSNCase Maker (2007), <http://www.esafetycase.com>
3. Real-Time Workshop Embedded Coder (2007), <http://www.mathworks.com/products/rtwembedded>
4. Audsley, N.C., Bate, I.J., Crook-Dawkins, S.K.: Automatic Code Generation for Airborne Systems. In: IEEE Aerospace Conf., pp. 8–15. IEEE, Los Alamitos (2003)
5. Basir, N., Denney, E., Fischer, B.: Constructing a Safety Case for Automatically Generated Code from Formal Program Verification Information. In: Harrison, M.D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 249–262. Springer, Heidelberg (2008)
6. Basir, N., Denney, E., Fischer, B.: Deriving Safety Cases from Automatically Constructed Proofs. In: 4th IET Intl. Conf. on System Safety (2009)
7. Chen, D.-J., Johansson, R., Lönn, H., Papadopoulos, Y., Sandberg, A., Törner, F., Törngren, M.: Modelling Support for Design of Safety-Critical Automotive Embedded Systems. In: Harrison, M.D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 72–85. Springer, Heidelberg (2008)
8. Cockram, T., Lockwood, B.: Electronic Safety Cases: Challenges and Opportunities. In: Safety Critical Systems Symposium 2003. Springer, Heidelberg (2003)
9. Denney, E., Fischer, B.: A Generic Annotation Inference Algorithm for the Safety Certification of Automatically Generated Code. In: GPCE 2006, pp. 121–130. ACM, New York (2006)
10. Denney, E.: A Verification-Driven Approach to Traceability and Documentation for Auto-Generated Mathematical Software. In: ASE 2009, pp. 560–564. IEEE, Los Alamitos (2009)
11. Eurocontrol: Preliminary Safety Case for Enhanced Air Traffic Services in Non-Radar Areas using ADS-B Surveillance (2008)
12. Hause, M.C., Thom, F.: Integrated Safety Strategy to Model Driven Development with SysML. In: 2nd IET Intl. Conf. on System Safety, pp. 124–129 (2007)
13. Kelly, T.P.: Arguing Safety a Systematic Approach to Managing Safety Cases. PhD Thesis, University of York (1998)
14. Leveson, N.G., Cha, S.S., Shimeall, T.J.: Safety Verification of ADA Programs using Software Fault Trees. IEEE Software 8(4), 48–59 (1991)
15. Littlewood, B., Wright, D.: The Use of Multilegged Arguments to Increase Confidence in Safety Claims for Software-Based Systems: A Study Based on a BBN Analysis of an Idealized Example. IEEE Trans. Software Eng. 33(5), 347–365 (2007)
16. O'Halloran, C.: Model Based Code Verification. In: Dong, J.S., Woodcock, J. (eds.) ICFEM 2003. LNCS, vol. 2885, pp. 16–25. Springer, Heidelberg (2003)
17. Rushby, J.: A Safety-Case Approach For Certifying Adaptive Systems. In: AIAA Infotech@Aerospace Conference (2009).
18. Schloegel, K., Oglesby, D., Engstrom, E., Bhatt, D.: Composible Code Generation for Model-Based Development. In: Krall, A. (ed.) SCOPES 2003. LNCS, vol. 2826, pp. 211–225. Springer, Heidelberg (2003)
19. UK Ministry of Defence: 00-56 Safety Management Requirements for Defence Systems, Issue 4 (2007)
20. Vallado, D.A.: Fundamentals of Astrodynamics and Applications, 2nd edn. Microcosm Press and Kluwer Academic Publishers, Dordrecht (2001)
21. Weaver, R.A.: The Safety of Software-Constructing and Assuring Arguments. PhD Thesis, University of York (2003)
22. Weiss, K.A.: Component-Based Systems Engineering for Autonomous Spacecraft. MSc Thesis, Massachusetts Institute of Technology (2003)

Assurance of Automotive Safety – A Safety Case Approach

Robert Palin¹ and Ibrahim Habli²

¹ Jaguar Land Rover, Coventry, UK
rpalin@jaguarlandrover.com

² University of York, York, UK
Ibrahim.Habli@cs.york.ac.uk

Abstract. A safety case should provide a clear, comprehensible and defensible argument, supported by evidence, that a system is acceptably safe to operate in a particular environment. This approach is not new. For example, in the nuclear industry, safety cases are approaching their 50th birthday. In stark contrast, the automotive industry has never been required to produce a safety case. Instead, it has relied on compliance with extensive regional and national regulation. With the imminent introduction of the automotive safety standard ISO 26262, the production of a safety case is now explicitly required by the standard for electrical and electronic systems. This presents both opportunities and challenges to safety practitioners and researchers within that industry. This paper sets out to look at the issues of what a safety case might look like for a complete vehicle and how the ISO 26262 fits into the existing framework of automotive safety. Using the ideas of modular safety case construction, this approach is developed into a number of reusable safety arguments to form an automotive safety case pattern catalogue. Evaluation of the approach is carried out through an industrial case study.

Keywords: Safety Cases, Automotive Safety, Functional Safety, ISO 26262.

1 Introduction

Road safety is an immensely complicated and diverse subject. Arguably, the road transport system is the most complex system the majority of the world population use on a daily basis. The latest 2008 figures for Road Casualties in Great Britain put the number of people killed at 2,538 and the total number of road accidents reported to the police at 170,591 [1]. The total number of deaths for car users was 1,257 with 11,535 car users seriously injured. Fortunately, the overall trend for Great Britain is one of continual reduction and has been since 1990. Overall the trend within Europe is downward. In 2006 the total number of road fatalities for the 27 European countries was approximately 43,000 [2].

As described by the Commission for Global Road Safety [3] “*Most of the time road traffic deaths and injuries remain invisible to society at large. Tragic to those involved but not newsworthy. This is a hidden epidemic*”. Clearly this sets difficult challenges for those involved in road safety of which automotive safety plays a key

contributing part. One of the most influential breakthroughs for effective road safety management was made by William Haddon. He described road transport as “*an ill-designed ‘man-machine’ system needing comprehensive systematic treatment*” [4]. Using a simple table (Table 1), he defined three phases of the time sequence of a crash event {pre-crash; crash; post-crash} and the three main factors {human; vehicles; environment} that interact during each phase. Although over 40 years old, this ‘systems’ approach still underpins the various strategies used for road safety today. For example, on reviewing brochures for new cars, it can be seen that safety now plays a significant role in the marketing of a new vehicle and that the safety features have been grouped according to the phases identified in the prevention (Active Safety) and mitigation (Passive Safety) of a crash scenario as identified by Haddon.

Table 1. The Haddon Matrix [4]

		Factors		
Phase	Goal	Vehicles & Equipment	Environment	Human
Pre-crash	Crash prevention	Roadworthiness; Active Safety Systems	Road design & layout; Speed limits	Attitudes; Police enforcement
Example		MOT test; Stability control option	Highway Guidelines; Speed cameras & speed bumps	Think! Road Safety Campaign
Crash	Injury prevention during crash	Crashworthiness (crash protection design); Passive Safety Systems	Crash protective roadside objects	Use of restraints; Impairment
Example		EuroNCAP Score; Airbags	Crash barriers	Think! Road Safety Campaign
Post crash	Life sustaining	Ease of access; Fire risk	Rescue facilities; Congestion	First aid skill; Access to medics
Example		Vehicle design (ingress / egress)	Close proximity to hospitals; Air ambulance	Use of paramedics; Use of fire service

Whereas active and passive safety systems can be physically seen, touched and experienced, the same is not always true for the output of functional safety. The draft ISO 26262 defines functional safety as the “*absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems*” [11]. Given that the roots of the automotive industry are based on mechanical engineering principles, the traditional view is that accidents are primarily caused by component failures and therefore increasing component reliability will reduce accident frequency [5]. The main technique used to capture this component reliability is typically Failure Modes and Effects Analysis (FMEA). While this approach has undeniably worked well, the functional safety view of taking a holistic approach to vehicle safety, considering complex interactions which may not require a component failure, is slowly gaining acceptance. This is important as the requirements on the electrical and electronic architecture expand and the amount of coupling between systems increases.

2 Current Thinking and the Development of ISO 26262

In response to the increasing complexity in vehicle functionality, the automotive industry has until recently mainly adopted IEC 61508 [12] as an example of best practice. In 2004 however, two national initiatives, one lead by the VDA/FAKRA group in Germany and the other by the BNA group in France, decided to merge and submit a proposal to ISO for an automotive specific standard. This was accepted and a new

ISO working group ISO/SC22/TC3/WG16 (26262) was convened in 2005. In brief, the standard itself is essentially an adaptation of IEC 61508 with the key deliverable being the generation of a safety case that shows why the developed system is believed to be acceptably safe for use (i.e. absence of unreasonable risk [11]).

The impending introduction of ISO 26262 will offer the Original Equipment Manufacturers (OEM) and suppliers an agreed industry standard for managing risk for electronic vehicle systems. However, the concept of a safety case is not readily well known to those that work in the industry. To this end, the overarching aim of the paper is to investigate and demonstrate how to produce automotive safety cases in order to justify that an automotive system is acceptable safe. Specifically, this paper presents a safety assurance approach which addresses the following objectives:

- Definition of top-level safety claims that can be made regarding the safety of automotive systems;
- Formulation of argument strategies and evidence that can substantiate the safety claims (using the new ISO 26262 as context, where appropriate);
- Definition of the arguments and evidence in the form of re-usable patterns.

The rest of this paper is organised as follows. Section 3 discusses key dependencies for automotive safety cases. Section 4 presents an approach to capturing automotive safety case in the form of reusable argument patterns. Section 5 evaluates these patterns by means of an industrial case study. The paper concludes in Sections 6 and 7 with observations concerning automotive safety cases, ISO 26262 and argument patterns.

3 Dependencies of Automotive Safety Cases

The validity of a safety case rests on different system and context dependencies. Figure 1 shows a dependency diagram for an automotive safety case. It is not claimed that the dependencies shown, numbered from 1 to 10, represent a complete set. Rather, they represent the major considerations that should be made.

Firstly, there are different types of automotive safety requirements. On the one hand, there are *predefined safety requirements* (1) which include the statutory regulations that must be met as a bare minimum in order to sell cars in the first instance (e.g. the UN-ECE and FMVSS regulations [6]). On the other hand, there are *developed or derived system safety requirements* (2) which specify the implementation of risk mitigation measures, typically generated from the specification and analysis of the system. In the context of the ISO 26262 Standard, these requirements are the ‘item’ safety goals. These requirements may also incorporate predefined safety requirements (3), for example a leg-form to bumper performance requirement as part of a pedestrian protection system.

Secondly, various items of evidence may be produced to support the satisfaction of the safety requirements. In view of the fact that the predefined safety requirements are explicit in what is required, some standards are also explicit in how these requirements can be satisfied (4). This leads to the production of product evidence taken from the testing or analysis of the design’s manifestation (5). In addition to the evidence that is directly related to the product, the adequacy of process (6) should be

considered (i.e. evidence concerning the quality of the process). In the context of ISO 26262, compliance with the standard could support process claims such as:

- The risk assessment scheme is valid;
- The process has been performed with the appropriate degree of rigour as given by the Automotive Safety Integrity Level (ASIL);
- The direct evidence relates to the actual product sold because proper process control is enforced (e.g. configuration control).

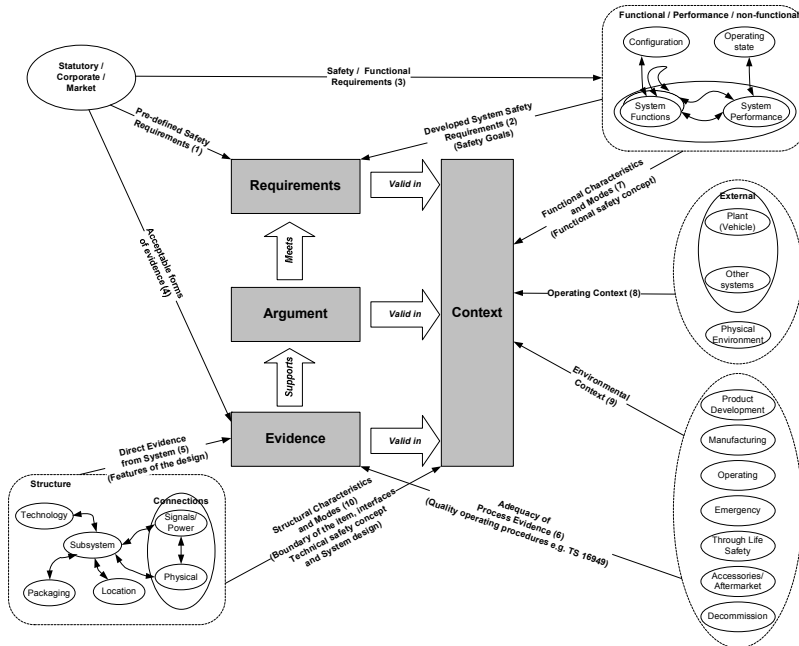


Fig. 1. Safety Case Dependencies (based on [7])

Thirdly, the context of the safety case needs to be accurately defined. This is crucial since “a safety case cannot argue the safety of a system in any context” [8]. For example, with reference to Figure 1, if an argument is being made about the functional characteristics of the system, such as its response time, then the operating, environmental and structural characteristic of the system would all typically become declared context. Figure 1 includes four context categories:

- The functional characteristics and modes (7) which contextualise the safety argument based on the system’s functions, performance and configuration
- The operating context (8) which contextualises the safety argument based on how the system is operated with respect to the vehicle, other vehicle systems and the physical environment (e.g. temperature, pressure, humidity, dust, vibration, shock, corrosion and static electricity);

- The environmental context (9) which contextualises the safety argument based on product development, manufacturing, operation, emergency, through-life safety, accessory/aftermarket modifications and decommissioning;
- The structural characteristics and modes (10) which contextualise the safety argument based on how the system has been physically implemented in terms of the technology used and its packaging and location.

4 An Approach to Creating Automotive Safety Cases

In this section we define a pattern catalogue of automotive safety arguments (Figure 2). The argument patterns are identified by their unique name. In total, 12 argument patterns are defined, some of which are designed to be connected together to produce integrated product and process arguments [13]. In order help comprehend how the various arguments are interrelated, the reader is advised to refer to Figure 2 when the individual patterns are discussed in the next sections. The patterns address aspects of safety related to safety requirements, hazard/risk analysis and through-life safety.

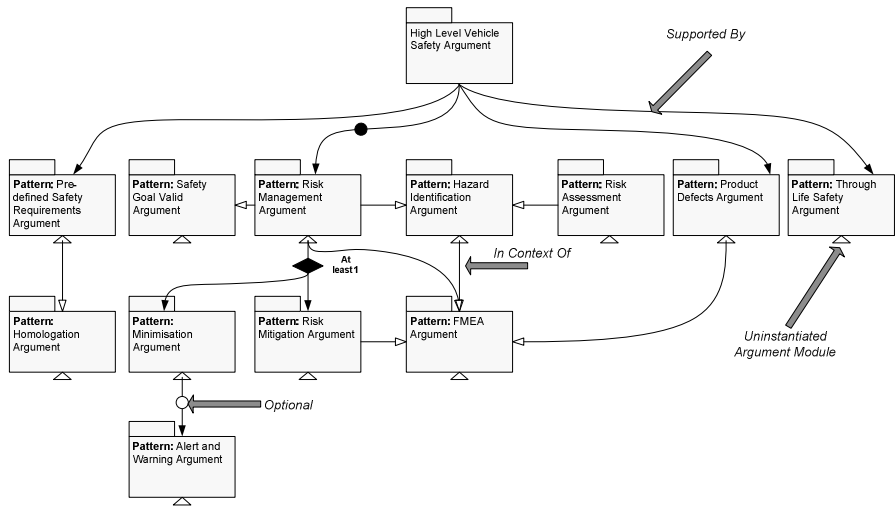


Fig. 2. Architecture for the Argument Pattern Catalogue

The argument patterns are created using the *Patterns* and *Modular* extensions of the Goal Structuring Notation (GSN) [8], [9], [14]. GSN is a graphical notation for the representation of safety arguments in terms of basic elements such as goals, solutions, and strategies. Arguments are created in GSN by linking these elements using two main relationships, ‘supported by’ and ‘in context of’ to form a goal structure. A goal structure represents a recursive decomposition of goals (i.e. claims), typically using GSN strategies, until the sub-goals can be supported by direct solutions (i.e. items of evidence). GSN has two extensions: *Patterns* and *Modular* extensions. The concept of a safety case pattern in GSN was developed as “a means of documenting and reusing

successful safety argument structures” [8]. Argument patterns support the generalisation and specialisation of GSN elements. They also support multiple, optional and alternative relationships between the GSN elements. Concerning the modular extension of GSN, it mainly supports the development of *modular and compositional safety cases*. These safety cases can be viewed as a set of well-defined and scoped modules, the composition of which defines the system safety case. For a detailed description of GSN and its extensions, the reader can refer to [8], [9], [14].

4.1 High Level Vehicle Argument Pattern

The ‘*High Level Vehicle Safety Argument*’ module in Figure 2 contains the high-level argument concerning the safety of a vehicle. This argument is shown in Figure 3. The top-level claim, ‘*The vehicle is acceptable safety*’, is made in the context of a definition of the vehicle (e.g. private passenger vehicle or commercial vehicle), a definition of the vehicle attributes (e.g. 0-60 time) and a physical representation of the vehicle.

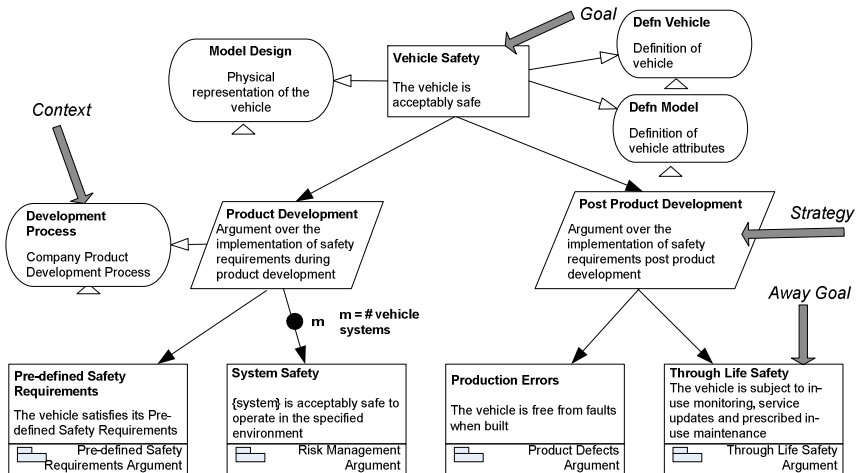


Fig. 3. High Level Vehicle Safety Argument Pattern

The two high level strategies developed to support the top-level claim are based on the stage of the product within the product lifecycle, namely *during* and *after* product development. Four different *Away Goals* support these strategies. An *Away Goal* is a goal reference which is used to support, or provide contextual backing for, an argument presented in one argument module. However, the argument supporting that goal is presented in another argument module (hence creating interdependencies between argument modules). The *Away Goals* used in Figure 3 are:

- ‘*Pre-defined Safety Requirements*’: The vehicle satisfies predefined safety requirements, i.e. it has been homologated against regulations which capture essential vehicle attributes (e.g. braking system and steering system);

- ‘*System Safety*’: A vehicle system is acceptably safe to operate in the specified environment;
- ‘*Production Errors*’: The vehicle was free from known safety related defects when it was built;
- ‘*Through Life Safety*’: The vehicle is subject to in-use monitoring, service updates and prescribed in-use maintenance. That is, the OEM has a dealer network capable of maintaining the vehicles correctly and has processes in place for evaluating and responding to field accidents or incidents.

4.2 Predefined Safety Requirements Argument Pattern

The Away Goal ‘*Pre-defined Safety Requirements*’ in Figure 3 refers to the argument pattern ‘*Pre-defined Safety Requirements Argument*’ in Figure 2. This argument is described in this section and depicted in Figure 4.

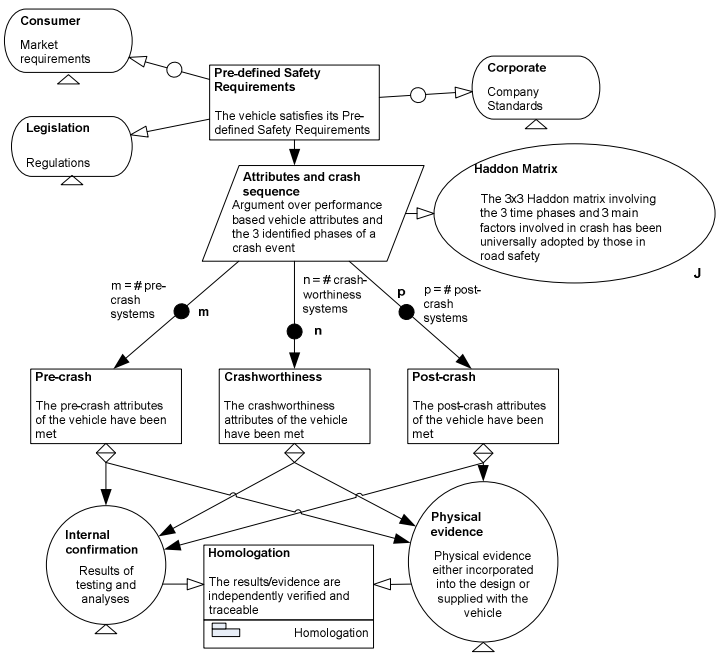


Fig. 4. Predefined Safety Requirements Argument Pattern

The pre-defined safety requirements are mainly based on applicable regulations. Regulations, whether international or regional, are an agreed way of assessing vehicle systems. It would seem appropriate to group the various regulations and vehicle assessment tests according to the initiatives in use within the bigger picture of road safety as defined by the Haddon matrix. In the argument in Figure 4, three main claims are made concerning the pre-crash, crashworthiness and post-crash attributes

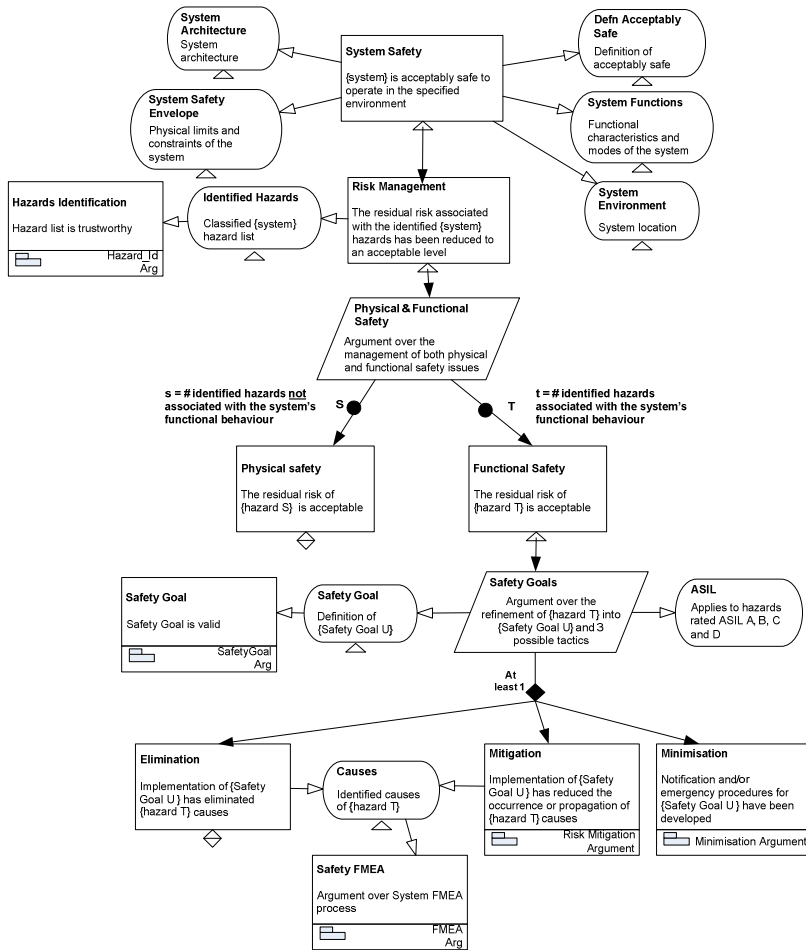


Fig. 5. Risk Management Argument Pattern

of the vehicle, which need to be developed and instantiated. These claims are eventually supported by evidence generated from testing, analysis and physical inspection of the vehicle. It is important to note that the evidence is used in the context of an Away Goal 'Homologation'. This Away Goal refers to an argument which justifies that the evidence is independently verified and traceable. This is normally called a process-based argument or backing argument [13], which aims at justifying the process by which the evidence used in the primary product-based argument is generated (e.g. justifying the thoroughness of the review, quality of the review methods and competency and independence of the reviewers). Process-based arguments play a key role in justifying the *trustworthiness* of the evidence (i.e. addressing the simple question: *why should anyone trust the evidence?*).

4.3 Risk Management Argument Pattern

The second Away Goal, ‘System Safety’, in Figure 3 refers to the ‘Risk Management Argument’ pattern in Figure 2. This argument is described in this section (shown in Figure 5). This argument is one of the most important arguments described in the catalogue as it explicitly addresses the hazards and risks posed by a vehicle system. The argument supports the claim that a vehicle system is acceptably safe by justifying that the residual risks associated with the identified hazards have been reduced to an acceptable level. The argument is then split into two parts, addressing both the physical and functional safety attributes of the system. In particular, the claims concerning the hazards related to the functional safety attributes are supported by the definition of safety goals which address these hazards. Finally, this argument addresses the claims concerning the safety goals by considering how the risks of the hazards have been managed by means of elimination, mitigation or minimisation [10].

Finally, within this argument pattern, there are three Away Goals, which refer to process-based arguments. The ‘Hazard Identification’ Away Goal refers to an argument which justifies the process by which the hazards have been identified. The ‘Safety Goal’ Away Goal refers to an argument which justifies the specification of the safety goals. The ‘System FMEA’ Away Goal refers to an argument which justifies the FMEA process. These Away Goals are developed in separate argument patterns.

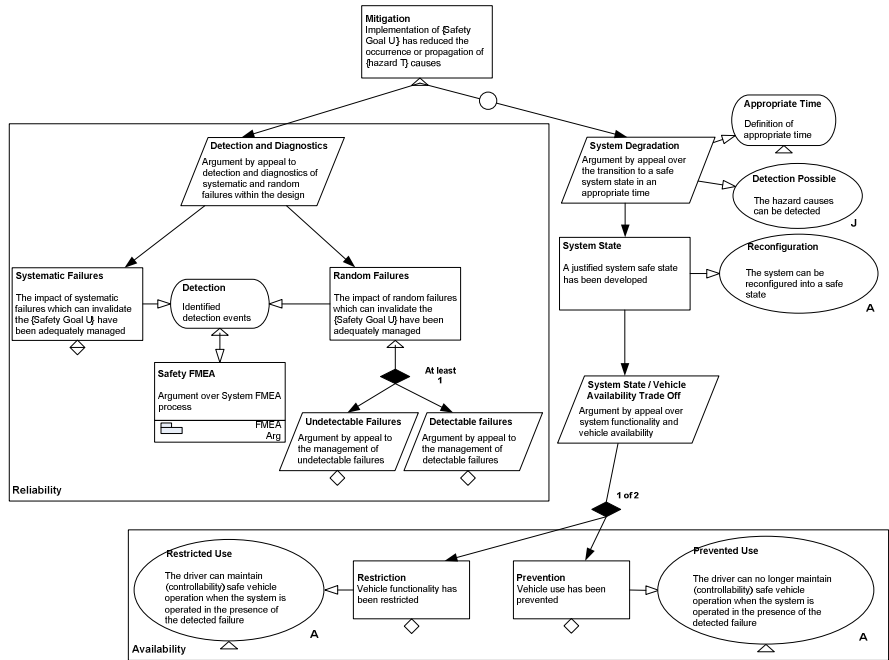


Fig. 6. Risk Mitigation Argument Pattern

4.4 Risk Mitigation Argument Pattern

In the previous argument pattern, risk mitigation was considered as a means for managing the risks of the hazards addressed by the safety goals. In this section we describe an argument pattern which appeals to mitigation by means of failure detection and diagnostics (reliability) and system degradation (availability).

This risk mitigation argument pattern is depicted in Figure 6. The structure of the pattern is based upon the ability to detect hazardous conditions and reconfigure the system to a justified safe state, referred to as ‘system degradation’ in the ISO 26262 terminology [11]. It is important to note that the system degradation leg is optional. This is because alternative strategies such notifying the driver or writing emergency procedures might be more applicable. With regard to restrictive or preventative use, there is an assumption in the argument regarding the driver being able to maintain the safety of the vehicle when the system or the vehicle is in the degraded state.

4.5 Alert and Warning Argument Pattern

It is sometimes the case that certain hazards cannot be contained and therefore require either driver intervention or the specification of emergency procedures. This case is considered in the ‘*Alert and Warning Argument*’ pattern. This argument is shown in Figure 7 (refer to Figure 2 to see how this argument pattern fits with other patterns). This argument supports a claim that the driver has been warned of a hazardous situation or system operating state. The structure of the pattern is split over the driver’s senses of sight, hearing and touch (e.g. concentrating on claims related to visual alerts such as the use of tell-tales and text within a modern vehicle instrument cluster). Within Europe and the US, the regulatory requirements for instrument clusters are contained within the UN-ECE 121 and FMVSS 101 standards respectively. The claims used in the argument in Figure 7 have been based on these regulatory requirements.

5 Case Study – Stop/Start System

We illustrate the uses of the argument pattern catalogue described in the previous section in a case study based on the Stop/start system. Stop/start systems have been developed by the automotive industry as one of the initiatives for supporting low-emission CO₂ vehicles. The system simply stops the internal combustion engine whenever the vehicle is stationary and restarts it immediately when the driver wishes to go. Envisaged traffic situations include queuing in congested traffic or waiting at traffic lights. The development of a safety case is an effective approach to explicitly justifying that all due diligence has been performed with respect to the Stop/start system operating in a particular environment. In this case study, we used the argument patterns described in the previous section for the development of the Stop/start safety case. In particular, we used the following the argument patterns:

- Risk Management pattern
- Hazard Identification pattern
- Risk Assessment pattern
- Safety Goal Pattern
- Risk Mitigation pattern
- FMEA pattern
- Production Failures pattern
- Through Life Safety pattern

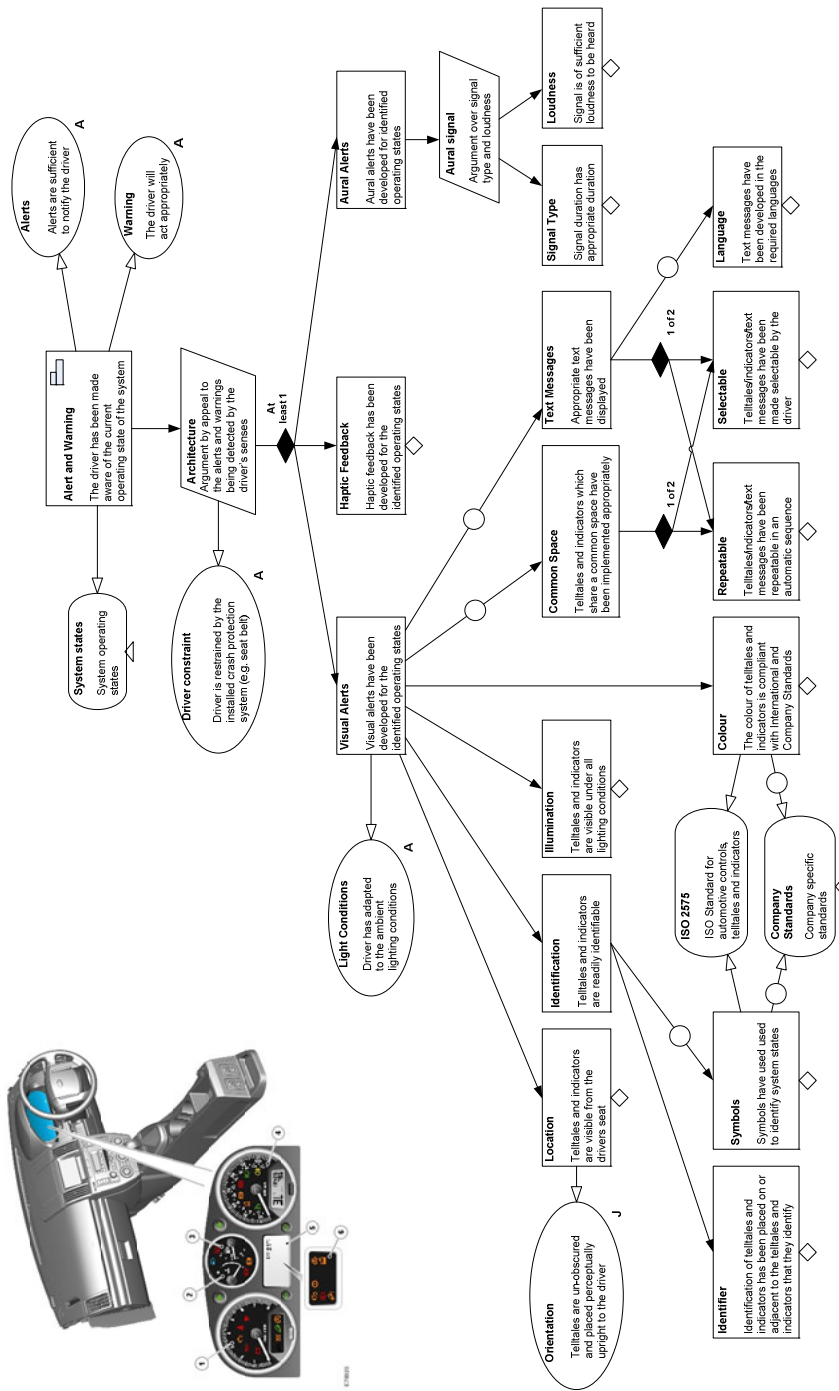


Fig. 7. Alert and Warning Argument Pattern

Given the extent of the argument patterns developed and the page constraint, this paper illustrates the instantiation of the ‘*Risk Management Argument Pattern*’ and part of the ‘*Risk Mitigation Argument Pattern*’ only. Firstly, in order to instantiate the ‘*Risk Management Argument Pattern*’, the required context at the top of the argument was developed (Figure 8). This included the development of various models such as:

- A component location and context diagram to help define the environment;
- An ‘item’ boundary diagram to define the system safety envelope;
- Hardware and software boundary diagrams to define the system architecture;
- A function cascade and sequence diagrams and state machine analysis to adequately define the system functions.

The main objective for creating these different views was to flush out assumptions regarding the operation of the system within the design and safety teams.

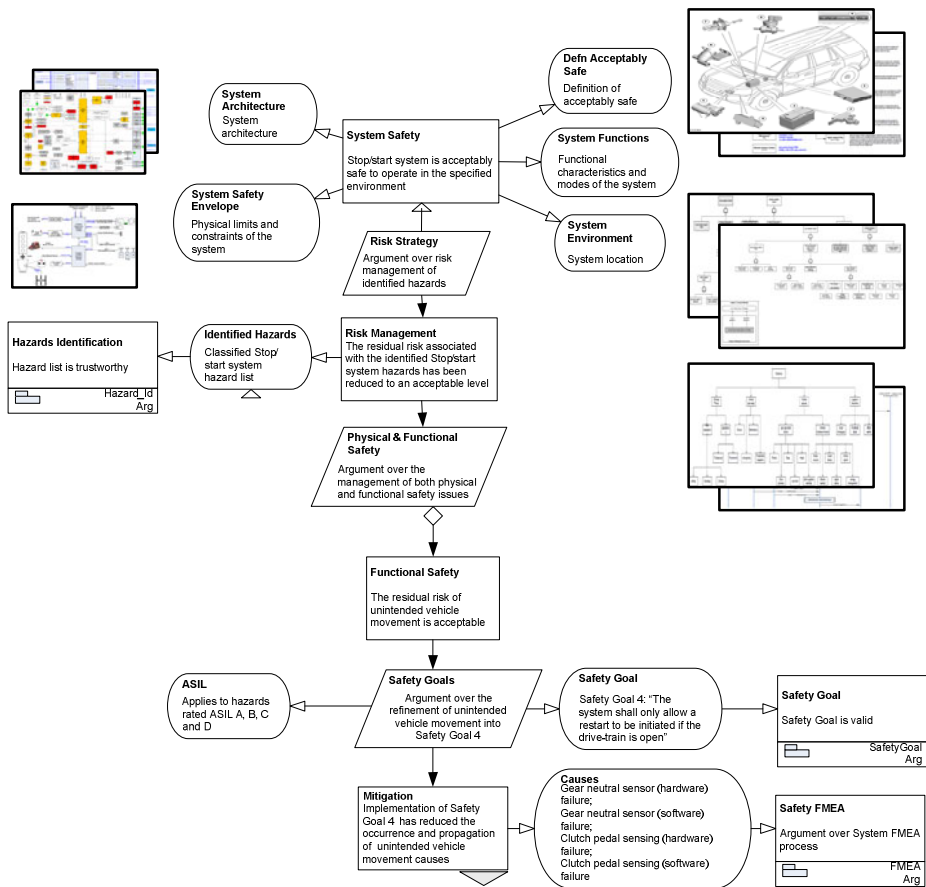


Fig. 8. Instantiation of the Risk Management Argument Pattern

In Figure 8, we only elaborate on the consideration of one potential hazard, ‘*Unintended Vehicle Movement*’. To address the risk associated with this hazard, a claim is made that a safety goal has been specified to manage this risk. The safety goal states that the ‘*system shall only allow a restart to be initiated if the drive-train is open*’. The drive train refers to all the components along a path of power from the engine to the drive wheels (e.g. clutch, transmission, drive shaft, differential and transaxle or rear axle). Finally, this part of the argument shows that the safety goal has been addressed by the reduction of the occurrence and propagation of the hazard causes. To address this claim, we instantiated the ‘*Risk Mitigation Argument Pattern*’, previously described in Section 4.4, in order to justify the adequacy of risk mitigation by means of failure detection and diagnostics and system degradation.

Figure 9 shows the instantiation of one part of the ‘*Risk Mitigation Argument Pattern*’. It addresses system degradation through the restriction of the functionality of the Stop/start system in the presence of a detected fault (after ensuring that the system is in a safe state). The claim concerning restriction of functionality is supported by evidence generated from vehicle testing.

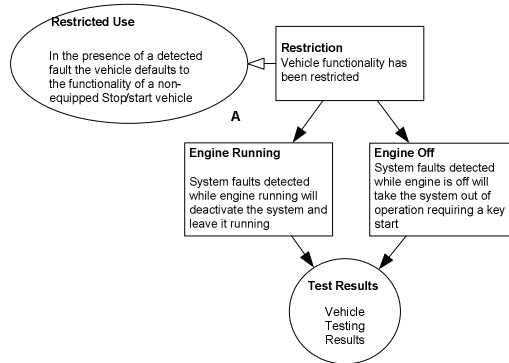


Fig. 9. Risk Reduction through Functionality Restriction

6 Observations

The argument pattern catalogue and case study have described how an assurance approach, based on explicit safety cases, can pave the way for greater understanding and transparency within the automotive industry. The following observations can be made concerning this approach:

- It is effective to create hazard and risk directed product-based arguments for an automotive system. That is, automotive safety practitioners can show compliance by embracing a ‘*product assurance mentality*’ rather than compliance through ‘*box ticking*’.
- The argument patterns capture the need for better integration between design and safety. This benefits both the design and safety teams, e.g. as shown in the Stop/start case study (generation of state, sequence and logic diagrams).

- In comparison with splitting the safety case argument at a high level into specific product and process-based components, the concept of creating integrated product-based and process-based arguments through the use of ‘*Away Goals*’ appears to generate a clearer and more traceable safety case.
- The development of ‘*Green Technologies*’, such as the Stop/start, presents many challenges for those in the automotive industry where implicit assumptions about the driver and vehicle behaviour may no longer hold true. The rigorous development of a safety case should help reveal these assumptions and ensure that a new technology not only delivers environmental and economic benefits but also does so in a safe manner.
- Modular GSN can support the development of modular arguments, which can be directly mapped onto the various parts of the ISO 26262 standard. This gives the opportunity for competitive advantage through the reuse of safety arguments.

Nevertheless, a number of issues and limitations have also been identified. Although GSN can help define a clear and structured safety case, any safety practitioner responsible for this task needs to first and foremost understand the system and the domain, otherwise the safety case could easily be misrepresentative.

7 Conclusions

The safety case approach presented in this paper is primarily intended to add value to safety engineers with prior knowledge of automotive system design, operation and maintenance. However, it should also be of interest to safety engineers within other domains and academics within the system safety community. Safety engineers should not regard these argument patterns as the only or preferred means for generating automotive safety cases. Rather, these patterns represent worked examples based on industry-driven research, illustrating how automotive safety arguments can be constructed and supported by direct items of evidence. Finally, it is hoped that this work will encourage safety practitioners and researchers to share and publish successful uses of safety cases within the automotive industry.

Acknowledgements

This work is based on developments and knowledge within Jaguar Land Rover, the authors would therefore like to thank Jaguar Land Rover Management and Safety teams, especially Mr Phil Whiffin and Mr Roger Rivett.

References

1. Department for Transport: Road Casualties in Great Britain: Main Results. Department for Transport (2008), <http://www.dft.gov.uk/pgr/statistics/datatablespublications/accidents/casualtiesmr/rcgbmainresults2008>
2. European Road Statistics, <http://www.irfnet.eu/images/stat/2008/>

3. Commission for Global Road Safety: Make Road Safe,
<http://www.fiafoundation.org/commissionforglobalroadsafety/>
4. Haddon, W.: The Changing Approach to the Epidemiology, Prevention and Amelioration of Trauma: The Transition to Approaches Etiologically Rather than Descriptively Based. *Am. J. Public Health* 58, 143–1438 (1968)
5. Leveson, N.G.: System Safety in Computer Controlled Automotive systems. In: SAE, vol. 1048 (2000)
6. Federal Motor Vehicle Safety Standards and Regulations,
<http://www.nhtsa.dot.gov/cars/rules/import/FMVSS/#SN101>
7. Dowding, M.: Maintenance of the Certification Basis for a Distributed Control System – Developing a Safety Case Architecture. MSc Thesis, University of York, UK (2002)
8. Kelly, T.P.: Arguing Safety – A Systematic Approach to Safety Case Management. DPhil Thesis, Department of Computer Science, University of York, UK (1998)
9. Bate, I.J., Kelly, T.P.: Architecture Consideration in the Certification of Modular Systems. *Reliability Engineering and System Safety* 81(3), 303–324 (2003)
10. Wu, W.: Architectural Reasoning for Safety Critical Software Applications. DPhil Thesis, Department of Computer Science, University of York, UK (2007)
11. International Organization for Standardization (ISO): ISO26262 Road vehicles – Functional safety. Draft, Baseline 15 (2009)
12. International Electrotechnical Commission (IEC): BS IEC 61508 – Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related System. BSC/IEC (2002)
13. Habli, I., Kelly, T.P.: Process and Product Certification Arguments: Getting the Balance Right. *Innovative Techniques for Certification of Embedded Systems*, CA, USA (2006)
14. Kelly, T.P., McDermid, J.A.: Safety Case Construction and Reuse using Patterns. In: 16th International Conference on Computer Safety, Reliability and Security (1997)

How to “Survive” a Safety Case According to ISO 26262

Torsten Dittel¹ and Hans-Jörg Aryus²

¹ Ford Werke GmbH, Henry-Ford-Straße 1, 50735 Köln
tdittel@ford.com

² SystemA Engineering Gesellschaft für Systemanalyse mbH,
Bürglen 11, 88090 Immenstaad
hans-joerg.aryus@SystemA-GmbH.de

Abstract. Modern road vehicles are equipped with driver assistance systems which support the safety of the vehicle in case of driver inattention. *Ford* is currently designing Lane Assistance functions which warn the driver haptically when leaving the lane or even generate a steering torque which brings the vehicle back into lane. The overlay of a steering torque includes the risk that an incorrectly performed function could lead to a safety issue. The ISO 26262 standard describes the process which has to be applied from a safety point of view. As with most standards the execution of the rules as laid down leaves room for interpretation and implementation which need to be solved in order to have a closed process. Another trap which has been identified as crucial is the level of detail. A too high level of detail contains the risk that the overview might get lost whereas a too low level of details contains the risk that safety issues might be overlooked. *Ford*, in conjunction with *SystemA Engineering*, has applied practical tools and methods which support the safety process according to ISO 26262. The safety steps and methods PHA, Safety Concept, FTA, FMDEA, Safety Requirements, as well as Validation and Verification are applied as an integrated approach which forms a part of the overall *Ford* development process. Practical experience has driven the methods and the interfaces between the various methods as well as the level of detail necessary for the safety case. This paper and the presentation will show a practical example how a great portion of the ISO 26262 safety case can be developed, documented, evaluated and managed without losing the overall picture. The example will also cover interfaces between different disciplines as well as between OEM and supplier.

Keywords: ISO 26262, Safety Case, Lane Assistance Functions, PHA, Safety Concept, FTA, FMDEA, ASIL, Safety Requirements, Validation and Verification.

1 System Description

Lane Assist (LA) is the collective name for a set of functions which help the driver of a vehicle to keep the lane the vehicle is driven in.

Lane Assist provides the following different functional levels which can be combined:

- Lane Departure Warning (LDW)
- Lane Keeping Aid (LKA)

1.1 Lane Departure Warning (LDW)

The LDW system informs the driver by a warning when the vehicle is about to leave the current lane. This warning is generated by optical and haptical means. A variant of haptic warning is the steering wheel vibration. It can either be generated by the EPAS¹ motor via a harmonic torque overlay or by a vibration motor in the steering wheel itself.

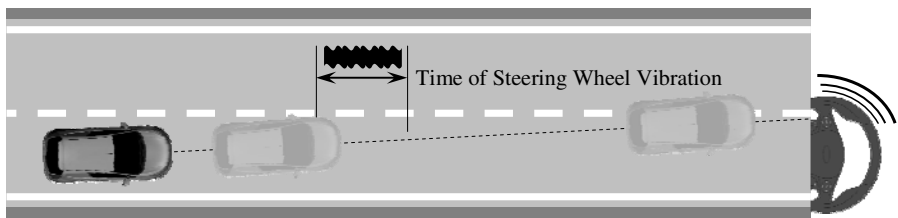


Fig. 1. Lane Departure Warning (LDW)

1.2 Lane Keeping Aid (LKA)

The LKA system intervenes by a steering torque action when the vehicle is about to leave the current lane. The intervention is generated by the steering system via a smooth torque overlay which can easily be overruled by the driver. But, if the driver does not overrule that torque, the vehicle does not leave the lane.

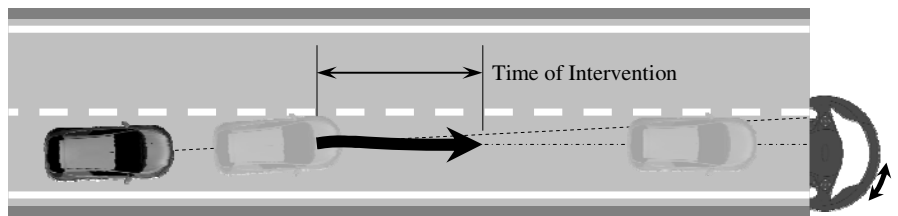


Fig. 2. Lane Keeping Aid (LKA)

Whereas the steering vibration for LDW is a feed forward function, the steering torque overlay for LKA is a more complex multi closed loop torque control, because the LKA System is not only applying a torque to the driver, but also needs to control the vehicle stable in case the driver does not overrule the LKA torque.

¹ Electric Power Assisted Steering (EPAS).

1.3 Camera Function Overview

The vision based main sensor of Lane Assist is a forward looking front camera, which also includes the LA main control unit. A simple track model is generated using image processing algorithms for road line marking detection. Based on the track information, LDW and LKA functions decide if the vehicle is going to leave the driving lane. It is also relevant for the calculation of the reference steering angle to keep the vehicle in the lane.

1.4 Reference Steering Angle

The closed loop vehicle control is enabled by the continuous calculation of the so called Reference Steering Angle. Looking from a birds eye perspective, this reference angle is the angle between the vehicle centre line and a line that connects the centre of the vehicle front axle with a point on the lane centre, which is in the distance D ahead of the vehicle (see also Fig. 3, D is about 50m).

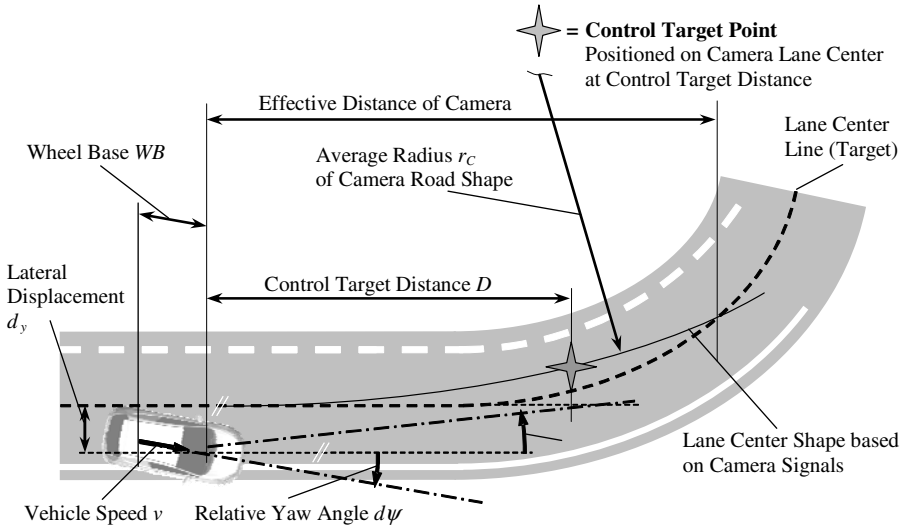


Fig. 3. Reference Steering Angle

The Reference Steering Angle is calculated based on three angle components:

- the angle to correct the vehicle's lateral displacement against the lane centre,
- the angle to align the vehicle relative yaw angle against the lane and
- the steering angle that is needed to follow the curvature of the road ahead.

The following formula shows, how the Reference Steering Angle δ_R is calculated:

$$\delta_R = \frac{d_y}{D} + d\psi + WB \cdot \frac{1}{r_c} \quad (1)$$

The values for of the Control Target Distance D and the Wheelbase WB are constant values, the Lateral Displacement d_y , the Relative Yaw Angle $d\psi$, and the Curvature $1/r_C$ are measured by the forward looking camera.

Along with the road curvature signal and the activation and deactivation trigger signals, the Reference Steering Angle is sent to the electrically powered steering system. In case LKA is active, the steering generates a torque that enables the vehicle to follow the Reference Steering Angle. Currently LKA is tuned not to drive the vehicle back to the lane centre, but to keep the vehicle at the inner of the lane border. The Reference Steering Angle behaves accordingly.

1.5 Warning and Intervention Zones

To activate and deactivate LDW and LKA, the camera detects the vertical vehicle position in the lane. Based on this position, the camera determines if the vehicle is in a Warning Zone for LDW or in an Intervention Zone for LKA. The Warning Zone is more lane outboard than the Intervention Zone as shown in Fig. 4. Warning or intervention starts once the lead tire enters the respective zone.

There are a number of conditions which suppress a warning or an intervention, i.e. a warning or an intervention is suppressed if the turn indicator is set to the respective side, or if the vehicle speed is below a threshold that indicates city driving.

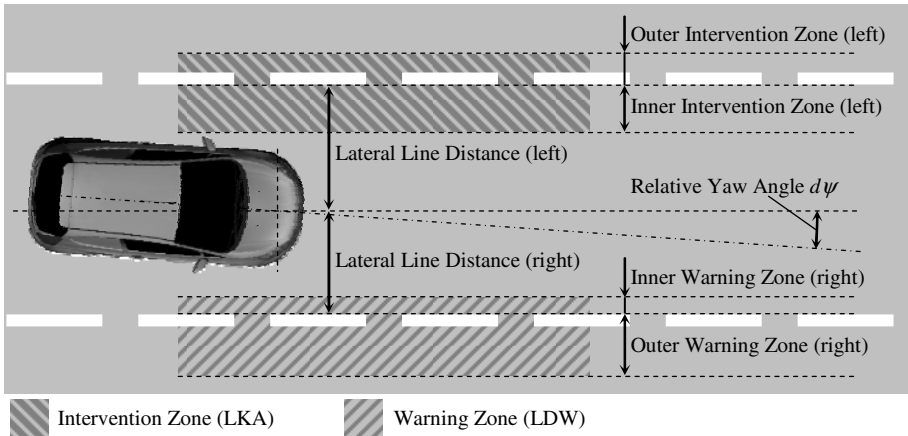


Fig. 4. Warning and Intervention Zone (not shown on both sides)

1.6 Steering Function Overview

To apply an additional torque overlay on the steering wheel, the Electric Power Assisted Steering (EPAS) is used. For LKA, the needed steering torque is calculated from the signals Reference Steering Angle and Curvature provided by the camera. For LDW, an oscillating steering torque overlay will be applied. An LA control unit processes LA feature activation as requested by the camera, but finally, steering torque overlay will only be applied, if generic LA requirements are fulfilled and checked by the EPAS system's base function.

1.7 Vehicle Control

The steering part of the LKA function shall ensure mainly two aspects: first, the electric motor needs to apply enough torque so that the vehicle follows the steering reference angle and second, it needs to be ensured under any circumstances that the applied torque is controllable by the driver. The first is achieved by adding another control loop that calculates the difference between the reference angle and the actual steering angle, and by adding a feed forward control path in parallel (see Fig. 5). The aspect of controllability is fulfilled by using a torque limitation function with a static and dynamic aspect.

Steering torques of a very low magnitude but high dynamics are as well controllable as higher torques with very low dynamics. Additionally, the steering controller has to protect itself from exceeding the controllable torque limit by entering an electronic failure mode in this case. To achieve this, the torque limiting is monitored, so that at the very rare event of the EPAS control unit's electric failure, no torque is applied by LKA.

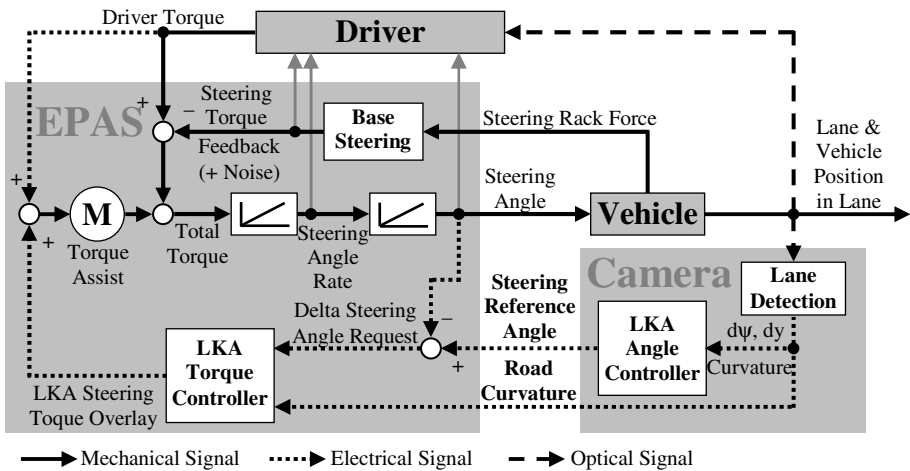


Fig. 5. Vehicle Control by Camera and EPAS as Actuator

1.8 Human Machine Interface (HMI)

The Human Machine Interface (HMI) mainly consists of a stalk located at the steering column close to the steering wheel and the display in the instrument cluster.

Single or all features of Lane Assist can be activated and deactivated using the stalk. Further settings can be done using the vehicle's general HMI systems such as menu toggle switches and voice recognition based menu navigation.

To inform the driver of the system's main states, the multi purpose display of the vehicle's instrument cluster is used as described below.

1.9 Cluster Function Example

An example display layout in the vehicle's cluster is shown in Fig. 6. The LA system status is indicated to the driver for both left and right sides by two dotted lines. If the

vehicle reaches the LKA speed range, grey circles beside the car change to grey dots. If the camera detects a lane marker line, the system becomes available on the respective side, indicated by green dots. In case of an LKA intervention, the dots become red.



Fig. 6. Cluster Display Example

1.10 Total System Architecture and Signals

The total system architecture overview in Fig. 7 summarizes the main components of the Lane Assist System: Human Machine Interface (HMI), Forward Looking Front Camera and Electrical Power Steering.

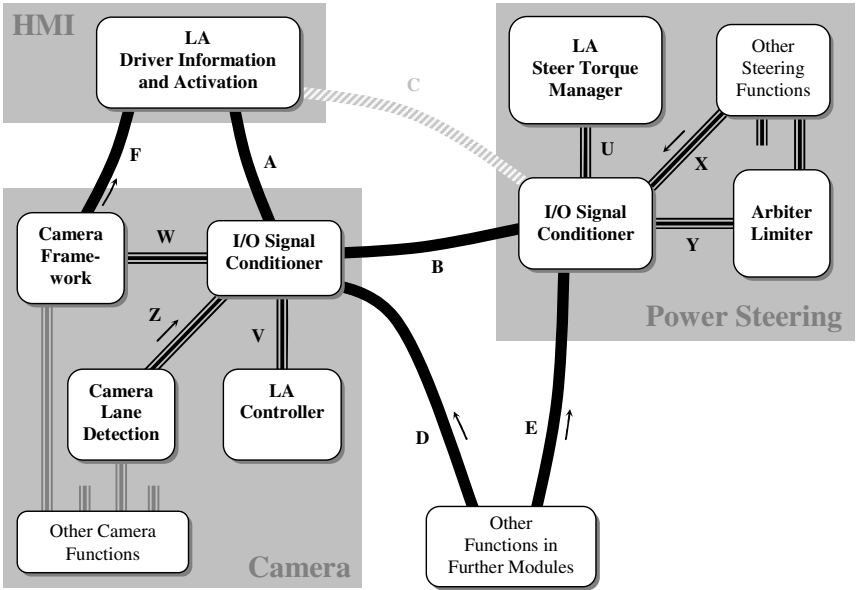


Fig. 7. Total System Architecture Overview

2 Safety Process

On the basis of the functions of the system, the use cases of the functions and also the non-use cases (see details in section 1), a safety process according to ISO 26262 needs to be performed necessarily to support the safety case. The safety case is the

collection of all arguments necessary to claim that the system is safe and does not contain unacceptable safety relevant risks.

The next sections show an example how such a process can be (and has been) performed in the real world. Some traps will be indicated which need to be avoided in order to generate a sound safe case with realistic resources and within a realistic schedule.

In order to maintain the focus on the system under development, the boundaries of the system and by this of the safety need to be defined and agreed by all involved parties. This determination and justified limitation is necessary in order to focus the resources to the tasks under responsibility of the team and to define unambiguous interfaces to other systems and to the safety case of such systems.

The safety process supports the development of a safe system. But it shall also support the system designers to make the system safe with the most practicable and feasible approach (i.e. the process needs to solve problems and define acceptable solutions). The safety process is an integrated part of the overall development process and it needs to advise how the safety shall be designed into the system (HW and SW), always bearing in mind that in most of the cases the best things in life are the simple things (i.e. the higher the complexity of the safety concept becomes the more is the risk that it contains inherent failures).

The next step of the safety process is the definition of the hazards of the system under assessment: A Preliminary Hazard Analysis (PHA) needs to be performed.

2.1 Preliminary Hazard Analysis (PHA)

Especially with regard to driver assistance systems, several designers will argue: “my system is designed to perform only some limited impact on steering and only under some few situations”. They do not see the point that a function can be performed (due to a failure) outside the defined usage range or outside a defined situation (use case).

The safety process and the PHA need to consider the overall picture and the worst case failures need to be identified. Even boundaries (limitations) which are defined by system engineering needs to be considered as a possible risk. This implies that the PHA needs to ask such questions. The problem of the PHA is that the PHA needs to ask all relevant questions without getting lost in an unmanageable amount of failure cases and scenarios.

ID	Hazard	Possible Consequences	S	E	C	ASIL	Safety Goal	Safe State
01	Unintended and unlimited steering Torque without feature activation.	The driver may lose the vehicle control and the vehicle may crash into obstacles or into other vehicles with high speed.	S 3	E4	C3	D	SG Ext 01: Unintended and unlimited (uncontrollable) torque shall be prevented.	LA Off
02	Unintended but limited steering Torque without feature activation.	The driver may lose the vehicle control and the vehicle may crash into obstacles or into other vehicles with high speed.	S 3	E4	C1	B	SG LKA 01: Unintended and limited (C1-controllable) torque shall be prevented.	LA Off

Fig. 8. PHA Extract of the LA System

The following rudimental facts of a possible system and the PHA process as defined by ISO 26262 (or also other safety standards) show the potential risk. The PHA starts from the following system information:

- Functions of the system (i.e. for the LA system described in section 1)
 - this can easily be more than 20 for a complex system
- Functional failures of each function
 - might be 5 for each function, i.e. “not supplied”, “supplied when not demanded”, “supplied too high”, “supplied too low”, “intermittent supply”, ...

The combination of the above stated functions and functional failures can already lead to a figure of 100 overall functional failures.

The next step is the combination of the functional failures with the situational and operational aspects. Situation catalogues show up to 200 (or even more) such conditions which can be considered for the safety process. As the safety process requires the consideration of functional failures under all situational conditions, a complete set of possible combinations leads to 20,000 PHA entries. Even if just one minute would be spend per PHA entry (without doing the documentation work), this results in 333 hours (i.e. 2 months), just for the generation of the PHA cases. This does not allow any discussion about classification or justification argument.

A complete and systematic coverage of all possible failure ↔ situation combinations could lead to the following PHA entry for a steering based driver assistance system. If (for example only) the generic situation catalogue shows:

- Low standing sun (bright sunlight)

This case may be useful for a function dealing with automotive light activation. If such a condition would be applied for the steering based driver assistance system, the PHA might investigate:

- Incorrect steering is applied at high speed with a low standing sun.

Let's assume that this is ASIL X. So the safety goal may ask that:

- Application of incorrect steering with low standing sun shall be prevented.

A safety concept for this could be:

- Suppression of the function when the sun is shining bright.

Formally this is sufficient to bring this hazard to an acceptable limit, but with a kind of functional limitation, which looks at least a little bit strange. It would require the implementation of a sun shine detection feature according to ASIL X – which can be possible but it would make the system very complex without any additional benefit for the system.

This small example shows that the work needs to be limited to a feasible and practical workload. On the other hand, even limitation features and other hazard relevant aspects need to be considered in sufficient detail.

The PHA covers some first traps which need to be avoided in the jungle of the safety process in order to make the safe case correct and manageable. Each failure which is made during the PHA process either leads to the non-implementation of a safety concept, if a hazard has not been identified or it leads to the effect, that the safety relevant resources are involved in superfluous workload dealing with unnecessary features and by thus keeping the resources away from the necessary safety tasks.

2.2 Safety Concept

The next step is the generation of the safety concept. The safety concept starts with the safety goal as identified within the PHA for each hazard (see Fig. 8, Column “Safety Goal”). The safety concept shows the system / technical solution to satisfy the determined safety goal and to reduce the hazard to an acceptable limit. The following questions need to be in the mind of the safety team when generating the safety concept:

- Is the safety concept adequate to deal with the hazard?
- Is the safety concept complete?
- Have I covered all the inputs?
- Are the safety relevant parameters defined?
- Are all the necessary requirements defined and accepted?
- Is the concept and the associated requirements sufficient to fulfill the safety goal?

The definition of the safety concept is the next trap of the safety process. The main consideration needs to be that the concept shall be as simple as possible. If the general features of the concept cannot be explained within a few sentences, the risk is that the correct definition and implementation itself will become so complex, that it becomes a risk by itself, or if all such functional risks can be avoided, it will be very work-intensive. Due to the limited number of resources, each increase in necessary workload is a potential source for failures during the safety process.

The safety concept is also an issue for making things easy of complex.

Example: the “unlimited and unintended application of a function” is rated to be ASIL D (see Fig. 8, Hazard 01). This could be realized by various safety concepts:

- A limitation feature could be implemented according to ASIL D.
- The unintended application of the function could be prevented according to ASIL D.
- A decomposed safety concept with both of the above features implemented according to ASIL B each could be taken.

For the majority of driver assistance functions, the computation of the function on the basis of various sensors (e.g. camera, radar, ultrasonic sensor) is a complex functionality whereas the implementation of a limiting function can be a simple feature for most functions. I. e. as ASIL D implementation requires a high workload, it should be the intention to link the highest workload to features with the lowest complexity. The decomposition concept saves workload for the implementation as both functions have to be implemented according to ASIL B only. However, as the safety goal is still

ASIL D, common cause considerations according to ASIL need to be performed under the inclusion of both ASIL B functions. I. e. the decomposition concept leads to some additional workload on system integration level.

The selection of an appropriate safety concept can, on the one hand, save a lot of workload, or on the other hand, it can lead to a lot of superfluous workload. Beside the technical decision about the best safety concept, it is necessary to document and validate the safety concept and the associated derived safety requirements.

ID	Hazard	ASIL	Safety Goal	Safety Requirement / Concept	Safe State	Toler. Time	Safety Parameter
01	Unintended and unlimited steering Torque without feature activation.	D	SG Ext 01: Unintended and unlimited (uncontrollable) torque shall be prevented.	FSR EPAS EXT The resulting steering wheel torque offset from all EPAS add-on features shall be limited that it is simple to control (C1 - torque ramp-in, torque control and torque ramp-out).	A&L C1 Limiter torque output shall be set to zero.	< 20msec	max C1 torque max C1 gradient
02	Unintended but limited steering Torque without feature activation.	B	SG LKA 01 Unintended and limited (C1-controllable) torque shall be prevented.	FSR LKA EPAS 001 EPAS shall limit the LKA steering wheel torque offset to an appropriate limit so that the worst case of the steering wheel torque offset is controllable in general (C0 - torque ramp-in, torque control and torque ramp-out).	LKA torque output shall be set to zero.	< 100msec	max C0 torque max C0 gradient

Fig. 9. Safety Concept (Extract) of the LA System

For the “major” hazards of the LA system a limitation of the feature to such levels has been selected in order to bring the safety implication of the function itself to an acceptable low level. The limitation features themselves have to be developed and implemented according to the PHA determined ASIL (see Fig. 8 and 9 above). The next steps is the breakdown of the safety concept to the system hierarchy and breakdown as already identified for the LA feature implementation (see Fig. 5 and 7).

The question to support the safety process and management is: How can all this information be generated in a structured way and how can all the associated technical and procedural aspects be documented and monitored? This question brings an analysis method into focus, which can be applied to support this process: The Fault Tree Analysis (FTA).

2.3 Fault Tree Analysis (FTA)

In several safety projects performed by *SystemA Engineering GmbH* for *Ford* (and also other companies), the method FTA has been applied to develop the safety concept and the associated safety requirements. Additionally, it supports the documentation process and the generation, validation and monitoring of the associated information. This covers the qualitative as well as the quantitative aspects of the safety requirements process.

The FTA shows the failure logic relevant to the hazard under assessment. The starting point of the FTA is the hazard and the associated safety goal as defined by the PHA (see Fig. 8). The failure logic shows the failure paths and by this the individual contributors which could lead to the hazard under assessment. The data flow is linked to the system logic as given in block diagrams (see Fig. 5 and 7) and later on in detailed models. This approach ensures that each level of the FTA can be linked to the associated block and signal flow within the design documentation.

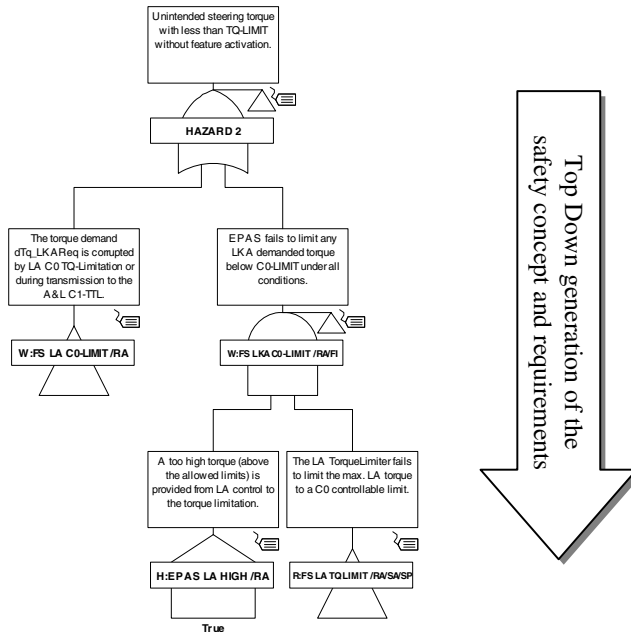


Fig. 10. Fault Tree (Extract for Hazard 02) of the LA System

At each step (gate) of the FTA it can therefore be analyzed and checked (validated) if the logic is correct. This means:

- Is the logic of the FTA (mainly OR or AND logic) correct?
- Are the models for the basic events correct (failure rate, fixed probability, dormant failure, etc.)?
- Are all inputs which are related to the gate covered?
- Is the ASIL selection / decomposition correct / acceptable?
- Are safety requirements identified and validated?
- Is the set of requirements sufficient to satisfy the safety goal?
- Are common cause failures present which could jeopardize a possible ASIL decomposition?
- ...

The FTA is historically mainly applied to compute and verify quantitative top event (hazard) probabilities on the basis of quantitative basic event data (failure rates, failure probabilities). By this, the FTA can be applied to perform the computation of the failure metrics relevant for the quantitative verification for the achievement of the ISO 26262 relevant results.

The same FTA performance can be applied to determine and validate the quantitative failure and monitoring requirements for individual contributors (basic events) necessary to achieve the ASIL relevant metrics for the safety goal as required by ISO 26262.

As the FTA is historically mainly a tool which supports in a graphical way the development of a failure logic and the computation of failure probabilities, it is only of limited help for purposes of requirement documentation and the associated management support. Therefore a management tool (data collection) needs to be linked to the FTA in order to support the validation and verification process of the safety case: the Validation & Verification (V&V) Plan.

2.4 V&V (Validation & Verification) Plan

The V&V plan basic structure can directly be extracted from the FTA, as the FTA shows the safety concept and all identified contributors to the hazard under assessment. In the current projects, an *Excel* file is linked to the FTA. The *Excel* file (the so called V&V plan) is generated by means of the *Isograph FT+* software export feature. The *Isograph FT+* export / import feature supports the combination of FTA for the generation of the safety concept / structure and an *Excel* file for the collection, documentation and management of the qualitative arguments, status and action items.

There are two sets of requirements which are associated with the safety concept / fault tree logic:

- Safety goals and technical requirements
- Process relevant rules and requirements

The V&V plan is considered to show all the details associated with each data element. The V&V plan is applied to show all the data elements for which safety relevant information is necessary. It includes the information as far as necessary from the relevant source in order to enable the reader to understand the arguments. A reference to the source of the information is mandatory in order to allow the investigation of more detailed arguments when considered to be necessary.

The following type of information is collected within the V&V plan (example):

- Is the safety concept derived and validated?
- Are requirements derived and documented in the applicable specification?
- Is the safe state defined?
- Is the maximum allowed failure tolerance time defined, justified and validated?
- Are safety relevant parameters (e. g. limitation of a function) defined, justified and validated?

- Are verification measures for each requirement determined, are they sufficient and are they in place?
- Are the actions necessary to close issues identified with responsible and deadline?
- Are the actions closed in time?
- ...

Experience with the application of a safety process with various involved partners has shown, that the generation and distribution of tasks and actions like “partner x has to implement a feature according to ASIL D” or “verification evidence according to ASIL D shall be provided” is only of limited help. It has been proven to be effective and useful to be as specific as possible in order to have a common understanding of the needs and the expected result for each entry.

The V&V plan is therefore not a simple repetition of the generic process requirements as given within the applicable safety standard (e. g. ISO 26262). It provides project specific rules and details (for example):

- What information is expected?
- Who has to provide the information?
- Which is the form in which the information needs to be provided?
- What is the deadline for providing the information?
- Who has to validate the information?
- Why is the information sufficient?

Name	Description	Goal / Requirm.	Process Safety Requirements	Who	What	Due Date
W:LK A- F01- H02- S01 /RA	Unin- tended steering torque with less than TQ- LIMIT.	FSR LKA EPAS 001 EPAS shall limit the LKA steer- ing wheel torque off- set to an appropriate limit so that ...	Completeness ... including the definition of the safety goal shall be assessed and verified.	SST	The PHA V1.08 has been ... validated at review meeting.	xxx
			The FTA shall be performed ... checked and validated (i.e. completeness and correctness).	SystemA SST	FTA Version 0.13 has been distributed for validation purpose.	xxx
			The requirements derived from the safety concept ...	FoE SystemA	The safety requirement has been harmonized with specification ...	
			Safety relevant parameters for “C0 = controllability in general” need to be determined and validated. - Max allowed torque gradient - Max allowed absolute torque	FoE Suppl. X SST	Verification test plan needs to be set up to determine the relevant parameters.	xxx

Fig. 11. V&V Plan (Extract for Hazard 02) of the LA System

The feedback from each contributor is collected within the V&V plan. The result is reviewed by the safety team and the result of the review is included within the V&V plan. This includes the documentation of associated actions necessary to close the issue. Fig. 11 is a simplified example how such a V&V plan could look like.

By this approach the FTA in conjunction with the V&V plan is the bracket covering the whole safety case for the system under assessment. It has been proven to be

effective in order to maintain an overview of the technical concept and requirement as well as the status of the activities and the associated open issues. Actions determined by this process are linked to the relevant field of the V&V plan and monitored as part of the V&V process.

2.5 Failure Modes, Effects and Diagnostic Coverage Analysis (FMEDA)

The system level safety analysis (FTA in our case) and the associated information (V&V Plan) needs to be linked to the lower level safety analysis and information as provided by the relevant supplier. This is done from the system side by means of the generation of qualitative and quantitative safety requirements which are linked to the associated fault tree gates and events. The supplier needs to link his safety process and analysis to the requirements as provided at the system interface. The analytical interface can be realized on the supplier side by means of an FTA or an FMEDA. The following sections describe the FMEDA approach. However, the general issues are also applicable for the FTA approach.

The FMEDA is a Hardware (in the following: HW) driven method which determines the impact on safety on the basis of possible HW failure modes and the associated means of failure detection and failure reaction.

Most FMEDAs (as reviewed by the author until today) do not distinguish between the various safety relevant functions of the device under assessment. This despite the fact, that not all resources are used by all safety relevant functions in the same way. I. e. the HW impact, the associated failure detection capability and by this the quantitative contribution to the individual safety functions of such a device may be very different.

Let's take the example of a sensor which is developed according to ASIL D. This ASIL D implies that at least 99% of the relevant failure are detected and brought into the safe state (sensor deactivation). The contribution of the sensor to failure cases like "provision of undetected corrupted data (A)" will be covered by this statement. If the loss of the signal is also safety relevant (the safe state is defined as "sensor deactivation (B)"), then this function cannot be supported by the sensor according to ASIL D, because > 99% of the failures will lead to this reaction. I. e. the FMEDA extract for the failure case "A" needs to be very different to the FMEDA extract for failure case "B".

Note: The generic ASIL D device is not possible. The "attribute" ASIL D (or others) are only given for a dedicated application.

Under consideration of this issue, a method has been applied to link a potential supplier FMEDA to the associated basic events of the system level (OEM) FTA. It is hardly possible to perform a HW based FMEDA for all software based contributions to the system level FTA. Therefore assumptions need to be made to assess the HW in such a way, that the most realistic but still conservative failure rate and failure detection information is transferred from the FMEDA to the FTA.

The feedback from FMEDA to FTA and hence to the system level safety case is monitored by means of the V&V plan. The FMEDA derived quantitative information is then applied by the OEM on safety goal level to determine the metrics regarding the ASIL relevant quantitative requirements of the ISO 26262 standard and the verify the achievement of the safe goal (see Fig. 12 below).

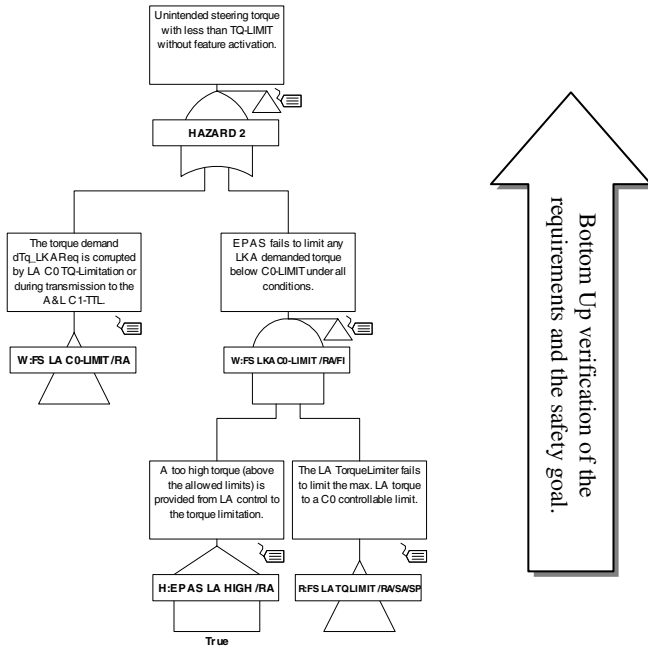


Fig. 12. Safety Verification by FTA (Extract for Hazard 02) for the LA System

3 Conclusion

Experience with the conduction of the safety process for innovative and complex functions at *Ford* has shown that it is possible to perform an integrated safety process without getting lost in the details of the process. Safety experts have been involved from the first definition of technical concept until the final end of safety verification. The tailoring of the process to the necessities and the application of a customised tool environment has provided evidence, that the complete safety relevant activities can be performed and monitored with an acceptable level of effort and manpower.

Benchmarking Software Requirements Documentation for Space Application

Paulo C. Vêras¹, Emilia Villani¹, Ana Maria Ambrósio²,
Rodrigo P. Pontes¹, Marco Vieira³, and Henrique Madeira³

¹ Department of Mechanical Engineering, Instituto Tecnológico de Aeronáutica,
Praça Marechal Eduardo Gomes, 50,
12228-900, São José dos Campos-SP, Brazil
{pcv, evillani, rpastl}@ita.br

² Ground System Division, National Institute for Space Research,
Av. dos Astronautas, 1758,
12227-010, São José dos Campos-SP, Brazil
ana@dss.inpe.br

³ DEI/CISUC, University of Coimbra,
3030, Coimbra, Portugal
{mvieira, henrique}@dei.uc.pt

Abstract. Poorly written requirements are a common source of software defects. In application areas like space systems, the cost of malfunctioning software can be very high. This way, assessing the quality of software requirements before coding is of utmost importance. This work proposes a systematic procedure for assessing software requirements for space systems that adopt the European Cooperation for Space Standardization (ECSS) standards. The main goal is to provide a low-cost, easy-to-use benchmarking procedure that can be applied during the software requirements review to guarantee that the requirements specifications comply with the ECSS standards. The benchmark includes two checklists that are composed by a set of questions to be applied to the requirements specification. It was applied to the software requirements specification for one of the services described in the ECSS Packet Utilization Standard (PUS). Results show that the proposed benchmark allows finding more with a low effort.

Keywords: benchmark; software requirements quality; space systems; ECSS standards; Packet Utilization Standard.

1 Introduction

Writing a high quality software requirements specification (SRS) is one of the hardest phases of the development life cycle of a software system [1]. Ill-defined requirements contribute to significant schedule delays and cost increases [2]. Furthermore, there are evidences that errors in the requirements can lead to serious problems during software development and usage [3].

Some problems in the specification of requirements, such as the occurrence of conflicts, can be resolved by using formal specification languages or formal methods. However, they are hard to be understood by non-experts, which limits their practical application to some restricted domains [4]. Despite the many problems of using natural languages, such as the lack of formality and ambiguity, this is still the most used mean to express software requirements and to communicate those requirements to customers [5]. Software for space systems is no exception and requirements specification based on natural languages are still widely used.

The European Cooperation for Space Standardization (ECSS) [6] provides a set of standards to support the development of space products. These standards cover a broad range of application areas, such as mechanical, software engineering, control engineering and ground system. Among the ECSS standards, ECSS-70-41A [7] proposes the Packet Utilization Standard, also known by its acronym PUS, which addresses the utilization of telecommand and telemetry for the remote monitoring and control of spacecrafts. It defines a set of services that covers all the fundamental requirements for spacecraft operation. PUS defines the protocol for the communication between the spacecraft and the ground segment (i.e. the control centre in Earth).

Leveson [8] studied in detail five software-related accidents in space systems. Among the factors that contributed to the accidents, there are some related to software engineering, such as poor or missing specifications, and inadequate review activities. Common sense is that these problems can be mitigated by applying a standardized process of assessing or reviewing the quality of the requirements. This work proposes a starting point to define a benchmark to be applied to the software requirements specification of space systems that adopt the PUS.

Typically, a benchmark is a systematic procedure to assess measures related to the behaviour of a computer system or computer component, and aims at comparing alternative solutions or evaluating its characteristics against a reference model (e.g., a standard). While a performance benchmark is composed by a workload and measurements, a dependability benchmark adds an additional component: a faultload that represents real faults experienced by systems in the field. The system under benchmarking (SUB) and the benchmark target must be well defined, as well as the benchmarking rules and procedures. The benchmark shall be validated taking into account its representativeness, repeatability, reproducibility, portability, non-intrusiveness, scalability, time (that has to be as short as possible) and cost (the perceived value shall be higher than the associated costs).

As the target of the benchmarking approach proposed in this work is a document, there is no workload to run or faultload to inject. Instead, a checklist composed by questions replaces the workload and is used to obtain measures that portray specific characteristics of the software requirements specification. The purpose of this benchmark is to guide/help the review of the requirements for the onboard computer software. This review is typically performed at the beginning of the space software development process. The proposed benchmark provides a standardized way for assessing the quality of the requirements and their accomplishment regarding the PUS ECSS standard.

The proposal of a benchmark for software requirements is motivated by the high costs of conducting a software requirement review over low quality software requirements specifications and by the very high costs of the rework caused by poorly written

requirements. In fact, problems detected at the end of the software development cycle may compromise the entire space mission timeline. Moreover, applications with high complexity, like space systems, demand the use of standards in order to guide the whole development life cycle. Naturally, if a project is in accordance with key standards, the overall software quality tends to increase.

Particularly, the paper discusses the definition of the benchmark checklist. As this checklist greatly influences the benchmark representativeness, we use two sources of information for defining two different checklists. The first is based on the description of the services in the PUS and basically aims at guaranteeing that the SRS complies with the standard (regarding its content). The second is based on the CoFI methodology [9], which aims at verifying whether the SRS covers system failure situations.

To show the feasibility and applicability of the proposed benchmarking approach, we use the telecommand verification service described in PUS as a case study. The two checklists are applied to a concrete software requirements specification of a space product. The results are analysed and discussed in detail. The idea of the whole work is to propose a benchmark to assess the overall SRS by defining checklists that cover all of the services described in PUS. Although the proposed benchmark is restricted to software that follows the PUS, the methodology we used to create it can be easily extended to embedded software that follows other standards.

The paper is organized as follows. Section 2 reviews related work. Section 3 presents the definition of the two checklists. Section 4 presents the concrete example for the telecommand verification service and discusses the results obtained from the application of the two proposed checklists. Section 5 concludes the paper.

2 Related Work

Many works aim at studying attributes of software requirements quality, such as: correctness, completeness, consistency, clarity, and feasibility. Halligan [10] presents a structured methodology for measuring the quality of requirements individually and collectively, based on each requirement statement (which provides a score for individual requirements). Davis [11] proposes metrics to measure the quality of software requirements following an approach based on the assessment of each requirement according to quality attributes similar to the ones proposed in [10]. This approach provides a score that reflects the quality of the overall requirements document. However, Davis does not define the point from which the document is considered good enough to proceed to next phase of the development process. Knauss [12] performed a study based on the metrics defined by Davis in order to find out a threshold that determines whether the requirements document can be considered good enough to serve as a foundation for project success.

Hofmann and Lehner [13] conducted a field study to identify requirements engineering practices that contribute to project success. Boehm [14] and Wilson [15] developed tools to help developers analysing requirements and identifying conflicts among them, as well as tools to assess requirements by searching for terms that are quality indicators. Kim [4] proposed an approach for systematically identifying and managing requirements conflicts based on requirement partition in natural language.

Gilliam [16] focused his work on the development of a software security checklist for the software life cycle, including, among others, the requirements gathering and specification process. Sheldon [17] discusses the validation of a SRS based on natural language

in terms of completeness, consistency and fault-tolerance. A method for detecting semantic level inconsistency in software requirements based on state transition is described in [18].

The works presented above (and many others) deal with SRS quality assessment in general and try to perform this by using some new method, some combination of existing methods or by proposing some new process of assessment. Although there is a considerable number of works in the literature, to the best of our knowledge none target specifically critical embedded software, neither systems that follow some specific standard. Furthermore, none of the existing works are based on a benchmarking process, with well-defined metrics and a very well contextualized scenario.

The Transaction Processing Performance Council (TPC) [19], an organization composed by major vendors of database and transaction processing software, has a long-standing tradition in proposing and managing performance benchmarks. Although, dependability measures have been largely absent of the TPC benchmarking effort, the TPC benchmarks have influenced many dependability benchmarks proposed so far (see, for example, the book edited by K. Kanoun and L. Spainhower [20]). Dependability benchmarking has been mainly focused in the evaluation and comparison of the dependability of COTS (Component Off-The-Shelf) and COTS-based systems in embedded, real-time and transactions systems. Both academy and industry have proposed benchmarks focusing on a wide range of types of systems [20]. Examples of benchmarks for embedded systems are [21] and [22]. The former focused on real-time kernels for on-board space systems and the latter addressed automotive systems.

Existing benchmarks aim at assessing and/or comparing computer systems or components. The purpose is to assess products, i.e., the systems under benchmarking are final products and they do not take into account the quality of the requirements documentation or the impact of low quality requirements in the final product. Our work opens a new research direction by extending the benchmarking concept to software documentation, namely requirements specifications for space systems.

3 Benchmarking Approach for Software Requirements

Although based on the concepts of dependability benchmarking, which aim at assessing and comparing key features of the behaviour of a computer system or component, the purpose of our approach is to assess software requirements specification, thereby requiring the redefinition of the main elements of a typical dependability benchmark [20], [23]. Besides allowing comparing some quality attributes of software requirements specifications, it allows comparing a given SRS against a reference model (in our case, the PUS standard).

In our approach, the workload and faultload are replaced by a checklist that consists of a set of questions to assess key features of the software requirements specification under benchmark. The purpose of these questions is to verify whether the SRS is in accordance with the PUS [7] (PUS-based checklist), as well as to verify whether the SRS describes the actions that the system shall perform in the case of a failure (CoFI-based checklist). All of these questions accept just “yes” or “no” as answer.

The measure of the proposed benchmark is the number of questions answered “yes” when applied to the software requirements document under benchmarking. The amount of time necessary to the specialist to execute the benchmark is also assessed in order to provide an idea of the ease of application of the benchmark.

Although the checklists proposed for this benchmark takes into account only the telecommand verification service of the PUS, they can be easily extended to the other 15 basic services of the PUS, which constitute the capabilities to be implemented on-board a satellite along with the corresponding monitoring and control facilities on the ground. The telecommand verification service provides the capability for explicit verification of each distinct stage of the execution of a telecommand packet, from the on-board acceptance through the completion of the telecommand execution. This service consists of the following stages: (1) acceptance of the telecommand by the destination application process, which includes syntax, validity and feasibility checking, (2) telecommand execution start, (3) intermediate stages of execution progress, (4) telecommand execution conclusion. The telecommand verification service shall generate a report if a telecommand fails at any of its identified stages of execution. It shall also generate a report of successful completion of the same stages if this has been requested in the acknowledgment flags in the telecommand packet header. These reports shall provide auxiliary data for the ground system to fully understand the report (e.g. to identify the nature and cause of a telecommand failure).

As mentioned before, the proposed benchmarking checklists were defined by using two different methods, which allows comparing the results obtained from the application of each one. The first is directly based on the telecommand verification service of the PUS and on the description of the telecommand packets structure. The second was generated by applying the CoFI methodology to the PUS.

3.1 Checklist Based on the PUS

The questions that compose this checklist were defined by analysing the specification of the telecommand verification service. For each mandatory statement of the standard, one or more questions were defined. When necessary, other parts of the standard were consulted to gather more information to define the questions (e.g., when the telecommand verification service makes reference to the structure of the telecommand and telemetry packets). For example, there is a question to verify if the SRS defines the type and size of each field of the telecommand and telemetry packets.

The resulting checklist was reviewed by a developer with a large experience (more than 6 years) that works on space application software in the context of PUS and by a researcher on space systems (whose knowledge is described in section 4.1 of this paper, specialist 3). The suggestions and recommendations received from the specialists were then incorporated into the final checklist.

This checklist verifies whether the SRS follows the PUS by taking into account the content of the PUS itself and of the SRS. The checklist is composed of a set of 92 questions that accept as answer only “yes” or “no”, where “yes” means that a given requirement specification complies with the PUS and “no” means that the requirement specification does not comply with the PUS or has some ambiguity in the context of the question. Three examples of questions are:

- Does the requirement specification define the telecommand verification service type as 1?
- Does the specification state that this service shall check whether the Packet Length field value is at least 16 octets and at most 65535 octets?
- Does the requirement specification state that the code 0 of the failure acceptance report means “illegal APID (PAC error)”?

As an example, let's take the first question above. Each one of the services defined in PUS has a number that represents its service type and that distinguishes each service in a unique manner. The telecommand verification service shall have its type defined as 1. If the answer given to this question is “yes”, this means that the SRS follows the standard. The full PUS-based checklist is available at [25]. An important aspect is that the PUS does not state all the features to be mandatory. This standard has some points that are optional and specific to the mission. In this kind of situation, the user can mark the question in the checklist as not applicable to the SRS under benchmarking. Therefore, the resulting percentage of “yes” answers is referred to the applicable questions considered by each specialist.

3.2 Checklist Based on the CoFI Methodology

The second checklist was defined by using the CoFI methodology. CoFI stands for Conformance and Fault Injection as it drives the conformance and robustness test cases generation. This methodology guides a tester to create simple finite state machines (FSMs) starting from a textual description, such as the software requirements specifications. Instead of relying on a single behaviour model of the system, it guides the creation of a set of small FSMs representing partial behaviours to cover test objectives. The first step is to identify the set of services that the system provides and then create different FSMs for each service, taking into account the following classes of inputs: (i) normal, (ii) specified exception, (iii) inopportune input (i.e., corrects but in wrong moments), and (iv) invalid inputs caused by external faults. Thus, decomposition of the system complexity in small FSMs is driven in terms of: (i) the services and (ii) types of behaviour, namely, normal, specified-exception, sneak-path, and, fault-tolerance. Once the FSMs are defined, they are submitted to the ConData tool [24], which can automatically generate test cases, as those used for protocol testing. In our work, the description of the service in PUS is used to create the FSM for the normal, specified exception and inopportune input behaviours. Then, instead of using the FSM of CoFI to generate test cases, we used it to generate questions to compose the benchmark checklist.

As mentioned before, this checklist does not take into account the content of the SRS and of the PUS. Instead, it considers the PUS as a kind of “black box”, it just considers the functional behaviour that the software shall have by verifying the output response to the provided inputs. It does not verify the steps that the software performs to accomplish what the standard defines.

The questions derived from the CoFI methodology look for evidences in the requirements specification that show that the developer considered not only the normal behaviour but also all the important cases of invalid inputs or sneak paths. The rationale behind it is to discover potential ‘holes’ in the requirement specification that would lead to the identification of failures when testing of the final software product.

The FSM models the behaviour of the telecommand verification service when communicating with both the ground station and the on-board application process. The ground station sends a telecommand to be executed on-board and wait for responses about the execution status. The telecommand verification service receives the telecommand and sends it to the application process. The application process is the part that actually executes the command and informs the telecommand verification service about its status. Based on the information provided by the application process, the telecommand verification service generates success or failure reports that are sent to the ground station via telemetry.

Each possible transition of a FSM represents an expected input/output relationship and originates a question. The question must characterize the initial state of the transition and the expected input/output, as well as the specific conditions under which the transition occurs.

Fig. 1 presents the FSM for the normal behaviour of the telecommand verification service. Events like TC_Arrival represent the arrival of a telecommand sent by the ground station to the telecommand verification service. Actions like RepSuccAcc,

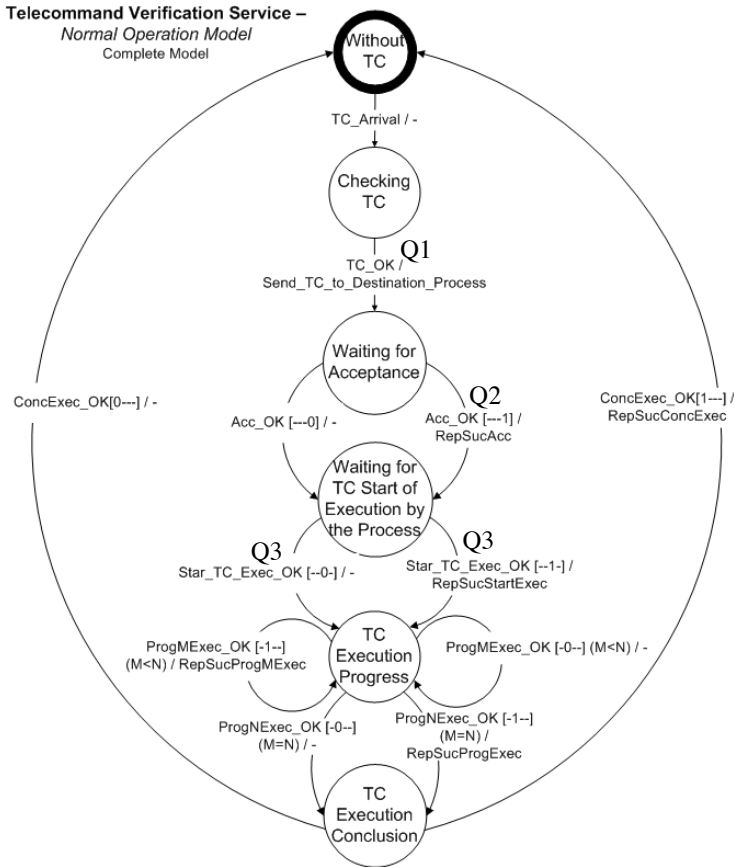


Fig. 1. Normal behaviour FSM

RepSucProgExec are different reports carried into the telemetry and sent to the ground station. The events Acc-OK, Start_TC_Exec_OK are related to the communication between the telecommand verification service and the application process. The numbers, such as [1---] and [-0--], are reference to the criteria specified in the PUS for generating a particular report of success.

Some examples of the questions defined for the FSM of Fig. 1 are:

- Does the requirement specification state that the telecommand verification service shall send the telecommands received from the ground to its destination process after its checking? (Q1)
- Does the requirement specification state that the telecommand verification service shall send a report of success acceptance to the ground station if this is requested through the first bit set? (Q2)
- Does the requirement specification state that the verification of the TC execution starting shall occur after the acceptance confirmation by the destination application process? (Q3)

Fig. 2 illustrates the sneak paths behaviour. Basically, the sneak paths model considers the case of receiving a valid response from the application process at the wrong moment. Some examples of questions defined from the sneak path FSM are:

- Does the requirement specification state the action of the telecommand verification service if it receives a confirmation of execution conclusion from the application process when it should receive a confirmation of execution start? (Q4)
- Does the requirement specification state the action of the telecommand verification service if it receives a confirmation of telecommand execution progress from the application process when it should receive a confirmation of execution conclusion? (Q5)
- Does the specification state the action of the telecommand verification service if it receives a confirmation of telecommand execution conclusion when it should receive a confirmation of successful acceptance? (Q6)

By using the CoFI methodology, a list with 36 questions was generated. Additional questions were added to verify sensible points indirectly suggested by the system modelling. Some examples are:

- Does the specification state that the confirmation of the progress given by the target application process shall identify the concerned step number?
- Does the requirement specification define the action of the service if some answer of the application process is not received?

The full CoFI-based checklist is available at [25]. By analysing both PUS and CoFI-based checklists, we conclude that the major contribution of using the CoFI methodology is the definition of the questions based on the sneak path FSM. The PUS-based checklist contemplates almost all of the questions based on the normal and the exceptional behaviour FSMs. However, none of the questions generated from the sneak path FSM are defined in the PUS-based checklist. Therefore, the portion of

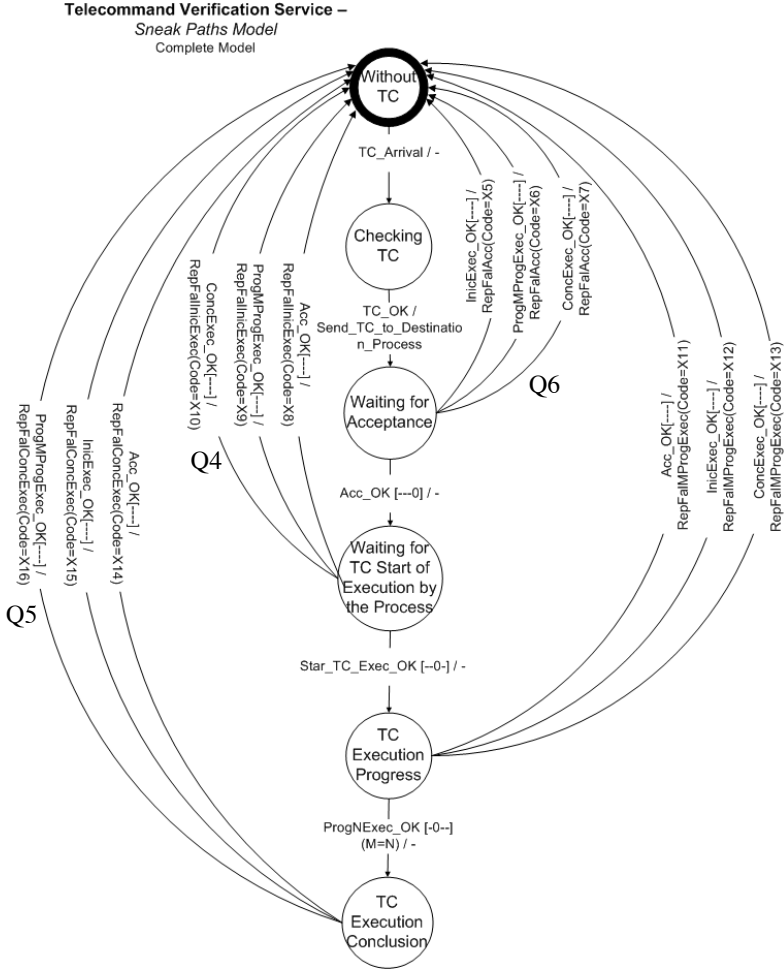


Fig. 2. Sneak path FSM

the CoFI-based checklist defined from the sneak path FSM can be considered complementary to the PUS-based checklist.

4 Example of Application

This section presents a concrete example of the application of the benchmark and discusses the results obtained.

4.1 Case Study

The example used in this work consists of a software requirements specification that describes the functional requirements of the telecommand verification service. This

example is taken from a real space project under development at INPE (Instituto Nacional de Pesquisas Espaciais, Brazil). This specification does not take into account the whole software to be implemented, but just the piece of software that implements the telecommand verification service. Some examples of requirements are:

- R4.3 - When a failure occurs during the acceptance of a telecommand, the OBDH (On-Board Data Handling) software shall generate a report informing this occurrence.
- R4.3.1 – The report mentioned in R4.3 shall have subtype 2.
- R4.3.2 – The format of the packet data field of the report mentioned in R4.3 shall conform to the format specified in clause 6.3.2 of the ECSS-E-70-41A standard.

The specification has a total of 39 functional requirements and is limited to the telecommand verification service. Some requirements make reference to specific sections of the PUS standard. When applying the checklists, these sections of the PUS standard were considered as part of the requirements specification.

Four specialists applied independently the benchmark to the requirements specification. The purpose was to compare the results of the specialists in order to verify if the process is easily applicable, repeatable and if its application is independent of possible interpretations from the different specialists. The specialists were also allowed to register additional comments to each question of the checklist.

All the specialists have knowledge in the PUS ECSS standard. Specialist 1 has been a professor and researcher in mechatronics and aerospace systems for 5 years and has experience in modelling, validation and verification of systems. Specialist 2 is a MSc student in aerospace systems and has worked with space projects that adopt ECSS standards. Specialist 3 has been researcher in space systems for 25 years and has worked with ground segment system, having also a broad experience in systems independent validation and verification. Specialist 4 is a PhD student and has worked with aerospace systems for 5 years, both in the academy and industry. All specialists had a participation in the discussion about this work since its preliminary steps.

Each specialist read the software requirements document one time before start answering the questions of the checklists. Particularly, specialist 2 is the one who actually wrote the requirement specification. The total time needed to apply the checklist (including the first read of the document) was measured. In practice, the attributes measured in the application of each of the checklists are the number of answers “yes” and the total time necessary to apply it.

4.2 Results and Discussion

The results obtained with the application of the PUS-based checklist by the four specialists are shown in Table 1, where the column A.Q. contains the number of Applicable Questions considered by the specialist.

By analysing the answers of the four specialists, we observed that 68 questions had the same answers (either “yes” or “no”) for all specialists, and 7 other questions had the same answers for only three of the specialists. These results indicate that the remaining 17 questions may be ambiguous and may need to be revised.

Table 1. Result of the application of the PUS-based checklist

	“Yes”	A.Q.	“Yes” (%)	Time (hh:mm)
Specialist 1	76	92	82.6	00:44
Specialist 2	74	87	85.1	00:37
Specialist 3	68	89	76.4	01:29
Specialist 4	68	87	78.2	00:42
Average	71.5	88.7	80.6	00:53

The analysis of the comments provided by the specialists allowed understanding the reason for the discrepancy in the positive answers. Specialists 3 and 4 were more rigorous in the interpretation of the SRS, while specialists 1 and 2 were more flexible. One example is the case of Specialist 4 considering two requirements in conflict. As a consequence, he answered “no” to four questions of the checklist. The other 3 specialists answered “yes” to the same questions and detected no conflict. Even in the presence of some discrepancy among the answers of the specialists, we can see that the maximum difference between the average percentage of “yes” (80.6%) and the percentage of “yes” for any specialist is less than 5%, which is quite acceptable once that the checklist and SRS have a margin to interpretation.

The mean time necessary to apply the PUS-based checklist to the SRS is 53 minutes, what gives us an idea of the ease of application of the 92 questions. In average, only three questions were considered not applicable to the SRS under assessment. In general, the questions that were considered not applicable are questions about aspects that are optional in the PUS. Thus, if the SRS does not intend to implement that point, the question about it is not applicable.

The results obtained with the application of the checklist generated by using the CoFI methodology are listed in the Table 2. In the case of this checklist, there is no evaluation of the applicability of the questions because all of them are applicable. The total number of questions whose answers were the same for the four specialist is 17 (again either “yes” or “no”), and the total number of (remaining) questions whose answers were the same for only three of the specialists is 14.

Table 2. Result of the application of the CoFI-based checklist

	“Yes”	“Yes” (%)	Time (hh:mm)
Specialist 1	13	36.1	00:16
Specialist 2	14	38.9	00:27
Specialist 3	7	19.4	00:30
Specialist 4	12	33.3	00:22
Average	11.5	31.9	00:23

Except for the questions generated by the sneak paths FSM, whose answers were all the same ones, the application of the questions to the requirements list were interpreted in different ways by the 4 specialists. Furthermore, the specialist 3 was much more rigorous than the other specialists. The main reason for the different interpretation is that the requirements are strongly based on the PUS standard and the PUS

standard leaves implicit the interaction with the application process. As a consequence, the requirements also leave this part poorly detailed. Some specialists considered the implicit interaction and answered “yes”, while other considered that there was no clear answer to the same question and answered “no”.

A good example is the question: “*Does the requirement specification state that the successful report of telecommand execution starting shall be generated after the confirmation of the execution starting sent by the target application process?*”. The requirements contain no information about a confirmation that should be received from the target application process. On the other hand, it says explicitly that the successful report should be sent after the execution start. This has resulted in different answers from the specialists.

Despite the SRS used in this work is from a real project, the number of answers “yes” given in the application of the CoFI-based checklist is very low. This can be explained by the fact that this checklist looks for system failure situations that the PUS does not cover. In this way, the SRS does not describe this kind of situation. The idea of applying these checklists during the software requirements definition phase is exactly to find this kind of weakness in the SRS.

As can be seen in tables 2 and 3, the mean time required to apply the PUS-based checklist is more than two times the time required to apply the CoFI-based checklist. This is mainly due to the number of questions of each checklist (92 question of the former, against 36 questions of the later).

It is important to emphasize that the questions in the PUS-based checklist aim to verify whether the software requirements specifications complies with the PUS. On the other hand, the CoFI-based checklist goes beyond the standard. Through the development of the FSMs, this methodology provides a mean for verifying key aspects that are not explicitly approached by the standard, such as the sequence of messages changing between the telecommand verification service and the target application process. In addition, the sneak path FSMs provides a way for thinking about operational conditions not handled by the PUS and allows verifying the robustness and dependability of the system.

This is the reason why the average percentage of “yes” answers in the CoFI checklist (31.9%) was much smaller than in the PUS-based checklist (80.6%). The software requirements specification used as case study does not describe how the system shall behave in the presence of faults. As the PUS-based checklist did not contemplate this, the number of “yes” answers was higher. On the other hand, the CoFI-based checklist is not as detailed as the PUS-based checklist.

Results show that the proposed checklists are quite complementary and can be merged to form a more comprehensive and representative checklist. Also, the definition of a representative checklist for a benchmark for requirements specifications must take into account different sources of information. It is important to emphasize that the results obtained by applying the proposed benchmark can be used as a feedback to the development team. In fact, those results provide a measure of the completeness, robustness and accomplishment with the followed standards (ECSS standards). The negative answers given to the checklist can be used to improve these aspects in the requirements specification.

5 Conclusion and Future Work

This work proposed a benchmarking approach for software requirements specifications for space applications. This benchmark is based in two checklists that help

assessing specific characteristics of a requirements specification. As a starting point, the proposed work is restricted to software that implements the telecommand verification service of the PUS, an ECSS standard.

The definition of the first checklist was based on the analysis of PUS specification and the second was based on the CoFI methodology. The former evaluates whether the software requirement specification complies with the PUS, the latter goes beyond this and verifies whether the document handles situations such as the presence of faults. Although the PUS-based checklist does not approach this kind of situation, it has detailed questions about telecommand and telemetry data that verify the compliance of the software requirements with the standard.

The proposed approach has been applied to a software requirements specification of a real space project currently under development at INPE. Four specialists have applied the proposed benchmark to provide a more consistent evaluation of the study. Results showed that the piece of the CoFI-based checklist that aims at verifying whether the SRS handles situations of presence of faults is complementary to the PUS-based checklist. The PUS-based checklist already contemplates the remaining parts of the CoFI-based checklist.

To each question of the checklist criticality could be associated, according to the severity of the aspect treated by that question. As a future work, we are planning to define how to measure the result of the benchmark considering that each question has a weight. Additionally, a minimum threshold value to decide whether the requirement specification is good enough to pass to next project phase needs to be determined.

Acknowledgment

This work was partially supported by CAPES – Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, AEB – Agência Espacial Brasileira, and by CISUC – Centro de Informática e Sistemas da Universidade de Coimbra.

References

1. Brooks, F.P.: No silver bullet: essence and accidents of software engineering. *IEEE Computer* 20(4), 10–19 (1987)
2. Mission critical systems: defense attempting to address major software challenges. US General Accounting Office (1992)
3. Heitmeyer, C.L., Jeffords, R.D., Labaw, B.G.: Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology* 5(3), 231–261 (1996)
4. Kim, M., Park, S., Sugumaran, V., Yang, H.: Managing requirements conflicts in software product lines: a goal and scenario based approach. *Data and Knowledge Engineering* 61(3), 417–432 (2007)
5. Davis, A.M.: Predictions and Farewells. *IEEE Software* 15(4), 6–9 (1998)
6. ECSS system: description and implementation, ECSS-S-00A Standard (2005)
7. ECSS space engineering: ground systems and operations – telemetry and telecommand packet utilization, ECSS-E-70-41A standard (2003)

8. Leveson, N.G.: The role of software in spacecraft accidents. *AIAA Journal of Spacecraft and Rockets* 41(4), 564–575 (2004)
9. Ambrosio, A.M., Martins, E., Vijaykumar, N.L., Carvalho, S.V.: Systematic generation of test and fault cases for space application validation. In: 9th ESA Data System in Aerospace (DASIA), Edinburgh, Scotland, May 30 - June 2, ESA Publications, Noordwijk (2005)
10. Halligan, R.J.: Requirements metrics: the basis of informed requirements engineering management. In: *Complex Systems Engineering Synthesis and Assessment Technology Workshop (CSESAT 1993)*, Calvados, MD, USA (1993)
11. Davis, A.M.: Just enough requirements management: where software development meets marketing. Dorset House Publishing Company (2005)
12. Knauss, E., Boustani, C., Flohr, T.: Investigating the impact of software requirements specification quality on project success. *Product-Focused Software Process Improvement* 32 Part 2, 28–42 (2009)
13. Hofmann, H.F., Lehner, F.: Requirements engineering as a success factor in software projects. *IEEE Software* (July/August 2001)
14. Boehm, B., In, H.: Identifying quality-requirement conflicts. *IEEE Software* 13(2), 25–35 (1996)
15. Wilson, W.M., Rosenberg, J.H., Hyatt, L.E.: Automated analysis of requirement specifications. In: 19th international conference on Software engineering, Boston, Massachusetts, United States, May 17–23, pp. 161–171 (1997)
16. Gilliam, D.P., Wolfe, T.L., Sherif, J.S., Bishop, M.: Software security checklist for the software life cycle. In: *Proceedings of the 12th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 243–248 (June 2003)
17. Sheldon, F.T., Kim, H.Y., Zhou, Z.: A case study: validation of guidance control software requirements for completeness, consistency and fault tolerance. In: *Eighth Pacific Rim International Symposium on Dependable Computing (PRDC 2001)*, Seoul, Korea, December 17–19, IEEE Computer Society, Los Alamitos (2001)
18. Zhu, X., Jin, Z.: Detecting of requirements inconsistency: an ontology-based approach. In: *Proceedings of the Fifth International Conference on Computer and Information Technology (CIT 2005)*, Shanghai, China, September 21–23 (2005)
19. Transaction Processing Performance Council, <http://www.tpc.org>
20. Spainhower, L., Kanoun, K. (eds.): *Dependability Benchmarking for Computer Systems*. Wiley-IEEE Computer Society Press, Hoboken (2008) ISBN: 9780470230558
21. Madeira, H., Some, R., Moreira, F., Costa, D., Rennels, D.: Experimental evaluation of a COTS system for space applications. In: *The International Conference on Dependable Systems and Networks*, Bethesda, Maryland, USA (2002)
22. Ruiz, J.C., Yuste, P., Gil, P., Lemus, L.: On benchmarking the dependability of automotive engine control applications. In: *IEEE/IFIP International Conference on Dependable Systems and Networks*, Florence, Italy (2004)
23. Koopman, P., Madeira, H.: Dependability benchmarking & prediction: a grand challenge technology problem. In: *1st IEEE Int. Workshop on Real-Time Mission-Critical Systems: Grand Challenge Problems*, Phoenix, Arizona, USA (November 30, 1999)
24. Martins, E., Sabião, S.B., Ambrosio, A.M.: ConData: a tool for automating specification-based test case generation for communication systems. *Software Quality Journal* 8(4), 303–319 (1999)
25. Vêras, P.C., et al.: Checklist of the software requirements documentation benchmark for space application, <http://eden.dei.uc.pt/~mvieira>

Verifying Mode Consistency for On-Board Satellite Software

Alexei Iliasov¹, Elena Troubitsyna², Linas Laibinis², Alexander Romanovsky¹,
Kimmo Varpaaniemi³, Pauli Väisänen³, Dubravka Ilic³, and Timo Latvala³

¹ Newcastle University, UK

² Åbo Akademi University, Finland

³ Space Systems Finland

{alexei.iliasov,alexander.romanovsky}@ncl.ac.uk,

{linas.laibinis,elena.troubitsyna}@abo.fi,

{Dubravka.Ilic,Timo.Latvala,Kimmo.Varpaaniemi,Pauli.Vaisanen}@ssf.fi

Abstract. Space satellites are examples of complex embedded systems. Dynamic behaviour of such systems is typically described in terms of operational modes that correspond to the different stages of a mission and states of the components. Components are susceptible to various faults that complicate the mode transition scheme. Yet the success of a mission depends on the correct implementation of mode changes. In this paper we propose a formal approach that ensures consistency of mode changes while developing a system architecture by refinement. The approach relies on recursive application of modelling and refinement patterns that enforce correctness while implementing the mode transition scheme. The proposed approach is exemplified by the development of an Attitude and Orbit Control System undertaken within the ICT DEPLOY project.

1 Introduction

Operational modes – mutually exclusive sets of the system behaviour [13] – form a useful structuring concept that facilitates the design of complex systems in different industrial sectors, including avionic, transportation and space. There are several well-known problems associated with mode-rich systems, e.g., correctness of complex mode transitions, mode consistency in distributed systems, mode confusion etc. However, there is still a lack of generic architectural-level approaches that would facilitate solving these difficult problems.

In this paper we propose a formal approach to developing complex mode-rich systems that allows us to ensure mode consistency. The approach generalizes the results of a large pilot development carried out by Space Systems Finland within the FP7 ICT project DEPLOY [11]. In collaboration with the academic partners the company has undertaken formal development and verification of (a part of) a satellite Attitude and Orbit Control System (AOCS) [6].

AOCS is a typical representative of mode-rich component-based control systems. It consists of several instruments that control the attitude and the orbit of a satellite as well as perform different scientific measurements. The distinctive

features of the system are long-running mode transitions and strong impact of component failures on the mode transition scheme.

In this paper we formalize the reasoning about mode consistency in layered mode-rich systems. As a result, we propose a generic pattern for specifying components of such systems. This pattern defines a generic module interface that can be instantiated by component-specific data and behaviour. We demonstrate that such systems can be developed by recursive instantiation of the proposed pattern. Our approach can be also seen as stepwise unfolding of architectural layers. The approach is illustrated by briefly describing the AOCs development.

2 Layered Mode-Rich Systems

It is recognized that a layered architecture is advantageous in designing complex component-based systems [16]. It provides the designers with a convenient mechanism for structuring system behaviour according to the identified abstraction levels. The lowest layer usually consists of the components that work directly with hardware devices. The layer above contains the components encapsulating the lowest components by providing abstract interfaces to them. Depending on system complexity and design decisions, there might be several intermediate layers. Finally, the top component provides an interface to the overall system.

In this paper we study the issues in designing and verifying layered mode-rich control systems. Leveson et al. [13] define *mode* as a mutually exclusive set of system behaviours. There is a large variety of mode-rich systems, including control systems that cyclically monitor the controlled environment. Here we focus on one particular aspect of the control system behaviour – mode management.

2.1 Mode Logic in Layered Architectures

In the core of mode management is the *mode logic* that consists of all the available modes and rules for transitioning between them [13]. The typical problem associated with mode-rich systems is to ensure mode consistency of the components residing at different layers, i.e., to correctly define the mode logic and guarantee that the system faithfully implements it. The (somewhat simplified) architecture of mode-rich layered systems is shown in Figure 1.

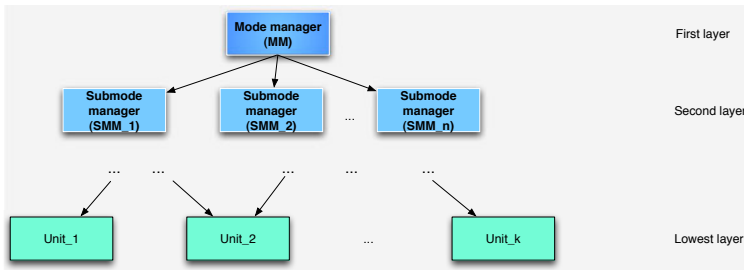


Fig. 1. Architecture of mode-rich layered systems

On the top layer is *Mode Manager (MM)* – the component implementing the mode logic on the system level. We assume that during its mission the system should execute a certain *scenario* defined in terms of its (global) modes. On the one hand, the *MM* mode logic is defined by this scenario. On the other hand, component failures can prevent the system from implementing the mode scenario and force it to redo certain steps. Hence, to fully define the mode logic, we should take into account both the component states and their possible errors.

The coarse-grained global modes allow us to represent the system-level mode logic as a process of instantaneous change from one mode to another. In reality, a mode transition may involve certain physical processes and hence have a duration. Indeed, to make a transition from the current to a new target mode, the system should bring all the involved components into the *consistent* states for entering the target mode. Therefore, while nominally being in one global mode, the system can be in two different states – stable and transitional.

When *MM* chooses a new target mode, it initiates (sequentially or in parallel) the corresponding mode transitions in lower layer components. As a result, the Submode Managers (*SMMs*) start to execute their own predefined scenarios to enable the global mode transition. Essentially, the behaviour of *SMMs* is similar to the behaviour of *MM*, i.e., while executing these scenarios they monitor the state of lower layer components to detect when the submode change is completed or an error has occurred. This allows us to define mode managers at different layers by instantiating the same generic specification pattern, as we show later.

If an error is detected, the responsible mode manager assesses the error and either initiates error recovery by itself or propagates the error to a higher layer. In mode-rich systems, error recovery is often implemented as a rollback to some preceding (and usually more degraded) mode in the predefined scenario.

The dynamic behaviour of the overall system is cyclic. At each cycle, *MM* assesses the *SMM* states and, based on these observations, it either

- initiates a forward transition according to the predefined scenario;
- initiates a backward transition (if error(s) has occurred). The actual target mode depends on severity of the occurred error(s);
- completes a transition to the target mode and becomes stable (if the conditions for entering the target mode are satisfied);
- maintains the current mode (if neither the conditions for entering the next global mode are satisfied nor an error has occurred).

While the system is recovering from one error, another error requiring a different mode transition might occur. Due to a large number of components and their failure modes, ensuring mode consistency becomes especially difficult. Next we define the mode consistency criteria for layered control systems more formally.

2.2 Formal Reasoning about Modes and Mode Transitions

Essentially, a mode can be understood as an abstraction of the system state, i.e., the class of states associated with a certain system functionality. The mode logic is usually defined as a set of all the modes and mode transition rules [13]. Thus

we can consider it as a special kind of a state transition system or, formally, as a triple $(Modes, Next, InitMode)$, where $Modes$ is a set of all possible modes of the system, $Next$ is a relation on $Modes$, containing all allowed mode transitions, and $InitMode$ is the initial mode. Since $Next$ is a relation, it can contain several predefined scenarios that can be executed by a mode manager.

Sometimes $Next$ can be defined more precisely as an *ordering relation*. Indeed, some predefined scenarios define how to take a system from a non-operational mode (e.g., *Off*) to a fully operational one. The predefined scenario of the AOCS system presented in Section 5 is a typical example of this. This scenario describes the sequence of modes from powering-on the instruments to bringing them into the mode that enables collection of valuable scientific data.

While $Next$ is based on the predefined scenario(s), the mode transitions executed as error recovery are governed by the function *Mode_error_handling*:

$$Mode\_error\_handling : MState \times (LocalErrors_1 \times \dots \times LocalErrors_k) \rightarrow Modes$$

where $MState$ is the component state and $LocalErrors_1 \dots LocalErrors_k$ are all the errors detected by lower layer components in the previous cycle. The function defines the mode to which the system should rollback to execute error recovery. The current and new modes should belong to the transitive closure of $Next$.

However, $Next$ represents only a possibility of mode transitions. For a mode transition to be completed, certain mode entry conditions should be satisfied. We can formally define this by introducing a function *Mode_ent_cond* of the type:

$$Mode\_ent\_cond : Modes \rightarrow \mathcal{P}(MState \times LocalModes_1 \times \dots \times LocalModes_k) \quad (1)$$

where $LocalModes_1, \dots, LocalModes_k$ are modes of the monitored components. For each (global) mode, the function returns a set of the allowed combinations of the component state and the monitored local modes. Here we assume that the local modes belong to the externally visible state of those components.

The mode entry conditions can be recursively constructed throughout the entire architecture for each pair of a mode manager component and a mode. We also use *Mode_ent_cond* to determine which components are affected when a mode manager initiates a new mode transition, i.e., to which components it should send the corresponding (local mode) transition requests.

To guarantee that the mode logic is unambiguous, we have to ensure that a component can be only in one mode at a time, i.e., the mode entry conditions for different modes cannot overlap:

$$\begin{aligned} \forall i, j \bullet M_i \in Modes \wedge M_j \in Modes \wedge i \neq j \Rightarrow \\ Mode\_ent\_cond(M_i) \cap Mode\_ent\_cond(M_j) = \emptyset \end{aligned} \quad (2)$$

Overall, the definition (1) and the property (2) define *mode consistency* conditions that should be guaranteed for each mode manager of a system.

Let us now address another important issue in designing mode-rich systems – ensuring *mode invariants*. These are system properties that are required to be preserved in each particular mode. However, in the systems where mode transitions take time and can be interrupted by errors, this is not a straightforward task. To tackle it, let us define the following attributes of a mode manager:

- *last_mode* – signifies the last successfully reached mode;
- *next_target* – signifies the target mode that a component is currently in transition to;
- *previous_target* – signifies the previous mode that a component was in transition to (though it has not necessarily reached it).

Collectively, these three attributes unambiguously describe the actual mode of a mode manager. Based on them, we define the notion of component *status* that might be either *Stable*, *Decreasing* or *Increasing* as follows:

- *Stable* $\triangleq last\_mode = previous\_target \wedge next\_target = previous\_target$
a component is maintaining the last successfully reached mode
- *Increasing* $\triangleq last\_mode = previous\_target \wedge previous\_target < next\_target$
a component is in transition to a next, more advanced mode;
- *Decreasing* $\triangleq next\_target < previous\_target$
component stability or a mode transition to *previous_target* was interrupted (e.g., by error handling) by a new mode request to a more degraded mode.

A graphical diagram showing mode status changes is given in Figure 2.

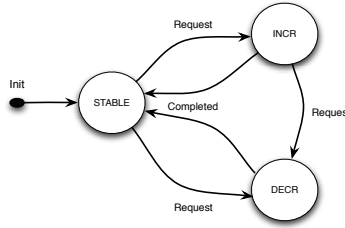


Fig. 2. Component mode status

We assume that, when a mode transition is completed, the component status is changed to *Stable*. The mode manager *MM* will maintain this status only if the final mode(s) of the scenario (defined by *Next*) is reached. On the lower layers, mode managers (*SMMs*) will maintain their stability until receiving a request for a new mode transition. In its stable state, *MM* would change its status to *Increasing* to execute the next step of the mode scenario, which in turn would trigger the corresponding mode transitions of the lower layer components. Irrespectively of the component status, an occurrence of an error would result in changing it to *Decreasing* that designates a rollback in the predefined scenario.

Now we can formally connect the mode status and a mode invariant. When a mode manager is stable, the mode entry condition is a mode invariant, i.e.,

$$\forall i \bullet m_i \in Modes \wedge last\_mode = M_i \wedge Stable \Rightarrow (s, l_1, \dots, l_k) \in Mode\_ent\_cond(M_i)$$

where $s : MState$ is the current state, and $l_1, \dots, l_k$ are the visible local modes. The other mode invariants are also preserved when a component is stable:

$$\forall i \bullet m_i \in Modes \wedge last\_mode = M_i \wedge Stable \Rightarrow Mode\_Inv(M_i)$$

Hence, in general, mode invariant properties are not preserved while a mode manager is engaged in a mode transition.

The discussion above sets the general guidelines for defining mode managers in layered mode-rich systems. While specifying a particular mode manager, we instantiate the abstract data structures *Modes*, *Next*, *Mode_ent_cond*, and *Mode_error_handling* and ensure that

- R1.** In a stable state, the mode manager makes its decision to initiate a new mode transition to some more advanced mode according to the relation *Next*;
- R2.** In a transitional state, the mode manager monitors the state of lower layer components. When *Mode_ent_cond(next_target)* becomes satisfied for the local state and the submodes of monitored components, the mode manager completes the mode transition and becomes stable;
- R3.** In both stable and transitional states, the mode manager monitors the lower layer components for the detected errors. If such errors occurred in the last cycle, the mode manager makes its decisions based on *Mode_error_handling*, which is applied to the mode manager state and all the detected errors.

In Section 4 we will show how these guidelines can be implemented in the proposed formal specification and development patterns.

3 Event B

Our chosen formal specification framework – Event B – allows us to model and verify state transition systems. Since it relies on theorem proving rather than model checking, it scales well to reasoning about systems with large state space.

3.1 Modelling and Refinement in Event B

The Event B framework [2] is an extension of the B Method [1]. The framework enables modelling of event-based (reactive) systems by incorporating the ideas of the Action Systems formalism [3]. Event B is actively used within FP7 ICT project DEPLOY to develop dependable systems from various domains.

The Event B development starts from creating a formal system specification. A simple Event B specification has the following general form:

```

MACHINE AM
SEES Context
VARIABLES v
INVARIANT Inv
EVENTS
  INITIALISATION = ...
   $E_1$  = ...
  ...
   $E_N$  = ...
END

```

Such a specification encapsulates a local state (model variables) and provides operations on the state. The operations (called *events*) can be defined as

```
ANY vl WHERE g THEN S END
```

where vl is a list of new local variables, the guard g is a state predicate, and the action S is an assignment on model variables. In case when vl is empty, the event syntax becomes **WHEN** g **THEN** S **END**. Both ordinary and non-deterministic assignments can be used to specify state change. The non-deterministic assignments are of the form $v :| Post(v, v')$, where $Post$ is the postcondition relating the variable values before and after the assignment.

The events describe system reactions when the given **WHEN** or **WHERE** conditions are satisfied. The **INVARIANT** clause contains the properties of the system (state predicates) that should be preserved during system execution. The model data types and constants are defined in a separate component called **Context**.

To check consistency of an Event B machine, we should verify two types of properties: event feasibility and invariant preservation. Formally,

$$\begin{aligned} Inv(v) \wedge g_e(v) &\Rightarrow \exists v'. Post_e(v, v') \\ Inv(v) \wedge g_e(v) \wedge Post_e(v, v') &\Rightarrow Inv(v') \end{aligned}$$

The main development methodology of Event B is *refinement* – the process of transforming an abstract specification to gradually introduce implementation details while preserving its correctness. Refinement allows us to reduce non-determinism present in an abstract model as well as introduce new concrete variables and events. The connection between the newly introduced variables and the abstract variables that they replace is formally defined in the invariant of the refined model. For a refinement step to be valid, every possible execution of the refined machine must correspond to some execution of the abstract machine.

The consistency of Event B models as well as correctness of refinement steps should be formally demonstrated by discharging *proof obligations*. The Rodin platform[19], a tool supporting Event B, automatically generates the required proof obligations and attempts to automatically prove them. Sometimes it requires user assistance by invoking its interactive prover. However, in general the tool achieves high level of automation (usually over 90%) in proving.

3.2 Modelling Modular Systems in Event B

Recently the Event B language and tool support have been extended with a possibility to define modules [10,15] – components containing groups of callable operations. Modules can have their own (external and internal) state and the invariant properties. The important characteristic of modules is that they can be developed separately and, when needed, composed with the main system.

A module description consists of two parts – *module interface* and *module body*. Let M be a module. A module interface MI is a separate Event B component. It allows the user of module M to invoke its operations and observe the external variables of M without having to inspect the module implementation details. MI consists of external module variables w , constants c , and sets s , the external module invariant $M\_Inv(c, s, w)$, and a collection of module operations, characterised by their pre- and postconditions, as shown below.


```

INTERFACE M1 =
  SEES M1.Context
  VARIABLES w
  INVARIANT M.Inv(c, s, w)
  OPERATIONS
    op1(par, res) =
      PRE M.Guard1(c, s, par, w)
      POST M.Post1(c, s, par, w, w', res')
... END

```

A module development always starts with the design of an interface. Once an interface is defined, it cannot be altered in any manner. This ensures correct relationships between a module interface and its body. A module body is an Event B machine, which implements each interface operation by a separate group of Event B events. Additional proof obligations guarantee that each event group faithfully implement the corresponding pre- and postconditions.

When the module M is "included" into another Event B machine, the including Event B machine can invoke the operations of M as well as read all the external variables of M . Several instances of the same module operating on disjoint state spaces can be created. Moreover, module abstract types and constants, defined in the interface context, can be instantiated with concrete data.

The modularisation extension of Event B was motivated by the pilot deployment of a satellite system within the DEPLOY project [11]. The extension was needed not only to enable architectural level reasoning but also to significantly improve scalability of Event B. As we show next, the modularisation extension also facilitates modelling and verification of layered mode-rich systems.

4 Development Pattern

In this section we propose a formal development pattern for layered mode-rich systems in the Event B framework. This pattern is based on formal reasoning about modes and mode transitions presented in Section 2.

4.1 Generic Interface

As discussed earlier, the structure and behaviour of mode managers at different layers are very similar. This suggests the idea of modelling such a component as a generic module that can be adapted to different contexts by instantiating its generic parameters. In Event B, we can formalise this by first creating a generic module interface that can be later implemented in different ways, thus creating implementations of specific mode managers. The proposed interface contains four operations that can be called from a higher layer. It also defines the external module variables that are visible from a calling component. An informal description of the interface pattern is given in Figure 3.

```

INTERFACE Mode_Manager
SEES Mode_Manager.Context (* introduces abstract Modes, Errors, and Next *)
VARIABLES last_mode, next_target, (* list of external variables of a module *)
             previous_target, error

INVARIANTS
  types of external variables
  other invariant properties

OPERATIONS

  SetTargetMode =
    ANY m
    PRE
      Component has not failed
      m is a new target mode
    POST
      new target mode is set

  ResetError =
    PRE
      the error flag is raised
    POST
      the error flag is cleared

  RunStable =
    PRE
      Component is stable and not failed
    POST
      Component either remains stable
      or changes its mode according to the scenario
      or raises the error flag

  RunNotStable =
    PRE
      Component is in a mode transition
    POST
      A mode transition is completed
      or a mode transition continues
      or the error flag is raised

```

Fig. 3. Interface of a generic mode manager

The external state of a component is formed by four variables – *last_mode*, *next_target*, *previous_target* and *error*. The first three variables define the component mode status, while the last one models the currently detected errors. Moreover, the interface context introduces the abstract sets *Modes* and *Errors*, and the abstract functions *Next* and *Mode_error_handling*. These structures should be instantiated with concrete data when a module instance is created. If *Next* is a partial order, its required properties are also checked during instantiation.

The operation *SetTargetMode* is called to set a new target mode, while *ResetError* is called after the detected errors are handled by an upper layer component (e.g., by initiating the appropriate error recovery).

Since the behaviour of the overall system is cyclic, we assume that within the cycle the control is passed from layer to layer to each component. The operations *RunStable* and *RunNotStable* model component behaviour when it receives the control while being correspondingly in a stable or a transitional state. The actual state is unambiguously determined by the external mode status variables.

Let us now show that our interface pattern follows the guidelines of Section 2. The requirement **(R1)** stipulates the use of the predefined scenario *Next* in a stable state. In the presented interface, this requirement is incorporated into the postcondition of *RunStable*. The requirement **(R2)** prescribes the use of *Mode_ent_cond* to monitor whether the current mode transition has successfully completed. The requirement is a part of the postcondition of *RunNotStable*.

Finally, the requirement **(R3)** calls for *Mode_error_handling* to be used when, upon detection of error(s), a new mode request has to be sent to lower layers. This requirement is defined in terms of the component state and the modes of lower layer components, i.e., in terms of two adjacent layers. This requirement can only be demonstrated during refinement, i.e., when lower layer components

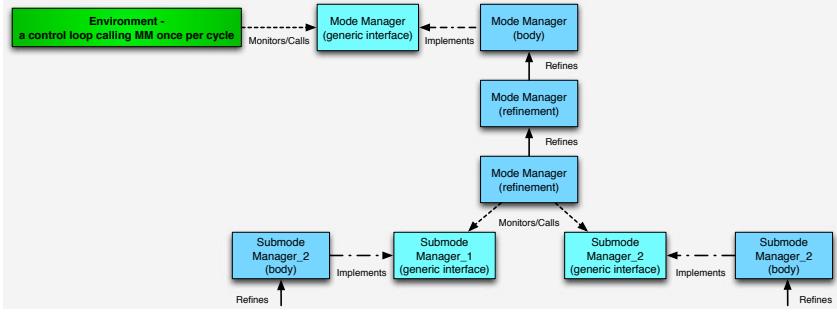


Fig. 4. Development hierarchy

are introduced. Nevertheless, even in the generic pattern we require that implementations of *RunStable* and *RunNotStable* have to use this function to assess the errors flagged by the lower layer components.

All the operations update the variables *last_mode*, *next_target*, *previous_target* to reflect the changing mode status. Due to a lack of space, we postpone presenting all formal details of the interface specification pattern until Section 5, where we discuss modelling of AOCs.

4.2 Refinement Strategy

In general, refinement process aims at introducing implementation details into an abstract system specification. However, in this paper we demonstrate that refinement can also be used to incrementally build the system architecture. This is especially well-suited for layered control systems, where refinement can be used to gradually unfold system layers by using the predefined specification and refinement patterns [12]. Indeed, the generic interface *MMC_I* that we described above can be seen as an abstract representation of the top level interface of a mode-rich system. Yet it can also be seen as an interface of any mode manager at a lower layer. Therefore, by instantiating *MMC_I* with the mode logic specific for a particular mode manager, we can obtain a mode manager of any layer. Hence our development strategy can be seen as a process of introducing specific module types into an Event-B development, as shown in Figure 4.

We assume that the system executes cyclically, with the environment periodically invoking the top mode manager. In its turn, it calls lower layer mode managers. This behaviour is recursively repeated throughout the hierarchy.

The refinement process starts by instantiating the top level mode manager interface with the global mode logic. The body of the obtained mode manager can be further developed by refinement. This is similar to building a normal refinement chain although the starting point is an interface rather than an abstract machine. At some point of our development, a number of lower layer mode managing components that the mode manager controls are introduced. This refinement step essentially introduces calls to the corresponding interface operations of these submode managers. At the same time, the submodes and

errors of the lower layer become visible for the mode manager. Hence we can define the mode consistency conditions as additional invariants that are verified in this refinement step. In a similar way we handle errors of new components.

On the architectural level, such a refinement step corresponds to unfolding one more layer of the system hierarchy. From this point, we can focus on refining bodies of the introduced submode managers. These bodies would implement their own mode logics and also, if needed, call operations of the mode managers residing on the layer below. Hence we follow the same refinement pattern as before, unfolding the architectural layers until the entire hierarchy is built.

The main strength of our development is that we ensure global mode consistency by simply conjuncting the mode linking conditions introduced at each level. Hence, despite a strict hierarchical structure, there is a simple procedure for enforcing conformance of mode changes for any two or more components of a system. We avoid reasoning about the entire global mode consistency and instead enforce by refinement mode consistency between any two adjacent layers.

Our approach allows us to design a layered mode-rich system in a disciplined structured way. It makes a smooth transition from architectural modelling to component implementation, yet ensuring the overall mode consistency. This approach generalizes our experience in developing AOCS [6], presented next.

5 Case Study – Attitude and Orbit Control System

The Attitude and Orbit Control System (AOCS) is a generic component of satellite onboard software, the main function of which is to control the attitude and the orbit of a satellite. Due to a tendency of a satellite to change its orientation because of disturbance from the environment, the attitude needs to be continuously monitored and adjusted. An array of sensors provide the information required to compute corrective commands and issue them to the actuators. An optimal attitude is needed to support the needs of payload instruments.

The AOCS architecture is an instance of a layered architecture shown in Figure 1. On the highest layer is *Mode Manager (MM)*. It controls *Unit Manager (UM)*, which, in its turn, is responsible for a number of hardware units. The AOCS system has seven units – four sensors (Star tracker, Sun Sensor, Earth Sensor and Global Positioning system), two actuators (Reaction Wheel and Thruster), and one payload instrument producing mission measurements. *UM* provides a generic interface to units. It hides from *MM* the number and types of units, while monitoring their states, modes and error flags. *MM* is responsible for implementing the AOCS mode logic. The predefined mode scenario defines the sequence of steps needed to reach the state where the payload instrument is ready to perform its tasks. This sequence includes the following modes:

- *Off* – The satellite is typically in this mode right after system (re)booting;
- *Standby* – This mode is maintained until separation from the launcher;
- *Safe* – A stable attitude is acquired, which allows the coarse pointing control;
- *Nominal* – The satellite is trying to reach the fine pointing control which is needed to use the payload instrument;

- *Preparation* – The payload instrument is getting ready;
- *Science* – The payload instrument is ready to perform its tasks. The mission goal is to reach this mode and stay in it as long as needed.

Mode Manager. While modelling AOCs, we assume that there is a cyclic scheduler that invokes *MM* at each execution cycle. Our generic specification template for defining the interface of a mode manager is shown in Figure 5.

```

INTERFACE Mode_Manager
...
INVARIANT
...
   $next\_target = previous\_target \implies next\_target = last\_mode$ 
   $next\_target \neq previous\_target \implies$ 
     $next\_target \mapsto previous\_target \in Next \wedge previous\_target \mapsto next\_target \in Next$ 
   $last\_mode \mapsto previous\_target \in Next \cup Next^{-1}$ 
   $last\_mode \mapsto next\_target \in Next \cup Next^{-1}$ 
OPERATIONS

  SetTargetMode(r) = ANY m
    PRE
       $error = NoError \wedge m \in MODES \wedge m \neq next\_target \wedge m \mapsto next\_target \in Next \cup Next^{-1}$ 
    POST
       $r' = last\_mode \wedge previous\_target' = next\_target \wedge next\_target' = m$ 

  ResetError(r) = PRE  $error \neq NoError$  POST  $r' = last\_mode \wedge error' = NoError$ 

  RunStable(r) = PRE  $next\_target \neq previous\_target \wedge error = NoError$ 
    POST
       $r' = last\_mode \wedge error' \in ERROR \wedge previous\_target \mapsto next\_target' \in Next \cup Next^{-1}$ 

  RunNotStable(r) = PRE  $next\_target = previous\_target \wedge error = NoError$ 
    POST
       $r' = last\_mode \wedge error' \in ERROR \wedge$ 
       $(last\_mode' \mapsto next\_target \in Next \cup Next^{-1} \wedge next\_target' = next\_target \wedge$ 
       $previous\_target' = previous\_target) \vee (next\_target' = next\_target \wedge$ 
       $previous\_target' = next\_target \wedge last\_mode' = previous\_target')$ 

```

Fig. 5. Specification of the Mode Manager Interface (an excerpt)

The first refinement step of *MM* is an abstract implementation of the interface operations. At this stage, the operations *ResetError* and *SetTargetMode* are each refined by single events. *RunStable* and *RunNotStable* have more complex postconditions and thus have to be represented by several events. The operation *RunStable* is realised by three events: an event for successful cycle (*run_success*), an event for mode advance (*run_mode_scenario*), and an event modelling error handling (*run_failure*).

<pre> run_success = WHERE next_target = previous_target error = NoError THEN WorkCycle.r := last_mode run_failure = WHERE next_target = previous_target error = NoError THEN WorkCycle.r := last_mode error := error' ≠ NoError </pre>	<pre> run_mode_scenario = ANY <i>m</i> WHERE next_target = previous_target error = NoError ∧ <i>m</i> ∈ MODES previous_target ↦ <i>m</i> ∈ Next ∪ Next⁻¹ THEN WorkCycle.r := last_mode next_target := <i>m</i> </pre>

Likewise, *RunNotStable* is implemented by the events modelling situations when no mode advance happens, when the mode advances but the target mode has not been reached and, finally, when the target has been successfully reached.

In addition to some preparatory steps towards integration with *UM*, the next refinement step also specifies the predefined mode scenario enforced by *MM*. The scenario is defined as a constant relation on modes such that the *Next* relation a transitive closure of it. The constant function *Scenario* defines a linear progression of modes from *Off* to *Science*. The axioms connecting *Scenario* with the relation *Next* are necessary to demonstrate correctness of the refinement step. The event *run_mode_scenario* uses *Scenario* to select the next mode.

$$\begin{aligned} \text{Scenario} &= \{ \text{OFF} \mapsto \text{STANDBY}, \text{STANDBY} \mapsto \text{SAFE}, \text{SAFE} \mapsto \text{NOMINAL}, \\ &\quad \text{NOMINAL} \mapsto \text{PREPARATION}, \text{PREPARATION} \mapsto \text{SCIENCE} \} \\ \text{Scenario} &\subseteq \text{Next} \wedge \text{Next}; \text{Scenario} \subseteq \text{Next} \wedge (\forall z \cdot \text{Scenario} \subseteq z \wedge z; \text{Scenario} \subseteq z \Rightarrow \text{Next} \subseteq z) \end{aligned}$$

Integration with the Unit Manager is the most complex refinement step in our development. Since we want to build a model adaptable to various hardware configurations, the *UM* interface comes in a parameterised form: some of its sets and constants may be instantiated at the point of composition. Instantiation examples for the correspondence relation between *MM* and *UM* modes (*um_mode*) and the *UM* mode scenario (*um_Scenario*) are given below:

$$\begin{aligned} \text{um\_mode} &= \{ \text{OFF} \mapsto \text{um\_InitMode}, \text{STANDBY} \mapsto \text{um\_InitMode}, \text{SAFE} \mapsto \text{um\_NAV\_EARTH}, \\ &\quad \text{SAFE} \mapsto \text{um\_NAV\_SUN}, \text{NOMINAL} \mapsto \text{um\_NAV\_ADV}, \text{PREPARATION} \mapsto \text{um\_NAV\_FINE}, \\ &\quad \text{SCIENCE} \mapsto \text{um\_NAV\_INSTR} \} \\ \text{um\_Scenario} &= \{ \text{um\_InitMode} \mapsto \text{um\_NAV\_EARTH}, \text{um\_InitMode} \mapsto \text{um\_NAV\_SUN}, \text{um\_NAV\_EARTH} \mapsto \\ &\quad \text{um\_NAV\_ADV}, \text{um\_NAV\_SUN} \mapsto \text{um\_NAV\_ADV}, \text{um\_NAV\_FINE} \mapsto \text{um\_NAV\_INSTR} \} \end{aligned}$$

Here and further, *um_* is a module instantiation prefix, i.e., all the constants, sets and variable starting with *um_* are a part of this particular module.

The *UM* linking invariant presented below specifies that the modes of *UM* and *MM* are in the correspondence relation defined during the *UM* instantiation. The last condition also states that there may be periods when the *UM* error flag is set but *MM* has not yet decided about its recovery actions. This allows the Mode Manager to mask errors by recovering locally rather than propagating an error. The module instantiation data and the correspondence relation together define the mode consistency conditions.

$$\begin{aligned} \text{next\_target} = \text{previous\_target} &\Rightarrow \text{last\_mode} \mapsto \text{um\_last\_mode} \in \text{um\_mode} \\ \text{next\_target} = \text{previous\_target} &\Rightarrow \text{next\_target} \mapsto \text{um\_next\_target} \in \text{um\_mode} \\ \text{next\_target} = \text{previous\_target} &\Rightarrow \text{previous\_target} \mapsto \text{um\_previous\_target} \in \text{um\_mode} \\ \text{async} = \text{FALSE} \wedge \text{um\_error} \neq \text{um\_NoError} &\Rightarrow \text{error} \neq \text{NoError} \end{aligned}$$

Further refinement steps of the *MM* module introduce the control loop that queries the sensors and sends commands to the actuators.

Unit Manager. *UM* is a generic module that can be configured during instantiation to any required hardware set-up. Having a parameterised interface allows a modeller to prove stronger properties by providing additional information during composition. In the case of *UM*, the parameters are *Modes* and *Next*.

Since we are applying the same development template once again, the general development strategy is similar to the one of *MM*, e.g., the initial refinement

steps are done to prepare for integration with unit modules. The instantiation and linking invariants of the units modules have a similar structure as well.

In our AOCs development generic specification and refinement patterns (proposed in Section 4) were applied several times to construct each major part of the system. The modelling was carried out in the Rodin Platform [19], extended by the modularisation plug-in [15]. The respective proof obligations were discharged using a collection of the provided automated theorem provers with a small number of interactive proofs. Full Event B models can be found in [9].

6 Related Work

Formal validation of the mode logic and, in particular, fault tolerance mechanisms of satellite software has been undertaken by Rugina et al [17]. They have investigated different combinations of simulation and model checking. In general, simulation does not allow the designers to check all execution paths, while model checking often runs into the state explosion problem. To cope with these problems, the authors had to experiment with combination of these techniques as well as heavily rely on abstractions. Our approach is free from these problems. First, it allows the developers to systematically design the system and formally check mode consistency within the same framework. Second, it enables exhaustive check of the system behaviour, yet avoiding the state explosion problem.

The mode-rich systems have been studied to investigate the problem of mode confusion and automation surprises [4,18]. These studies conducted retrospective analysis of mode-rich systems to spot the discrepancies between the actual system mode logic and the user mental picture of the mode logic. Most of the approaches relied on model-checking [4,8,18], while [5] relied on theorem proving in PVS. Our approach focuses on designing fully automatic systems and ensuring their mode consistency. Unlike [8], in our approach we also emphasize the complex relationships between system fault tolerance and the mode logic.

In our previous work [7], we have studied a problem of specifying mode-rich systems from the contract-based rely-guarantee perspective. These ideas have been further applied for fault tolerance modes [14]. According to this approach, a mode-centric specification of the system neither defines how the system operates in some specific mode nor how mode transitions occur. It rather imposes restrictions on concrete implementations. In this paper we have demonstrated how to combine reasoning about the system mode logic and its functioning.

7 Conclusions

In this paper we have proposed a formal approach to development of mode-rich layered systems. It is based on instantiation and refinement of a generic specification pattern for a mode manager. The pattern defined as a generic module interface captures the essential structure and behaviour of a component and can be instantiated by component specific data to model a mode manager at any layer of the system hierarchy. The overall process can be seen as a stepwise

unfolding of architectural layers. Each such unfolding is accompanied by proving its correctness, while also verifying mode consistency between two adjacent layers. Such an incremental verification allows us to guarantee the global mode consistency, yet avoid checking the property for the whole architecture at once.

The generic specification pattern relies on our formalisation of reasoning about systems with non-instantaneous mode transitions, the mode logic of which is also integrated with error recovery. The formalisation of what constitutes mode consistency and mode invariance properties together with establishing precise relationships between error recovery and the mode logic allowed us to derive design guidelines and logical constraints for components of mode-rich systems.

Our approach has been used in the development of AOCS. The approach has demonstrated good potential for facilitating design of complex mode-rich systems. Moreover, its support for formulating and verifying mode invariants has been especially appreciated in the industrial settings, since these invariants can be directly translated into assertions on the code level.

We are planning to further develop the proposed approach to enable reasoning about mode consistency in the presence of dynamic reconfiguration. Moreover, it would be also interesting to investigate how the mode ordering properties (when applicable) are inter-related with mode consistency.

Acknowledgments

This work is supported by the FP7 ICT DEPLOY Project and the EPSRC/UK TrAmS platform grant.

References

1. Abrial, J.-R.: *The B-Book*. Cambridge University Press, Cambridge (1996)
2. Abrial, J.-R.: *Modelling in Event-B*. Cambridge University Press, Cambridge (2010)
3. Back, R., Sere, K.: Superposition refinement of reactive systems. *Formal Aspects of Computing* 8(3), 1–23 (1996)
4. Buth, B.: Analysing mode confusion: An approach using fdr2. In: Heisel, M., Liggesmeyer, P., Wittmann, S. (eds.) *SAFECOMP 2004*. LNCS, vol. 3219, pp. 101–114. Springer, Heidelberg (2004)
5. Butler, R.W.: An introduction to requirements capture using PVS: Specification of a simple autopilot. Technical report, NASA TM-110255 (May 1996)
6. DEPLOY Deliverable D20 – Report on Pilot Deployment in the Space Sector. FP7 ICT DEPLOY Project (January 2010), <http://www.deploy-project.eu/>
7. Dotti, F., Iliasov, A., Ribeiro, L., Romanovsky, A.: Modal Systems: Specification, Refinement and Realisation. In: Breitman, K., Cavalcanti, A. (eds.) *ICFEM 2009*. LNCS, vol. 5885, Springer, Heidelberg (2009)
8. Heimdahl, M., Leveson, N.: Completeness and Consistency in Hierarchical State-Based Requirements. *IEEE Transactions on Software Engineering* 22(6), 363–377 (1996)
9. Iliasov, A., Laibinis, L., Troubitsyna, E.: An Event-B model of the Attitude and Orbit Control System, <http://deploy-eprints.ecs.soton.ac.uk/>

10. Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Ilic, D., Latvala, T.: Supporting Reuse in Event B Development: Modularisation Approach. In: Frappier, M., Glässer, U., Khurshid, S., Laleau, R., Reeves, S. (eds.) *Abstract State Machines, Alloy, B and Z*. LNCS, vol. 5977, pp. 174–188. Springer, Heidelberg (2010)
11. Industrial deployment of system engineering methods providing high dependability and productivity (DEPLOY). IST FP7 project, <http://www.deploy-project.eu/>
12. Laibinis, L., Troubitsyna, E.: Fault tolerance in a layered architecture: a general specification pattern in B. In: *Proc. of the 2nd Int. Conference on Software Engineering and Formal Methods (SEFM)*, Beijing, pp. 346–355. IEEE Press, Los Alamitos (2004)
13. Leveson, N., Pinnel, L.D., Sandys, S.D., Koga, S., Reese, J.D.: Analyzing Software Specifications for Mode Confusion Potential. In: *Proc. of Workshop on Human Error and System Development*, Glasgow, Scotland, pp. 132–146 (1997)
14. Lopatkin, I., Iliasov, A., Romanovsky, A.: On fault tolerance reuse during refinement. In: *Proc. of 2nd International Workshop on Software Engineering for Resilient Systems* (April 2010)
15. RODIN modularisation plug-in. Documentation at, http://wiki.event-b.org/index.php/Modularisation_Plug-in
16. Rubel, B.: Patterns for Generating a Layered Architecture. In: Coplien, J.O., Schmidt, D.C. (eds.) *Pattern Languages of Program Design*, Addison-Wesley, Reading (1995)
17. Rugina, A.E., Blanquart, J.P., Soumagne, R.: Validating failure detection isolation and recovery strategies using timed automata. In: *Proc. of 12th European Workshop on Dependable Computing, EWDC 2009*, Toulouse (2009)
18. Rushby, J.: Using model checking to help discover mode confusion and other automation surprises. In: *Reliability Engineering and System Safety*, vol. 75, pp. 167–177 (2002)
19. The RODIN platform, <http://rodin-b-sharp.sourceforge.net/>

Computational Concerns in the Integration of Unmanned Airborne Systems into Controlled Airspace

Christopher W. Johnson

Department of Computing Science, University of Glasgow, Scotland
johnson@dcs.gla.ac.uk

Abstract. Unmanned Airborne Systems (UAS) offer significant benefits for long duration missions. They can also be used in situations where it is inappropriate to expose aircrew to increased levels of risk. Partly in consequence, they continue to experience accident rates that are significantly higher than those for most conventional aircraft. It can also be argued that increased mishap rates are due to lower standards of design and maintenance. UAS are, therefore, largely confined to segregated areas that are well away from other airspace users. There are growing commercial and political pressures for them to be integrated into airspace that is directly under the control of air traffic management. Police agencies would like to deploy miniature UAS in populated areas, for example, to augment conventional helicopter operations. There are proposals to operate unmanned freight operations from existing airports. Longer-term proposals include the use of UAS technology to replace the co-pilot. Automated systems might intervene only if the single human pilot is incapacitated. The following pages focus on the computational issues that form one part of wider set of ‘system safety’ concerns that must be addressed before UAS operations can be integrated into controlled airspace.

Keywords: UAV, UAS, Air Traffic Management, Safety, Controlled Airspace.

1 Introduction

The next 20 years will see enormous changes in air traffic. One aspect of this is the probable introduction of Unmanned Airborne Systems (UAS) into controlled air space. Traditionally, these aircraft have been segregated away from other airspace users in areas that are not directly under the control of Air Navigation Service Providers (ANSPs). However, a range of commercial and political pressures are acting to challenge existing safeguards. Within the US military funding for Unmanned Airborne Vehicles (UAVs) development has increased from \$3 billion in the early 1990s to over \$12 billion for 2004-2009 [1]. The civil UAS market could reach €120 million annually by 2011.

UAVs offer numerous benefits. They support long duration missions that would be difficult, if not impossible, to resource using conventional aircraft. Examples include the monitoring work being undertaken by the Customs and Border Patrol on the US-Mexico border. They also include more speculative proposals to incorporate UAVs into the security systems for the 2012 London Olympics. A new generation of

unmanned freight vehicles is being deployed to help the US military avoid the Improvised Explosive Devices that are being deployed against coalition forces in Iraq and Afghanistan. Civil versions are (arguably) less than a decade away from production [2].

UAS also raise a number of concerns. They have a significantly higher accident rate than conventional aircraft [3]. Some of these mishaps stem from the human factors issues associated with maintaining remote situation awareness. It is also apparent from previous incidents that the standards that are used in the engineering of UAV platforms often fall below those required in conventional aircraft. Documents such as the Federal Aviation Administration's (FAA) 08-01: Unmanned Aircraft Systems Operations in the U.S. National Airspace System (NAS) [4], as well as EUROCONTROL's Spec-0102 on the Use of Military Unmanned Aerial Vehicles as Operational Air Traffic outside Segregated Airspace [5] and the UK Civil Aviation Authority's CAP 722 [6], therefore, place strict limits on the operation of UAS.

The commercial opportunities created by UAS make it difficult to defend these restrictions in the long run. ANSPs and regulators face growing political pressure to allow the integration of UAS operations with other airspace users [2]. CAP722, cited above, recognizes this; "While UAV flights beyond the limits of visual control ... are currently restricted to segregated airspace, the ultimate aim is to develop a regulatory framework which will enable the full integration of UAS activities with manned operations throughout UK airspace." In order for this to happen, UAV operations must not increase the risk to other airspace users. The CAA also argue that Air Traffic Management (ATM) procedures for interactive with UAS should mirror those applicable to manned aircraft and that the provision of air traffic services to UAVs should be transparent to controllers [1, 7].

The following pages focus on early-stage planning for the integration of UAS into controlled airspace. The computational hazards of UAV operations form part of a range of 'systemic' safety concerns. These range from the difficulties that Air Traffic Control Officers (ATCOs) experience when trying to identify who is commanding UAS operations over distributed communications systems to the problems associated with semi-autonomous operations, for example during the execution of lost link profiles, described below.

2 Airworthiness Concerns

UAV platforms must, typically, meet the same airworthiness requirements as conventional aircraft. For instance, Section 6 of FAA Interim Guidance 08-01 stipulates that all UAS must be in a fit state to conduct operations in the NAS. In particular, there is a requirement that the components of the system be maintained and conform to "the same airworthiness standards as defined for the 14 Code of Federal Regulations parts under which UAS are intended to be operated". In Europe, UAVs over 150kg must have an airworthiness certificate provided by EASA. Platforms below that mass can operate under national airworthiness certification. In both cases, the approach tends to follow that outlined within FAA 08-01. UK CAA CAP722 guidance states that "If

the UAS industry is to produce UAS capable of operating in all classes of airspace, it is essential... that they demonstrate equivalence to the regulatory and airworthiness standards that are set for manned aircraft”.

Civil and military accident investigations have shown that airworthiness requirements are not always met by existing UAS operations [1, 2, and 9]. For example, a Canadian UAV was lost while the ground crew struggled with the engine management software in order to generate sufficient power to sustain flight. A lean mixture preset screw was set incorrectly, probably during routine maintenance. This led to a gradual fall in power during subsequent flights. The power reduction was not observed by the operating and maintenance teams until the day of the accident. Although the manufacturer’s service bulletins described different launch profiles, they did not consider engine management issues to the level of detail that characterises most conventional aircraft bulletins [9].

The same UAV design was involved in a further accident when its parachute retrieval system failed to fully deploy following a strong gust of wind. The avionics software exacerbated this failure because it would not deploy the airbags that protected the vehicle against hard landings until it had sensed the deployment of the parachute. Software upgrades and operational changes were introduced to address the cause of these accidents. They arguably illustrate a ‘fly-fix-fly’ approach to UAS airworthiness that would not be acceptable in other forms of aviation.

Airworthiness concerns are compounded by a loss of first-person liability. In other words, it is easier to maintain meticulous standards in maintenance and operation when an aircrew’s life is at risk. There are other potential explanations. For example, many UAV airframes use innovative designs in terms of their aerodynamics, avionics and ground control software. We have less experience in maintaining these platforms, compared to the long history of maintenance procedures for conventional aircraft. Other concerns relate to the difficulty of communicating maintenance information and software upgrades given that UAS are operated in many different areas around the globe. It can also be difficult for the operational teams to provide maintenance engineers with information about particular problems given that they, in turn, have to rely on limited information feeds provided to the Ground Control Systems.

Political pressures can also compromise airworthiness requirements. For instance, the NTSB report into the loss of the US Customs and Border Patrol’s (CBP) Predator near Nogales Arizona found that “At the time of the accident, CBP was unable to certify to the FAA that BP-101 was airworthy. Because of national security issues and past experience with similar UASs, the FAA temporarily waived this requirement for the issuance of the Certificate of Waiver or Authorization (COA) to operate in the National Airspace System (NAS)”. Section 6 of FAA 08-01 was not enforced because the CBP had been directed to start flying the Predator programme at short notice. There was political pressure to deploy UAVs as a means of reducing illegal cross-border immigration [2].

Airworthiness concerns affect all aspects of UAS operations from avionics to ground control software. They also have a secondary impact by increasing the workload of ground crews and, in the future, may place significant perceptual and cognitive burdens on air traffic controllers. Such preoccupations erode safety margins and can undermine the situation awareness of ATM personnel and ground control teams. It is critical that any future studies into the integration of UAVs build on these

observations by adopting a **systems perspective** that considers these knock-on effects, for example, on ATCOs' ability to operate complex computation infrastructures, including conventional communications systems, Medium Term Conflict Detection tools etc, while also responding to the UAS failure.

3 Radar Profiles and Transponders

A key requirement for the integration of UAVs into controlled airspace is that they must be 'visible' to the computational systems used by Air Traffic Management. This is particularly important given the anticipated role of 4D trajectory software systems within both the European SESAR and US NextGen initiatives. Many UAV airframes are considerably smaller than manned vehicles even though they have the potential to cause significant damage during any collision. The military heritage of these aircraft has also led to legacy designs that are deliberately hard to see on conventional radar systems. These effects are exacerbated when composite materials are used to reduce the mass of the UAV and, thereby, extend the operational duration of the platforms.

Some of these concerns can be overcome by the development of lightweight transponders and Automatic Dependent Surveillance-Broadcast (ADS-B) avionics that transmit identification and trajectory information. However, this is not a panacea. During the Nogales crash, mentioned above, the Predator's avionics software responded to a loss of communications with the Ground Control System by reducing power requirements. This was intended to extend the range of the UAV as it flew to an area where contact might be resumed. The transponder was one of the first systems to be disabled as a means of conserving power. The software, therefore, removed potential identification information at a time when Air Traffic Management personnel had the greatest need to locate the UAV.

A recent series of simulator studies have identified additional concerns that arise from the integration of UAS, even when transponder information is available [7]. Vogt and Udovic connected a Mission Management System (MMS) for a search task to a high altitude, high endurance (HALE) UAS simulator. This was then linked to an air traffic control simulator. Eight different traffic scenarios were studied for seven emergency situations. These included the combined failure of radio communication and data link connections to the UAV. Their results showed that ATCOs could cope with these failures in civil controlled airspace using current procedures. However, the ATCOs in the study argued that the transponder Emergency Squawk 7700 was not suitable for UAS data link losses. In this situation, the UAV is still fully operational and not necessarily in an emergency situation. They, therefore, proposed that a new unique transponder code would help ATM staff to identify this failure mode in autonomous flight. Vogt and Udovic's work is important because it provides a more optimistic view of UAS integration than the stance adopted by this paper. The divergence of views may, in part, be due to their use of simulation studies compared to the accident and incident data that motivates our work. It seems likely that the experience of future integration may lie somewhere between our alternate perspectives.

4 On-Board Software Verification and Certified FMS

UAV development costs have, typically, been lower than those of conventional aircraft. However, future integration requires that on-board avionics are of a comparable standard to those used in conventional aircraft. The FAA recognises that the testing of on-board software remains one of the biggest hurdles to the integration of UAS operations into controlled airspace. It is for this reason that they have entered into cooperative research and development agreements (CRDA) with commercial organizations to evaluate the integration of existing Flight Management Systems (FMS) into UAV platforms. The first flights began in December 2009. The purpose of this work is to demonstrate that a pre-certified system could be used within these unmanned platforms, significantly reducing the costs associated with the approval of a new UAV-specific FMS. During the initial test flights, the ground crew performed lateral path guidance to airways, flew direct to waypoints and issued airspeed and altitude commands during climb, cruise and descent. This established the potential use of the FMS to implement ATM commands in real-time trajectory negotiation. The immediate objectives for the next phase are to introduce 4D FMS control systems into FAA simulations of the National Airspace System (NAS). These simulations will be similar to those conducted by Vogt and Udovic [7].

The relative speed of innovation is a key issue. The re-use of an existing commercial FMS provides an accelerated route towards UAV integration. It is less easy to interpret the possible consequences for overall systems safety [8]. On the one hand, the application of a pre-certified FMS can be argued to free analytical resources to identify systems level risks. Commercial and regulatory bodies can re-task the teams that would otherwise have been used on UAS FMS certification to consider the wider hazards of integrated operations. A more sombre interpretation of the accelerated timetable might conclude that political and commercial pressures are working to ensure the rapid integration of reduced crew, freight operations into controlled air space.

The introduction of new technology also creates novel demands for software testing. For example, previous attention on the role of software in the integration of UAS into controlled air space has focuses on the development of advanced 'sense and avoid' techniques [1]. These systems use data fusion from a range of airborne systems to first identify and then respond to potential conflicts with other airspace users, in the same way that a pilot might respond to an Airborne Collision Avoidance System (ACAS) warning. It is difficult to identify appropriate verification and validation criteria for these future sensing technologies, especially given potential interactions between multiple UAVs with conventional aircraft.

5 Lost Link Profiles

The autonomous operation of lost link software raises further concerns for flight safety. Most UAS enable their operators to pre-program a number of waypoints that define a lost-link profile. This forms the route that the vehicle will fly should it loose contact with the Ground Control System. Many UAVs provide the ability to programme and store a number of these trajectories. The ground team must, therefore, nominate the most appropriate recovery route for the particular mission that they are

about to conduct. The ‘active’ lost link profile can also be changed during a mission as operational requirements lead the UAV away from its intended flight path.

The intention behind the creation and maintenance of pre-programmed lost link profiles is that the vehicle will autonomously navigate to a holding position where recovery teams can attempt to restore communications. In the worst case, these profiles indicate the bounds within which the UAV should come down when its power reserves are exhausted. FAA 08-01 requires that “In all cases, the UAS must be provided with a means of automatic recovery in the event of a lost link. There are many acceptable approaches to satisfy the requirement. The intent is to ensure airborne operations are predictable in the event of lost link...It is the applicant’s responsibility to demonstrate that injury to persons or property along the flight path is extremely improbable... UA with performance characteristics that impede normal air traffic operations may be restricted in their operations”. Similarly, UAS operations should avoid routes with heavy traffic or with open assemblies of people. These can only be approved in emergency or relief situations if ‘the proposed mitigation strategies are found to be acceptable’. This requires that applicants conduct a formal risk assessment with associated safety argument to demonstrate that the residual hazards are “extremely improbable”.

Experience has illustrated the practical difficulties that can arise when trying to meet lost-link safety requirements. For instance, the crew of the Nogales Predator were uncertain about its likely flight-path after communications were lost. There was a concern that it might crash on the route from its last known operating position to the first of the lost link waypoints. In the event, the UAV descended from segregated airspace into controlled airspace creating a significant hazard as it autonomously navigated towards its rendezvous. This incident demonstrates that unless greater care is taken to update and maintain pre-programmed lost-link profiles then there is a considerable danger for adjacent ATM operations even while UAVs operate in segregated airspace.

6 Ground Control Software Reliability Issues

Concerns over the role of computational systems in the integration of UAS into controlled airspace must extend beyond the airborne systems to also include the design, operation and maintenance of Ground Control Systems (GCSs). These applications provide the crucial link between the operating staff and the airborne systems. In addition to reliability concerns, there is a lack of standardisation both in terms of the user interfaces but also in terms of the training that different crews have when they operate ground control software.

Previous sections have identified the lack of ‘first person liability’ and ‘corporate responsibility’ that combined with the experimental nature of UAV operations to explain some safety concerns. Previous accident reports have also shown that poorly documented maintenance procedures and inadequate reviews of corrective actions have undermined the reliability of UAS Ground Control Systems [1]. For instance, several of the losses documented in our previous studies have been triggered by bugs that cause the ground control software to ‘lockup’ [2, 9, and 10]. In one instance, a subsequent review of the GCS software logs showed that there had been nine previous

lockups in the three months before the loss of the UAV. Two of these occurred before launch on the day of the accident. The ground team had developed an informal practice of correcting these failures by cycling the power on the frozen working position. Although this addressed the immediate symptoms of the problem, it did not identify longer term solutions nor did it provide any assurance that recycling the power avoided undesirable side-effects on the GCS configuration.

UAS, typically, exploit redundancy as a primary means of improving the reliability of Ground Control Systems. There are typically two or more workstations that can be reconfigured to support a number of roles. For instance, many UAVs distinguish between the Mission Planner who coordinates current and future operations and reports to outside agencies; the Air Vehicle Operator who controls and monitors the vehicle and the Payload Operator who performs similar functions for the imaging equipment. The Mission Planner and Air Vehicle Operator workstations are usually identical and provide additional redundancy in the case of failure. However, this redundancy is not always exploited in the ways envisaged by UAS suppliers. Another accident occurred when the processor board of one working position was simply exchanged with that of a neighbouring 'frozen' position [1]. Although this provided a temporary fix, the second position subsequently suffered the same hardware problems that had affected the first workstation.

7 Human Factors, SOPs and Situation Awareness

Doubts remain over the short term feasibility of the 'sense and avoid' techniques required to support autonomous operation. In consequence, it seems likely that UAS will continue to rely upon ground based operators. This creates enormous difficulties. It is extremely hard to create and maintain a high degree of remote situation awareness during UAS operations. This can be illustrated by the issues that arise in determining an optimum allocation of on-board sensing systems during different phases of flight. A recent accident occurred when the UAV was circling a potential landing site. The Payload Operator had skewed their camera at 90 degrees to acquire the recovery area. They did not see the mountain before the UAV flew into it. The loss of visual situation awareness was compounded because they crew had set the automated altitude warning at 200 rather than 300m AGL. This reduced the number of spurious alarms that were generated during routine flights in mountainous terrain. It also delayed the automated alarm so that the crew were only warned of a potential collision immediately before the accident. The large number of spurious alarms may also explain why the crew habitually ignored the aural warning associated with the altitude alarm. Situation awareness was also compromised by the crew's decision to display engine monitoring information on their workstations rather than the altitude screen that might have shown the potential danger from rising terrain.

In most other areas of aviation, Standard Operating Procedures (SOPs) would have provided guidance on the appropriate use of the software systems that are intended to support situation awareness. However, the crew involved in this accident did not have access to detailed SOPs. As in many other areas of UAS operations, the impetus to achieve operational benefits often overrides necessary caution. The crew also lacked a Standard Manoeuvre Manual, crew standard procedures, standard crew

terminology etc. These factors combined to create the context in which it is hardly surprising that they suffered such an extreme loss of orientation on their approach to the landing zone. The lack of SOPs and associated documentation also creates problems for ATCOs who must monitor and support any eventual integration into controlled airspace. Air Traffic Management personnel cannot assume that similar UAS control teams will respond in similar ways to similar situations unless they have been trained to use similar operating procedures.

Even when SOPs have been drafted to govern the operation of UAS, there is no guarantee that crews can maintain the level of situation awareness that is necessary to support interaction both with conventional aircrews and with ATM organisations [10]. In particular, it can be difficult to draft procedures that govern every possible contingency that might arise during the operation of a UAV. These include the failures of on-board avionics software but also changing meteorological conditions as well as the changing demands of particular missions. Any one of these factors can undermine situation awareness. Further problems arise from our very limited experience of UAS applications even in segregated airspace.

8 ATM Software Integration: Safety Nets

The integration of UAVs into controlled air space has particular importance for the configuration and operation of safety net applications. These are used to alert ATCOs or pilots to potential hazards with sufficient time for an adverse situation to be resolved. Ground-based safety nets exploit both radar and ADS-B surveillance data. These include the short term alerting systems that provide warnings times up to two minutes before an adverse event. Additional work would also be required to reconfigure medium term altering systems, which tend to operate up to 20 minutes before a potential incident. Area Proximity Warning (APW) systems alert ATM staff to the potential infringement of restricted airspace volumes. Similarly, Minimum Safe Altitude Warnings (MSAW) are issued when aircraft descend below pre-specified 'safe' altitudes. Approach Path Monitors (APM) will warn ATCOs about deviations from a pre-specified glide path. The integration of UAV's into controlled airspace poses particular problems for the operation of these software applications because these vehicles are difficult to track using conventional surveillance techniques. In consequence, the existing ground based safety nets may not function to the level of reliability that is assumed for existing aircraft. Further problems arise because increasing the sensitivity of surveillance equipment is likely to increase the number of false alarms.

In contrast, airborne safety nets warn the crews of conventional aircraft about potentially hazardous situations. Some of these provide warnings about on-board hazards, including fires or hull depressurisation and have an indirect impact on Air Traffic management. However, other airborne safety nets provide stall warnings, Ground Proximity Warnings (GPWS) and Airborne Collision Avoidance warnings (ACAS). ACAS is particularly significant in this respect as the berlingen accident has illustrated the consequences that can arise with on-board warnings contradict the clearances provided by ATC [11]. It is difficult to predict the impact that a UAV separation infringement might have on the crews of multiple conventional aircraft that would be alerted by airborne safety nets.

9 ATM Software Integration: Ground Movement Systems

Ground movement systems represent one of the most neglected areas of concern over the integration of UAVs into controlled airspace. Plans for future UAS freight operations, typically, assume that it will be possible to land these aircraft at existing airports. The economic arguments depend upon the integration of UAV cargo deliveries with existing ground-based distribution networks from regional aerodromes. This creates potential problems both for approach and airport control. The Linate runway incursion shows the increased workload that arises when ATCOs synchronise heterogeneous traffic movements [12]. In this accident, ATM personnel had to coordinate large passenger aircraft with the requirements of general aviation and of small commuter jets on the same runway infrastructures. The causes of the collision were exacerbated by communications problems between the ATCOs and the aircrews. Unless considerable care is taken then there is a danger that many of these existing concerns over the safety of ground movement will be exacerbated as ATM personnel coordinate conventional and unmanned operations.

One aspect of this integration between UAVs and ground movement operations is the ability to use Surface Movement Radar (SMR) systems. These software tools present ground controllers with real time updates on the location of aircraft and vehicles. They are intended to augment direct visual observations from the Control Tower and are particularly useful in poor weather or when ATM personnel have restricted views, for instance, following alterations to piers etc. However, SMRs also suffer from the reduced radar profile and composite fabrication of UAVs that create problems for airborne surveillance. During initial trials, it has been very difficult to distinguish UAS movements from those of ground vehicles. In consequence both the FAA and UK ANSP have begun studies to upgrade SMR as a means of helping Tower staff to monitor UAVs.

The challenges posed by UAS ground movements are not simply related to the software and hardware technologies that are used to support Air Traffic Management. Remote modes of operation and control create entirely new hazards. For example, in the past it has been possible to provide redundant communications infrastructures that ATCOs can use to make direct contact with aircrews if the primary ATM transmitter is not available. In the future, this will not be sufficient when the UAS GCS team may be many thousands of miles beyond the range of any transmitters that are directly under the control of the ANSP. The meta level point here is that the integration of UAVs will force ‘root and branch’ changes to the safety arguments and risk assessments that were previously developed to demonstrate that ATM operations were acceptably safe.

10 Longitudinal UAV GCS Crew Interaction

It is relatively straightforward to integrate models of UAV operations into existing ATM training simulators [7]. However, it can be hard to recreate a representative sample of traffic patterns or to identify the operating procedures that might be introduced to support UAS integration. For instance, we do not know how UAV GCS crews will interact with long haul freight operations. In some scenarios, control may

be handled from a single remote base. In other prototypes, control can be transferred in flight from a GCS at the point of origin to a second team handling the arrival and processing of the UAV at the destination airport. These different control models have a profound impact upon the workload both of GCS crews and also on ATM personnel. Most ANSPs lack the resources necessary to explore a broad range of possible scenarios for the integration of UASs. Hence it is difficult to ensure that we have simulated the broad range of potential hazards that might emerge from interactions between different computational systems within different categories of controlled airspace.

It is also difficult to predict the impact of UAS operations on the informal and undocumented working practices that continue to be a common feature of many Air Traffic Management organisations. These may violate recognised operating procedures. Examples include the Single Manned Operation and the informal shift hand-over practices that have been identified as a cause of previous accidents [11, 12]. Many of these behaviours emerge over long periods of time as teams of co-workers develop undocumented methods of working – often in response to the demands of high levels of traffic or conversely to deal with the boredom of quiet times. It may take weeks or months before teams identify these ad hoc working practices. In this case, uncertainty about future operating practices combines with uncertainty over potential control models to undermine the use of longitudinal simulations as a means of identifying hazardous interactions between ATM systems, UAV operations and other airspace users.

11 UAV Crew Interaction with ATCOs and Other Airspace Users

The economic arguments in favour of UAV operations often assume reduced manning levels or lower training requirements for GCS teams compared to conventional aircrews. This creates problems when ATM staff apply existing SOPs to support UAS integration with controlled airspace. Ground control teams may lack the levels of skill and expertise possessed by existing pilots. These problems are compounded when waivers are granted so that UAV crews do not even meet these reduced requirements. Such exemptions have contributed to the causes of several previous UAS accidents [1, 2, and 9]. They create significant concerns over the ability of UAV GCS teams to operate the software systems that are provided by their ground based workstations. Further concerns surround the uncertainty that exists over the roles to be performed by members of the UAS GCS crews. On the conventional flight deck, the division of tasks and responsibilities has emerged over several decades. The roles performed by the teams who operate Ground Control Systems seem to rely on a process of trial and error. A recent military deployment began with two working positions for a Mission Planner and Air Vehicle Operator. An Air Vehicle Commander role was then added when it was realised that two-person crews lacked sufficient expertise either as a pilot or a navigator. ‘Airworthiness concerns’ then led to the introduction of a fourth member of the GCS teams.

The gradual development of operational roles raises concerns about the exchange of information within UAV GCS teams as well as concerns for interaction with external agencies, including Air Traffic Management. Potential communications problems

can be illustrated by the lack of contact between the operators of the Nogales Predator and Air Traffic Management after the link was lost. Regulations required that the UAV pilot in charge must notify ATM of the following information after any failure in the communications link:

1. The UAS call sign.
2. UAS IFF [Identification, Friend or Foe] squawk.
3. Lost link profile.
4. Last known position.
5. Pre-programmed airspeed.
6. Usable fuel remaining (expressed in hours and minutes).
7. Heading/routing from the last known position to the lost link emergency mission loiter.

Instead, it was an ATCO who contacted the GCS crew after they lost contact with the UAV and the transponder had stopped working. The pilot did not inform them that the Predator had descended below its minimum permitted flight level (14,000 feet MSL). At this point, the pilot or the ATCO should have declared an emergency and taken measures to alert traffic in the area. They should have alerted neighbouring centres to monitor the missing vehicle. The ATC could also have started efforts to increase the level of surveillance on the UAV, for instance by contacting the Western Area Defence Sector to gather information using their height finding radar. These omissions illustrate the more general problems that arise from the understandable reluctance of a GCS crew to admit to any external agency that they have lost control of their vehicle.

Many UASs offer entirely new models of interaction. In particular, it is now possible to consider the circumstances under which an ATCO might order the destruction of a UAV without recourse to military action. The decision to destroy a vehicle in flight might be necessary if it could not be ordered off a trajectory that would take it towards a major population centre and there were intermittent communications failures. Such actions would be inconceivable for conventional aircraft. It is unclear how such an intervention would be initiated, for example whether it would necessarily involve consultation between the GCS crew and senior ANSP personnel or whether it could be directly commanded using future ATM workstations.

12 Conclusions and Further Work

Unmanned Airborne Systems (UAS) offer significant benefits. They can be operated for long duration missions by multiple teams of operators. They can also be used in situations where it would be inappropriate to risk conventional aircraft. However, UAVs continue to experience accident rates that are significantly higher than those for most conventional aircraft [3]. Partly in consequence, existing regulations segregate UAS operations away from areas under Air Traffic Management ('controlled airspace').

There are strong pressures to ease these restrictions. Police agencies would like to deploy miniature UAVs in populated areas, for example, to augment conventional helicopter operations. There are proposals to operate civil UAV freight operations

from existing airports. Longer-term proposals include the use of UAV technology to replace the co-pilot and allow single crewed operation. Automated systems might intervene only if the single human pilot is incapacitated.

This paper has identified a number of software-related safety concerns that arise from UAV operations within controlled airspace. In particular, we have identified the prominent role of software avionic failures in previous UAS accidents as well as issues relating to the programming of lost link profiles. We have also identified the difficulties that many UAV GCS teams face in maintaining remote situation awareness when they must simultaneously interact with ATM personnel, with other airspace users and with the remote vehicle.

We have also considered hazards from UAS integration for existing ATM software. Although much of the attention has been devoted to airborne safety nets, there are significant concerns over the ground movement systems that help ATCOs monitor and plan airport operations. The closing sections of this paper have looked at issues that cut across these different areas. For instance, there are concerns about the use of simulator studies to anticipate longer term interactions as ATCOs and GCS crews work together to integrate UAS operations with conventional traffic patterns. It seems very unlikely that the political and commercial pressures promoting UAS operations will provide sufficient time to adequately address all of these potential hazards.

References

1. Johnson, C.W.: The Safety Research Challenges for the Air Traffic Management Of Unmanned Aerial Systems (UAS). In: Kirwan, B. (ed.) Proceedings of the 6th EUROCONTROL Experimental Centre Safety Research and Development Workshop, Munich, Germany, October 21 & 22. EUROCONTROL Experimental Centre, Brétigny (2009)
2. Johnson, C.W.: Insights from the Nogales Predator Crash for the Integration of UAVs into the National Airspace System under FAA Interim Operational Guidance 08-01. In: Livingston, J.M., Barnes, R., Swallom, D., Pottraz, W. (eds.) Proceedings of the 27th International Conference on Systems Safety, Huntsville, Alabama, USA 2009, pp. 3066–3076. International Systems Safety Society, Unionville (2009)
3. Patchett, C., Sastry, V.: A Preliminary Model of Accident Causality for Uninhabited Autonomous Air Systems and Its Implications for Their Decision Architectures. In: Proceedings of the Tenth International Conference on Computer Modelling and Simulation (UKSIM 2008), pp. 487–492 (2008)
4. FAA Aviation Safety Unmanned Aircraft Program Office AIR-160, Unmanned Aircraft Systems Operations in the U. S. National Airspace System. Interim Operational Approval Guidance 08-01, Washington DC, USA (March 13, 2008)
5. EUROCONTROL Specifications For The Use Of Military Unmanned Aerial Vehicles As Operational Air Traffic Outside Segregated Airspace, Brussels, Belgium, EUROCONTROL-SPEC-0102 (July 26, 2007)
6. UK Civil Aviation Authority, Directorate of Airspace Policy, Unmanned Aircraft System Operations in UK Airspace – Guidance, London, UK (April 6, 2010), <http://www.caa.co.uk/docs/33/CAP722.pdf> (last accessed June 2010)

7. Vogt, J., Udovic, A.: Validation of Unmanned Aircraft Systems (UAS) Integration into the Airspace – The VUSIL Project. German Aviation and Space Congress, Darmstadt, Germany (September 2008)
8. Williams, K.W.: A Summary of Unmanned Aircraft Accident/Incident Data: Human Factors Implications, DOT/FAA/AM-04/24, Office of Aerospace Medicine (December 2004)
9. Johnson, C.W.: Act in Haste, Repent at Leisure: An Overview of Operational Incidents Involving UAVs in Afghanistan (2003-2005). In: Casely, P., Johnson, C.W. (eds.) Third IET Systems Safety Conference, NEC, Birmingham, UK, IET Conference Publications, Savoy Place (2008)
10. Johnson, C.W., Shea, C.: The Hidden Human Factors in Unmanned Aerial Vehicles. In: Simmons, R.J., Mohan, D.J., Mullane, M. (eds.) Proceedings of the 26th International Conference on Systems Safety, Vancouver, Canada, International Systems Safety Society, Unionville (2008)
11. Johnson, C.W., Kirwan, B., Licu, T., Statsny, P.: Recognition Primed Decision Making and the Organisational Response to Accidents: Überlingen and the Challenges of Safety Improvement in European Air Traffic Management. *Safety Science* (47), 853–872 (2009)
12. Johnson, C.W.: Linate and Überlingen: Understanding the Role that Public Policy Plays in the Failure of Air Traffic Management Systems. In: Balducelli, C., Bologna, S. (eds.) Proceedings of the ENEA International Workshop on Complex Networks and Infrastructure Protection, International Emergency Management Society/Italian National Agency for New Technologies, Energy and the Environment, Rome, Italy, pp. 508–519 (2006)

Residual Error Probability of Embedded CRC by Stochastic Automata

Frank Schiller and Tina Mattes

TU München, Department of Mechanical Engineering,
Chair of Information Technology in Mechanical Engineering,
Boltzmannstr. 15
D-85748 Garching near Munich, Germany
{schiller,mattes}@itm.tum.de

Abstract. Cyclic Redundancy Check (CRC) is an approved coding technique to detect errors in industrial communication. Using a checksum calculated with the help of a generator polynomial, CRC guarantees a low probability of undetected errors (residual error probability, P_{re}). The choice of an appropriate polynomial has significant impact on the quality of error detection. Since CRC itself is very efficient, it is obvious to embed safety-critical data protected by an additional CRC into the net data protected by the original CRC in order to increase the error detection of the safety-critical data. The paper introduces a method to determine the corresponding P_{re} by means of stochastic automata. Using the example of the fieldbus PROFIBUS-PA as embedding communication protocol, polynomials for the additional CRC were analyzed. As result, the impact of generator polynomials in the additional CRC on the P_{re} as well as the improvement of the error detection capabilities is shown.

Keywords: Cyclic Redundancy Check (CRC), Residual Error Probability, Safety-critical Communication, PROFIBUS-PA.

1 Introduction

The transmission of digital data (e.g. from sensors to processing units) is an essential function of industrial plants. Especially in safety-critical applications, the integrity of data is of high importance because undetected errors could cause dangerous accidents. Since falsification of data during transmission cannot be precluded, it is important to detect transmission errors to initiate the transition of the plant into a safe state.

There are various coding techniques for error detection. An efficient and common coding technique is Cyclic Redundancy Check (CRC). Efficiency means here, that a low probability of undetected errors can be assured with a relative small number of checksum bits. These checksum bits can be considered as the “cost” of the detection of errors and therefore, the number of those bits should be as small as possible. The checksum bits are calculated with the help of a so-called generator polynomial, which has remarkable impact on the quality of error detection [1], [2].

Since CRC itself is very efficient, it is obvious to implement, whenever possible, an additional CRC in order to increase error detection capabilities. A practical scenario is to embed safety-critical data protected by an additional CRC into the net data protected by the original CRC. The resulting telegram of embedded CRC has the following structure:



Fig. 1. Embedded CRC

In Fig. 1, ND₁ denotes safety-critical data, FCS₁ the checksum of the additional CRC, ND₂₁, ND₂₂ denote standard data, and FCS₂ the checksum of the original standard CRC.

The idea of embedded CRC is not new. Embedded CRCs are applied in communication e.g. when a standard fieldbus is used with a CRC (usually implemented in one of the lower layers of the well known ISO/OSI model) and an additional CRC is implemented in the application layer. ND₁ denotes in this scenario data of the application layer, and ND₂₁, ND₂₂ denote data of the regular fieldbus communication. In the regular safety proof, the CRC of the lower layer is not exploited. One reason is that no methods to determine the quality of these embedded CRC were at hand.

To clear this shortcoming, the paper presents a method to calculate the residual error probability (P_{re}) of embedded CRC¹. The residual error probability is the probability that a falsified telegram is regarded to be transmitted correctly. Furthermore, the impact of the choice of an appropriate polynomial for the additional CRC on the P_{re} is shown, using the example of PROFIBUS-PA [3].

The paper is structured as follows. Mathematical principles of CRC are given in the next section. After that, the calculation of P_{re} by stochastic automata is explained. Then the nesting and the determination of its residual error probability are introduced. Results and examples in the example of PROFIBUS-PA follow before final conclusions are drawn.

2 Basics of CRC

In this section, basic principles of CRC are summarized. For detailed information see e.g. [5].

2.1 Modeling CRC by Polynomial Modulo Division

Using CRC, a checksum for telegrams is calculated and evaluated for error detection. The encoding and decoding can be modeled by a polynomial modulo division. In the sender, the original message (called information bits or net data ND) consisting of k bits is handled as a binary polynomial $nd(x)$ of degree $k-1$ in the sense that the rightmost bit is the coefficient of x^0 , the next bit to the left is the coefficient of x^1 , ... For

¹ This paper is an analog to [4]. Here, the determination of the P_{re} of a slightly different nesting is explained by a different method that is more suitable in the case of long telegrams.

calculation of the checksum, a so called generator polynomial $g(x)$ of degree r has to be chosen. The polynomial $nd(x)$ is multiplied by x^r and divided by $g(x)$.

$$(nd(x) \cdot x^r) \bmod g(x) = fcs(x) \quad (1)$$

The remaining polynomial of the division $fcs(x)$ represents the checksum FCS, i.e. the coefficients of $fcs(x)$ are handled as a pattern of r bits, where the coefficient of x^{r-1} represents the leftmost bit and the coefficient of x^0 the rightmost bit. To build a telegram T , the FCS is attached to ND (usually at the end, i.e. $T = [ND \text{ FCS}]$).

For instance, $ND = [100101]$ leads to $nd(x) = 1 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0 = x^5 + x^2 + 1$. Assuming the generator polynomial $g(x) = x^3 + 1$, $fcs(x)$ is obtained by applying equation (1): $(x^5 + x^2 + 1) \cdot x^3 \bmod (x^3 + 1) = 1$. In consequence, $FCS = [001]$ and $T = [100101001]$.

The receiver checks if (2) holds, where $t'(x)$ is the polynomial counterpart of the received telegram T' :

$$t'(x) \bmod g(x) = 0? \quad (2)$$

If check (2) does not hold then the received telegram is erroneous and the error is detected, otherwise T' is regarded to be transmitted correctly.

2.2 Residual Error Probability

Obviously, there are some errors that cannot be detected by CRC. Therefore it is necessary to measure the quality of error detection. A precise criterion for error detection is the residual error probability P_{re} , which is the probability that an erroneous telegram is regarded to be transmitted correctly. The exact calculation of P_{re} is usually very complex.

Transmission errors can be modeled by superimposed error patterns F . These patterns have the same length n (number of bits) as T . A bit of F is allocated by value 0, if the corresponding bit in T is transmitted correctly, and a bit of F is allocated by value 1, if the corresponding bit in T was falsified during the transmission. As a result, T is superimposed by F such that $T' = T + F$ holds². A transmission error is undetectable by CRC if and only if $f(x)$, the polynomial counterpart of F , is divisible by the generator polynomial $g(x)$. Therefore, $t'(x) \bmod g(x)$ is equal to zero if and only if $f(x) \bmod g(x)$ is equal to zero. This fact is used to calculate P_{re} in Section 3.

2.3 Implementation of CRC

CRC can be implemented in software or hardware. A standard way for hardware implementation is the application of linear feedback shift register (LFSR, see Fig. 2). The number of bits of the register is equal to the degree r of $g(x)$. The g_i denote the coefficients of $g(x)$. The overall bit pattern of $z = [z_{r-1} \dots z_0]$ represents the state of the register. The input is symbolized by u . For initial state $z(0) = [0 \dots 0]$ and input u bits of $u(x) = nd(x) \cdot x^r$, the final state corresponds to the solution to bits of the FCS and for input u bits of $u(x) = t'(x)$ the final state corresponds to the solution of $t'(x) \bmod g(x)$.

² '+' represents exclusive-or in the space of binary polynomials and bit patterns.

The mode of operation of the LFSR is a basis to understand the method of stochastic automata presented in the following.

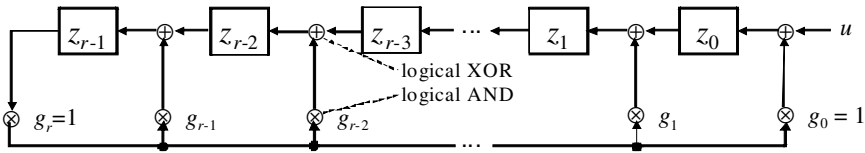


Fig. 2. Linear Feedback Shift Register

3 Method of Stochastic Automata

The idea of this method is to model the behavior of the LFSR by a deterministic automaton (Section 3.1), whose states relate to the states of the LFSR. The state transitions are extended by probabilities in a stochastic automaton to compute the residual error probability (Section 3.2). This method is well described in [6].

3.1 Deterministic Automaton

Let the input u to the LFSR be the error pattern F that is bitwise shifted into the register. The present state $z(k)$ in the register and the present input bit $u(k) = f(k)$ determine the next state $z(k+1)$ in the register. Hence, the register content represents the state z of a deterministic automaton. The appropriate polynomial notation of the final state at $k = n$, $z(n)$, is equivalent to the result of the operation $f(x) \bmod g(x)$. Table 1 shows three different notations of the state $z(k)$: bit patterns of length r , state numbers as natural numbers, and vectors of the vector space $\{0,1\}^r$. The state number notation is used in Fig. 3 and the vector notation in eqn. (3).

Table 1. Different notations of states

bit pattern	state no.	vector $z(k)$
[0 ... 0 0 0]	0	$(1\ 0\ 0\ 0\ 0\ \dots\ 0\ 0)^T$
[0 ... 0 0 1]	1	$(0\ 1\ 0\ 0\ 0\ \dots\ 0\ 0)^T$
[0 ... 0 1 0]	2	$(0\ 0\ 1\ 0\ 0\ \dots\ 0\ 0)^T$
[0 ... 0 1 1]	3	$(0\ 0\ 0\ 1\ 0\ \dots\ 0\ 0)^T$
...	...	...
[1 ... 1 1 1]	$2^r - 1$	$(0\ 0\ 0\ 0\ 0\ \dots\ 0\ 1)^T$

The deterministic automaton can be formulated by:

$$\begin{aligned}
 z(k+1) &= M_{u=0} \cdot z(k) \cdot (1-u(k)) + M_{u=1} \cdot z(k) \cdot u(k) \\
 z(0) &= (10\dots0)^T
 \end{aligned}
 \tag{3}$$

Matrices M_u denote transition matrices for given input $u = 0$ or $u = 1$, respectively. The first row and column of the transition matrices represent state 0, the second row and column represent state 1, $\dots$. Each row and each column contain exactly one element 1 and all others are 0. A coefficient m_{ij} of $M_{u=0}$ is at value 1 if the state j is reached from state i , if a bit at value 0 is put in the LFSR. Since no error could have occurred in the beginning, the process starts with state number 0, i.e. $z(0) = [0\dots 0]$. The corresponding signal flow graph is depicted in Fig. 3.

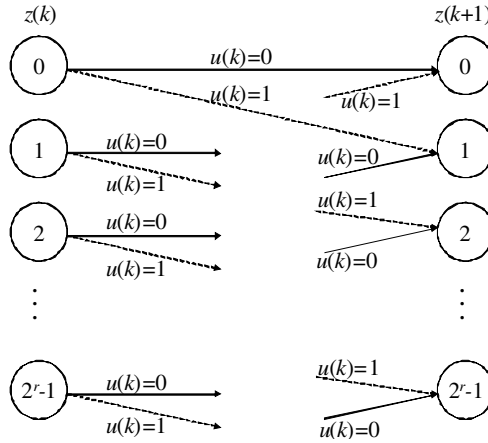


Fig. 3. Signal flow graph of the deterministic automaton

Let for example be $g(x) = x^2 + 1$. Hence, the states of the LFSR consist of three bits. The initial state is $z(0) = [z_2 \ z_1 \ z_0] = [000]$. If the first input bit is zero, then the state remains zero, $z(1) = [000]$, if the first input bit is 1, the following state is $z(1) = [001]$. Fig. 6 visualizes the correlations of states and input bits.

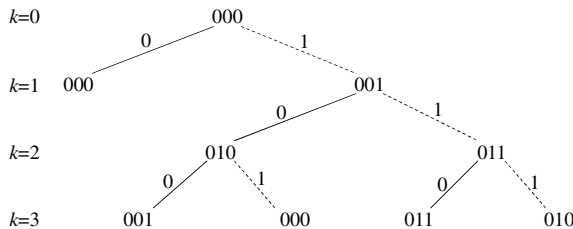


Fig. 4. Correlation of input bits and states of the LFSR in the example

The corresponding signal flow graph is given in Fig. 5.

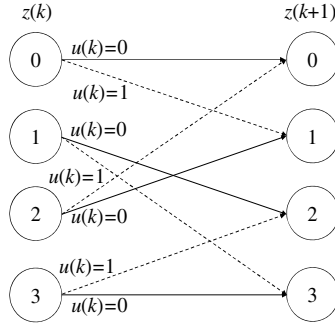


Fig. 5. Signal flow graph of the example

States 1, 2, ..., 2^r-1 represent detectable errors. At input bit $k = n$, state 0 or vector $z(n) = (1 \ 0 \ \dots \ 0)^T$ represent the case of $f(x) \bmod g(x) = 0$. Therefore, this state number 0 is the state of interest. Actually it contains two situations: the telegram is not falsified and the telegram is falsified undetectably. To separate these situations into different states, the automaton is extended in the way that state 0 is partitioned into two states: 0_{ok} for correct transmission and 0_f for an undetectable error, see Fig. 6.

The process starts here with the error-free state number 0_{ok} . Obviously, this state can only be reached from 0_{ok} with input 0.

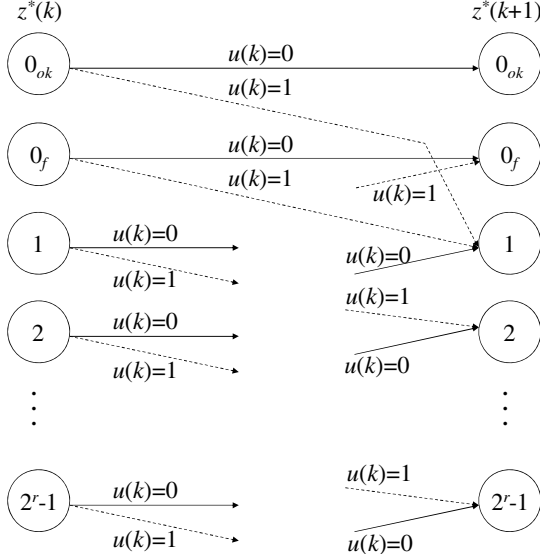


Fig. 6. Signal flow graph of the extended deterministic automaton

The transition matrices are adapted by an additional row at the top and an additional column at the left to present state 0_{ok} . The row and column that represented

originally state 0 is now assigned to state 0_j . The adaption is denoted by a^* (cp. (4)). State vectors are enlarged by one additional component correspondingly.

$$\begin{aligned} z^*(k+1) &= M_{u=0}^* \cdot z^*(k) \cdot (1-u(k)) + M_{u=1}^* \cdot z^*(k) \cdot u(k) \\ z^*(0) &= (10 \dots 0)^T \end{aligned} \quad (4)$$

3.2 Stochastic Automaton

In order not to put in each possible fault pattern into the automaton, the probability distribution $P(z^*)$ which denotes in every component the probability of being in the state z^* is involved in the model. Since error patterns occur with specific probabilities³ according to the bit error probability p , the input $u(k) = f(k)$ is at value 1 with probability p and is at value 0 with probability $1-p$. (cp. (5)). The transition matrices M^* are identical to the matrices in model (4).

$$\begin{aligned} P(z^*(k+1)) &= M_{u=0}^* \cdot P(z^*(k)) \cdot (1-u(k)) + M_{u=1}^* \cdot P(z^*(k)) \cdot u(k) \\ P(z^*(0)) &= (10 \dots 0)^T \end{aligned} \quad (5)$$

The final result of application of this model at $k = n$ is the probability distribution $P(z^*(n))$, where its second element (probability of state number 0_j) is equal to the residual error probability P_{re} .

The application of stochastic automata here enables an efficient analysis of polynomials for telegrams of lengths up to kilobytes. Additionally, the residual error probabilities of all telegrams of length k smaller than n are calculated as a side effect.

4 Determination of Pre of Embedded CRC

In this section, the embedded CRC is modeled and the determination of its residual error probability by means of stochastic automata is explained.

4.1 Embedded CRC

The embedded CRC is modeled as follows (cp Fig. 7). First data (e.g. safety-critical data), denoted by ND_1 , are used to calculate the first checksum, FCS_1 , according to (1) with generator polynomial $g_1(x)$, which is attached to ND_1 . Then the bit pattern $[ND_1, FCS_1]$ is embedded into a telegram such that the resulting bit pattern is given by $[ND_{21}, ND_1, FCS_1, ND_{22}]$ that is used to calculate a second checksum, FCS_2 according to (1) with generator polynomial $g_2(x)$. The structure of the corresponding telegram is given in Fig. 1.

³ The model of the binary symmetric channel is assumed, where bits are falsified independently and a bit is falsified during transmission with probability p (so called bit error probability).

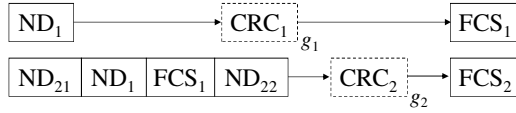


Fig. 7. Model of embedded CRC

The receiver checks if the received telegram $[ND_{21}', ND_1', FCS_1', ND_{21}', FCS_2']$ is divisible by $g_2(x)$ and if $[ND_1', FCS_1']$ is divisible by $g_1(x)$. Only if both checks hold, net data ND_1 are regarded to be transmitted correctly.

4.2 Determination of the Residual Error Probability

Only the detection of errors of safety-critical ND_1 is of interest. Therefore the residual error probability refers solely to falsification of ND_1 , i.e. undetectable errors caused by falsification of only ND_{21} , ND_{22} , FCS_1 , and FCS_2 are not included in P_{re} .

The determination of P_{re} bases on a crossing of the states of the automata z_1^* and z_2 that refer to the combination of two LFSR in Fig. 8. Since the falsification of ND_{21} and ND_{22} is not of interest, it is sufficient to use automaton z_2 instead of z_2^* .

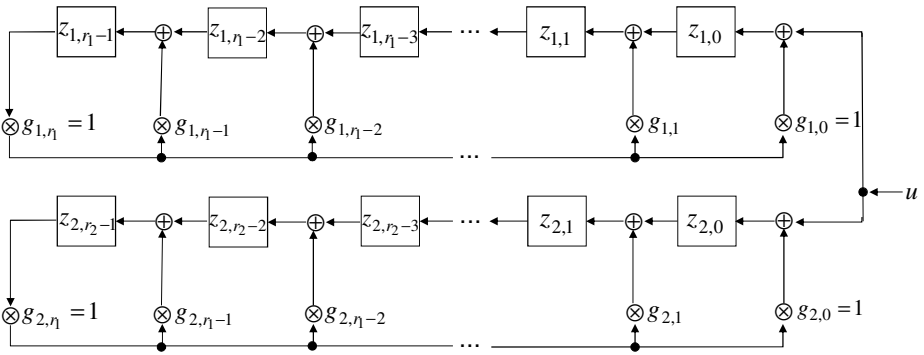


Fig. 8. LFSR for embedded CRC

Let $z_1^*(k+1) = f_1(z_1^*(k), u(k))$ and $z_2(k+1) = f_2(z_2(k), u(k))$ be the states of the automata for CRC with generator polynomial $g_1(x)$, respectively $g_2(x)$. A third automaton $z_3^*(k+1) = f_3(z_3^*(k), u(k))$ where $z_3^*(k) = z_1^*(k) \times z_2(k)$ is defined. This automaton has the states given in Table 2.

Table 2. States of crossed automata

$(0_{ok}, 0)$	$(0_{ok}, 1)$	$\dots$	$(0_{ok}, 2^{r_2-1})$
$(0_f, 0)$	$(0_f, 1)$	$\dots$	$(0_f, 2^{r_2-1})$
$(1, 0)$	$(1, 1)$	$\dots$	$(1, 2^{r_2-1})$
$\vdots$	$\vdots$	$\dots$	$\vdots$
$(2^{\eta-1}, 0)$	$(2^{\eta-1}, 1)$	$\dots$	$(2^{\eta-1}, 2^{r_2-1})$

A cut-out of the corresponding signal flow graph is given in Fig. 9.

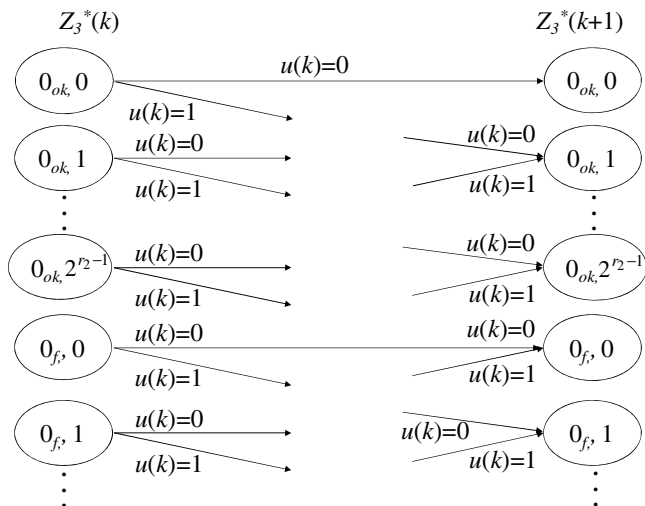


Fig. 9. Signal flow graph of automaton with crossed states

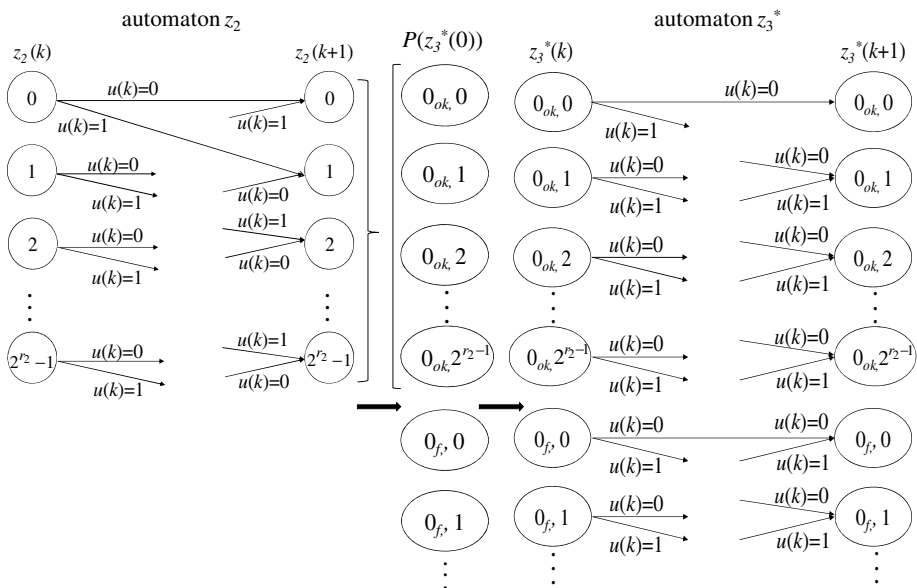


Fig. 10. Scheme 1 of the calculation of P_{re} for embedded CRC

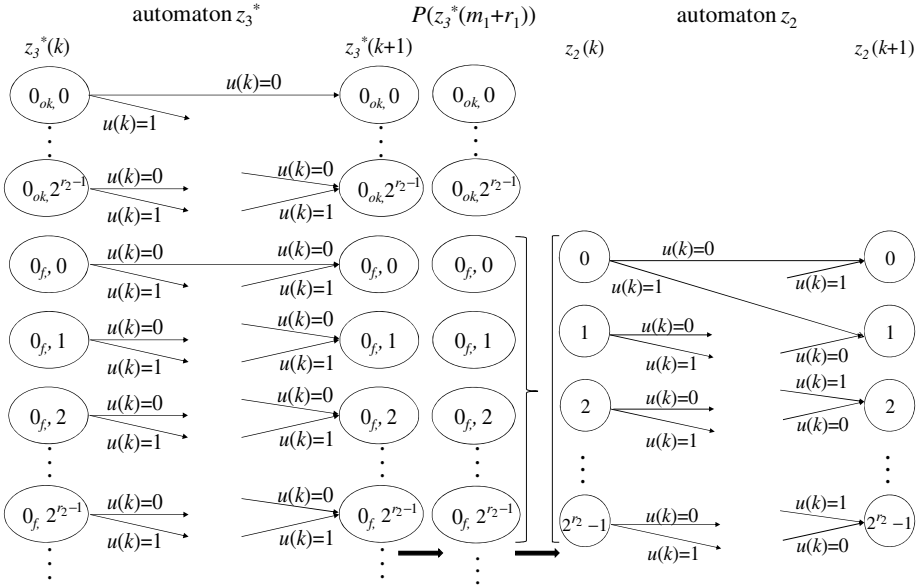


Fig. 11. Scheme 2 of the calculation of P_{re} for embedded CRC

Let m_{21} , m_{22} , m_1 denote the number of net data bits, ND_{21} , ND_{22} , ND_1 and r_1 , r_2 the number of bits of FCS_1 , FCS_2 , respectively. For the determination of P_{re} of the embedded CRC, first the probability distributions of the states for input bit one to input bit m_{21} , $P(z_2(1))$, ..., $P(z_2(m_{21}))$ are calculated by means of the stochastic automaton z_2 and initial vector $P(z_2(0)) = (10...0)^T$. The values of $P(z_2(m_{21}))$ correspond to the values of states $(0_{ok}, 0)$, $(0_{ok}, 1)$, $(0_{ok}, 2)$, ..., $(0_{ok}, 2^{r_2} - 1)$ of the initial vector $P(z_3^*(0))$ for z_3^* (cp. Fig. 10). $P(z_3^*(0))$ has further $2^{r_1} - 1$ components of value 0.

With $P(z_3^*(0))$ and z_3^* , $P(z_3^*(m_1+r_1))$ is calculated according to (5). Then, the probabilities of states $(0_f, 0_f)$, $(0_f, 1)$, $(0_f, 2)$, ..., $(0_f, 2^{r_2} - 1)$ are of interest since the inner CRC does not detect the falsification there. These states build the second initial vector $P(z_2(0))$ which is used to calculate $P(z_2(m_{22}))$ with automaton z_2 (s. Fig. 11). The residual error probability is the first component of $P(z_2(m_{22}))$ which is the probability of state $(0_f, 0)$, that represents a falsification that affects ND_1 and is undetectable by both CRC.

5 Analysis of Various First CRC Embedded in PROFIBUS-PA

The residual error probability of various generator polynomials for the inner CRC for various lengths of ND_1 that is embedded in the PROFIBUS-PA telegram frame has been calculated.

The structure of a PROFIBUS-PA telegram frame is given in Fig. 12. Here, the following abbreviations are used: PR: Preamble, P-SD: Physical Start Delimiter, SD: Start Delimiter, DA/SA: Destination/Source Address, FC: Control Byte, P-ED:

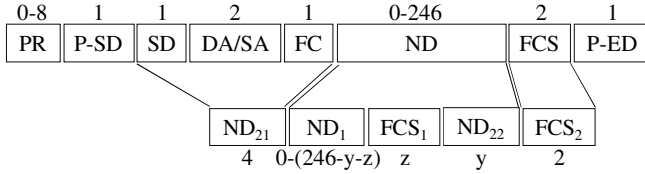


Fig. 12. PROFIBUS-PA telegram frame

Physical End Delimiter. The numbers above the elements of the telegram denote their length in byte.

If not all of the allocated 246 bytes are needed for the net data, an additional CRC can be implemented in the application layer. To map the elements of the telegram to the denotation of Section 4, SD, DA/SA and FC are summed up to ND_{21} , ND is parted into ND_1 and ND_{22} , FCS is denoted by FCS_2 , z denotes the number of bytes of the additional CRC ($z = \lceil n_1/8 \rceil$) and y the number of bytes of ND_{22} . The generator polynomial used in PROFIBUS-PA is $g_2(x) = 11021h^4$, in consequence $r_2 = 16$. Since SD, DA/SA, and FC are all together of length 4 byte, $m_{21} = 32$.

Various generator polynomials of degree 8 and 12 for various lengths of ND_1 and no ND_{22} (i.e. $y = 0$, Fig. 12) where compared with respect to their residual error probability in the nesting. The results for polynomials of degree 8 are described in Section 5.1 and for polynomials of degree 12 in Section 5.2.

5.1 First Polynomial of Degree 8

For net data length $m_1 = 256$, all binary polynomials of degree eight have been analyzed. For net data length $m_1 = 64$, only so-called primitive⁵ polynomials and polynomials that are a product of $(x+1)$ and a primitive polynomial of degree seven have been analyzed.

In Fig. 13, the residual error probability for ND_1 of length 256 over variable bit error probability p without additional CRC is given by the graph with markers. The drawn through, dashed, and dotted lines display P_{re} for PROFIBUS-PA with an additional CRC using the generator polynomials 111h, 171h, and 103h respectively. All these polynomial have degree eight and therefore cause the same online-effort of eight additional bits for the FCS_1 . Fig. 13 shows, that P_{re} can be reduced by factor ten if an additional CRC with the reducible generator polynomial 111h is implemented and that it can be reduced by factor 100 if the primitive polynomial 171h is chosen. A significant reduction of P_{re} is achieved, if the reducible polynomial 103h is generator polynomial in the first CRC. Assuming a bit error probability of $p = 10^{-10}$, P_{re} of the nested CRC with $g_1 = 103h$ is approximately 10^{20} times smaller than P_{re} of the plain CRC of PROFIBUS-PA.

The example shows, that the implementation of an additional CRC can improve the error detection capabilities significantly. Moreover, it makes obvious that the analyses

⁴ Polynomials are denoted hexadecimally, i.e. $11021h \rightarrow 1\ 0001\ 0000\ 0010\ 0001_2 \rightarrow x^{16} + x^{12} + x^5 + 1$.

⁵ A polynomial $g(x)$ of degree r is primitive if its only divisors are $g(x)$ and 1 and if the smallest number e with $x^e \bmod g(x) = 1$ equals $2^r - 1$.

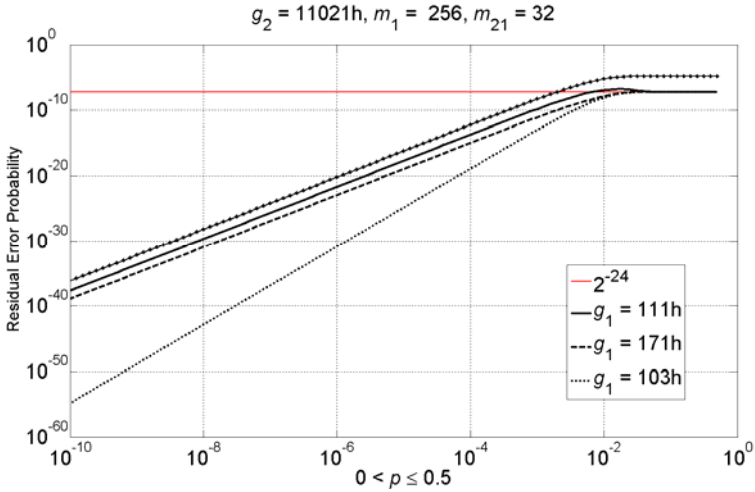


Fig. 13. P_{re} for $m_1 = 256$ and three generator polynomials of degree eight

of polynomials for the additional CRC is worth its efforts since there are remarkable differences in the quality of polynomials.

In general, the analyses showed, that in case of 256 net data bits and generator polynomials, that are a product of $(x+1)$ and a primitive polynomial of degree seven in the additional CRC, are a very good choice. Here, 15 of the 18 polynomials in this class guarantee same P_{re} as 103h. Regarding primitive and irreducible polynomials, only half of them show the same quality as 103h. In the case of reducible polynomials, only one third are as good as 103h.

The achieved results are similar in the case of 128 net data bits. In the class of products that are a product of $(x+1)$ and a primitive polynomial of degree seven, 17 of 18 polynomials guarantee the same bit error probability as 103h (the only exception and therefore not recommendable is 15Dh) and 13 of the 16 primitive polynomials (exceptions here are 12Bh, 14Dh and 18Dh).

Regarding the example of 64 net data bits, all polynomials that are a product of $(x+1)$ and a primitive polynomial of degree seven guarantee the same residual error probability as the primitive ones except 14Dh.

Resuming the achieved results concerning the various data lengths, it can be said, that the data length has to be considered for the choice of generator polynomials and therefore, one single polynomial that is good for each application cannot be pointed out.

5.2 First Polynomial of Degree 12

Since the calculation of P_{re} of three polynomials of degree twelve takes approximately 24 hours, not all 2024^6 polynomials were tested for a specific data length. The net data length chosen here is $m_1 = 800$.

⁶ There are actually $2^{12} = 4096$ binary polynomials of degree twelve but only polynomials that contain the monomial x^0 are considered.

In Fig. 14, P_{re} of ND_1 without additional CRC is given by the marked graph. It can be reduced by factor 1000, if an additional CRC with generator polynomial 1B6Fh (dotted line), which is a product of $(x+1)$ with a primitive polynomial of degree eleven, is implemented. A further reduction by factor ten can be achieved when the reducible polynomial 16BBh (dashed line) is used as generator polynomial. The best polynomial in this example is the primitive polynomial 1185h (drawn through line).

Altogether, eight of the 27 tested primitive polynomials, but only seven of the 32 polynomials that are a product of $(x+1)$ with a primitive polynomial of degree eleven, have the same quality as 1185h.

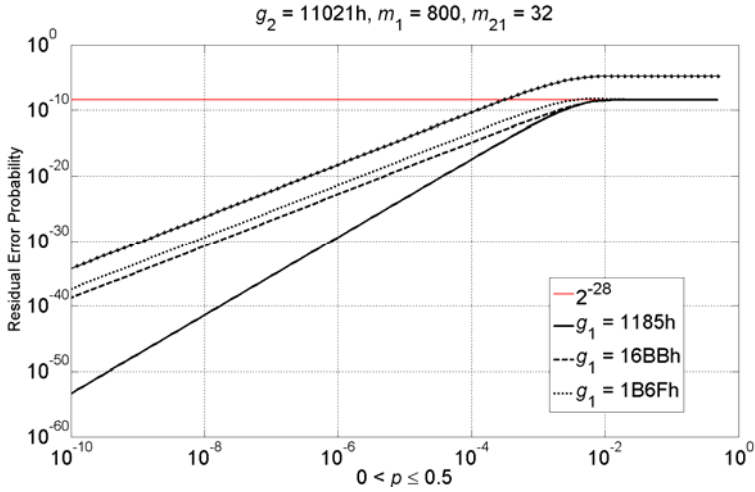


Fig. 14. P_{re} for $m_1 = 800$ and three generator polynomials of degree twelve

6 Conclusions and Future Work

A method to determine the residual error probability of embedded CRC was introduced in this paper. It is applicable:

- to exploit the outer CRC for the calculation of P_{re} to keep the number of bits of FCS_1 as small as necessary when new protocols are developed,
- to check in an existing certified embedded communication if an extension of ND_1 is feasible without any changes or if adoptions are necessary to maintain safety levels,
- to open new applications, e.g. by using an existing certified communication according to SIL 2 to send some data (ND_1) according to SIL 3 requirements.

This method is recommendable especially for long telegrams but is because of memory requirements restricted on a usual computer to applications where the sum of degrees of generator polynomials is smaller than 27. It has been shown that:

- with an additional CRC in the application layer the residual error probability can be reduced significantly,
- the choice of an appropriate polynomial for the additional CRC has significant impact on P_{re} ,
- the length of net data affects the choice of suitable generator polynomials and therefore polynomials cannot be recommended in general.

Because of the long computation time for the calculation of P_{re} for a specific polynomial, an efficient procedure that excludes improper generator polynomials from the beginning has to be developed, so that solely for potentially appropriate generator polynomials the residual error probability can be calculated. But the relatively long computation time is worthwhile, since the reduced residual error probability will open new fields of application.

References

1. Swoboda, J.: Codierung zur Fehlerkorrektur und Fehlererkennung. R. Oldenbourg Verlag, München, Wien (1973)
2. Sweeney, P.: Codierung zur Fehlererkennung und Fehlerkorrektur. Carl Hanser Verlag, München, Wien, Prentice-Hall Inc. London (1992)
3. Diedrich, C., Bangemann, T.: PROFIBUS-PA. Oldenbourg Industrieverlag, München (2006)
4. Mattes, T., Schiller, F., Mörwald, A., Honold, T.: Analysis of Nested CRC with Additional Net Data in Communication. In: Harrison, M.D., Suján, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 126–138. Springer, Heidelberg (2008)
5. Kløve, T., Korzhik, V.I.: Error Detecting Codes. Kluwer Academic Publishers, Boston (1995)
6. Schiller, F., Mattes, T.: An Efficient Method to Evaluate CRC-Polynomials for Safety-Critical Industrial Communication. Journal of Applied Computer Science 14, 57–80 (2006)

ANB- and ANBDmem-Encoding: Detecting Hardware Errors in Software

Ute Schiffel, André Schmitt, Martin Süßkraut, and Christof Fetzner

Technische Universität Dresden
Department of Computer Science
Dresden, Germany

`{ute, andre, suesskraut, christof}@se.inf.tu-dresden.de`
`http://wwwse.inf.tu-dresden.de`

Abstract. It is expected that commodity hardware is becoming less reliable because of the continuously decreasing feature sizes of integrated circuits. Nevertheless, more and more commodity hardware with insufficient error detection is used in critical applications. One possible solution is to detect hardware errors in software using arithmetic AN-codes. These codes detect hardware errors independent of the actual failure modes of the underlying hardware. However, measurements have shown that AN-codes still exhibit large rates of undetected silent data corruptions (SDC). These high rates of undetected SDCs are caused by the insufficient protection of control and data flow through AN-codes. In contrast, ANB- and ANBD-codes promise much higher error detection rates because they also detect errors in control and data flow. We present our encoding compiler that automatically applies either an AN-, ANB- or ANBD-code to an application. Our error injections show that AN-, ANB-, and ANBD-codes successfully detect errors and more important that indeed ANB- and ANBD-codes reduce the SDC rate more effectively than AN-codes. The difference between ANBD- and ANB-codes is also visible but less pronounced.

1 Introduction

In the future, decreasing feature sizes of integrated circuits will lead to less reliable hardware [6]. Currently used hardware-based solutions to detect hardware errors are expensive and usually an order of magnitude slower than commodity hardware [3]. Thus, due to economic pressure, more and more critical systems will be based on unreliable commodity hardware. However, commodity hardware not only exhibits fail-stop behavior but also more difficult to detect and to mask silent data corruptions (SDCs), i.e., they generate erroneous output instead of crashing. To use this unreliable hardware in critical systems, it is required to extend its limited failure detection capabilities with the help of software. We implemented a system that turns SDCs into much easier to handle stop failures – without the need for custom hardware.

When implementing detection of hardware errors in software, more CPU cycles are needed to execute an application. However, instead of custom reliable,

commodity hardware can be used. Typically, commodity hardware is not only cheaper than custom reliable hardware but also faster because it uses the newest hardware components. Furthermore, in many systems, only a few application components are critical and only these components need to be protected by additional error detection. Hence, we can bound the performance impact of software-based error detection by focusing on critical application components.

Our error detection approach is based on *arithmetic codes* (see Sec. 2) that support end-to-end software-implemented hardware error detection, i.e., protect data from undetected errors during storage, transport, and computations as well. Their error detection capabilities are decoupled from the actual hardware.

For using arithmetic codes, it is required to enable programs to cope with arithmetically encoded data. Therefore, we developed our *encoding compiler* that supports different arithmetic codes that we will introduce in Section 2: 1. AN-code, 2. ANB-code, and 3. ANBDmem-code. These codes provide different error detection rates at different runtime costs. Thus, systems engineers can balance gain and costs.

This paper presents the extension of our AN-encoding compiler presented in [14] with support for ANB- and ANBDmem-encoding. In contrast to the AN-code, the newly added ANB- and ANBDmem-code facilitate also the detection of data and control flow errors. While ANB-encoding of arithmetic operations was already presented in [17], this paper focuses on ANB/ANBDmem-encoding of control and data flow (see Sec. 3). In contrast to existing solutions such as [8], our encoding compiler supports arbitrary control and data flow that is not predictable at encoding, i.e., compile, time. Our evaluation (see Sec. 4) shows that indeed the amount of SDCs for ANB- and ANBDmem-encoded programs compared to unencoded programs goes down by 99.2% and 99.7% respectively. In contrast, AN-encoding leads only to a reduction by 93.5%. Furthermore, we show that compiler-based ANB/ANBDmem-encoding induces much less runtime overhead than our previously presented interpreter-based approach [18] that not even was as complete as the encoding compiler is.

2 Arithmetic Codes

Arithmetic codes are a technique to detect hardware errors during runtime. The encoding adds redundancy to all data words. Valid code words are only a small subset of all possible data words.

Correctly executed arithmetic operations preserve the code, i.e., given valid code words as input, the output is also a valid code word. A faulty arithmetic operation or an operation called with non-code words with high probability produces a result which is an invalid code word [2]. Furthermore, arithmetic codes also detect errors modifying data during storage or transport.

When an application is encoded using an arithmetic code, it will solely process encoded data, i.e., all inputs have to be encoded and all computations use and produce encoded data. Thus, we have to use solely operations that preserve the code in the error-free case.

AN-code. For an *AN-code* the encoded version x_c of variable x is obtained by multiplying its original functional value x_f with a constant A . To check the code, we compute the modulus of x_c with A , which is zero for a valid code word.

An AN-code can detect *faulty operations*, i.e., incorrectly executed operations, and *modified operands*, i.e., data that is for example hit by a bit flip. These errors are detected because they result in data that is not a multiple of A with high probability. The probability that such an error results in a valid code word is approximately $\frac{1}{A}$ [8]. Yet, when a bit flip happens on the (unencoded) address bus, a wrong memory word will be accessed that with high probability contains also a multiple of A . Thus, this so-called *exchanged operand* is not detectable with an AN-code because the error is also a multiple of A . A bit flip in the instruction unit of a CPU might cause the execution of a wrong operation (*exchanged operator*) that might also not be detected by an AN-code because many operators preserve an AN-code.

ANB-Code. Forin in [8] introduced *static signatures* (so-called “B”s). The resulting *ANB-code* can additionally detect exchanged operator and exchanged operand errors. The encoding of a variable x in *ANB-code* is defined as $x_c = A * x_f + B_x$ where B_x is chosen for each input variable with $0 < B_x < A$. To check the code of x_c , x_c ’s modulus with A is computed. The result has to be equal to B_x that is either assigned or precomputed at encoding time. Consider the following unencoded C code:

```
int f(int x, int y, int z) {
    int u = x + y;
    int v = u + z;
    return v;
}
```

Its ANB-encoded version¹ uses solely ANB-encoded data:

```
int_c f(int_c xc, int_c yc, int_c zc) {
    int_c uc = xc + yc;    // uc = A*x_f+B_x + A*y_f+B_y
                          //      = A(x_f+y_f)+B_x+B_y
    int_c vc = uc + zc;    // vc = A(x_f+y_f+z_f)+B_x+B_y+B_z
    return vc;             // expected: vc mod A == B_x+B_y+B_z
}
```

When encoding the program **f**, we assign static signatures to the input variables x , y , and z . Knowing the program, we can precompute the result’s expected signature $B_v = B_x + B_y + B_z$. Note that for implementing dynamically allocated memory, we use dynamic signatures that we introduced in [18]. These are assigned at runtime. If now an error exchanges variable y_c with another encoded variable $u_c = A * u_f + B_u$, the result’s computed signature $v_c \bmod A$ would be $(B_x + B_u + B_z)$ instead of the precomputed, i.e., expected, $(B_x + B_y + B_z)$. If the first addition is erroneously replaced by a subtraction, the resulting computed signature is $(B_x - B_y + B_z)$ instead of $(B_x + B_y + B_z)$. Thus, an ANB-code can detect exchanged operands and operators additional to faulty operations and

¹ The presented pseudo code is simplified and ignores the over- and underflow issues described in [17]. The comments depict the variable content in the error-free case.

modified operands. However, now consider that there is a bit flip on the address bus when storing variable y_c . Thus, we have a *lost update* on y_c because y_c is stored in a wrong memory location. When reading y_c the next time, the old version of y_c is read – which is correctly ANB-encoded but outdated.

ANBD/ANBDmem-Code. To detect the use of outdated operands, i.e., lost updates, Forin introduced a version D that counts variable updates [8]. In the resulting *ANBD-code*, the encoded version of x is $x_c = A * x_f + B_x + D$. The code checker has to know the expected D to check the validity of code words.

Currently, our ANBD-code implementation does only apply versions to memory that is accessed using load and store instructions but not to registers. Thus we denote it as *ANBDmem-code* in the following.

3 Encoding an Application

Encoding an application, i.e., enabling it to process encoded data, can be done at different stages of an application’s lifetime: before compilation by encoding the source code, during compilation by encoding an intermediate representation of the program, or at runtime by encoding the binary during execution.

Forin’s *Vital Coded Processor (VCP)* [8] ANBD-encodes an application on source code level. As we pointed out in [19], VCP requires knowledge of the complete data and control flow of the encoded program to precompute the signatures of all output variables for code checking. This prohibits the usage of dynamically allocated memory and function pointers. Furthermore, encoding loops and nested control flow structures at source code level is cumbersome and not described by Forin. Forin presents neither an evaluation of the error detection capability of VCP nor any runtime measurements.

Software Encoded Processing (SEP) introduced by us in [18] implements ANBD-encoding on assembler level at runtime. Therefore, we developed an interpreter for programs given as binary that itself is encoded using the principles of VCP [8]. Thus, we can encode arbitrary programs with arbitrary control flow. To encode dynamically allocated memory, *dynamic signatures* that are determined at runtime were introduced. The error injection results presented in [18] show that SEP successfully prevents erroneous output. However, the observed slowdowns make SEP unusable in practice.

In this paper, we present our *compiler based encoding (CBE)*. CBE encodes programs at the intermediate code level. In our case, by instrumenting LLVM code [9]. Adding the encoding at intermediate code level at compile time needs new concepts to encode the control flow. However, it makes encoding control flow easier compared to VCP because we have not to handle nested control structures explicitly. In contrast to the VCP, CBE provides support for programs with arbitrarily nested control structures and dynamically allocated memory. Furthermore, all programming languages for which an LLVM compiler exists can be supported. So far, we tested our implementation for C programs.

In contrast to SEP, CBE provides a more complete protection because: 1. it also encodes bitwise logical operations and floating point operations not covered by SEP, and 2. it also protects against bugs in the compiler back-end that

generates code for a specific machine. At the same time, CBE introduces much less overhead than SEP because no expensive interpretation is required. Furthermore, CBE restricts usage of expensive dynamic signatures to dynamically allocated memory. CBE uses *static* signatures (i.e., computed at compile time) for all statically allocated memory. In contrast, in SEP, every data item has a dynamic signature because all signatures are assigned at runtime due to the interpreter-based implementation.

For encoding a program with an AN-, ANB-, or ANBDbmem-code, every instruction and every variable has to be replaced with its appropriate encoded version. Thus, we need

1. encoded versions of all instructions supported by LLVM,
2. to encode all constants and initialization values,
3. to handle calls to external libraries, and
4. to encode control and data flow, that is, we have to check that instructions are executed in the correct order with the right operands and that all conditional jumps are executed correctly.

(1) Encoded Instructions. How we encode basic arithmetic and boolean operations we described in [17]. Encoding more complex operations such as bitwise logical operations, type casting, shifting, or floating point operations we described in [14]. In this paper here we focus on encoding control and data. This was not yet supported by our AN-encoding compiler presented in [14].

(2) Encoding Constants and Initializers. Since we choose A and the static signatures at encoding time, i.e., compile time, we can replace the unencoded constants and initializers with their encoded versions at compile time.

(3) External Calls. In contrast to SEP, the static instrumentation of CBE does not allow for protection of external libraries whose source code is not available at compilation time. For calls to these libraries, we currently provide hand-coded decoding wrappers, which decode (including code check) parameters and, after executing the unencoded original, encode the obtained results. For implementing those wrappers, we rely on the specifications of the external functions.

(4) Data and Control Flow (CF). While an AN-code only detects operation and modified operand errors, we can use an ANB-code in a way that ensures also the detection of exchanged operands and operators and arbitrary combinations of these errors. The ANBDbmem-code can also detect lost updates of memory.

VCP requires statically predictable control flow and allows output only at one specific point in the program execution. Only at this point execution errors are detectable because only there the code of the output is checked. In contrast, we implement for CBE a continuous checking of the program execution because 1. CBE allows output at arbitrary positions, 2. we do not know the control flow statically, and 3. CBE provides *fail-fast* behavior, that is, detects errors as fast as possible, thereby, allowing for an earlier reaction to them.

Therefore, our encoded application continuously produces check values, which it sends to a *watchdog*. The goal of the encoding is that if an execution error

happens, the encoded application will not send the expected check value to the watchdog. The expected check values are statically determined and given to the watchdog as an ordered list s , which is indexed by a counter i . i counts the received check messages. The encoded application also has a counter i for sent check messages. This allows the application to provide the expected check value in an error-free run. Therefore, the application contains a list δ , which has the same size as the watchdog's list s . However, δ contains the differences of consecutive elements of s , i.e., $\delta[i] = s[i + 1] - s[i]$.

We assign signatures to all input variables (parameters, memory reads, and return values of external functions) at encoding time. Using these signatures, we compute – also at encoding time – for every basic-block a *block signature* (BBx) that is the sum of the signatures of all results produced in this block.

Furthermore, we add an *accumulator* acc to the application. acc is initialized for each basic-block x so that it contains the next $s[i]$ minus the basic-block signature BBx . While the basic-block is executed, the signatures of all produced results are added to acc . At the end of the block, acc should equal $s[i]$ and is sent (**send**) to the watchdog. acc will not contain the expected value if any error modified the data flow, computations, or data. After sending acc , it is adapted for the next basic-block. Thereby, we can provide control flow checking. In contrast to existing solutions, our control flow checking provides more than inter-basic-block checking. We also check that every instruction was executed in the correct order, with the right operands, and its execution itself was error-free.

To prevent jumping from anywhere before a **send** of acc to any other **send**, we assign to each basic-block an ID BBx_{id} . The ID BBx_{id} is subtracted from acc before a block is executed and it is also sent to the watchdog. The watchdog checks if $acc + BBx_{id} == s[i]$. If not, the watchdog shuts down the application.

Inter-basic-block-CF and Unconditional Jumps. Consider the following example in LLVM bytecode:

```

1 bb1:
2   x = a+b
3   y = x-d
4   br bb2

```

Our ANB/ANBDMem-encoding compiler transforms this example to:

```

1 bb1: ; acc=s[i]-BB1-BB1_id   BB1=Bx+By=(Ba+Bb)+(Ba+Bb-Bd)
2   xc = addc(ac,Ba,bc,Bb) ; Bx=Ba+Bb
3   acc += xc mod A        ; acc=s[i]-BB1-BB1_id+Ba+Bb
4   yc = subc(xc,Bx,dc,Bd) ; By=Bx-Bd=Ba+Bb-Bd
5   acc += yc mod A        ; acc=s[i]-BB1-BB1_id+2*Ba+2*Bb-Bd
6                           ;   =s[i]-BB1-BB1_id+BB1 =s[i]-BB1_id
7   send(acc, BB1_id)
8   acc += delta(i)        ; acc=s[i+1]-BB1_id
9   i++                   ; acc=s[i]-BB1_id
10  acc += BB1_id-BB2-BB2_id ; acc=s[i]-BB2-BB2_id
11  br bb2

```

The comments (denoted by ‘;’) show the expected value of the accumulator. Note that **xc** means the encoded version of **x** where **x** can be either a variable or a function/instruction. Line 1 shows which value acc has at the beginning of

bb1. This is ensured by the previously executed block. Lines 2 and 4 contain the encoded versions of the original instructions whose signatures are added to *acc* directly after executing the instructions. In line 5, *acc* has the value $s[i] - BB1\_id$. In the next line, *acc* and the constant *BB1_id* are sent to the watchdog who checks if the sum of both values equals the expected $s[i]$. The following lines adapt *acc* for the next basic block. Line 8 ensures that *acc* will contain the next check value $s[i + 1]$ and line 10 adds $BB1\_id - BB2 - BB2\_id$. Note that this value is computed at compile time and, hence, is constant at runtime. Its addition removes this block's ID *BB1_id* and instead introduces the next block's ID *BB2_id* and signature *BB2*.

Conditional Jumps. Encoding conditional jumps additionally requires to check that the reached jump destination matches the actual branching condition. Consider the following example in which *cond* is the branch condition:

```

1 bb1:
2   cond = ...
3   br cond bb_true, bb_false

```

The encoded version is:

```

1 bb1: ; acc=s[i]-BB1-BB1_id    BB1=Bcond
2   condc = ...                ; condc=A*0+Bcond if cond is false
3   ;                          ; or A*1+Bcond if cond is true
4   acc += condc mod A          ; acc=s[i]-BB1-BB1_id+Bcond
5   send(acc, BB1_id)           ; acc=s[i]-BB1_id
6   acc += delta(i)             ; acc=s[i+1]-BB1_id
7   i++                        ; acc=s[i]-BB1_id
8   acc += BB1_id-BBtrue-BBtrue_id-(A*1+Bcond)
9   ;                          ; acc=s[i]-BBtrue-BBtrue_id-(A*1+Bcond)
10  cond = condc / A            ; get functional value of condc
11  acc += condc                ; acc=s[i]-BBtrue-BBtrue_id-(A*1+Bcond)+condc
12  br cond bb_true, bb_false_correction
13
14 bb_true: ; condc = A*1+Bcond => acc=s[i]-BBtrue-BBtrue_id
15   ...
16
17 bb_false_correction: ; condc=A*0+Bcond=Bcond
18   ; => acc=s[i]-BBtrue-BBtrue_id-A*1
19   acc += A+BBtrue+BBtrue_id-BBfalse-BBfalse_id
20   ; => acc=s[i]-BBfalse-BBfalse_id
21   br bb_false
22
23 bb_false: ; acc=s[i]-BBfalse-BBfalse_id
24   ...

```

In line 4 *acc* is used to check the computation of the condition *condc* with the already introduced approach. After sending *acc*, we adapt it in line 8 for the basic-block *bb_true* and for checking if the executed branch matches *condc*. For the latter, we subtract $A * 1 + Bcond$, the value *condc* has if *cond* is true. The value added in line 8 is a constant known at encoding time. In line 11, we add *condc*. If the condition is true, *acc* now contains the correct block signature and ID at the start of *bb_true*. If it is false, we have to do additional corrections which are executed in the basic-block *bb_false_correction* before jumping to the actual destination *bb_false*. These corrections ensure that when *bb_false* is entered, *acc* contains *bb_false*'s signature and ID. If the branch in line 12 does not match

condc, *acc* will not contain the expected block signature and ID and thus a wrong check value will be sent to the watchdog. Therefore, it is required that $BBfalse + BBfalse\_id \neq BBtrue + BBtrue\_id$.

Function Call. For a function call, we have to validate that 1. the correct function is called, 2. with the correct unmodified parameters, and 3. the function is executed correctly.

To ensure **1.**, we assign every function a *function signature* by which it has to modify *acc*. Before the function returns, it adapts *acc* for the remainder of the calling basic-block minus this function signature. For non-void functions, an additional signature is assigned to the return value. This guarantees a predictable signature for the return value.

For ensuring **2.**, we add the expected signatures of the parameters (known at encoding time and thus constant) to *acc* before entering the function. In the function, we subtract the signatures of the actual used parameters (computed at runtime). If they do not match, *acc* will become invalid. Afterwards, the signatures of the parameters are corrected to function-specific ones which are independent of the call-site. Therefore, statically computed correction values are used that depend on the call-site and, thus, are given as constant function parameters.

Before starting executing the function, *acc* is adapted. The remaining signature and ID of the basic-block which contains the call-site is removed and the signature and ID of the first basic-block of the function are added. The used correction value is determined at encoding time and provided as constant function parameter. Thereafter, the execution continues as described before – now executing and checking the basic-blocks of the function called. These measures ensure **3.**

Watchdog. The watchdog is used to check the correct execution of the encoded program during its runtime. It is not part of the encoded program. The watchdog needs to be executed reliably outside of the encoded program.

To check the execution, the watchdog checks if the sum of the received values *acc* and basic-block ID equals $s[i]$. If the watchdog encounters an unexpected $s[i]$ or the application stops sending values (detected using a timeout), the watchdog terminates the application. If the end of s is reached, both application and watchdog start again at the beginning of s by setting i to zero. In improbable scenarios, this might lead to undetected errors. Yet, the more entries s has, the smaller is the probability of such undetected errors.

The watchdog has to iterate over s , do periodic comparisons with the check values received, and has to test if the application is still alive. Its easy implementation supports the application of various mechanisms to make its execution safe, e.g., redundant execution on different hardware such as onboard FPGAs or the graphics unit, or hand-encoding according to VCP [8]. Additionally, we can use multiple watchdogs in parallel to further reduce the risk of an erroneous watchdog.

Memory. Up to now we focused on values stored in registers where we use static signatures known at encoding time. Since we cannot predict memory access patterns at encoding time, we need to use dynamic signatures, calculated at runtime, for values stored in memory. When storing a value, we convert its static signature into a dynamic signature that depends on the address the value is stored to. When loading a value from memory, we convert the dynamic signature back into a static signature that depends on the load instruction. These changes are also encoded.

Memory with Versions. The dynamic signature used for memory with versions depends additionally on the number of previously by the application executed stores (*version*). The version counter used is encoded, i.e., modifies *acc*. For a load, we have to remove the expected dynamic signature and version and replace them with the static signature of the destination register. These modifications and the signature management have to be encoded. The following listing demonstrates an ANBDbmem-encoded load operation. The ANB-encoded version looks similar but does not include the version removal in line 5. The `getVersion` function returns the expected version for a given address. It is also encoded. For the implementation of this encoded version management see [18]. We use version management with check-pointing because it provides good results for applications with high and low data locality.

```

1 uint64_t loadc(ptrc, Bptr, corr){ // corr=A*Br+Bptr
2   ptr = ptrc / A;                // decode address
3   vc = load(ptr);                // load value => vc=A*r+ptr+version
4   t = (ptrc-corr)/A;             // t=((A*ptr+Bptr)-(A*Br+Bptr))/A = ptr-Br
5   rc = vc-t-getVersion(ptr);     // rc=vc-(ptr-Br)-version
6   rc += (ptrc-corr) % A;         // additional check that (ptrc-corr) % A == 0
7   return rc;                    // =A*r+Br
8 }

```

The load takes an encoded pointer ptr_c , the expected signature B_{ptr} of ptr_c , and a correctional value $corr$. During encoding, we choose a value $B_r < A$ for the result's signature. Since B_{ptr} and A are also chosen at encoding time, for each call to load $corr = A * B_r + B_{ptr}$ is also constant at runtime. If a wrong or outdated address is read, the return value will not have the expected signature B_r in line 7. A store is implemented similarly.

4 Evaluation

We evaluated our approach using the following applications: `md5` calculates the md5 hash of a string, `tcas` is an open-source implementation of the traffic alert and collision avoidance system [1] which is mandatory for airplanes, `pid` is a Proportional-Integral-Derivative controller [21], `abs` implements an antilock braking system, and `primes` implements the Sieve of Eratosthenes.

Performance. Figure 1 depicts the slowdowns of encoded applications compared to their unencoded, i.e., unsafe, versions for the different codes. Time is measured for the complete application including I/O-operations.

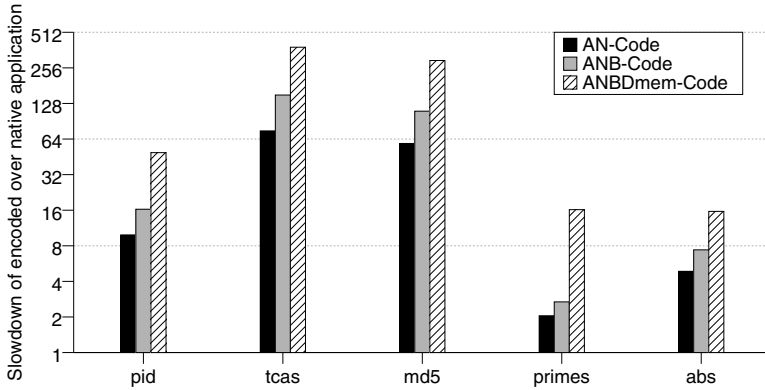


Fig. 1. Slowdowns of encoded application compared to their native versions

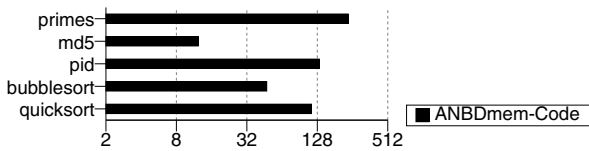


Fig. 2. Speedup of CBE compared to SEP

For the AN-code, the slowdown ranges from 2 (*primes*) to 75 (*tcas*). Applications using more expensive encoded operations such as multiplications or floating point operations exhibit larger slowdowns [17]. This leads to these strongly varying slowdowns. For example, *md5* contains an above average number of bitwise logical operations, which, in their encodable version, make extensive use of expensive encoded multiplications. The encoded version of *tcas* is much slower because of the extensive use of floating point operations.

The ANB-code is on average 1.9 times slower than the AN-code because it provides encoded control and data flow and the encoded operations used have to consider the signatures as well.

The slowdown of the ANBDmem-code compared to the ANB-code is on average 2.6. The main reason is the additional overhead needed to safely store and retrieve version information for dynamic memory. This overhead depends on the degree of the locality of the memory accesses executed.

One objective for CBE was to be faster than the interpreter-based SEP. Figure 2 compares for some applications the speedup of the most expensive CBE-variant (ANBDmem) compared to SEP. *tcas* and *abs* are not supported by SEP due to missing system calls. CBE always clearly outperforms SEP. We observe that the obtained speedups depend on the executed program. Especially *md5* has smaller speedups. *md5* contains an above average number of bitwise logical operations. However, SEP is incomplete. It especially does not support encoded versions of bitwise logical operations, shift operations, and casts. Those operations are just executed unencoded in SEP while they are encoded by CBE.

Error Detection. For evaluating the error detection capabilities of our encoded programs we used our error injector EIS [12]. It injects the software-level symptoms of possible hardware failures. We injected the following symptoms: exchanged operands, exchanged operators, faulty operations, modified operands, and lost stores. Further errors can be represented by combinations of these symptoms.

We applied those errors in three different modes:

Deterministic (Det) injects per run exactly one error. We execute approximately 50,000 runs for each benchmark and protection mechanism: 10,000 for each symptom. In each run another error is triggered. This tests the ability of a detection mechanism to cope with seldom occurring errors.

Probabilistic (Prob) injects an error with a probability given by us. We use the same error probability for all error detection mechanisms evaluated. At each possible point where an error (of any symptom) could be triggered an error is injected with the given probability. Thus, one execution might be hit by several different errors. With this mode we executed 6,000 runs per benchmark and per detection mechanism.

Permanent errors (Per) injects permanently faulty operation errors simulating permanent logic errors in the processor. Permanent errors are only applied to arithmetic integer operations, and loads and stores of integer values. We are injecting approximately 1,700 different permanent errors per benchmark, per detection mechanism – one error only per run.

All example applications are of similar size and we distribute the injections evenly over the program execution. Hence, with our fixed number of fault injection runs we achieve similar coverages for all applications. We chose the number of fault injection runs so that the experiments complete in a feasible time.

We compared the results of injection runs to the results of an error-free run to determine if the error injected resulted in a silent data corruption (SDC), i.e., a failure of the error detection, in an abort or a correct output, i.e., the error was masked. Figure 3 presents the results of the described error injection experiments. It focuses on the amount of SDCs because these identify a failure of the detection. Note the logarithmic scale.

We make the following observations: We see that in contrast to native, i.e., unprotected, programs the AN-encoded versions dramatically reduce the amount of SDCs, i.e., undetected errors. However, the AN-encoded versions still have a considerable amount of SDCs: on average 0.96%. The highest rate of undetected errors is 7.6% for **abs** and *det*. ANB-encoding reduces the amount of undetected errors to on average 0.07%. ANBDMem-encoding again halves the rate to on average 0.03% SDCs.

In contrast to unprotected applications, none of the encoded versions – independent of the code used – is vulnerable to permanent errors. Probabilistically (*Prob*) injected errors are also more often detected. The reason is that for both injection modes programs are more often hit by several errors. This increases the probability of detection as we have shown in [18].

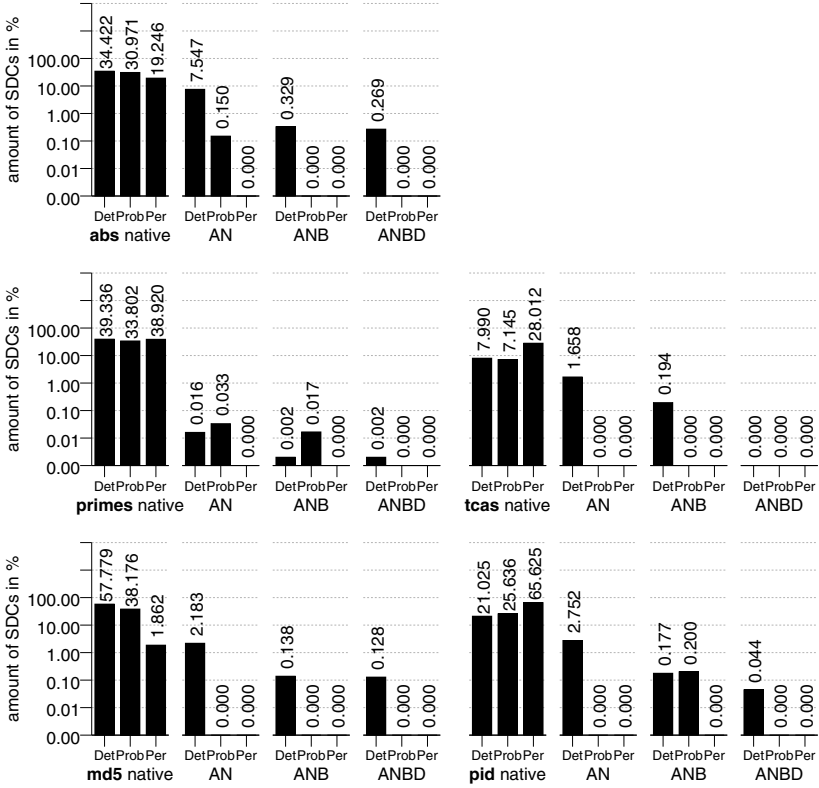


Fig. 3. Error injection results: Amount of SDCs for all error injection runs

To show the advantage of ANB and ANBDbmem-code over AN-code we compare the overhead with the detection rate. On average, the ANB-code has a about 14 times higher error detection rate than the AN-code while the slowdown increases on average only 1.9 times. The ANBDbmem-code has an about 32 times higher detection rate than the AN-code which comes at the cost of an about 5 times higher slowdown. Both the ANB- and the ANBDbmem-encoded programs are checked on the instruction level.

5 Related Work

Control flow checking, which can be implemented in hardware, e.g., [4], or software, e.g., [5], provides means to recognize invalid control flow for the executed program, that is, execution of sequences of instructions that are not permitted for the executed binary. In contrast to encoding, control flow checking cannot detect errors which do only influence processed data. Usually control flow checking is only done for inter-basic-block control flow. Our ANB- and ANBDbmem-encoded programs are checked on the instruction level.

Algorithm-based fault tolerance [15] and self-checking software [20] use invariants to check the validity of the generated results. Appropriate invariants have to exist which provide a good failure detection capability. They are not easy – if not impossible – to find for most applications.

Other software approaches work with replicated execution and comparison (voting) of the obtained results. The protected software is modified during or before compilation – rarely, dynamic binary instrumentation is used [11]. Replication is applied at different levels of abstraction. Some approaches duplicate single instructions within one thread, e. g., [7,11,4]. Others execute duplicates of the whole program using several threads, e. g., [16]. For all approaches that are based on replication, it is not possible to provide guarantees with respect to permanent hardware errors [13].

Instead of duplication, or in addition to, arithmetic codes can be used to detect errors. Therefore, the program and the processed data are modified. AN-encoding was already used by [10,7,14]. For all approaches the error injection experiments show a non-negligible amount of undetected failures. This is even higher for [10] and [7] because the used encoding is incomplete.

Forin's Vital Coded Processor (*VCP*) [8] and Software Encoded Processing (*SEP*) [18] use both an ANBD-code. We compared both approaches already to CBE in Sec. 3.

6 Conclusion

We introduced compiler based encoding (CBE) – especially control flow encoding using ANB- and ANBDbmem-encoding. Our experiments have shown that these two new encodings reduce the number of undetected errors more than AN-encoding. The reduction of undetected errors is higher than the increase in runtime that has to be paid for the more sophisticated protection of ANB- and ANBDbmem-encoding. Thus, safety engineers can balance the error detection coverage and the performance overhead by choosing the appropriate arithmetic encoding.

Our second goal was to provide a faster encoding mechanism than SEP. We clearly achieved this goal: on average, ANBDbmem-encoded applications are 108 times faster than their SEP version. Furthermore, CBE is more complete than SEP. In contrast to CBE, SEP does not support encoded bitwise logical operations, casts, shifts, and floating point operations.

References

1. The Paparazzi Project (2009), http://paparazzi.enac.fr/wiki/Main_Page
2. Avizienis, A.: Arithmetic error codes: Cost and effectiveness studies for application in digital system design. Transactions on Computers (1971)
3. Barnaby, H.J.: Will radiation-hardening-by-design (RHBD) work? Nuclear and Plasma Sciences, Society News (2005)

4. Bolchini, C., Miele, A., Rebaudengo, M., Salice, F., Sciuto, D., Sterpone, L., Violante, M.: Software and hardware techniques for SEU detection in IP processors. *J. Electron. Test* (2008)
5. Borin, E., Wang, C., Wu, Y., Araujo, G.: Software-based transparent and comprehensive control-flow error detection. In: *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, IEEE Computer Society, Los Alamitos (2006)
6. Borkar, S.: Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro* (2005)
7. Chang, J., Reis, G.A., August, D.I.: Automatic instruction-level software-only recovery. In: *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, Washington, USA (2006)
8. Forin, P.: Vital coded microprocessor principles and application for various transit systems. In: *IFA-GCCT*, September 1989, pp. 79–84 (1989)
9. Lattner, C., Adve, V.: LLVM: A compilation framework for lifelong program analysis & transformation. In: *Proceedings of the international symposium on Code generation and optimization (CGO)*, USA, IEEE Computer Society, Los Alamitos (2004)
10. Oh, N., Mitra, S., McCluskey, E.J.: ED4I: Error detection by diverse data and duplicated instructions. *IEEE Trans. Comput.* 51 (2002)
11. Reis, G.A., Chang, J., August, D.I., Cohn, R., Mukherjee, S.S.: Configurable transient fault detection via dynamic binary translation. In: *Proceedings of the 2nd Workshop on Architectural Reliability, WAR* (2006)
12. Schiffl, U., Schmitt, A., Süßkraut, M., Fetzer, C.: Slice Your Bug: Debugging Error Detection Mechanisms using Error Injection Slicing. In: *Eighth European Dependable Computing Conference (EDCC 2010)* (2010)
13. Schiffl, U., Schmitt, A., Süßkraut, M., Fetzer, C.: Software-Implemented Hardware Error Detection: Costs and Gains. In: *The Third International Conference on Dependability, DEPEND 2010* (2010)
14. Schiffl, U., Süßkraut, M., Fetzer, C.: AN-encoding compiler: Building safety-critical systems with commodity hardware. In: Buth, B., Rabe, G., Seyfarth, T. (eds.) *SAFECOMP 2009*. LNCS, vol. 5775, pp. 283–296. Springer, Heidelberg (2009)
15. Stefanidis, V.K., Margaritis, K.G.: Algorithm based fault tolerance: Review and experimental study. In: *International Conference of Numerical Analysis and Applied Mathematics* (2004)
16. Wang, C., Kim, H.S., Wu, Y., Ying, V.: Compiler-managed software-based redundant multi-threading for transient fault detection. In: *International Symposium on Code Generation and Optimization, CGO* (2007)
17. Wappler, U., Fetzer, C.: Hardware failure virtualization via software encoded processing. In: *5th IEEE International Conference on Industrial Informatics, INDIN 2007* (2007)
18. Wappler, U., Fetzer, C.: Software encoded processing: Building dependable systems with commodity hardware. In: Saglietti, F., Oster, N. (eds.) *SAFECOMP 2007*. LNCS, vol. 4680, pp. 356–369. Springer, Heidelberg (2007)
19. Wappler, U., Müller, M.: Software protection mechanisms for dependable systems. In: *Design, Automation and Test in Europe, DATE 2008* (2008)
20. Wasserman, H., Blum, M.: Software reliability via run-time result-checking. *J. ACM* (1997)
21. Wescott, T.: PID without a PhD. *Embedded Systems Programming* 13(11) (2000)

Field Test Methods for a Co-operative Integrated Traffic Management System

Thomas Gruber, Egbert Althammer, and Erwin Schoitsch

AIT Austrian Institute of Technology GmbH, Austria

{thomas.gruber, egbert.althammer, erwin.schoitsch}@ait.ac.at

Abstract. The European Project COOPERS (Co-operative Networks for Intelligent Road Safety) aims at developing co-operative systems based innovative telematics solutions to increase road safety. In the COOPERS approach, co-operative traffic management is implemented by using intelligent services interfacing vehicles, drivers, road infrastructure and operators. These services which involve various types of embedded systems and wireless communication channels are finally demonstrated in six European countries and evaluated with respect to their influence on driver behaviour and road safety. This scientific investigation requires good system reliability as well as accurate and deterministic system behaviour. The required system properties, including quantitative tolerance limits for temporal and spatial behaviour of the system, were specified as generic requirements in an early phase of the project. Before the final demonstrations, these requirements were verified, including statistical evaluations regarding the degree of fulfilment of single quantitative requirements. This paper presents the test bench and the test methods for validating this complex distributed real-time system. It explains how time synchronisation between the subsystems was handled and how the potential safety-criticality of the system was treated. It gives an insight into the values and parameters measured, and finally it presents some of the first results from the technical validation of COOPERS.

Keywords: Test, verification, validation, co-operative traffic management, traffic telematics, road safety, distributed systems, embedded systems.

1 Introduction

In the sixth framework program of the European Commission, one of the thematic main lines deals with road traffic safety. Several projects address this topic, but COOPERS¹ [1] takes a specific position with novel ways and methods to attain a safety improvement through an intelligent network which exploits existing technologies for co-operative services.

¹ Research supported in part by COOPERS (Co-Operative Networks for Intelligent Road Safety), an integrated project funded by the EU within priority “Information Society Technologies (IST)” in the sixth EU framework programme (contract no. FP6-IST-4-026814).

Based on information sources of the infrastructure, the driver is provided with real time data on the current traffic situation ahead (see [1]). In each car, a receiver for the I2V (infrastructure to vehicle) communication and a display offer information about accidents, traffic jams, road construction sites and other location and time related events. Only messages relevant for the driver on a particular segment are passed on. The information is accurate and precise both in terms of location and time.

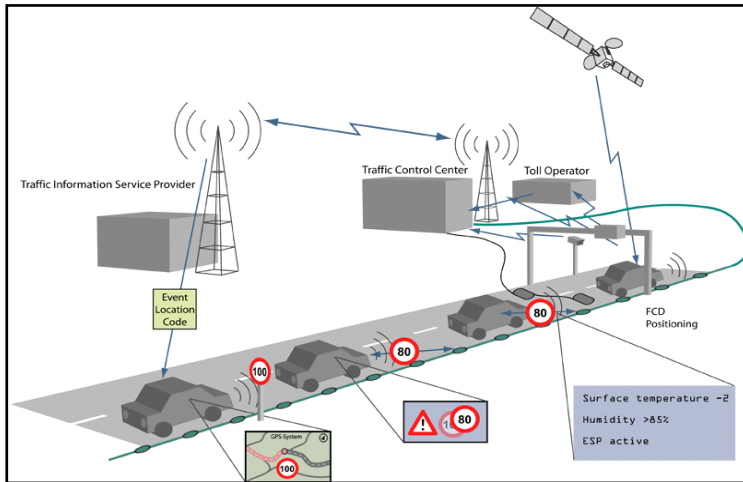


Fig. 1. Intelligent infrastructure and smart cars plus individual location based services – I2V and V2I communication

An implementation of COOPERS was developed with two different variants for the in-car devices and specific road-side infrastructure subsystems for the demonstration sites in six European countries. This implementation has been tested in the field with an efficient approach specifically developed for COOPERS.

The paper is structured as follows: section 2 gives an overview on the COOPERS system from a technical perspective. Section 3 gives an insight into the requirements which form the basis of the tests. Section 4 describes the verification and validation method we used; it explains the log file based combined subsystem and system test and proves why the assumption for our resource-saving approach is legitimate. Considerations on the measured parameters and the critical questions of temporal precision and special accuracy follow. Section 5 describes the value of the methods described here for the project COOPERS and gives an outline at the first test results. A summary and outlook on further work follows in section 6.

2 COOPERS System Overview

From a technical point of view, COOPERS is subdivided into two major subsystems communicating with one another on a wireless communication channel with various technologies. Figure 2 presents an overview on this structure:

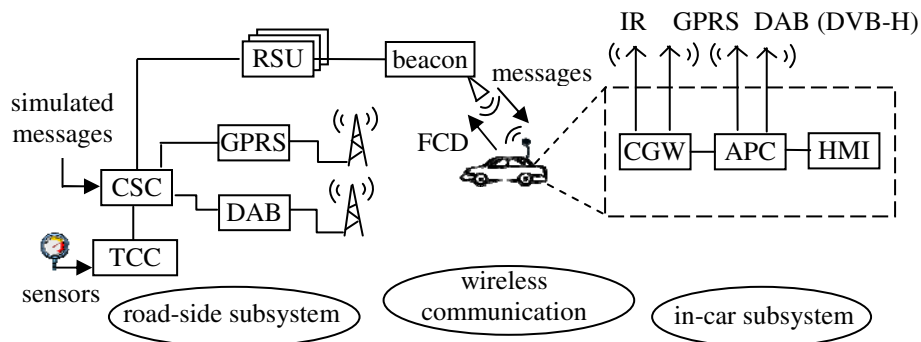


Fig. 2. COOPERS subsystem structure with communication links

Legend:

APC	Automotive PC	FCD	Floating Car Data
CGW	Communication GateWay	GPRS	General Packet Radio Service
CSC	COOPERS Service Centre	HMI	Human Machine Interface
DAB	Digital Audio Broadcast	IR	InfraRed
DVB-H	Digital Video Broadcast - Handheld	RSU	Road Side Unit
		TCC	Traffic Control Centre

2.1 Road-Side Subsystem

The road-side subsystem consists of the TCC which is operated by the road operator, receives information from various road-side sensors and from public information sources and controls variable message signs with warnings or variable speed reductions. The TCC sends the information to the CSC, which encodes the information as TPEG documents². The CSC communicates via appropriate servers with the GPRS and DAB networks and via RSUs with IR beacons on gantries above the motorway. The information is sent to the car using wireless communication. On the other hand the car can serve as a moving sensor transmitting FCD to the TCC. For testing purposes, the CSC contains a facility for inserting simulated messages. The GPRS server represents the link into a public mobile communication network. The DAB server keeps the COOPERS messages in a message loop, which is sent typically once per two minutes. New messages are usually inserted at the current end of the loop, except for high priority warnings like wrong-way driver messages.

2.2 In-Car Subsystem

In conjunction with the CGW and the HMI, the APC constitutes the in-vehicle platform for the demonstration of the COOPERS services. It contains the application software for storing, prioritizing and displaying the COOPERS messages on the HMI

² TPEG = Transport Protocol Expert Group, a standard for road telematics messages, for which two variants (TPEG TEC and TPEG RTM) have been elaborated.

in a 7" graphics touch screen mounted on top of the dashboard. Those messages which are relevant for the current position and driving direction are displayed on the display in the car. The information is shown in fixed sections on the display, there is a map window and defined sections for displaying the currently valid speed limit as well as the warnings. The CGW offers communication facilities for an infrared connection and a GPRS module; alternatively a built-in GPRS module in the APC can be used. The DAB receiver is also located in the APC.

3 Requirements for COOPERS

In an early phase of COOPERS, a RAMS analysis was conducted (cf. [2, 3]), which addressed the safety aspects of the project. As one consequence of its results, specific attention had to be dedicated to the HMI; so a specific analysis was performed, see [4]. Exploiting the results of these analyses and also based on the user needs compiled from interviews with all involved stakeholder groups like road operators, authorities and car drivers, the architectural work package documented the requirements for COOPERS. These define and delimit the expected behavior of the cooperative traffic management system with respect to content, user interface layout, prioritization of messages, temporal and local accuracy and tolerance limits for failures. They are classified according to subsystems and service type (depending on criticality), but they are general with respect to wireless technologies. They form the basis for the verification of COOPERS and are stored in the test database mentioned below; for traceability reasons they are linked with the corresponding test cases.

Due to the huge number of requirements it is impossible to cite them all here. Therefore, only a few examples of requirements are given in the following.

Examples for Requirements

1. Service S1a "Accident Warning", communication TCC → OBU (on-board unit): System latency shall be 1 minute average, 5 minutes maximum.
2. Service S1b "Incident warning", OBU: Message display requirements: locality: longitudinal: 30 meters, transversal: lane-specific.
3. Service S2 "Weather condition warning", OBU: Drivers have to be informed at least 2km ahead of a weather hazard scene.

Considering the first example, it becomes evident that a high number of message transfers has to be evaluated in order to obtain the mean value and the upper limit of the latency time with sufficient statistical significance. From the second example results that – assuming a motorway speed of around 30 meters per second – the necessary temporal accuracy must be better than one second. With a similar requirement for S2 follows for the third example that the weather warning must be displayed 1970 meters before the scene at latest. It is clear that in reality the location of a hazardous weather phenomenon can reasonably not be given with 30 meters precision, but the location assigned to the warning message has a resolution which allows for this precision.

4 Verification and Validation of COOPERS

4.1 Term Definitions

Our task was to perform dynamic tests in order to verify the requirements and to validate that the system is adequate for the intended use. The generic functional safety standard ISO/IEC 61508-4 [5] contains the respective definitions:

Verification: Confirmation by examination and provision of objective evidence that the requirements have been fulfilled.

Validation: Confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use are fulfilled.

Dynamic testing: Executing software and/or operating hardware in a controlled and systematic way, so as to demonstrate the presence of the required behaviour and the absence of unwanted behaviour.

In the following, we use the term "test" instead of "dynamic test".

4.2 State of the Art

COOPERS covers a remarkable variety of system aspects; it is composed of partly autonomous, mobile, location-aware, possibly safety-critical embedded subsystems connected by several structurally different wireless communication media with stationary road-side infrastructure subsystems. Literature on testing covers most of these aspects in some way or another. A good overview on testing methods for embedded systems can be found in [6]. Publications on testing methods for mobile distributed systems described e.g. in [7] focus on mobile interactive applications but lack location-awareness. The paper emphasizes the importance of time synchronization on different testing points in the system. There is literature on location-aware services, especially for cellular networks as e.g. in [8], but there is hardly information on field test methods for them. Another paper (see [9]) proposes efficient test cases derived from a formal system specification, which was, however, not available for COOPERS. There is even a specific European white paper [10], giving advice what to test, but it is very high-level and contains no details on how to test. So, in the absence of a proven, detailed field test approach directly applicable for the verification and validation of COOPERS, we had to develop the method described below.

4.3 Goal and Scope

The result of COOPERS is a prototypic demonstrator for cooperative traffic management systems based on an intelligent combination of existing technologies. The goal is to demonstrate that COOPERS works in a satisfactory manner and to evaluate its effects on different levels. Considering this, the goal of the COOPERS tests is to ensure a defined quality of the components, the subsystems and the overall system in order to allow efficient and friction-free demonstration drives at the test sites and to eventually provide evidence for the feasibility of COOPERS technology and applications. So, the test object had the following structure:

1. The in-car subsystem, of which two implementations exist
2. The road-side subsystem, which varies on the single test sites
3. The various wireless communication channels

In-car subsystem and road-side subsystem were verified both in laboratory and in field tests. The component tests were performed by the development team; the work presented here focuses on the field tests, which comprise system and subsystem integration tests as well as application level tests of the various wireless connections.

4.4 Method for Validation and Verification

One result of the RAMS analysis mentioned in section 3 was that COOPERS shall not control the car directly because otherwise the safety lifecycle prescribed by relevant safety standards like the generic functional safety standard ISO/IEC 61508 [11] or the new automotive domain specific norm ISO 26262 [12] would have to be followed, inducing a huge additional effort and therefore hampering future commercial use. As a consequence, we could renounce specific tests of safety requirements and considered instead the aspects described below.

The method for technical verification and validation must be adequate for the system structure as well as the requirements to be verified. For the first aspect we have to start from the fact that COOPERS is a distributed embedded system and as such composed of several subsystems as described in section 2. The second aspect pertains to the requirements described in section 3, i.e. the test bench must be appropriate for measuring them.

The task was to verify the correct implementation of the subsystems and the wireless communication, and to validate the overall system. Moreover it has to be stated that we performed tests with real data (from sensors along the motorway), but also many systematic tests with simulated services injected into the CSC, cf. Fig. 2.

From a technical point of view, COOPERS represents a data path from sensors with evaluation logic via a transmitter and a receiver with prioritization logic to a display. The following diagram illustrates this data path:

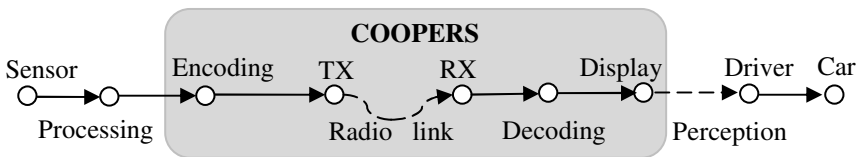


Fig. 3. Signal flow path of the data

The sensors including the evaluation logic are already installed and not specific for COOPERS, they are therefore not considered a test object. Also the driver and the car do not belong to the COOPERS system.

Testing COOPERS on technical level means executing this data transfer while observing correctness with respect to time and location as well as content of the messages. For this purpose, certain points of observation have to be built into the system where the messages together with precise time and accurate location are recorded. For

this purpose, log files are written at several defined locations; they are evaluated and eventually yield the test results. The following subsection describes the COOPERS log file concept, which additionally allows an efficient combined test method.

4.5 Combined Subsystem and System Tests Using Log File Comparison

Fig. 4 shows the log file concept for the different wireless technologies used in COOPERS. As the subsystems support different wireless communication technologies, the log files content is taken from different sources depending on the technology used. With broadcast technology (DAB), GPRS was used in parallel for uploading FCD (Floating Car Data). For this case, the vertical arrows in Fig. 4, which symbolize the data flow, have to be thought with reverse orientation.

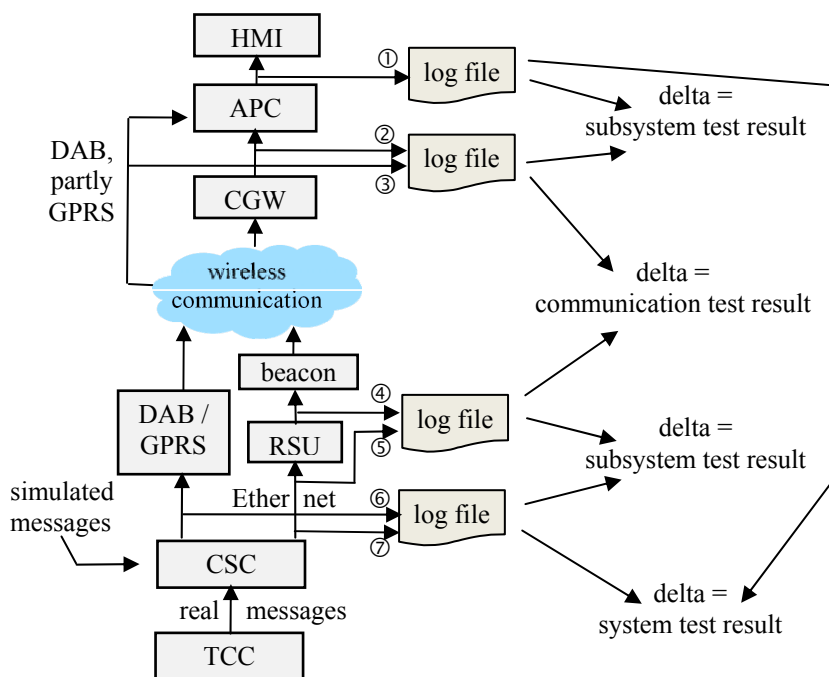


Fig. 4. Combined system and subsystem test using log file comparison

The oblique arrows in the right area of the figure pointing at the "delta" descriptions show that evaluating the different log files and comparing them with one another allows the necessary evaluations for subsystem tests, communication tests as well as overall system tests.

For efficiently performing the tests, we combined subsystem, communication and system tests in the test runs and reduced the tests to system tests for the majority of the test cases, renouncing even evaluations for the subsystem and communication tests. The effort-saving philosophy behind the approach is that if a system test case yields success, then – under a kind of single-fault hypothesis – it is justified to assume

that the single subsystems and the wireless channel in between have performed correctly, too. Only in case a deviation from the requirements is observed in the system test case, a more detailed analysis of the subsystems has to be started.

The following paragraph gives evidence why the above mentioned assumption is valid that subsystems and communication can be considered correct if the related system test case is successful. For analyzing this, we considered the following failures modes:

1. The delay between the CSC and the APC is greater than specified in the requirements
2. The message is sent by the CSC but is not received by the APC
3. The information content is wrong (e.g. incorrect speed value)
4. The incorrect display of messages in respect to the temporal service validity (display of outdated or future messages/no display of currently valid messages)
5. The service is displayed for the opposite direction
6. The display position on the screen is wrong (prioritization error)
7. The display/vanish location in respect to the spatial service validity on the road is wrong (no display/too early/too late)

Regarding the first kind of failure we have to consider that latency times are given as overall system latencies, and delays can only add up; so a subsystems failure can be excluded if the system latency is correct.

For proving the assumption for the other six failure modes listed above, we postulate a subsystem or communication failure while the corresponding system test case yields a correct result. This can only occur if another subsystem or the wireless channel, respectively, presents an exactly compensating misbehaviour in the same test case. It is evident that this is highly improbable; we can assume this with a much higher certainty than the often used but comparably weak single-fault hypothesis. It is rather similar to the probability of undetected double faults in diversitary systems.

Summarizing, through the above described concept we were able to perform system, subsystem and wireless communication tests by executing system test cases with simulated and real test data while recording the log files. These allow for various evaluations like minimum, maximum, median or mean values, and all this referring to single drives, single service types, certain wireless technologies, or a single test site.

After executing each test run, the log files are loaded into data base tables of the test management tool do.ITEM [13], which allows for evaluations formulated as SQL queries. By comparing corresponding entries in the log files tables of the different locations, the transmission of the messages is tracked, using time stamps and location indication for log entries with the same message ID. Additionally, TPEG documents are recorded, which enables a verification of the integrity of the content at the recipient.

4.6 Measured Parameters

The COOPERS tests have to prove sufficient technical functionality of the COOPERS equipment by objective evidence that the system behaviour is within the constraints given by the requirements, cf. section 3. This behaviour is measured in terms of

various parameters which are either identical with or derived from measured technical values. These values are recorded during the technical tests as well as during the subsequent demonstration drives. So, they are essential not only for technical acceptance tests but also for the evaluation of the demonstration drive results.

The set of technical values measured varies depending on the location, i.e. subsystem, where they are captured and stored in log files. The following subsections give an overview on the recorded values and their frequency (in parentheses). The numbers given in curly brackets point to the numbered log file sources (horizontal arrows) in Fig. 4.

HMI {1}

- Display / vanish time of the message and location on the screen (1 Hz)
- Location changes of symbols (1 Hz)

APC {2,3}

- Receive time of the message (at receive time)
- Message details from TPEG document (at receive time)
- Selected CAN data: rotations per minute of the vehicle engine, intake air temperature, vehicle speed (4 – 50 Hz, depending of test site)
- GPS position, heading and vehicle speed (1 Hz)

RSU {4,5}

- Receive time of the message (at receive time)
- Send time of the message (at transmission time)
- Message details from TPEG document (at receive time)

CSC {6,7}

- Send time of the message (at transmission time)
- Message details from TPEG document (at transmission time)

The RSU log files apply only to IR transmission as can be seen in Fig. 4, arrows 4 and 5.

4.7 Prerequisites for Calculating Parameters

In order to compare the log files in different subsystems, the respective parameters must be comparable, i.e. precision with respect to the measured dimension is required. For temporal parameters this implies time synchronization as an important prerequisite.

4.7.1 Time Synchronization and Temporal Precision

The necessary temporal resolution of the requirements is defined by two aspects:

1. The smallest explicit time interval to be evaluated as a requirement, which is 10 seconds delay time for the emergency call,
2. The necessary spatial resolution, which is given by requirements referring to distances of 2000m+/-30m. As mentioned in section 3, at motorway speed, 30m resolution typically corresponds to 1 second.

It follows from both constraints that a temporal precision of 1 second is sufficient for the tests. However, the different subsystems use different time bases.

Most fixed systems of the road side infrastructure use DCF77, which is a clock signal transmitted periodically by a long wave transmitter near Braunschweig in Germany with a precision in the range of several milliseconds in Central Europe (as light speed causes a distance-dependent offset). Certain test sites use an NTP (network time protocol) server to synchronize the COOPERS subsystems; this service, which is available via Internet, allows a precision in the range of milliseconds, too. The in-car subsystem, on the other hand, synchronizes with GPS time, which has a comparable precision.

Summarizing, we may state that the temporal precision of the used time bases was basically to be judged sufficient. An important issue is, however, the synchronization interval, i.e. how often the internal clock is synchronized with the time base. In the course of the tests it turned out that some of the subsystem clocks exposed a temporal deviation of several seconds per day from true time, so it was not sufficient to synchronize only when the system was booted, usually in the morning when the test runs started, but an explicit re-synchronization interval of less than an hour had to be implemented for those subsystems. After this adjustment the time precision met the requirement for performing the tests.

4.7.2 Accuracy of Location Measurement

In COOPERS, two different positioning algorithms are used. One of them is simple GPS positioning, the other one is implemented in the so-called RPU (robust positioning unit).

GPS positioning works correctly only when a sufficient number of satellites is visible and allows an accuracy of roughly 12 m in each direction. This deviation from the true position is not constant with time; as a consequence a stopped car (e.g. in a traffic jam) seems to be slowly moving in different directions within a radius of 12 m around the true location. This is of importance for algorithms deriving the driving direction or simply the fact of a traffic congestion from the GPS position.

The RPU uses – in addition to GPS positioning – a built-in gyroscope and an accelerometer, and it obtains the turning rates of the single wheels and the steering angle from CAN (control area network) bus. With this information the RPU calculates a trajectory and can give a highly accurate position of the vehicle also in case GPS is not available, like for instance in tunnels. Moreover, it can – due to the additional positioning information, which represents redundancy – integrate the GPS information over a longer interval and thereby calculate the position with a higher accuracy.

The spatial accuracy corresponds to the temporal precision. As mentioned above, 30 m is the smallest distance indication in a requirement, which is covered by the temporal accuracy of <1s in conjunction with the GPS location accuracy of 12 m in each direction. So the test environment allows a sufficient longitudinal local accuracy with both GPS and the RPU. For transversal accuracy, tests of the RPU have proven that lane-accurate positioning is possible. But the GPS signal alone with roughly 12 m accuracy is clearly insufficient.

The RPU tests were performed widely independent of the COOPERS service tests by PWP systems with support of University Lodz and are not described further in this paper. A description of RPU simulations can be found in [14].

4.8 Derived Parameters

In the following, two examples of derived parameters, latency time and accuracy of the display/vanish location will be discussed.

4.8.1 Latency Time

The first example for a derived parameter is the latency time between generating a message in the CSC and the moment when the corresponding traffic sign or text shows up on the display in the car. The latency time depends strongly on the wireless technology used and is normally influenced by several contributing factors.

When using GPRS, the latency time is composed of

1. the rate with which the vehicle transmits its position to the CSC
2. the time consumption of the server application for composing the message set applicable to the individual car at its current position
3. the latency caused by the technical limitations of the GPRS network (depending on the network load, the priority of the used SIM card)
4. the processing time in the APC.

In the case of broadcasting technologies (DAB), the messages are stored in a loop of messages for the entire broadcasting region of the transmitter and transmitted one by one. New messages are added to the end of the loop, only high-priority warnings are inserted at the beginning. At the Berlin test site, this loop is repeated typically every two minutes. This effect adds to latency. In the installation used in COOPERS, an additional server stores the COOPERS messages, which are then polled once per minute by the transmitter server with the (continuously running) message loop. So for broadcast transmissions, the latency time is composed of

1. the polling rate of the transmitter server
2. the message loop frequency
3. the processing time in the APC.

For gantry-based point-to-point technologies – in COOPERS infrared transmission – there is an additional stage between the CSC and the car, namely the RSU. There is a comparably low delay caused by the Ethernet-based transmission between the CSC and the RSU, which may be neglected compared to the delay caused by the time between the CSC transmission time and the time when the driver passes the next gantries. The distance between gantries is typically several kilometres, which results in average message delays in the range of several minutes. Nevertheless, the test bench is prepared to observe also the small contribution of the CSC-RSU connection.

4.8.2 Accuracy of the Display/Vanish Location

Another example for a derived parameter is the accuracy of the location where the service is displayed first and where it vanishes from the screen in respect to the specified values in the TPEG document of the message and the distance of the car as specified in the requirements. These values are recorded in the corresponding log files during the test drives and in the laboratory tests – for the latter, of course, partly simulated values apply. The values are extracted from the log files and evaluated in order to derive minimum and maximum, mean and median values. The evaluations serve as a criterion for acceptance of the installation at a particular demonstration site.

5 Results

5.1 Value of the Measurement for COOPERS

The measurement of the values and parameters described above is necessary

- on the one hand for the technical verification and the acceptance tests of the COOPERS equipment and the installation on each demonstration site,
- on the other hand during the demonstration drives and in the subsequent evaluation, where the values and parameters are used to get a clear picture of the situation with which the driver is confronted (e.g. speed or exact location), and finally to investigate the driver reaction to the COOPERS services (like for instance speed reduction, lane changes).

So, the described test bench is beneficial for the system validation with respect to all aspects considered in the project.

5.2 Experience from COOPERS Tests

At the time of the submission of this paper the COOPERS field tests are running; final results are therefore only partly available. Experience tells that our tests usually run through several stages:

1. First integration of the local road-side equipment including the traffic control centre with the in-car equipment aiming to get the whole local equipment running
2. Repeated test runs with detailed log file analyses and software updates containing bug fixes with the goal of finally obtaining a stable version which sufficiently fulfills all applicable requirements
3. After stability has been reached, an acceptance test consisting of three consecutive successful test drives is performed with a detailed assessment on the fulfillment of the requirements
4. Recording of the technical parameters is continued during the demonstration runs in order to allow for exactly correlating the observations by biomedical equipment with the technical parameters of the drive.

6 Conclusion and Further Work

As described in this paper, a powerful test bench for COOPERS has been developed and is being used successfully. It enables detailed analyses of the requirements conformity of the test equipment on each single demonstration site and offers a stable repository for the results for further use in the evaluation of the demonstration outcome.

Currently (June 2010), demonstrations are ongoing. Generally speaking, the fulfillment of the requirements was mostly very good; however, single failures were observed but their frequency was within the tolerance interval stated in the requirements. All these results are evaluated and documented in a project-internal report

including performance statistics, detailed according to demonstration sites and wireless technologies. First results are promising and let us expect successful implementation and use of cooperative traffic management systems like COOPERS on European level in the future.

We plan to make detailed results of the technical COOPERS tests available to the scientific public in a further publication.

References

1. <http://www.coopers-ip.eu>
2. Selhofer, A., Gruber, T.: COOPERS RAMS Analysis. In: Saglietti, F., Oster, N. (eds.) SAFECOMP 2007. LNCS, vol. 4680, pp. 87–92. Springer, Heidelberg (2007)
3. Gruber, T., Althammer, E.: Sicherheitsanforderungen und Validierung eines kooperativen integrierten Verkehrsmanagementsystems. In: Proceedings Informationstagung Mikroelektronik ME 2008, Vienna, pp. 320–326 (2008)
4. Gruber, T., Althammer, E., Schoitsch, E.: Safety Requirements for a Cooperative Traffic Management System: The Human Interface Perspective. In: Buth, B., Rabe, G., Seyfarth, T. (eds.) SAFECOMP 2009. LNCS, vol. 5775, pp. 159–172. Springer, Heidelberg (2009)
5. EN ISO/IEC 61508-4/FDIS, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems – Part 4: Definitions and abbreviations
6. Broekman, B., Notenboom, E.: Testing Embedded Software. Addison-Wesley, London (2003)
7. Lorençot, P., Salva, S.: Testing Mobile and Distributed Systems: Method and Experimentation
8. Küpper, A., Fuchs, F., Schiffers, M., Buchholz, T.: Supporting Proactive Location-Aware Services in Cellular Networks. In: Conti, M., Giordano, S., Gregori, E., Olariu, S. (eds.) PWC 2003. LNCS, vol. 2775, pp. 349–363. Springer, Heidelberg (2003)
9. Chang, S.-J., Hsu, G.-Y., Huang, S.-J.: Location-aware Mobile Transportation Information Service. In: IEE Mobility Conference, IET Conference Publications (2005)
10. White Paper Certification / Validation of Automobile Telematics Components, Systems & Services in Europe, European project "GST Global System for Telematics" number FP6-2002-IST-1-507033
11. EN ISO/IEC 61508, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 1 – Part 7 (1998-2001)
12. ISO/DIS 26262, Road vehicles - Functional safety, Part 1-10 (2009)
13. Web-based test management tool for COOPERS, based on the do. ITEM framework of Dornier Consulting, Germany
14. Pfister, J., Mey, L., Plank-Wiedenbeck, U.: Lane-specific Positioning for COOPERS Services, The Virtual Galileo Approach, ITS World Congress, Stockholm (2009)

100% Coverage for Safety-Critical Software – Efficient Testing by Static Analysis

Daniel Kästner, Reinhold Heckmann, and Christian Ferdinand

AbsInt GmbH, Science Park 1, 66123 Saarbrücken, Germany

Abstract. Safety-critical embedded software is used more and more pervasively in the automotive, avionics and healthcare industries. Failures of such safety-critical embedded systems may cause high costs or even endanger human beings. Also for non-safety-critical applications, a software failure may necessitate expensive updates. Making sure that an application is working properly means addressing many different aspects. Development standards like DO-178B, IEC 61508 and the new revisions DO-178C, or ISO 26262 require to identify potential functional and non-functional hazards and to demonstrate that the software does not violate the relevant safety goals.

For ensuring functional program properties automatic or model-based testing, and formal techniques like model checking become more and more widely used. For non-functional properties identifying a safe end-of-test criterion is a hard problem since failures usually occur in corner cases and full test coverage cannot be achieved. For some non-functional program properties this problem is solved by abstract interpretation-based static analysis techniques which provide full coverage and yield provably correct results. In this article we focus on static analyses of worst-case execution time, stack consumption, and runtime errors, which are increasingly adopted by industry in the validation and certification process for safety-critical software. We explain the underlying methodology and identify criteria for their successful application. The integration of static analyzers in the development process requires interfaces to other development tools, like code generators or scheduling tools. Using them for certification requires an appropriate tool qualification. We will address each of these topics and report on industrial experience.

1 Introduction

The use of safety-critical embedded software in the automotive, avionics and healthcare industries is increasing rapidly. Failures of such safety-critical embedded systems may create high costs or even endanger human beings. Also for non-safety-critical applications, a software failure may necessitate expensive updates. Therefore, utmost carefulness and state-of-the-art techniques for verifying software safety requirements have to be applied to make sure that an application is working properly. To do so lies in the responsibility of the system designers.

Standards like DO-178B, IEC 61508 and the new revisions DO-178C, or ISO 26262 increasingly emphasize ensuring software safety. They require to identify functional and non-functional hazards and to demonstrate that the software

does not violate the relevant safety goals. Examples for non-functional safety-relevant software characteristics relevant for DO-178B and IEC 26262 WD are runtime errors, execution time and memory consumption. Depending on the criticality level of the software the absence of safety hazards has to be demonstrated by formal methods or testing with sufficient coverage.

Classical software validation methods like code review and testing with debugging cannot really guarantee the absence of errors. Formal verification methods provide an alternative, in particular for safety-critical applications. One such method is *abstract interpretation* [4], which allows to obtain statements that are valid for all program runs with all inputs. Such statements may be absence of violations of timing or space constraints, or absence of runtime errors. Static analysis tools are in industrial use that can detect stack overflows, violation of timing constraints [23], and can prove the absence of runtime errors [6].

The advantage of static analysis based techniques is that they enable full test coverage, but at the same time can reduce the test effort. Identifying end-of-test criteria for non-functional program properties like timing, stack size, and runtime errors is an unsolved problem. In consequence the required test effort is high, the tests require access to the physical hardware and the results are not complete. In contrast, static analyses can be run by software developers from their workstation computer, they can be integrated in the development process, e.g., in model-based code generators, and allow developers to detect timing and space bugs in early product stages.

From a methodological point of view, static analyses can be seen as equivalent to testing with full coverage and, as such, are candidates for meeting all testing requirements listed in the above-mentioned standards. Thus, in the areas where validation by static analysis is technically feasible and applied in industry, e.g., for worst-case execution time analysis, stack size analysis or runtime error analysis, it defines the state-of-the-art testing technology.

This article is structured as follows: First we explain the basic methodology of static analysis and then present the underlying concepts of tools from three different application areas: aiT for worst-case execution time analysis, Stack-Analyzer for stack usage analysis and Astrée for runtime error analysis. In Sec. 5 we describe tool couplings and qualification support kits which allow a seamless integration into the development process and improve test efficiency. Industrial experience is summarized in Sec. 6 and Sec. 7 concludes.

2 Validation of Safety Properties

Static data flow analyses compute invariants for all program points by fixed point iteration over the program structure or the control-flow graph. The theory of abstract interpretation [4] offers a semantics-based methodology for static program analyses. The concrete semantics is mapped to an abstract semantics by abstraction functions. While most interesting program properties are undecidable in the concrete semantics, the abstract semantics can be chosen for them to be computable. The static analysis is computed with respect to that abstract

semantics. Compared to an analysis of the concrete semantics, the analysis result may be less precise but the computation may be significantly faster. By skillful definition of the abstract domains a suitable trade-off between precision and efficiency can be attained.

For program validation there are two essential properties of static analyzers: soundness and safety. A static analysis is called *sound* if the computed results hold for any possible program execution. Abstract interpretation supports formal correctness proofs: it can be proven that an analysis will terminate and that it is sound, i.e. that it computes an overapproximation of the concrete semantics. In a *safe* static analysis imprecision can occur, but it must only occur on the safe side.

Let's illustrate this with two application scenarios: In runtime error analysis, soundness means that the analyzer never omits to signal an error that can appear in some execution environment. If no potential error is signalled, definitely no runtime error can occur, i.e. there are *no false negatives*. If a potential error is reported, the analyzer cannot exclude that there is a concrete program execution triggering the error. If there is no such execution, this is a false alarm (*false positive*). This imprecision is on the safe side: it can never happen that there is a runtime error which is not reported. In WCET analysis, soundness means that the computed WCET bound holds for any possible program execution. Safety means that the only imprecision occurring is overestimation: the WCET must never be underestimated.

3 Static WCET and Stack Analysis

In this section we will illustrate the principles of static program analyzers at the binary level with the examples of aiT WCET analyzer [25] and StackAnalyzer [10]. A comprehensive survey of methods and tools for determining worst-case execution time can be found in [27].

3.1 Value Analysis

An essential component of WCET and stack analysis is the so-called *value analysis*. At the binary level value analysis tries to determine the values stored in the processor's memory for every program point and execution context. Often, restricting value analysis to the processor registers suffices, but sometimes, it is useful to get information about main memory as well.

Value analysis determines for each program point and execution context an *abstract state* that maps memory locations to abstract values. Each machine instruction is modeled by a transfer function mapping input states to output states in a way that is compatible with the semantics of the instruction. At control-flow joins, the incoming abstract states are combined into a single outgoing state using a combination function. Because of the presence of loops, transfer and combination functions must be applied repeatedly until the system of abstract states stabilizes. According to the theory of Abstract Interpretation, termination of

this fixed-point iteration is ensured by the monotonicity of transfer and combination functions and the fact that a memory location can only hold finitely many different values.

Depending on the *abstract domain*, i.e. on what kinds of abstract values are used, there are several variants of value analysis. The simplest form of value analysis is *constant propagation*: it computes whether the result of a statement is a unique constant value. An abstract value is either a single concrete value, i.e. the constant, or the value *unknown*. A more elaborate form of value analysis, the *interval analysis* computes safe lower and upper bounds for the possible concrete values, i.e. abstract values are intervals that are guaranteed to contain the exact values.

Value analysis can also determine that certain conditions always evaluate to true or always evaluate to false. As consequence, certain paths controlled by such conditions are never executed. Therefore, their execution time or stack usage does not contribute to the overall maximum, and need not be determined in the first place.

3.2 Stack Usage Analysis

A possible cause of catastrophic failure is a stack overflow which might cause the program to behave in a wrong way or to crash altogether. When they occur stack overflows can be hard to diagnose and hard to reproduce. The problem is that the memory area for the stack usually must be reserved by the programmer. Underestimation of the maximum stack usage leads to stack overflow, while overestimation means wasting memory resources. Measuring the maximum stack usage with a debugger is no solution since one only obtains a result for a single program run with fixed input. Even repeated measurements with various inputs cannot guarantee that the maximum stack usage is ever observed.

By concentrating on the value of the stack pointer during value analysis, StackAnalyzer computes how the stack increases and decreases along the various control-flow paths. This information can be used to derive the maximum stack usage of the entire task. The results of StackAnalyzer are graphically presented as annotations in a combined call graph and control-flow graph.

StackAnalyzer provides automatic tool support to calculate precise information on the stack usage. It shows the critical path, i.e. the path on which the maximum stack usage is reached which gives important feedback for optimizing the stack usage of the application under analysis. StackAnalyzer is available for many variants of HC11, HCS12/Star12(X), Freescale PowerPC, Infineon C16x/ST10/XC16x, Texas Instruments C3x, ARM7-9, NEC V850, LEON2, LEON3, Motorola M68k, Infineon TriCore and ARC.

3.3 WCET Analysis: Worst-Case Execution Time Prediction

Many tasks in safety-critical embedded systems have hard real-time characteristics. Failure to meet deadlines may be as harmful as producing wrong output or failure to work at all. Yet the determination of the Worst-Case Execution Time (WCET) of a task is a difficult problem because of the characteristics of modern

software and hardware [26]. Embedded control software (e.g., in the automotive industries) tends to be large and complex. The software in a single electronic control unit typically has to provide different kinds of functionality. It is usually developed by several people, several groups or even several different providers. Code generator tools are widely used. They usually hide implementation details to the developers and make an understanding of the timing behavior of the code more difficult. The code is typically combined with third party software such as real-time operating systems and/or communication libraries. Concerning hardware, there is typically a large gap between the cycle times of modern microprocessors and the access times of main memory. Caches and branch target buffers are used to overcome this gap in virtually all performance-oriented processors (including high-performance micro-controllers and DSPs). Pipelines enable acceleration by overlapping the executions of different instructions. Consequently the execution behavior of the instructions cannot be analyzed separately since it depends on the execution history. Cache memories usually work very well, but under some circumstances minimal changes in the program code or program input may lead to dramatic changes in cache behavior. For (hard) real-time systems, this is undesirable and possibly even hazardous. Making the safe yet – for the most part – unrealistic assumption that all memory references lead to cache misses results in the execution time being overestimated by several hundred percent.

The widely used classical methods of predicting execution times are not generally applicable. Software monitoring and dual-loop benchmark modify the code, which in turn changes the cache behavior. Hardware simulation, emulation, or direct measurement with logic analyzers can only determine the execution time for some fixed inputs. They cannot be used to infer the execution times for all possible inputs in general.

In contrast, abstract interpretation can be used to efficiently compute a safe approximation for all possible cache and pipeline states that can occur at a program point in any program run with any input. These results can be combined with ILP (Integer Linear Programming) techniques to safely predict the worst-case execution time and a corresponding worst-case execution path.

AbsInt's WCET tool aiT is available for various microcontrollers with the following cores: ARM7, Infineon C16x/ST10, Texas Instruments C33, Motorola HC11, HCS12/STAR12, Intel i386, i486, LEON2, LEON3, Motorola M68020, Freescale MPC 5xx, MPC603e, MPC55xx, MPC755, Infineon TriCore, NEC V850. It determines the WCET of a program task in several phases [11], which makes it possible to use different methods tailored to each subtasks [24]. First, the control-flow graph (CFG) is reconstructed from the input file, the binary executable. Then value analysis computes value ranges for registers and address ranges for instructions accessing memory; a loop bound analysis determines upper bounds for the number of iterations of simple loops. Subsequently, a cache analysis classifies memory references as cache misses or hits [9] and a pipeline analysis predicts the behavior of the program on the processor pipeline [17]. Finally the path analysis determines a worst-case execution path of the program [24].

Cache analysis classifies the accesses to main memory. The analysis in our tool is based upon [9], which handles analysis of caches with LRU (Least Recently Used) replacement strategy like in the Motorola MPC603e core. However, it had to be modified to reflect the non-LRU replacement strategies of other common microprocessors: the pseudo-round-robin replacement policy of the ColdFire MCF 5307 or the Freescale MPC55xx, and the PLRU (Pseudo-LRU) strategy of the PowerPC 750, 755, or 7448. The modified algorithms distinguish between sure cache hits and unclassified accesses. The deviation from perfect LRU is the reason for the reduced predictability of the cache contents in case of ColdFire 5307 and PowerPC 750/755/7448 compared to processors with perfect LRU caches [14].

Pipeline analysis models the pipeline behavior to determine execution times for sequential flows (basic blocks) of instructions as done in [22]. It takes into account the current pipeline state(s), in particular resource occupancies, contents of prefetch queues, grouping of instructions, and classification of memory references by cache analysis. The result is an execution time for each basic block in each distinguished execution context. Like value and cache analysis, pipeline analysis is based on the framework of abstract interpretation. Pipeline analysis of a basic block starts with a set of pipeline states determined by the predecessors of the block and lets this set evolve from instruction to instruction by a kind of cycle-wise simulation of machine instructions. In contrast to a real simulation, the abstract execution on the instruction level is in general non-deterministic since information determining the evolution of the execution state is missing, e.g., due to non-predictable cache contents. Therefore, the abstract execution of an instruction may cause a state to split into several successor states. All the states computed in such tree-like structures form the set of entry states for the successor instruction. At the end of the basic block, the final set of states is propagated to the successor blocks. The described evolution of state sets is repeated for all basic blocks until it stabilizes, i.e. the state sets do not change any more. The output of pipeline analysis is the number of cycles a basic block takes to execute, for each context, obtained by taking the upper bound of the number of simulation cycles for the sequence of instructions for this basic block. These results are then fed into path analysis to obtain the WCET for the entire task.

aiT's results are written into a report file. In addition, aiT produces a visualization which enables users to interactively explore the analysis results. Figure 1, left, shows the graphical representation of the call graph for some small example. The calls (edges) that contribute to the worst-case runtime are marked by the color red. The computed WCET is given in CPU cycles and in microseconds provided that the cycle time of the processor has been specified.

Figure 1, right, shows the basic block graph of a loop. The number `sum #` is the number of traversals of an edge in the worst case, while `max t` describes the maximal execution time of the basic block from which the edge originates (taking into account that the basic block is left via the edge). The worst-case path, the iteration numbers and timings are determined automatically by aiT.

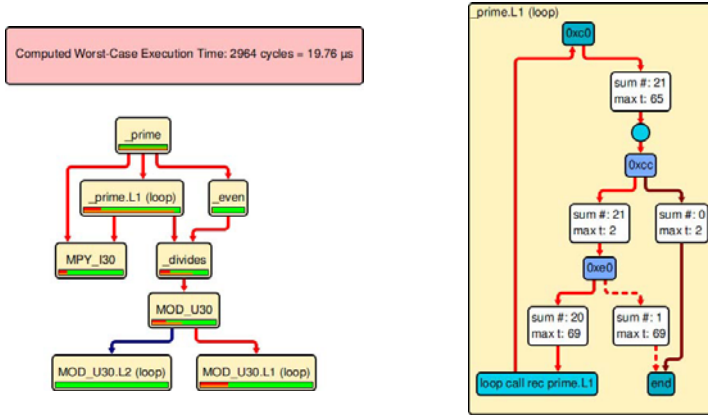


Fig. 1. Call graph and basic-block graph with WCET results

Upon special command, aiT provides information on the origin of these timings by displaying the cache and pipeline states that may occur within a basic block.

4 Proving the Absence of Runtime Errors

Another important goal when developing critical software is to prove that no runtime errors can occur. Examples for runtime errors are floating-point overflows, array bound violations, division by zero, or invalid pointer accesses. A well-known example for the possible effects of runtime errors is the explosion of the Ariane 5 rocket in 1996 [18]. As detailed above, software testing can be used to detect errors, but not to prove their absence. The success of static analysis is based on the fact that safe overapproximations of program semantics can be computed. For runtime error analysis this means that the analysis result for a statement x will be either “(i) statement x will not cause an error”, or “(ii) statement x may cause an error”. In the first case, the user can rely on the absence of errors, in the second case either an error has been found, or there was a false alarm. This imprecision allows to compute results in acceptable time, even for large software projects.

Each alarm has to be manually investigated to determine whether there is an error which has to be corrected, or whether it was just a false alarm. If all the alarms raised by an analysis have been proven to be false, then the proof of absence of runtime errors is completed. This could be checked manually, but the problem is that such a human analysis is error-prone and time consuming, especially since there might be interdependencies between the false alarms and in some cases deviations from the C standard may be willingly accepted. Therefore the number of alarms should be reduced to zero, since then the absence of runtime errors is automatically proven by the analyzer run.

To that end, it is important that the analyzer is precise, i.e. produces only few false alarms. This can only be achieved by a tool that can be “specialized”

to a class of properties for a family of programs. Additionally the analyzer must be parametric enough for the user to be able to fine-tune the analysis of any particular program of the family. General software tools not amenable to specialization usually report a large number of false alarms which is the reason why such tools are only used in industry to detect runtime errors, and not to prove their absence.

Additionally the analyzer must provide flexible annotation mechanisms to communicate external knowledge to the analyzer. Only by a combination of high analyzer precision and support for semantic annotations the goal of *zero false alarms* can be achieved. A prerequisite is that users get enough information to understand the cause of an alarm so that they can either fix the bugs or supply the missing semantic information.

Astrée (*Analyseur statique de logiciels temps-réel embarqués*) [1] has been specifically designed to meet these requirements: it produces only a small number of false alarms for control/command programs written in C according to “ISO/IEC 9899:1999 (E)” (C99 standard) [3]. And it provides the user with enough options and directives to help reduce this number significantly. Thus, in contrast to many other static analyzers Astrée cannot only be used to detect runtime errors, but to actually prove their absence. A detailed comparison of Astrée with other code analyzers can be found in [5]; in the following we will only summarize the most important aspects.

Astrée [2] can be adapted to the software project under analysis in order to improve analysis precision. The key feature here is that Astrée is fully parametric with respect to the abstract domains. There is a variety of predefined abstract domains, including the following ones: The interval domain approximates variable values by intervals. The octagon domain [21] covers relations of the form $x \pm y \leq c$ for variables x and y and constants c . Floating-point computations are precisely modelled while keeping track of possible rounding errors [20]. The memory domain empowers Astrée to exactly analyze pointer arithmetics and union manipulations. The clock domain has been specifically developed for synchronous control programs and supports relating variable values to the system clock. With the filter domain [13] digital filters can be precisely approximated. Based on inspecting reported false alarms the abstract domains can be stepwise refined for a given program class. It is also possible to incorporate additional abstract domains into Astrée.

In a further step, there are two mechanisms for adapting Astrée to individual programs from a program family. First, abstract domains can be parameterized to tune the precision of the analysis for individual program constructs or program points [19]. On the other hand there are annotations for making external information available to Astrée. As current experience shows the parameterization of the programs under analysis rarely has to be changed when the analyzed software evolves over time. So in contrast e.g., to theorem provers the parameterization is very stable.

5 Integration in the Development Process

Worst-case execution time analysis and stack usage analysis can be integrated seamlessly into the development process and are not only applicable at the validation stage but also at the development stage. One advantage of static analysis methods is that no testing on physical hardware is required. Thus the analyses can be called just like a compiler from a workstation computer after the linking stage of the project. As an example, aiT and StackAnalyzer feature batch versions which facilitate the integration in a general automated build process. This enables developers to instantly assess the effects of program changes on WCET and stack usage.

5.1 Tool Qualification

In the *validation stage* the goal is to verify that the stack limits or worst-case execution time bounds of the application are not exceeded and that no runtime errors may occur. To be amenable for certification according to DO-178B, analysis tools have to be qualified. The qualification process can be automated to a large degree by *Qualification Support Kits*. The qualification kits for aiT, StackAnalyzer and Astrée consist of a report package and a test package. The report package lists all functional requirements and contains a verification test plan describing one or more test cases to check each functional requirement. The test package contains an extensible set of test cases and a scripting system to automatically execute all test cases and evaluate the results. The generated reports can be submitted to the certification authority as part of the DO-178B certification package.

5.2 Integration with Model-Based Code Generators

Solving timing problems may require costly re-designs, so the later in the design cycle timing problems become apparent the higher will be the cost for repairing them. Therefore it is highly desirable to incorporate static code-level timing and stack usage analysis into the model-based design process in order to prevent such late-stage integration problems. Since they do not require direct access to the physical hardware, static analysis methods are well suited for such an integration. The viability of this approach could be established in the STReP projects INTEREST/INTERESTED by coupling aiT and StackAnalyzer with the model-based design tools SCADE from Esterel [7] and ETAS ASCET [8]. In the case of the SCADE coupling, aiT and StackAnalyzer are fully integrated in the SCADE GUI (cf. Fig. 2) as SCADE Suite Timing/Stack Verifier. The SCADE Suite Code Generator generates C code from a SCADE model, together with a traceability file that provides the detailed relation from SCADE operators to generated C functions. This code is compiled and linked with any user integration code to get an executable. Then aiT computes the WCET analysis on the executable and produces results for the routines in the executable, which correspond to C functions. The traceability information produced from the SCADE Suite Code

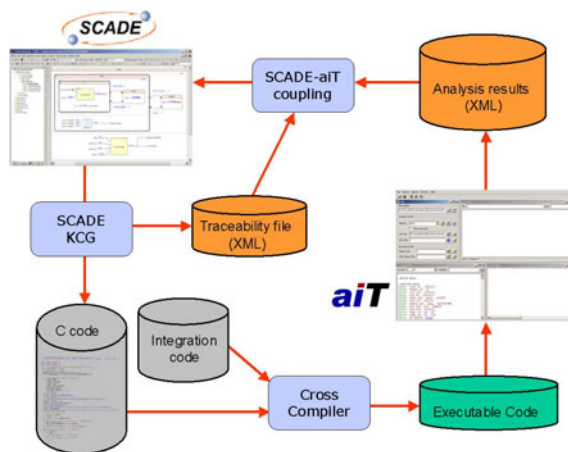


Fig. 2. Overview of the tool coupling between aiT and SCADE

Generator is used together with the result from the timing analysis for feeding back WCET results to the SCADE model.

The generated code is automatically analyzed; after a successful analysis, a code-level report is generated, both in graphical form and as an XML file. WCET and stack results are mapped to the modelling level and are displayed for each SCADE operator. Thus developers can see the timing and stack effect for the model or any parts of it without having to inspect the generated C code. The effect of changes to the SCADE design and/or the code generation options on WCET and stack consumption become instantly apparent [12]. Experiments conducted during the project INTEREST indicate that by the aiT/StackAnalyzer coupling more than 60% of the validation effort can be saved, compared to standard test-based methods.

The same considerations are also true for a coupling between model-based code generators and static runtime analysis tools. A tool coupling between Esterel SCADE and Astrée has been started in the context of the European ES_PASS project.

5.3 Integration with System-Level Analysis Tools

For complex multi-tasking systems system-level schedulability analysis has to be performed. An example of a tool for such a system-level analysis is SymTA/S [15]: it performs scheduling analysis and optimization for controllers, networks and entire systems. SymTA/S computes the worst-case response times (WCRTs) of tasks and the worst-case end-to-end communication delays. It takes into account the worst-case execution times of the tasks as well as information about RTOS scheduling, bus arbitration, possible interrupts and their priorities. Obviously, for computing timing and stack information at the system level it is necessary to know the worst-case execution time and the maximal stack usage at the task

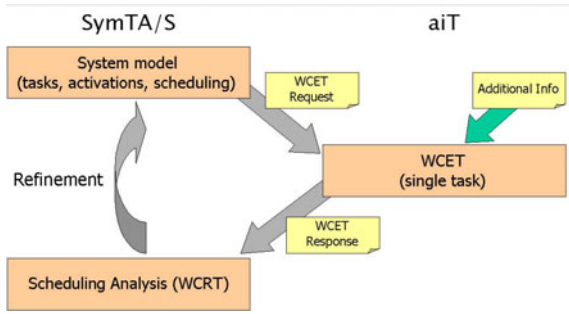


Fig. 3. Overview of the tool coupling between aiT and SymTA/S

level. So for a seamless approach to system-level timing a coupling between the system level and the code level is required. Between SymTA/S and aiT resp. StackAnalyzer [16] such a tool coupling has been established [16]. An overview of the tool coupling is given in Fig. 3.

6 Experience

In recent years tools based on static analysis have proven their usability in industrial practice and, in consequence, have increasingly been used by avionics, automotive and healthcare industries. In the following we report some experiences gained with aiT WCET Analyzer, StackAnalyzer and Astrée.

6.1 WCET and Stack Analysis

StackAnalyzer results are usually precise for any given program path and can be computed faster than the results of aiT. Thus for the remainder of this section we will concentrate on experience with aiT. Since the real WCET is usually unknown for typical real-life applications, statements about the precision of aiT are hard to obtain. For an automotive application running on MPC 555, one of AbsInt's customers has observed an overestimation of 5–10 % when comparing aiT's results and the highest execution times observed in a series of measurements (which may have missed the real WCET). For an avionics application running on MPC 755, Airbus has noted that aiT's WCET for a task typically is about 25 % higher than some measured execution times for the same task, the real but non-calculable WCET being in between [23]. Measurements at AbsInt have indicated overestimations ranging from 0 % (cycle-exact prediction) till 10 % for a set of small programs running on M32C, TMS320C33, and C166/ST10. Table 1 shows the results for C166. The analysis times were moderate – a few seconds till about 3 minutes for EDN.

6.2 Runtime Error Analysis

Astrée has been used in several industrial avionics and space projects. One of the examined software projects from the avionics industry comprises 132,000 lines

Table 1. Precision of aiT for some C166 programs

Example		from external RAM			from flash		
Program	Size	measured cycles	predicted cycles	over-estimation	measured cycles	predicted cycles	over-estimation
fac	2.9k	949	960	1.16 %	810	832	2.72 %
fibo	3.4k	2368	2498	5.49 %	2142	2228	4.01 %
coverc1	16k	5670	5672	0.04 %	3866	4104	6.16 %
coverc	4.3k	7279	7281	0.03 %	5820	6202	6.56 %
morswi	5.9k	17327	17332	0.03 %	8338	8350	0.14 %
coverc2	24k	18031	18178	0.82 %	12948	14054	8.54 %
swi	24k	18142	18272	0.72 %	13330	14640	9.83 %
edn	13k	262999	267643	1.77 %	239662	241864	0.92 %

of C code including macros and contains approximately 10,000 global and static variables [2]. The first run of Astrée reported 1200 false alarms; after adapting Astrée the number of false alarms could be reduced to 11. The analysis duration was 1h 50 min on a PC with 2.4 GHz and 1 GB RAM.

The report [6] gives a detailed overview of the analysis process for an Airbus avionics project. The software project consists of 200,000 lines of preprocessed C code, performs many floating-point computations and contains digital filters. The analysis duration for the entire program is approximately 6 hours on a 2.6 GHz PC with 16 GB RAM. At the beginning, the number of false alarms was 467 and could be reduced to zero in the end.

7 Conclusion

The quality assurance process for safety-critical embedded software is of crucial importance. The cost for system validation grows with increasing criticality level to constitute a large fraction of the overall development cost. The problem is twofold: system safety must be ensured, yet this must be accomplishable with reasonable effort.

Tools based on abstract interpretation can perform static program analysis of embedded applications. Their results are determined without the need to change the code and hold for all program runs with arbitrary inputs. Especially for non-functional program properties they are highly attractive, since they provide full coverage and can be seamlessly integrated in the development process.

We have presented three exemplary tools in this article: aiT allows to inspect the timing behavior of (time-critical parts of) program tasks. It takes into account the combination of all the different hardware characteristics while still obtaining tight upper bounds for the WCET of a given program in reasonable time. StackAnalyzer calculates safe upper bounds on the maximum stack usage of tasks. To give an application example StackAnalyzer and aiT are used by Airbus in the development of various safety-critical applications for the A380. Astrée can be used to prove the absence of runtime errors in C programs. It can be specialized to the software under analysis and achieves very high precision.

Industrial synchronous real-time software from the avionics industry could be successfully analyzed by Astrée with zero false alarms.

aiT, StackAnalyzer and Astrée can be used as analysis tools for the certification according to development standards like DO-178B or ISO 26262. The tool qualification process can be automatized to a large extend by dedicated Qualification Support Kits. They are used by many industry customers from avionics and automotive industries and have been proven in industrial practise.

Acknowledgement

The work presented in this paper has been supported by the European FP6 project INTEREST, the European FP7 project INTERESTED, and the ITEA2 project ES.PASS.

References

1. AbsInt Angewandte Informatik GmbH. Astrée Website, <http://www.astree.de>
2. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A Static Analyzer for Large Safety-Critical Software. In: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI 2003), San Diego, California, USA, June 7-14 2003, pp. 196–207. ACM Press, New York (2003)
3. JTC1/SC22. Programming languages – C (December 16, 1999)
4. 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 ACM Symposium on Principles of Programming Languages, Los Angeles, California, pp. 238–252 (1977)
5. Cousot, P., Cousot, R., Feret, J., Miné, A., Mauborgne, L., Monniaux, D., Rival, X.: Varieties of Static Analyzers: A Comparison with ASTRÉE. In: First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering, TASE 2007, pp. 3–20. IEEE Computer Society, Los Alamitos (2007)
6. Delmas, D., Souyris, J.: ASTRÉE: from Research to Industry. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 437–451. Springer, Heidelberg (2007)
7. Esterel Technologies. SCADE Suite, <http://www.esterel-technologies.com/products/scade-suite>
8. ETAS Group. ASCET Software Products, http://www.etas.com/en/products/ascet_software_products.php
9. Ferdinand, C.: Cache Behavior Prediction for Real-Time Systems. PhD thesis, Saarland University (1997)
10. Ferdinand, C., Heckmann, R., Kästner, D.: Static Memory and Timing Analysis of Embedded Systems Code. In: Proceedings of the IET Conference on Embedded Systems at Embedded Systems Show (ESS), Birmingham (2006)
11. Ferdinand, C., Heckmann, R., Langenbach, M., Martin, F., Schmidt, M., Theiling, H., Thesing, S., Wilhelm, R.: Reliable and precise WCET determination for a real-life processor. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 469–485. Springer, Heidelberg (2001)
12. Ferdinand, C., Heckmann, R., Le Sergeant, T., Lopes, D., Martin, B., Fornari, X., Martin, F.: Combining a high-level design tool for safety-critical systems with a tool for WCET analysis on executables. In: 4th European Congress ERTS Embedded Real Time Software, Toulouse, France (January 2008)

13. Feret, J.: Static Analysis of Digital Filters. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 33–48. Springer, Heidelberg (2004)
14. Heckmann, R., Langenbach, M., Thesing, S., Wilhelm, R.: The influence of processor architecture on the design and the results of WCET tools. *Proceedings of the IEEE* 91(7), 1038–1054 (2003), Special Issue on Real-Time Systems
15. Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., Ernst, R.: System level performance analysis – the SymTA/S approach. In: *IEEE Proceedings on Computers and Digital Techniques*, vol. 152(2) (March 2005)
16. Kästner, D., Wilhelm, R., Heckmann, R., Schlickling, M., Pister, M., Jersak, M., Richter, K., Ferdinand, C.: Timing Validation of Automotive Software. In: 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA) 2008. CCIS, vol. 17, pp. 93–107. Springer, Heidelberg (2008)
17. Langenbach, M., Thesing, S., Heckmann, R.: Pipeline modeling for timing analysis. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 294–309. Springer, Heidelberg (2002)
18. Lions, J., et al.: ARIANE 5, Flight 501 Failure. Report by the Inquiry (1996)
19. Mauborgne, L., Rival, X.: Trace Partitioning in Abstract Interpretation Based Static Analyzers. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 5–20. Springer, Heidelberg (2005)
20. Miné, A.: Relational Abstract Domains for the Detection of Floating-Point Run-Time Errors. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 3–17. Springer, Heidelberg (2004)
21. Miné, A.: The Octagon Abstract Domain. *Higher-Order and Symbolic Computation* 19(1), 31–100 (2006)
22. Schneider, J., Ferdinand, C.: Pipeline behavior prediction for superscalar processors by abstract interpretation. In: *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*, pp. 35–44 (May 1999)
23. Souyris, J., Le Pavec, E., Himbert, G., Jégu, V., Borios, G., Heckmann, R.: Computing the worst case execution time of an avionics program by abstract interpretation. In: *Proceedings of the 5th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*, pp. 21–24 (2005)
24. Theiling, H., Ferdinand, C.: Combining abstract interpretation and ILP for microarchitecture modelling and program path analysis. In: *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, pp. 144–153 (December 1998)
25. Thesing, S., Souyris, J., Heckmann, R., Randimbivololona, F., Langenbach, M., Wilhelm, R., Ferdinand, C.: An abstract interpretation-based timing validation of hard real-time avionics software. In: *Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN 2003)*, pp. 625–632. IEEE Computer Society, Los Alamitos (2003)
26. Wilhelm, R.: Determining bounds on execution times. In: Zurawski, R. (ed.) *Handbook on Embedded Systems*, pp. 14-1 –14-23. CRC Press, Boca Raton (2005)
27. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems* 7(3), 1–53 (2008)

MODIFI: A MODEL-Implemented Fault Injection Tool

Rickard Svenningsson¹, Jonny Vinter¹, Henrik Eriksson¹, and Martin Törngren²

¹ Department of Electronics, SP Technical Research Institute of Sweden
{rickard.svenningsson,jonny.vinter,henrik.eriksson}@sp.se

² Department of Mechatronics, KTH Royal Institute of Technology
martin@md.kth.se

Abstract. Fault injection is traditionally divided into *simulation-based* and *physical* techniques depending on whether faults are injected into hardware models, or into an actual physical system or prototype. Another classification is based on *how* fault injection mechanisms are implemented. Well known techniques are *hardware-implemented fault injection* (HIFI) and *software-implemented fault injection* (SWIFI). For safety analyses during model-based development, fault injection mechanisms can be added directly into models of hardware, models of software or models of systems. This approach is denoted by the authors as *model-implemented fault injection*. This paper presents the MODIFI (MODEL-Implemented Fault Injection) tool. The tool is currently targeting behaviour models in Simulink. Fault models used by MODIFI are defined using XML according to a specific schema file and the fault injection algorithm uses the concept of minimal cut sets (MCS) generation. First, a user defined set of single faults are injected to see if the system is tolerant against single faults. Single faults leading to a failure, i.e. a safety requirement violation, are stored in a MCS list together with the corresponding counterexample. These faults are also removed from the fault space used for subsequent experiments. When all single faults have been injected, the effects of multiple faults are investigated, i.e. two or more faults are introduced at the same time. The complete list of MCS is finally used to automatically generate test cases for efficient fault injection on the target system.

1 Introduction

As fault injection (also known as fault insertion testing) has become widely used as an experimental dependability validation method, many different techniques for injecting faults have been developed. Fault injection accelerates the occurrences of faults in a system and the main purpose is to evaluate and debug error handling mechanisms. It is used at various abstraction levels and phases of the development process. Fault injection is e.g. mandatory in safety standard IEC 61508 and recommended or highly recommended in the automotive standard ISO DIS 26262, when the claimed diagnosis coverage is at least 90%.

Fault injection is traditionally used for emulating hardware faults, where different techniques normally are divided into *simulation-based* and *physical* techniques

depending on whether faults are injected into hardware models (e.g. VHDL models), or into an actual physical system or prototype. To avoid focusing only on the target for fault injection, the classification presented in the survey, in Section 2, is instead based on *how* fault injection mechanisms are implemented.

Well known approaches are *hardware-implemented fault injection (HIFI)* and *software-implemented fault injection (SWIFI)*. For safety analyses during model-based development which are increasingly being used during system development, fault injection mechanisms can be added directly into models of hardware, models of software or models of systems. Thus, in the same manner as for HIFI and SWIFI, we define this approach as *model-implemented fault injection (MIFI)*.

Using MIFI is attractive since design deficiencies which are found early on in the development are less costly to correct. Different versions of a model can be compared with respect to a specific faultload and vulnerable parts of the models can be found. These vulnerable parts can be protected with new fault handling mechanisms which can be exercised and evaluated using additional MIFI experiments. Doing system design in Simulink [1] and to generate code directly from the system models is also becoming more and more widespread. Thus, fault handling mechanisms added to the model will be included in the generated software and compiled and downloaded into the target system. These are the main driving forces behind the development of the MODIFI tool.

The remainder of the paper is organized as follows. Section 2 briefly describes techniques and aspects of fault injection in general, whereas Section 3 is dedicated to the description of the MODIFI tool. Finally, the conclusions are given in Section 4.

2 Related Work

This Section presents fault injection properties, different approaches for implementing fault injection mechanisms and fault injection tools presented in the literature. The approaches for fault injection presented in this survey are for simplicity divided into only three groups based on *how* fault injection mechanisms are implemented.

2.1 Fault Injection Properties

The various fault injection techniques can be characterized according to different properties. One such property is *reachability*, expressing the ability of the fault injection technique to reach possible fault locations in the system. Another property is *controllability*, with respect to *space* and *time*, denoting the ability to control *where* and *when* the faults are injected among the reachable locations. *Repeatability* denotes the ability to accurately repeat a single fault injection experiment while *reproducibility* refers to the ability to statistically reproduce the results of several experiments for a given set-up. *Intrusiveness* relates to the level of undesired impact the fault injection technique may have on the behavior of the target system and can be divided into *space* and *time* properties. In order to achieve experiments corresponding to faults in the real world, it is important that the intrusion is low. Intrusiveness in time relates to the temporal overhead caused by the fault injection technique while intrusiveness in space relates to the hardware/software overhead. Other properties include *flexibility*,

denoting the ease of changing fault injection targets in the system, *effectiveness* with respect to the ability to activate and exercise various fault handling mechanisms in the system and *efficiency* with respect to the amount of time and effort needed to conduct the experiments. Another important property is *observability*, which refers to the ability to provide means for observing and measuring the effects of faults in the system.

2.2 Hardware-Implemented Fault Injection

Hardware-implemented fault injection techniques (also referred to as “Hardware-based”) are applied on actual implementations or prototypes of systems during later phases in the development process. While the observability and controllability can be limited the efficiency is often high. Another advantage is that the actual implementation of the system is validated instead of a system model. One of the most common hardware-implemented techniques is *pin-level fault injection*. With pin-level fault injection [2], the faults are either injected by *forcing* the pins of ICs to faulty values using probes, or by *insertion*. The insertion technique uses a hardware fault injection module that is inserted between the pins of the fault injected target IC and the rest of the system. This allows the fault injected IC to be properly isolated from the other parts of the system. MESSALINE [2] is a pin-level fault injection tool which uses forcing while the RIFLE [3] tool uses the insertion technique to inject faults. The main drawback of the pin-level fault injection technique is the high intrusiveness on the system. Another drawback is the limited reachability, since only the pins of the ICs are accessible and not the internal parts of the circuits. Heavy-ion fault injection [4] is an example of radiation induced fault injection. In the heavy-ion technique, the target circuits are bombarded with radioactive particles (heavy ions). If the energy of the particles is sufficiently high, they may cause bit-flips in internal locations of the circuits. Another physical technique is electromagnetic interference (EMI) fault injection [5] which uses electromagnetic interference generated by a burst generator to inject faults. The advantage of using heavy-ion fault injection or EMI fault injection is that they feature low intrusiveness on the system as compared to pin-level fault injection. Another benefit is the high reachability since all locations within ICs can be affected. One drawback of these techniques is that the results of single fault injection experiments are not reproducible. *Scan-chain implemented fault injection* (SCIFI) uses scan-chains, i.e. built-in logic common in modern microprocessors for testing and on-chip debugging, to inject faults. The FIMBUL tool [6] injects faults using scan-chains conforming to the IEEE 1149.1 standard [7] and has been used to inject faults via the test access port of the Thor CPU [8]. Transient faults are injected in locations accessible by *internal* and *boundary* scan-chains. Other types of on-chip debug techniques exist beside those based on IEEE 1149.1. Background Debug Mode (BDM) is a proprietary technique from Freescale which has been successfully exploited for fault injection [9]. Both SCIFI and BDM fault injection feature low space intrusiveness since built-in test logic is used for injecting the faults without any extra hardware being required. Other on-chip debugging techniques, e.g., those supporting real-time tracing such as the IBM RISCTrace and the Nexus standard [10] are also suitable for fault injection as they feature minimal time overhead for injecting faults

and observing the system as the system does not need to be halted. Nexus-based fault injection experiments have been conducted with the GOOFI tool [11] and the INERTE tool [12].

2.3 Software-Implemented Fault Injection

Software-Implemented Fault Injection (SWIFI) can be used to emulate the effects of physical faults with software in a physical system. SWIFI can emulate faults in various parts of the hardware such as CPU registers, the ALU and the main memory. The various SWIFI techniques can be divided into *pre-runtime* injection techniques and *runtime* injection techniques depending on whether the faults are injected before the system starts executing the software or during the software execution. With pre-runtime SWIFI, one or more bits are inverted in the program and/or data image (i.e. in the executable file) before the program is executed, see e.g. DOCTOR [13]. There is no intrusion on the execution time using pre-runtime SWIFI, but the fault models are limited. Typical fault models are memory faults at the point in time the memory is first accessed. In runtime SWIFI, faults are injected during the execution of the application. The Xception tool [14] uses advanced debugging and performance monitoring features that already exist in many modern processors, e.g. Pentium, Alpha, MIPS and PowerPC, to inject faults. To be able to inject faults into real-time systems at full speed, an enhanced version of the Xception tool has been developed resulting in the real-time fault injection tool RT-Xception [15]. Another tool that takes advantage of processor debugging facilities is MAFALDA [16]. The aim of this tool has been to evaluate the use of COTS microkernels in safety-critical systems. The FERRARI tool [17] uses UNIX operating system calls to carry out fault injection. Hardware faults and errors are emulated by corrupting the program state during execution. This results in the same system behavior as would be found if an internal fault had been present, e.g. bit-flips in a CPU register. Software-implemented fault injection has also been used for validating object-oriented applications during runtime. FIRE [18] and Jaca [19] uses reflective programming to inject and monitor C++ and Java applications respectively. The PROPANE (PROPagation ANalysis Environment) tool [20] injects faults in C code executing on desktop computers. The code is instrumented with fault injection mechanisms and probes for logging traces of data values and memory areas. PROPANE supports the injection of both software faults (by mutation of source code) and data errors (by manipulating variable and memory contents).

2.4 Model-Implemented Fault Injection

In this paper, *model-implemented fault injection* is defined as a technique where fault injection mechanisms are developed as model blocks which can be inserted into:

- Hardware models, e.g. in VHDL specifications, at a development phase where no physical prototypes are available. Fault injection in hardware models are usually denoted as *simulation-based fault injection* in the literature.
- Software models where a fault may e.g. result in a data error.
- System models where a fault can e.g. emulate a faulty sensor, actuator or a sub system.

Fault injection in hardware models

Simulation-based fault injection can be used to study effects of faults in computer systems at various levels of abstraction. Simulations can be conducted at the electrical circuit level, the gate level or at various subsystem and functional levels. The main advantage of simulation-based fault injection is that simulations can be carried out early in the design cycle, before a physical prototype of the system is available. Simulations also provide high observability and controllability. The main drawback is that simulating the execution of a computer can be many orders of magnitude slower than the execution in a real system. A common way is to use simulation-based fault injection at the logic (gate) level by injecting faults into VHDL models of systems. The MEFISTO tool [21] uses two approaches for injecting faults into VHDL models. The first is based on the use of *saboteurs* and *mutants*. Saboteurs are extra VHDL components that carry out the fault injection, while mutants are modified (faulty) VHDL components replacing the original (fault-free) ones. The second approach uses built in commands of the simulator to manipulate signals and variables. The VERIFY tool [22] uses an extension of VHDL. In the description of each basic component (e.g. AND, NOT and OR gates), the mean time between faults, their duration and their effects can be specified. DEPEND [23] is an integrated design and fault injection environment. DEPEND supplies a C++ library of elementary and complex objects such as fault injectors, CPUs, N-modular redundant processors, communication links, voters and memory. These objects can be used to build and analyze a wide range of fault-tolerant architectures.

Fault injection in system models and software models

In the ESACS [24] and ISAAC [25] projects, a technique has been developed which inject faults in system models developed with the modeling language SCADE, to simulate failures of external hardware components, for example sensor failures. The technique exploits the principle of minimal cut sets (MCS) which is well-known from risk analysis methods like fault tree analysis (FTA) to identify combinations of faults causing a safety requirement violation. Another model-implemented fault injection approach for models developed with SCADE is evaluated in [26]. Here, the authors present a tool which automatically replaces original operators with fault injection nodes. A fault injection node encapsulates the original operator so the operator can be replaced or the operator output can be manipulated. During execution of the generated source code, the SCADE simulator is controlled to execute the model, inject the fault, and log the results. The tool allows the user to inject faults in all signals in the model. Work has also been done to inject faults at a functional level via MATLAB/Simulink to evaluate automotive control functions [27]. Here, abstract blocks (for example implementing an adaptive cruise control or an active body control) are marked as faulty and the effects of other blocks are monitored during simulation. Another work targeting fault injection at a functional level in Simulink models has been presented [28]. During execution of the Simulink model, fault injection blocks are added and executed to inject faults. An approach called *model based safety analysis* is suggested [29] where executable models of the system are executed by using existing

commercial tools such as Simulink and SCADE. To investigate how well faults are tolerated, the original model is extended with, for example, the stuck-at fault model and the extended model is formally verified using the SCADE Design Verifier. The AutoFMEA tool [30] generates failure mode and effects analyses (FMEAs) automatically. It targets Simulink models and supports safety engineers during the design stages when using model-based engineering techniques.

3 MODIFI

This section presents the capabilities and options of the MODIFI tool and, additionally, how MODIFI can generate MCSs which can be used as an optimized set of experiments for physical fault injection during system test.

3.1 Overview of the MODIFI Tool

The MODIFI (MODEL-Implemented Fault Injection) tool injects faults in behavior models developed using the Matlab/Simulink environment.

The purpose of the tool is to:

- *Carry out early evaluation of model robustness against faults.* Different design choices can be compared (benchmarked) to find the most robust version for the specific set of faults considered. Result from the fault injection experiments is used to design error detection mechanisms (EDMs) and error recovery mechanisms (ERMs) which can be added to the model.
- *Exercise and evaluate added EDMs and ERMs in the model.* Results from experiments, with and without fault handling mechanisms, are compared to verify that the capability of the model to handle faults is improved. Considering software models, added EDMs and ERMs will be automatically generated inside the source code from the software model. Hence, the EDMs/ERMs are then automatically downloaded to the actual system together with the rest of the software.
- *Create test cases for fault injection on the real system.* During fault injection at model level, effective errors leading to failures (safety requirement violations) are candidates for fault injection on the embedded system later on. To be able to e.g. reproduce faults injected on signals in the model a mapping is carried out between signal names in the model and physical addresses in a microcontroller memory where the signal (variable) value is stored. Test cases for physical fault injection are based on the XML output from the MODIFI which contains information about the golden (fault-free) run and all experiments.

The following Simulink model of a pedal voter will be used for illustrative purposes when different parts of the MODIFI tool are explained.

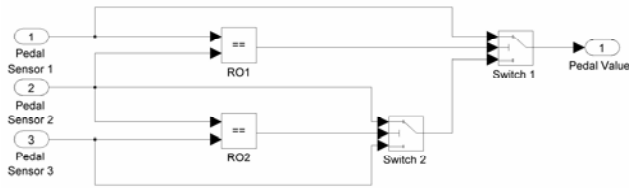


Fig. 1. Simulink model of a pedal sensor voter

Figure 1 show a TMR (triple modular redundancy) system which detects and masks a single faulty sensor value. Three input values are received from corresponding sensors and if one sensor is faulty, the output is assigned the value of one of the two fault-free sensor values.

The fault-injection experiments in MODIFI are configured using a Java-based Graphical User Interface (GUI) which is started from within Matlab, while the fault injection engine, which carries out the actual experiments, is implemented in Matlab code. The data information exchange format that is used in the communication between the GUI and the fault injection engine, as well as for the experiment logging, is XML. This makes it suitable for information exchange with external tools.

The first step for a user of MODIFI is to supply a Simulink model as target for the fault injection and a corresponding stimuli file. The stimuli file serves two purposes: to define the simulation time (time step and length) and provide signal values for all input ports in the entire system for the duration of the simulation. During the first step of the configuration the model is executed to verify the model as well as the stimuli file. The data gathered during this initial fault-free simulation is saved and kept as the golden run. The golden run is the simulation to which all experiments will be compared. The verification process also serves an additional purpose, to compile the Simulink model. This is necessary in order to determine the data types of Simulink block signals that are configured to have data type automatically selected by Simulink.

The next step is to select fault models, which are implemented as failure mode functions (FMF) in MODIFI, for each of the signals in the behavior model that shall be subject to fault injection. Figure 2 shows how the behavior model is represented in a tree structure. All the internal signals of the pedal sensor voter model are available as a level below the top level in the tree structure, including the three input signal blocks (PedalSensor1, PedalSensor2, PedalSensor3). The FMFs available are derived from the XML description supplied during the first configuration step. Only FMFs that are applicable to the data type of the location (e.g. an input port) are shown. In the example depicted, the data type of the selected signal is uint16, i.e. 16 bit unsigned integer. Thus only FMFs that are applicable to unsigned integers are displayed (Flip bits FMF).

The third step during configuration of fault-injection experiments is to define safety requirements. After an experiment has been executed, the simulation result is evaluated to check that no requirements have been violated. If there are violations, it

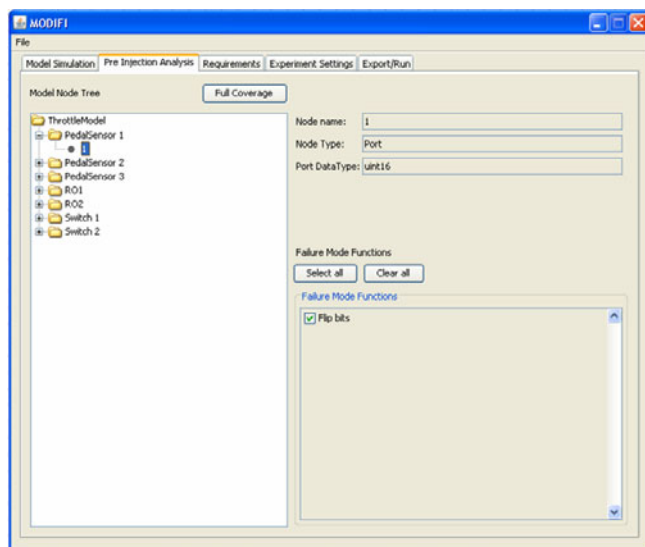


Fig. 2. MODIFI GUI – Assignment of fault models to model locations (signals)

will be indicated, since it is a proof that the requirement can be falsified using fault injection. It is neither mandatory nor necessary to use requirements for the fault injection experiments. In some cases it might be useful to perform fault injection experiments without evaluating the outcome during run-time, e.g. when analysis is to be performed in an external tool after the experiments are finished.

MODIFI can currently handle two ways of describing requirements. Requirements can be modeled into the behavior model by using the standard Simulink block “Assertion”. This is useful when requirements are depending on e.g. internal variables, states, or when complex requirements are described, e.g. requirements that are dependent on more than the current value of a behavior model system output variable. The other way to describe requirements in MODIFI is to use a requirement editor provided by MODIFI. This is useful for requirements that involve instantaneous values of the behavior model output, e.g. a value must not deviate from its nominal behavior or must not exceed a certain threshold. As already mentioned, the nominal behavior is gathered during the golden run, which is used as comparison for the fault injection experiments.

The fourth and final configuration step is related to different aspects of time. The user can define when to observe the system behavior (observation time window) and when to inject faults (fault injection time window). Since the systems analyzed in this tool are causal, it makes no sense to observe the system behavior before any faults are injected, which saves execution time. It also makes no sense to inject faults after the observation window has elapsed. The window time settings can be useful to prohibit the tool from injecting faults e.g. before the system has been simulated to a certain point in time.

A parameter that can be set is the time-steps per fault model/location-pair and it defines how many transient fault injection experiments that shall be performed for every combination of fault model and location (signal). Within the fault-injection time window, the fault injection times are selected randomly. Another option which is possible to select is to *collapse time domain*. If a safety requirement is violated, this option tells the fault injection engine to cancel any further experiments for the same combination of fault model and location. That is, ignoring that the same combination of fault model and location can lead to different system behavior depending on when they are injected.

After the final configuration step, the fault injection engine is invoked to perform the experiments. The user can then visually follow the fault injection experiment progress in a progress cube, see Figure 3.

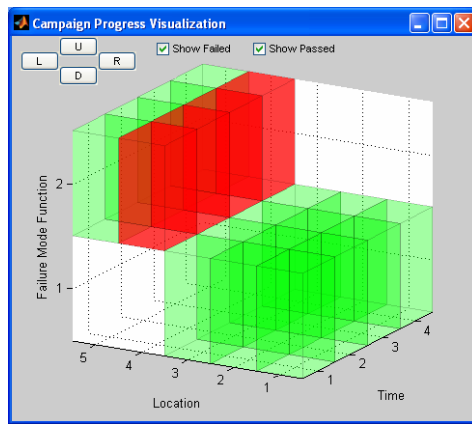


Fig. 3. MODIFI – Progress window

The progress window shows one cube for each campaign (set of fault-injection experiments) that is executed. The three axes in a cube represent the locations (signals), FMFs as well as the positions in time when faults are injected into the behavior model. Each position in the cube represents a single experiment, and the user can visually distinguish between experiments to be performed, experiments that were unsuccessful (e.g. due to timeout), experiments that were successful but violated requirement(s), and experiments that were successful and did not falsify any requirement.

In the particular example shown in Figure 3, the pedal voter model is used. Locations 1 to 3 are the three pedal sensor inputs, and location 4 and 5 are the outputs of the switches. In this example the stimuli is set to simulate the model for 4 time steps only. Different fault models are applicable for different locations. In this example the offset fault model (implemented by failure mode function 1) is used on the pedal sensor inputs and the bit-flip fault model (failure mode function 2) is used on the switch outputs. The safety requirement in this case is: *there shall be no deviation between the output from the golden run and the output of a fault-injection experiment.*

The green color represents passed experiments whereas red color represents failed ones, i.e. experiments where the safety requirement was violated. As expected, for this campaign it was only faults injected at location 4 (the output of switch 1), regardless of fault-injection time, that was able to create a faulty output.

The output from a fault-injection experiment is captured in an XML file. The XML file is actually the input stimuli file which has been appended with information about the golden run and fault-injection results. This way, the output file can be fed back and used as input for regression testing when modifications, e.g. by adding fault tolerance mechanisms, have been made to the model.

3.2 Fault-Injection Based Minimal Cut Sets Generation

In addition to conventional fault injection, MODIFI offers the possibility to determine MCSs based on fault-injection experiments. Traditionally, MCS are associated with fault-tree analysis where they represent the minimal set of basic causes which activate the top-level event.

As presented in Section 2, fault injection together with model checking (a formal verification technique) has been used to determine MCS in SCADE models [25]. This technique is based on the introduction of a safety requirement, proof objective, which is checked for falsification under the influence of faults during a backwards traversal of the model. This is similar to the top-down approach used in FTA, see Figure 4

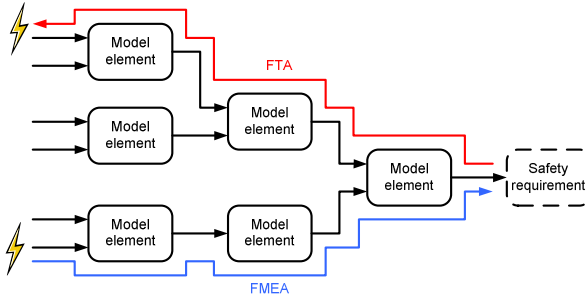


Fig. 4. FMEA versus FTA

The MODIFI tool, being a fault-injection tool and thereby also capable of performing automated FMEA, uses a different, experimental, approach to determine MCSs. Single faults or combinations of faults are injected on block outputs and possibly propagated towards the output signals; see the FMEA arrow in Figure 4. After the propagation, the safety requirements are checked for violations. The MCS algorithm [31] in MODIFI starts by injecting all defined/configured single faults one at a time. If a safety requirement is violated, the present fault is added to the MCS list as well as being removed from the experiments which search for MCSs of higher order. When all possible single faults have been injected, double faults are injected and consequently MCSs of order two are possibly found. This continues until the maximum MCS order, specified by the user of MODIFI, is reached. Despite being a

time-consuming task (the number of experiments grows exponentially when locations and fault models are added), the MCSs give valuable information on the fault tolerance of the system represented as a model.

The pedal voter model and the selected fault models are again used as an example. The offset fault model (F1) is used on the sensor inputs (locations L1-L3) and the bit-flip fault model (F2) is used on switch 1 and switch 2 (locations L4 and L5). There are four time steps (T1-T4) at which faults can be injected. For each time step, Tx, the list of single faults to be injected consists of the following fault model-location pairs: {F1,L1}, {F1,L2}, {F1,L3}, {F2,L4}, and {F2,L5}. Thus, as Figure 3 shows, after all single faults have been injected, the list of MCSs is: {F2,L4,T1}, {F2,L4,T2}, {F2,L4,T3}, and {F2,L4,T4} since these faults violated the safety requirement. Consequently, these faults are removed from the list of double faults to be injected. As a result, the list of double faults to inject for each Tx becomes: ({F1,L1},{F1,L2}), ({F1,L1},{F1,L3}), ({F1,L1},{F2,L5}), ({F1,L2},{F1,L3}), ({F1,L2},{F2,L5}), and ({F1,L3},{F2,L5}), in total 24 experiments if four time steps are chosen for each combination. Since the voter is only single fault tolerant, the majority of these faults leads to a safety requirement violation and these combinations are added to the MCS list, e.g. ({F1,L1,T1},{F1,L3,T1}). For a more detailed description of this example, see [31].

3.3 MODIFI Properties

Reachability as well as *controllability* obtained with MODIFI is high since all blocks with output signals can be subject to fault injection. Both *repeatability* and *reproducibility* are considered high since all fault injection experiments are fully repeatable and reproducible. *Intrusiveness* is considered low since the added blocks does not affect the model behavior over time. The *flexibility* is considered high since any Simulink block with an output can be subject to fault injection. The *effectiveness* of the MODIFI approach is high, since the faults are applied directly to block signals and fault handling mechanisms can easily be exercised. The *efficiency* is also considered high since the process is fully automated and an experiment typically takes less than one second to perform. The *observability* is also considered high since all output signal values are stored for every time step and additional output signals can be connected as probes within the model.

4 Conclusions

In this paper different techniques and views on fault injection have been discussed and the concept of model-implemented fault injection has been defined. A tool, MODIFI, which is one implementation of the model-implemented fault-injection concept, has been introduced. MODIFI targets behavioral models in Simulink and over 30 different fault models can be fetched from a library and injected at locations in the model. MODIFI also offers the possibility to experimentally determine MCSs for systems represented as Simulink models. MODIFI enables early evaluation, where different design choices can be benchmarked to find the most robust version for the specific set of faults considered, and added fault handling mechanisms can be exer-

cised and evaluated with new fault injection experiments. The MODIFI output is described in an XML format which can be automatically transformed to a suitable input for efficient fault injection during system test.

Acknowledgments. This work has been funded by the research project MOGENTES within the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no 216679.

References

1. Matlab/Simulink, <http://www.mathworks.com/products/simulink/>
2. Arlat, J., Aguera, M., Amat, L., Crouzet, Y., Fabre, J.C., Laprie, J.C., Martins, E., Powell, D.: Fault injection for dependability validation: a methodology and some applications. *IEEE Transactions on Software Engineering* 16(2), 166–182 (1990)
3. Madeira, H., Relat, M., Moreira, F., Silva, J.G.: RIFLE. A general purpose pin-level fault injector. In: *Proceedings of the 1st European Dependable Computing Conference (EDCC-1)*, Berlin, Germany, October 4-6, pp. 199–216 (1994)
4. Karlsson, J., Liden, P., Dahlgren, P., Johansson, R., Gunneflo, U.: Using heavy-ion radiation to validate fault-handling mechanisms. *IEEE Micro* 14(1), 8–23 (1994)
5. Karlsson, J., Folkesson, P., Arlat, J., Crouzet, Y., Leber, G., Reisinger, J.: Application of three physical fault injection techniques to the experimental assessment of the MARS architecture. In: *5th IFIP Working Conference on Dependable Computing for Critical Applications (DCCA-5)*, pp. 267–287. Urbana Champaign, IL, USA (September 1995)
6. Folkesson, P., Svensson, S., Karlsson, J.: A comparison of simulation based and scan chain implemented fault injection. In: *Proceedings of 28th International Symposium on Fault Tolerant Computing*, Munich, Germany, June 23-25, pp. 284–293 (1998)
7. Test technology technical committee of the IEEE computer society. IEEE standard test access port and boundary-scan architecture. USA (1990)
8. Saab Ericsson Space AB, Microprocessor Thor, product information (1993)
9. Rebaudengo, M., Sonza Reorda, M.: Evaluating the fault tolerance capabilities of embedded systems via BDM. In: *Proceedings 17th IEEE VLSI Test Symposium*, Dana Point, CA, USA, April 25-29, pp. 452–457 (1999)
10. IEEE-ISTO, The Nexus 5001 forumTM standard for a global embedded processor debug interface, pp. 9-10 (1999)
11. Vinter, J., Aidemark, J., Skarin, D., Barbosa, R., Folkesson, P., Karlsson, J.: An overview of GOOFI - a generic object-oriented fault injection framework, Technical Report No. 05-07, Department of Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden, 2005. This report extends the paper GOOFI: generic object-oriented fault injection tool. In: *Proceedings of the International Conference on Dependable Systems and Networks (DSN-2001)*, Göteborg, Sweden (July 2001)
12. Yuste, P., De Andres, D., Lemus, L., Serrano, J.J., Gil, P.: INERTE: Integrated NEXus-based Real-Time fault injection tool for Embedded systems. In: *Proceedings of the International Conference on Dependable Systems and Networks*, San Francisco, CA, United States, p. 669 (June 22-25, 2003)
13. Seungjae, H., Shin, K.G., Rosenberg, H.A.: DOCTOR: an integrated software fault injection environment for distributed real-time systems. In: *Proceedings of 1995 IEEE International Computer Performance and Dependability Symposium*, Erlangen, Germany, April 24-26, pp. 204–213 (1995)

14. Carreira, J., Madeira, H., Silva, J.G.: Xception: a technique for the experimental evaluation of dependability in modern computers. *IEEE Transactions on Software Engineering* 24(2), 125–136 (1998)
15. Cunha, J.C., Rela, M.Z., Silva, J.G.: Can software implemented fault-injection be used on real-time systems? In: *Proceedings of EDCC-3: Third European Dependable Computing Conference*, Prague, Czech Republic, pp. 209–226 (September 15–17, 1999)
16. Fabre, J.-C., Salles, F., Moreno, M.R., Arlat, J.: Assessment of COTS microkernels by fault injection. *Dependable Computing for Critical Applications* 7, 25–44 (1999)
17. Kanawati, G.A., Kanawati, N.A., Abraham, J.A.: FERRARI: a flexible software-based fault and error injection system. *IEEE Transactions on Computers* 44(2), 248–260 (1995)
18. Martins, E., Rosa, A.C.A.: A fault injection approach based on reflective programming. In: *Proceedings of International Conference on Dependable Systems and Networks (includes FTCS 30 30th Annual International Symposium on Fault Tolerant Computing and DCCA 8)*, New York, NY, USA, pp. 407–416 (June 25–28, 2000)
19. Martins, E., Rubira, C.M.F., Leme, N.G.M.: Jaca: a reflective fault injection tool based on patterns. In: *Proceedings International Conference on Dependable Systems and Networks*, Washington, DC, USA, pp. 483–487 (June 23–26, 2002)
20. Hiller, M.: A software profiling methodology for design and assessment of dependable software, Ph.D thesis, Chalmers University of Technology, Göteborg, Sweden (2002)
21. Jenn, E., Arlat, J., Rimen, M., Ohlsson, J., Karlsson, J.: Fault injection into VHDL models: the MEFISTO tool. In: *Proceedings of IEEE 24th International Symposium on Fault Tolerant Computing*, Austin, TX, USA, pp. 66–75 (June 15–17, 1994)
22. Sieh, V., Tschache, O., Balbach, F.: VERIFY: evaluation of reliability using VHDL-models with embedded fault descriptions. In: *Proceedings of IEEE 27th International Symposium on Fault Tolerant Computing*, Seattle, WA, USA, pp. 32–36 (June 24–27, 1997)
23. Goswami, K.K.: DEPEND: a simulation-based environment for system level dependability analysis. *IEEE Transactions on Computers* 46(1), 60–74 (1997)
24. Enhanced Safety Assessment for Complex Systems. FP5-GROWTH contract no. G4RD-CT-2000-00361
25. Improvement of Safety Activities on Aeronautical Complex systems, FP6-AEROSPACE project reference 501848
26. Vinter, J., Bromander, L., Raistrick, P., Edler, H.: FISCADE - a fault injection tool for SCADE models. In: *Proceeding of the 3rd IET Conference on Automotive Electronics*, University of Warwick, UK (June 28–29, 2007) ISBN: 978-0-86341-815-0
27. Vulinovic, S., Schlingloff, B.H.: Model based dependability evaluation for automotive control functions. In: *Invited Session: Model-Based Design and Test, 9th World Multi-Conference on Systemics, Cybernetics and Informatics*, Florida (2005)
28. Isacson, J., Ljungberg, M.: Fault injection in Matlab/Simulink. Master's Thesis Report, Department of Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden (2008)
29. Joshi, A., Heimdahl, M.P.E.: Model-based safety analysis of Simulink models using SCADE Design Verifier. In: Winther, R., Gran, B.A., Dahll, G. (eds.) *SAFECOMP 2005*. LNCS, vol. 3688, pp. 122–135. Springer, Heidelberg (2005)
30. Ricardo AutoFMEA, <http://www.ricardo-autofmea.com>
31. Svenningsson, R., Vinter, J., Eriksson, H., Törngren, M.: Towards fault injection based minimal cuts sets generation. In: *Proceedings of the Microelectronics Conference ME 2010* (2010)

Automated Test Coverage Measurement for Reactor Protection System Software Implemented in Function Block Diagram

Eunkyoung Jee¹, Suin Kim², Sungdeok Cha³, and Insup Lee¹

¹ University of Pennsylvania, 3330 Walnut Street, Philadelphia, PA 19104, USA
eunkjee@seas.upenn.edu, lee@cis.upenn.edu

² KAIST, 335 Gwahangno Yuseong-gu, Daejeon, Republic of Korea
suin.kim@gmail.com

³ Korea University, Anam-dong Seongbuk-gu, Seoul, Republic of Korea
scha@korea.ac.kr

Abstract. We present *FBDTestMeasurer*, an automated test coverage measurement tool for function block diagram (FBD) programs which are increasingly used in implementing safety critical systems such as nuclear reactor protection systems. We have defined new structural test coverage criteria for FBD programs in which dataflow-centric characteristics of FBD programs were well reflected. Given an FBD program and a set of test cases, *FBDTestMeasurer* produces test coverage score and uncovered test requirements with respect to the selected coverage criteria. Visual representation of uncovered data paths enables testers to easily identify which parts of the program need to be tested further. We found many aspects of the FBD logic that were not tested sufficiently when conducting a case study using test cases prepared by domain experts for reactor protection system software. Domain experts found this technique and tool highly intuitive and useful to measure the adequacy of FBD testing and generate additional test cases.

Keywords: test coverage measurement, test automation, function block diagram, programmable logic controller.

1 Introduction

As programmable logic controllers (PLCs) are widely used to implement safety-critical systems such as nuclear reactor protection systems, testing of PLC programs is getting more important. Among the five standard PLC programming languages defined by the International Electrotechnical Commission (IEC) [1], function block diagram (FBD) is a commonly used implementation language. The Korea Nuclear Instrumentation and Control System R&D Center (KNICS) project, whose goal is to develop a comprehensive suite of digital reactor protection systems, is an example in which PLC programs implementing safety critical systems were implemented in FBD. For such safety critical systems to be approved for operation, developers must demonstrate compliance to strict quality requirements including unit testing and test result evaluations [2,3].

Current FBD testing relies on mostly functional testing in which test cases are manually derived from natural language requirements. Although functional testing and structural testing are complementary each other and both are required to be applied to safety critical software [3], there have been little research and practices on structural testing for FBD programs.

Another difficulty of current FBD testing is lack of test evaluation techniques. Regulation authorities such as U.S.NRC require that test results be documented and evaluated to ensure that test requirements have been satisfied [2]. Although test results for FBD programs implementing safety critical software need to be evaluated thoroughly, there have been no other methods directly applicable to FBD programs except manually reviewing and analyzing test documents for assuring test quality. Domain experts have felt that manual reviews only were not adequate to assure test quality. More systematic and quantitative ways to evaluate the adequacy of the test cases have been strongly required.

In order to enable the structural testing for FBD programs, we, software engineers, have defined structural test coverage criteria suitable to FBD programs in which the unique characteristics of the FBD language were fully reflected [4]. An FBD program is interpreted as a directed data flow graph and three test coverage criteria have been defined using the notion of the data flow path (d-path) and the d-path condition (DPC) for each d-path.

To work out a solution to lack of systematic test evaluation methods, we present an automated test coverage measurement tool, *FBDTestMeasurer*, which measures the coverage of a set of test cases on the FBD program with respect to the test coverage criteria proposed in [4]. Given a unit FBD program, a set of test cases, and selected test coverage criteria, *FBDTestMeasurer* generates test requirements with respect to the selected test coverage criteria and measures the coverage of the test cases automatically. It provides coverage score and unsatisfied test requirements as a result. Uncovered d-paths can be visually presented on the graphical view of the FBD program.

The proposed technique has following contributions: 1) automated quantitative and systematic test evaluation for FBD programs gives concrete basis of quality assurance, 2) visual representation of uncovered d-paths on the FBD program helps testers analyze the uncovered test requirements intuitively, and 3) unsatisfied test requirements provided by *FBDTestMeasurer* reveal inadequately tested parts and help testers generate additional test cases.

We conducted a case study using representative trip (shutdown) modules of the Bistable Processor (BP) of the Reactor Protection Systems (RPS) in the KNICS project. The test cases had been manually generated by the FBD testing experts working in the KNICS project. It took nearly 3 man-months to generate the test cases for the whole BP system. We could find many insufficiently tested aspects of the FBD program by the set of test cases.

The remainder of this paper is organized as follows: Section 2 provides the background for the study including a literature survey of the most relevant research. Section 3 explains the test coverage criteria for FBD programs. Section 4 presents the automated test coverage measurement techniques for FBD

programs and the related issues. Section 5 demonstrates the results of the KNICS BP case study. We conclude the paper in Section 6.

2 Related Work

PLC programs are executed in a permanent loop. In each iteration of the loop, called a scan cycle, the PLC program reads inputs, computes a new internal state and outputs, and updates outputs. This cyclic behavior makes PLCs suitable for control tasks and interaction with continuous environments [5].

FBD, one of the standard PLC programming languages, is widely used because of its graphical notations and suitability for developing applications with a high degree of data flow among the components. FBD is a data flow language based on viewing a system in terms of the flow of signals between processing elements [6]. A collection of blocks is wired together like a circuit diagram as shown in Figure 1. An example FBD network of Figure 1 is a part of the fixed-set-point-falling trip logic of the BP for the RPS. The output variable *th_X_Logic_Trip* is set to *true* if the *f_X* value falls below the trip set-point (*k_X_Trip_Setpoint*) for longer than the specified delay (*k_Trip_Delay*). The trip signal *true* would safely shut down a nuclear reactor. Blocks of FBD programs are categorized into functions and function blocks. A function does not have internal states and its output is determined solely by current inputs. In contrast, a function block maintains internal states and produces outputs. In Figure 1, the TON block is a function block, and all other blocks (e.g., ADD.INT, LE.INT, and SEL) are functions.

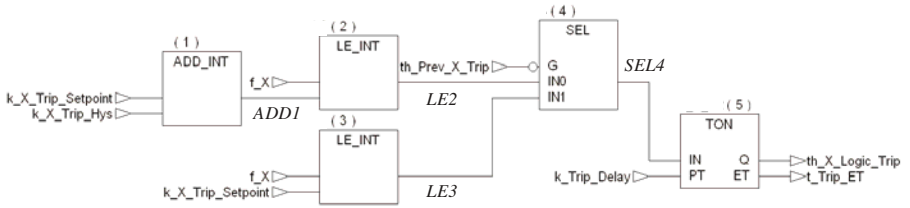


Fig. 1. A small FBD program for calculating *th_X_Logic_Trip*

We focus on unit testing for FBD programs. A unit FBD consists of blocks necessary to compute a primary output (e.g., *th_X_Logic_Trip* in Figure 1) according to the unit definition on FBD programs [7]. There have been functional testing research and practices to test FBD units. In [8], authors have developed a simulation-based validation tool named *SIVAT* which uses ANSI C code generated from FBD programs internally. In [9], an FBD program is transformed into a High Level Timed Petri Nets (HLTPN) model, and the simulation-based testing is performed on the HLTPN model. An integrated tool environment named *PLCTOOLS* has been developed to support the entire development process including specification, transformation, and simulation. Unfortunately, these approaches support

only functional testing. Neither internal structure nor the dataflow-centric aspects of FBD programs were analyzed in their testing approaches.

As a member of the KNICS project, due to lack of structural testing techniques and coverage criteria readily applicable to FBD programs, we tried to apply conventional test coverage criteria to FBD programs by transforming FBD programs into equivalent control flow graphs (CFGs) [10,11]. Although this approach contributed to make structural testing for FBD programs possible, it has limitations in that CFGs does not accurately reflect the data flow-centric characteristics of FBD. Our experience made it clear that conventional structural testing techniques and coverage criteria, originally developed for procedural programming languages, do not work well on FBD programs. We have developed new test coverage criteria for FBD programs by focusing on the data flow aspects of the FBD language [4]. Test coverage criteria can be really useful when they are integrated with automated tool supports. In this paper, we propose a test coverage measurement procedure and an automated tool based on the test coverage criteria defined in our previous work.

There have been a lot of research and tools for code coverage [12,13,14,15,16]. These approaches and our approach have common basic principles of test coverage measurement. However, these tools target the procedural languages such as C, C++, Java, Cobol, Perl, PHP, Ada, etc., not the data flow languages such as FBD and Lustre. They use the test coverage criteria defined on control flow graphs (e.g., statement coverage, decision coverage, etc.) while we use the different test coverage criteria defined on data flow graphs (e.g., basic coverage, input condition coverage, etc.).

Research of test coverage criteria for data flow languages is not new. A. Lakehal et al. [17] have defined the structural test coverage criteria for Lustre, a synchronous data-flow declarative language, based on the activation condition which specifies when the data flow from an input edge to an output edge may occur. Depending on the path length and the values taken along the edges, multiple coverage criteria were defined. They developed *Lustructu* [18], a tool for the automated coverage assessment of Lustre programs. While the concept of activation condition was useful, the approach presented in [17] was unable to cope with complex function block conditions of FBD programs because the target operators of [17] were limited to simple temporal operators. We developed a systematic way to deal with nontrivial function block conditions by identifying the internal variables and involving them in the function block conditions. In addition to customizing the activation condition concept to properly reflect the characteristics unique to FBD, we also extended their approach by supporting multiple outputs as well as non-Boolean edges.

3 Test Coverage Criteria for FBD Programs

3.1 D-Path and D-Path Condition

We have defined the structural test coverage criteria for FBD programs [4]. An FBD program is considered as a directed graph with multiple inputs and outputs.

The FBD program shown in Figure 1 consists of five blocks and 13 edges. A *d-path* is a finite sequence $\langle e_1, e_2, \dots, e_n \rangle$ of edges in the directed graph of an FBD program. A unit d-path is a d-path with the length 2 in the form of $\langle e_i, e_o \rangle$. A d-path is guaranteed to be finite because FBD programs have no internal feedback loops. The *d-path condition* (DPC) of a d-path is the condition along the d-path under which the input value plays a role in computing the output. We use the *d-* prefix to distinguish the *d-path* and the *d-path condition* from the traditional *path* and the *path condition* defined in control flow graphs. The d-path condition of a d-path p , $DPC(p)$, is defined by conjunction of function condition, $FC(\langle e_i, e_{i+1} \rangle)$, for each function and function block condition, $FBC(\langle e_j, e_{j+1} \rangle)$, for each function block along the d-path.

3.2 Function Condition (FC) and Function Block Condition (FBC)

$FC(\langle e_i, e_o \rangle)$, is the condition under which the value at the output edge e_o is influenced by the value at the input edge e_i through a single function. If a function has n inputs, there exist n FCs for each d-path from an input to the output. There are three types of FCs.

For the functions belonging to type 1, all inputs always play a role in determining the output. Best illustrated by the *ADD* function, FCs for all the unit d-paths are *true*. In the type 2 functions, an input value appears unchanged on the output edge in a certain condition. The *SEL* function is an obvious example in that either e_{IN0} or e_{IN1} flows into the output unchanged depending on the value of e_G . The *AND* block is another example. If e_{IN1} is *true*, the value *true* flows into the output only if the other input e_{IN2} is also *true*. If e_{IN1} is *false*, the output is also *false* without any further constraints. Formal definitions of FCs for the *AND* block with two inputs $IN1$ and $IN2$ are:

if $p_1 = \langle e_{IN1}, e_{OUT} \rangle \wedge p_2 = \langle e_{IN2}, e_{OUT} \rangle \wedge e_{OUT} = AND(e_{IN1}, e_{IN2})$,

$$\begin{aligned} FC(p_1) &= \text{if } e_{IN1} \text{ then } e_{IN2} \text{ else true} \\ &= \neg e_{IN1} \vee e_{IN2} \\ FC(p_2) &= \neg e_{IN2} \vee e_{IN1} \end{aligned}$$

The type 3 functions have characteristics such that some or all input values are used in determining the output computation under specific conditions. Unlike the type 2 functions, the output of the type 3 function is not necessarily same as one of the inputs.

We categorized all FBCs into type 4. $FBC(\langle e_i, e_o \rangle)$ is same as $FC(\langle e_i, e_o \rangle)$ except e_i and e_o are connected by a single “function block”. Whereas FC definitions are relatively simple, FBC definitions are more complex due to the internal variables which are modeled as the implicit edges to the function block in this approach. For example, the semantics of TOF (Timer Off Delay) function block, shown in Figure 2(a), is such that it generates the Q output *false* when the IN input remains *false* during the delay time specified by the variable PT ever since the IN value turned to *false* from *true*. Otherwise, the output Q is *true*.

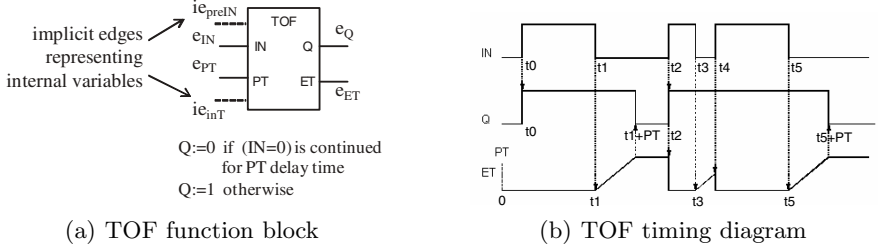


Fig. 2. TOF function block and its behavioral definition

The behavioral definition of timer blocks such as the TOF is described by timing diagrams as shown in Figure 2(b). It shows how outputs Q and ET vary in response to the different IN values as time passes.

When we formally defined the TOF semantics by representing all the possible inputs and output combinations of relevant variables using a condition/action table, two internal variables were identified. $preIN$ and inT denote the value of the IN stored in the previous scan cycle and the internal timer, respectively.

FBCs for the output Q of TOF are defined as follows:

if $p_1 = \langle e_{IN}, e_Q \rangle \wedge p_2 = \langle e_{PT}, e_Q \rangle \wedge e_Q = TOF\_Q(e_{IN}, e_{PT})$,

$$\begin{aligned} FBC(p_1) &= \text{if } e_{IN} \text{ then true else } (\neg ie_{preIN} \wedge (ie_{inT} = 0 \vee (ie_{inT} \geq e_{PT}))) \\ &= e_{IN} \vee (ie_{preIN} = 0 \wedge (ie_{inT} = 0 \vee ie_{inT} \geq e_{PT})) \end{aligned}$$

$$FBC(p_2) = (ie_{inT} > 0)$$

For the $FBC(p_1)$, when the e_{IN} is *true*, it flows into the output e_Q without any constraints. If the e_{IN} is *false*, the output e_Q is also *false* only if $(\neg ie_{preIN} \wedge (ie_{inT} = 0 \vee (ie_{inT} \geq e_{PT})))$. The ie represents an implicit edge as opposed to an explicit edge. We defined all FCs and FBCs for the functions and function blocks in the IEC standard[1]. Detailed definitions can be found in [19].

3.3 FBD Test Coverage Criteria

Three different test coverage criteria for FBD programs are defined based on the definition of DPCs. The process of deriving d-path condition (DPC) is similar to the one used in backward symbolic execution. Starting from the output edge of the given d-path, each FC or FBC is expanded. For example, when there are two functions and one function block in the d-path $p_{4.1} = \langle f\_X, LE2, SEL4, th\_X\_Logic\_Trip \rangle$ in Figure 1, $DPC(p_{4.1})$ is conjunction of two FCs and one FBC as follows:

$$\begin{aligned} DPC(p_{4.1}) &= DPC(\langle f\_X, LE2, SEL4, th\_X\_Logic\_Trip \rangle) \\ &= FC(\langle f\_X, LE2 \rangle) \wedge FC(\langle LE2, SEL4 \rangle) \wedge FBC(\langle SEL4, th\_X\_Logic\_Trip \rangle) \end{aligned} \tag{1}$$

When the backward symbolic computation is completed, the DPC should contain only input and internal variables because all the expressions corresponding to the intermediate edges would be replaced. For example, the expression of (1) is transformed into the expression with only input and internal variables by substituting the FCs and the FBC with the corresponding expressions from (2) to (4) and then substituting the intermediate edge names with the expressions from (5) to (8).

$$FC(\langle f\_X, LE2 \rangle) = true \quad (2)$$

$$FC(\langle LE2, SEL4 \rangle) = th\_Prev\_X\_Trip \quad (3)$$

$$FBC(\langle SEL4, th\_X\_Logic\_Trip \rangle) = \\ SEL4 \vee (preSEL4 = 0 \wedge (inT5 = 0 \vee inT5 \geq k\_Trip\_Delay)) \quad (4)$$

$$SEL4 = \neg th\_Prev\_X\_Trip ? LE3 : LE2 \quad (5)$$

$$LE3 = f\_X \leq k\_X\_Trip\_Setpoint \quad (6)$$

$$LE2 = f\_X \leq ADD1 \quad (7)$$

$$ADD1 = k\_X\_Trip\_Setpoint + k\_X\_Trip\_Hys \quad (8)$$

Building on the definition of DPC, the basic coverage, the input condition coverage, and the complex condition coverage have been defined for FBD programs. Let DP denote the set of all d-paths from input edges to output edges.

Definition 1. *A set of test data T satisfies the basic coverage criterion if and only if $\forall p \in DP \exists t \in T |DPC(p)|_t = true$.*

The basic coverage (BC) focuses on covering every d-path in the FBD program under test at least once. Test requirements for BC are DPCs for all d-paths of the target program. A test case t is meaningful if the input of the d-path p has influence in determining the output of p . Such condition is captured by $|DPC(p)|_t = true$ in the above definition. Otherwise (e.g., $|DPC(p)|_t = false$), the test case t is unable to make the input of the p flow down the given d-path and survive all the way to the output. Such test case is surely ineffective in testing the correctness of the d-path, and it fails to contribute towards meeting the coverage requirement.

While the basic coverage is straightforward in concept, it is often ineffective in detecting logical errors that FBD programs might have. Another stronger coverage is needed.

Definition 2. *A set of test data T satisfies the input condition coverage criterion if and only if, $\forall p \in DP, \exists t \in T |in(p) \wedge DPC(p)|_t = true$ and $\exists t' \in T |\neg in(p) \wedge DPC(p)|_{t'} = true$ where $in(p)$ is a Boolean input edge of the d-path p .*

To satisfy the input condition coverage (ICC) criterion, it is no longer sufficient to choose an arbitrary value for the input edge whose value would influence the outcome. One must now choose a set of test data such that input values include

both *true* and *false* for Boolean inputs (e.g., $DPC(p_{3.1}) \wedge th\_Prev\_X\_Trip$ as well as $DPC(p_{3.1}) \wedge \neg th\_Prev\_X\_Trip$ for $p_{3.1} = \langle th\_Prev\_X\_Trip, SEL4, th\_X\_Logic\_Trip \rangle$).

Definition 3. A set of test data T satisfies the complex condition coverage criterion if and only if, $\forall p \in DP, \exists t \in T | e_i \wedge DPC(p)|_t = true$ and $\exists t' \in T | \neg e_i \wedge DPC(p)|_{t'} = true$ where e_i is a Boolean edge in the d-path p of length n and $1 \leq i \leq n$.

The complex condition coverage (CCC) criterion which is stronger than the ICC requires that every Boolean edge's variation in the d-path be tested at least once with the satisfied DPC. Every test set satisfying the ICC criterion also satisfies the BC criterion. Similarly, the CCC criterion subsumes both the ICC and the BC criteria.

4 Automated Test Coverage Measurement for FBD Programs

4.1 FBDTestMeasurer

Test coverage measurement is a general method to evaluate test adequacy. We developed *FBDTestMeasurer* to measure coverage of a set of test cases with respect to the structural test coverage criteria for FBD programs automatically. Figure 3 shows the architecture of *FBDTestMeasurer*.

Parsing. *FBDTestMeasurer* receives a unit FBD program in the standard XML format and extracts d-paths for the selected outputs. A unit FBD program may have many outputs, but there are usually one or a few primary outputs on which analysis should focus. *FBDTestMeasurer* allows users to choose output variables which they want to analyze.

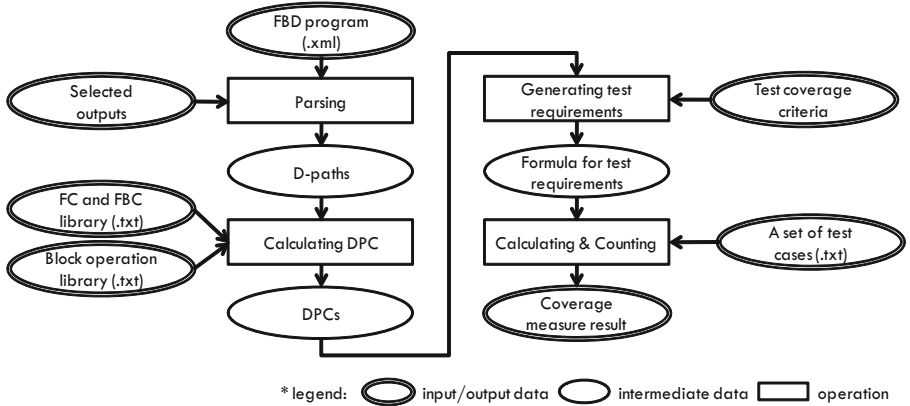


Fig. 3. *FBDTestMeasurer* architecture

Calculating DPC. DPC calculation requires an FC and FBC library and a block operation library as well as d-path information. DPC is the conjunction of FCs and FBCs. When calculating DPC, *FBDTestMeasurer* searches the necessary FC or FBC templates from the FC and FBC library which includes FCs and FBCs for all functions and function blocks in the IEC standard[1].

For example, expressions (2), (3), and (4) are calculated by extracting corresponding FC and FBC templates from the FC and FBC library. The block operation library is necessary to replace intermediate edge names in the DPC with corresponding operational descriptions. *FBDTestMeasurer* searches for the corresponding block's operational description from the block operation library in order to make the DPC contain only input and internal variables. For example, *SEL4*, an intermediate edge name, included in the expression (4) is replaced by $\neg th\_Prev\_X\_Trip ? LE3 : LE2$ shown in expression (5) after extracting the operational description template for the *SEL* from the block operation library. *LE3* and *LE2* are replaced by expression (6) and (7), respectively.

We decided to keep FC/FBC information and block operation information in separate library files for flexible capability to cope with new blocks. FBD programs can have various kinds of blocks and many PLC case tools allow users to make user-defined blocks. When new blocks are used in the FBD program, DPC calculation still works well if users simply insert the FCs or FBCs and the operational descriptions of the new blocks into the library files.

Generating test requirements. *FBDTestMeasurer* enables users to select test coverage criteria which they want to specify. One or more test coverage criteria can be selected. According to the selected test coverage criteria, *FBDTestMeasurer* generates test requirements. All test requirements are represented by logical formula connected by conjunction.

Calculating and Counting. *FBDTestMeasurer* receives a set of test cases. We made a textual file template for specifying test cases of FBD programs. If assigning input values of a test case to a test requirement makes the test requirement *true*, the test requirement is covered by the test case. *FBDTestMeasurer* counts test requirements covered at least once by the test cases. After counting covered test requirements, *FBDTestMeasurer* provides test coverage score, i.e., percentage of the number of covered test requirements divided by the number of all test requirements, and uncovered test requirements.

Figure 4 shows a screen shot of *FBDTestMeasurer* which consists of several parts: input files open, d-path finder, user's selection, graphical view of the target FBD program, and result console. Given a unit FBD program and a set of test cases by opening files, *FBDTestMeasurer* presents a graphical view of the target program. When a user selects test coverage criteria and output variables in the left window of the tool, *FBDTestMeasurer* shows the coverage measurement result in the output console and produce a log file.

We implemented a D-Path Finder feature which visually highlights a d-path with the number which the user specifies. Specially, this function is highly effective to reveal which parts of the program were not covered. When the

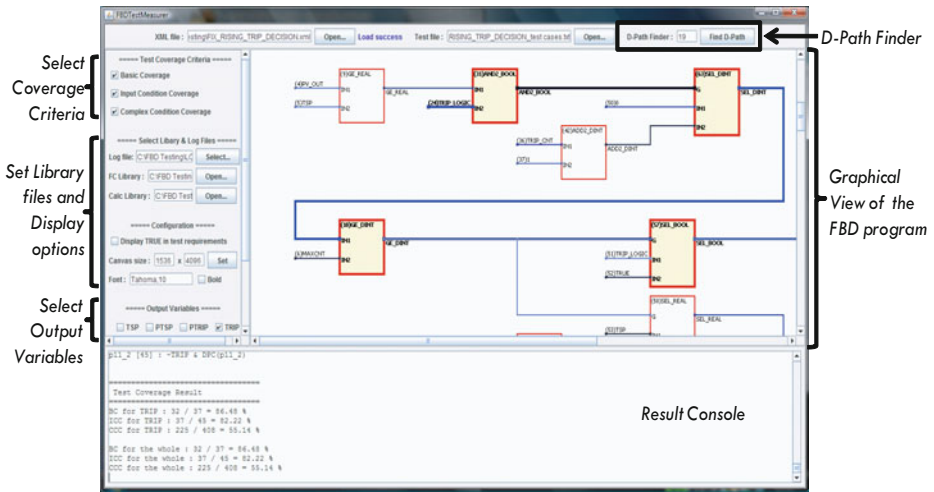


Fig. 4. Screenshot of *FBDTestMeasurer*

FBDTestMeasurer has generated uncovered test requirements, a user can identify uncovered parts in the target program clearly by inserting the number of the uncovered d-path into the D-Path Finder.

4.2 Test Coverage Measurement Issues

Internal Variables. FBD consists of functions and function blocks. If an FBD program under test consists of functions only, test requirement formulas only contain input variables. However, when the target FBD program includes function blocks, test requirement formulas contain internal variables as well as input variables. For example, $DPC(p_{3-1})$ for $p_{3-1} = \langle th\_Prev\_X\_Trip, SEL4, th\_X\_Logic\_Trip \rangle$ in Figure 1 as follows:

$$DPC(p_{3.1}) = (SEL4 \vee (preSEL4 = 0 \wedge (inT5 = 0 \vee inT5 \geq k\_Trip\_Delay))) \quad (9)$$

In the *DPC* equation (9), *preSEL4* and *inT5* are internal variables denoting the value *SEL4* stored in the previous scan cycle and the internal timer of *TON*, respectively. When we measure test coverage of a set of test cases for the FBD program including function blocks, we should track the values of internal variables as well as input variables.

There are two input variables, `f_X` and `th_Prev_X_Trip`, in the FBD program shown in Figure 1 assuming that `k_X_Trip_Setpoint`, `k_X_Trip_Hys`, and `k_Trip_Delay` are constants whose values are 95, 1, and 100, respectively. We assume that the scan time is 50ms. Table 1 shows three test cases for the FBD program in Figure 1. The test case description includes two internal variables as well as two input variables because the FBD program contains a function block

Table 1. A set of test cases for the FBD program shown in Figure 1

Test cases	Inputs				Expected output		
	Internal vars (Precondition)		Input vars		Output var	Internal vars (Post condition)	
	preSEL4	inT5	f_X	th_Prev_X_Trip	th_X_Logic_Trip	preSEL4	inT5
T1	false	0	90	false	false	true	50
T2	true	50	87	false	false	true	100
T3	true	100	85	false	true	true	100

and two internal variables involved in the internal state of the FBD program. Internal variables on inputs are considered as the precondition and internal variables on outputs are considered as the postcondition of the test cases. Every internal variable's value should be traced since they are used in the DPC computation. *FBDTestMeasurer* can deal with FBD programs including function blocks as well as functions by keeping track of all internal variables.

Loop. D-paths in FBD programs are always finite because FBD programs do not allow internal loops. On the other hand, the cyclic and infinite execution, an essential characteristic of the PLC programs, can be considered an “external loop”. We assumed that a test case is executed on a scan cycle. Testing of FBD programs containing only functions (e.g., no internal states) is straightforward. Each test case is independent from others, and the ordering of test cases is irrelevant. However, if an FBD program contains function blocks, the sequence of test cases becomes important due to internal states.

Infeasible Test Requirements. It may be impossible for a set of test cases to achieve 100% coverage for any coverage criterion because some test requirements may turn out to be infeasible. Even though infeasible test requirements do not necessarily imply FBD programming errors, such possibility is high. Analyzing causes of the infeasible test requirements can give valuable information to find programming errors or improve the logical structure of the FBD program.

5 Case Study

We applied the proposed technique to two submodules of the 18 trip logics, `_5_TRIP` and `_D_TRIP_LOGIC`, in the BP design from the KNICS project. The BP performs a core logic to determine the trip status which makes nuclear reactor stop. The BP is a safety critical system required to be tested thoroughly by government regulation authority. The BP has 190 pages of software design specification and the whole BP consists of over one thousand function blocks and about one thousand variables. The unit test report [20] for the BP consists of 139 pages and has more than 300 test cases. Testers have executed the set of test cases on the BP PLC using a signal generator.

Table 2. Submodule information and coverage assessment result

sub-module	blocks	inputs	test cases	output variable	d-paths	BC	ICC	CCC
_5_TRIP	33	15	11	TRIP	37	86% (32/37)	82% (37/45)	55% (225/408)
				PTRIP	37	86% (32/37)	82% (37/45)	55% (225/408)
_D_TRIP_LOGIC	52	23	19	TRIP_LOGIC	305	69% (209/305)	62% (232/375)	48% (1843/3870)
				PTRIP_LOGIC	1259	32% (408/1259)	28% (426/1546)	20% (3545/17540)

In the KNICS project, once a testing team finished unit testing, a separate V&V team examined and validated the testing result. The testing team experienced a problem of assuring whether they performed adequate tests and the V&V team felt difficulties in measuring the adequacy of the executed tests because there had been no readily applicable test coverage criteria and automated testing tools for FBD programs.

Table 2 shows the size information and the coverage assessment result. _5_TRIP submodule is simple and _D_TRIP_LOGIC submodule is rather complex. We chose two modules representative enough of the BP design in terms of size and complexity. According to the unit test result document [20], there were 11 and 19 test cases for each, respectively. We made no simplification on the FBD design, and we used test cases prepared by FBD testing professionals in entirety for evaluating the adequacy of the test cases. It took about 6 weeks for two skilled FBD engineers to document the FBD testing plan and to generate test cases for the whole BP system.

_5_TRIP submodule consists of 33 functions and more than 80 edges, and there are 37 d-paths for the output TRIP whose length varies from 2 to 11. Eleven different test cases, each with 9 inputs, were subject to coverage analysis with respect to BC, ICC, and CCC. The other 6 inputs of total 15 inputs are constant inputs. Test requirements for the output TRIP grew from 37 for BC to 45 and 408 for ICC and CCC, respectively. We found that five DPCs for the output TRIP were never covered, and the BC coverage measure was 86% (or 32 out of 37). When the same design and test cases were evaluated using ICC and CCC, coverage measure dropped to about 82% (or 37 out of 45) and 55% (or 225 out of 408), respectively. Coverage measurement result for the output PTRIP was same as for TRIP. Coverage achievement for the _D_TRIP_LOGIC submodule was much lower than for the _5_TRIP submodule. BC, ICC, and CCC of the test set for the output PTRIP_LOGIC was only 32%, 28%, and 20%, respectively.

Test cases derived by domain experts achieved only 86%, 86%, 69%, and 32% of the BC for the outputs of two submodules, respectively, although the definition is relatively simple. In fact, when informed on coverage measures, domain experts were surprised that their test cases failed to investigate FBD programs in adequate depth.

Visual highlighting of d-path, one of functions supported by *FBDTestMeasurer*, was helpful to detect which d-paths were not adequately tested and which d-paths were involved in making infeasible test requirements.

This case study convincingly demonstrated that the proposed idea is highly effective in revealing which logical aspects of the FBD design remain untested, assessing quality of test cases, and monitoring progress towards meeting the mandated quality goals.

6 Conclusion

We presented *FBDTestMeasurer*, an automated test coverage measurement tool for FBD programs. We have defined new structural test coverage criteria suitable for FBD programs in our previous research. Given an FBD program and a set of test cases, *FBDTestMeasurer* generates test requirements with respect to the chosen structural test coverage criteria and performs coverage assessment of the set of test cases. *FBDTestMeasurer* provides testers with the unsatisfied test requirements and also supports visual representations of the uncovered d-paths. These features help testers to find inadequately tested parts of the FBD program and to generate additional test cases efficiently. The result of the KNICS case study convincingly demonstrated the effectiveness of the proposed techniques. Our experiment revealed which logical aspects of the FBD design were not sufficiently tested by the test cases prepared by the FBD testing professionals. The domain experts found the techniques and the tool highly useful to demonstrate the adequacy of the FBD testing quantitatively and to improve it. We are currently developing automated test case generation techniques for FBD programs.

Acknowledgments. This research was supported in part by NSF CNS-0720518, NSF CNS-0721541, and NSF CNS-0720703.

References

1. IEC: International Standard for Programmable Controllers: Programming Languages Part 3
2. USNRC: Software Test Documentation for Digital Computer Software Used in Safety Systems of Nuclear Power Plants, Regulatory Guide 1.170 (September 1997)
3. USNRC: Software Unit Testing for Digital Computer Software Used in Safety Systems of Nuclear Power Plants, Regulatory Guide 1.171 (September 1997)
4. Jee, E., Yoo, J., Cha, S., Bae, D.: A data flow-based structural testing technique for FBD programs. *Information and Software Technology* 51(7), 1131–1139 (2009)
5. Mader, A.: A classification of PLC models and applications. In: *Proceedings of the 5th International Workshop on Discrete Event Systems* (2000)
6. Lewis, R.: *Programming industrial control systems using IEC 1131-3*, IEE Control Engineering Series, Revised ed. The Institute of Electrical Engineers (1998)
7. Yoo, J., Park, S., Bang, H., Kim, T., Cha, S.: Direct control flow testing on function block diagrams. In: *Proceedings of the 6th International Topical Meeting on Nuclear Reactor Thermal Hydraulics, Operations and Safety* (October 2004)

8. Richter, S., Wittig, J.: Verification and validation process for safety I&C systems. *Nuclear Plant Journal*, 36–40 (May-June 2003)
9. Baresi, L., Mauri, M., Monti, A., Pezze, M.: Formal validation, and code generation for programmable controllers. In: *Proceedings of the IEEE International Conference on System, Man, and Cybernetics*, pp. 2437–2442 (October 2000)
10. Jee, E., Yoo, J., Cha, S.: Control and data flow testing on function block diagrams. In: *Proceedings of the 24th International Conference on Computer Safety, Reliability and Security*, pp. 67–80 (September 2005)
11. Jee, E., Jeon, S., Bang, H., Cha, S., Yoo, J., Park, G., Kwon, K.: Testing of timer function blocks in FBD. In: *Proceedings of the 13th Asia Pacific Software Engineering Conference*, pp. 243–250 (December 2006)
12. Yang, Q., Li, J.J., Weiss, D.M.: A survey of coverage-based testing tools. *The Computer Journal* 52(5), 589–597 (2009)
13. Parasoft: Insure++, <http://www.parasoft.com/jsp/products/insure.jsp>
14. IBM: Rational Test RealTime, <http://www-01.ibm.com/software/awdtools/test/realtime/>
15. Aivosto: VB Watch, <http://www.aivosto.com/vbwatch.html>
16. VectorSoftware: VectorCAST/Cover, <http://www.vectorcast.com/software-testing-products/embedded-code-coverage.php>
17. Lakehal, A., Parissis, I.: Structural test coverage criteria for Lustre programs. In: *Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems* (September 2005)
18. Lakehal, A., Parissis, I.: Lustrctu: A tool for the automatic coverage assessment of Lustre programs. In: *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, pp. 301–310 (November 2005)
19. Jee, E.: A Data Flow-Based Structural Testing Technique for FBD Programs. PhD thesis, KAIST (2009)
20. Korea Atomic Energy Research Institute: KNICS-RPS-STR141 (Rev.00) - Software Test Result for the Bistable Processor of the Reactor Protection System (2006)

Overcoming Non-determinism in Testing Smart Devices: A Case Study

Peter Bishop^{1,2} and Lukasz Cyra²

¹ Centre for Software Reliability, City University, London
Northampton Square, London, EC1V 0HB, United Kingdom

² Adelard LLP
Northampton Square, London, EC1V 0HB, United Kingdom
{pgb,lc}@adelard.com

Abstract. This paper presents a case study in “black-box” assessment of a “smart” device where, based only on the user manuals and the instrument itself, we try to build confidence in smart device reliability. To perform the black-box assessment, we developed a test environment which automates the generation of test data, their execution and interpretation of the results. The assessment was made more complex by the inherent non-determinism of the device. For example, non-determinism can arise due to inaccuracy in an analogue measurement made by the device when two alternative actions are possible depending on the measured value. This non-determinism makes it difficult to predict the output values that are expected from a test sequence of analogue input values. The paper presents two approaches to dealing with this difficulty: (1) based on avoidance of test values that could have multiple responses, (2) based on consideration of all possible interpretations of input data. To support the second approach we use advanced modelling and simulation techniques to predict all the likely interpretations and check whether any of them is observed at the smart device output.

Keywords: testing, non-determinism, smart instruments, safety.

1 Introduction

The justification of smart instruments has become an important topic in the nuclear industry [1]. These instruments have operational and safety benefits as they are more accurate and require less calibration, but since they are programmable devices, there is a potential for software defects within the device, which could result in unpredictable behaviour.

Ideally some independent assessment of the development information and firmware should be performed to gain assurance that the behaviour is predictable. In practice, however, it may not be possible and independent “black-box” testing [2] may be necessary to gain confidence in the device. “Black-box” testing is based solely on publicly available artefacts—typically the user manuals (for operation and maintenance) and the instrument itself.

In this paper we describe the test environment we have developed to automate the process of test data generation, test execution and result checking. This was used to perform a range of automated black-box tests on a commercially available smart device. We found that a key difficulty in automating these tests was the *non-deterministic* response of the smart device to certain input values. We describe the strategies we have used to overcome this problem.

2 Non-determinism in Smart Devices

Smart instruments are physical devices to which we feed the input and read the corresponding output. For the purpose of this section we are assuming that a smart device:

- has an analogue input to measure some plant parameter
- has a configurable alarm, which is set on when the input value exceeds a certain limit
- has a configurable deadband, to avoid “jitter” the alarm is only turned off when the input value drops below the deadband

No matter what test environment we use to analyse such an instrument, non-determinism is unavoidable. It arises from a number of different sources that are inherent to smart devices. In particular:

- smart device accuracy
- smart device sample rates
- smart device response lags

In addition, the test harness can be a further source of uncertainty when checking the correctness of the results.

2.1 Smart Device Accuracy

Inaccuracy in the measured input value leads to a non-deterministic output result when the input value is close to some configured alarm limit as shown in **Fig. 1**.

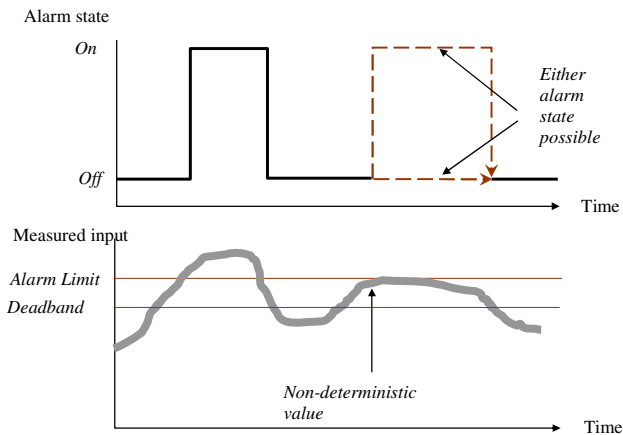


Fig. 1. Non-determinism due to smart device inaccuracy

The smart device measurement inaccuracy is represented by the thick grey line. If the internally measured value could lie on either side of the alarm limit (due to smart device measurement inaccuracy) then two alternative alarm states are possible. The test results only become deterministic when the measured value ensures that only a single alarm state is possible (e.g. when the measured value drops well below the deadband).

2.2 Discrete Sampling Intervals

A similar non-determinism problem arises if smart device measurements are only performed at discrete time points (see Fig. 2).

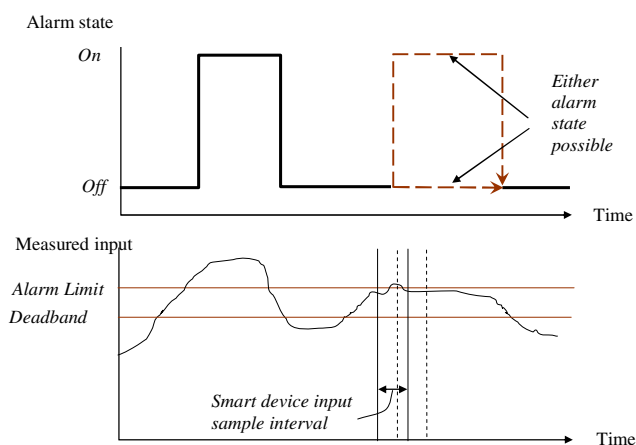


Fig. 2. Non-determinism due to discrete smart device sample intervals

It can be seen that the device will not “see” a short excursion above the alarm limit if the measurement samples straddle the excursion (solid lines) but the device would see the excursion if a sample coincided with the excursion (dashed lines). As we have no way of knowing when the samples occur within the smart device, we cannot determine which alarm state is expected *even if the test equipment and smart device were absolutely accurate*.

2.3 Smart Device Lags

Even if the alarm activation decision is deterministic, there is still uncertainty about *when* the expected response will appear at the smart device output. This is illustrated in Fig. 3 below.

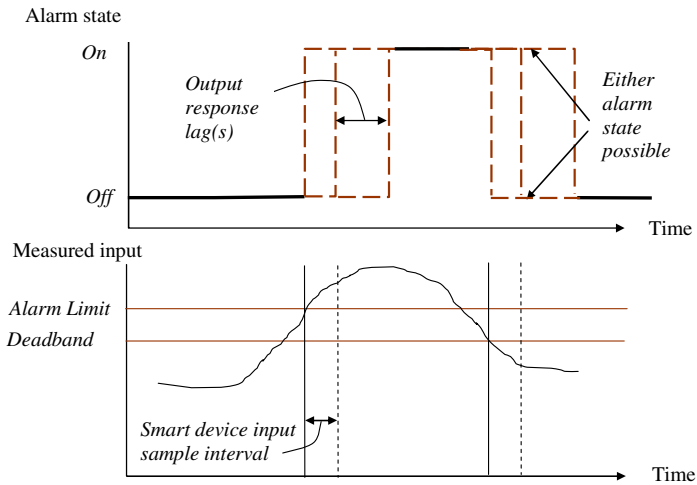


Fig. 3. Non-determinism due to uncertain response times

This arises from:

- uncertainty about when the smart device samples the input
- lags introduced by internal filtering algorithms
- uncertainty about the computation time taken to decide the alarm state
- variation in the time needed to physically switch an alarm relay to another state.

2.4 Non-determinism in the Test Harness

The test harness is also a source of non-determinism due to:

- Finite accuracy. This increases the uncertainty in the measured value as seen by the smart device.
- Lags in the test relays. Special test relays controlled by the PC can simulate events like broken wires or power interruption. It can take a finite time to activate and deactivate a test relay, and this can extend the apparent time taken by the device in responding to the event.

3 Testing Non-deterministic Systems

To test smart instruments we have to address all the types of non-determinism described above. We can do it in the following way:

- smart device accuracy – by taking one of the approaches: (1) avoidance of inputs which are “close” to an alarm limit (i.e. beyond the inaccuracies introduced by the smart device and the test harness) or (2) detailed analysis of all the possible interpretations of the inputs by the device to assess which outputs may be correct

- smart device sample rates – by holding the test value steady for a fixed “settling time” before reading the result
- smart device response lags – by holding the test value steady for a fixed “settling time” before reading the result
- test harness – by allowing for the operation of test relays in the “settling time” (lags in the test relays) and taking the steps mentioned in bullet 1 (finite accuracy)

Avoidance of inputs which are “close” to an alarm limit should, in theory, ensure that the smart device will respond in a deterministic way where each input measurement is interpreted as being definitely above or below the decision point. So we should be able to predict the alarm output values for each.

However this restricts the scope for realistic testing and may make the detection of some bugs impossible, e.g. a device may enter erroneous state after receiving a value which is interpreted as equal to the decision point. If we wish to simulate typical plant operation we should allow *any* realistic sequence of input values, including

- continuously changing values
- inputs close to the alarm limit

If this is allowed, then analysis of the smart device outputs becomes much more difficult. The test result checker has to:

- identify when input values can result in different responses (given the uncertainties in accuracy and timing in the smart device)
- maintain different execution threads for the different input interpretations
- accept a set of output results that agree with any current thread.

This mechanism is illustrated in **Fig. 4**. If the actual result does not correspond with a given thread, the thread is discarded. If all the threads are removed this way, the result checker indicates a potential error.

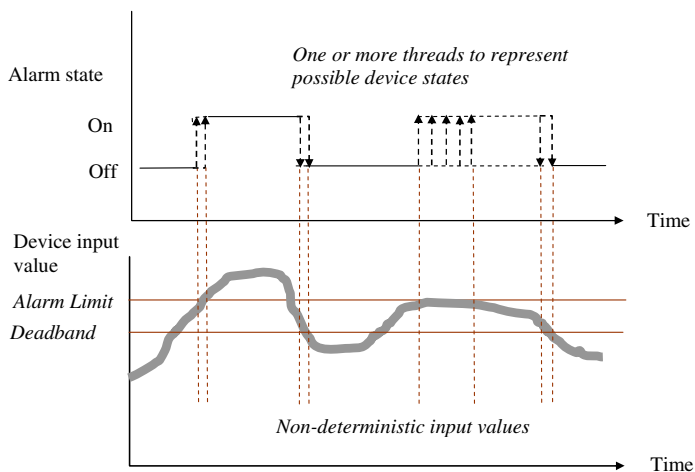


Fig. 4. Handling non-deterministic test results

The figure shows a simplified version of the problem. Smart sensors often set an alarm after a configurable delay (i.e. the input must stay above the limit for at least certain time). If we remain “close” to a decision point for some time, assuming that a delay is set, the number of threads may be thousands or millions rather than the two shown in **Fig. 4**. The complexity of the analysis depends on the frequency with which the device can change interpretation of a constant input being “close” to a decision point. We can influence the complexity of the analysis by changing this assumption and as a result obtaining more or less false positives (Section 6 gives an example).

4 Black-Box Testing Case Study

The smart device chosen for this study is a smart sensor alarm unit that can measure a plant parameter and raise an alarm if a programmable limit is violated. The smart sensor used in this study can monitor a wide range of plant measurements (via thermocouple, resistance, or 4-20mA input signals). The device controls four relay outputs to announce different alarm states.

The smart device can be configured to:

- use different plant input sensors
- trigger an alarm on high or low measured values
- avoid alarm “jitter” by specifying latching, alarm delays and deadbands
- detect rapid rates of change
- raise an alarm if internal errors are detected

The device also has LED status indicators which reflect the current state of each alarm relay. There is a push-button input that can be used to manually reset alarms that are configured to latch to the “on” state. The device can be configured via front panel pushbuttons and a display screen. It can also be programmed via configuration software running on a PC connected to the device by a serial interface.

5 Test Harness

We implemented a test harness to support functional test definition, execution and analysis of the results. The harness is composed of three parts (as shown in **Fig. 5**).

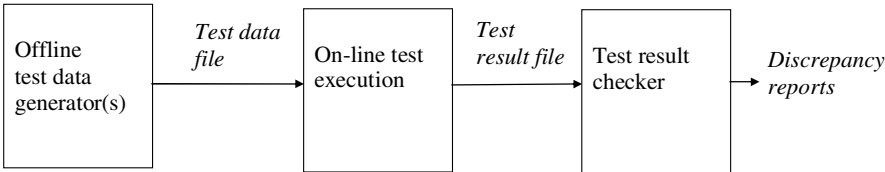


Fig. 5. Test harness elements

The harness comprises:

- An off-line test data generator. This produces test cases according to some strategy. The results are stored in a test data file.
- An on-line test execution system, which reads the test data file, applies the test to the smart device and records the result.
- An off-line result checker, which takes the test data file, computes the expected results and compares this against the result generated by the smart device.

We chose to use off-line test generation and checking because this gives us additional flexibility. If there are errors in the checker, the test result file is still valid, so we only need to fix the checker.

5.1 Test Data Generator

We developed an application for test data generation. The application generates tests using two different strategies:

- Statistical tests. This is an approximation of plant transients where the input increases to some limit with random fluctuations. An example of a transient is depicted in **Fig. 6**.

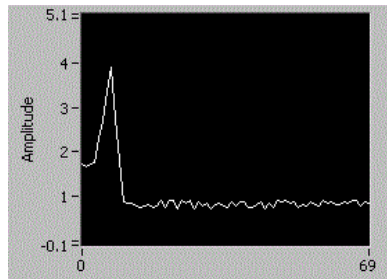


Fig. 6. Simulated transient test sequence

- Random tests. These are designed to maximise changes in the alarm outputs. An example is depicted in **Fig. 7**.

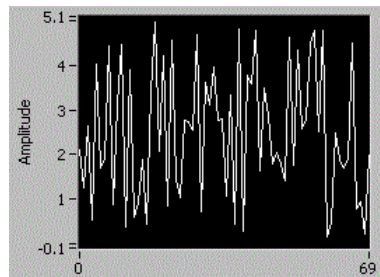


Fig. 7. Random test sequence

5.2 On-Line Test Execution System

The on-line test execution system comprised:

- a PC with analogue and digital interfaces connected to the smart device
- a LabVIEW program to read the data file with test cases to the smart device and record the results

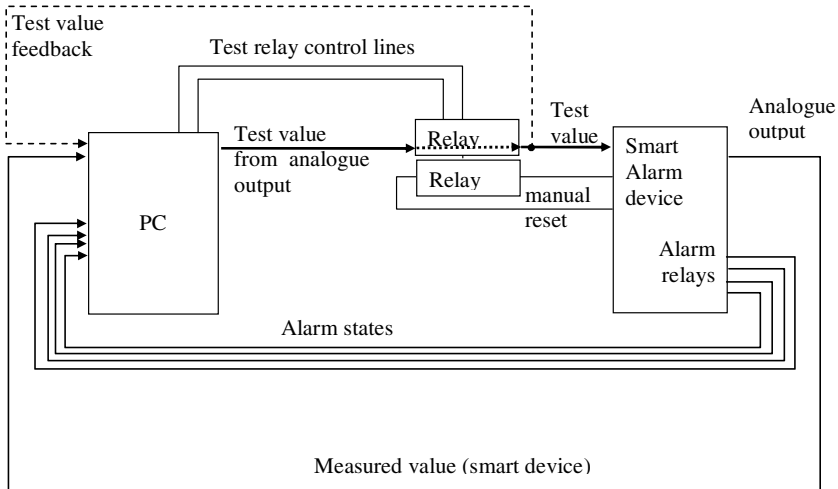


Fig. 8. Test harness schematic

The test environment (shown in **Fig. 8**) includes control of test relays to simulate broken wire conditions, and the operation of a manual reset button. We were also reading the output of the smart sensor to compare it with the test input and understand if differences observed can influence interpretation of the results. We tried to read back the test signal generated by the analogue interface by feeding it back to an analogue input so that it could be read by the PC. This however, affected the accuracy of the other analogue interfaces. Therefore, after doing some experiments we decided to remove this connection.

The test execution system is composed of:

- a personal computer, which runs a LabVIEW application controlling test execution and another application for configuring the smart sensor
- a card with relays extending the PC interface (InLog PCX-4288)
- a data acquisition device extending the PC interface (National Instruments USB-6008/6009)
- a smart sensor undergoing tests
- an analogue lab board providing voltage source for testing the state of the smart sensor

In this configuration, the test signal error was about $\pm 10\text{mV}$ (around 0.2% of the full scale value).

The hardware was controlled by two Microsoft Windows application written in LabVIEW [3]. We developed:

- a manually operated test-bench application
- an automated test execution application

The test-bench application was used to check out the hardware of our test execution system. It has a knob to adjust the input, diagrams, indicator lamps presenting the state of the alarms, and switches to simulate “broken wire” and pressing the reset button of the smart sensor. It also has four charts which display a diagram with the output from the alarms.

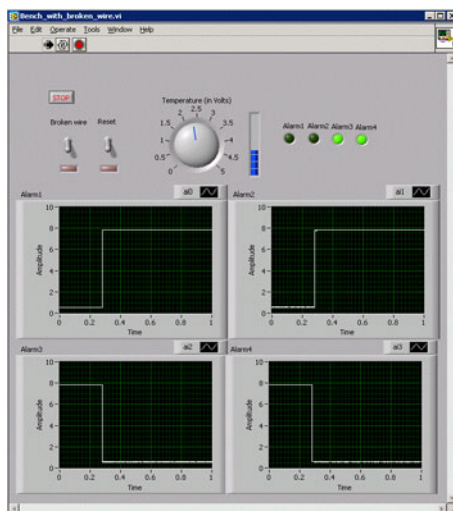


Fig. 9. Test-bench application (GUI)

The test execution application performs a set of pre-defined tests stored in a file created by the application for test data generation. The application has the following functionality:

- It reads input from a data file (i.e. the voltage for the analogue input of the smart sensor, the state of the reset button and the state of the “broken wire” relay).
- It provides input in a loop to the device acquisition card and sets the state of relays.
- It reads output (i.e. the state of the analogue output and the state of the alarms) in a loop and writes it together with the corresponding input to a file.

The model of the application is given in **Fig. 10**.

Execution of one test takes 2s or 6s depending on whether a reset is part of the test. We found that we needed 2s for simulating the input because with shorter delays (e.g. 1s) behaviour of the smart sensor was unpredictable, i.e. many errors were recorded which could not be reproduced with the delay of 2s. We needed an additional 4s for simulating the reset button because the relay card we used is slow and with

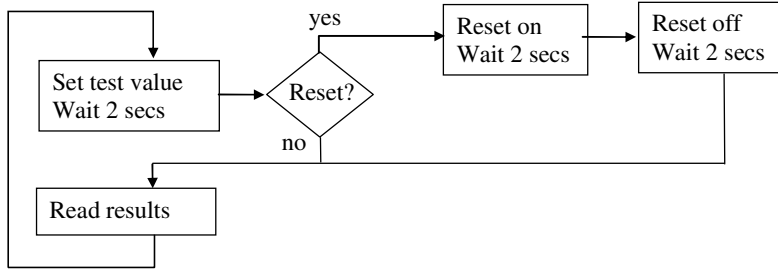


Fig. 10. Test execution application model

shorter delays it would be hard to be sure that the reset operation was simulated properly. A timing accuracy of 2s is acceptable for the device we have been analysing, but we plan to perform a more accurate response time analysis in subsequent research.

Apart from these two applications, we used another one developed by the manufacturer of the smart sensor in order to configure the smart device via a PC and read the state of the process variable of the device.

5.3 Test Analysis

We implemented an off-line test results checker (an oracle) as an application. The result checking process is identical no matter which test data generation procedure is used. It takes the test result file comprising a sequence of smart sensor inputs and outputs and then checks if they agree with the internal model of the device built into the application. The model is a finite state machine defined based on the behaviour specified in the user manual. As the documentation is not precise, the model must try to take the weakest assumptions possible, i.e. permit all behaviours which do not conflict with the manufacturer's documentation. If the outputs of the oracle and the actual result disagree, the checker generates a list of potential errors.

The result checker is highly configurable. It specifies the configuration of each alarm relay (e.g. type, limit, deadband, latch mode, and relay delay). Its interface is shown in **Fig. 11**.

Given the input data, smart device configuration and delay data, the oracle can compute the result and compare it with the actual result. This is not as straightforward as it may first appear, as discrepancies can be generated if the expected result is not unique. Section 2 discusses the possible sources of non-deterministic results. The result checker has to identify all the possible interpretations of the inputs, and based on this data calculate all the possible states of the modelled device. If any of the threads has output values that match the test result, the result is accepted. If none of the states resulting from the inputs corresponds to the test result, a discrepancy is identified.

Our non-deterministic model initially assumed that the test value measured by the smart device is unchanged between test input changes. However, by monitoring the smart device display, we established that the measured value does in fact differ on successive internal measurements made by the device. This inherent variability due to

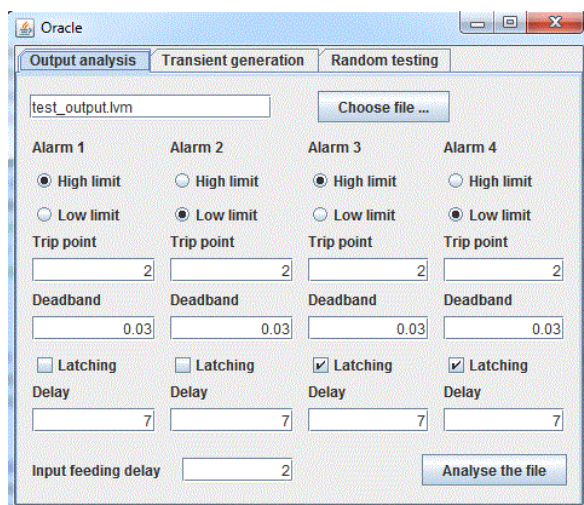


Fig. 11. Result checker interface

periodic internal sampling had to be included in the non-deterministic result checking model by computing new threads at each potential sample point (using a configurable sample rate).

6 Testing and Test Results

We applied three test strategies to derive test cases for the device:

- Statistical testing. We generated and executed 4600 simulated plant transient test sequences. For realistic plant transients, this should give 99% confidence in 10^{-3} failures per demand if no discrepancies are detected by the result checker. Execution of one set of transients takes 8 days of continuous execution by the testing harness.
- Random testing. We generated and executed 10 000 random tests. Execution of the tests took 6 hours.
- Negative testing. We tested the device with:
 - out of range inputs (20% above and below the accepted range)
 - valid, equivalent inputs and checking for variation in the output
 - testing functional independence (i.e. where changes in the device configuration influence unrelated functionality)
 - random testing (this time addressing independence and consistency of the behaviour of alarms)
 - testing based on transients (this time addressing independence and consistency of the behaviour of alarms)

Execution of the tests led to identification of a few potential discrepancies. Statistics about the detection rates for each of the strategies are presented in Table 1.

Table 1. Discrepancy detection rates

Test strategy	Number of discrepancies for a sampling rate			
	non	1/sec	2/sec	30/sec
Statistical testing	25	13	10	10
Random testing	0	0	0	0
Negative testing	1	1	1	1

The table shows the number of discrepancies detected for each of the strategies in the function of the assumed device sampling rate. We can see that the number of discrepancies detected falls when we assume that the device may change interpretation of a constant input. The number of discrepancies becomes constant once the sampling rate is increased to 2/sec, which may correspond to the actual sampling rate of the device.

For each of the discrepancies detected we were able to identify the corresponding inputs and outputs and present them in the context of adjacent inputs and outputs using our analysis tool. An example is shown in **Fig. 12**.

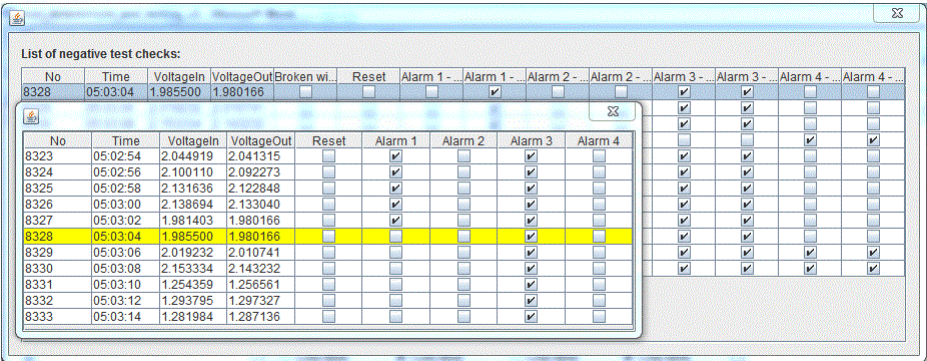


Fig. 12. Discrepancy analysis

Analysis of the identified discrepancies revealed that all of the discrepancies occurred at values close to the configured alarm limits. These discrepancies can be explained if we change some the assumptions made about the behaviour of the device. Some of the potential explanations are given below:

- We had assumed that all alarms use the same interpretation of the process variable. The data captured seems to suggest however that each of the alarms may interpret the input independently, e.g. an input value close to the limit may be interpreted by one of the alarms as being above and by another as being below the limit.
- Our test automation system assumes 2 seconds is sufficient time to wait for a stable response, however, for a small fraction of inputs, it seems this delay is insufficient.
- We might have assumed too small a range around a decision point where the interpretation of inputs is considered to be non-deterministic.

7 Discussion

From our case study it is clear that black-box testing is a non-trivial task. The number of discrepancies detected is very dependent on the accuracy of the model checker and its coverage of all sources of non-determinism in the expected result. We have independent evidence for believing that the smart device does in fact implement its alarm logic correctly, so the residual discrepancies are likely to be due to other sources of non-determinism that have not been included in the model checker.

Clearly we can continue to refine the model checker of smart device behaviour to reduce the discrepancies even further, but we need to consider whether this is a realistic approach. Effectively, we are inventing a smart device design that can explain the observed discrepancies, but in a black-box context, we can never be sure the inferred design is actually valid.

What we really need is a result checker that does not need to know about internal details (such as the sampling rate of the device) but is based solely on the published behaviour. One possible avenue that we intend to explore is the idea of defining a set of “invariant behaviours” derived from the public documentation, e.g.:

```
Test_input >> alarm_lim → Alarm=ON
```

```
Test_input << alarm_lim_deadband → Alarm=OFF
```

Note that the `>>` operator is used to denote “definitely greater given the published inaccuracy, and the condition has remained true for the smart device’s published maximum response time”.

If we do this, we have to accept that there are points in the test sequence where none of the invariants will apply and the model-checker simply has to ignore the results generated—discrepancies can only be checked when the invariant condition is active. There could also be difficulties with this approach because it assumes test points where the answer is unique, i.e. the possible states of the smart device converge to a single state. This may not always be true. For example:

- An alarm is configured to latch permanently if the limit is exceeded
- The test sequence rises up to the limit then falls

In this case there are two possible responses so the smart device states will diverge for any subsequent test values that stay below the alarm limit. A more complex expression of expected behaviour over time would be needed like invariants expressed in some form of timed temporal logic [4]. We hope to investigate the feasibility of this approach in further research.

8 Conclusions

The paper presents a case study in which a test harness for a smart device was developed and used to execute a set of black-box test cases defined using several test strategies.

We showed that non-determinism of the smart device response makes the testing significantly more difficult. We have sought to address this problem by designing the result checker to explicitly allow for non-determinism. When a potentially non-deterministic test condition is detected, the result checker creates different

“threads” representing alternative potential states of the device. Normally the threads recombine at a later point in time, when the input leaves the area “close” to limits because the threads will converge to an identical internal state. For example, after a certain amount of time above a trip limit, all threads will be in the alarm state.

This strategy has proved to be quite successful, but there are still some residual discrepancies that we suspect are false positives. These could probably be removed by increasing the complexity of the non-deterministic model of the device, but it is difficult to justify the model given that the device is a black-box.

In the next stage of research we plan to investigate less strict forms of model checking where we ignore all test cases where there is any uncertainty about the expected results.

Acknowledgments. The authors wish to acknowledge the support of UK C&I Nuclear Industry Forum (CINIF) who funded the research presented in this paper.

References

1. Bishop, P., Bloomfield, R., Guerra, S., Turlas, K.: Justification of Smart Sensors for Nuclear Applications. In: Winther, R., Gran, B.A., Dahll, G. (eds.) SAFECOMP 2005. LNCS, vol. 3688, pp. 194–207. Springer, Heidelberg (2005)
2. Alur, R., Henzinger, T.: A Really Temporal Logic. *Journal of the ACM (JACM)* 41(1), 181–203 (1994)
3. National Instruments, LabVIEW, <http://www.ni.com/labview> (visited on 24/02/2010)
4. Krichen, M., Tripakis, S.: Black-Box Conformance Testing for Real-Time Systems. In: Graf, S., Mounier, L. (eds.) SPIN 2004. LNCS, vol. 2989, pp. 109–126. Springer, Heidelberg (2004)

Software Testing by People with Autism

Suzanne Haanappel and Sjaak Brinkkemper

Department of Information and Computing Sciences, Utrecht University,
PO Box 80.089, 3508TB Utrecht, The Netherlands
{s.haanappel,s.brinkkemper}@cs.uu.nl

Abstract. Having a regular job is of great value for people with autism, but acquiring a regular job is not that easy. People with autism do not only face prejudices when applying for a job, they also have social impairments that make this process very difficult for them. Characteristics of people with autism are that they have eye for detail and enjoy repetitive work. These characteristics are characteristic of software testing, seems to make people with autism highly suitable for software testing. From our research it shows that people with autism possess the right qualities to become a test engineer. Companies employing people with autism acquire test engineers with the same level of skill as people without autism and additionally they also practice corporate social responsibility.

1 Introduction

It is widely known that ICT has a major influence on the lives of people today; we use it to communicate, to find information etc. ICT is a specialism that focuses among others on information systems, computers and telecommunication. Developing ICT systems involves software and on software we will zoom in during this research. The wide range of ICT applications cover makes it important that the quality of used software is high, otherwise many processes run would not be executable anymore. According to research done by the US National Institute for Science and Technology in 2004, unreliable and poor quality software was costing Europe billions every year [1]. To decrease the costs companies have to make because of bad software, more and better software testing should be executed. Software testing is an important aspect of the software development cycle, that enables the development of high quality software by for example finding faults in the software. A simple solution to the problem would for organizations to perform more software testing, but unfortunately this is not always possible.

The software testing phase is not often perceived as the most important phase of the software life cycle this because of many reasons, a few examples are; budget reasons and time-frame reasons [2]. Some of the software testing characteristics that make it less favorable are the repeatability and predictability of the job. People without autism soon find software testing not challenging enough for them anymore, but for people with autism these characteristics make it a perfect job. In this case study we will present the possibilities of software testing by people with autism. For this we will use information provided by the

Autest foundation. Autest is a Dutch non-profit foundation whose objective it is to let people with autism test ICT-systems [3] and to provide companies with high quality testing services [4]. These testing services however currently exclude testing services for technical and embedded software.

We will start with introducing the two concepts on which this paper is based; autism and software testing. After that we will explain why regular jobs are so important for people with autism and the Autest initiative that originated from this. From there on we will move forward to the educational arrangements that have to be made to enable people with autism to become software tester and some experiences. We will finalize with a conclusion where we include the lessons learned.

2 Autism and Software Testing

2.1 Autism

Autism was first categorized as a disease by Leo Kanner in 1943 [5] [6]. Before this, children with autism were merely thought of as having schizophrenia. Kanner states that although there are similarities with schizophrenia autism differs in many cases from the known cases of childhood schizophrenia. His paper called *Pathology* describes his research with different children who appear to suffer from autism. It provides us with a detailed description about what characteristics of autism can look like, because there are many differences between the characteristics and they do not all seem as severe on different children [7].

Autism is defined in the international classification of diseases as a pervasive development disorder with onset before the age of 3, which involves deficits in social relationships, communication impairments, repetitive behaviors and restricted interests [5] [8] [9]. Around 0.6% of the population has this lifelong disorder [9].

Autistic disorder is the most severe of the neurodevelopmental disorders referred to as autism spectrum disorders (ASDs), which share the common feature of dysfunctional reciprocal social interaction [10]. The symptoms fall over a continuum from severe to milder forms of the disorder, which explains the term autism spectrum disorders [8]. ADS covers several clinically defined conditions; pervasive development order not otherwise specified (PDD-NOS), autistic disorder and Asperger syndrome [10].

In the research of [5] an overview of the impairments of people with autism is described, examples are: social impairment, difficulties in grasping emotions and understanding implicit rules and social conventions as well as problems with generalization and poor adaption to change. Consequences on the execution of regular work are difficulties in social intercourse and cooperation, this because people with autism not always know what the appropriate response to a situation is. Another consequence is that the employer has to provide a suitable working environment and also provide the people with autism with clear rules and guidelines, because implicit rules are not understood.

2.2 The TMap Test Method

Test Management Approach (TMap) was published in 1996 as a revolutionary management approach for structured testing. It took only a few years for TMap to become adopted worldwide by companies searching for a structured way to improve their information systems [2]. It currently is applied in hundreds of companies and organizations, including banks, pension funds and government organizations. The strength of TMap comes from the practical experience that is basis for the method [2].

Software testing has several characteristics and one of them is the structure of the tests. TMap is an example of what we call a structured testing approach [2]. One of the benefits of this approach is that it can be applied in many situations, regardless of the client or system development approach that is used [2]. Additionally TMap allows for recognition of defects at an early stage and prevent defects and also the test process is comprehensible and manageable.

Characteristics of this structured testing approaches are that the structure provides full insides into what has to be done by whom etc. Secondly the full scope is covered, there are concrete footholds and test activities are managed in the context of time, money and quality [2]. The TMap test method consists of four so called essentials, which will be explained below [2]:

1. Business driven approach
2. Methodical
3. Support tooling
4. Adaptiveness

Since it is not possible for companies to test aspect of their applications, a balance has to be found between what to test and what to leave outside the testing scope. Business driven means that the choices that have to be made to decide what the testing scope will be depend on; risks, results, time and costs. Mostly there is a business case at the basis of an IT-project, providing a scope or a short explanation about what the scope should contain. TMap assumes that the project approach is based on a business case. Some of the characteristics of this business case are that the approach focuses on achieving predefined results. To receive these results the costs have to be similar to the benefits that will be achieved. The TMAP test method also has a structured form for the test process. In TMap the overall process consists of several processes, examples are the master test plan, acceptance and system tests and developmental tests. The master test plan provides insight into the various tests and the evaluation levels that are going to be used. The test plan is used to optimize the total test process. The test process consists of two phases: the planning phase of the total test process and the control phase of the total test process. Additionally TMap contains a complete toolbox. This means that several tools are offered during the execution of the structured test process to enable the activities. The tool box focuses on three subjects: technique, infrastructure and organization. The techniques used enable different how tests can be done. Examples of techniques are; test estimation, test design and product risk

analysis. The infrastructure enables the testing to be done by providing a test environment where tests can be executed but also tools that can support the testing process. Finally, organization is about who does the testing. TMap is an adaptive test method because it can be applied in a variety of test situations and the approach can be combined with many types of system development methods. Test engineers can select the elements of TMap they want to use when performing tests. Additionally, it has to be mentioned that the flexibility of the method allows for usage in changing environments. TMaps adaptability can be summarized in four properties: (1) respond to changes, (2) (re)use products and processes, (3) learn from experiences and (4) try before use.

3 People with Autism and Regular Work

It is known that there are many people with autism who are unemployed. Apart from the enrichment of their lives when they would have the opportunity to work, this would also be an enrichment for society, because otherwise these people stay at home and receive social securities. Employing people with autism provides them with structure and regularity and makes them feel of use in today's society. When being able to actively participate in society and receiving income, it makes them feel better.

To analyze what type of work is suited for people with autism the ideal situation would be to ask these people themselves what they are good at, however this is a very time consuming task. In the research of [5] this is already done and a comparison is made between this self-reported skills and experimental data to analyze the validity. From this comparison it showed that the reported skills corresponded to the experimental data, implying that the people with autism state skills that according to scientific research are what they are good at [5]. From the majority of people it was reported that they have exceptional abilities in one or several specific domains including; music, the animal world, mathematics, reading memory, meteorology, computer science or foreign languages [5].

Characteristics of people with autism are that they have some limitations in the area of social skills and communicative abilities, implying that they need some additional support, guidance and clear patterns. Qualities are that they are observant, have eye for detail and can work highly focused. This makes them highly qualified for software testing, because of the repetitional aspect of the job that provides structure and predictability [3].

3.1 The Autest Initiative

One can imagine that it is hard for people with autism to function in everyday society and that it can be even harder for them to find a regular job. For people with autism, regular work can be an enrichment of life, but still there are only few companies who work with people with autism. People with autism encounter difficulties that make finding a job more difficult examples are the invisibility of the handicap and the lack of support from employers. There are many prejudices about why working with people with autism is difficult, for example that they

are to difficult to communicate with, have difficulties with social interaction and are difficult to make head for. Although these prejudices apply to a part of people with autism, these drawbacks can be easily overcome by some additional training in the employing company and some adjustments to the work environment. Some of these adjustments can be a sufficient amount of peace and quiet, no stimuli and no disturbances.

Autest is a Dutch non-profit foundation founded in 2005. The objective of the company is to be sustainable responsible business that allows people with autism to execute commercially based testing services. Autest provides their employees with a structured job in a safe environment within Autest or located at one of their customers [4]. During 2008 and 2009 Autest trained 11 people from which 8 are currently still employed.

The mission of the company is [4]:

1. Limiting the worries that companies can have towards the quality of the to be developed system by offering high quality testing services
2. Letting people with autism execute these testing services, because they are equipped with the skills and they are trained to perform these type of activities
3. As a corporate social responsible company, providing in a structured way a safe and secure working environment for people with autism who want to be a test engineer

The process of finding and training new employees starts with finding people with Autism who are interested in becoming a test engineer. Sometimes these people are found by Autest, but the majority of people subscribe through the website of Autest. Van Melis explains that Autest needs people with at least a Bachelor degree thinking level and who have an interest in ICT. People who meet these demands are invited for a first interview. There are several interviews to find out if the person has the skills and qualities needed for the job. Since people with autism do not respond well to exams, Autest cannot let them make several tests to see if they have the right amount of skill and potential. If a potential employee is approved they are admitted to a two month course at the company Ordina, the training provider.

Before Autest starts educating new employees they select organizations where people with autism can be positioned when they have finished the internal education program. This is make sure that the potential employees will have the guarantee that there is a job waiting for them, once they have finished the program. Companies where people with autism can be placed are mostly large-scaled companies. These companies have the resources to provide (adjusted) work environments and provide the right guidance to these people and their special needs. When a company wants to hire test engineers from Autest it is determined whether or not a company can position one or more employees of Autest. This happens based on several conversations with both parties to make sure the company is really suitable to place some of the Autest employees.

The training provider provides a course where the employees are introduced to the TMap testing method that will be used when they are execution their test

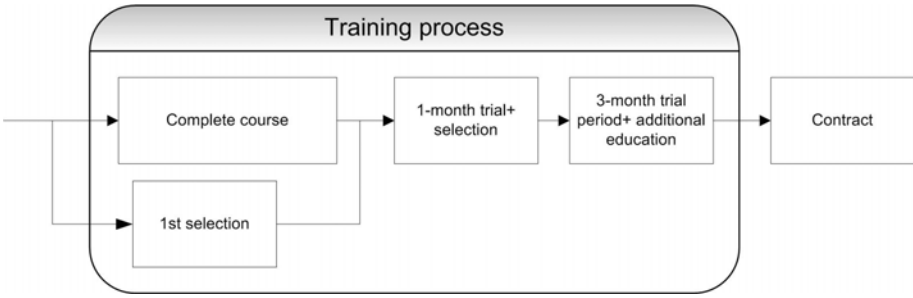


Fig. 1. Training Process

engineer function. The training provider has a selection process for the students to see once again if the students really are suited to become a test engineer. This is a subjective selection process where it is analyzed if students are seriously interested in being a test engineer and if they fit the job description, additionally it is analyzed if they have the right knowledge level for the training. After the students have completed the course they will have a trial period of one month with another selection. If a student also completed this trial period they will enter another trial period of three months where they get additional education. When this period is over they will receive a contract. The training process is visualized in Fig. 1.

3.2 Job Arrangements

Most applications the employees of Autest test are business applications in the financial industry, banks and insurance companies. These applications mostly are large-scaled which provides long term testing engagements for the employees of Autest. So far Autest does not provide testing services for technical and embedded software. Autest closely collaborates with the employer of the people with autism, the relationships that exist between the different parties are shown in Fig. 2.

The overall picture consists of three parties; Autest, the employee and the company where the employee is positioned. The employee communicates with his jobcoach, an expert on Autism, about things he needs of experiences within the company where he works. The jobcoach and manager of the employee communicate to make arrangements the employee needs to create a healthy working environment, for example additional guidelines about procedures within the company. Additionally the employee also communicates with the manager and his team within the company, this is work-related communication. Communication within the team can be about the creating a division of the jobs at hand and communication with the manager can be about the functioning of the employee within the company.

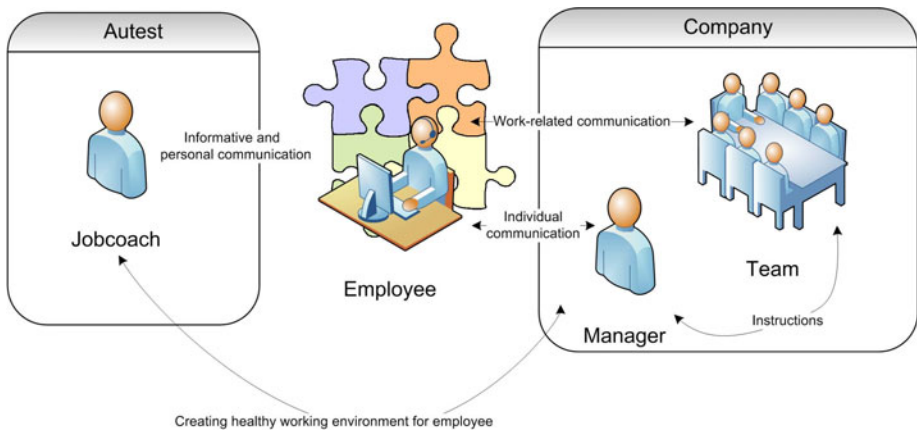


Fig. 2. Working relationships

4 Educational Arrangements

As stated above, Autest has established a software testing training program with the company Ordina. Ordina is a company that as Autest practices corporate social responsibility. The projects in which the company engages result in profit i.e. like an increase of knowledge and empowerment of employees that Ordina want all parties to use to their benefit. It is this corporate social responsibility practice which stimulated Ordina to collaborate with Autest. Together they enable companies to do something for society.

4.1 The Course

The test method that is thought is the TMap method that we introduced in 2.2. The students are thought the TMap method, because the customers of Ordina use TMap to test their software. Therefore it is important that the students are familiar with the terminology and the techniques used. TMap is suitable because the method provides the test engineer with everything needed to perform a structured test process. Using a roadmap the engineer starts at the beginning and finishes with a test script as end result. Ordina and Autest have chosen the TMap method, because it is the Dutch standard for testing.

The training provider therefore offers the students a course when they are thought everything they need to know about TMap, which will result in a certificate that verifies their knowledge. This certificate is the same certificate that people without autism receive when they have completed this course at the company.

4.2 Course Sessions

During the course the theory is always presented and discussed in the sessions. Course material is carefully dosed, there are never too many slides and it is

checked during the lesson if the level of speed is not too high for any of the students. After the theory is introduced the assignments are introduced and handed to the students. The students individually work on assignments, but they are allowed to ask questions to the lecturer. The lecturer walks around the classroom to check on the progress of the students and answer questions if necessary. When the assignment and theory are not finished on the day they are given they will be finished during the next lecture. There are seven guidelines that structure the sessions which will be introduced below.

1. Availability of autism expert

Working with people with autism requires experienced people to guide them. During the sessions there are several measures applied by which the training provider supports the students and their special needs. First of all there is a job coach who is expert in the area of autism. The job coach is provided by Autest and stays in contact with the students and the trainers to make sure that everything is correctly arranged for both parties. Additionally there is one day in the week where the students take a training aimed at developing better social skills.

2. Social engagement norms

At the beginning of the course some group rules are agreed upon. These rules are printed out and put up in the classroom. An example of one of the rules is: Raise your hand if you have a question. The rules are evaluated with the job coach and refined if necessary. It is also possible that a student need some additional personal rules, which are than agreed upon with the student.

3. Lecture content and structure

The emphasis is on the answering and discussing of the assignments, less on the sheets and the story accompanying these sheets. Also the amount of information provided per sheet is limited for the students. This to make sure that the content of the sheets can be presented under the most optimal conditions.

4. Level of process

The level of process is adapted to the slowest student in the group. Faster students are provided with additional assignments after this has been discussed with the job coach. An effort is made to make sure that all the students can handle the level of process., when students are insecure they are provided with much positive feedback. There is a scheme, but the trainers can change it if that is what the group prefers as long as the changes are in line with the rules of the course.

5. One form of explanation

All lecture content is explained in one way to the students. When the explanation is unclear the same explanation is provided again, this to prevent ambiguity. If the explanation still is not clear the student can have a one on one explanation where the subject is explained using another explanation or example. This additional explanation is not given to the whole group, because it showed in

the past that other students could become confused because of this additional explanation.

6. No jokes

Jokes are avoided and when a joke is made it is emphasized that it was a joke to avoid ambiguity. This clear indication that a joke was made keeps the atmosphere calm. People with autism are known for not being able to handle jokes as they take all that is being said literally.

7. Distribution of materials

And finally all the materials are handed out chapter by chapter, so the students cannot take a look at what is going to be explained during the next few lessons. Therefore all the materials the students have in their possession are already discussed.

5 Experiences

Experiences about software testing by people with autism are only available in a qualitative way. There is no data available, because of two reasons. First of all it is a sensitive subject to be making a comparison between people with and without autism. This makes the gathering of this data rather difficult. Additionally Autest is a small organization that is only in their start-up phase, therefore there has not been a possibility to gather data yet.

5.1 A Personal Account

The approach of Autest where people with autism are enabled to become test engineer seems to be bearing fruit. Employees on average stay two years with the employer where they execute testing activities. One of these employees is Eelke Pothoven. Pothoven is positioned by Autest at UWV and is working there as test engineer for over 1.5 years. UWV is company that pays employee insurances. The job description of Pothoven consists of the development and execution of test specifications and the controlling of existing test suites.

Pothoven agreed to tell us about his work at Autest and the UWV and about his autism. The results of this conversation are meant for the analysis of how people with autism experience working at a regular employee and to see if there are similarities with previous set characteristics of people with autism.

During the interview Pothoven told us that he did not only executed testing activities at Autest, but also at another company called Alstom Transport. We will use this company to make a comparison between the different situations and see what the preferences of Pothoven were. Alstom Transport was small scale in comparison to the UWV and the activities Pothoven had to perform were closer to his interests. This small scale design took away the feeling of anonymity Pothoven experiences at the UWV. Pothoven had the possibility at UWV to be placed between his colleagues without autism. This took away the separation between people with and without autism, something that Pothoven

prefers. He admits that this is a personal preference that not all people with autism may share.

During the interview we noticed that Pothoven is very detailed when explaining about his activities at UWV. Pothoven is aware of this level of detail, because it is one of the characteristics he mentions when talking about people with autism. People with autism have the ability to detect more and smaller mistakes than people without autism. During software testing this can be an important benefit. These mistakes also can cause a loss of focus. Pothoven tells us that his work rate is not as high compared to his colleagues without autism and that he sometimes loses overview. Pothoven is able to find mistakes as described by his employee, additionally he says that he notices much more than just what he needs to find. All different types of errors he will just notice, an example is errors in a human interface, for example a textarea that is not consistently placed throughout the different screens. As all test engineers who had their education with Autest they know what type of software errors there can be. The efficiency of detection of errors as Pothoven said sometimes is somewhat lower than his other coworkers because he does not notice or is able to just find the types of errors he is ordered to find, but also different types of errors. There is however no data available in scientific literature or from his employer to validate in which ratio this is. To solve this last problem the UWV developed a roadmap. This roadmap was already in use at the test centre, but was put on paper for the Autest employees. Together with a schedule provides clarity and structure for tasks.

5.2 Comparison between Participants with and without Autism

The course lecturer tells us that the level of the course on some aspects can be higher, but on average is lower. Students may for example have a better memory, but the overall speed of processing is slower. Additionally the students work lesser hours a week, because of the stimuli they perceive in everyday life. It shows that it takes students with autism longer to get the concept that is explained. Sometimes students have to translate it to their way of thinking and back to get it right.

It is also explained that working with students with autism it not always that easy. The students autism forces her to really think about the structure of the concept thought. But there are also more difficult aspects, for example dealing with the group dynamics. Sometimes tense situations arise quite suddenly which have to be resolved. Additionally the lecturer has to be strict about the rules, which sometimes is hard. Some of the students even have concentration problems, so it is possible that one of the students falls asleep during the explanation.

6 Conclusion and Future Outlook

We have analyzed software testing by people with autism from different perspectives and throughout this analysis it showed more and more that people with autism are right for the job of software tester, because they have the right

qualities to become a test engineer, like having eye for detail and being able to find more mistakes than people without autism.

Necessary changes to work environments do not have to be seen as drawbacks, because they are only minor investments for companies. Working spaces will always have to be provided and when a company decides to work with people with autism, additional changes are part of executing a corporate social responsible practice.

An example of drawbacks to software testing by people with autism is that they are more easily distracted and are fewer hours available. The first drawback of being distracted is a problem that can be easily overcome by introducing guidelines employees with autism can use to stay focused. The second drawback we mentioned was that people with autism have fewer hours they are available to work. A company therefore would need more employees with autism in comparison to people without autism to reach the same goal. However one could also research if there are possibilities to use IT to let these people work from their home. Additionally it is easy to contact these people when they are working at home when something goes wrong, because you can use for example chat or videoconferencing to discuss problems. This drawback need further research to make further claims about what companies should do, because it can be that having more employees with autism results in better testing results. Additionally companies should consider that employing people with autism can also be part of executing a corporate social responsible focus, meaning that sometimes companies have to do something without looking just at the economical benefits. Additionally more research has to be in cooperating with organizations who have people with autism as employees to gather data on the efficiency of people with autism when comparing them to employees without autism. This would make the discussion move from the social discussion it is now, to a discussion in which organizations can also see what potential economic benefits from hiring people with autism as test-engineers could be.

With the data at we hand we think that people with autism are right for the job of test engineer and if it turns out to be that they are not the most efficient employees, companies can always think about the social contribution they are performing.

Acknowledgements

We would like to thank Hermain van Melis, Marloes van Dongen and Eelke Pothoven who made the development of this paper possible by providing us with detailed inside information. Thank you for your time and answers to all of our questions.

References

1. IEEE: Bad software cost millions. IEEE Review 50(4), 12 (2004)
2. Koomen, T., van der Aalst, L., Broekman, B., Vroon, M.: TMap Next for result-driven testing. UTN Publishers, s-Hertogenbosch (2006)

3. Nieuwenhuizen, M.: Autest zet autisten in om it-systemen te testen. *LanVision Augustus*, 30–33 (2006)
4. Autest: Homepage autest.nl (2007), <http://www.autest.nl/default.aspx> (Retrieved May 5)
5. Chamak, B., Bonniau, B., Jaunay, E., Cohen, D.: What can we learn about autism from autistic persons? *Psychother Psychosom* 77(5), 271–279 (2008)
6. Gillham, J.E., Carter, A.S., Volkmar, F.R., Sparrow, S.S.: Toward a developmental operational definition of autism. *Journal of Autism and Developmental Disorders* 30(4), 269–278 (2000)
7. Kanner, L.: Autistic disturbances of affective contact. *Nervous Child* 2, 217–250 (1943)
8. Walker, D.R., Thompson, A., Zwaigenbaum, L., Goldberg, J., Bryson, S.E., Mahoney, W.J., Strawbridge, C.P., Szatmari, P.: Pdd-nos: A comparison of pdd-nos, asperger syndrome, and autism. *Journal of the American Academy of Child&Adolescent Psychiatry* 43(2), 172–180 (2004)
9. Hill, E.L., Frith, U.: Understanding autism: Insights from mind and brain. *Philosophical Transactions: Biological Sciences* 358(1430), 281–289 (2003)
10. Abrahams, B.S., Geschwind, D.H.: Advances in autism genetics: on the threshold of a new neurobiology. *Nature Review Genetics* 9, 341–355 (2008)

Information Flow Analysis of Energy Management in a Smart Grid

Ravi Akella* and Bruce M. McMillin

Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri 65409-0350, United States
Phone: +1(573)341-6435 Fax: +1(573)341-4501
`{rcaq5c,ff}@mst.edu`

Abstract. Information flow security within the context of multilevel security deals with ways to avoid unwanted information flow from a high level domain to a low level domain. Several confidentiality and information flow properties have been formalized in literature. However, applying them to Cyber-Physical Systems (CPSs) adds to the challenge of protecting confidentiality. This paper performs an information flow analysis of a future power CPS that has complex information flow and confidentiality requirements. Confidentiality properties such as non-deducibility are applied to the infrastructure considered. The proposed approach provides a unique direction for formalizing information flow properties for such systems with inherent complexity and security requirements.

Keywords: Security, Information Flow, Confidentiality, Cyber-physical system, Non-inference, Bisimulation based Non-deducibility on Compositions.

1 Introduction

Information flow analysis of a system reveals the design and implementation issues that divulge its confidentiality. Confidentiality is usually regarded as being violated when there is an information flow from a high level domain to the low level domain. Non-interference, Non-deducibility [1] and extensions [2] are information flow properties concerned with preventing information from being downgraded through covert channels and other such potential causes. This paper extends the application of these information flow properties to more complex Cyber-Physical Systems (CPSs) which are integrations of physical and computational processes. Information flow analysis in CPSs is made more complex by inherently observable cyber and physical events, any of which may divulge confidentiality within the system.

Of particular interest are modern “Smart Grid” systems. Smart grid is a term that embraces many concepts, from smart metering, to smart distribution, to smart transmission systems [3] [4]. The economics and power management of

* This work was supported in part by the Future Renewable Electric Energy Distribution Management Center; a National Science Foundation supported Engineering Research Center, under grant NSF EEC-0812121 and NSF CSR award CCF-0614633.

such microgrids are only beginning to be understood [5] [6]. While dynamic distribution of energy and protection in microgrids involving control strategies within the system was discussed in [7], less attention has been paid to system security properties, and, in particular, confidentiality properties. This paper addresses confidentiality properties based on information flow analysis with respect to an observer being able to infer about or interfere with the actions that take place within a smart grid. Smart power meters, as a first smart grid step, are being installed in the United States to both monitor and control energy usage. Such systems are not without security concerns, however [8]. A recent MSNBC article pointed out a potential risk of this increased monitoring [9].

“Would you sign up for a discount with your power company in exchange for surrendering control of your thermostat? What if it means that, one day, your auto insurance company will know that you regularly arrive home on weekends at 2:15 a.m., just after the bars close? ”

The result from tying cyber systems with physical systems open up a new realm of the privacy and confidentiality issues. Continuing forward with regard to renewable energy resources, consider the case of two neighbors Fred and Barney who agree to each purchase a renewable resource, and then share their power output as shown in Figure 1.

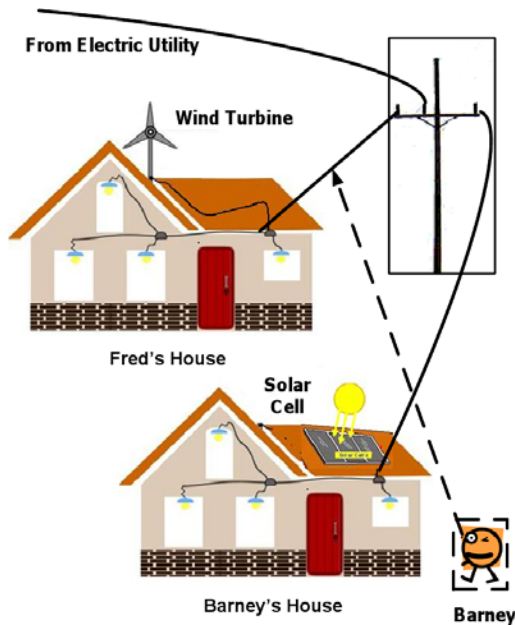


Fig. 1. A Simple Case of Information Flow

Example 1. Fred purchases a Wind Turbine and Barney purchases a Solar Panel (Photo-Voltaic Array). When the sun shines, Barney's power is used by both, when the wind blows, Fred's power is used by both. When there is excess, they agree to sell it back to the electric utility grid for a profit. The system operates well for a while and both Fred and Barney are satisfied as they enjoy reduced energy costs.

Fred, however, gets greedy and doesn't necessarily want Barney to share in these profits, so he also buys a battery. Fred now changes the operation of his system; when the wind blows, he sends the excess to his battery. Later, at a time of economic opportunity, he sells this stored energy back to the utility. Barney becomes suspicious of Fred and sneaks over and monitors Fred's power transfer to the utility (over Fred's power line coming into his house). Barney also monitors his own power line and observes that he is drawing power from the utility, Fred is not providing power, but Fred's wind turbine is spinning. From Barney's point of view, since he doesn't know about the battery, Fred's power transfer is consistent with Fred using all the power he generates. Thus, Fred's actions are completely hidden from Barney. Fred, however, monitors Barney's power, and when Barney is not drawing power from the grid, Fred discharges his battery to the electric utility, making a profit for himself. Now, if Barney observes his own power from the utility, Fred's power to the utility, and the spinning of the wind turbine, if the wind turbine is not spinning, information flows from Fred to Barney and, thus, Barney can deduce that Fred is not being honest. However, if the wind turbine is spinning when Barney observes Fred's behavior, he still cannot deduce anything about Fred's behavior. Thus, no information flows from Fred to Barney.

This simple example exhibits the complexities of observation and action of a simple physical system. Smart grid systems couple (intelligent) cyber action with physical operation, dramatically increasing the complexity of determining information flow. This paper examines a future generation smart grid, the *Future Renewable Electric Energy Delivery and Management* (FREEDM) System [10], which is a National Science Foundation(NSF)-funded Energy Research Center (ERC). The FREEDM system is a smart grid managed with a Distributed Grid Intelligence (DGI) to optimize the utilization of renewable energy generation and storage resources, to be integrated with the existing legacy grid. DGI consists of cyber processes that perform distributed computation to efficiently manage physical system resources. The FREEDM system contains subtle complexity in its combined information flow and confidentiality requirements, making it challenging to analyze the system with respect to the known information flow properties. In this paper, we uncover potential confidentiality violations within the system by performing such an analysis.

Several models by which a low-level passive observer may divulge confidentiality within the context of FREEDM are discussed in this paper. These models differ by the relative setting of the observer and the system; one in which it could be completely external to the system, one in which it is a part of the physical system and one in which it is a part of the combined CPS. A major challenge is to express the physical invariance of power flow and economics in semantics

of information flow, so that potential violations of confidentiality due to unrestricted information flow are revealed. This paper treats each of these observer models within this semantic context.

Section 2 introduces aspects of formal methods, which are extensively used throughout this paper. Section 3 presents a deeper insight into the FREEDM system and the implementation of the above mentioned distributed power balancing scheme. In Section 4, the proposed approach for the analysis of information flow is discussed. In Section 5, a preliminary investigation on how an active internal observer can manipulate the power balancing scheme for its gains, is presented. Finally, conclusions and future work are presented in Section 7.

2 Background

2.1 SPA

Security Process Algebra (SPA, for short) [2] is an extension of the Calculus of Communicating Systems (CCS). The BNF Syntax of SPA to describe the system is [2]: $E ::= 0 \mid \mu.E \mid E_1 + E_2 \mid E_1 \mid E_2 \mid E \setminus L \mid E / L \mid E[f] \mid Z$, where 0 is the empty process, which cannot do any action; $\mu.E$ can do action μ and then behaves like E ; $E_1 + E_2$ can alternatively choose to behave like E_1 or E_2 ; $E_1 \mid E_2$ is the parallel composition of E_1 and E_2 , where the executions of the two systems are interleaved, $E \setminus L$ can execute all the actions E is able to do, provided that they do not belong to $L \cup \bar{L}$ ($\bar{L}$ refer to the output); The operation $E_1 \underbrace{\parallel}_{A} E_2$

represents the synchronized parallel composition of E_1 and E_2 upon the events from set A .

2.2 Bisimulation-Based Non-Deducibility on Composition Model

A system is considered to have the Bisimulation-based Non-Deducibility on Composition (BNDC) property, if it can preserve its security after composition [2]. A system, ES is BNDC if for every high-level process Π , a low-level user cannot distinguish ES from $(ES \mid \Pi)$ (ES composed with any other process Π). In other words, a system, ES is BNDC if what a low-level user sees if the system is not modified by composing any high-level process Π with ES . $BNDC(ES) \equiv \forall \pi \in E_H, ES \setminus H \approx_B (ES \mid \Pi) \setminus H$ where $ES \setminus H$ changes all the H events in ES into internal events. A system is BNDC-preserving if the above property holds for all possible behaviors of the system.

3 FREEDM System Architecture and Power Balancing Scheme

FREEDM is envisioned as an architecture for future “Smart Distribution” systems [10] [3]. The FREEDM microgrid is a smart grid with advanced technologies of a Solid State Transformer (SST), Distributed Renewable Energy Resource (DRER), and Distributed Energy Storage Device (DESD) managed

with Distributed Grid Intelligence (DGI) to meet the goals of optimal energy management and reliability enhancement. Photo-Voltaic (PV) arrays and Wind turbines are the elements within DRER while DESD consists of high capacity batteries for efficient energy storage. As shown in Figure 2, every residential node, called the Intelligent Energy Management (IEM) node, consists of an SST that manages DRER, DESD and a LOAD which is the consumption of power at the household. The DGI is a major cyber aspect in the FREEDM system with each IEM node running a portion of DGI as a process or processes. The DGI process coordinate among themselves through message passing. The IEM nodes control power flow to and from a shared electrical bus, under the direction of cooperating *DGI* processes.

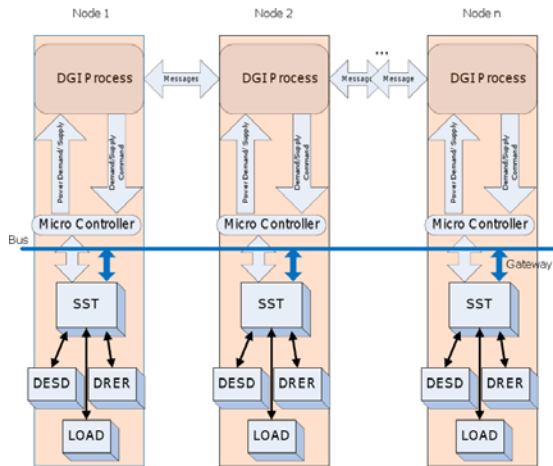


Fig. 2. FREEDM Power Management Architecture

3.1 Distributed Power Balancing Scheme

Distributed load balancing algorithms [11] in computer science are designed to normalize the load of process execution among the peers of a distributed system. Intuitively, the nodes participating in a load balancing algorithm communicate their load changes with each other in an attempt to migrate the process execution task from a node with *Demand* to a node with *Supply*. The result of such a migration is that the nodes normalize their loads, thereby achieving a roughly balanced load computation. Among various algorithms adopted by the DGI is the proposed *Power Balancing* scheme, to efficiently balance power flow through optimal distribution of energy within the system. The implementation of such a Power balancing algorithm in the FREEDM is explained below.

Every IEM computes the SST's actual load on the distribution grid and decides the state of a node as having *Supply* or *Demand* or *Normal* state of load. The algorithm consists of concurrent sub-processes with message passing communication among the IEMs on critical load changes. Each DGI maintains a

Table 1. Load Table maintained at each node

Node	State	Node	State	..	Node	State
1	Supply	1	Supply	..	1	Normal
2	Demand	2	Demand	..	2	Demand
.	.	.	.	..	.	.
.	.	.	.	..	.	.
n	Supply	n	Supply	..	n	Normal
At IEM 1		At IEM 2		..	At IEM n	

(potentially out-of-date) *Load table* as shown in Table 1, to store information it receives about other nodes in the system. Load table updating strategies are adopted to minimize cyber message traffic during frequent load changes. An IEM node, on entering into a *Supply* state, advertises a *Draft Request* message to the nodes in its load table that are in *Demand* state and waits for response. A *Demand* node, on receiving a *Draft Request* message, responds to the sender by sending its demand cost with a special message called *Draft Age*. The *Supply* node, on receiving Draft ages from different *Demand* nodes, will compute a *Draft Standard* which is an optimized selection of the node it is going to supply power to by evaluation of factors like its own predicted need, economics and other optimization metrics. The *Supply* node, on computation of draft standard, sends a unique *Draft Select* message and initiates the power migration by making a set point on the *Gateway power* which is the local SST's individual contribution on to the shared power bus. On receiving the *Draft Select* message from the *Supply* node, the IEM which was in demand receives this power from the shared bus. The migration takes place in unit step size until the time the *Supply* node can supply to the *Demand* node or the *Demand* node meets its sufficient demand, or there is a change of load state in either of the nodes. The algorithm continues till all the nodes are in *Normal* state. A sample DGI trace involving a Drafting node (which can *Supply*) and the source (which is in *Demand*) is shown below:

	DGI_Draft: Request bid from known loaded DGIs
DGI_Source: Respond to bid request if loaded	DGI_Draft: Order the response messages arbitrarily.
	DGI_Draft: Selects power to migrate based on cost
DGI_Source: Responds to select message and commands local SST	DGI_Draft: Sends select message and commands local SST

If a user has access to the state of their DGI, further information can be obtained. Unrestricted information flow can potentially be used against the system for economic gains, under anticipated cap and trade schemes, generators of renewable energy may withhold power to sell at a premium. Such models of information flow are discussed in Section 4.

4 Models of Information Flow

A subnetwork of the FREEDM system with three nodes is depicted in Figure 3. The events in the system are *DRER*, *DESD*, *Load*, *Bus*, *SST* and *Utility* which are the actions associated with DRER, state of DESD, house load, the total power on the shared power bus and strategy of the SST for local management at the node level and utility grid respectively. For notational convenience, the events are distinguished from the actual abbreviations by italicizing them through out the paper. Event classification in to High and Low security levels differ in different scenarios.

Lemma 1. *Power flow in the shared power bus is an invariant function of individual gateway loads of the participating nodes and the draw from or contribution to the utility grid.*

Proof. Assuming the utility grid to be an infinite source and sink of power, the power flow in the shared power bus of local grid can be expressed by the Equation 1.

$$P_{Bus} = \sum_{i=1}^n P_{Gateway} + P_{Utility} \quad (1)$$

where n is the number of nodes and $P_{Utility}$ is the total power draw from or contribution to the utility grid. This is obvious since the flow in the subnetwork is preserved due to Kirchoff's current laws. The net demand or supply on the bus is compensated as a net draw from or contribution to the utility grid, respectively. $\square$

Each node without the DGI process is modeled as in Equation 2. The invariant on the bus shown in Equation 1 can be modeled as in Equation 3. The microgrid consisting of n such nodes can be modeled as in Equation 4.

$$Node_{noDGI} \cong (DRER.\overline{DRER} \sqcap DESD \sqcap Load) \rightarrow X_{SST} \rightarrow (\overline{DESD} \sqcap \overline{Load}).Gateway \rightarrow Node_{noDGI} \quad (2)$$

$$Bus \cong (Gateway_{Node\ 1} | Gateway_{Node\ 2} \dots | Gateway_{Node\ n}) \rightarrow Utility \quad (3)$$

$$E \cong ((Node\ 1_{noDGI} | Node\ 2_{noDGI} | \dots | Node\ n_{noDGI}) \rightarrow Bus).E \quad (4)$$

4.1 External Observer on Physical System

The external observer can know visible information about the DRER like the size of the facility, weather factors impacting the DRER output (represented by *DRER*), but not the output energy generated at any given instance of time ($\overline{DRER}$). As in Figure 3, the external observer could use an inductive pickup to obtain the reading on the shared power bus or even the gateway at each node since the power lines are physically visible and open. The following conclusions can be made on the information flow in the case of such an observer.

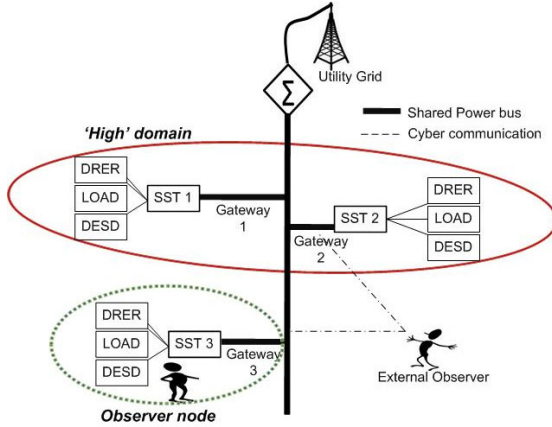


Fig. 3. FREEDM subsystem with no DGI, two nodes and two observers

Lemma 2. *A node without DGI is BNDC-secure with respect to a low-level external observer with limited physical observability.*

Proof. Assuming that the low-level observer can only observe the visible DRER sources, the classification of events at any node as defined in Equation 2 is $Low = \{DRER\}$, $High = \overline{DRER}, DESD, Load, X_{SST}, \overline{DESD}, \overline{Load}, Gateway$. Restricting all the high level events within the node yields, $Node_{noDGI} \setminus H \equiv \{DRER\}$. For any high level process Π , say, $X_{SST}.Gateway$ or $\overline{DRER}.X_{SST}$ the restriction of the composed system, $(Node_{noDGI}|\Pi) \setminus H \equiv \{DRER\}$. Therefore, $E \setminus H \approx_B (E|\Pi) \setminus H$. We can conclude that it cannot distinguish between $Node_{noDGI}$ and $(Node_{noDGI}|\Pi) \forall \Pi \in E$. $\square$

Lemma 3. *A node without DGI is BNDC-secure with respect to a low-level external observer which can read the gateway at the node.*

Theorem 1. *The physical system in FREEDM is BNDC-secure with respect to a low-level external observer as shown in Figure 3.*

Proof. From Lemmas 2 3, it follows that low-level observations on DRER and gateway at individual nodes is BNDC-secure. When composed with the bus as in Equation 4, the system still satisfies the BNDC property. Assuming that the low-level observer can observe the visible DRER sources as well as the *Bus*, the classification of events within the system as defined in Equation 4 is $Low = \{DRER_{i=1}^n, Bus\}$, $High = \{Node_{1noDGI}, Node_{2noDGI} \dots Node_{nnoDGI}, Utility\}$. Restricting all the high level events within the system yields, $E \setminus H \equiv \{DRER_{i=1}^n, Bus\}$. For any high level process Π , say, $X_{SST}^1.Gateway 1 \rightarrow X_{SST}^1.Gateway 2$ the high-level restriction on composed system, $(Node_{noDGI}|\Pi) \setminus H \equiv \{DRER_{i=1}^n, Bus\}$. Due to Lemma 1, observation of *Bus* is always consistent since $\sum_{i=1}^n Gateway + Utility = \sum_{i=1}^n Gateway' + Utility'$. Therefore, $E \setminus H \approx_B (E|\Pi) \setminus H$. $\square$

Given that the observer can observe all the gateway loads, the observer can match every unique *Gateway* event with a corresponding *Bus* event, thereby divulging the confidentiality of the system. In that case, restricting all the high level events within the system yields, $E \setminus H \equiv \{DRER_{i=1}^n, Gateway_{i=1}^n, Bus\}$. For any $\Pi \equiv X_{SST}^1.Utility \rightarrow X_{SST}^2.Utility, (Node_{noDGI}|\Pi) \setminus H \equiv DRER_i, Gateway'_i, Bus'\}$ where Bus' is inconsistent with the event, Bus . In that case, the system is not BNDC-secure.

4.2 Internal Observer on the Physical System

If the nodes are not involved in the DGI power balancing process, the low-level internal observer as shown in Figure 3, who is a part of the physical grid can observe a change on the shared power bus, whenever a *Supply* node renders its excess generation to the utility grid or a *Demand* node absorbs power from the utility grid. However, the observer cannot exactly tell who performed the change (as in the case of Fred and Barney in Example 1). Therefore, we can say that the system without the DGI process is non-deducible secure. This leads to the following Lemma 4.

Lemma 4. *The system without the DGI process is non-deducible secure.*

Theorem 2. *The physical system in FREEDM is BNDC-secure with respect to a low-level internal observer as shown in Figure 3.*

Proof. Assuming that the low-level internal observer, IO can observe the visible DRER sources as well as the *Bus*, the classification of events within the system as defined in Equation 4 is $Low = \{DRER_{i=1}^n, Node\ IO_{noDGI}, Bus, Utility\}$, $High = \{Node\ 1_{noDGI}, Node\ 2_{noDGI} \dots Node\ n_{noDGI}\}$. Restricting all the high level events within the system yields, $E \setminus H \equiv \{DRER_{i=1}^n, Node\ IO_{noDGI}, Bus, Utility\}$. For any high level process Π , say, $X_{SST}^i.Gateway\ i \rightarrow X_{SST}^j.Gateway\ j$ where $i, j \neq IO$ the high-level restriction on the composed system, $(Node_{noDGI}|\Pi) \setminus H \equiv \{DRER_{i=1}^n, Node\ IO_{noDGI}, Bus\}$. As with the case with external observer in Theorem 1, following the Lemma 1, observation of *Bus* is always consistent since $\sum_{i=1}^n Gateway + Utility = \sum_{i=1}^n Gateway' + Utility'$. Therefore, $E \setminus H \approx_B (E|\Pi) \setminus H$. $\square$

4.3 Internal Observer without DGI, on the Physical System Composed with DGI

The system composed with power balancing process preserves non-deducibility. Intuitively, this is possible due to the invariance of physical flow as in Equation 1. The nodes participating in power management process make their changes in such a way that the net power flow at the bus remains constant. We have proved this case in our previous work [12] using a gas pipeline system as test case. With Π being the DGI process, LB as defined in Equation 5, each Node can now be defined as in Equation 6.

$$LB \cong (SendDraftRequest \sqcap ReceiveDraftRequest) \rightarrow (ReceiveCost \sqcap SendCost) \rightarrow (ComputeDraftStandard.DraftSelect \sqcap AcceptDraft) \rightarrow (\text{Supplier} \underbrace{\parallel}_{Migrate} \text{Demander}) \quad (5)$$

$$IEM \cong (DRER.\overline{DRER} \sqcap DESD \sqcap Load) \rightarrow X_{SST} \rightarrow LB \rightarrow (\overline{DESD} \sqcap \overline{Load}).Gateway \rightarrow IEM \quad (6)$$

$$E|II = [IEM\ 1 | IEM\ 2 | \dots]_n \rightarrow Bus \quad (7)$$

The system composed with the DGI process, $E|II$ can be defined as in Equation 7. Assuming that the low-level internal observer, IO can observe the visible DRER sources, the classification of events within the system as defined in Equation 7 is $Low = \{DRER_{i=1}^n, IEM\ IO, Bus, Utility_{IO}\}$, $High = \{IEM\ 1, IEM\ 2 \dots IEM\ n\}$.

Theorem 3. *The system composed with the DGI process, as modeled in Equation 7 satisfies BNDC property with respect to an internal observer without DGI.*

Proof. An internal observer without DGI cannot see the high-level message exchanges associated with the DGI process. Given this, it is unaware of any power migration due to the power balancing algorithm. The high-level restriction of on the system is $E \setminus H = \{DRER_{i=1}^n\} \rightarrow Bus$ and the high-level restriction on the system composed with the DGI, $(E|II) \setminus H = \{DRER_{i=1}^n\} \rightarrow Bus'$. However, Bus is consistent with Bus' due to the invariant as defined in Equation 1. The total power on the bus connecting the three nodes as shown in Figure 4 to the physical grid is given by $P_{Bus} = P_{Gateway1} + P_{Gateway2} + P_{Gateway3}$. As a result of load balancing, if the migrated power from Node 1 to Node 2 is ζ KW, then

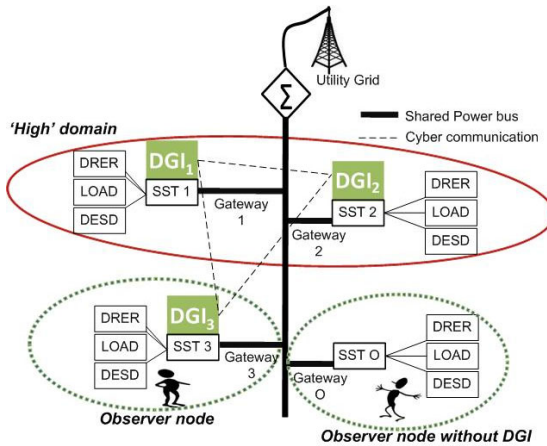


Fig. 4. A FREEDM subsystem with DGI, two nodes and two observers

$P'_{Bus} = (P_{Gateway1} - \zeta) + (P_{Gateway2} + \zeta) + P_{Gateway3}$. That is, $P_{Bus} = P'_{Bus}$. Also, this event Bus' could also be due to any process, $(X_{SST}^i.Gateway.Utility_i) \rightarrow (X_{SST}^j.Utility_j)$ where $i, j \neq IO$. Therefore $E \setminus H \approx_B (E \setminus \Pi) \setminus H$, making the system BNDC-secure. $\square$

5 Internal Observer with DGI, on the System Composed with DGI

For an internal observer with DGI as shown in Figure 4, if Node 1 is in Supply state, it could be either supplying to Node 2 or selling power to the utility grid. On the other hand, if Node 2 is in Demand state, it is either receiving power from Node 1 or receiving from utility grid. Such an observer can infer about the global state of the system by the analysis of load table traces that are updated within its DGI process. A load table trace at every node as shown in Table 1, can be represented in the trace model as a sequence of time varying tuples containing the state information. For example, $\mathfrak{S}_{\Delta t} = \{ (\text{State}(\text{Node } 1) \text{ at time } t1, \dots, \text{State}(\text{Node } n) \text{ at time } t1), (\text{State}(\text{Node } 1) \text{ at time } t2, \dots, \text{State}(\text{Node } n) \text{ at time } t2), \dots \}$. The observer's view of the system changes depending on the current state of the node, leading to different cases of information flow as below.

5.1 Observer in Demand State

From its Load table trace, observer can see the nodes that are in *Demand* state and *Supply* state. The quantity of information that is observable is more in this case, since it receives Draft requests from all the nodes that are in *Supply* state. The observer in *Demand* state responds to the draft requests by sending its demand cost (Draft age). If it receives a *Refusal*, it could be because the *Supply* node it responded to has an inadequate matching cost to satisfy its requirement or the *Supply* node has selected to draft with another *Demand* node which has a higher demand cost. In the case with only three IEMs, this doubt can be resolved as follows: If there is no other *Demand* node that the observer can see, then the *Supply* node does not have enough power to match its requirement. In this case, it can advertise a lesser cost till the time it succeeds. However, at the time it succeeds, it now has an estimate of the excess power the *Supply* node has, with which it can infer its *Load*. Formally, this information flow can be represented as below:

Theorem 4. *The DGI power balancing process is not BNDC-secure with respect to an internal observer in Demand state.*

Proof. Let Π be a power balancing process between IEM 1 and IEM 2 as shown in Equation 5. From its load table trace $\mathfrak{S}_t = \{(\text{Supply}, \text{Demand})\}$, IEM 3 initiates the high-level power balancing process Π' with IEM1. It advertises a cost, $\hat{Cost}_3$ and experiences a refusal, $\mathcal{R}$.

$$IEM\ 1|IEM\ 2 \cong ([\mathfrak{S}_t \rightarrow \Pi \rightarrow \hat{Cost}_2]_{IEM1} \underbrace{\parallel}_{Migrate} [\mathfrak{S}_t \rightarrow \Pi \rightarrow \hat{Cost}_2]_{IEM2}) \quad (8)$$

$$\begin{aligned} (IEM\ 1|IEM\ 2|\Pi') &\cong ([\mathfrak{S}_t \rightarrow \Pi \rightarrow \hat{Cost}_2]_{IEM1} \underbrace{\parallel}_{Migrate} [\mathfrak{S}_t \rightarrow \Pi \rightarrow \hat{Cost}_2]_{IEM2}) \sqcap ([\mathfrak{S}_t \rightarrow \Pi' \rightarrow \hat{Cost}_3]_{IEM1} \underbrace{\parallel}_{Migrate} [\mathfrak{S}_t \rightarrow \Pi' \rightarrow \hat{Cost}_3]_{IEM3}) \\ (IEM\ 1|IEM\ 2) \backslash H &\cong \mathcal{R} \\ (IEM\ 1|IEM\ 2|\Pi') \backslash H &\cong \mathfrak{S}_t \rightarrow \Pi' \rightarrow \hat{Cost}_3 \rightarrow \mathcal{R} \end{aligned}$$

From the above set of equations, obviously $(IEM\ 1|\Pi) \backslash H \not\approx_B (IEM\ 1|\Pi|\Pi') \backslash H$. The proof can easily be extended to n IEMs in the system. Hence the system is not BNDC-secure with respect to an internal observer in *Demand* state. $\square$

Alternatively, the observer, on experiencing a *Refusal* of its Draft age, can bid a higher cost until the time it receives a *Draft Select*, meaning that it is selected by the *Supply* node to Draft. In this case, cost of the other *Demand* node is divulged, along with interference of high level activity between the *Demand* node and the *Supply* node.

5.2 Observer in *Supply* State

The observer in *Supply* state can have information on the nodes that are in *Demand* state with certainty. It initiates the Draft request to obtain the *Draft ages* from the *Demand* nodes which include their respective demands. It is possible that the *Demand* node experiences a refusal, $\mathcal{R}$ since the observer is not actually ready for migration and the observer can continue this process by which results in the *Demand* node not satisfying its request from any other IEMs in supply state. However, this case can be handled by not accepting any Draft requests from the presumably *Supply* node after a certain number of Refusals. Along with the low level physical observation and these demands advertised by the *Demand* nodes, the observer can infer critical information about DESD, Loads and strategy of SST at the *Demand* node.

Theorem 5. *The DGI process is not BNDC-secure with respect to an internal observer with DGI in Supply state.*

The case for the observer in the *Normal* state is omitted due to space constraints.

6 Results

Figure 5 shows the result of power migration from the DRER among three IEM nodes. These results were obtained by integrating the DGI process with a

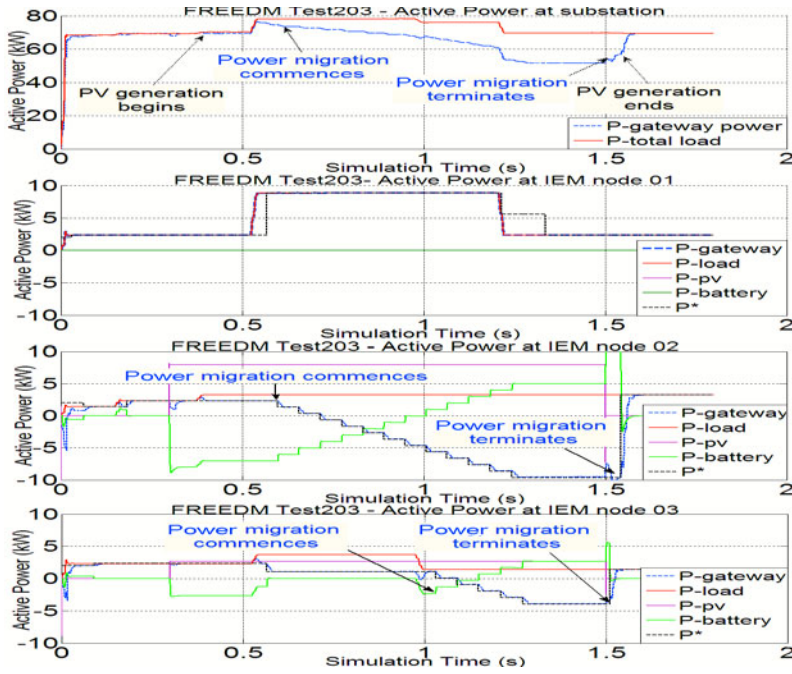


Fig. 5. Power Balancing scheme on 3 IEM nodes

Simulink[®] model of the FREEDM system with three IEM nodes [13]. It can be observed in Figure 5 that IEM node 02 begins migrating power to IEM node 01 which was in a *Demand*. Later at $t = 0.9761$ s, IEM node 03 also reaches to a *Supply* state and then both IEM nodes 02 and 03 migrate power to IEM node 01. An Internal Observer at IEM03, according to Theorem 5, when composed with the power management algorithm, by equation 5, can deduce that IEM node 01 is in *Demand* state. Similarly, IEM 01, by Theorem 4, can deduce that IEM 02, and later IEM 03 are in a supply state.

The information flow models discussed in Section 4 reveal critical ways in which information can be divulged in the context of FREEDM. The goal of such an analysis is to formally prove how physical observability and the inherent nature of the cooperating DGI processes lead to partial deducibility of information. These models present real-time scenarios in which the participants of the system can breach confidentiality with such an information leakage and sometimes, co-operate among themselves to cause unfair energy distribution and non-economic management.

7 Conclusions and Future Work

Information flow analysis, with its origins in cyber systems, has been extended to a Cyber-Physical Smart Grid, quantifying potential confidentiality violations.

The FREEDM system design includes the notion of smart loads wherein, the house load can be managed by the DGI process to optimally schedule the component loads. This only adds to the complexity and potential information flow leakage. The next step is to formally show these information flow properties using model checking of the SPA. A future work is to extend the current information flow analysis by considering models in which the observer can perform physical attacks on the infrastructure and in cases where it can manipulate other DGIs by being an internal part of the system, etc. The main contribution of this work is not limited to the analysis of FREEDM, but shows how to apply formal information flow theory to a Cyber-Physical System

References

1. McLean, J.: Security models and information flow. In: *Procs. of the 1990. IEEE Computer Society Press, Los Alamitos* (1990)
2. Focardi, R., Gorrieri, R.: A classification of security properties for process algebras. *Computer Security* 3(1), 5–33 (1994/1995)
3. Smart Distribution System Design: Automatic Reconfiguration for Improved Reliability. In: *IEEE General Meeting, Minneapolis, MN* (2010)
4. Phillips, L.: Hamilton Link, R.S., Welland, L.: Agent-based control of distributed infrastructure resources. Technical report, Sandia National Laboratories, Albuquerque, New Mexico (2006), SAND2005-7937
5. IEEE: IEEE P1547.4 Draft Guide for Design, Operation, and Integration of Distributed Resource Island Systems with Electric Power Systems
6. Lehtonen, M., et al.: Distribution energy management in the environment of deregulated electricity market. In: *Proc. Energy Management and Power Delivery*, vol. 2, pp. 516–521 (1995)
7. Nikkhajoei, H., Lasseter, R.H.: Microgrid protection. In: *Proc. IEEE Power Engineering Society General Meeting*, pp. 1–6 (2007)
8. McDaniel, P., McLaughlin, S.: Security and privacy challenges in the smart grid. *IEEE Security Privacy* 7(3), 75–77 (2009)
9. Chronicles, M.R.: What will talking power meters say about you (2009), <http://redtape.msnbc.com/2009/10/would-you-sign-up-for-a-discount-with-your-power-company-in-exchange-for-surrendering-control-of-your-thermostat-what-if-it.html> (accessed March 19, 2010)
10. Huang, A.: Renewable energy system research and education at the NSF FREEDM systems center. In: *Power & Energy Society General Meeting, PES 2009*, pp. 1–6. IEEE, Los Alamitos (2009)
11. Ni, L.M., Xu, C.W., Gendreau, T.B.: A distributed drafting algorithm for load balancing. *IEEE Trans. Softw. Eng.* 11(10), 1153–1161 (1985)
12. Akella, R., McMillin, B.M.: Model-Checking BNDC Properties in Cyber-Physical Systems. In: *Annual International Computer Software and Applications Conference*, vol. 1, pp. 660–663 (2009)
13. Akella, R., Meng, F., Ditch, D., McMillin, B., Crow, M.: Distributed Power Balancing for FREEDM system. Technical report, Missouri University of Science and Technology, Rolla, MO, USA (2010), <http://filpower.mst.edu/documents/Akella-FreedmAC10.pdf>

Integrated Cyber-Physical Fault Injection for Reliability Analysis of the Smart Grid

Ayman Faza, Sahra Sedigh, and Bruce McMillin

Missouri University of Science and Technology

Rolla, MO, USA

{azfdmb, sedighs, ff}@mst.edu

Abstract. The term “Smart Grid” broadly describes emerging power systems whose physical operation is managed by significant intelligence. The cyber infrastructure providing this intelligence is composed of power electronics devices that regulate the flow of power in the physical portion of the grid. Distributed software is used to determine the appropriate settings for these devices. Failures in the operation of the Smart Grid can occur due to malfunctions in physical or cyber (hardware or software) components.

This paper describes the use of fault injection in identifying failure scenarios for the Smart Grid. Software faults are injected to represent failures in the cyber infrastructure. Physical failures are concurrently represented, creating integrated cyber-physical failure scenarios that differentiate this work from related studies. The effect of these failure scenarios is studied in two cases: with and without fault detection in the distributed software. The paper concludes by utilizing the information gained to refine and improve the accuracy of the quantitative reliability model presented in our earlier work.

Keywords: Smart Grid, Fault Injection, Cyber-Physical System, Reliability.

1 Introduction

The high complexity of the electric power grid, exacerbated by increased stress on its operation, have motivated the use of cyber infrastructure to fortify the operation of the grid. The intelligence provided by this cyber infrastructure led to the concept of the Smart Grid [18]. Different definitions proposed for the Smart Grid concur that it improves the dependability of its predecessors by using intelligent power electronics devices that communicate with each other to prevent line overloads and cascading failures that can lead to blackouts. The addition of this computing and communication capability creates a cyber-physical system that incorporates both conventional components of the power grid (physical infrastructure), and the computing and communication elements (cyber infrastructure) used for monitoring and control.

Our research considers a Smart Grid where Flexible AC Transmission Systems (FACTS) devices are used to prevent cascading failures by controlling power

flow. These devices are deployed on a number of critical transmission lines in the system, and communicate to collectively determine flow values that would prevent overloads from occurring in the physical system, and hence prevent the system from failing, even in the presence of transmission line outages [4], [10]. The settings for the FACTS devices are determined using the Maximum Flow (MaxFlow) algorithm [2], which computes (cyber) the maximum amount of flow that can be carried by each transmission line without violating its capacity constraint (physical).

Incorrect operation of the MaxFlow algorithm can lead to incorrect settings on the FACTS devices, which may or may not lead to errors in the operation of the grid. In this paper, we use fault injection to analyze the effect of errors in the operation of the MaxFlow algorithm. For our analysis, we use an instance of the IEEE118 bus system, shown in Figure 1 as our case study. In this system, FACTS devices F_1 through F_7 collectively execute the MaxFlow algorithm. A summary of the potential cascade-initiating transmission lines and the placement of the FACTS devices is shown in Table 1.

We use simulation to uncover cases where erroneous operation of the FACTS devices can lead to a failure in the operation of the physical portion of the grid. The overarching objective of our work is to develop a quantitative reliability model for the Smart Grid as a cyber-physical system, based on understanding the semantics of the operation of the Smart Grid and the interaction among its components. This model, and the research leading to its development, has been presented in our previous publications [5,6,7]. Each additional failure

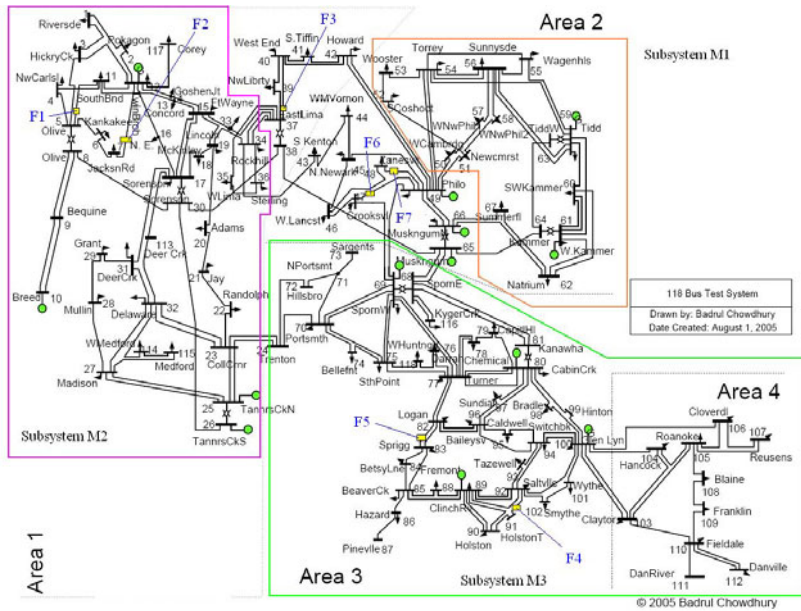


Fig. 1. The IEEE118 bus system, with FACTS devices deployed. Reprinted from [7].

Table 1. Locations of FACTS devices required for mitigation of failures. Reprinted from [7].

Cascading Failure	Initiating Line	1 st Device/Line	2 nd Device/Line
1	(4-5)	F1/(5-11)	F2/(7-12)
2	(37-39)	F3/(37-40)	
3	(89-92)	F4/(91-92)	F5/(82-83)
4	(47-69)	F6/(47-49)	F7/(48-49)

scenario identified for the cyber-physical system as a whole refines our model and increases its accuracy. The work presented in this paper aims to discover failure scenarios that would be missed by independent analysis of the cyber and physical infrastructures, respectively. The focus on integrated cyber-physical fault injection as a means to identify failure scenarios differentiates this paper from our previous work.

The remainder of this paper is organized as follows. Section 2 presents a summary of related literature. Section 3 describes the software faults that were injected in the cyber infrastructure, and Section 4 describes the effect of these faults on the physical operation of the grid. Results and analysis are presented in Sections 5 and 6. Section 7 concludes the paper.

2 Literature Review

The Smart Grid was first mentioned in The Energy Independence and Security Act of 2007 [18], where it was established that the electricity transmission and distribution system should be modernized to maintain a reliable and secure electricity infrastructure that can meet future growth in demand. Since then, several studies have been published [13,19] that represent efforts in improving the operation of the power grid according to the requirements set by the act or to discuss specific concerns or activities such as security, reliability [12], or smart metering [14].

Our vision is broader and considers a longer-term vision of a Smart Grid transmission system, with emphasis on reliability of such a system. The presence of intelligent equipment in the grid should theoretically help improve the overall system reliability, but it may also cause problems in an otherwise functioning purely physical network. In this paper, we assess the ability of the intelligent devices (FACTS devices in this case) to improve on the reliability of the grid.

Another category of related work pertains to critical infrastructure, the modern version of which is cyber-physical. Protection of the power grid using intelligent equipment has been discussed in [3,11]. Also relevant are studies such as [9], which models interdependencies among infrastructures that interact with each other. Modeling and simulation of critical infrastructure is an area of active research, especially in Europe. CRUTIAL [15] and DIESIS [16] are especially noteworthy projects in this area, as is ARTEMIS, where the focus lies on interoperability and interdependencies [17].

Our work, while related to the aforementioned studies, is significantly different, as we develop a quantitative model that captures the effect of cyber and physical failures on the operation of the power infrastructure. The ultimate objective of our work is to identify cases where supplementing the physical infrastructure with cyber computing and communication will be most effective in fortifying the system. The remainder of this paper articulates our approach to system characterization with fault injection, and describes how the failure scenarios identified as a result are used to refine and improve the quantitative reliability model described in our previous work [5,6,7].

3 Failures in the Cyber Infrastructure

As described in Section 1, our work considers a Smart Grid where power distribution is streamlined and fortified by using FACTS devices that control the flow of power on certain critical transmission lines. The settings for each FACTS device, i.e., the amount of flow on the corresponding line, are determined by the MaxFlow algorithm [2], which uses information about the system topology and line capacities to determine the optimal flow for each line in the grid [8].

Figure 2 presents an example of what could happen as the result of an error in the software used to implement the MaxFlow algorithm. The resulting software fault could lead to incorrect operation of the MaxFlow algorithm, where the flow in one of the lines could be erroneously increased by 10%. This incorrect increase in the flow of one line can lead to changes in the flow of many other lines, and will eventually cause the MaxFlow algorithm to produce incorrect settings for the FACTS devices. As an example, a FACTS device could erroneously set the flow on a certain transmission line to 80% of the rated value (the typical flow on the line, under normal operating conditions). When such an error occurs,

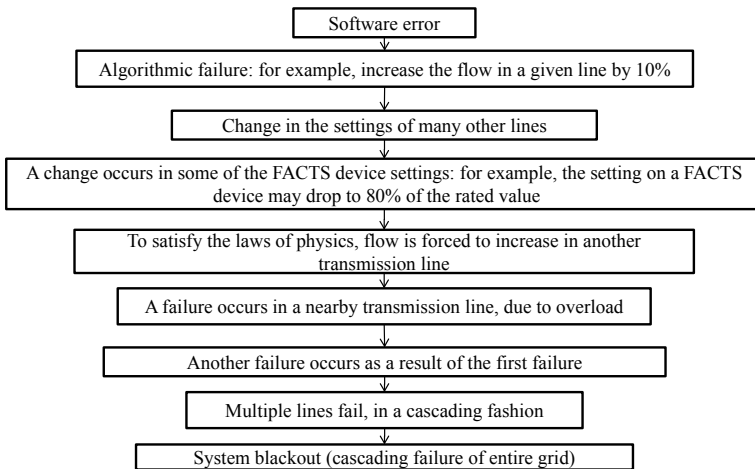


Fig. 2. Software errors lead to cascading failures

the flow in other transmission lines is forced to increase to satisfy the laws of physics. As a result, overload can occur in a nearby transmission line, causing a line outage that in turn leads to a number of additional overloads, eventually causing a system-level failure.

3.1 Software Faults Injected in the MaxFlow Algorithm

It is clear that faults in the operation of the MaxFlow algorithm can lead to erroneous settings on the transmission lines. We describe a number of such faults below, and using fault injection, investigate their effect on the operation of the algorithm. These errors are not comprehensive; their selection is due to the fact that they are among the most typical results of algorithm corruption, and can result from cyber attacks in the form of program modification [1]. Our main goal in this analysis is to identify patterns in how software faults can lead to cyber-physical system failures, and to achieve a better understanding of failure propagation from the cyber infrastructure to the physical infrastructure.

1. **All-Excess Fault.** During the operation of the MaxFlow algorithm, each vertex in the graph that represents the power grid can hold a certain amount of excess flow. The All-Excess software fault decreases the excess value for each vertex by one unit. This will cause a number of incorrect results in the MaxFlow output, and may lead to erroneous FACTS device settings.
2. **Excess-Excess Fault.** The Excess-Excess fault increases the excess value of a given vertex by one unit. Since there are 118 vertices in the system (corresponding to the 118 buses of the physical power network), the fault is injected 118 times, each time at a different vertex.
3. **One-Time-Adjust Fault.** In this fault, the amount of flow in all edges is increased by 10 units. This fault is applied to all the edges in the network.
4. **Adjust-Amount Fault.** This fault adjusts the flow in a given edge by increasing the flow to 110% of the original value.

3.2 Physical Limitations on the Operation of the FACTS Device

The settings determined by MaxFlow for the FACTS devices may change due to the occurrence of any of the software faults mentioned above. The FACTS devices, however, have a certain rating based on their power electronics component ratings, and can only operate between 80% and 120% of their rated value. This imposes a limitation on how severely the fault can affect the operation of the FACTS device. If the erroneous output of the MaxFlow algorithm suggests that the setting on the FACTS device needs to be at a value lower than 80% of the rated value, the FACTS device will simply set itself to exactly 80% of the rated value, due to this physical limitation. Similarly, a FACTS device cannot be set to a value greater than 120% of its rating, and if the MaxFlow setting happens to be higher, the FACTS device will simply limit it to 120% of the rated value.

Furthermore, a FACTS device can be programmed to not allow the flow on a transmission line to go beyond the capacity of the line. If this precautionary measure is not carried out, a FACTS device can allow the flow to become as

high as 120% of the rated value of the device, which may be higher than the capacity of the transmission line on which the device is deployed. The result can be overload and subsequent outage of the transmission line.

4 Effects of Cyber Failures on Physical Operation of the Smart Grid

The faults described in the previous section can lead to incorrect operation of a FACTS device. In this section, we describe our fault injection experiments and analyze the effects of the faults injected on the behavior of the FACTS devices and on the operation of the Smart Grid as a whole.

4.1 Fault Injection Experiments

Cyber Failures - No Prior Physical Line Outage. The initial fault injection experiments were performed on a fully-functional physical system; i.e., one with no transmission line outages. The purpose of the experiments was to determine whether any of the cyber faults injected can cause incorrect settings to be determined for the FACTS devices.

Simulation results showed that many of the faults described in Section 3.1 can lead to incorrect MaxFlow settings. On the other hand, correct settings can still be determined by the algorithm, despite the presence of a number of these faults. Table 2 presents a summary of the faults that can lead to a MaxFlow settings of less than 80% or higher than 120% of the rated value, respectively. While all of the values determined were either less than 80% of the rated value or higher than 120%, constraints on the operation of the FACTS device will only allow the setting to go as low as 80% or as high as 120% of the rated value. A number of cases resulted in values within the range of 80%-120%, but are omitted from the tables for brevity.

Table 2. Cyber failure: software fault injection, no prior physical line outage

No physical line outages			
Fault type: Excess-Excess			
Parameter(s)	FACTS/Transmission line	% of rated value	Failure mode
11	F1/5-11	72.3%	Limit to 80%
1-23,25-34	F3/37-40	0%	Limit to 80%
Fault type: Adjust-Amount			
Parameter(s)	FACTS/Transmission line	% of rated value	Failure mode
8	F7/48-49	62.8%	Limit to 80%
46	F6/47-49	18.3%	Limit to 80%
49	F2/7-12	218%	Limit to 120%
49	F3/37-40	158%	Limit to 120%

Simultaneous Cyber and Physical Failures. To further analyze the system, we performed software fault injection on a system with a pre-existing line outage - a physical failure. We chose three distinct outages, corresponding to lines 4-5, 37-40, and 89-92, respectively. These lines were chosen because in the absence of FACTS devices (a purely physical grid), outage of each of them can cause a cascading failure in the grid (see Table 1 in Section 1). The deployment of FACTS devices and resulting cyber control of the grid prevents these cascading failures.

For brevity, we show only the results for software fault injection on a grid with a prior outage of line 4-5. Table 3 summarizes the results. The other two cases (outage of lines 37-40 and 89-92, respectively) provided similar results.

Table 3. Cyber-physical failure: software fault injection, prior outage of line 4-5

Outage: Line 4-5			
Fault type: Excess-Excess			
Parameter(s)	FACTS/Transmission line	% of rated value	Failure mode
1-84,86-118	F3/37-40	0%	Limit to 80%
1-118	F1/5-11	67.9%	Limit to 80%
Fault type: Adjust-Amount			
Parameter(s)	FACTS/Transmission line	% of rated value	Failure mode
1-84,86-118	F3/37-40	0%	Limit to 80%
1-52, 54-112, 114-118	F6/47-49	176%	Limit to 120%
Fault Type: One-Time-Adjust			
Parameter(s)	FACTS/Transmission line	% of rated value	Failure mode
N/A	F7/48-49	56.7%	Limit to 80%

4.2 FACTS Device Failures Resulting from Software Fault Injection

Software fault injection on the simulated Smart Grid led to three cases: a MaxFlow setting that is 80% of the rated value of the FACTS device, a MaxFlow setting that is 120% of the rated value, and a MaxFlow setting that is somewhere in between these two ranges. The consequences of each of these erroneous settings depend on whether the injected software fault is detected by the algorithm. Two cases are described below, fault detection is enabled for the first and disabled for the second.

Fault Detection Enabled. If we assume that the presence of a software fault can be detected (using executable assertions on the algorithm’s correctness [1]), but cannot be corrected, one of the following options can be used as a protective action.

1. *Bypass the FACTS device.* The protective measure taken in this case is to disconnect the FACTS device from the power grid, returning it to the purely physical mode. The advantage of this measure is that it prevents a software-induced error from affecting an otherwise functioning system. The disadvantage is that if a line outage occurs, it might lead to a cascading failure, as cyber control is effectively disabled.

2. *Limit to line capacity.* If a FACTS device cannot determine the actual setting from the MaxFlow algorithm, it can still prevent the transmission line on which it is deployed from overload and subsequent outage. A cascading failure can still occur as a result of outage of a neighboring line, but simulation shows that this protective action prevents cascading failure in some cases.
3. *Use the most recent setting.* FACTS devices can be programmed to revert back to the most recent correct setting if the device is unable to determine the correct setting as the result of a software fault. This is a good option if the system was otherwise functioning properly. The occurrence of a line outage may render this protective action ineffective.

Fault Detection Disabled. Fault detection may not be feasible for all implementations of the MaxFlow algorithm. The three cases below describe the possible scenarios resulting from undetected software faults.

1. *Set flow to 80% of rated value.* If the MaxFlow setting is below 80% of the rated value, the FACTS device will be set to 80%. This will not cause outage of the transmission line on which the FACTS device is deployed, but it may cause changes to occur in the flow values of the remaining lines in the system, which could lead to failures elsewhere in the system.
2. *Set flow to 120% of rated value.* If the MaxFlow setting is above 120% of the rated value, the FACTS device will be set to 120%. If this value is below the line capacity, it will not cause a failure, but it may force the flow values in other transmission lines to change in a fashion that could cause the system to fail.
3. *Set flow to the erroneous value obtained from the MaxFlow algorithm.* If the erroneous setting determined by MaxFlow is within 80%-120% of the rated value, it will be used by the FACTS device to set the flow on the corresponding transmission line. Again, depending on the overall system topology and status, this may or may not lead to a failure in the system.

Using power system load flow simulations, we tested all of the aforementioned scenarios, and identified the cases that lead to failures at the system level. The results are presented in the following section.

5 Results and Analysis

Table 4 summarizes the results obtained from simulating the effects of failures when fault detection is not possible. Each row indicates the system status for a particular pre-existing line outage. Each column corresponds to one failure scenario resulting from software fault injection. An entry labeled as “SAFE” denotes that no cascading failure has occurred in the system. An entry labeled “FAILED” denotes that a cascading failure has occurred as a result of the line outage and/or software fault.

The results show that in the absence of a prior line outage, in no case were software faults and the resulting erroneous FACTS device settings detrimental to

Table 4. Simulation results, fault detection disabled

Outage	No FACTS	Perfect FACTS	80% of rated value on $F_1/L_{(5-11)}$	80% of rated value on $F_2/L_{(7-12)}$	120% of rated value on $F_1/L_{(5-11)}$	120% of rated value on $F_2/L_{(7-12)}$
None	SAFE	SAFE	SAFE	SAFE	SAFE	SAFE
$L_{(4-5)}$	FAILED	SAFE	FAILED	SAFE	FAILED	FAILED
			$F_5/L_{(82-83)}$	$F_4/L_{(91-92)}$	$F_5/L_{(82-83)}$	$F_4/L_{(91-92)}$
None	SAFE	SAFE	SAFE	SAFE	SAFE	SAFE
$L_{(89-92)}$	FAILED	SAFE	SAFE	SAFE	SAFE	SAFE
			$F_3/L_{(37-40)}$		$F_3/L_{(37-40)}$	
None	SAFE	SAFE	SAFE		SAFE	
$L_{(37-39)}$	FAILED	SAFE	FAILED		SAFE	

Table 5. Additional simulation results, fault detection disabled

Outage	No FACTS	Perfect FACTS	80% of rated value on $F_1/L_{(5-11)}$	80% of rated value on $F_2/L_{(7-12)}$	120% of rated value $F_1/L_{(5-11)}$	120% of rated value $F_2/L_{(7-12)}$
8-30	SAFE	SAFE	FAILED	FAILED	SAFE	FAILED
6-7	SAFE	SAFE	FAILED	FAILED	SAFE	FAILED
1-3	SAFE	SAFE	SAFE	SAFE	SAFE	SAFE
			on $F_5/L_{(82-83)}$	on $F_4/L_{(91-92)}$	$F_5/L_{(82-83)}$	$F_4/L_{(91-92)}$
92-93	SAFE	FAILED	FAILED	FAILED	SAFE	FAILED
92-94	SAFE	FAILED	FAILED	FAILED	SAFE	FAILED

the system operation. In other words, a functioning system remained functional despite the software failure. However, in the presence of a line outage, software failure and the resulting malfunction of a FACTS device can be the last straw, causing failure of a system that is highly-stressed, but had been tolerating the initial line outage. An example of this case, where malfunctioning cyber control causes the failure of an otherwise operational physical system, can be seen in Table 4, where prior outage of line 4-5, combined with erroneous FACTS device setting on F_2 (120% of the rated value), leads to cascading failure.

Tables 5 and 6 provide additional insight into the effects of failure in the cyber infrastructure, by identifying interesting operational scenarios for the grid. As opposed to Table 4, the line outages considered in this case are those that would not cause a cascading failure in a purely physical grid, as shown in the second (“No FACTS”) column in each table. The addition of FACTS devices, however, introduces cases where concurrent line outage and malfunction of a FACTS device, or alarmingly, in some cases even correct operation of a FACTS device, will lead to cascading failure.

The FACTS devices represented in the columns of Tables 5 and 6, F_1/F_2 and F_5/F_4 , were deployed to prevent the outage of lines 4-5 and 89-92, respectively, as described in Section 1, specifically in Table 1. In the simulations summarized in

Table 6. Additional simulation results, fault detection enabled

Outage	Use most recent setting on $F_1/L_{(5-11)}$	Use most recent setting on $F_2/L_{(7-12)}$	Bypass FACTS device $F_1/L_{(5-11)}$	Bypass FACTS device $F_2/L_{(7-12)}$
8-30	<i>FAILED</i>	FAILED	SAFE	FAILED
6-7	SAFE	FAILED	SAFE	SAFE
1-3	SAFE	SAFE	SAFE	SAFE
	on $F_5/L_{(82-83)}$	on $F_4/L_{(91-92)}$	$F_5/L_{(82-83)}$	$F_4/L_{(91-92)}$
92-93	FAILED	FAILED	SAFE	FAILED
92-94	FAILED	FAILED	SAFE	FAILED

Table 4, we investigated the effect of software failures concurrent with the outage of these cascade-triggering lines. In Tables 5 and 6, we investigate scenarios where these lines remain intact, but other lines in their vicinity experience an outage concurrent with the software failure described. Table 5 shows the simulation results with fault detection disabled. Fault detection was enabled for the simulation cases shown in Table 6. The protective action taken is listed next to the fault detected, in the headings of columns 2-4. Two options were considered after fault detection: bypass the FACTS devices, or use the most recent FACTS device setting, both of which were described in Section 4.2.

The simulation results summarized in Tables 5 and 6 show that the deployment of a FACTS device could be detrimental to an otherwise functioning physical system, despite the original intent of their deployment, which is prevention of line outages that lead to cascading failures. As an example, the italicized entry in Table 5 represents a case where the purely physical system was able to withstand the outage of line 8-30, but a malfunctioning FACTS device reduced the fault-tolerance of the system to the point where the same outage causes a cascading failure. This detrimental effect persists even when the software fault leading to malfunction of the FACTS device is detected, and protective action is taken. The italicized entry in Table 6 represents such a case.

An important conclusion of our fault injection experiments is that the net effect of deploying FACTS devices cannot be determined by superficial analysis. Extensive simulation is required to reveal pathological cases that may lead to a negative effect on system reliability. Such extensive simulation can be prohibitively expensive for any non-trivial grid.

6 A Cyber-Physical Reliability Model for the Smart Grid

The ultimate goal of our research is to encompass both cyber and physical aspects of a cyber-physical system in a single reliability model. In our prior work [6,7], we performed full simulation of single-line outages in the IEEE118 bus system, and determined the effect of each contingency on the operation of

the system. In the course of these simulations, the outage of 120 (of 186) lines was found to NOT cause any further outages, while the remaining 66 triggered a cascading failure.

Our subsequent work investigated the effect of deploying FACTS devices. Power system load flow simulation was used to determine cyber-physical “SAFE” states, i.e., states that do not trigger cascading failures. We developed a general model, shown in Equation 1, that determines system-level reliability of the Smart Grid as a function of the reliabilities of the transmission lines (physical) and FACTS devices (cyber),

$$R_{sys} = p_L^{186} + A * p_L^{185} q_L + \sum_{\forall states \in S} p_L^{185} q_L * f(p_{F_1}, p_{F_2}, \dots, p_{F_7}) \quad (1)$$

where:

p_L : is the reliability of a transmission line. For simplicity, all transmission lines are assumed equally reliable.

$q_L = 1 - p_L$: is the unreliability of a transmission line.

A : is the total number of states in which there is one physical line outage, but the system is still functional, regardless of the presence or absence of FACTS devices in the system.

S : is the set of “SAFE” states that are added to the system due to the (correct) preventative action of FACTS devices.

$f(\cdot)$: is a function of the reliabilities of FACTS devices F_1 through F_7 , respectively. This function depends on the particular FACTS device failure mode, and has the form $A_n p_F^n + A_{n-1} p_F^{n-1} + \dots + A_1 p_F$, where p_F is the reliability of a FACTS device, assuming all devices have equal reliabilities.

This model is modified for each FACTS device failure mode, to reflect the behavior of the FACTS device in response to various software faults, and the ability of the system to detect the fault and take protective action. The “SAFE” states to be added to the equation will differ for each case, based on the behavior of the FACTS device.

Figures 3 and 4 were generated based on the model of Equation 1 and compare the system-level reliability of the purely physical grid to that of the Smart Grid (with and without fault detection).

Figure 3 shows the system-level reliability as a function of transmission line reliability, assuming a 95% reliable FACTS device for three of the simulated cases; namely, “erroneously set the flow to 80% of the rated value,” “erroneously set the flow to 120% of the rated value,” “and use the most recent setting.” The figure also compares these three cases with the case where no FACTS devices were deployed, i.e., a purely physical grid.

Figure 4 compares the system-level reliability achieved by fault detection, for two different protective action modes: bypass the FACTS device upon the detection of a cyber fault, or use the most recent FACTS device setting. The purely physical grid is also included for reference.

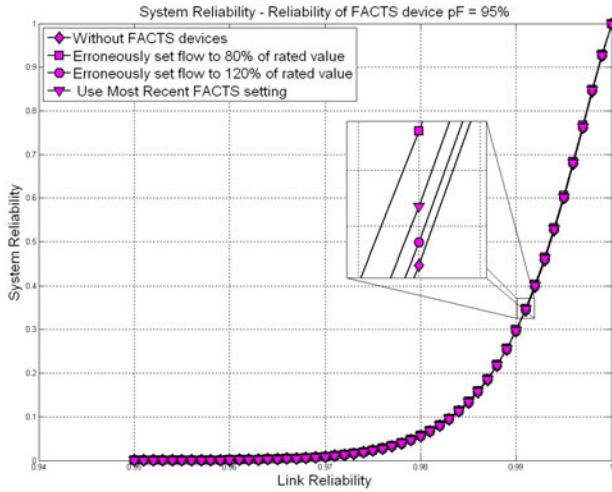


Fig. 3. System-level reliability, with and without fault detection

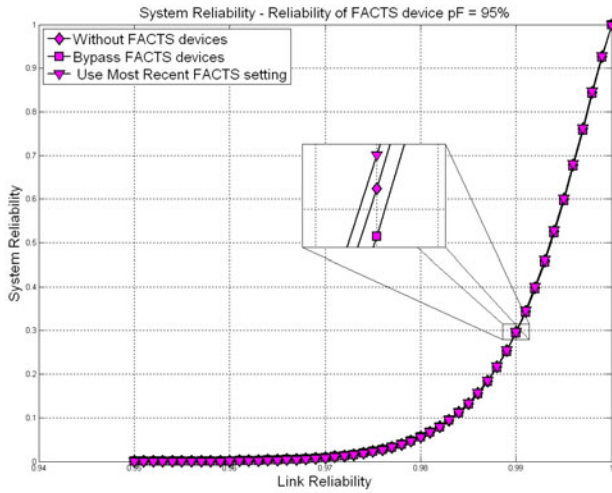


Fig. 4. System-level reliability, with fault detection, in two different protective action modes

7 Conclusion

Fortification of the physical power infrastructure with cyber control is a costly task, undertaken with the intent of making power distribution more reliable. The research presented in this paper identifies cases where failures in the cyber infrastructure compromise this objective. Several different software faults were injected into the Smart Grid, and their results were studied. The reaction of the

FACTS devices was found to vary, based on the type of software fault and the ability of the system to detect the fault. Physical constraints on the operation of FACTS devices limit their settings to between 80% and 120% of the rated value, which in turn limits the potential detrimental effect of failures in cyber control. A number of interesting cases, however, were identified, where a malfunctioning FACTS device caused cascading failure in an otherwise functional physical infrastructure. This discovery reiterates the importance of careful investigation of the effects of cyber control.

In future research, it is our goal to determine the conditions under which FACTS devices improve the overall reliability of the Smart Grid, and to quantify the detrimental effect of failures in the cyber infrastructure. The IEEE118 bus system was used as a case study in the work presented in this paper. Our goal is to generalize the work to similar systems, by studying the effects of cyber failure on operation of a cyber-physical system as a whole.

References

1. Armbruster, A., Gosnell, M., McMillin, B., Crow, M.: Power Transmission Control Using Distributed Max-Flow. In: Proc. of the 29th Annual Int'l Computer Software and Applications Conference (COMPSAC 2005), pp. 256–263 (2005)
2. Armbruster, A., Gosnell, M., McMillin, B., Crow, M.: The Maximum Flow Algorithm Applied to the Placement and Steady State Control of FACTS Devices. In: Proc. of the 2005 North American Power Symposium, pp. 77–83 (2005)
3. Chiaradonna, S., Lollini, P., Giandomenico, F.D.: On a Modelling Framework for the Analysis of Interdependencies in Electric Power Systems. In: Proc. of the 37th Int'l Conf. on Dependable Systems and Networks DSN 2007, pp. 185–195 (2007)
4. Chowdhury, B.H., Baravc, S.: Creating Cascading Failure Scenarios in Interconnected Power Systems. In: IEEE Power Engineering Society General Meeting (June 2006)
5. Faza, A., Sedigh, S., McMillin, B.: Reliability Modeling for the Advanced Electric Power Grid. In: Saglietti, F., Oster, N. (eds.) SAFECOMP 2007. LNCS, vol. 4680, pp. 370–383. Springer, Heidelberg (2007)
6. Faza, A., Sedigh, S., McMillin, B.: The Advanced Electric Power Grid: Complexity Reduction Techniques for Reliability Modeling. In: Harrison, M.D., Sujan, M.-A. (eds.) SAFECOMP 2008. LNCS, vol. 5219, pp. 429–439. Springer, Heidelberg (2008)
7. Faza, A., Sedigh, S., McMillin, B.: Reliability Analysis for the Advanced Electric Power Grid: From Cyber Control and Communication to Physical Manifestations of Failure. In: Buth, B., Rabe, G., Seyfarth, T. (eds.) SAFECOMP 2009. LNCS, vol. 5775, pp. 257–269. Springer, Heidelberg (2009)
8. Kalyani, R., Crow, M., Tauritz, D.: Optimal Placement and Control of Unified Power Flow Control Devices using Evolutionary Computing and Sequential Quadratic Programming. In: Power Systems Conference and Exposition PSCE 2006, pp. 959–964. IEEE PES, Los Alamitos (November 2006)
9. Klein, R., Rome, E., Beyel, C., Linnemann, R., Reinhardt, W., Usov, A.: Information Modelling and Simulation in Large Interdependent Critical Infrastructures in IRRIS. In: Third Int'l Workshop on Critical Information Infrastructure Security, CRITIS, Rome, Italy, pp. 36–47. Springer, Heidelberg (2009)

10. Lininger, A., McMillin, B., Crow, M., Chowdhury, B.: Use of Max-Flow on FACTS Devices. In: North American Power Symposium, pp. 288–294 (2007)
11. Luijff, E., Nieuwenhuijs, A., Klaver, M., Eeten, M., Cruz, E.: Empirical Findings on Critical Infrastructure Dependencies in Europe. In: Third Int'l Workshop on Critical Information Infrastructure Security, CRITIS, Rome, Italy, pp. 302–310. Springer, Heidelberg (2009)
12. McDaniel, P., McLaughlin, S.: Security and Privacy Challenges in the Smart Grid. *IEEE Security and Privacy* 7(3), 75–77 (2009)
13. Olofsson, M.: Power Quality and EMC in Smart Grid. In: Proc. of the 10th International Conference on Electrical Power Quality and Utilization, pp. 1–6 (September 2009)
14. Prassana, G., Lakshmi, A., Sumanth, S., Simha, V., Bapat, J., Koomullil, G.: Data Communication Over the Smart Grid. In: Proc. of the Int'l Symp. on Power Line Communications and its Applications ISPLC 2009, pp. 273–279 (2009)
15. Romani, F., Chiaradonna, S., Giandomenico, F.D., Simoncini, L.: Simulation Models and Implementation of a Simulator for the Performability Analysis of Electric Power Systems Considering Interdependencies. In: Proc. of the 10th IEEE High Assurance Systems Engineering Symp (HASE 2007), Washington, DC, USA, pp. 305–312. IEEE Computer Society, Los Alamitos (2007)
16. Rome, E., Bologna, S., Gelenbe, E., Luijff, E., Masucci, V.: DIESIS - Design of an Interoperable European Federated Simulation Network for Critical Infrastructures. In: Proc. of the 2009 SISO European Simulation Interoperability Workshop (ESIW 2009), pp. 139–146 (2009)
17. The High-Level Group on Embedded Systems: Building Artemis (retrieved June 2004), https://www.artemis-association.org/attachments/647/Building-ARTEMIS-final-report_en.pdf
18. The United States Congress: The Energy Independence and Security Act of 2007 (2007)
19. Wei, X., Yu-hui, Z., Jie-lin, Z.: Energy-efficient Distribution in Smart Grid. In: Proc. of the Int'l Conf. on Sustainable Power Generation and Supply, SUPERGEN 2009, pp. 1–6 (2009)

A Metric for Measuring the Strength of Inter-dependencies

Silvia Ruzzante*, Elisa Castorini, Elena Marchei, and Vincenzo Fioriti

ENEA, Centro Ricerche Casaccia, Via Anguillarese 301, S. Maria di Galeria,
00123 Roma, Italy

Phone number: 0630484367; Fax number: 0630486511

`silvia.ruzzante@enea.it`

Abstract. We propose a metric for the analysis and estimation of the inter dependencies in networks of dynamic systems, formally defining the dependency among nodes and showing that the metric approximates the strength of the dependency. We propose a data driven metric based on known direct functional input/output relations among nodes, derived from the generic constitutive equations of the systems, giving a physical and rigorous meaning to the otherwise elusive word “dependency”. Our metric is also related to the input/output physical quantities, realizing a data driven approach discarding the internal node dynamics. This metric is particularly suited for the analysis of the Critical Infrastructures (CI) where typically a number of input/output measurements are available. It is vital for these CI, represented as technological networks, to characterize and to measure the inter-dependencies among their components in order to avoid destructive phenomena such as cascading failures. The proposed metric is algorithmically simple and can be used as a real-time tool. It was also shown how this approach is suited to the analysis of large technological networks.

Keywords: interdependencies, generalized synchronization, critical infrastructure.

1 Introduction

Today the infrastructure protection and resilience is a major issue in some of the most important research programs in the world. The necessary, preliminary step, is the implementation of a metric for quantifying interdependencies in generic networks of systems. Here we propose a metric for the analysis and estimation of the inter-dependencies in dynamic system networks, defining the strength of the dependence as the norm of the form the generic constitutive equations of the system of systems (SoS) [2, 3, 4, 5, 6, 7, 20, 21, 22]. Central to our problem are the interdependencies among the elements of the networks, present in all the well-known four “dimensions” defined by Rinaldi [8] and recently recognized as the primary cause of the electricity large outages [10]. Describing the interdependencies is a difficult task: not only at

* Corresponding author.

least four “dimensions” (physical, geographic, cyber, logical) are to be taken into consideration [8], but are present feedbacks, feedforwards, non-physical paths, a variety of dynamical modifications, non-linear relations among components, time delays, different temporal scales ranging from milliseconds to years, granularity levels, stochastic events and human factors. Nevertheless, it seems possible to implement a tool to produce a metric. Carreras and Dobson [3] have shown that the Self Organizing Criticality (SOC) is operating in the power grids to let the system grow in a stable manner. Secondly, complex systems to remain stable must arrange a self-adapting mechanism (feedback is one of these) at the cost of minor breakdowns. These examples clearly demonstrate that a SoS is highly structured [1] and even capable of self-adjusting some internal parameters in order to obtain stability or at least to stay at the edge of instability. This critical property, called structural stability, is the basis of any viable mathematical modelling [11, 13]. If these descriptions of the SoS are plausible, we have to admit a causal, input-state-output, non linear, finite dimensional representation to describe the technological interacting networks as a whole (the linear state-space representation could be acceptable if we guarantee in advance stability and convergence around some equilibrium point limiting the amplification of the inputs, but unfortunately this is not generally possible). Anyway, the important state-space properties of reachability and observability are still valid and we identify at a theoretical level, the dependency with the reachability. In the paragraph 3 this notion will be generalized to synchronization. Let are x, y, z, u time dependent vectors, and φ, η vector functions. The Theory of Systems states [11]:

$$x(t) = \varphi(t, t_0, x(t_0), u(\bullet)) \quad (1.1)$$

where $u(\bullet)$ is the input trend. A state $x(t)$ is reachable from the state $x(t_0)$, if $\exists t > t_0$ and $\exists u_{[t_0, t]}$ such that:

$$\varphi(t, t_0, x(t_0), u_{[t_0, t]}) = x(t) \quad (1.2)$$

where φ is the state transition function and $u_{[t_0, t]}$ is the input function interval. The output is:

$$y(t) = \eta(x(t), t, u(t)) \quad (1.3)$$

and with no generality loss, we assume $\eta = I$, where I is the identity operator.

For the observability of a state variable $x(t)$ we must have:

$$\eta(x(t), t, u(t)) \neq \eta(x(t_0), t_0, u(t_0)) \quad (1.4)$$

We also relax (1.2) requiring only that for the state vector x $\varepsilon \ll 1, \varepsilon > 0$:

$$\|x(t) - x(t_0)\| = \varepsilon \quad (1.5)$$

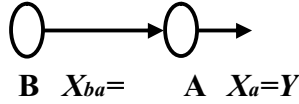


Fig. 1. Output from node B, input to node A, output from node A.

If an input produces a modification in the state of the system we say x depends somehow from the input: of course, input-output relations are often complex, because of the nonlinearity involved and of the closed loops or feedbacks that realize the interdependencies between two systems. It should be observed that here we will avoid to consider directly the almost intractable actual feedbacks present in the real world because the state-of-the-art SCADA (Supervisory Control And Data Acquisition) sensors today are able to produce a high sampling rate data describing the SoIS state variables, therefore enabling an open loop description strategy. In fact, if we are allowed to assume a low sensitivity to initial condition and/or that Δt is small, a local nonlinearity could be neglected. The first condition is met if the maximum Lyapunov exponent is not positive, but to set exactly the sign of this exponent requires to know exactly the equations of the SoIS, so we are compelled to rely mostly on a small Δt (i.e. high sampling rate T_s). Moreover, this way the problem becomes a quasi-stationary problem. It is straightforward to identify a node of the graph with a SoIS (sub) system, and the link with the output of a system (and the input of the other one, Figure 1). State variables x are typical quantities of industrial plants (power, voltage, frequency, pressure), but we could as well add many others non-physical parameters as for example reliability, information, topological values. In order to have a homogeneous set of numbers these values will be normalized.

2 Measuring Interdependence and Synchronization

As stated in [17] finding interdependence is equivalent to find synchronization, thought measuring or detecting the first is much more difficult respect to the second. From this point of view, the most striking form of interdependence is the complete synchronization between two (or more) systems:

$$\lim_{t \rightarrow \infty} (x(t) - z(t)) = 0 \quad (2.1)$$

or, considering phases $\theta(t)$ and $\phi(t)$ instead of amplitudes:

$$\lim_{t \rightarrow \infty} (\theta(t) - \phi(t)) = 0 \quad (2.2)$$

Recently this last case was studied in the distributed energy generation network and it was shown to be a robust synchronization [18]. Of course this form of interdependence is rare. More frequently we have a generalized synchronization, typically for a drive-response couple [16]. If x and z are vectors (Figure 4):

$$\dot{z} = P(z(t)) \quad (2.3)$$

$$\dot{x} = H(x(t), z(t)) \quad (2.4)$$

the generalized synchronization is:

$$x = G(z) \quad (2.5)$$

when G is a smooth function a strong dependence results, otherwise it is weak. Here is the system Z that acts on system X and the dependence is directed toward X : a more complicated situation arises when a third system Q drives both X and Z (Figure 5). Yet generally the big problem is we do not have the exact form of $P(\cdot)$ and $H(\cdot)$ not to mention $G(\cdot)$, but only the noisy time series $x(t)$ and $z(t)$. Moreover, G is often nonlinear, excluding the use of any standard linear correlation technique, so a number of nonlinear measures [17] have been devised in order to assess strength and direction of interdependencies (synchronization). All these measures rely on the reconstructed Takens pseudo space of delayed state vectors [17, 23] obtained from the time-series of the two systems. It is well known [23] these reconstructed pseudo state spaces of X and Z are topologically equivalent to the originals, so if the originals are actually coupled through a functional relation ψ :

$$Z \xrightarrow{\psi} X \quad (2.6)$$

it is conceivable to look for a measure of ψ in the reconstructed state space too.

Let $z_n = (z_n, \dots, z_{n-m+1})$ and $x_n = (x_n, \dots, x_{n-m+1})$ be m -dimensional delay vectors constructed from two simultaneously measured univariate time series ($n = 1, \dots, N$). Let $r_{n,j}$ and $s_{n,j}$, $j = 1, \dots, k$, denote the time indices of the k nearest neighbours of z_n and y_n , respectively. For each z_n , the squared mean Euclidean distance among its k neighbours is defined as

$$R_n^{(k)}(Z) = \frac{1}{k} \sum_{j=1}^k (z_n - z_{r_{n,j}})^2, \quad (2.7)$$

and the X -conditioned squared mean Euclidean distance is defined by replacing the nearest neighbours by the equal time partners of the closest neighbours of x_n ,

$$R_n^{(k)}(Z | X) = \frac{1}{k} \sum_{j=1}^k (z_n - z_{r_{n,j}})^2. \quad (2.8)$$

It turns out that assuming stationary time series, this measure exists and takes values between 0 and 1 i.e. independence and complete dependence (or synchronization) [24]:

$$H(Z | X) = \left(\frac{1}{N} \right) \sum_j^N \ln \left(\frac{R_j(Z)}{R_j^{(k)}(Z | X)} \right) \quad (2.9)$$

That means a relation between the squared mean distance of the nearest neighbours points of every data point in the pseudo state space Z (r index) and the correspondent distance in the pseudo state space X (s index), with N the number of data points in the pseudo state space. Thus, the (2.9) transforms a geometrical relation into a statistical inference. If noise is absent and data are infinitely long this asymmetric measurement is able to detect the “driver” system. Other dependence measures are available, like the standard linear cross correlation or the Kuramoto order parameter [18]; the choice strongly depends on data features.

Drawbacks of this promising method are noise, the length of time series to be recorded, some inherent mathematical difficulties in the reconstructing algorithm [23] and above all its computational complexity $O(N^2)$, where N is the number of data points in the pseudo state space. It is not feasible to execute an exhaustive search to determine the most relevant dependencies if the set of nodes (sub-systems) to be studied is large (each couple of nodes requires two pseudo state spaces). The technological infrastructures endowed with SCADA sensors are able to record many original time series from sensors making more easy reconstructing the state spaces, nevertheless it would be a hard task. In a few words, reducing the problem of interdependencies to the analysis of time series is an important step ahead, yet still too much cumbersome. Therefore, we suggest to use only a few time series samples (without resorting to the pseudo state spaces) to find a viable algorithm, as done in the next paragraphs.

3 Definition of Dependence for CI

Dependence should be primary intended as a causal relation between two entities; here we will focus on nodes of an oriented graph (network). A qualitative good definition is the Rinaldi’s one: “Dependency is a linkage between two infrastructure through which the state of one influences or is correlated to the state of the other” [8]. Deciding the direction of the dependence is usually a hard problem [16, 17] that is out of the scope of this paper because in the technological networks is a good practice [8] to find out only the clearly oriented dependencies. In our framework this is not a limitation. If we can observe input-output data, dependencies are naturally defined [15] and the reachability property assures these data are able to influence nodes. Otherwise it makes no sense assessing a probable hidden dependence that acts on the unreachable/unobservable parts of the system (node). Actually, we should consider the inter-dependency meaning one or more nodes react to inputs by means of a direct feedback but we simplify reducing every inter-dependence to simple dependencies introducing some dummy nodes as in Figure 2 (oriented links indicate inputs from a node and the causal relation acting from a node to the other). A dependence between two nodes (considered as dynamical systems) is a link on the graph and will be characterized by: direction, position, strength, delay, order. The position is related to the geographical setting with respect to a reference point. Delay is the reaction or latency time interval and may vary from milliseconds to hours. Order is the number of links connecting

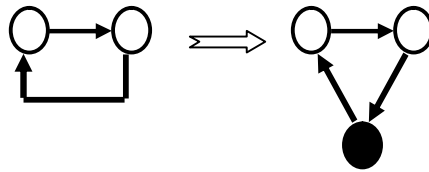


Fig. 2. An inter-dependency splits into two dependencies

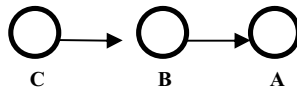


Fig. 3. Node A (target) depends from node C with a Π^2 order dependence

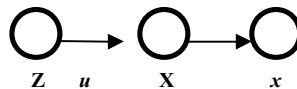


Fig. 4. The output from node Z is the input to node X. Thus Z influences (drives) X.

two nodes (source and target) according to a same direction on the graph (see Figure 3). Therefore before attempting to evaluate the strength of coupling we have to accomplish the identification of nodes, observable input–output quantities, direction and latency delays.

Now consider the situation of Figure 4. System (node) X is influenced by another system Z; $u(t)$ is the output of this driver system Z. The differential system

$$\dot{x} = F(x(t), u(t), t) \quad (3.1)$$

completely describes the dependence. As said before we are not interested in the form of F (which we must generally assume nonlinear), rather we look for some estimation of the input–output coupling strength, approximating (3.1) to an autonomous system. Since the correlations between the pseudo state space of Z and X can be expressed as metric relations [15, 16, 17], it is useful to exploit this circumstance.

The (3.1) can be expressed according to Taylor about an equilibrium point $P_0 = (x_0, u_0)$ up to order first, discarding higher infinitesimal orders:

$$\dot{x} = F(P_0, t_0) + J(x, u)_{|P_0} \begin{vmatrix} x(t) - x(t_0) \\ u(t) - u(t_0) \end{vmatrix} \quad (3.2)$$

with $v = \begin{vmatrix} x(t) - x(t_0) \\ u(t) - u(t_0) \end{vmatrix} \neq 0$,

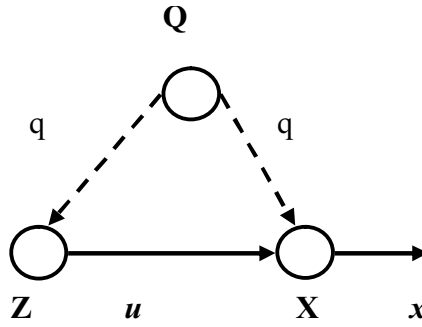


Fig. 5. Q influences (drives) both X and Z. It is not clear if X is driven from Z or Q.

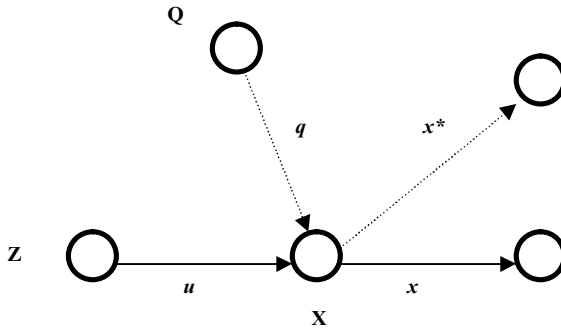


Fig. 6. Multiple links case. Only in the linear case is possible to asses that x is the output in response to the input u without considering q .

where J is the Jacobian and assuming $F(x(t_0), u(t_0), t_0) = 0$, thus we write:

$$\frac{\|\dot{x}\|}{\|v\|} = \|J(x, u) \mid_p\|, \text{ with the } L_2 \text{ norm.} \quad (3.3)$$

We define the (3.3) as the coupling strength of the dependence between node Z and node X. This norm represents the coupling parameter ρ_{xv} and it is the following ratio:

$$\rho_{xv} = \frac{\|\dot{x}\|}{\|v\|}. \quad (3.4)$$

Hence, recalling that vectors x, u are normalized (i.e. variables take between $[0,1]$), when $\rho_{xv} \rightarrow 1$ the dependence is complete.

Note that in general, being the systems nonlinear we cannot assume the superposition principle always valid, i.e. we cannot study an output vector related to an input vector separately, without considering the others inputs at the same time (Figure 6).

Fortunately the industrial control apparatus tends to maintain plants toward a roughly linear behaviour most of the time [8]. Yet this means that any dependence measurement should be considered a rough estimate of reality.

4 The Linear Approach

Sometimes we can suppose that systems are *linear and quasi-static*; in this case the approach is simpler, of course. Again the problem is to find a causal mapping between normalized vectors, in the form of a matrix operator. We deal with a linear transformation A mapping u into x . Let be:

$$A : R^n \rightarrow R^n$$

and $x \in R^n$ such that $x = Au$.

If A is a non singular n rows n columns matrix, using the $L2$ -norm:

$$\|x\| = \|Au\| \leq \|A\| \cdot \|u\|,$$

then

$$\frac{\|x\|}{\|u\|} \leq \|A\| \quad \text{with} \quad u \neq 0.$$

As before, this last relation measures the dependence strength:

$$\beta_{xu} = \frac{\|x\|}{\|u\|}. \quad (4.1)$$

Again we obtain a coupling parameter β (the strength); when $\beta_{xu} \rightarrow 1$ the dependence is complete.

5 Usage of the Metric: A Toy Example

Let us illustrate how to use (Figure 7) the coupling parameter β in a simple case. Oriented links are the directions of dependencies among nodes, β_{ij} is the strength of dependence from node i to node j . At this point, adding the normalized β values of the links along a path we get a dependence value for that path between two nodes with regard for a certain dependence order.

Path search is the most computationally expensive procedure to perform on the oriented graph. As seen in paragraph 2 there exist many algorithms able to search for this type of path in a efficient way. Moreover, limitation such as in Figure 6 reduces the range of compatible paths.

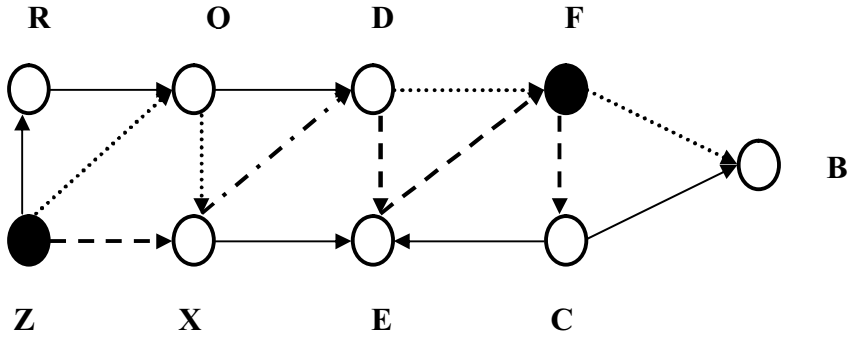


Fig. 7. 4th order analysis between nodes Z and F (target). The first maximum dependence path is dotted, the second max dependence path is dashed. Nodes X, D and link XD are common to both paths.

We are interested in revealing the maximum dependence paths (MDP) between two nodes (because these are the most probable path for a cascading failure), ordering them in descending values. For example in Figure 7 considering only two paths for clarity reasons, we have the first MDP from Z to the target node F (links ZQ, QX, XD, DF) with a total of 1.8 and the second MDP (links ZX, XD, DE, EF) with a total of 0.98 (4th order dependence). Link XD and nodes X, Z, are common to both maximum paths, thus these elements should be regarded as critic elements.

The normalized values of physical quantities for links of the first path are:

$$XD = [.20, .80, .30]$$

$$QX = [.10, 0.1, .20]$$

$$ZQ = [.189, .32, .21]$$

$$DF = [.10, .24, .01]$$

$$FB = [.51, .13, .21]$$

The normalizing values of physical quantities for links of the second path are:

$$XD = [.20, .80, .30]$$

$$ZX = [.42, .30, .16]$$

$$EF = [.20, .10, .21]$$

$$DE = [.123, .21, .30]$$

$$FC = [.20, .13, .20]$$

The normalized values of dependencies calculated are

$$\beta_{ZQ} = 0.1337$$

$$\beta_{QX} = 1.000$$

$$\beta_{XD} = 0.1123$$

$$\beta_{DF} = 0.5555$$

$$\beta_{ZX} = 0.4142$$

$$\beta_{XD} = 0.1123$$

$$\beta_{DE} = 0.2026$$

$$\beta_{EF} = 0.2588.$$

Another analysis can be performed when delays come into play just replacing β_{ij} with time delays τ_{ij} . Naturally every element of vectors x and u has its own time delay, therefore some sort of average must be carried out to obtain a unique value τ_{ij} .

After the average we can execute the same procedure as above. In Figure 7 we have for the first MDP (links ZR (10s), RQ (200s), QX (24s), XD (0.1s), DF (37s), FB (7s)) a total of 278.1s; for the second MDP (links ZX (3s), XD (0.1s), DE(9), EF (20s), FC (4s)) a total of 36.1s, for the 4th order. Deciding which one of the two MDP is the most important is referred to the risk policy.

Summarizing, the nodes dependences list (in descend order) is QX, DF, ZX, EF, DE, ZQ, XD, the non normalized maximum coupling strength is $\beta_{QX} = 3.9$. Max 4th Order Dependence Paths between nodes Z and F (the target node) are: {ZQ, QX, XD, DF} and {ZX, XD, DE, EF} (the fastest path). Critical nodes are: Z, X, critical link is ZX.

This analysis has to be iterated for any couple of nodes and any desired dependency order, producing an ordered report. Dangerous paths and critical elements are immediately pointed out enabling proper off line resilience actions. During the on line operation the above parameter can be used as early warning alarms or to perform simulations. Note that eventually deactivated nodes during operations are taken into count setting to extremely low values the related inputs/outputs.

6 Conclusions

Today infrastructure protection and resilience is a major issue in some of the most important research programs in the world. Despite these efforts [20, 21, 22], many researches still seem to concentrate on specific networks by means of statistical methodologies or advanced simulation models, but it is now clear that is impossible to

understand an isolate network behaviour. Too much high level or too low level models do not capture the most interesting features of service outages. Besides, standard control system methodologies requiring a formal knowledge of the interconnected plants are not useful for large problems such as CI protection because decoupling the dynamics is very difficult if not impossible (see [19] for the distribute generation case). Thus the lack of control tools compels to relax the problem from a control problem to an interdependency analysis. The necessary [8], preliminary step, is the implementation of a metric for quantifying interdependencies in generic networks. Here we propose a data driven metrics based on known direct functional input-output relations among nodes derived from the generic constitutive equations of the systems defining the strength of the dependence as the norm of the expression of the system. The norm is strictly related to input output physical and abstract quantities at the time t realizing a data driven approach, discarding the internal node dynamics. It was also shown how this approach easily extracts important parameters and is suited to the on line analysis for large networks.

Acknowledgements

Authors gratefully acknowledge C. Balducelli, S. Bologna, E. Ciancamerla, G. D'Agostino, E. Djanbazova, A. Fioriti, M. Laganà, M. Minichino, M. Ruscitti, R. Setola, A. Tripi, G. Vicoli for useful discussions. The authors also acknowledge many fruitful discussions with R. Setola (Campus Biomedico Roma).

References

1. Barabasi, A., Albert, R.: Emergence of scaling in random networks. *Science* 286 (1999)
2. Carreras, A., Dobson, A.: Evidence of self organized criticality in power systems. In: Hawaii International Conference on System Science (2001)
3. Carreras, A., Dobson, A.: Critical points in an electric power transmission model for cascading failure. *Chaos* 12, 985–992 (2002)
4. Jamakovic, J.: On the relationships between topological metrics in real world networks. In: Proceedings of ECCS, Dresden (July 2007)
5. Motter, A.: Cascade based attacks on complex networks. *Phy. Rev. E* 66, 065102 (2002)
6. Newman, D.: Self organizing criticality as a paradigm of turbulent transport. *Physics of Plasma* 3, 1858–1862 (1996)
7. Osorio, L.: Seismic response of critical interdependent networks. *Earthquake Eng. Struct. Dyn.* 36, 285–293 (2007)
8. Rinaldi, J., et al.: Identifying critical infrastructure interdependencies. *IEEE Control System Magazine* 21, 337–351 (2001)
9. Rocco, S., Moreno, J.: Network reliability assessment. *Reliability Eng.* 78, 3–7 (2002)
10. Rosas-Casals, R.: Topological vulnerability of the EU power grid, DELIS-TR-437, EU Integrated Project (2006)
11. Ruberti, A., Isidori, A.: *Teoria dei Sistemi*, Boringhieri (1979)
12. Strogatz, S.: Exploring complex networks. *Nature* 3, 410–412 (2001)
13. Thom, R.: *Stabilità strutturale e morfogenesi*, Einaudi, Milano (1975)

14. Zio, E.: From complexity to reliability efficiency. *Int. J. Critical Infrastructures* 3, 3–31 (2007)
15. Wu, W.: Nonlinear system theory: another look at dependence. In: *PNAS*, vol. 102 (2005)
16. Pecora, L., Carrol, T.: Synchronization in chaotic systems. *Phy. Rev. Lett.* 64, 821 (1990)
17. Arnhold, J., Grassberger, K., Lehmertz, K., Elger, C.: vol. 134, p. 419 (1999)
18. Fioriti, V., Ruzzante, S., Castorini, E., Marchei, E., Rosato, V.: Stability of distributed generation networks using Kuramoto model. In: Setola, R., Geretshuber, S. (eds.) *CRITIS 2008. LNCS*, vol. 5508, pp. 14–23. Springer, Heidelberg (2009)
19. Cardell, J., Ilic, M.: Maintaining Stability with Distribute Generation. In: *IEEE Power Eng. Soc. Meeting* (2004)
20. Italian Government MURST CRESCO Project, UE Commission FP6 SAFEGUARD Project, UE Commission FP7 IRRIS Project, UE JLS Directorate EPCIP MIA Project, US DoE TCIP Project, US DoE TRUST Project
21. Ciancamerla, E., Minichino, M., et al.: IRRIS Deliverable 2.2.2 UE Commission FP7, IRRIS Project (2006)
22. Bloomfield, R., et al.: Analysis of Critical Infrastructure dependence. In: *CRITIS 2008 Conference, Frascati* (2008)
23. Takens, F.: *Lecture notes in mathematics*, vol. 898, p. 366. Springer, Berlin (1981)
24. Quiroga, R.: Learning driver-response relationship from synchronized patterns. *Phy. Rev. E* 61, 5142 (2000)

Security Analysis of Open Building Automation Systems

Wolfgang Granzer and Wolfgang Kastner*

Vienna University of Technology,
Institute of Computer Aided Automation, Automation Systems Group
Treitlstr. 1–3, 1040 Vienna, Austria
`{w,k}@auto.tuwien.ac.at`
<https://www.auto.tuwien.ac.at/>

Abstract. With the integration of security-critical services into Building Automation Systems (BAS), the demands on the underlying network technologies increase rapidly. Relying on physically isolated networks and on “Security by Obscurity”, as it is still common today, is by no means an adequate solution. To be reliable and robust against malicious manipulations, the used communication services must support advanced security mechanisms that counteract potential security threats. This paper identifies important security requirements and challenges within the building automation domain. Based on this analysis, state-of-the-art technologies are carefully examined. Finally, an outlook on advanced security concepts is given.

Keywords: Building Automation, Embedded Networks, Security.

1 Introduction

Building Automation Systems (BAS) aim at improving control and management of mechanical and electrical systems in buildings – more generally, interaction among all kinds of devices typically found there. The core application area is the automatic control of traditional building services like lighting/shading as well as Heating, Ventilation, and Air Conditioning (HVAC). Services from the security domain (e.g., intrusion alarm systems, access control) are often provided by separated, application-specific subsystems.

Today, a trend towards the integration of these separated subsystems into the core BAS can be observed. The advantages of such a resulting “all-in-one” BAS are manifold. First, the application area can be extended since services from the security domain can also be served by such an all-in-one system. Second, traditional services like HVAC and lighting/shading are also improved since a comprehensive security concept will also protect the BAS against among others vandalism acts.

* This work was funded by FWF (Österreichischer Fonds zur Förderung der Wissenschaftlichen Forschung; Austrian Science Foundation) under the project P19673.

To be able to fulfill the requirements of such a secure all-in-one BAS, the underlying technologies must be reliable and robust against malicious manipulations. However, available BAS installations rely on physical isolation and “Security by Obscurity”. This is obviously unacceptable within modern BAS since preventing physical access to the network by isolation is not always possible (e.g., WLANs, public buildings). Moreover, “Security by Obscurity” is a technique that (if at all) provides only temporary protection.

This paper provides a comprehensive analysis of the integrated security concepts of available BAS solutions. First, important security requirements as well as domain-specific challenges are identified. Based on these requirements, existing standards within the BAS domain are analyzed. This analysis is focused on the most important open BAS standards (i.e., BACnet, LonWorks, KNX, and ZigBee). The paper is concluded with a summary about the opportunities and drawbacks of today’s BAS technologies regarding their suitability within security-critical environments.

2 Security Requirements and Domain-Specific Challenges

To be able to serve as a BAS solution for security-critical environments, the used network technologies must fulfill different security requirements. Based on [1,2,3], the following *Functional Requirements (FR)* for secure BAS are identified. First, the communication entities that want to securely exchange data (e.g., sensors, actuators, controllers, management devices) must prove their identities i.e., it must be verified whether the entities are what they claim to be (*entity authentication FR1*¹). Then, it must be verified if the entities have the necessary access rights to participate in the communication (*authorization FR2*). Afterwards, the data exchanged between authenticated entities must be protected in a secure manner. This is done by establishing a so called *secured channel*. A secured channel uses non-cryptographic (e.g., physical or organizational measures) and/or cryptographic techniques to protect data against security attacks while they are transmitted over a network. Depending on the requirements of the application, a secured channel guarantees the following security objectives:

- *Data integrity (FR3)* proves that the data was not modified.
- *Data origin authentication (FR4)* is a stronger form of data integrity where a receiver can also verify the data origin i.e., the data source.
- *Data freshness (FR5)* guarantees that the transmitted data is recent and valid at the time of transmission. Replaying of previously sent data can be detected by the entities.
- *Data confidentiality (FR6)* ensures that only authorized entities have access to confidential information. A typical example of confidential information would be a PIN code that is entered by a user at a security door.
- *Data availability (FR7)* guarantees that the communication is not disturbed and that the authorized entities have access to the data.

¹ This numbering style is used throughout the paper to uniquely identify the different requirements and challenges.

Besides these functional requirements, various *Domain-Specific Challenges (DC)* that reflect the characteristics of the environment exist. They are the main reasons why it is not possible to directly use security concepts from other domains. For example, mapping Information Technology (IT) security mechanisms to the BAS domain is not possible in a native way since they are tailored to the use in the IT world. This is also true for closely related domains like industrial automation. The domain-specific characteristics of BAS lead to the following challenges. BAS typically consist of embedded networks where *low-power embedded devices (DC1)* are used. Due to reasons of cost and space efficiency, these devices are equipped with limited system resources. This concerns the amount of available memory, processing power but also the power supply (e.g., bus-, battery, or self-powered devices). However, since security mechanisms are computationally intensive (especially cryptographic algorithms), their use must not exceed the available resources. Therefore, it is essential to find a good balance between a required level of security and available resources (“*good enough security*”). For example, if the non-disclosure of the transmitted data is not strictly necessary, data confidentiality is unnecessary.

An important difference between BAS and communication systems within other domains is the required support for different *communication models (DC2)*. While in other domains the client/server model is predominantly used, group communication patterns based on multicast or broadcast are well-established in the BAS domain. This also concerns the amount of devices used within a network. BAS usually consist of hundreds or even thousands of devices. Thus, *scalability (DC3)* of the integrated security mechanisms and security management services (e.g., distribution of secret keys) is of major concern.

IT security mechanisms are geared towards different requirements regarding the used network technology. While in the IT world IP based network protocols are dominant, *non IP field networks (DC4)* are mainly used at the field level within the BAS domain. The main reasons for the use of such networks are robustness, flexibility, and cost efficiency.

Finally, the required *Quality-of-Service (QoS) (DC5)* parameters of BAS field networks differ from the IT/office world, too. In the IT/office domain, the data volume to be transferred is commonly high (in the order of mega- or gigabytes) with usually no real-time requirements. Control data typically transmitted in BAS has a small volume (in the order of some bytes) with perhaps soft real-time requirements (e.g., the reaction time of a lighting system). Additionally, QoS properties like *reliability* and *ordering* of messages have to be considered. While these QoS properties are normally of less concern in the IT/office world, they may be an important issue in the BAS domain.

3 Security in Home and Building Automation Standards

Today, many different BAS protocol standards exist. The most important open ones that can be considered as reasonable solutions for all-in-one systems are BACnet [4,5], LonWorks [6,7], KNX [8,9], and ZigBee [10].

3.1 BACnet

BACnet offers several services which pretend to provide support for data confidentiality, data origin authentication (and thus data integrity), and data freshness as well as entity authentication [4,11]. Authorization is provided on a per-device basis. The security mechanisms are based on Data Encryption Standard (DES) and a trusted key server which is responsible for managing session keys. These session keys are used to secure the transmitted data between two devices. To establish a secure connection to the key server, each device must own an initial secret key.

Due to several security flaws [11,12,13], this security concept was completely replaced by a new one that is defined in BACnet Addendum g [14]. At the time of writing, BACnet Addendum g has finished the 4th public review process and is now waiting for final publication. BACnet Addendum g specifies security services that are designed to be applicable to all BACnet media and device types. To protect the transmitted data, symmetric cryptographic algorithms are used exclusively. The required shared secret keys have to be distributed in advance or they have to be retrieved from a so called *key server* during runtime. In BACnet Addendum g, six different key types are distinguished. **General-Network-Access** keys are shared between all members of a network. **User-Authenticated** keys are used for requests where the user identity can be assumed to be authenticated properly. The user authentication has to be performed by an external mechanism (e.g., via a user interface). Alternatively, if a device does not have a user interface, the user identity can be configured directly at the device. **Application-Specific** keys are dedicated to a dedicated application domain (e.g., HVAC or access control). These keys are only distributed to a subset of devices that require a higher level of security. **Installation** keys are temporally used for management purposes. **Distribution** keys have the aim to secure the retrieval of other keys from the online key server. Finally, **Device-Master** keys are only used to receive **Distribution** keys. Since they act as initial secrets, their distribution must be done within a physically secured environment.

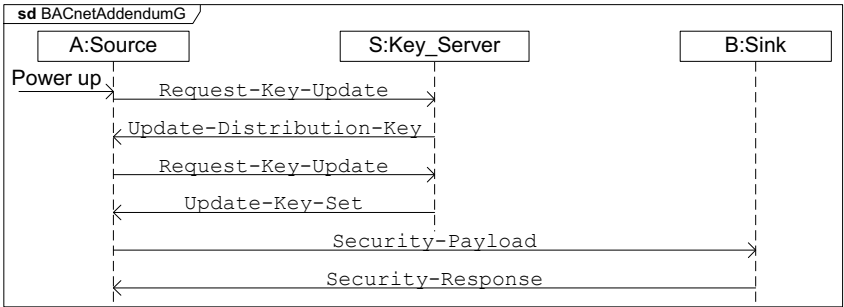


Fig. 1. Security services in BACnet Addendum g

BACnet Addendum g specifies eight secure communication services that are incorporated into the network layer of BACnet. The **Security-Payload** service is used to securely transmit data messages. To respond to them, the **Security-Response** service is available which indicates either the successful retrieval of a secured message or an error condition. The **Challenge-Request** service is used to verify the identity of a device. The device that is challenged has to answer with a **Security-Response** message that contains the result of the challenge. To request the distribution of the secret keys from the key server, the **Request-Key-Update** service is available. Upon retrieval of a **Request-Key-Update**, the key server responds with an **Update-Key-Set** or with an **Update-Distribution-Key** message which contains the requested key set. These two services can also be used by the key server to force key changes. Finally, the **Request-Master-Key** and **Set-Master-Key** are used to change the **Device-Master** key. However, since these two services are not secured at all, their use has to be limited to physically secured environments.

Fig. 1 shows an example how these security services can be used. After having powered up, device *A* requests a **Distribution** key from the key server *S* by sending a **Request-Key-Update** message (secured with its **Device-Master** key). The key server validates the request and transmits a newly created **Distribution** key to *A*. Afterwards, *A* sends another **Request-Key-Update** message to retrieve the current keys. This request is secured using the **Distribution** key retrieved before. After having received the key set from the key server, *A* is now able to securely communicate with device *B* using the appropriate key. Note that it is assumed that device *B* is also in possession of the used key (e.g., **General-Network-Access** or **Application-Specific** key).

Network messages are classified into *plain*, *signed*, and *encrypted* messages. While plain messages are not secured at all, signed messages provide data integrity and freshness. To guarantee data integrity, Keyed-Hash Message Authentication Code (HMAC) in combination with Message Digest 5 (MD5) or Secure Hash Algorithm (SHA) is used. Data freshness is achieved by using a timestamp (32 bit standard UNIX timestamp) in combination with a 32 bit message ID. Encrypted messages are additionally encrypted using Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode. Entity authentication is implicitly guaranteed due to the used symmetric algorithms and due to the use of so called device instance numbers. Device instance numbers uniquely identify secure BACnet devices and are assigned to the devices independently.

3.2 LonWorks

The communication protocol of LonWorks (called LonTalk) provides a rudimentary security concept based on a four step challenge-response protocol. During this protocol, the identity of the sender is verified. Furthermore, it pretends to guarantee data integrity and freshness. Fig. 2(a) shows the different steps: a sender which desires to secure a request sets the so called authentication bit of the corresponding message. All receivers have to reply with a 64 bit random number. The sender receives these random numbers and individually calculates

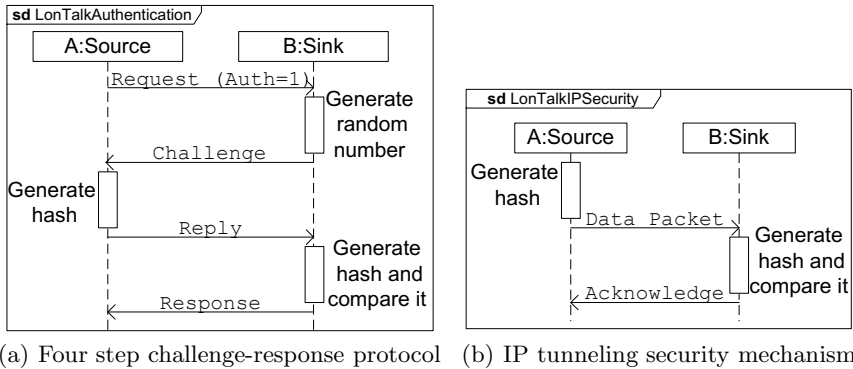


Fig. 2. LonTalk security mechanisms

a 64 bit hash value over the content of the message and the random number using a shared secret key. These hash values are sent back to the receivers where the same calculation is performed and compared with the previously received value.

In addition to the basic challenge-response protocol, the IP tunneling scheme of LonTalk defines its own security mechanism (cf. Fig. 2(b)). It uses MD5 together with a shared secret to calculate a hash value. This hash value is sent together with the message to the intended receiver(s). After having received a secured message, the receiver calculates its own hash value using the same shared secret and compares it with the received one. If both values are equal, the message is accepted – otherwise it is discarded. Note that the four step challenge-response mechanism mentioned above is not used here.

3.3 KNX

KNX only provides a basic access protection scheme based on clear text passwords (cf. Fig. 3). Up to 255 different access levels can be defined, each of them is associated with a different (otherwise unspecified) set of privileges. Access level 0 has the highest privilege and access level 255 is the lowest one. For each of these access levels, a 4 byte password can be specified. This scheme is only available for engineering communication. Control data exchange remains insecure.

To be able to use IP networks for KNX installations, KNXnet/IP has been introduced. In the corresponding specification [8], some rudimentary security guidelines are additionally presented. These guidelines are based on network isolation (e.g., use of firewalls or KNXnet/IP only Intranets) and on “Security by Obscurity” (e.g., use of non-standard IP addresses, rely on the missing expertise of an attacker). Since preventing physical access to the network by isolation is not always possible (e.g., WLAN) and “Security by Obscurity” is a technique that (if at all) provides only temporary protection, these security guidelines do not provide an effective protection.

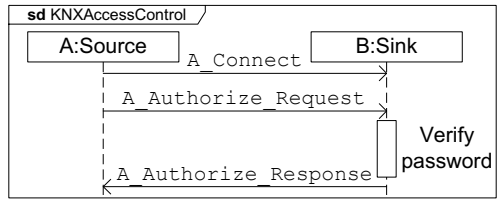


Fig. 3. Access control mechanism of KNX

3.4 ZigBee

ZigBee (version 2007) is the most well-known protocol that builds upon IEEE 802.15.4. ZigBee uses the data link layer of IEEE 802.15.4 (version 2003) and enhances the available features by specifying an application layer and a network layer. Additionally, new services that provide the opportunity for multi-hop routing and advanced security services have been added. Although, while ZigBee uses the transmission services of the data link layer of IEEE 802.15.4, it defines its own security architecture that is independent from IEEE 802.15.4. Thus, the security services provided by IEEE 802.15.4 are entirely not used.

The security concept of ZigBee is exclusively based on symmetric cryptographic schemes. In particular, AES and a variant of Counter with CBC-MAC (CCM*) are used. Entity authentication as well as data origin authentication, freshness, and confidentiality are provided at the network and/or application layer. Additionally, ZigBee provides services for management and distribution of the required shared secret keys. Depending on their use, ZigBee distinguishes three different key types. *Link keys* are shared between two devices. They are used to secure communication between them. *Network keys* provide security across the whole network segment. Finally, so called *master keys* are optionally available. Master keys are only required during the establishment of link keys.

Beside the possibility to manually install shared secret keys in advance, it is possible to retrieve secret keys during runtime. This runtime distribution of shared secret keys is handled by a single entity called *Trust Center*. To exchange secret keys, three different distribution methods are available in ZigBee:

- *Pre-installation*: Here, the keys are uploaded to the devices before runtime using, for instance, a proprietary management tool. The exact method how pre-installation is performed is not defined by the ZigBee specification.
- *Key-transport*: Using key-transport, the trust center sends the keys directly to the devices using a dedicated communication service. Key-transport is used to distribute the actual network key during the device joining process and to distribute link keys during runtime. Fig. 4(a) shows an example how key-transport can be performed to distribute a link key. To retrieve a link key, the initiating device sends a **Request-Key** message to the trust center. The trust center generates a new link key and distributes it to both devices using a **Transport-Key** message. The message is secured with the trust center link key that is shared between the trust center and the corresponding devices.

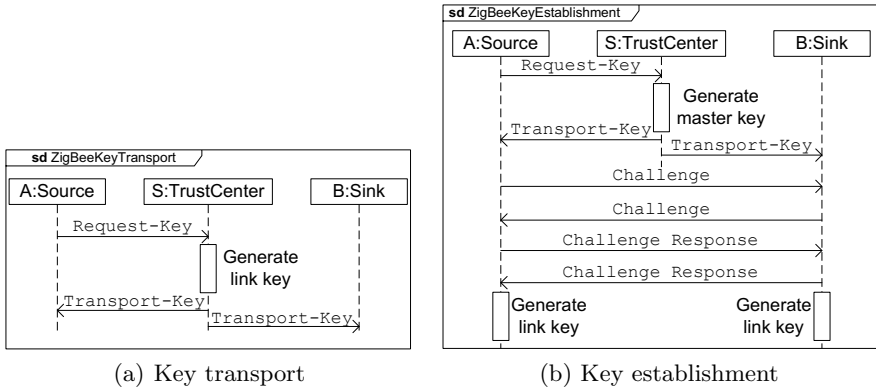


Fig. 4. ZigBee security mechanisms

- *Key-establishment*: Key-establishment is only available for link keys. In contrast to key-transport, both devices are involved in the key generation process. The key-establishment is performed using the so called Symmetric-Key Key Exchange (SKKE) protocol (Fig. 4(b)). To start the key-establishment process, the initiating device sends a **Request-Key** message to the trust center. The trust center generates a master key and distributes it to both devices using the **Transport-Key** service. Afterwards, the devices start the SKKE protocol. First, each device generates a random challenge that is sent to the other device. Using this challenge and the previously retrieved master key, each device calculates a challenge response which is sent to the other device. After having retrieved the challenge response, both devices verify it. If it is valid, a link key is calculated out of both challenges which can later be used to secure the communication between the two devices.

To be able to securely retrieve network, master, or link keys from a trust center, the requesting device must share a link or master key with the trust center. These initial trust center keys can either be pre-installed or distributed using unsecured key-transport messages. However, in the latter case, the exchange has to be done in a physically secured environment.

4 Evaluation

To evaluate available BAS standards regarding their suitability within security-critical environments, their integrated security concepts were analyzed with respect to the requirements and challenges identified in Section 2. Fig. 5 summarizes the results of this security analysis.² At the left hand side, the BAS standards that have been described in Section 3 are listed. To show that using security mechanisms from other domains is not trivially possible, two of the most

² “+” denotes that the requirement or challenge is (nearly) satisfied, “~” means that it is only partly fulfilled, and “-” implies that the used mechanism is insecure.

	BAS				IT mechanisms	
	BACnet	LonTalk	KNX	ZigBee	IPsec	TLS
Entity authentication (FR1)	+	-	-	+	+	+
Authorization (FR2)	~	-	~	~	+	+
Data integrity (FR3)	+	~	-	+	+	+
Data origin authentication (FR4)	~	-	-	+	+	+
Data freshness (FR5)	+	~	-	+	+	+
Data confidentiality (FR6)	+	-	-	+	+	+
Data availability (FR7)	-	-	-	-	-	-
Embedded devices (DC1)	+	+	+	+	-	~
Communication models (DC2)	-	~	-	-	~	-
Scalability (DC3)	-	-	-	-	-	-
Non IP networks (DC4)	+	+	+	+	-	+
QoS parameters (DC5)	-	~	-	~	~	~

Fig. 5. Evaluation of available standards

important IT security mechanisms were investigated, too. The corresponding results are shown at the right hand side of the figure.

4.1 BACnet Addendum g

BACnet Addendum g provides a solid base for securing BAS. However, the following aspects are missing or left open:

- *Authorization (FR2)*: The distribution of the keys is handled by the key server. The actual distribution to the devices predefines which devices are able to communicate with each other and which devices are excluded from a relationship. As a result, the assignment of the keys to the devices defines the devices' access rights and thus their authorization. Since this procedure is not specified by the standard, authorization has to be realized by the application.
- *Data origin authentication (FR4)*: Guaranteeing data origin authentication is only possible if a key is limited to two devices. If, for example, the **General-Network-Access** key or an **Application** key that is distributed to multiple devices are used, the sender cannot be identified in a secure manner.
- *Data freshness (FR5)*: The security mechanisms require the existence of (loosely) synchronized device clocks. Otherwise, data freshness cannot be guaranteed since the used mechanisms rely on timestamps.
- *Data availability (FR7)*: Mechanisms to protect against interruption attacks (e.g., Denial-of-Service (DoS) attacks) are not supported. Therefore, data availability cannot be guaranteed.
- *Communication models (DC2)*: BACnet only provides support for the client/server model – exchanging data within groups is not supported.
- *Scalability (DC3)*: The use of a single key server introduces a single-point-of-failure. Therefore, a scheme based on multiple key servers is desirable. While the use of multiple key servers is possible, the realization of such a concept is not specified. Important details like synchronization of key servers and

the selection of the key server to be used (especially in case of a faulty key server) are not discussed.

- *QoS parameters (DC5)*: Since multicast is not supported at all, reliability and ordering within in communication groups are not supported, too. For broadcast communication, QoS features cannot be specified.

4.2 LonWorks

LonTalk's security concept suffers the following security flaws [11,15]:

- *Entity authentication (FR1)*: The used protocol only supports the verification of the sender's identity. The identity of the receiver cannot be checked. Furthermore, the challenge-response mechanism can only be initiated by the sender. A receiver does not have the opportunity to demand secured requests.
- *Authorization (FR2)*: Authorization is not supported since the same key is used for all LonTalk devices.
- *Date integrity (FR3) and freshness (FR5)*: The length of the used shared secret keys is limited to 48 bits which is too short to avoid brute force attacks. Additionally, only the data portion of the application layer is used as input for the hash calculation. Headers from other layers including the address information are not protected.
- *Data origin authentication (FR4)*: Each device can only use one authentication key. This means that all devices that want to communicate with each other must share the same secret key. As a result, data origin authentication cannot be guaranteed in networks with more than two members.
- *Data confidentiality (FR6)*: Disclosure of confidential data cannot be avoided, since the data is transmitted in clear text.
- *Data availability (FR7)*: There are no countermeasures that avoid an interruption of communication.
- *Communication models (DC2)*: The usage of the authentication protocol is restricted to acknowledged services. If an unacknowledged transmission mode is used, the identity of the sender cannot be verified.
- *Scalability (DC3)*: Using authenticated multicast, each receiver generates its own random number and sends it to the sender. As a result, the sender must respond to all receivers with an individual calculated hash value. If a communication group contains n members, the sender must calculate $(n - 1)$ hash values. Furthermore, it is not possible to establish communication sessions and so, it is always necessary to transmit four messages for secured requests.
- *QoS parameters (DC5)*: LonTalk provides support for acknowledged communication services. However, a defined ordering within multicast groups cannot be guaranteed.

While these security flaws are related to the standard challenge-response protocol of LonTalk, most of them also apply to the security mechanism of the IP tunnelling scheme of LonTalk. Instead of the used cryptographic algorithms, LonTalk/IP uses MD5. However, since MD5 is not collision resistant, it is insecure, too. Another difference is that data freshness is not guaranteed at all

due to the absence of a nonce (e.g., random number). Therefore, the security mechanism of LonTalk/IP cannot be considered as an improvement.

4.3 KNX

Since KNX's access protection is very rudimentary, it does not provide the necessary mechanisms to guarantee a secure environment [15]:

- *Entity authentication (FR1)*: It is not provided since the identity of the receiver is not verified.
- *Authorization (FR2)*: The passwords are transmitted in clear text. If an adversary has access to the network, the adversary can simply intercept and retrieve the transmitted password to impersonate devices. Furthermore, the source address of a transmitted message can be spoofed very easily and so, an adversary can inject malicious messages without knowing the password.
- *Data integrity (FR3), data origin authentication (FR4), data freshness (FR5), data confidentiality (FR6)*: These objectives are not guaranteed at all.
- *Data availability (FR7)*: Interruption attacks cannot be avoided.
- *Communication models (DC2)*: The access protection mechanism cannot be applied to control data communication in KNX. An unauthorized use of these services cannot be avoided.
- *Scalability (DC3)*: KNX does not support mechanisms to manage, generate, and distribute passwords in a secure manner. Therefore, the passwords must be specified manually. It is up to the system administrator to guarantee that this configuration is performed in a physically secured environment. Furthermore, the single management tool called ETS needs to be used. ETS uses only one password for the whole installation. Hence, the rudimentary access protection scheme does not scale to large systems since compromising a single device discloses the password of all devices.
- *QoS parameters (DC5)*: KNX only provides acknowledged communication services for unicast communication. For multicast or broadcast communication, only unacknowledged end-to-end communication services are available. A defined ordering is also not possible for these services.

4.4 ZigBee

The security concept of ZigBee provides a solid base for secure communication. However, the following requirements and challenges are not satisfied:

- *Authorization (FR2)*: The smallest security context in ZigBee is a device. Using different secret keys for different user applications on a single device is not possible. Therefore, access control is only provided on a per-device basis.
- *Data integrity (FR3)*: The security services provided by IEEE 802.15.4 are not used by ZigBee. As a result, the data link header is not secured since ZigBee only protects the network and/or application layer parts of the messages. Furthermore, data link layer services like sending beacon frames and associate requests are not secured. Therefore, security threats that are dedicated

to the data link header or to data link services cannot be avoided (e.g., re-routing of network traffic).

- *Data availability (FR7)*: Interruption threats are not considered in ZigBee. Especially the joining procedure is vulnerable to DoS attacks. The first part of the joining process (i.e., address assignment, synchronization with coordinator) is not secured since entity authentication is only provided afterwards.
- *Communication models (DC2)*: While ZigBee defines a multicast communication service, it is not clear how group communication is secured in ZigBee. It seems that the only possibility is to use the network key. However, a secure separation between different multicast groups is not possible if the network key is used. Furthermore, data origin authentication cannot be guaranteed. Link keys cannot be used to secure multicast communication, since link keys can only be shared between two devices.
- *Scalability (DC3)*: Key management is handled by a single trust center which may result in a single-point-of-failure. Furthermore, in wide-range networks, multiple hops may be necessary to reach the trust center. Therefore, a security concept based on multiple trust centers is desirable.
- *QoS parameters (DC5)*: ZigBee provides a mechanism to detect duplicates. Acknowledged communication services are only available for unicast communication – acknowledged multicast or broadcast services are not provided.

4.5 Security Mechanisms for the IT Domain

Due to the widespread use of the Internet, security has been a major research field in the IT world for years. Therefore, many well-established IT security mechanisms exist. If available BAS standards do not provide the necessary countermeasures against security attacks, an obvious solution would be the use of already existing security schemes from the IT world. Therefore, two of the most well-known IT security extension that may come into consideration for the BAS domain are presented. These are Internet Protocol Security (IPsec) [16] and Transport Layer Security (TLS) [17].

As shown in Fig. 5, both extensions provide a solid base for securing the communication. However, since both mechanisms are tailored towards the use within IP networks, the domain-specific challenges of BAS are not fully satisfied:

- *Data availability (FR7)*: Counteracting interruption attacks and thus guaranteeing data availability is out of the scope of both extensions.
- *Embedded devices (DC1)*: While special implementations of TLS are suitable for embedded devices [18], porting IPsec to embedded environments is not easy to achieve. The main reasons are the introduced protocol overhead and the computational cost of the used cryptographic algorithms.
- *Communication models (DC2)*: Using IPsec with multicast is only possible with special implementations [19]. TLS cannot be used to secure communication within groups since it is dedicated to the client/server model.
- *Scalability (DC3)*: Both extensions demand an existing key server infrastructure that is used to manage the required security primitives. However, the

exact implementation is not specified by the standards. Therefore, special implementations that scale to large systems are necessary.

- *Non IP networks (DC4)*: Due to its nature, IPsec is dedicated to the use for the IP protocol. Using it within field networks requires major changes in the current IPsec protocol.
- *QoS parameters (DC5)*: Since IPsec is located at OSI layer 3, guaranteeing reliability and a defined ordering of messages are left to higher protocol layers. TLS uses sequence numbers to detect missing messages or duplicates. However, providing a retransmission service or a defined ordering of messages are left to the other protocol layers, too.

5 Conclusion

As shown in this paper, available BAS solutions do not satisfy the demands of security-critical applications. While some technologies provide a solid base for a secure communication (e.g., BACnet, ZigBee), there are even communication standards where security is still neglected (e.g., LonWorks, KNX). To reduce this lack of security, a possible solution is to enhance available BAS technologies by integrating existing security concepts from other domains (e.g., from the IT world). However, due to the domain-specific challenges, mapping available security mechanism into the BAS domain is not trivially possible.

As a result, many important issues remain unsolved. As shown in Fig. 5, guaranteeing data availability is not provided by any solution. The main reason is that relying on cryptographic techniques does not fully counteract DoS attacks. Another major problem is that most security concepts are based on the use of a single key server. However, within large networks, such a single entity results in a single-point-of-failure. Finally, guaranteed QoS parameters like reliability or a defined ordering of messages are also not fully supported by available solutions. However, these features are of great interest for all-in-one BAS solutions especially if services from the safety domain need to be integrated, too [20].

While this paper analyzes the security features of available BAS standards, the development of new approaches and schemes that overcome the lack of security of current solutions is already under way. To achieve data availability, an advanced security concept based on organizational countermeasures that counteracts DoS attacks is presented in [21]. Furthermore, to eliminate a single-point-of-failure within the used secret key management protocols, two concepts that are based on multiple, redundant key servers have also been published [22,23].

References

1. Pfleeger, C.P., Pfleeger, S.L.: Security in Computing, 4th edn. Prentice Hall, Englewood Cliffs (2006)
2. Dzung, D., Naedele, M., Hof, T.V., Crevatin, M.: Security for Industrial Communication Systems. Proceedings of the IEEE 93(6), 1152–1177 (2005)

3. Perrig, A., Szewczyk, R., Tygar, J.D., Wen, V., Culler, D.E.: SPINS: Security Protocols for Sensor Networks. In: 7th Annual International Conference on Mobile Computing and Networking, pp. 189–199 (2001)
4. BACnet – A Data Communication Protocol for Building Automation and Control Networks, ANSI/ASHRAE 135-2008 (2008)
5. Building Automation and Control Systems (BACS) – Part 5: Data Communication Protocol. ISO 16484-5 (2007)
6. Control Network Protocol Specification. ANSI/EIA/CEA 709 Rev. B (2002)
7. Open Data Communication in Building Automation, Controls and Building Management – Control Network Protocol. ISO/IEC 14908 (2008)
8. Information Technology – Home Electronic Systems (HES) Architecture. ISO/IEC 14543-3 (2006)
9. KNX Specification Version 2.0. Konnex Association, Diegem (2009)
10. ZigBee Specification, ZigBee Alliance, San Ramon (2007)
11. Schwaiger, C., Treytl, A.: Smart Card Based Security for Fieldbus Systems. In: 9th IEEE Conference on Emerging Technologies and Factory Automation, vol. 1, pp. 398–406 (2003)
12. Holmberg, D.G.: BACnet Wide Area Network Security Threat Assessment. Technical report, National Institute of Standards and Technology, NISTIR 7009 (2003)
13. Zachary, J., Brooks, R., Thompson, D.: Secure Integration of Building Networks into the Global Internet. Technical report, National Institute of Standards and Technology, NIST GCR 02-837 (2002)
14. BACnet – A Data Communication Protocol for Building Automation and Control Networks. ANSI/ASHRAE 135-2008: Addendum g (2009)
15. Granzer, W., Kastner, W., Neugschwandtner, G., Praus, F.: Security in Networked Building Automation Systems. In: 6th IEEE International Workshop on Factory Communication Systems, pp. 283–292 (2006)
16. Kent, S., Seo, K.: Security Architecture for the Internet Protocol. RFC 4301 (2005)
17. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (2008)
18. Gupta, V., Wurm, M., Zhu, Y., Millard, M., Fung, S., Gura, N., Eberle, H., Shantz, S.C.: Sizzle: A Standards-based End-to-End Security Architecture for the Embedded Internet. *Pervasive and Mobile Computing* 1(4), 425–445 (2005)
19. Weis, B., Gross, G., Ignjatovic, D.: Multicast Extensions to the Security Architecture for the Internet Protocol. RFC 5374 (2008)
20. Kastner, W., Novak, T.: Functional Safety in Building Automation. In: 14th IEEE Conference on Emerging Technologies and Factory Automation, pp. 1–8 (2009)
21. Granzer, W., Reinisch, C., Kastner, W.: Denial-of-Service in Automation Systems. In: 13th IEEE Conference on Emerging Technologies and Factory Automation, pp. 468–471 (2008)
22. Granzer, W., Reinisch, C., Kastner, W.: Key Set Management in Networked Building Automation Systems using Multiple Key Servers. In: 7th IEEE International Workshop on Factory Communication Systems, pp. 205–214 (2008)
23. Granzer, W., Lechner, D., Praus, F., Kastner, W.: Securing IP Backbones in Building Automation Networks. In: 7th IEEE International Conference on Industrial Informatics, pp. 410–415 (2009)

A UML Profile for Requirements Analysis of Dependable Software

Denis Hatebur^{1,2} and Maritta Heisel¹

¹ Universität Duisburg-Essen, Germany, Fakultät für Ingenieurwissenschaften
maritta.heisel@uni-due.de

² Institut für technische Systeme GmbH, Germany
d.hatebur@itesys.de

Abstract. At Safecom 2009, we presented a foundation for requirements analysis of dependable software. We defined a set of patterns for expressing and analyzing dependability requirements, such as confidentiality, integrity, availability, and reliability. The patterns take into account random faults as well as certain attacks and therefore support a combined safety and security engineering.

In this paper, we demonstrate how the application of our patterns can be tool supported. We present a UML profile allowing us to express the different dependability requirements using UML diagrams. Integrity conditions are expressed using OCL. We provide tool support based on the Eclipse development environment, extended with an EMF-based UML tool, e.g., Papyrus UML. We illustrate how to use the profile to model dependability requirements of a cooperative adaptive cruise control system.

1 Introduction

Dependable systems play an increasingly important role in daily life. More and more tasks are supported or performed by computer systems. These systems are required to be safe, secure, available, and reliable. For such systems, it is of utmost importance to thoroughly analyze, understand, and consolidate the requirements.

In an earlier paper [8], we have presented a foundation for requirements analysis of dependable systems, based on problem frames [12]. In this paper, we show how the approach of [8] can be tool supported. To this end, we have defined a Unified Modeling Language (UML) profile [17] that allows us to represent problem frames in UML. This UML profile is then augmented with stereotypes that support the expression of dependability requirements. The stereotypes are complemented by constraints expressed in the Object Constraint Language (OCL) [15] that can be checked by existing UML tools. These constraints express important integrity conditions, for example, that security requirements must explicitly address a potential attacker. By checking the different OCL constraints, we can substantially aid system and software engineers in analyzing dependability requirements.

We work with the following definitions of dependability attributes [8]: *Safety* is the *inability* of the system to have an undesirable effect on its environment, and *security* is the *inability* of environment to have an undesirable effect on the system. To achieve safety, systematic and random faults must be handled. For security, in contrast, certain attackers must be considered. Security can be described by confidentiality, integrity and availability requirements. Also for safety, integrity and availability must be

considered. For safety, integrity and availability mechanisms have to protect against random (and some systematic) faults. *Reliability* is a measure of continuous service accomplishment. It describes the probability of correct functionality under stipulated environmental conditions.

Dependability requirements must be described and analyzed. Problem frames [12] are a means to describe and analyze functional requirements, but they can be extended to describe also dependability requirements and domain knowledge, as also shown in earlier papers [8,10]. In Section 2, we present problem frames and the parts of the problem frames profile that extends the UML meta-model [17]. We describe the parts of our profile relevant to model dependability features. In Section 3, we show how we build tool support for the problem frame approach and for describing and analyzing dependability requirements. Section 4 contains our profile extension to describe dependability, and it also describes the OCL constraints for applying the elements introduced to describe dependability. Section 5 describes the process to work with our UML profile for problem frames for dependable systems. The case study in Section 6 applies that process to a cooperative adaptive cruise control system. Section 7 discusses related work, and the paper closes with a summary and perspectives in Section 8.

2 UML Profile for Problem Frames

Problem frames are a means to describe software development problems. They were introduced by Jackson [12], who describes them as follows: “A *problem frame* is a kind of pattern. It defines an intuitively identifiable problem class in terms of its context and the characteristics of its domains, interfaces and requirement.”

We describe problem frames using class diagrams extended by stereotypes (see Fig. 1). All elements of a problem frame diagram act as placeholders, which must be instantiated to represent concrete problems. Doing so, one obtains a problem description that belongs to a specific problem class.

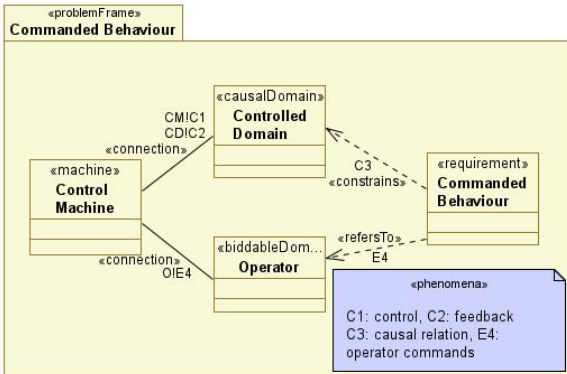


Fig. 1. Commanded Behaviour problem frame using UML notation

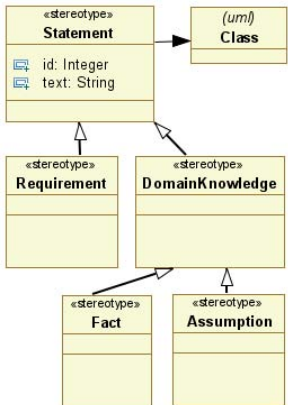


Fig. 2. Requirement stereotype inheritance structure

The class with the stereotype `<<machine>>` represents the thing to be developed (e.g., the software). The other classes with some domain stereotypes (e.g., `<<CausalDomain>>` or `<<BiddableDomain>>`) represent *problem domains* that already exist in the application environment.

In frame diagrams, *interfaces* connect domains, and they contain *shared phenomena*. Shared phenomena may be events, operation calls, messages, and the like. They are observable by at least two domains, but controlled by only one domain, as indicated by an exclamation mark. For example, in Fig. 1 the notation *O!E4* means that the phenomena in the set *E4* are controlled by the domain *Operator*. These interfaces are represented as associations, and the name of the associations contain the phenomena and the domains controlling the phenomena.

The associations can be replaced by interface classes in which the operations correspond to phenomena. The interface classes are either controlled or observed by the connected domains, represented by dependencies with the stereotypes `<<controls>>` or `<<observes>>`. Each interface can be controlled by at most one domain. A controlled interface must be observed by at least one domain, and an observed interface must be controlled by exactly one domain.

Problem frames substantially support developers in analyzing problems to be solved. They show what domains have to be considered, and what knowledge must be described and reasoned about when analyzing the problem in depth. Developers must elicit, examine, and describe the relevant properties of each domain. These descriptions form the *domain knowledge*. The domain knowledge consists of *assumptions* and *facts*. Assumptions are conditions that are needed, so that the *requirements* are accomplishable. Usually, they describe required user behavior. For example, it must be assumed that a user ensures not to be observed by a malicious user when entering a password. Facts describe fixed properties of the problem environment regardless of how the machine is built.

Domain knowledge and requirements are special statements. A statement is modeled similarly to a Systems Modeling Language (SysML) requirement [16] as a class with a stereotype. In this stereotype a unique identifier and the statement text are contained as stereotype attributes. Fig. 2 shows that the stereotype `<<Statement>>` extends the metaclass Class of the UML metamodel.

When we state a requirement, we want to change something in the world with the machine to be developed. Therefore, each requirement constrains at least one domain. This is expressed by a dependency from the requirement to a domain with the stereotype `<<constrains>>`. Such a constrained domain is the core of any problem description, because it has to be controlled according to the requirements. Hence, a constrained domain triggers the need for developing a new software (the machine), which provides the desired control.

A requirement may refer to several domains in the environment of the machine. This is expressed by a dependency from the requirement to a domain with the stereotype `<<refersTo>>`. The referred domains are also given in the requirements description.

In Fig. 1, the Controlled Domain domain is constrained, because the Control Machine has the role to change it on behalf of user commands for achieving the required Commanded Behaviour.

Jackson distinguishes the domain types *biddable domains* that are usually people, *causal domains* that comply with some physical laws, and *lexical domains* that are

data representations. The domain types are modeled by the stereotypes `<<BiddableDomain>>` and `<<CausalDomain>>` being subclasses of the stereotype `<<Domain>>`. A lexical domain (`<<LexicalDomain>>`) is modeled as a special case of a causal domain. This kind of modeling allows to add further domain types, such as `<<DisplayDomain>>` being also a special case of a causal domain. In Figure 1, the stereotypes `<<CausalDomain>>` and `<<BiddableDomain>>` indicate the domain types. To describe the problem context, a *connection domain* between two other domains may be necessary. Connection domains establish a connection between other domains by means of technical devices. They are modeled as classes with the stereotype `<<ConnectionDomain>>`. Connection domains are, e.g., video cameras, sensors, or networks. Other problem frames besides the commanded behavior frame are *required behaviour*, *simple workpieces*, *information display*, and *transformation*. [12]

Software development with problem frames proceeds as follows: first, the environment in which the machine will operate is represented by a *context diagram*. Like a frame diagram, a context diagram consists of domains and interfaces. However, a context diagram contains no requirements (see Fig. 6 for an example). Then, the problem is decomposed into subproblems. If possible, the decomposition is done in such a way that the subproblems fit to given problem frames. To fit a subproblem to a problem frame, one must instantiate its frame diagram, i.e., provide instances for its domains, phenomena, and interfaces. The instantiated frame diagram is called a *problem diagram*.

Successfully fitting a problem to a given problem frame means that the concrete problem indeed exhibits the properties that are characteristic for the problem class defined by the problem frame. A problem can only be fitted to a problem frame if the involved problem domains belong to the domain types specified in the frame diagram. For example, the Operator domain of Fig. 1 can only be instantiated by persons, but not for example by some physical equipment like an elevator.

Since the requirements refer to the *environment* in which the machine must operate, the next step consists in deriving a *specification* for the machine. The specification describes the machine and is the starting point for its construction.

The different diagram types make use of the same basic notational elements. As a result, it is necessary to explicitly state the type of diagram by appropriate stereotypes. In our case, the stereotypes are `<<ContextDiagram>>`, `<<ProblemDiagram>>`, and `<<ProblemFrame>>`. These stereotypes extend (some of them indirectly) the meta-class Package in the UML meta-model.

3 Tool Support

We have developed a tool called UML4PF to support the requirements engineering process sketched in Section 2. Our tool is integrated into the Eclipse development environment [3] as a plug-in. After the developer has drawn some diagram(s) using some EMF-based editor, for example Papyrus UML [5], UML4PF provides him or her with the following functionality:

- It checks if the developed model is valid and consistent by using our OCL constraints.
- It returns the location of invalid parts of the model.
- It automatically generates model elements, e.g., it generates observed and controlled interfaces from association names.

We defined a set of stereotypes in a profile that extends the UML meta-model. The most important stereotypes are presented in Section 2. This UML profile can be extended independently from our tool. Our tool is based on the Eclipse Modeling Framework (EMF [4]) and should be inter-operable with other EMF-based UML development tools being extendable using UML-profiles [17].

Our plugin UML4PF checks (using the Eclipse Modeling Framework) that the stereotypes are used correctly according to integrity conditions, e.g., that each statement constrains at least one domain. The tool is an open source tool under development and is free for download from <http://swe.uni-due.de/en/research/>.

4 Dependability Extension

We developed a set of patterns for expressing and analyzing dependability features (requirements and domain knowledge). Our patterns consist of UML classes with stereotypes and a set of rules describing possible relations to other model elements. The stereotype contains specific properties of the dependability feature (e.g. the probability to be achieved), a unique identifier, and a textual description that can be derived from the properties and the relations to other model elements. The patterns can be directly translated into logical predicates [8]. These predicates are helpful to analyze conflicting requirements and the interaction of different dependability requirements, as well as to find missing dependability requirements.

An important advantage of our patterns is that they allow dependability requirements to be expressed without anticipating solutions. For example, we may require data to be kept confidential during transmission without being obliged to mention encryption, which is a means to achieve confidentiality. The benefit of considering dependability requirements without reference to potential solutions is the clear separation of problems from their solutions, which leads to a better understanding of the problems and enhances the re-usability of the problem descriptions, since they are completely independent of solution technologies.

The dependability features can be described independently from the functional description. This approach limits the number of patterns, and allows one to apply these patterns to a wide range of problems. For example, the functional requirements for

```

1 Class.allInstances()->select (
2 (getAppliedStereotypes().name->includes('Confidentiality') or
3 getAppliedStereotypes().name->includes('Integrity') or
4 getAppliedStereotypes().name->includes('Availability') or
5 getAppliedStereotypes().name->includes('Reliability'))
6 and getAppliedStereotypes().name->includes('Requirement'))
7 ->forAll(clientDependency->select(d |
8   d.oclAsType(Dependency).getAppliedStereotypes().name ->
9     includes('supplements'))
   .oclAsType(Dependency).target.getAppliedStereotypes().name
   -> includes('Requirement')->count(true)>=1 )

```

Listing 1.1. Each Dependability Statement Supplements a Requirement

data transmission or automated control can be expressed using a problem diagram. Dependability requirements for confidentiality, integrity, availability and reliability can be added to that description of the functional requirement.

A dependability requirement always supplements (stereotype `<<supplements>>`) a functional requirement. This can be validated with the OCL expression in Listing 1.1. In this OCL expression, all classes with a stereotype indicating a dependability statement (e.g., `<<Integrity>>` or `<<Availability>>`) and additionally the stereotype `<<Requirement>>` are selected in Lines 1-6. In all of these requirement classes, it is checked that their dependencies (Line 7) with the stereotype `<<supplements>>` (Line 8) point to at least one class with the stereotype `<<Requirement>>` (Line 9).

Our patterns help to structure and classify dependability requirements. For example, requirements considering integrity can be easily distinguished from the availability requirements. It is also possible to trace all dependability requirements that refer to a given domain.

The patterns for integrity, reliability, and availability considering random faults are expressed using probabilities, while for the security requirements no probabilities are defined. We are aware of the fact that no security mechanism provides a 100 % protection and that an attacker can break the mechanism to gain data with a certain probability. But in contrast to the random faults considered for the other requirements, no probability distribution can be assumed, because, e.g., new technologies may dramatically increase the probability that an attacker is successful. For this reason we suggest to describe a possible attacker and ensure that this attacker is not able to be successful in a reasonable amount of time.

In the following, we present a selection of our dependability analysis patterns. More patterns and details are given in our technical report [9].

4.1 Confidentiality

A typical confidentiality statement is to

Preserve confidentiality of *StoredData* / *TransmittedData* for *Stakeholders* and prevent disclosure by a certain *Attacker*.

A statement about confidentiality is modeled as a class with the stereotype `<<Confidentiality>>` in our profile. This stereotype is a specialization of the stereotype `<<Statement>>`, described in Section 2. Three dependencies must be specified for a confidentiality requirement:

1. A causal domain representing the *StoredData* or *TransmittedData* must be constrained (using `<<constrains>>`). Even if data is usually modeled using lexical domains, we derive *StoredData* or *TransmittedData* from *CausalDomain*, because in some cases the storage device and not the data is modeled.
2. The statement needs to refer to the considered attacker. This attacker must be described in detail. We suggest to describe at least the attacker's objective, its skills, equipment, knowledge, and the time the attacker has to prepare and to perform the attack. A similar kind of description is suggested in the Common Methodology for Information Technology Security Evaluation (CEM) [1]. As shown in Fig. 3, the stereotype `<<Attacker>>` is a specialized `<<BiddableDomain>>`. The reference to an *Attacker* is necessary, because we can only ensure confidentiality with respect to an *Attacker* with given properties.

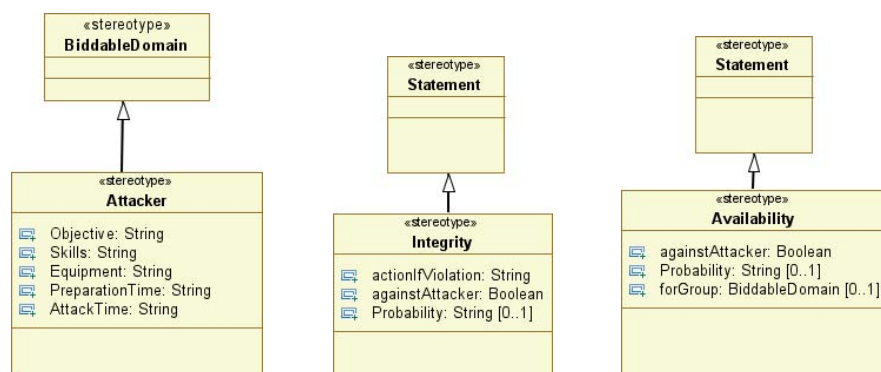


Fig. 3. Attacker in UML Problem Frames Profile **Fig. 4.** Integrity Statement in UML Problem Frames Profile **Fig. 5.** Availability Statement in UML Problem Frames Profile

3. A confidentiality statement also needs to refer to the data's stakeholder. The *Stakeholder* is referred to, because we want to allow the access only to *Stakeholders* with legitimate interest. The instances of *Stakeholder* and *Attacker* must be disjoint. The corresponding OCL expression requires that each confidentiality statement refers to at least one biddable domain that is not the attacker.

It is possible to generate the statement text from other model information: In the typical confidentiality statement the *StoredData* / *TransmittedData* can be obtained from the names of the domains constrained by this statement, the *Attacker* can be instantiated with the name of the domain referred to with the stereotype `<<Attacker>>`, and the *Stakeholder* can be instantiated with the name of the referred domain with the stereotype `<<BiddableDomain>>` (or a subtype different from `<<Attacker>>`). Additionally, the names of supplemented functional requirements can be added to the statement text if they exist. A confidentiality requirement is often used together with functional requirements for data transmission and data storage.

4.2 Integrity

Typical integrity statements considering random faults are:

With a probability of P_i , one of the following things should happen: service (as described in the functional statement) *with influence on / of the domain constrained in the functional requirement* must be correct, or a specific action must be performed.

Typical security integrity statements are:

The *influence (as described in the functional statement) on / content of domain constrained in the functional statement* must be either correct, or in case of any modifications by some *Attacker* a specific action must be performed.

In contrast to the dependability statement considering random faults, this requirement can refer to the content of a domain (instead of the functionality), because security engineering usually focuses on data. For security the domain constrained in the functional requirement is usually a display or some plain data. The specific action could be, e.g.:

- write a *log* entry
- switch off the *actuator*
- do *not* influence the *domain constrained in the functional statement*
- perform the same action as defined in the functional statement on *domain constrained in the functional statement*. In this case we talk about reliability.
- inform *User*

Integrity statements are modeled as classes with the stereotype `<<Integrity>>`. In our profile, this stereotype is a specialization of the stereotype `<<Statement>>`, as shown in Fig. 4.

The domain mentioned in the specific action must be constrained by the integrity statement. The last specific action directly refers to the *User*. The *User* is a biddable domain and cannot be directly constrained. Therefore, the *User* must be informed by some technical means that can be constrained, e.g. a display. The assumption that the *User* sees the *Display* (being necessary to derive a specification from the requirements) must be checked later for validity.

An integrity requirement needs to refer to the domain constrained by the supplemented functional requirement. The class defining the stereotype `<<integrity>>` also has attributes. The attribute `ActionIfViolation` in Fig. 4 contains the textual description of the specific action as a string. The boolean attribute `againstAttacker` shows that the statement is a security statement, or if it is set to *false* that it is a statement considering random faults. In that case also the `Probability` must be specified. For all integrity statements (Lines 1-3 of Listing 1.2), it is checked if the inverted value of the stereotype attribute `againstAttacker` (Lines 4-6) implies that the value `Probability` is set, i.e. not equal to *null* (Lines 8-10).

```

1 Class.allInstances() -> select(oe |
2   oe.oclAsType(Class).getAppliedStereotypes().name ->
3     includes('Integrity'))
4 -> forAll(c |
5   not c.oclAsType(Class).getValue(c.oclAsType(Class)
6     .getAppliedStereotypes() -> select(s |
7       s.oclAsType(Stereotype).name ->
8         includes('Integrity'))
9     -> asSequence() -> first(), 'againstAttacker')
10    .oclAsType(Boolean)
11    implies
12      c.oclAsType(Class).getValue(c.oclAsType(Class)
13        .getAppliedStereotypes() -> select(s |
14          s.oclAsType(Stereotype).name ->
15            includes('Integrity'))
16        -> asSequence() -> first(), 'Probability') <> null )

```

Listing 1.2. Integrity Statements Contain Probabilities

The probability is a constant, determined by risk analysis. The standard ISO/IEC 61508 [11] provides a range of failure rates for each defined safety integrity level (SIL). The probability P_i could be, e.g., for SIL 3 systems operating on demand $1 - 10^{-3}$ to $1 - 10^{-4}$.

If the stereotype attribute `againstAttacker` is true, it is necessary that the statement refers to an attacker. The attacker must be described in the same way as for confidentiality in Section 4.1.

4.3 Availability

A typical availability statement considering random faults is:

The service (described in the functional statement) *with influence on / of the domain constrained in the functional statement* must be available (for *Users*) with a probability of P_a .

When we talk about availability in the context of security, it is not possible to provide the service to everyone due to limited resources. The availability statement considering an attacker is expressed as follows:

The service (described in the functional statement) *with influence on / of the domain constrained in the functional statement* must be available for *Users* even in case of an attack by *Attackers*.

Availability statements are modeled as classes with the stereotype `<<Availability>>`. In our profile, this stereotype is a specialization of the stereotype `<<Statement>>`, shown in Fig. 5.

Availability requirements constrain the domains constrained by the supplemented functional requirement. The stereotype class for availability contains the attributes `againstAttacker`, `Probability`, and `forGroup`. If `againstAttacker` is *false*, the stereotype attribute `Probability` must be specified. This can be checked in the same way as for integrity, described in Section 4.2.

If `againstAttacker` is *true*, the stereotype attribute `forGroup` must be specified, and an attacker must be referred to. Both conditions can be expressed similarly as described in Listing 1.2.

4.4 Reliability

Reliability is defined in a similar way as availability (see Section 4.3). The same failure rates as for integrity (see Section 4.2) can be used.

5 Procedure to Use the Dependability Extension

This section describes how to work with the UML profile for problem frames for dependable systems. To use our profile and apply the dependability patterns, we assume that *hazards and threats are identified, and a risk analysis* has been performed. The next step is to *describe the environment*, because dependability requirements can only be guaranteed for some specific intended environment. For example, a device may be dependable for personal use, but not for military use with more powerful attackers or a non-reliable power supply. The *functional requirements are described* for this intended environment using problem frames (see Section 2). The requirements describe how the environment should behave when the machine is in action. The requirements should be expressed in terms of domains and phenomena of the context diagram. From hazards and threats an *initial set of dependability requirements can be identified*. These requirements supplement the previously described functional requirements.

For each dependability requirement, a pattern from Section 4 should be selected. After an appropriate pattern is determined, it must be connected with the concrete domains from the environment description. The connected domains must be described. For an attacker, at least the attributes of the stereotype must be defined (objective, equipment, skill, time to attack, time to prepare). Via these assumptions, *threat models* are integrated into the development process using dependability patterns. The values for probabilities can be usually extracted from the risk analysis.

Our paper [8] describes how to *find missing, interacting, and related requirements or domain knowledge* by selecting generic mechanisms. New requirements and new domain knowledge is described using the same notation as used for the initial requirements and analyzed in the same way.

6 Case Study

The approach is illustrated by the same case study as presented in [8]; the development of a cooperative adaptive cruise control (CACC) maintaining string stability. Such a system controls the speed of a car according to the desired speed given by the driver and the measured distance to the car ahead. It also considers information about speed and acceleration of the car ahead which is sent using a wireless network.

The *hazard* to be avoided is an unintended acceleration or deceleration (that may lead to a rear-end collision). The considered *threat* is an attacker who sends wrong messages to the car in order to influence its speed.¹ Examples for domain knowledge of the

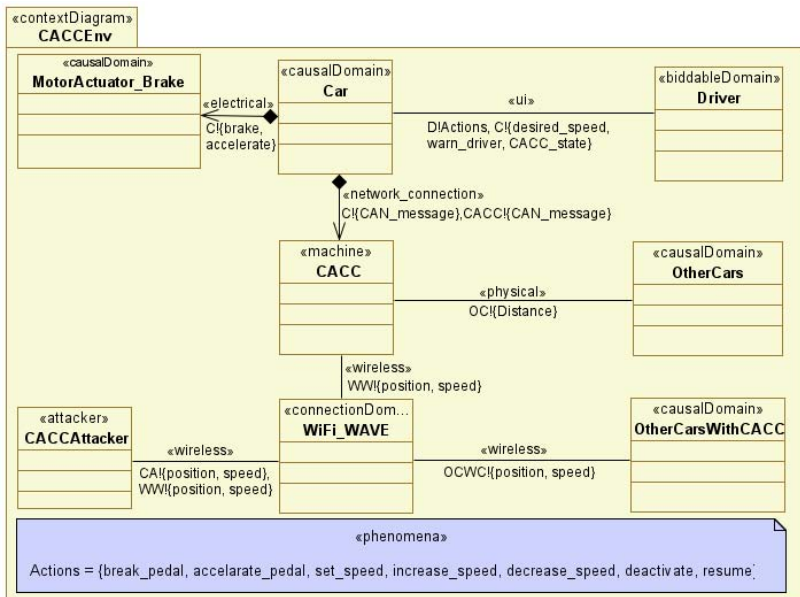


Fig. 6. CACC Context Diagram

¹ The *risk analysis* is left out here.

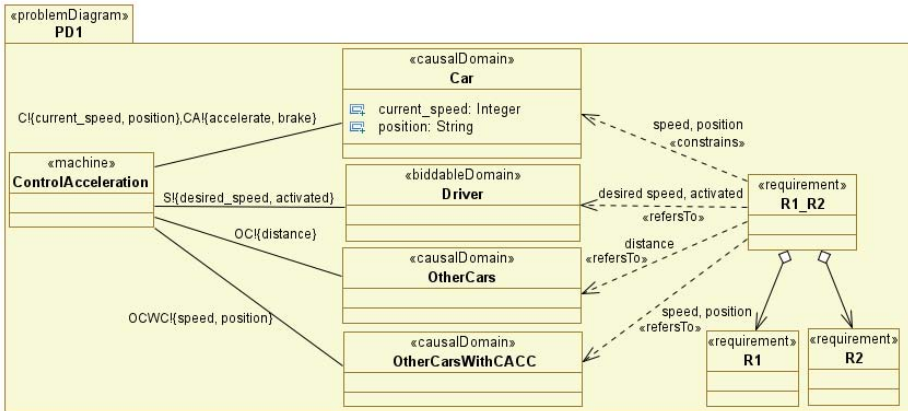


Fig. 7. CACC Problem Diagram for Control Acceleration and Brake

CACC in the *described environment* are physical properties about acceleration, braking, and measurement of the distance (relevant for safety). Other examples are the assumed intention, knowledge and equipment of an attacker. We assume here that the attacker can only access the connection domain *WiFi_WAVE interface*. The context diagram for the CACC is shown in Fig. 6. It also contains the type of connection as stereotypes at the associations between domains (e.g. <<wireless>> for wireless connections). These connection types are not considered in this paper.

The *functional requirement* for the CACC is to maintain string stability:

- R1.** The CACC should accelerate the car if the desired speed set by the driver is higher than the current speed, the CACC is activated and the measured distance and the calculated distance to the car(s) ahead are safe.
- R2.** The CACC should brake the car if the desired speed set by the driver is much (30 km/h) lower as the current speed, the CACC is activated and the measured or calculated distance to the car(s) ahead is decreasing towards the safe limit.

As an example, the problem diagram for R1 and R2 is depicted in Fig. 7. The problem diagram describes the interfaces between the machine and the environment necessary to implement requirements R1 and R2, e.g., it describes that the machine (a submachine of the CACC in the context diagram) can accelerate the car (CAI{accelerate}), and it describes the relation of the requirements R1 and R2 to the domains in the environment. The requirements constrain the current speed of the car and therefore indirectly its position. The requirements refer to the information in the domains necessary for the described decision, e.g., the desired speed and the distance to the car ahead.

The next step is to *identify an initial set of dependability requirements*. For the functional requirements R1 and R2, the following security requirement can be stated using the textual pattern from Section 4.2:

The influence (as described in R1 and R2) on the car (brake, accelerate) must be either correct, or in case of any modifications by CACCAttacker the car (MotorActuator_Brake) should not be influenced (no brake, no accelerate).

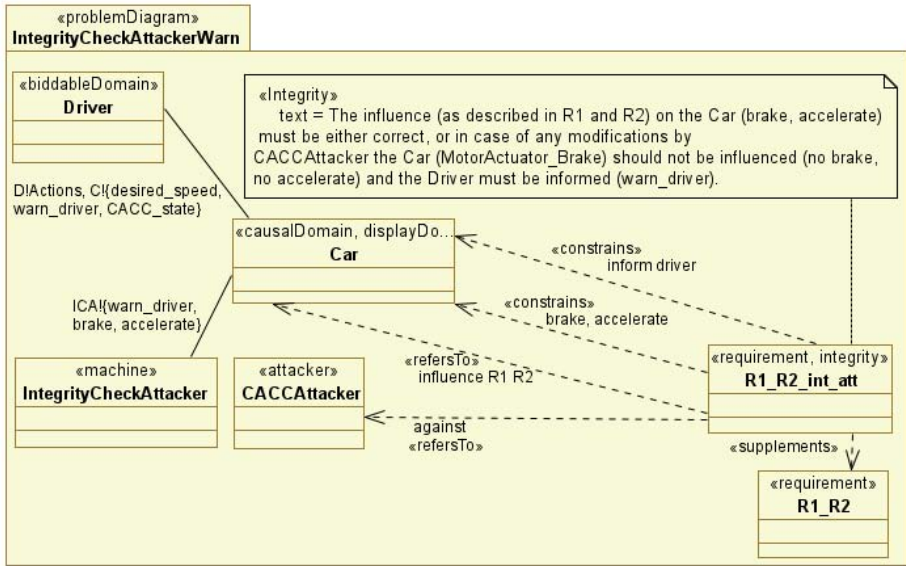


Fig. 8. CACC Problem Diagram for Integrity Checks considering an Attacker

A problem diagram including this integrity requirement is depicted in Fig. 8. It supplements the requirements R1_R2. It refers to an attacker (the CACCAttacker) and also refers to the domain constrained by R1_R2 (the Car). The Car is constrained because the MotorActuator_Brake as part of the car should not be influenced.

All OCL constraints defined for the profile were checked. With checking these constraints, we detected several minor mistakes (e.g., wrong names), and we detected that the original version of our problem diagram did not refer to the domain constrained in the requirement.

We also defined the problem diagrams and the predicates for the other initial dependability requirements (integrity considering random faults, availability, and reliability). Details can be found in our technical report [9].

To find missing, interacting, and related requirements or domain knowledge, we used the table with dependability predicates presented in [8]. This analysis resulted in a set of additional requirements and additional domain knowledge. For example, to preserve integrity considering the described attacker, we need a protection of the messages sent using a wireless interface. To protect the messages, we chose Message Authentication Codes (MAC). For creating and validating MACs, session secrets are necessary. These secrets and the processing data in the machine itself must be kept confidential. The pattern also requires to refer to the stakeholder (here: the Manufacturer) and the attacker. The statement about protection of the secrets should not be realized by the software to be built and is therefore considered to be domain knowledge.

The complete case study consists of 40 classes, 50 associations, and about 150 dependencies. A total of 65 OCL constraints were checked using our tool, 15 of which concerned dependability. The OCL constraints showed for previous versions of the CACC case study, e.g., that our integrity requirement did not refer to an attacker. As a final result, the CACC model has been successfully validated.

7 Related Work

This paper extends the patterns for requirements (and domain knowledge) presented in [8] by a formal metamodel to provide tool support.

The Common Criteria [2], Part 2 define a large set of so-called *Security Functional Requirements (SFRs)* as patterns for requirements. But some of these SFRs directly anticipate a solution, e.g. the SFR *cryptographic operation* in the class *functional requirements for cryptographic support* (FCS_COP) specifies the cryptographic algorithm, key sizes, and the assigned standard to be used. The SFRs in the Common Criteria are limited to security issues. In a technical report [9], we relate some of the CC SFRs to our patterns for dependability statements.

Lencastre et al. [13] define a metamodel for problem frames using UML. In contrast to our metamodel, it only consists of a UML class model. Hence, the OCL integrity conditions of our metamodel are not considered in their metamodel.

Hall et al. [7] provide a formal semantics for the problem frame approach. Their model focuses on a formal specification language to describe problem frames and problem diagrams.

Seater et al. [14] present a metamodel for problem frame instances. They formalize requirements and specifications. Their integrity conditions focus on correctly deriving specifications from requirements.

Charfi et al. [6] use a modeling framework called *Gaspard2* to design high-performance embedded systems-on-chip. They use model transformations to move from one level of abstraction to the next. To validate that their transformations have been correctly performed, they use the OCL language to specify the properties that must be checked in order to be considered as correct with respect to *Gaspard2*. We have been inspired by this approach. However, we do not focus on high-performance embedded systems-on-chip. Instead, we target dependable systems development.

SysML [16] also provides the stereotype `<<Requirement>>` for classes. It can be used to express dependabilities between requirements and the relation to realization and tests (e.g., with the stereotypes `<<refine>>`, `<<trace>>`, `<<satisfy>>`). We relate the requirements to domains of the environment to make their purpose explicit and provide support for requirements interaction analysis.

8 Conclusions and Future Work

In this paper, we have presented an extension to our UML profile for problem frames to describe dependability. In this profile, we defined a set of stereotypes for dependability requirements and domain knowledge. We set up 65 OCL constraints for requirements engineering, 15 of which concern dependability. These constraints show how functional requirements can be supplemented by dependability requirements.

In summary, our concept has the following advantages:

- Artifacts from the analysis development phase that are part of a model created with our profile can be re-used in later phases in the software development process.
- The notation is based on UML. UML is commonly used in software engineering, and many developers are able to read our models.
- The concept is not tool-specific. It can be easily adapted to other UML2 tools that allow to specify new stereotypes.

- The dependability statements are re-usable for different projects.
- A manageable number of statement types can be used for a wide range of problems, because they are separated from the functional requirements.
- Statements expressed using our profile refer to the environment description and are independent from solutions. Hence, they can be easily re-used for new product versions.
- A generic textual description of the requirement or the domain knowledge can be generated from other model elements.
- Statements expressed using our profile help to structure and classify the dependability requirements. For example, integrity statements can be easily distinguished from availability statements. It is also possible to trace all dependability statements that refer to one domain.

In the future, we plan to extend our tool to support the identification of missing and interacting requirements. We also want to support traceability links to trace our (dependability) requirements to artifacts developed later, e.g. components in the software architecture.

Acknowledgment. We thank Isabelle Côté for her valuable comments on this work.

References

1. Common Methodology for Information Technology Security Evaluation (August 2005), <http://www.commoncriteriaportal.org/public/expert/>
2. Common Criteria for Information Technology Security Evaluation, Version 3.1 (September 2006), <http://www.commoncriteriaportal.org/public/expert/>
3. Eclipse - An Open Development Platform (May 2008), <http://www.eclipse.org/>
4. Eclipse Modeling Framework Project (EMF) (May 2008), <http://www.eclipse.org/modeling/emf/>
5. Papyrus UML Modelling Tool (January 2010), <http://www.papyrusuml.org/>
6. Charfi, A., Gamatié, A., Honoré, A., Dekeyser, J.-L., Abid, M.: Validation de modèles dans un cadre d'IDM dédié à la conception de systèmes sur puce. In: 4èmes Journées sur l'Ingénierie Dirigée par les Modèles, IDM 2008 (2008)
7. Hall, J.G., Rapanotti, L., Jackson, M.: Problem frame semantics for software development. *Software and System Modeling* 4(2), 189–198 (2005)
8. Hatebur, D., Heisel, M.: A foundation for requirements analysis of dependable software. In: Buth, B., Rabe, G., Seyfarth, T. (eds.) *SAFECOMP 2009*. LNCS, vol. 5775, pp. 311–325. Springer, Heidelberg (2009)
9. Hatebur, D., Heisel, M.: A UML profile for requirements analysis of dependable software (technical report). Technical report, Universität Duisburg-Essen (2010), <http://swe.uni-due.de/techrep/depprofile.pdf>
10. Hatebur, D., Heisel, M., Schmidt, H.: A pattern system for security requirements engineering. In: Werner, B. (ed.) *IEEE Transactions Proceedings of the International Conference on Availability, Reliability and Security (AREs)*, pp. 356–365. IEEE, Los Alamitos (2007)
11. International Electrotechnical Commission IEC. Functional safety of electrical/electronic/programmable electronic safety-relevant systems (2000)
12. Jackson, M.: *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley, Reading (2001)

13. Lencastre, M., Botelho, J., Clericuzzi, P., Araújo, J.: A meta-model for the problem frames approach. In: WiSME 2005: 4th Workshop in Software Modeling Engineering (2005)
14. Seater, R., Jackson, D., Gheyi, R.: Requirement progression in problem frames: deriving specifications from requirements. *Requirements Engineering* 12(2), 77–102 (2007)
15. UML Revision Task Force. OMG Object Constraint Language: Reference (May 2006), <http://www.omg.org/docs/formal/06-05-01.pdf>
16. UML Revision Task Force. OMG Systems Modeling Language (OMG SysML) (November 2008), <http://www.omg.org/spec/SysML/1.1/>
17. UML Revision Task Force. OMG Unified Modeling Language: Superstructure (February 2009), <http://www.omg.org/docs/formal/09-02-02.pdf>

Model-Based Safety Engineering of Interdependent Functions in Automotive Vehicles Using EAST-ADL2

Anders Sandberg¹, DeJiu Chen², Henrik Lönn³, Rolf Johansson⁴, Lei Feng³, Martin Törnngren², Sandra Torchiaro⁵, Ramin Tavakoli-Kolagari³, and Andreas Abele⁶

¹ Mecel AB, Mölndalsvägen 36-38, SE-400 20 Gothenburg, Sweden
anders.sandberg@mecel.se

² Royal Institute of Technology, 10044 Stockholm, Sweden
{chen,martin}@md.kth.se

³ Volvo Technology Corp., 40508 Gothenburg, Sweden
{henrik.lonn, lei.feng, ramin.tavakoli-kolagari}@volvo.com

⁴ Mentor Graphics Corp., 41755 Gothenburg, Sweden
rolf_johansson@mentor.com

⁵ Centro Ricerche Fiat, Strada Torino, 50-10043 Orbassano (TO), Italia
sandra.torchiaro@crf.it

⁶ Continental Automotive GmbH, 93055 Regensburg, Germany
andreas_abele@continental-corporation.com

Abstract. For systems where functions are distributed but share support for computation, communication, environment sensing and actuation, it is essential to understand how such functions can affect each other. Preliminary Hazard Analysis (PHA) is the task through which safety requirements are established. This is usually a document-based process where each system function is analyzed alone, making it difficult to reason about the commonalities of related functional concepts and the distribution of safety mechanisms across a system-of-systems. This paper presents a model-based approach to PHA with the EAST-ADL2 language and in accordance with the ISO/DIS 26262 standard. The language explicitly supports the definition and handling of requirements, functions and technical solutions, and their various relations and constraints as a coherent whole with multiple views. We show in particular the engineering needs for a systematic approach to PHA and the related language features for precise modeling of requirements, user functionalities, system operation contexts, and the derived safety mechanisms.

Keywords: PHA, ISO/DIS 26262, EAST-ADL2, function definition, functional safety, hazard analysis, dependent functions, model-based design, meta-modeling.

1 Introduction

For safety critical functions it is essential to perform a Preliminary Hazard Analysis (PHA) in order to reveal the possible effects and consequences of failures and thereby plan necessary safety engineering tasks. Since safety is a system property it is

inherently difficult to assess whether a component, when analyzed on its own, has any consequence on safety. This is a significant issue when analyzing the individual components implementing system functionalities. For example, a display device may look harmless, but if a failure in it causes a warning message to be missed or wrongly presented it may lead to a hazardous situation. The outcomes of PHA also provide an important basis for the construction and allocation of safety requirements as well as the reasoning of necessary safety mechanisms. In automotive embedded systems, safety mechanisms are typically incorporated into the target systems. The success of this strategy depends on the ability to reveal the dependencies of various safety mechanisms in user functions and to manage their integration.

The success of PHA fundamentally relies on the existence of a system definition specifying the functions of concern and in particular their operation situations with respect to the design intent. To this end, not only must the system boundaries in terms of the measured and controlled parameters of functions be defined [14], but also the behavioral agreements between the functions and their environments have to be clearly specified. From an engineering point of view, such a system definition also implies that not only are functional requirements known, but also the contexts in terms of environment states and operation situations in which these requirements are valid have to be given. There are many techniques for the identification of hazards and their causes. Techniques like FFA [3], HAZOP [4], FMEA [5], FTA [6], as well as formal safety analysis techniques [7], normally require information about the design of system functions or implementation. They do not replace the need of examining the target systems from a functional point of view as covered by PHA.

Model-based engineering (MBE) aims to promote the system development efficiency and effectiveness for advanced computer systems [8]. It provides a means of modeling and organizing the information gathered in different lifecycle stages and for different system aspects. One approach to MBE is the EAST-ADL2 – an architecture description language for automotive Electronic/Electrical (E/E) systems [9]. The language is tailored to support the ISO/DIS 26262 [1] standard for Functional Safety of automotive vehicles. By formalizing information modeling in an automotive development lifecycle, EAST-ADL2 allows requirements, functional and technical solutions, as well as their relations and constraints to be captured in a common language. This makes the language a solid modeling basis for PHA. Since PHA is oriented around system functions, a clear traceability from safety concerns to the related functional requirements and functional concepts has to be explicitly supported. Compared to a traditional document-based approach, a model-based approach makes it possible to enforce the consistency and completeness of functional assumptions and safety concerns shared in different documents and to automate the tasks of analysis, information exchange, and view generation. For example, when desired, all safety related outcomes from all PHA can be presented as a dedicated safety view.

This paper introduces the recent advances of EAST-ADL2 in supporting requirement specification and function definition in regards to PHA and some of its related tasks. It complements our other related work on safety engineering introduced in [2 and 15]. The paper is organized as follows: We first introduce the key concepts adopted in the language meta-model for a multi-leveled architecture modeling and in particular for system function definition and behavior modeling. We describe thereafter in detail through an industrial case study the usage of EAST-ADL2 for

requirements specification and function modeling to achieve effective PHA. In this paper, the term *function* refers to an end-to-end vehicle functionality for a driver/user. Other key terms used in this paper include [1]: *ASIL* (Automotive Safety Integrity Level) – an indicator for specifying the amount of development and process measures to avoid a risk; *Malfunction* – inability to meet an intended function due to system failure or abnormal behavior; *Functional safety* – absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems; *Hazard* – potential source of harm, which can be the consequence of a malfunction; *Hazardous event* – the occurrence of a hazard in an operation situation; *Safety mechanism* – functional or technical additions to a feature or a system function to make it safe; *Safety architecture* – a combination of functional architecture and safety mechanisms that fulfills the safety requirements including redundancy and independency concepts.

2 EAST-ADL2’s Support for Safety Engineering

EAST-ADL2 aims to formalize and consolidate the information that are of particular concern for a number of key work tasks in the development of automotive E/E systems [9]. In this section, we focus on the language support relevant to PHA.

2.1 An Overview of EAST-ADL2

The support of EAST-ADL2 ranges from a multi-leveled system definition to the specification of requirements and related verification and validation efforts. See Fig. 1.

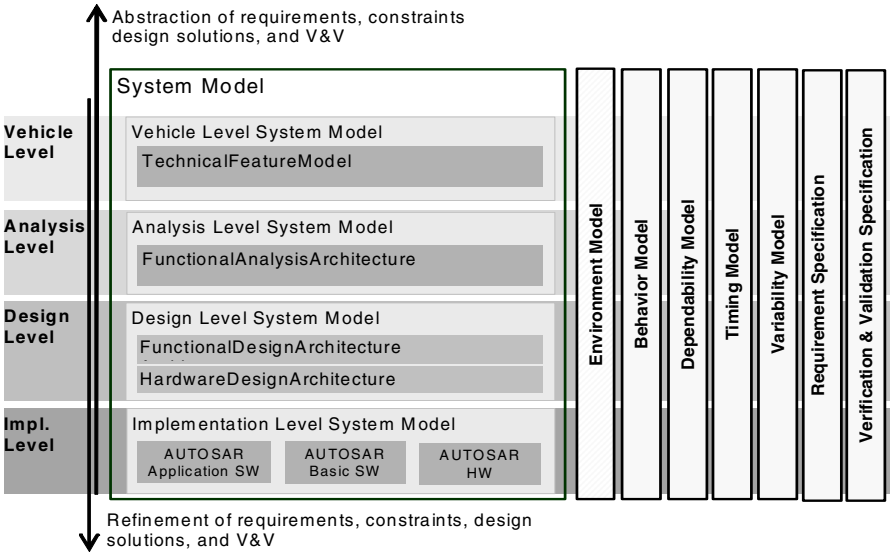


Fig. 1. An overview of system aspects and levels-of abstraction supported by EAST-ADL2

Table 1. An overview of safety engineering tasks supported by EAST-ADL2

Abstraction	Work Task	Purpose
Vehicle_Level	Function Requirement Elicitation (FRE)	Gathering descriptions of <i>functions</i> , their operation needs and other requirements from stakeholders.
	Technical Feature Tree Construction	Creating Feature Tree Model for structuring the <i>function</i> description artefacts.
	Requirement Allocation to Features	Allocating functional requirements to the nodes in the Feature Tree Model.
	Preliminary Hazard Analysis (PHA)	Performing Safety Analysis based on <i>functions</i> and their associated requirements.
	Safety Goal Construction	Deriving the safety goals and possible safe states for each hazard found in the PHA.
Analysis_Level	Initial Functional Analysis Architecture Design	Creating an initial Functional Analysis Architecture for capturing the nominal system design.
	Requirement Derivation and Traceability Support	Deriving requirements at the analysis level from requirements at the vehicle level and managing the requirement traceability.
	Requirement Allocation to Analysis Functions	Allocating the derived analysis level requirements to the functional analysis architecture.
	Functional Analysis Architecture Design	Analyzing the combined Logical architecture and changing the structure.
	Safety Architecture Design	Adding Safety Mechanisms to the Functional Analysis Architecture.
	Functional Safety Requirement Construction and Allocation	Deriving Safety Requirement from the PHA and allocating them to the functional analysis architecture.
	Function Requirement Validation	Analyzing the outcome of Functional Analysis Design and validating the satisfactions in regards to the <i>function</i> descriptions in FRE.

EAST-ADL2 facilitates safety engineering by offering direct language support for the specification of safety requirements and their allocations, the definition of system functions together with their behaviors and environmental assumptions, and the description of hazardous events, system malfunctions, and fault models [2]. Table 1 provides an overview of the language support directly related to PHA.

The levels of abstraction in EAST-ADL2 correspond to parts of automotive E/E system development lifecycle. Each level represents a specific view of the complete E/E architecture by capturing the properties emergent at that level. The highest abstraction level for a system definition is the *Vehicle_Level*, containing implementation independent information regarding the vehicle and its system functions. Key concepts at this abstraction level include requirements, vehicle features, mode (see section 2.2 and 2.3). The second highest abstraction level is the *Analysis_Level*, where the functional analysis architecture (*FunctionalAnalysis-Architecture*) is specified. This system definition details the vehicle level system model by specifying the required logical functions and how these logical functions are structured to realize the expected user functions. It provides a means of communicating and managing the logical requirements and design solutions such as the control logics, overall function timing, as well as the data sets. System definitions at lower abstraction levels capture various implementation specific concerns towards the final system realization. The system definition at *Design_Level* refines the functional analysis architecture by considering the deployment of software and hardware resources as well as necessary partitioning and allocation of logical functions. The system definition at *Implementation_Level* specifies the actual software and hardware configuration according to AUTOSAR [10].

2.2 Requirements and Technical Feature Modeling

In Fig. 2, an overview of the key constructs, on vehicle level, in the EAST-ADL2 meta-model for *function* and requirement definition is given. By having associations from requirements to behavior constraints, the language allows explicit treatments of behavior semantics behind different requirements and functions.

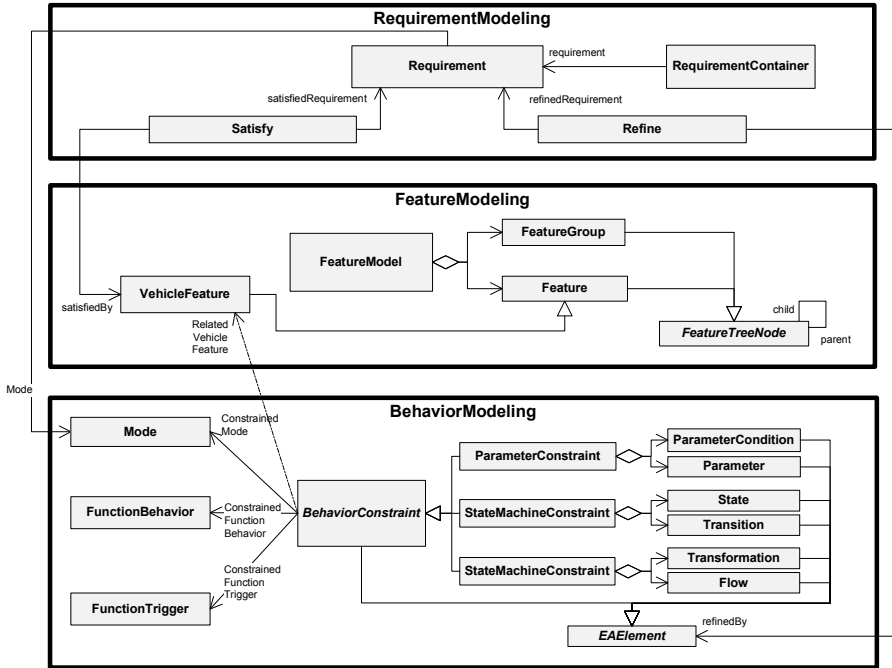


Fig. 2. Key constructs for *function* and requirement modeling in the EAST-ADL2 meta-model

A requirement expresses a condition or capability that must be met or possessed by a system in a certain *mode* to satisfy a contract or other formally imposed properties. Requirements can be introduced in different development phases for different reasons. With EAST-ADL2, a requirement can have *Satisfy* associations to the artifacts providing the implementation, *DeriveReq* associations to the derived requirements, and *Verify* associations to the specifications of verification and validation efforts. Specified initially in the requirement model with textual descriptions, a requirement can be enhanced with behavior constraints providing more precise definitions about the implied parameters, state machine and computation behaviors. This is supported in EAST-ADL2 through the requirement *Refine* relationship. For a functional requirement, a behavior constraint can be used for the following purposes:

- Operation conditions – statement of the environmental conditions, such as the monitored and controlled environmental parameters and parameter conditions;

- Operation behaviors – statement of the required internal parameters, parameter conditions, operational states and transitions, transformations and flows.

A *feature* denotes one functional or nonfunctional characteristic of a vehicle from the point of view of end-users or product-line. With EAST-ADL2, system functions are first captured and managed with feature models (*FeatureModel*). See Table 2. A feature model can be used either in variability modeling for capturing the product-line variability at any abstraction level of the target E/E system, or in system modeling for vehicle level recording of the end-to-end functionalities of the target E/E system. Features in the latter case are referred to as *VehicleFeature*.

Table 2. Key constructs for *function* modeling.

Attribute	Definition
<i>FeatureTreeNode</i>	an abstract construct for forming the hierarchical inheritance structure of features. The bi-directional parent-child-association is inherited by both <i>FeatureGroup</i> and <i>Feature</i> .
<i>Feature</i>	a functional characteristic, constraint, or property that can be present in a vehicle product line
<i>VehicleFeature</i>	statement of functional characteristic of an E/E system from an end-user's viewpoint.
<i>FeatureGroup</i>	statement of the grouping of multiple <i>Features</i> for variability management
<i>FeatureModel</i>	a place holder for feature tree consisting of <i>FeatureGroup</i> and <i>Feature</i> . In a feature model, a feature may have other features as well as the feature groups as children and each feature group may have other feature groups as well as features as children.

A technical feature model constitutes the basis for PHA by capturing the agreed system end-to-end functionalities and their dependencies such as shared requirements, operational modes, and environment assumptions. Other information referenced by the vehicle level's technical features includes the vehicle level requirements, derived hazards and safety goals, and possibly behavior constraints such as the operational modes of *functions*, and the environmental conditions where the *function* requirements are valid. The *FeatureModel* of EAST-ADL2 supports the logical configuration of *features* with the following constructs.

A *feature* satisfies one or multiple requirements and thereby gets the behavior constraints refining the satisfied requirements as its own behavior constraints, such as in regards to the expected states and environmental conditions. Similarly, by its satisfied requirements, a *function* gets the definition of its operational modes. For example, the alarm function of a car is only active when the vehicle is parked and locked. The related semantics rules defining the feature functionality is: 1. *a child VehicleFeature inherits all its ancestors(all parents)*; 2. *the inheritance is only applied to Requirements (Satisfy links)*. Any parent defined context in inherited requirements is still valid in the child feature. In EAST-ADL2 terminology rule 1 is the 'Selection' semantics used in the Vehicle level configuration mechanism of a technical feature tree. In logical terms a parent has the intersection of the requirements associated with itself and its children.

The linking between design solutions at different levels is based on the *Realization* and *DeriveReq* concept in EAST-ADL2. The realization links of a parent forms the union of the links associated with its children. Functionality definition is based on which requirements are valid for a specific feature. As functionality on vehicle level

is independent on the analysis level it does not matter that there are more realization links to the analysis level than what are needed for the functionality of the individual feature.

To apply vehicle level requirements to analysis level we will use *DeriveReq* links between requirements and apply the appropriate context in the new requirement formulation.

2.3 Nominal Behavior Modeling

The behavior within an E/E system can be considered as a combination of multiple end-to-end computation and execution chains across its components under the constraints of certain environmental conditions and behaviors. Each component is associated with some segments of the total computation and execution chains as it carries out data transformations, interacts and synchronizes with its own environment. The support of EAST-ADL2 for system behavior definition includes *Modes*, *Function Behavior*, *Function Trigger*, and *Behavior Constraint*. See Fig. 3 and Table 3 for an overview.

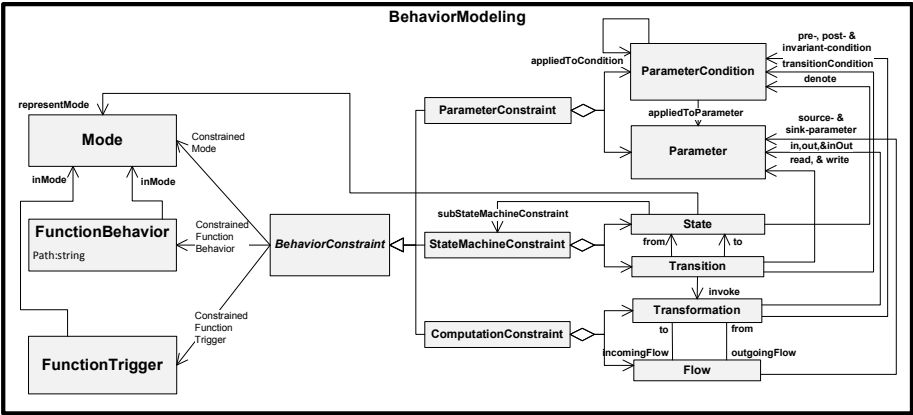


Fig. 3. Key behavior modeling constructs and their constraints in the behavior modeling package of EAST-ADL2 meta-model

Mode. The approach of EAST-ADL2 for supporting mode behavior is to associate each computation and execution behavior with declared vehicle modes. A system mode definition can be further detailed by behavior constraints capturing the related parameter and state-machines constraints. In a state-machine constraint description, each state can represent one or multiple system modes.

Function Behavior. Function behaviors specify the transfer functions of a system artifact that perform some data computations or the operations of a system artifact that can be externally invoked. Such a system artifact can be a system function at the *Analysis_Level* or *Design_Level*, or a plant or traffic function in the system environment. Each function behavior contains a reference path to the external model (e.g., based on Simulink/Matlab) defining the behavior.

Funtion Trigger. A Function trigger is used to declare the triggering policy on how the execution of a function behavior is dispatched, either in a time-triggered (periodically) pattern or on an event-triggered (sporadically or aperiodically according to the received data) basis.

Behavior Constraint. Behavior constraints in EAST-ADL2 are introduced to allow the declarations of key behavior properties and thereby to support the formalization of textual requirements as well as the reasoning of behaviors related to vehicle features and environmental situations in early development stages. EAST-ADL2 behavior constraints are given in three categories: 1. *Parameter Constraints*; 2. *State Machine Constraints*; and 3. *Computation Constraints*. See also Table 3.

The parameter constraints (ParameterConstraint) specify the quantities to be owned or processed by a mode, a function behavior, or a function trigger, as well as the conditions under which each of these quantities is used. A parameter can indicate an environmental variable such as the monitored or controlled variable of an E/E system, or a piece of application or event data being processed within the E/E system. For a function behavior, parameter conditions (ParameterConditions) can be used to specify its pre-, post-, and invariant-conditions, the computational states, as well as the relations of parameters (e.g., input and output mapping, event to output mapping). The state machine constraints (StateMachineConstraint) specify the states and state transitions of a mode, a function behavior, or a function trigger. The computation constraints (ComputationConstraint) define the logical transformations of data in a function behavior.

Table 3. Definitions of behavior modeling constructs in EAST-ADL2

Attribute	Definition
ParameterConstraint	placeholder for statements of the parameters of a behavior and their conditions
Parameter	statements of in-, out-, or local variables to be processed or the attributes owned by a behavior.
ParameterCondition	statements of the conditions of individual parameters in nominal and erroneous operating situations.
StateMachineConstraint	a placeholder for statements of states and transitions
State	statements of the states representing the value conditions of one or multiple parameters of a behavior. Within each state, there can be subordinate state machines.
Transition	statements of state transitions in response to certain event data given as the read parameters and when certain guard conditions are met. During a state transition, a transformation can be invoked.
ComputationConstaint	a placeholder for statements of the computational transformations and flows.
Transformation	statements of the data processing activities mapping two sets of parameters. A computation transformation has input and output parameters and is further characterized by some pre-, post-, and invariant conditions.
Flow	statements of the control-flow of transformations and data flows of transformation parameters.

3 Modeling an Industrial Case with EAST-ADL2

To investigate the engineering needs and evaluate related language support, an industrial case study based on a set of functions relating to vehicle longitudinal control has been performed. These functions are: *Manual Driving*, *Cruise Control(CC)*, and *Adaptive Cruise control(ACC)*, shown in Fig. 4. These functions

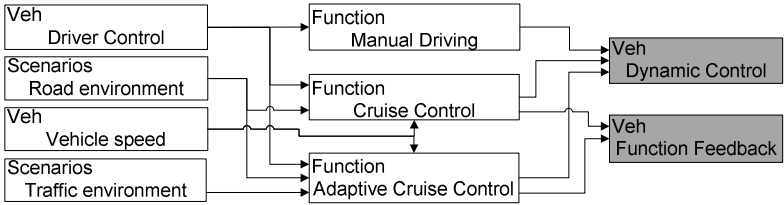


Fig. 4. Initial functional concept for the three functions in the use-case

Table 4. Requirements and *function* definitions for the functions

Funtion	Definition
Manual Driving	• Purpose: Manual control of vehicle speed • Modes referenced by parents: Power mode,Road type; Road slope; Target properties. • Defined modes: None • Defined operational states: None • Functional requirement: In all RoadSlope: Provide vehicle torque proportional to the accelerator pedal position.
Cruise Control	• PurposePlaceholder for requirements and modes common to derived Cruise control related Features. • Modes referenced by parents: Power mode, Road type; Road slope; Target properties. • Defined modes: CC_mode • Defined operational states: None • Functional requierements 1.In RoadSlope: 'FlatRoad', 'Slight uphill': In On:SteadyState state ; Keep the vehicle speed within 95% and 100% of a driver selected set-speed. 2. Driver requested torque shall have priority over CC requested torque. (Override) 2.In On state: The function shall transition to On:Standby when the driver Cancels the function. 3.If in On:Standby state: Cruise can be activated using the current speed as set-speed using the 'Set' button. 4.If in On state: A lamp shall indicate that cruise control is available. ...
Adaptive Cruise Control	• Purpose: Maintain a safe headway to traffic ahead if traveling below user defined set speed or if no traffic is present maintain vehicle speed close to a user defined set speed. • Defined modes: ACC_mode • Defined operational states: None • Modes referenced by parents: Power mode; CC_mode, Road type; Road slope; Target properties. • Inherited requirements: All 'Cruise control' requirements. • Functional requirements: 1. In RoadSlope: 'Slight downhill': In CC_mode:On:SteadyState state ; Keep the <i>vehicle speed</i> within 95% and 100% of a driver selected <i>set-speed</i>. 2. In RoadSlope: Slight downhill, Flat road, Slight uphill: In ACC_mode:Follow; Maintain a steady state distance within 10% of 2s headway*current vehicle speed to the vehicle in the same lane in front. ...
Basic Cruise Control	• Purpose: Maintain vehicle speed close to a user defined set speed. • Modes referenced by parents : Power mode; CC_mode, Road type; Road slope; Target properties. • Defined modes: None • Defined operational states: None • Inherited requirements:All 'Cruise control' requirements. - Functional requirements:None

have many common requirements and share the control of vehicle forward torque, indicating global restrictions on their co-existence and behaviors in a vehicle. The functions under safety analysis, ISO 26262 - Items, are the 'Function' blocks in Fig 4.

3.1 Requirement Elicitation (FRE) and Technical Feature Tree Construction

Table 4 provides a simplified view of the *function* definitions for the longitudinal control functions and their functional requirements.

With EAST-ADL2, the functional requirements are specified by referencing operation states and environment parameters as their contexts, such as road slope, target movement properties. A vehicle level feature is defined by referencing related functional requirements, operational mode, and operational situation. One snapshot of the EAST-ADL2 support for defining the requirements and vehicle features shown in Fig. 5 and Fig. 6.

```
<<VehicleFeature>> CruiseControl
{
}

<<ModeGroup>> CC_Mode
{
  Mode[1] = OFF; Mode[2] = STANDBY; Mode[3] = STEADY_STATE;
  Mode[4] = OVERRIDE; Mode[5] = REGULATING; Mode[6] = FAIL }
<<ModeGroup>> PowerMode
{
  Mode[1] = OFF; Mode[2] = IGNITION; Mode[3] = CRANK; Mode[4] = ENGINE_RUNNING }
<<ModeGroup>> RoadSlope
{
  Mode[1] = STEEP Uphill; Mode[2] = Uphill; Mode[3] = Flat;
  Mode[4] = Downhill; Mode[5] = SteepDownhill }

<<SatisfyRequirement>>
{
  from <<VehicleFeature>> -> CruiseControl
  to <<Requirement>> -> CC_CtrlRqs1 }

<<Requirement>> CC_CtrlRqs1
{
  Mode[1] -> PowerMode:ENGINE_RUNNING
  Mode[2] -> RoadSlope:FLAT
  Mode[3] -> CC_Mode:STEADY_STATE }
```

Fig. 5. Declarations of features, requirements, and their mapping

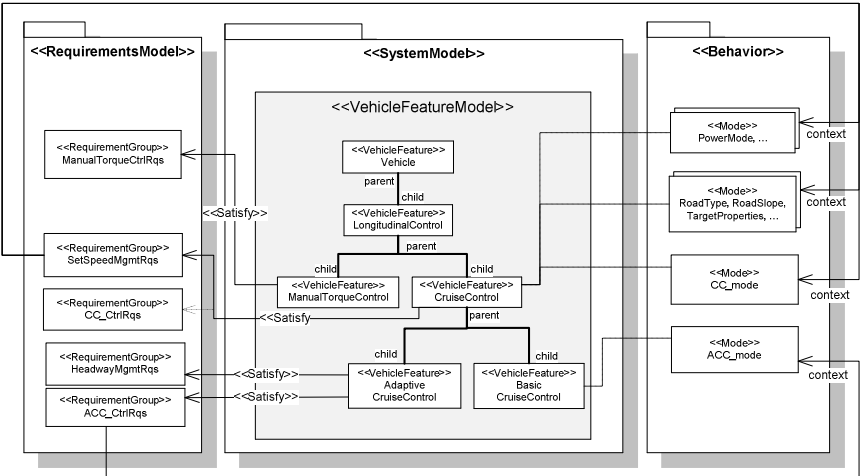


Fig. 6. Technical Feature Tree model of the functions

3.2 PHA and Safety Goal Construction

Hazards are the possible events on vehicle level that can arise from the failure modes in defined drive scenarios. Since the analysis is performed on function level, we only

Table 5. Related hazards and safety goals of some longitudinal control features

Feature	Hazard	Safety Goal	ASIL
Manual Driving, CruiseControl, ACC	Unintended torque request increase.	SG1: Torque request shall always be proportional to pedal position.	C
Manual Driving, CruiseControl, ACC	Unintended torque request decrease.	SG1: Torque request shall always be proportional to pedal position.	A
CruiseControl, ACC	Unintended acceleration above set speed.	SG2: Vehicle shall not accelerate above set speed.	B
CruiseControl, ACC	Unintended acceleration caused by unintended set speed increase.	SG3: Set speed shall not increase without driver intent.	B
CruiseControl, ACC	Unintended acceleration when CC is off.	SG4: No CC torque request shall be made in Off mode.	C
CruiseControl, ACC	Unintended acceleration when CC is cancelled.	SG5: No CC torque request shall be made in Cancel mode.	C

consider the operational behaviors as indicated by the requirements and feature definitions. Table 5 presents the hazards identified based on the given features and requirements. By classifying the consequence of each hazardous event with respect to its severity, exposure, and controllability, the related safety goals and ASIL assignments are derived.

3.3 Initial Functional Analysis Architecture Design

For each vehicle feature, an initial functional analysis architecture design is constructed. The initial architecture for the manual driving function is illustrated in Fig. 7. The initial architecture contains the necessary elements to perform the user function and its association to the environment.

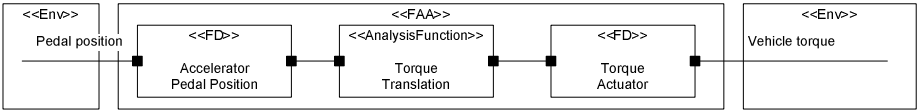


Fig. 7. Initial architecture for the manual driving function

The interdependencies of these functions can be detected based the definitions of related operation behaviors and environment parameters, and thereafter handled when merging the functional analysis architectures of individual functions.

3.4 Safety Architecture Design, Functional Safety Requirement Construction and Allocation

The initial functional analysis architecture merging the entire feature set is shown in **Fig. 8**. This initial architecture is then augmented with additional capabilities, such as redundancy, safety mechanisms, and other logical elements, to form a safety architecture satisfying the given safety goals A structured way to find elements that need to be supervised is to perform a *FTA* (Fault Tree Analysis) on the nominal

architecture using the safety goals as the root event. For the vehicle longitudinal control, the safety architecture consists of the nominal architecture with an additional torque supervision and torque disabler function shown as the darker elements in Fig. 8. In the figure, safety related data paths are shown with dashed lines

It is important to remember that safety mechanisms specified in ISO 26262 only cover the electrical system. As the system interface functions in terms of *functional devices* (FD) only maps data from the electrical and physical world, any logic associated with the supervision can only be performed by the application functions in terms of *AnalysisFunction* (AF).

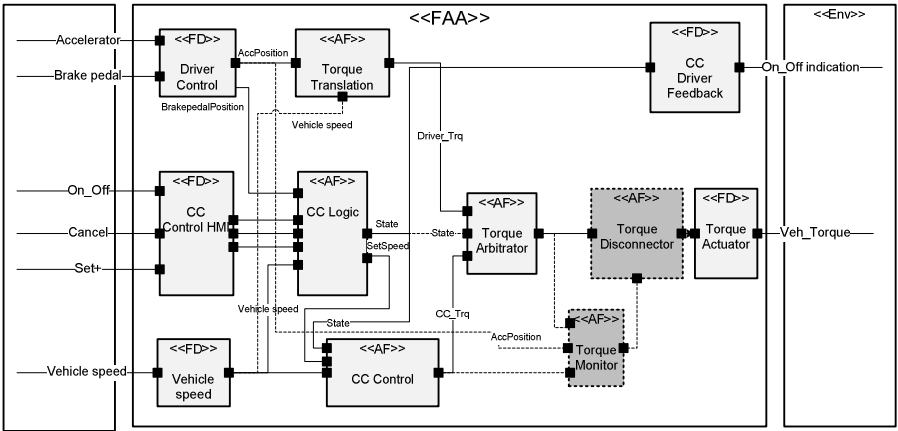


Fig. 8. Safety architecture for MTC and CC features

Table 6 provides an overview of the derived functional safety requirements and their allocation for the common elements for MTC, CC and ACC functions..Making a complete FSR table is out of scope as it is both time and mainly space consuming, tha table focuses on elements having an effect on the Veh_Torque output in Fig 8.

Table 6. Functional Safety Requirement allocated to Safety architecture

Architecture entity	Functional Safety Requirement	Safety Goal	ASIL
Torque actuator (External measure)	If possible the technical solution for torque actuation shall provide a mechanism to limit executed torque in case of no torque request.	SG1	C
Torque translator	Pedal position shall be evaluated every time the torque request is calculated.	SG1	C
Torque translator	Actuated torque shall be reported by the TorqueTranslator.	SG1	C
Torque arbitrator	CC_Trq requests shall be ignored in other CC_Modes than On:SteadyState and On:Override.	SG4 & SG5	C
Torque disconnect	If a TrqDisconnect signal is active the output torque shall be disabled.	SG1	C
Torque monitor	Actuated torque shall be monitored and compared with calculated torque request.	SG1	C
Torque monitor	In case of a mismatch between actuated torque and requested torque a TrqDisconnect shall be activated.	SG1 & SG3	C
Driver control (external measure)	Pedal position shall mechanically be forced to 0% when driver does not press the pedal.	SG1	C

The results show that the *safety goals* that are derived from three different functions produce similar *safety requirements* across the safety architecture. This is expected since they share actuator requests and requirements. But it makes the architecting process critical as issues like safety mechanism independence and common cause scenarios becomes more complex. It shows that a structured way to handle the distribution of safety mechanisms is needed.

4 Discussion

The example describes a simple structure with few open issues on how the functions depend on each other. To make the same tree for a whole vehicle presents both challenges and possibilities. The challenges are structures where a child has more than one parent, and how to handle functional inheritance in these cases. Another challenge is how to make the structure scalable and with the correct level of detail. A third challenge is how to handle overridden requirement, where normally a *deriveReq* is used. With an unconditional *inherit* this possibility does not exist, to adapt the structure to the selection rules will be necessary. An example would be if there are requirements specific to BasicCC that are not applicable to ACC. These would be placed in the BasicCC child. The semantics makes this a perfectly viable solution. But with the structured way to view features comes the possibilities Global modes are defined in the behavior package and are available in the function definition for all vehicle features. Legal requirements valid for a type of feature can be placed at the right detail level in the technical feature model. Examples are EMC requirements that are typically placed on root level to be valid across all variants and configurations. Using PowerMode, a typical global mode stating the power state of the car, as an example. It will be used in requirements all through the feature tree, hence it will be implicitly linked to most features through parent requirements. The visibility of global modes becomes a tooling question where the implicit links can be used to support modeling. The requirements for the actual PowerMode implementation will be made somewhere in the feature tree though. It is a challenge modeling both variability of product features and feature functionality in the same information structure, this paper shows one solution this this problem using EAST-ADL2.

The results of the PHA and Safety architecture construction shows that it is necessary to perform certain activities more than once as dependencies on architecture level will not be visible on vehicle level requirements. This gives added importance to describing how iterations are performed and what inputs are needed to be reworked as details on analysis level are added.

It is not clear how *function* dependencies should be handled best. It can either be with explicit requirements on use of status and feedback to govern *function* behavior in situations where the dependency affects behavior or that a common PHA for the dependent situation is performed. From an ISO/DIS 26262 point of view the first method is probably preferable since it allows for a *function* based analysis. But from a 'proven in use' argumentation supported by ISO/DIS 26262 it is preferable that *function* requirements are kept unchanged when introducing new functionality into a system. Since all features are treated as independent when analyzed with respect to safety it is difficult to address whether a tree structure is always correct. Maybe a

structure of some other form is in its place when *functions* are modeled as branches breaking the tree structure.

5 Related Work

For safety-driven system development in the automotive domain, an emerging standard is the ISO/DIS 26262 on Functional Safety for Road Vehicles [1], representing the domain consensus on the state-of-the-art approaches to functional safety. The ISO standard implies a greater necessity for exchanging precise design information among partners in an automotive work flow. As most development today is distributed among several companies and departments, it is important that all information exchange is precise enough to enable the OEM (Original Equipment Manufacturer) to take full responsibility for the entire functional safety. Still, there is also a need to protect Intellectual Property (IP), because giving away detailed design information may jeopardize any commercial advantage.

A further standardization effort being taken into consideration is the SAE “Architecture and Analysis Description Language” (AADL), which has its roots in the avionics domain [13]. Compared to EAST-ADL2, AADL has a narrower scope: no explicit support is provided for variability management or requirements refinements and traceability. Specifics for automotive systems such as networks are weakly supported. The AADL is not designed for mass-produced systems and therefore has less emphasis on optimized overall solutions e.g. by considering compact run-time systems. For the automotive domain, the clash with AUTOSAR concepts is also a problem. However, wherever applicable, AADL concepts were reused, e.g. for dependability modeling.

Another approach is provided by the AltaRica language [11], which is also the basis for commercial safety analysis tools. AltaRica is a formal language transition system semantics for modeling erroneous behaviors in physical and computer systems. The language has been used in the ESACS project for system modeling [12]. While AltaRica and its tools enable the modeling of errors and formal safety analysis, there is currently no support for formal links to system design.

6 Conclusions

Feature based analysis is what is prescribed by the ISO/DIS 26262 standard but at early stages it is difficult to see *function* dependencies that will result in updated *function* requirements. This work shows that it is necessary to perform iterative analysis to manage changes in the safety architecture on analysis level and still meet the *function* specific Safety Goals derived on vehicle level.

Defining the semantics for requirement selection is necessary to ensure correct inclusion of requirements for a function definition. The semantics for requirements inheritance in an EAST-ADL2 model is presented in the paper. There may be a need to cover requirement exclusion with other methods than creating a parent feature that has the common functionality and child feature that carries the functionality not needed in the new child. A more structured approach for exclusion support could be useful.

Acknowledgements

This work is supported by the EU FP7 Project ATESSST2 (Grant 224442).

References

1. International Organization for Standardization: ISO/DIS 26262 (2009)
2. Chen, D., Johansson, R., Lönn, H., Blom, H., Walker, M., Papadopoulos, Y., Torchiario, S., Tagliabò, F., Sandberg, A., Biehl, M.: Integrated Fault Modeling for Safety-Critical Automotive Embedded Systems. Submitted to ISARCS (2010)
3. SAE ARP-4761: Aerospace recommended practice: guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment, 12th edn., SAE, 400 Commonwealth Drive Warrendale PA United States (1996)
4. Kletz, T.: HAZOP and HAZAN Identifying and assessing process industry standards, 3rd edn. Hemisphere, Washington (1992) ISBN: 1-56032-276-4
5. Palady, P.: Failure Modes and Effects Analysis. PT Publications, West Palm Beach (1995) ISBN: 0-94545-617-4
6. Vesely, W.E.: Fault Tree Handbook, US Nuclear Regulatory Committee Report NUREG-0492, US NRC, Washington, DC (1981)
7. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 1(1), 11–33 (2004)
8. Törngren, M., Chen, D., Malvius, D., Axelsson, J.: Model-based Development of Automotive Embedded Systems. In: Navet, N., Simonot-Lion, F. (eds.) *Automotive Embedded Systems Handbook*. Series: Industrial Information Technology. Taylor and Francis CRC Press (2008)
9. ATESSST2 (Advancing Traffic Efficiency and Safety through Software Technology), EU FP7 Project, <http://www.atesst.org>
10. AUTOSAR Development Partnership, <http://www.autosar.org>
11. Kehren, C., et al.: Advanced Multi-System Simulation Capabilities with AltaRica. In: 22nd Int. System Safety Conf., System Safety Society (2004)
12. Bieber, P., et al.: Safety Assessment with Altarica - Lessons Learnt Based on Two Aircraft System Studies. In: 18th IFIP World Computer Congress, Topical Day on New Methods for Avionics Certification, Toulouse France, August 26-26. IFIP (2004)
13. Papadopoulos, Y., Grante, C.: Evolving car designs using model-based automated safety analysis and optimisation techniques. *Journal of Systems and Software* 76(1), 77–89 (2005)
14. Parnas, D., Madey, J.: Functional document for computer systems. *Science of Computer Programming* 25, 41–61 (1995)
15. Tagliabò, F., Torchiario, S., Johansson, R., Lönn, H., Walker, M., Papadopoulos, Y., Sandberg, A., Chen, D.: Dedicated Modeling Support for the Automotive Functional Safety Standard. Submitted to SAFECOMP 2010 (2010)

Experiences in Applying Formal Verification in Robotics^{*}

Dennis Walter, Holger Täubig, and Christoph Lüth

Deutsches Forschungszentrum für Künstliche Intelligenz
Bremen, Germany

{Dennis.Walter,Holger.Taeubig,Christoph.Lueth}@dfki.de

Abstract. Formal verification efforts in the area of robotics are still comparatively scarce. In this paper we report on our experiences with one such effort, which was concerned with designing, implementing and certifying a safety function for autonomous vehicles and robots. We outline the algorithm which was specifically designed with safety through formal verification in mind, and present our verification methodology, which is based on formal proof and verification using the theorem prover Isabelle. The necessary normative measures that are covered are discussed. The algorithm and our methodology have been certified for use in applications up to SIL 3 of IEC 61508-3 by a certification authority. Throughout, issues we recognised as being important for a successful application of formal methods in the domain at hand are highlighted. These pertain to the development process, the abstraction level at which specifications should be formulated, and the interplay between simulation and verification, among others.

1 Introduction

While in some areas such as avionics, formal verification is well established in the development process, in other areas its use is still rare. One such area is robotics, in particular service robotics. This paper reports on our experiences when applying formal verification techniques to the certification of an algorithm calculating dynamic safety zones for an autonomous vehicle or robot which prevent it from colliding with stationary obstacles. Robotics as an application area offers its own challenges. Algorithms, often based on approximations and heuristics and implementing rather sophisticated computations such as the area covered by a braking robot in our case, play a central role. This leads to an increase in the importance of functional correctness. Further, the development process should be flexible, and allow us to iteratively develop both algorithms and specifications from the overall safety requirements. This can be contrasted to a rigid V-model with its strict separation of development phases which makes it expensive to ‘go back’ to earlier phases of the development when one discovers that the implemented algorithm is, in fact, safe but unusable in practice.

^{*} This work was funded by the German Federal Ministry of Education and Research under grants 01 IM F02 A and 01IS09044B.

Our methodology for specifying and verifying functional properties emphasises proving as well as testing. We demonstrate which measures can be covered by formal proof, even though the relevant standard IEC 61508-3 [10] focuses on testing, and argue that formal proof, on the other hand, allows us to relax the bureaucratic elements of the development process without losing reliability.

This paper is structured as follows: in Sec. 2, we give an overview over the project, showing the actual algorithm, the formal domain model, and our approach to specification and verification. In Sec. 3, we review our experiences made during the certification of the algorithm, pertaining to formal verification in the robotics domain and the development process we used, and highlight benefits and limitations of our approach.

2 The SAMS Project

2.1 The Safety Algorithm for Collision Avoidance

The algorithm which has been verified in SAMS is a collision avoidance algorithm, which protects a vehicle moving in a plane, e.g. an automated guided vehicle or service robot, from colliding with static obstacles. For that purpose, a safety zone is computed using the algorithm described in this section and then checked via a laser scanner whether there is an obstacle inside the safety zone. If so, the moving vehicle has to stop, otherwise it can safely continue its movement. The purpose of the verified algorithm is to compute a *safety zone* that is a superset of the *braking area* covered by the vehicle during braking (Fig. 1a).

Input. The algorithm takes as input intervals $[v_{min}, v_{max}]$ and $[\omega_{min}, \omega_{max}]$ which safely cover the measured translational and rotational velocities v and ω of the vehicle, a set of points $[R_i]_{i=1}^n$ which define the robots shape as their convex hull, and a list $(v_1, s_1), \dots, (v_m, s_m)$ of braking measurements for straight forward movements of the vehicle. Each pair consists of a velocity v_j and the corresponding measured braking distance s_j . We assume there is at least one measurement, taken at maximum speed. Furthermore, a latency t is given which parameterises the time the vehicle continues to drive with velocity (v, ω) before it starts to brake; it comprises the sum of the safety functions cycle time as well as any latency in the input data and the reaction time of the brakes.

Assumptions. First and foremost, we assume the robots braking trajectory to be a straight line or a circular arc; in other words, the steering of the vehicle remains fixed until the vehicle has completely stopped. A second assumption of the braking model is an energy consideration. It allows to transfer the braking distance measurement from straight motion to motion along a circle or even turning on the spot. Together, both assumptions establish the main braking model computation, which takes the initial velocity vector (v, ω) and delivers the so called *braking configuration* (s, α) . The braking configuration (s, α) consists of arc length s and angle α of the robots circular braking trajectory. In the case of $\omega = 0$, the angle α becomes zero and the braking configuration (s, α) describes a straight line of length s . Finally, besides static obstacles we assume the braking behaviour of the vehicle to be time-independent and location-independent.

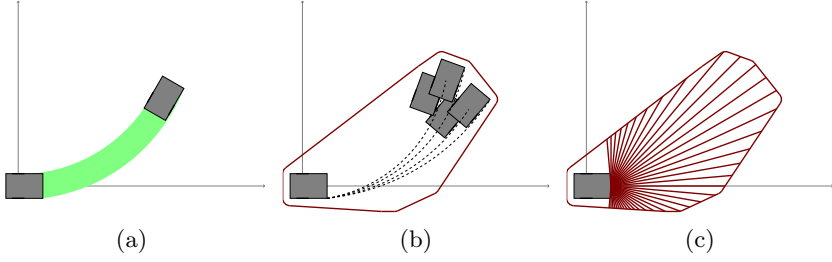


Fig. 1. Calculating the safety zones: (a) area covered by the vehicle during braking with a single braking configuration (s, α) ; (b) safety zone covering all braking configurations $[s_{min}, s_{max}] \times [\alpha_{min}, \alpha_{max}]$; (c) safety zone transformed into a laser-scan.

Algorithm. First, conservatively transform $[v_{min}, v_{max}] \times [\omega_{min}, \omega_{max}]$, the velocity configuration area, into the braking configuration area $[s_{min}, s_{max}] \times [\alpha_{min}, \alpha_{max}]$. The transformation *braking-configuration* (v, ω) describes a movement consisting of moving for time t with velocity (v, ω) , and then braking on a circular arc that retains the radius defined by v and ω .

[Step 1] For (v, ω) in $\{v_{min}, v_{max}\} \times \{\omega_{min}, \omega_{max}\}$, compute the braking configuration (s, α) as follows, and determine minimum and maximum s_{min} , s_{max} , α_{min} , and α_{max} of the four results:

$$(s, \alpha) = \text{braking-configuration}(v, \omega) \quad (1)$$

Then, compute the safety zone in terms of a finite set of points $[P_k]_{k=1}^K$ and a buffer radius q (Fig. 1b). The safety zone is an area $A^+([P_k]_{k=1}^K; q)$, given by the union of the convex hull of $[P_k]_{k=1}^K$ and the set of all points having distance of at most $q > 0$ to any point of that convex hull:

$$A^+([P_k]_{k=1}^K; q) = \{P + Q \mid P \in \text{conv} \{[P_k]_{k=1}^K\}, |Q| \leq q\} \quad (2)$$

[Step 2a] To compute points P_k : For all $(s, \alpha) \in \{s_{min}, s_{max}\} \times \{\alpha_{min}, \alpha_{max}\}$, compute the safety zone for the single braking configuration (s, α) in terms of a set of $n \cdot (L + 2)$ points¹ as the convex hull of

$$H_{s,\alpha} = \{[U_{i,s,\alpha}^1, U_{i,s,\alpha}^2, V_{i,s,\alpha}^0, \dots, V_{i,s,\alpha}^{L-1}]_{i=1}^n\},$$

where $U_{i,s,\alpha}$ and $V_{i,s,\alpha}$ are given as follows (for i in $1, \dots, n$):

$$\begin{aligned} U_{i,s,\alpha}^1 &= R_i & U_{i,s,\alpha}^2 &= T\left(\frac{s}{L}, \frac{\alpha}{L}\right) \cdot R_i \\ V_{i,s,\alpha}^0 &= U^1 + Q\left(\frac{\alpha}{L}\right) \frac{1}{2} (U^2 - U^1) & V_{i,s,\alpha}^j &= T\left(\frac{j \cdot s}{L}, \frac{j \cdot \alpha}{L}\right) \cdot V^0 \end{aligned} \quad (3)$$

with $T(s, \alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha & s \operatorname{sinc} \frac{\alpha}{2} \cos \frac{\alpha}{2} \\ \sin \alpha & \cos \alpha & s \operatorname{sinc} \frac{\alpha}{2} \sin \frac{\alpha}{2} \\ 0 & 0 & 1 \end{pmatrix}$ and $Q(\alpha) = \begin{pmatrix} 1 & \tan \frac{\alpha}{2} & 0 \\ -\tan \frac{\alpha}{2} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

¹ The parameter L determines the number of auxiliary points when computing the convex hull of an arc.

Now, $[P_k]_{k=1}^K$ is the result of a standard convex hull algorithm like Graham scan applied to the union of the $H_{s,\alpha}$ for all $(s, \alpha) \in \{s_{\min}, s_{\max}\} \times \{\alpha_{\min}, \alpha_{\max}\}$.

[Step 2b] The buffer radius q includes a conservative error approximation for the algorithm, and is given as

$$q = \frac{1}{6} \left(\frac{\alpha_{\max} - \alpha_{\min}}{2} \right)^2 \max \{|s_{\max}|; |s_{\min}|\} + \left(1 - \cos \frac{\alpha_{\max} - \alpha_{\min}}{2} \right) \max_{1 \leq i \leq n} \{|R_i|\} \quad (4)$$

[Step 3] Finally, $[P_k]_{k=1}^K$ is transformed into scanner coordinates, and the safety zone $A^+([P_k]_{k=1}^K; q)$ is sampled into a laser-scan like representation (Fig. 1c).

Output and guarantees. If the vehicle satisfies the assumptions described above and if the input parameters are correct or at least conservative, the algorithm guarantees the correctness of the safety zone. This means that the vehicle will always be able to stop within the area defined by the braking zone. More precisely, no part of the vehicle will leave that area at any time while first driving with constant velocity (v, ω) for time t and then braking down to standstill.

The guarantee given consists of two major properties: first, correctness of the braking model computation (Step 1), and second, the correct and strictly conservative computation of the area $A^+([P_k]_{k=1}^K; q)$ (Step 2). Both properties have been formally proven in the Isabelle theorem prover.

2.2 Formalising the Domain Model

In order to be able to state and verify safety properties about the algorithm introduced in Sec. 2.1, we need a formal model of the *domain* of the algorithm, that is the world of two-dimensional moving objects. This model is used for the specification and verification of the concrete source code. Therefore, by formal model we mean a collection of theories of the theorem prover Isabelle [14], and not merely a pen-and-paper formalisation.

The contour of the robot, for instance, is modelled as a convex polygon, and obstacles are simply connected sets of points. The main safety property we will need to formalise is that the area traversed by the robot while braking from velocity (v, ω) is covered by the safety zone calculated for that velocity, given by equation (2) above. In the notation of Sec. 2.1, we require the following to hold:

$$\text{braking-area}(v, \omega) \subseteq A^+([P_k]_{k=1}^K; q) \quad (5)$$

Isabelle provides a rich base of theories concerned with concepts such as real numbers, polynomials, or set theory. In the process of domain formalisation this base is extended with the concepts and theorems relevant to our concrete model. For example, the following shows the definition of the property of convex sets of points; this definition has been copied verbatim from the corresponding Isabelle theory file ($*_R$ denotes scalar multiplication):

definition

convex :: "Point set $\Rightarrow$ bool"

where

"convex $K \equiv (\forall x \in K. \forall y \in K. \forall t. (0 \leq t \wedge t \leq 1) \longrightarrow$
 $(t *_{\mathcal{R}} x + (1-t) *_{\mathcal{R}} y) \in K)$ "

Or in words, a set of points is convex iff for any two points x, y all points on the line between them are in the set as well. We can now define the convex hull of a set of points X as the intersection of all convex sets K containing X :

definition

convex_hull :: "Point set $\Rightarrow$ Point set"

where

"convex_hull $X = \bigcap \{K . \text{convex } K \wedge X \subseteq K\}$ "

These definitions are almost identical to what one can find in mathematical textbooks. This is particularly valuable in a certification context, as theory files can be reviewed without an in-depth knowledge of Isabelle syntax. The Isabelle equivalent of (5) involves slightly too many concepts whose definitions we must elide for reasons of brevity. To give the reader a taste of a more involved model concept, we present a theorem about the approximation of an arc by the convex hull of three points:

lemma *arcpoint_in_convex_hull'*:

"[$|\varphi| < \pi$; $0 \leq t \wedge t \leq 1$;
 $Q = \text{arcendpoint } s \ \varphi \ P$; $K = \text{convexpoint } P \ Q \ \varphi$]
 $\implies \text{arcendpoint } (t*s) \ (t*\varphi) \ P - P \in \text{convex\_hull } \{K-P, Q-P, 0\}$ "

If Q is the endpoint of a circular arc (defined by its length s and angle φ), starting at P , then the convex hull of P, Q and a third point K (whose computation via *convexpoint* is irrelevant here) will contain every point on the arc (computed via *arcendpoint* by scaling s and φ).

Our domain modelling consists of 11 theory files, containing about 110 definitions and 510 lemmas and theorems. It was developed in about five months by a researcher with a good background in mathematics, but no previous knowledge of Isabelle, supporting the claim that mathematics is the key, and the technicalities of Isabelle do not distract one unduely from the actual formal development.

2.3 Specification and Verification

Certifying a software module involves verification on several levels: design requirements need to be traced back to system (safety) requirements, code needs to be verified according to the V&V plan and against the design specification, and during integration there are several verification activities based on requirements set forth in the corresponding specification phase. For each of these levels a well-defined procedure for specifying requirements is needed. In the SAMS project we stressed *functional correctness*, the verification of the functional behaviour of concrete code. This comprises both the absence of runtime errors like array-out-of-bounds or division-by-zero (a property we call *program safety*), and correctness of the results of computations, as defined by formal specifications.

We consider functional correctness important because it is mandated by standards like IEC 61508-3 to ensure program safety properties on the code level, and moreover, because robotics algorithms as the one described in Sec. 2.1 involve very complex computations whose correct implementation is hard to verify by a mere code review. In the terminology of Heitmeyer et al. [8] they deserve a thorough and detailed analysis which is not possible in *operational models* (e. g. state machine models) and for which an *axiomatic* approach like the one presented here is well suited.² We consider a purely operational analysis insufficient, because it is equally important to analyse the data-related and the control-related aspects of software systems in robotics, where operational models focus on the latter. The former lends itself ideally to a declarative, ‘axiomatic’ specification. Moreover, models for robotics require non-discrete data as measurements of real world entities are involved.

As an example from our own code: one subroutine of the algorithm of Sec. 2.1 approximates the arc along which the vehicle brakes by a polygon. The complete specification (roughly stating that the arc is included in the area covered by the polygon) invariably leads to the use of mathematical concepts like convex sets of points, set intersection, etc. Furthermore, the correct execution of the system’s safety function crucially depends on the correct design and implementation of this approximation. In particular, a flaw in this subroutine can be very hard to detect. This is an important difference to, e. g., program crashes (which an external watchdog may detect) or Boolean circuits (whose input domain can be tested much more thoroughly).

To express the functional properties of interest we designed a formal language for the high-level specification of the functional behaviour of C programs. The language lies in the tradition of design by contract languages like JML [4] or ACSL [2], where program functions are annotated with preconditions, postconditions and a modification frame limiting the effect of function execution on memory changes. Our language additionally allows to include higher-order logic expressions in the syntax of the theorem prover Isabelle in specifications. This gain in language expressivity is the crucial ingredient for allowing more abstract specifications in which program values are put in relation to their corresponding domain values. The desired properties of functions are then expressed in terms of the domain language as it was formalised in Isabelle.

An example specification is given in Fig. 2: it uses the concepts *is-RT* and *RT* from the domain formalisation as well as the function composition operator $\circ$ of Isabelle/HOL to concisely express that `comp_transform` is an operation that composes two rigid body transforms. *RT* is what we call a representation function, which lifts a C value of type `RigidTransform` into its domain equivalent. *is-RT* is a predicate that recognises all C values that actually represent rigid body transforms. (The internal representation of `RigidTransform` are 3×3

² We observe a slight collision of terminologies here: in the theorem proving community, an axiomatic approach is distinguished from a definitional approach, in which theories are derived from first principles and new concepts are built on top of existing ones. We do not use this interpretation of ‘axiomatic’ in this paper.

```

/*@
  @requires $!is_RT(a2b) && $!is_RT(b2c)
  @memory \ valid {a2b, b2c, a2c} &&
    *a2c <*> (*a2b <+> *b2c)
  @ensures $!is_RT(a2c) &&
    ${ ^RT{a2c} = ^RT{b2c} o ^RT{a2b} }
  @modifies *a2c
  @*/
void comp_transform( const RigidTransform * a2b,
                    const RigidTransform * b2c,
                    RigidTransform * a2c );

```

Fig. 2. An example specification of a C function, directly using the domain vocabulary as defined by the formalisation in Isabelle/HOL. The **@memory** annotation requires that **a2c** is not aliased with **a2b** nor with **b2c** and that all three are valid pointers.

matrices, which hence include other transformations, too.) Further details about the specification language and how functions can be proven correct in Isabelle w.r.t. their specification have been described in a previous paper [12].

We now sketch the steps that are taken in the specification and verification workflow. To reiterate our setting: To attach specifications to code, there must be code; while in the classic V-model, code enters rather late in the process, to specify we merely need the function interfaces, not the complete implementation. Moreover, in our case implemented prototypes are available early in the development, a point we elaborate on in Sec. 3.2. Assuming that the code for those parts of the program that are put under formal scrutiny are available, and that the domain model has been sufficiently formalised so that at least the relevant definitions that will be used in specifications exist, our workflow looks as follows (see also Fig. 3): (1) C function interfaces (declarations) are annotated with their respective specifications: those functions that implement operations with an analogue in the domain, like geometric transformations, are specified with the help of the domain vocabulary given by the formalisation; (2) In specification reviews both the completeness of the specification w.r.t. more high-level, natural language specifications as well as implementation-related issues are discussed. The latter include issues like the restructuring of code to ease verification, or the elimination of language constructs that are not supported by the verification environment; (3) After the functions have been implemented, they are checked for obvious deviations from their specified behaviour in code reviews. At least one programmer, one specifier or domain expert and one verifier takes part in these meetings. Such meetings give the verifier an understanding of how the code works, which is crucial for the verification to succeed. (4) The translation of C functions into the Isabelle formalisation is done modularly: to translate function **foo** for a verification attempt, only its source code and the specifications of all functions called by **foo** as well as obviously that of the function itself are required. A front-end parses and analyses these entities and emits Isabelle terms representing their abstract syntax; (5) Making use of the domain formalisation

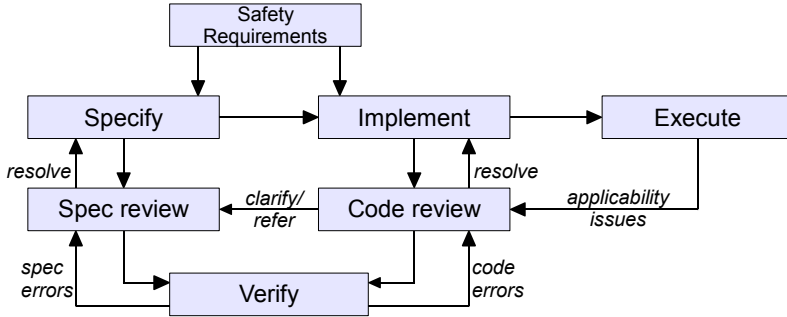


Fig. 3. The specification and verification workflow, not enforcing a temporal dependency between design specification and code, which is ensured by the final verification

and a couple of automatic procedures written in Isabelle, a human verifier interactively proves these functions correct. This is a labour-intensive part, and the core of formal verification. (6) In the case that a function cannot be verified, a specification or code review is called and as soon as the error has been understood, modifications on either the design specification, the code, or the proof script are undertaken, leading to a reiteration of the process just described.

3 Lessons Learnt

3.1 Formal Verification in the Robotics Domain

Challenges. The functionality of robotics systems is mainly implemented in software, making use of increasingly complex algorithms. With growing system functionality the difficulties of ensuring safety increase, in particular in the face of moving towards mixed human-robot work places instead of physically separating machine operators and robotic devices. Functional correctness occupies a growing fraction of the overall integrity of systems. This requires deep analytical safety considerations and makes formal verification an eligible tool in the robotics domain. Characterised from a safety point of view, the domain stands out by its rich specifications, which usually are not as simple as “temperature never exceeds 90 °C”, and its rich domain involving high level concepts from mathematics, kinematics, and other areas of physics.

In practice, applying formal verification in the robotics domain faces the conflict between real-world applications involving unstructured environments and inaccurate sensors, and their idealised modeling in specifications and the formalised domain. Addressing this problem is not unique but especially important for projects applying formal verification in real world applications. Safety requires that reality conforms to the assumptions made in the model.

Another challenge in the robotics domain is the conflict between safety and practical issues like availability. It is quite easy to design a safe algorithm which is unusable in practice. For example, a service robot will not be able to navigate

through doors anymore if its safety zones are calculated too large. Avoiding these safe but non-applicable results as well as identifying the aforementioned kind of conflicts between models and reality is mainly done by evaluating parts of the implementation in practice. Occuring conflicts may result in changes of the design specifications. For that reason, our development process is very iterative and code-centric. Availability is not verified, it is tested in practice and in simulations, so it is important to be able to run the algorithm early in the development process.

Successful design and verification of robotics algorithms. Two concepts that proved helpful were the explicit use of intervals to accommodate for imprecision, and algorithms and representations from computational geometry.

To overcome the discrepancy between the real world and the idealised domain, and to account for imprecise measurements, our algorithm calculates safety zones for sets of velocities $[v_{min}, v_{max}] \times [\omega_{min}, \omega_{max}]$ instead of single ones.

Another benefit came from the representation of objects as sets of points, which not only led to efficient computations, but also allowed for mathematically pleasing proofs for major parts of the algorithm. This seems to hold true for many representations and algorithms from computational geometry. Of course, other proofs were pure grind work, such as the proof of the following (for $\omega \neq 0$):

$$\sqrt{\left(\frac{\omega - \sin \omega}{\omega^2} - \frac{1 - \cos \omega}{\omega}\right)^2 + \left(-\frac{\sin \omega}{\omega} + \frac{1 - \cos \omega}{\omega^2} - \frac{\omega - \sin \omega}{\omega^3}\right)^2} \leq \frac{2}{3}$$

Domain. Robotics is well suited for formal verification. Formalising high-level concepts is admittedly very time-consuming. Nevertheless, much can be taken directly from textbooks so that the formalisation in Isabelle went rather smoothly. Moreover, the effort is worthwhile, as it allows simpler specifications and verification. The domain modelling is reusable for other projects, independent of a reuse of the implementation.

3.2 Specification Process

Verification as a joint effort. One aspect of formal verification is that because correctness relies on formal proof, it is not that crucial anymore to strictly separate the roles of tester and implementer. In contrast, the close cooperation between the verifier and the implementer boosted productivity in our case: verification became a joint effort. Writing specifications which validate the safety requirements, and can be formally verified, is not easy; it requires an understanding of the implementation, the domain model, and how the verification works. It is easy to specify something which is correct but cannot be verified; on the other hand, it is also a temptation to write low-level specifications which just restate what the code is doing in elementary terms without the abstraction required to state useful safety properties.

A somewhat unusual example of a close collaboration between implementer and verifier is a change of the implementation induced by verifiability considerations. The function `abstasten` converts the buffered polygon into a sequence of

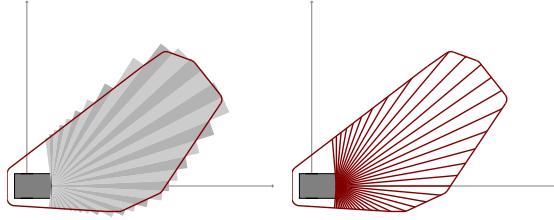


Fig. 4. Two ways of converting a buffered polygon into a laser scan representation

vectors corresponding to a laser scan (Step 3 from the algorithm presented in Sec. 2.1; see Fig. 4). Initially, the specification interpreted the resulting sequence as the rays of an idealised laser scanner. We switched both specification and implementation to a sector-based interpretation, in which each result describes the whole area of a sector. This fitted in well with the other specifications and allowed us to specify the result simply as a superset of the actual safety zone, and was easier to verify formally.

Code-centric specification and verification. We experienced an interesting interplay between specification, implementation and application: at first, the specification required that if the speed of the vehicle exceeded the maximum speed for which a braking distance was measured (cf. Sec. 2.1), an emergency stop should be initiated. However, this turned out to be too restrictive: in typical applications, the measured maximum velocity v_m may be exceeded occasionally by a small margin, and initiating an emergency stop in these situations would severely reduce availability. Hence, the braking distance for speeds larger than v_m was safely overapproximated, and the specification amended accordingly.

The importance of being formal. Formal specification necessitates to state requirements precisely. A beneficial side effect is that it focuses discussions and manifests design decisions. Besides the well-known issue of the ambiguities in natural language specifications, it turned out to be easier for specifiers *and* implementers to use the vocabulary of the domain formalisation to state these requirements and to reach agreement on their respective meaning. For quick sanity checks of specifications written down or modified during meetings, we provide tool support for the type-checking of specifications. This pertains both to code-related specification expressions (e.g., types of program variables) as well as Isabelle expressions used in code specifications. A typical specification meeting would end with a function specification reviewed and typechecked.

Beyond that, formal verification can uncover hidden assumptions, both in the specification and even in the domain (see also [6]). As an example, when verifying the overapproximation for speeds beyond maximum speed, it turned out that the initially defined quadratic approximation was not enough, and a cubic one was necessary to formally prove the relevant properties.

3.3 Formal Verification in a Certification Process

V-model. The V-model of IEC 61508-3 asks for traceability between adjacent phases on the downward leg, i. e. from the system safety requirements down to the code, as well as ‘horizontal’ verification on the upward leg from code to the integrated and validated system, where appropriate tests ensure the satisfaction of all requirements. The model somewhat neglects model-based analysis and does not assign it a specific level; it might be considered part of the software architecture, but in any case has a direct link to the safety requirements. A definite strength of our methodology is the very strong link between this analysis level and the concrete source code (at the bottom of the V-model): Formal code verification in our methodology ensures both traceability between code and module design, *and* between module design and the analysis level. The main reason for this is the high level of abstraction of code specifications, in which the domain formalisation is directly embedded. For example, take the basic function computing a polygonal approximation of the curve described by a single point of the vehicle’s contour during an emergency stop. Its specification directly expresses that the area described by the returned polygon completely contains the braking curve in the two-dimensional environment model.

Modularity. Modular verification on a function-by-function basis allowed us to focus formal verification on those functions which are crucial to functional correctness; other functions may contain constructs that our tool cannot reason about, or may not pertain to global correctness (e. g., logging), and can be treated more adequately by manual review or informal proofs.

Open-minded authorities. To our surprise the external reviewers from the certification authority were quite open-minded towards the use of expressive (higher-order) formal logic for specifications and an interactive theorem prover for doing the actual verification. In our case this was Isabelle/HOL, but its specifics did not play an important role and HOL4 or Coq or any other well-known prover with an active research community, proper documentation and a large enough number of global usage hours would have worked.³

Certification of the tool itself. Even though there are indications that structured specification and verification actually increase cost-effectiveness [1], their use is most often induced by the external requirement of a safety certification. A convincing argument for a tool like ours is that its use covers several items on the list of required design and verification measures. Concretely, to claim compliance with IEC 61508-3, the measures listed in its Annexes A and B have to be considered. As confirmed by the certification authority, our tool covers several of these, which we will now briefly discuss. With regard to software design and development (A.4), four out of six measures are covered: the use of formal

³ We actually estimated the number of hours that Isabelle has been in serious use (as $2 \cdot 10^6$ hrs). This technique of showing that a tool is ‘proven in use’ is commonly applied for non-certified compilers.

methods, of computer-aided design tools, of design and coding guidelines, and of structured programming. Missing are defensive programming and modularisation. The standard interprets modularisation structurally, and our tool does not apply code metrics.⁴ In contrast, the modularity we do achieve is of a more *behavioural* nature: the effect of a function is summarised in its interface specification, even though the function body might be of arbitrary size and complexity.

Concerning software verification measures (A.9), we cover formal proofs and static analysis. The latter includes the measures marginal value analysis, control as well as data flow analysis and symbolic execution. Whereas our Hoare-logic style verification resembles a symbolic execution, many properties that are derived from the other analyses, like ensuring that only initialised variables are read, are also subsumed by formal verification.

However, most of the work in a verification effort goes into testing, so one would require that the overall amount of functional testing can be reduced in a development process using formal verification. In our case, the only tests that had to be performed on the module level were related to over-/underflow and numerical stability. No functional testing had to be performed for the formally verified units, due to the level of detail at which both specifications and the programming language are modelled.

Limitations. Our tool focuses on functional correctness, and does not consider aspects like execution time analysis and bounds, resource consumption, concurrency, and the interface between hardware and software. This is a clear separation of concerns, as it is becoming common consensus that only the use of multiple, specialised tools and methodologies can achieve a high level of confidence in software [9]. There are further limitations in the realm of functional properties and run-time errors. Like other formalisations, we idealise the numerical domains that programs work on from bounded integers and floating-point numbers to mathematical integers and real numbers, which may in exceptional cases result in undetected run-time errors (see [12] for an example). The price we had to pay to obtain a formalisation in which interesting, abstract, functional properties can be proved with tolerable effort was a slight mismatch between the actual and the formal semantics.

Some more notes on practical formal verification. There are of course also problems with using formal verification as described here. A major annoyance is the fragility of proofs, i. e. their lack of robustness w. r. t. changes in source code. This particularly hurts in the face of interactive verification: proofs are not generated automatically by a push-button tool, but proofs scripts are written by humans – even if they sometimes only consist of a sequence of calls to automatic proof tactics. We easily support ‘regression verification’, i. e. the automatic checking of all existing proofs of correctness against modified source code as well as modified specifications. Unfortunately, however, many proofs ‘break’ even under only minor modifications like the rearrangement of statements or a semantics-preserving rewriting of expressions, so that the proof scripts need to be adapted manually.

⁴ In practice, functions which can be formally verified with tolerable effort adhere to these structural properties anyway.

4 Conclusions

The SAMS project is an example of the successful application of formal verification in a certification context. The algorithm and implementation have been certified as conformant to the requirements of IEC 61508-3 Software SIL 3 development by TÜV Süd. The same goes for the verification environment, which has also been confirmed as according to IEC 61508-3, covering various measures in the appendix as elaborated above. Both the tool and the domain modelling can be reused in other projects.

Related Work. We are not aware of many other formalisations in the robotics domain, except for specific, idealised algorithms like Graham scan [13]. Other certification efforts using formal verification in our sense include [1], which also use pre-/postconditions, but in a discrete domain; cf. also recent work concerning the verification of operating systems using Isabelle [16], or the VCC framework [5]. Most of these results idealise floating-point numbers to reals; an exception are Boldo and Filliâtre [3], who verify floating-point computations with exact error margins, something which in the robotics domain would be particularly valuable. Peleska [15] integrates formal approaches and testing, using abstract interpretation. It would be interesting to reuse the results of analyses like these for the formal verification, in particular to discharge program safety proof obligations. Haddadin et al. [7] perform a systematic evaluation of safety in human-robot interaction, quantifying injury risks based on experiments and classifying contact scenarios. Their work is upstream to ours, as it contributes important data for a hazard analysis and helps to improve safety by construction, while our approach assumes the criticality of collisions and aims at avoiding them. Krishna et al. [11] claim to develop provably safe motion strategies for mobile robots, but provide validation merely in terms of classical simulation and experimental results.

Summary. This paper has presented our experiences when conducting the formal verification and certification of a robotics algorithm. To close, we would like to recap our three main points. Firstly, we have argued that *functional correctness* is a key aspect of system integrity in robotics applications. Secondly, it is important to have a *strong link* from safety concepts down to the executable code. This was achieved by including domain model concepts directly in the specifications. Thirdly, because the correctness of all proofs are checked by Isabelle, we could relax some of the formalities of the development process to the benefit of all. Instead of using a rigid V-model, we had a convergence of both design specifications and implementation down to verified implementation in an *iterative process* starting from the initial safety requirements.

Besides the tool itself, we hope that the experiences laid out in this paper might be of use to other researchers and practitioners. We envisage a similar approach, using our tool or similar ones, to be applicable in all areas concerned with functional correctness to the degree that robotics is.

References

1. Barnes, J., Chapman, R., Johnson, R., Widmaier, J., Cooper, D., Everett, B.: Engineering the tokeneer enclave protection software. In: ISSSE 2006, IEEE Computer Society, Los Alamitos (2006)
2. Baudin, P., Filliâtre, J.-C., Marché, C., Monate, B., Moy, Y., Prevosto, V.: ACSL: ANSI C specification language (October 2008), Version 1.4, http://frama-c.cea.fr/download/acsl_1.4.pdf
3. Boldo, S., Filliâtre, J.-C.: Formal verification of floating-point programs. In: ARITH18, Montpellier, France, June 2007, IEEE Computer Society, Los Alamitos (2007)
4. Burdy, L., Cheon, Y., Cok, D.R., Ernst, M.D., Kiniry, J.R., Leavens, G.T., Leino, K.R.M., Poll, E.: An overview of JML tools and applications. *Int. J. STTT* 7(3), 212–232 (2005)
5. Cohen, E., Dahlweid, M., Hillebrand, M., Leinenbach, D., Moskal, M., Santen, T., Schulte, W., Tobies, S.: VCC: A practical system for verifying concurrent C. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) *Theorem Proving in Higher Order Logics*. LNCS, vol. 5674, pp. 23–42. Springer, Heidelberg (2009)
6. Frese, U., Hausmann, D., Lüth, C., Täubig, H., Walter, D.: The importance of being formal. In: *SafeCert 2008. ENTCS*, Elsevier Science, Amsterdam (2008)
7. Haddadin, S., Albu-Schaffer, A., Hirzinger, G.: Requirements for safe robots: Measurements, analysis and new insights. *Int. J. Robot. Res.* 28(11-12), 1507–1527 (2009)
8. Heitmeyer, C., Jeffords, R., Bharadwaj, R., Archer, M.: RE theory meets software practice: Lessons from the software development trenches. In: *RE 2007*, pp. 265–268. IEEE Computer Society, Los Alamitos (2007)
9. Hoare, C.: Viewpoint retrospective: an axiomatic basis for computer programming. *Commun. ACM* 52(10), 30–32 (2009)
10. IEC. IEC 61508 – Functional safety of electrical/ electronic/ programmable electronic safety-related systems. IEC, Geneva, Switzerland (2000)
11. Krishna, K.M., Alami, R., Simeon, T.: Safe proactive plans and their execution. *Robot. Auton. Syst.* 54(3), 244–255 (2006)
12. Lüth, C., Walter, D.: Certifiable specification and verification of C programs. In: Cavalcanti, A., Dams, D.R. (eds.) *FM 2009*. LNCS, vol. 5850, pp. 419–434. Springer, Heidelberg (2009)
13. Meikle, L.I., Fleuriot, J.D.: Mechanical theorem proving in computational geometry. In: Hong, H., Wang, D. (eds.) *ADG 2004*. LNCS (LNAI), vol. 3763, pp. 1–18. Springer, Heidelberg (2006)
14. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)
15. Peleska, J.: A unified approach to abstract interpretation, formal verification and testing of C/C++ modules. In: Fitzgerald, J.S., Haxthausen, A.E., Yenigun, H. (eds.) *ICTAC 2008*. LNCS, vol. 5160, pp. 3–22. Springer, Heidelberg (2008)
16. Tuch, H.: Formal verification of C systems code. *J. Autom. Reasoning* 42(2-4), 125–187 (2009)

Evolving a Safe System Design Iteratively

Alexandre Mota¹, Joabe Jesus¹, Adriano Gomes¹,
Felipe Ferri², and Edson Watanabe²

¹ Centro de Informática - Universidade Federal de Pernambuco
P.O. Box 7458 - Zip 50740-540
{acm,jbjj,ajog}@cin.ufpe.br

² Embraer - Av. Brig. Faria Lima, 2170 - Zip 12227-901
{felipe.ferri,edson.watanabe}@embraer.com.br

Abstract. ARP4754 suggests that, whenever possible, aeronautical safety critical systems may be developed as well as checked in an incremental way. But in practice the safe design emerges from the functional essential design in a discontinuous fashion. Engineers take several decisions in the direction of safety that sometimes can loose some of the desired functional characteristics. This can increase the development cost by only detecting functional problems in late phases of the development life cycle. In this paper we propose a strategy that starts from an initial proposed design, where functional behavior is investigated using model checking, and evolves to a reliable and safe design in a stepwise fashion. At each step, where safety aspects are introduced, safety constraints are checked using probabilistic model checking (Markov analysis). The final design emerges when we cannot find any safety violation.

1 Introduction

Safety critical systems must assure that a given design satisfies safety constraints [2]. Ideally, the safe design¹ may emerge by a gradual evolution of the functional design, but in practice they appear to be disconnected [18,16]. Therefore, part of or all the effort spent in the functional design assessment can be repeated in the safe design as well as late corrections can only be detected at the system design phase of the development life cycle.

Nowadays model checking [5] and probabilistic model checking [15] are being used extensively in proposals to investigate characteristics of functional as well as safe designs, respectively. But the two worlds are still disconnected [8,18]. Furthermore, due to such a disconnection model checking is sometimes used in phases where the problem of state explosion is more severe, and the need of application of techniques and strategies to decrease model complexities are inevitable [6,9]. Checking functional properties in a final (safe) design is much more expensive than checking them in a functional essential design.

¹ By safe design we mean a design that satisfies safety requirements only. In this paper we are not considering other non-functional requirements such as overall cost, weight, energy consumption, etc.

In this paper we show that an alternative to connecting the two worlds can be applying behavior preserving refactoring using Simulink [19] elements themselves. This is interesting because we can use model checking in functional designs where models are smaller, probabilistic model checking to detect if an intermediate model already satisfies the desired safety constraints and apply refactoring exactly where a safety constraint is violated. But we still consider engineers are responsible for choosing a refactoring and the place to apply it.

The main contributions of this paper are:

- A strategy to create a safe design from a functional design that investigates functional properties only once and in the best phase of the process: when the model is small (the essential behavior);
- A quantitative safety analysis whose results are useful to guide the introduction of safety aspects in the design;
- The idea of Simulink-based refactoring to add safety aspects in a design.

This work is organized as follows. In the next section we present our main contribution, where in Section 2.1 we show our developed functional analysis based on the process algebra CSP [23], in Section 2.2 we briefly present our safety analysis based on the probabilistic model checker Prism and in Section 2.3 we discuss the use of refactoring as a mean to introduce safety aspects in a functional design in a stepwise and conservative way. In section 3 we show the application of our strategy in a simple but common aeronautical subsystem. Finally, in Section 5 we show our conclusions and future work.

2 The Proposed Strategy

ARP4754 suggests that, whenever possible, an aeronautical critical system may be developed gradually, in such a way that if a certain vulnerability is found the system design can be updated locally. This is also pointed out by the work reported in [18] as future work.

In this work we propose a strategy to tackle this problem; it is illustrated in Figure 1. This strategy considers as inputs two artifacts: the system requirements (1) and an essential functional design (2). From the functional design, in general a Simulink block diagram, we automatically extract a behavioral model described in CSP using the work reported in [14] (but other models can also be extracted [8,26]). To analyse this model (A), we manually create functional properties based on the system requirements and apply the CSP model checker FDR [10] (see Section 2.1). In general, a first design does not satisfy all desired properties. It needs to be adjusted, which generates a new functional design. As such an adjustment is very specific to each problem being solved, we consider it is done by engineers decisions (B)². This effort can take several cycles of re-work until the design satisfies all desired functional requirements, originating a preliminary system design (4).

² Engineers can use information provided by a model checker (for instance, a counter-example) or a theorem-prover (for instance, a dependence in a predicate that does not reduce to true).

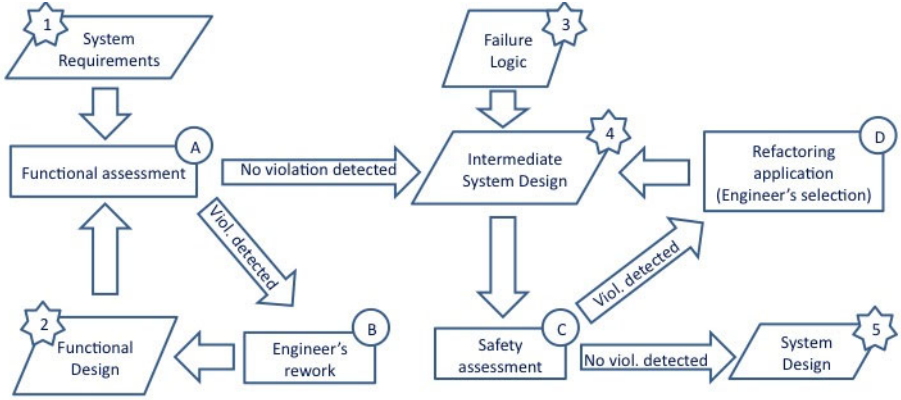


Fig. 1. The proposed strategy

With the intermediate system design, another concern takes place: perform safety assessment (C). As safety constraints—at validation level—are mostly quantitative, ignoring a quantitative analysis of them can be dangerous [4]. Typically such constraints are stated as “failure condition X must not occur more frequently than 10^{-y} times per flight hour”, where y depends on the criticality associated to failure condition X. Following ARP4761 [2], hazardous ($y = 7$) and catastrophic ($y = 9$) failure conditions must be satisfactorily analyzed on a quantitative basis (in addition to a qualitative analysis as well), because these kinds of failures are critical.

To perform safety assessment, engineers must provide the failure logic of the system (3) [21]. The goal in this stage is to satisfy criticality constraints (for instance, assuring that a failure condition has an average failure probability inferior to 10^{-9} per flight hour) [2]. Traditionally, this means using fault-tree analysis (FTA) to detect safety violations. But we also have alternative ways of doing this [11,12,15]. This work considers the probabilistic model checking solution reported in [11] (see Section 2.2), which is based on the Prism model checker [15]. A Prism model as well as CSL formulas are extracted directly from an annotated Simulink diagram corresponding to all possible failures. By detecting that some CSL formula is *false*, which means the corresponding failure condition was violated, engineers may decide on several options available to fix the problem [7]:

1. Add redundancy (more components of the same type);
2. Add dissimilarity (replace a single component by a set of other components with similar functionality and different design principles);
3. Replace a component by another equivalent one of superior reliability;
4. Add monitors (to avoid latent failures);
5. Increase maintainability tasks associated to critical components (Although this does not change the architecture).

Traditionally, the previous decisions are incorporated in the system design by engineers using their past experiences. In this work we assume a different viewpoint.

Instead of introducing safety aspects by using engineers experience only, the idea is to provide to engineers behavior preserving refactoring³ [22], corresponding to the previous safety options (see Section 2.3). We assume engineers decide on the most convenient refactoring to apply and where. These refactorings have some practical interesting properties. They are:

1. Scalable - a refactoring is independent of the size of the design;
2. Mechanisable - easy to implement in simulators with extensions/plugin-ins/add-ins features (For instance, Simulink [19]);
3. Stepwise - they are applied gradually in a step-by-step fashion;
4. Analysable - they can be proved correct by theorem proving [13] and trustworthy by testing [24].

Furthermore, we propose that the introduction of safety aspects is guided by the result of the safety analysis. Based on the violated failure conditions, engineers locate the region of the design where this violation occurs and apply a refactoring (D). As the application of such a refactoring yields a new design where functional behavior is preserved, we only need to update the failure logic (3) and perform safety analysis (C) again until no more violations occur. When this happens, we obtain a safe system design (5). It is worth noting that in this work we are assuming that engineers are responsible to provide the failure logic; they can consider special situations that cannot be inferred in general. But there are works in the literature that already indicate a possible solution in this direction [3].

2.1 Checking Functional Behavior

Our analysis of functional behavior is guided by the strategy depicted in Figure 2 and reported in [14]. It starts with the translation of the system architecture (requirements) into a CSP specification, a non-automatic step. Afterwards, another CSP specification is generated through the translation from the Simulink block diagram of the system controller (Similar to the work reported in [26]). These specifications are combined to build a complete model that has all controller components and the *nominal behavior* of the architectural components of the system. Then, desired properties of the system are converted into CSP processes. These processes monitor the complete model, deciding whether the complete model satisfies or not the given property (similar to a test oracle). The complete model is abstracted to avoid state-explosion and analysed using the FDR model-checker. The analysis can produce counter-examples used as input to change the specifications (block diagram and architecture) of the system.

Although other languages could be used to formally validate the functional design, we adopted the process algebra CSP because it has a rich set of operators suitable for describing complex systems from a composition of simpler ones.

³ By behavior preserving refactoring we mean a transformation that preserves functional properties by construction. That is, they do not need to be checked each time they are applied.

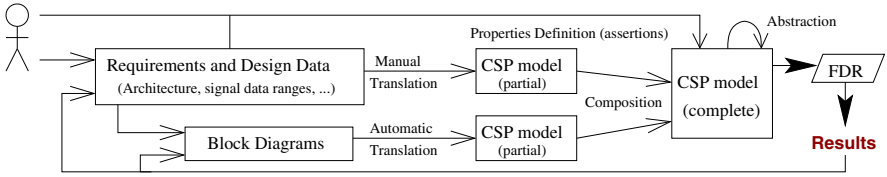


Fig. 2. Analysing Functional Properties

Furthermore, CSP is a high level language useful for describing behavior of components and connect components, allowing us to easily combine the architectural components with the block diagrams; many of the translated requirements identify protocols of communicated values, an interesting area for applying CSP [23]; and the third reason is that systems, components, requirements and properties can be all defined using the same concepts, simplifying the learning curve of people from other fields or engineer disciplines, a fundamental requirement for industrial applications.

2.2 Checking Safety Constraints

Safety constraints can be analysed using probabilistic formal models specified in Prism [15]. The work [11] uses this idea by providing a rule-based mapping from a Simulink diagram, annotated with failure logic⁴, to a Prism model as well as probabilistic temporal logic formulas to analyse the safety aspects of the resulting model.

As we can see in Figure 3, this strategy creates CSL formulas that perform transient or steady-state analysis, using operators of Prism, such as P (transient) and S (steady-state). For instance, with the formula

$$S \leq 10^{-9}[\text{"FailureCondition"}]$$

we calculate steady-state solutions on closed-loop models (that is, models with repairs), composed of a set of discrete states, where each of them is the (operational, faulty or degraded) representation of each component failure mode. The transition occurs over constant rates and represent which state changes are possible and how often they occur. Such an analysis reveals whether at least a single failure mode associated to the system is violated. We can also use transient formulas to obtain the peak instantaneous failure rate over a single period T , enabling us to assess the model sensitivity.

From the results provided by Prism, control engineers can adjust the control diagram by inserting more fault-tolerance features. If the decision needs further assistance, engineers can build fault-trees corresponding to the violated failure conditions indicated by Prism. And when all safety requirements are satisfied, the current system design (including its failure and repair rates) is acceptable.

One of the main advantages of using Prism is that we can detect failure condition violations without building any fault-tree. Another advantage of the work [11] is that it analyses the Prism model in a hidden way, only providing

⁴ Similar to what is described in [21].

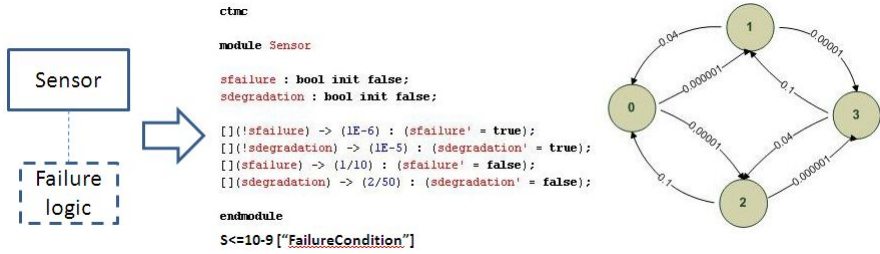


Fig. 3. Basic idea of the safety assessment based on Prism

the result of the analysis to the engineer. A final advantage is that the model generated in Prism covers all failure conditions of the system, allowing to check all possible failure violations with a single model. We believe that creating all fault-trees, to present to the certification authorities, only when our Prism safety analysis cannot find any safety violation is more cost-effective than creating fault-trees all the time as a way to find possible failure violations (the traditional and current safety assessment process).

2.3 Stepwise Development

Stepwise development is an old solution in the formal methods community to the problem of evolving an abstract model into a more concrete model. This development can follow two main alternatives:

1. Proposing a new design and showing that it is a refinement of the old design: this alternative has the advantage of exploiting the developer's creativity. Although checking a refinement can be hard and thus consuming a lot of time effort [23];
2. Creating a new design from a previous design by applying a proved valid transformation [20]: the advantage here is that the refinement is already guaranteed. However, creating a powerful catalog of valid transformations requires a considerable effort [22].

In this work we follow the second alternative. Furthermore, our proposal do not suffer too much from the difficulty of creating an interesting catalog because engineers already have interesting solutions in this direction. For instance, any reliability book [7] has certain patterns to introduce safety aspects in a system. This is a basic knowledge provided to engineers.

To save space we present a very simple refactoring to illustrate the idea. It consists in introducing a replica of a given component and a monitor equipped with an internal switch (to choose between one of the two components). The monitor continuously checks whether the components as well as the switch are working properly. Figure 4 illustrates such a transformation.

In this figure we consider that C represents a generic source component, C' its duplicate and *Monitor* the component responsible for managing the best of

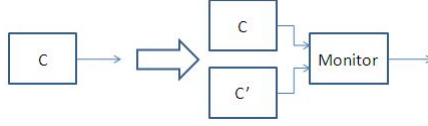


Fig. 4. A simple refactoring

the two available sources. From the options introduced in Section 2, we can see such a transformation is not basic. Thus we call it a refactoring (combination of more basic transformations). To apply such a transformation we need to guarantee some conditions (also known as *proviso*). Let $type(C)$ be the category of a component C (for example, a sensor or actuator), $state(C)$ be the current state of C (the value of all its parameters), $in(C, i)$ be the type (allowed values) of the i -th input of C , and $out(C, o)$ be the type of the o -th output of C ⁵. Hence the proviso of this refactoring is:

- $type(C)$ must be equal to $type(C')$, and
- $state(C)$ must be equal to $state(C')$, and
- $in(Monitor, 1)$ must be equal to $out(C)$, and
- $in(Monitor, 2)$ must be equal to $out(C')$, and
- $out(Monitor)$ must be equal to $out(C)$

Note that, in Figure 4, the boxed arrow points from left to right. This indicates the direction we intend to use this transformation⁶.

As we said previously, we consider behavior preserving refactoring. As Simulink has no formal semantics, we follow the same direction of the works [8,14,26]: define a semantics to Simulink based on Mathworks documentation [19]. To show that a refactoring is valid, we can follow two main approaches:

1. We can reuse the material of Section 2.1, by translating both sides of a transformation and then introducing generic elements to create a theorem. Finally we use the theorem prover for CSP [13] to prove that the transformation is a functional preserving transformation⁷;
2. We can also generate code corresponding to both sides of the refactoring and use the work reported in [24] to exhaustively test or use the simulation facility of Simulink to validate this refactoring [25]. A weakness of this solution is that we need to provide some concrete design in order to apply it.

3 A Case Study

In this section we illustrate our strategy using a very simple example to ease understanding all elements of the strategy. We demonstrate our proposed strategy (the functional, safety analysis and refactoring application) using a feedback

⁵ When a single input/output is present, the index can be suppressed.

⁶ The reverse order of this refactoring, which provides a behavior preserving abstraction, was used in [14] to analyse a fly-by-wire control system more easily using FDR.

⁷ Works reported in [14,26,8] can also be used for this same purpose.

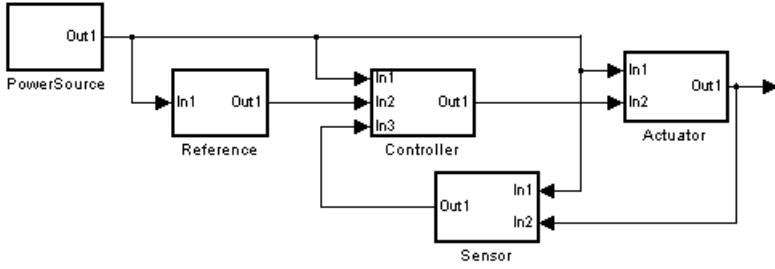


Fig. 5. Functional Model of the Elevator Control System

control system—the Elevator Control System (ECS), which is responsible for commanding an electro-hydraulic actuator, according to the longitudinal orientation desired by the pilot.

In most aircrafts, the pitching movement (the up-and-down motion of the aircraft’s nose) is controlled by elevator surfaces at the rear of the aircraft. These surfaces are driven by electrical-hydraulic actuators controlled according to the pilot intent. Figure 5 shows the main components of the ECS: the reference unit (**Reference**) captures commands from the pilot and it is usually a side-stick (or yoke) providing longitudinal deflections in degrees, the controller (**Controller**) is an Actuator Control Electronics device (ACE) responsible to process the reference signal and the elevator position provided by the sensor component (**Sensor**) to generate the correct commands to the associated power control unit (PCU or **Actuator**). Moreover, this system is powered by a single power source (**PowerSource**).

3.1 Checking the Functionality of the ECS

Recall from Figure 2 that our first effort to formal validation is to apply the process of Section 2.1. To validate the ECS we generate a formal model from the conception (requirements) and design artifacts. This consists in translating all architectural components, except the ACE modelled in Simulink (Figure 2), into a CSP specification. For example, the behavior of the Reference (**SideStick**) component is essentially the behavior of a Linear Variable Differential Transformer (LVDT). It was captured by the process **LVDT**, which inputs the deflection (ranging from -25 to 25 degrees and stated in the set T_LVDT^8 . This set is used in the renaming $[[\dots]]$, changing the generic inputs and outputs to specific ones) of the side-stick, when its core moves (**coreMoved.x**); and it changes its output voltage (**lvdtOutput.x**), representing the stick position.

```

T_LVDT = { I.x | x<-{ -25..25 } }
LVDT = Sensor(I.0, ..., ...)(Sampler__80Hz)(T_LVDT) [[
    in.1.x <- coreMoved.x,
    out.1.x <- lvdtOutput.x | x<-T_LVDT ]]

```

⁸ In this notation, the symbol $<-$ stands for $\in$.

Moreover, the top level **System** process is given by the parallel composition of the processes **SideStick** and **ActuatorSensor** (via interleaving—`|||`—because these do not interact), the **ACE** (which embeds the Simulink model), and the **PCU**. In this late case, these processes must synchronise to exchange data (thus they use the parallel operator `[|...|]`, where the `...` is the synchronisation set).

```
SideStick = LVDT [|...|]
ActuatorSensor = LVDT [|...|]
PCU = ...
ACE_Simulink_Model = ...
ACE = ( ACE_Simulink_Model [| aTick |] Clock ) \ aClock
System = ( SideStick ||| ActuatorSensor )
        [|...|] ACE [|...|] PCU
```

Additionally, engineers may supply the desired properties as CSP specifications as well, since CSP is founded in a strong process refinement theory. One possible desired property is simply checking that the actuator works as expected. That is, for certain input deflections it outputs the corresponding actuator position. Note that if the position does not correspond to the imposed deflection, the process **Observer** behaves as **defect** $\rightarrow$ **STOP**, which signals a defect event and crashes the process (deadlock). Such a situation reveals a functional violation.

```
Observer = coreMoved.1?deflection  $\rightarrow$  ...  $\rightarrow$ 
          lvdtOutput.2?position  $\rightarrow$ 
          if (notAssociated(deflection, position))
            then defect  $\rightarrow$  STOP else Observer
```

After modelling all components and the property in CSP, we performed some simplifications. For example, the pitch rate signal is a real number ranging between -25 and 25 with accuracy of 1.907×10^{-4} . To overcome this problem we applied our earlier data abstraction approach proposed in [9]; it allows one to find the minimum values of the data types (the abstract domains) that are relevant to capture the system's behavior of interest. This revealed that the LVDT signal could assume five (discrete) values: -25, -1, 0, 1 and 25, corresponding to the positions: *lower*, *negative*, *neutral*, *positive* and *higher*. Moreover, components which depend on the LVDT signal were also abstracted to a finite domain.

As our example is very simple such a check is straightforward and the CSP model checker, FDR, reports that this functional property is satisfied.

3.2 Checking the Safety of the ECS

By assuming that our functional design is stable, we add the failure informations. This consists basically in annotating the Simulink model with failure logic as described in more detail in [21]. Thus we can apply the material of Section 2.2 to determine if this design also satisfies the safety constraints or there is at least one failure mode that is violated. Checking safety constraints consists in translating the augmented Simulink model into a Prism model. This translation yields the (partially illustrated) Prism model of Figure 6. To save

```

ctmc
module PowerSource
  powersource_lowpower : bool init false;
  [] (!(powersource_lowpower) & !(SystemFailure)) -> (5E-4) :
    (powersource_lowpower' = true);
  [] (powersource_lowpower & !(SystemFailure)) -> (1/5) :
    (powersource_lowpower' = false);
  [SystemFailure] (SystemFailure) -> (1) :
    (powersource_lowpower' = false);
endmodule

...

formula OmissionSpeed_Actuator_Out1 = actuator_lossofdriver |
  actuator_lossofmotor | actuator_mechanismjamming |
  LowPower_PowerSource_Out1 | OmissionSignal_Component3_Out1;
formula WrongPosition_Actuator_Out1 = ... ;
formula CommissionSpeed_Actuator_Out1 = ... ;

```

Fig. 6. Part of the Prism code of the functional model of the Elevator

space we only present the module **PowerSource** and the formulas used to check this specification. This module captures the chance of the power source to fail, based on its failure modes. For instance, it can fail (**powersource_lowpower**) with a probability of 5×10^{-4} . Note that we use a boolean condition named **SystemFailure** to characterise the fact that the module is updated (changes its states) only if the complete system failure does not occur. The formulas simply capture each failure condition according exactly to the Simulink annotation. For instance, the actuator can produce the output omission speed in port **Out1** (**OmissionSpeed_Actuator_Out1**) if it had a loss of driver, a loss of motor, or a mechanism jamming (The complete model can be found in [1]).

Just for illustrative purposes, we consider probabilities not usually checked in the aerospace industry (Following ARP4754, only catastrophic, hazardous, and

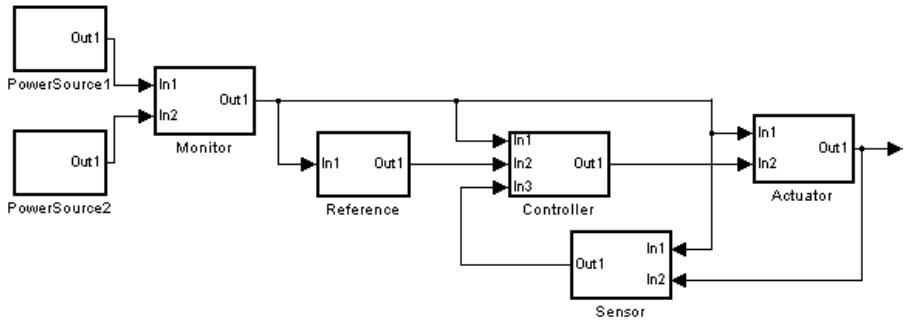


Fig. 7. Safe Model of the Elevator Control System

```

ctmc
module PowerSource1
  powersource1_lowpower : bool init false;
  [] (! (powersource1_lowpower) & !(SystemFailure)) ->
    (5E-4) : (powersource1_lowpower' = true);
  [Monitor_In1_Dependent_Repair] (powersource1_lowpower &
    !(SystemFailure)) -> (1/5) : (powersource1_lowpower' = false);
  [Monitor_In1_Repair] (powersource1_lowpower) -> (1) :
    (powersource1_lowpower' = false);
  [SystemFailure] (SystemFailure) -> (1) :
    (powersource1_lowpower' = false);
endmodule

module PowerSource2 ... endmodule

module Monitor
  monitor_switchFailure : bool init false;
  [] (! (monitor_switchFailure) & !(SystemFailure)) -> (1E-4) :
    (monitor_switchFailure' = true);
  [] (monitor_switchFailure & !(LowPower_PowerSource1_Out1 |
    LowPower_PowerSource2_Out1) & !(SystemFailure)) -> (1/50)
    : (monitor_switchFailure' = false);
  [Monitor_In1_Repair] (!monitor_switchFailure & !(SystemFailure))
    -> (1/5) : (monitor_switchFailure' = monitor_switchFailure);
  [Monitor_In2_Repair] ... ;
  [Monitor_In1_Dependent_Repair] (monitor_switchFailure) -> (1) :
    (monitor_switchFailure' = false);
  [Monitor_In2_Dependent_Repair] ... ;
  [SystemFailure] (SystemFailure) -> (1) : ... ;
endmodule
...

```

Fig. 8. Part of the Prism code for the safe model of the Elevator

major failures are analysed quantitatively) to use Prism to analyse our example. By running the Prism model checker using the CSL formulas

$$\begin{aligned}
 S &\leq 3 \times 10^{-3}[\textit{“OmissionSpeed\_Actuator\_Out1”}] \\
 S &\leq 3 \times 10^{-3}[\textit{“WrongPosition\_Actuator\_Out1”}] \\
 S &\leq 3 \times 10^{-3}[\textit{“CommissionSpeed\_Actuator\_Out1”}]
 \end{aligned}$$

the tool reports a *false* as result to the formula *OmissionSpeed_Actuator_Out1*. Thus, we need to adjust our design to accommodate such a safety constraint. At this point we apply the material introduced in Section 2.3. From engineers decision, the refactoring of Figure 4 is applied in the *PowerSource* component, resulting in the model of Figure 7. After that we update the failure informations, which corresponds to introducing the logic associated to the second power source and the monitor; the resulting Prism model is illustrated in Figure 8. The main difference of the (partially illustrated) Prism code in Figure 6 to the one in Figure 8 is the presence of two *PowerSource* modules and a *Monitor* module. The monitor receives signals from both power sources and makes them available to the other components. In a case of internal failure of

one power source, the monitor is capable of switching to receive the signal from the remaining power source only. This monitoring action can reveal a situation of failure over the monitored components. So the monitor determines the repairs situations of the power sources and this is represented by the set of synchronized repair transitions commands defined in its module (The complete model can be found in [1]).

By running the Prism model checker again, we get no violation this time. Therefore, this design is safe with respect to the safety requirements.

4 Discussion

Formal methods are still difficult to be applied in practice due to two main reasons: (1) when they are fully automatic the model of the system must be finite (for instance, model checking) and (2) when they can analyse infinite state systems they demand great effort from their users (for example, theorem proving). This is the main reason we propose that functional model checking may be applied only in the first design. To give some idea of the involved complexity, model checking required 233456 states and 13986700 transitions in the first model and it would require 245763 states and 14394600 transitions in the second design. In the first case the analysis took 1h 32m 17s and 1h 41m 8s in the second analysis⁹ in a Core 2 duo T5750, running Ubuntu Linux 10.04 LTS 32bits with 1 GB under Virtual Box 3.16. With respect to safety analysis, the main advantage is that the Prism models basically use booleans and thus they are not so complex. To give an idea of the probabilistic model checking complexity, the effort to analyse the first design required 32 states and 62 transitions and the second 126 states and 262 transitions. But in both cases only a few seconds were necessary to analyse them using Prism 3.3 beta 1 in an Intel Core 2 Duo of 1.67 GHz, 2GB RAM, HD 120GB, Windows XP Professional SP3. It is worth noting that Prism supports models of more than 10^7 reachable states.

5 Conclusion

One of the crucial drawbacks of system development is addressed here: the disconnection between functional and safe design. Several papers mention possible directions for doing this, for instance [8,17,21], but none presents strategies and how to apply them in practice. In this paper we show that a functional design can gradually evolve to a safe design by means of stepwise transformations and show the applicability of such a strategy in a simple subsystem.

As future work we intend to develop a full catalog of Simulink based rules. There are two main difficulties: (1) Learn from engineers the common practice of introducing safety aspects in design; (2) Checking whether a given rule really preserves original behavior. To tackle (1) we intend to interact with our

⁹ This little increase in the time analysis is due to FDR hierarchical engine. It does not create a completely new model for each redundancy. It reuses similarity as much as possible. But in general, a model checker will increase the time analysis considerably.

industry partner and (2) by using formal proofs based on the work reported in [14,13,26,5,8] or automatic model based tests and simulation [24,25].

As our strategy is based on model checking, another future work is to check whether avionic models lie in the category of models dealt with fully automatic abstraction strategies such as [9,6]. Also we intend to investigate incremental probabilistic model checking since the difference between two safe designs is relatively small when transformed by refactoring. Such an incremental approach can turn probabilistic model checking more scalable.

Another direction is to consider optimization algorithms, such as simulated annealing, to aid engineers to decide on the most effective refactoring to apply at a certain point in the development process.

Acknowledgements. This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES¹⁰), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08, by CNPq grant 482462/2009-4 and by the Brazilian Space Agency (UNIESPAÇO 2009).

References

1. Prism and CSP models, <http://www.cin.ufpe.br/~acm/ecs/>
2. Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems. Aerospace Recommended Practice ARP4761, SAE International, Warrendale, PA (December 1996)
3. Alexander, R., Herbert, N., Kelly, T.: Deriving Safety Requirements for Autonomous Systems. In: 4th SEAS DTC Technical Conference (2009)
4. Alexander, R.D., Kelly, T.P.: Escaping the non-quantitative trap. In: 27th International System Safety Conference, pp. 69–95 (2009)
5. Bozzano, M., Villaflorita, 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)
6. Damasceno, A., Farias, A., Mota, A.: A Mechanised Strategy for Safe Abstraction of CSP Specifications (Best paper). In: Brazilian Symposium on Formal Methods, vol. 12, pp. 118–133 (2009)
7. Ebeling, C.E.: An Introduction to Reliability and Maintainability Engineering, 2nd Har/Cdr edn. Waveland Pr Inc., Prospect Heights (2009)
8. Kerlund, O.A., et al.: ISAAC, a framework for integrated safety analysis of functional, geometrical and human aspects. In: European Congress on Embedded Real Time Software, ERTS 2006 (2006)
9. Farias, A., Mota, A., Sampaio, A.: Compositional Abstraction of CSP_Z Processes. Journal of the Brazilian Computer Society 14(2) (June 2008)
10. Goldsmith, M.: FDR: User Manual and Tutorial, version 2.77. Formal Systems (Europe) Ltd. (2001)
11. Gomes, A.J.O.: Model based safety analysis using probabilistic model checking. Master's thesis, Federal University of Pernambuco (2010)

¹⁰ <http://www.ines.org.br>

12. Grunske, L., Colvin, R., Winter, K.: Probabilistic Model-Checking Support for FMEA. In: QEST 2007: Proceedings of the Fourth International Conference on Quantitative Evaluation of Systems, pp. 119–128. IEEE Computer Society, Los Alamitos (2007)
13. Isobe, Y., Roggenbach, M.: Csp-prover – a proof tool for the verification of scalable concurrent systems. *Journal of Computer Software, Japan Society for Software Science and Technology (JSSST)* 25, 85–92 (2008)
14. Jesus, J.B.J.: Designing and formal verification of fly-by-wire flight control systems. Master's thesis, Federal University of Pernambuco (2009)
15. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review* 36(4), 40–45 (2009)
16. Laurent, O.: Using formal methods and testability concepts in the avionics systems validation and verification (v&v) process. In: 2008 International Conference on Software Testing, Verification, and Validation, pp. 1–10 (2010)
17. Lisagor, O., Kelly, T.: Incremental safety assessment: Theory and practice. In: 26th International System Safety Conference, published by the System Safety Society (2008)
18. Lisagor, O., McDermid, J., Pumfrey, D.J.: Towards a practicable process for automated safety analysis. In: 24th International System Safety Conference, pp. 596–607 (2006)
19. The MathWorks Inc. Simulink Validation and Verification 2 User's Guide (2008)
20. Morgan, C.: Programming from Specifications, 2nd edn. Prentice Hall International (UK) Ltd., Englewood Cliffs (1994)
21. Papadopoulos, Y., McDermid, J., Sasse, R., Heiner, G.: Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering & System Safety* 71(3), 229–247 (2001)
22. Ramos, R., Sampaio, A., Mota, A.: Conformance notions for the coordination of interaction components. *Science of Computer Programming* 75(5), 350–373 (2010)
23. Roscoe, A.: The Theory and Practice of Concurrency. Prentice Hall PTR, Englewood Cliffs (1997)
24. Soares, G., Gheyi, R., Massoni, T., Cornelio, M., Cavalcanti, D.: Generating unit tests for checking refactoring safety. In: Brazilian Symposium on Programming Languages, pp. 159–172 (2009)
25. Stephenson, Z., McDermid, J., Choy, J.: Using simulation to validate style-specific architectural refactoring patterns. In: SEW 2006: Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop, pp. 123–132. IEEE Computer Society, Los Alamitos (2006)
26. Zeyda, F., Cavalcanti, A.: Mechanised Translation of Control Law Diagrams into Circus. In: Leuschel, M., Wehrheim, H. (eds.) IFM 2009. LNCS, vol. 5423, pp. 151–166. Springer, Heidelberg (2009)

An Approach to Using Non Safety-Assured Programmable Components in Modest Integrity Systems

Peter Bishop^{1,2}, Kostas Tourlas², and Nick Chozos²

¹ Centre for Software Reliability, City University, London
Northampton Square, London, EC1V 0HB, United Kingdom

² Adelard LLP
Northampton Square, London, EC1V 0HB, United Kingdom
{pgb, kt, nc}@adelard.com

Abstract. Programmable components (like personal computers or smart devices) can offer considerable benefits in terms of usability and functionality in a safety-related system. However there is a problem in justifying the use of programmable components if the components have not been safety justified to an appropriate integrity (e.g. to SIL 1 of IEC 61508). This paper outlines an approach (called LowSIL) developed in the UK CINIF nuclear industry research programme to justify the use of non safety-assured programmable components in modest integrity systems. This is a seven step approach that can be applied to new systems from an early design stage, or retrospectively to existing systems. The stages comprise: system characterisation, component suitability assessment, failure analysis, failure mitigation, identification of additional defences, identification of safety evidence requirements, and collation and evaluation of evidence. In the case of personal computers, there is supporting guidance on usage constraints, claim limits on reliability, and advice on “locking down” the component to maximise reliability. The approach is demonstrated for an example system. The approach has been applied successfully to a range of safety-related systems used in the nuclear industry.

Keywords: Programmable components, safety integrity, safety assurance.

1 Introduction

Programmable components like personal computers (PCs) or smart devices can offer considerable benefits in terms of usability and functionality in a safety-related system. However there is a problem in justifying the use of programmable components if they have not been safety justified to an appropriate integrity (e.g. to SIL 1 of IEC 61508 [3]).

To address this issue, the UK Control and Instrumentation Nuclear Industries Forum (CINIF) sponsored a research project (called LowSIL) to assure the safety of modest integrity systems that used “non safety-assured programmable components” (NSPC). The development of the guidance took place in a series of projects:

- A review of approaches actually used when PCs were used in safety-related systems.
- Production of guidance for using PCs in safety-related systems.

- Generalisation of the approach to other types of NSPC.
- Updating the guidance in response to user feedback after the guidance had been applied to actual systems.

In parallel to development of the guidance, we also undertook research, primarily on Microsoft Windows-based PCs, to establish:

- Options for “locking down” Windows to make it more reliable and secure.
- Experimental validation by applying the guidance on a test PC
- Monitoring over extended periods to establish realistic reliability figures for typical applications running under Windows in a locked-down state.

The LowSIL guidance has been applied within the nuclear industry and has been updated to reflect user feedback. In the sections below we present the most recent version of the guidance that has been produced. This guidance is intended for use when:

- Failure of the system can affect nuclear safety, environmental protection or industrial safety, the integrity of plant actuations, safety-related information presented to operators, or the safety integrity of components or calibration data that will be used in the plant at some time in the future.
- The required integrity of the system safety function is at or below SIL 1. Typically no more than 10^{-1} failures per demand or 10^{-4} dangerous failures per hour
- The system contains one or more NSPC. An NSPC is a programmable device such as a PC, a programmable logic controller, or a configurable device such as a smart sensor, and does not have sufficient assurance of its safety integrity.

The guidance can be applied equally to the assessment of new and pre-existing systems. Examples of systems where the guidance has been applied are PC-based monitoring and logging control systems, maintenance support and control of equipment tests.

2 Safety Assurance Context

In order to assure the safety of the modest integrity system we first need to identify the context in which the system operates as illustrated in Fig.1 below.

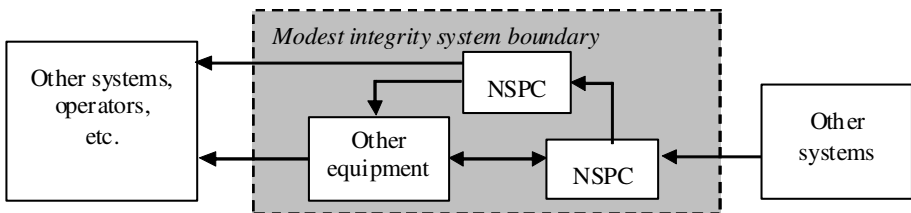


Fig. 1. Safety Assurance Context

In this context we have:

- The modest integrity system, containing one or more NSPC (possibly of different types). Other parts of the modest integrity system can be affected by NSPC failures, but could also contain defences to detect and mitigate failures.
- External systems beyond the modest integrity system boundary (like other C&I equipment, system operators, etc). Failures that propagate beyond the system boundary could affect these external systems. But again there could be defences in the wider system that could mitigate the failures.

3 Structure of the Guidance

The guidance contains the following elements:

- Generic guidance that can be applied to any system containing NSPCs.
- Annexes containing guidance about specific NSPCs.

This structure was chosen because it could be readily extended to include new types of NSPC. Currently the focus has been on Windows-based PCs, but the structure is designed to be readily extended to other NSPC, like smart devices or PLCs.

The specific guidance contains:

- Limitations on use, e.g. Windows PCs are precluded from use for real-time control.
- Guide performance figures, such as reliability, performance, fail-safety and diagnostic coverage.
- Lock-down guidance to enhance reliability and security. Better guide figures for reliability can be used if the component is locked-down.

4 Overview of the Assessment Process

For a new modest integrity system, or replacement system, this assessment process should start as early in the lifecycle as possible, while it is still feasible to determine and influence the modest integrity system design and implementation.

The steps in the LowSIL process and the resultant documentation outputs are shown in Fig. 2. The decision points are shown as diamonds and represent points where the modest integrity system could be rejected as unsuitable.

It can be seen that the process consists of seven discrete steps, which are summarised below:

- Step 1** Characterise the plant context, modest integrity system, embedded NSPC(s) and their types (PC, PLC, etc). Include a clear statement of whether the modest integrity system is a new system, a replacement system or a pre-existing system.
- Step 2** Characterise the requirements placed on each NSPC within the system and assess if the component can feasibly meet them. This characterisation considers:

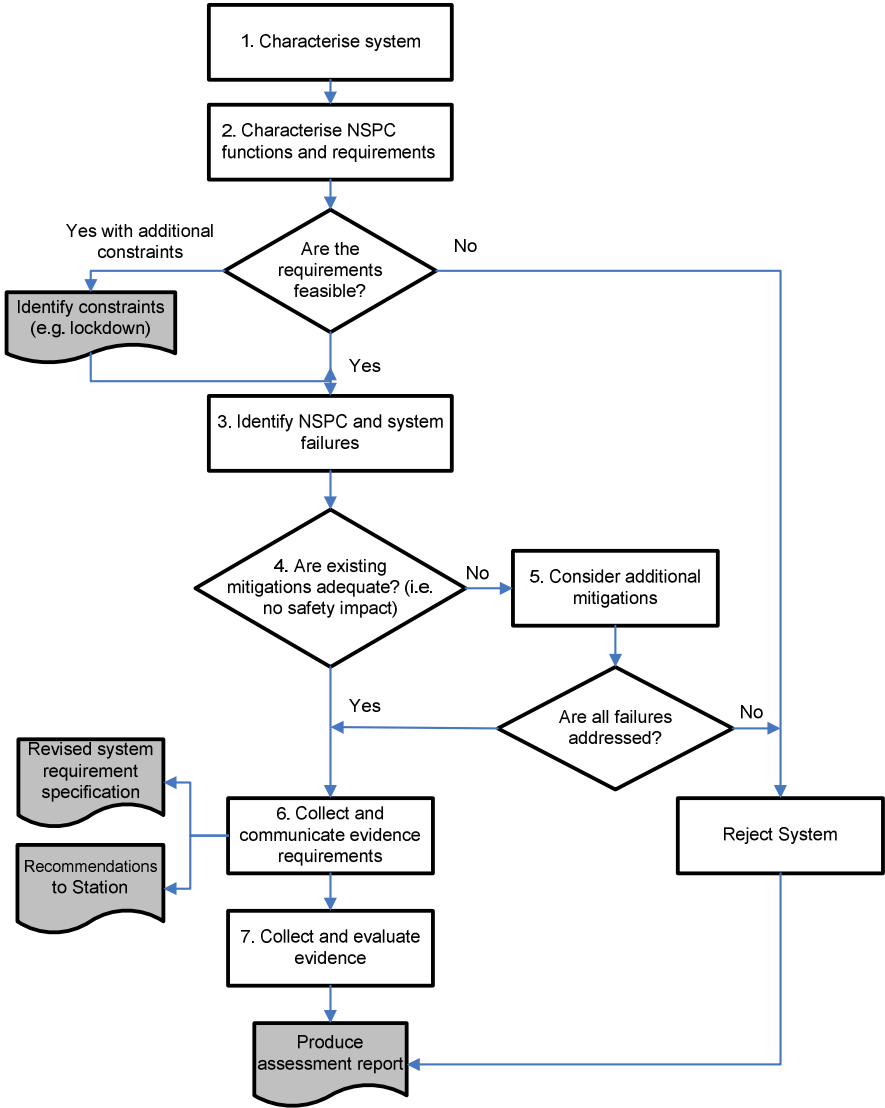


Fig. 2. LowSIL Safety Assessment Process

- The type of function (e.g., advisory, monitoring, control) performed by each NSPC within the system.
- The performance targets for the function, especially reliability, time-response, etc.

In a prospective assessment (of a new/replacement system), compare the performance targets with reasonable limits provided in the relevant NSPC Annex. If the targets exceed these limits, appropriate changes to

the design are required (e.g. “hardening” a PC operating system by a lock-down of its features). These need to be included in the requirements specification for the system. If this is not possible, the system is not considered acceptable.

Similarly, in a retrospective assessment (of a pre-existing system), continued use of the existing system is not acceptable if the performance requirements exceed the guide limits unless there is a statistically valid basis from prior use showing that the system can meet its targets.

Step 3 For each NSPC within the system, identify how it may fail, including performance failures such as slow response. Consider how a failure by the NSPC could lead to a hazard at the system’s boundary. For example, an incorrect command or instruction issued by the system could affect the safety performance of other systems or the wider plant.

Step 4 Assess the safety impact of hazardous NSPC failures, by considering the effectiveness of the mitigation available for these failures.

If there are residual NSPC failures that are not effectively mitigated or controlled, the existing mitigation should be strengthened or new mitigation added. If this is not possible, the system is not considered acceptable.

Step 5 If shown to be necessary by Step 4, consider additional mitigation options and select those that are feasible for the system at hand.

In retrospective assessments this additional mitigation is likely to be procedural. Make specific recommendations about the changes and additions to the existing procedures that will be needed in order to justify the continued use of the system.

In prospective assessments, precedence should be given to technical mitigations. Those that are feasible will result in design change proposals and new system requirements for inclusion in the requirements specification for the system (and possibly in the supply contract).

If at the end of this step there are still failures of the NSPCs that cannot be shown to be effectively mitigated or controlled by the additional measures, the system cannot be assessed as acceptable.

Step 6 Collect the requirements that emerged from the previous steps (new design requirements, changes to procedures, etc). A successful assessment must show that these new requirements have been met, so in this step, determine the additional documentation, verification and validation demonstration and other activities (such as independent assessment of the software) that will be needed as evidence. Ensure that these evidence requirements are included in the Supplier’s contract, communicated to the end-user, etc., as appropriate.

Step 7 Collect and evaluate the evidence available. Produce the assessment report, resulting in a clear recommendation as to whether the system can be accepted or not.

5 Example Application of the Guidance

To illustrate the approach, the assessment steps will be applied to a modest integrity system that forms part of a Unit Maintenance Facility (UMF) in a power plant. The units maintained by the UMF will be used later in plant operation and hence the system is safety-related.

5.1 Step 1: Characterisation of the Modest Integrity System and Environment

The modest integrity system incorporates a PC server, a terminal, a printer and local area network. It is part of the UMF and acts as the Human-Machine Interface (HMI) for the facility’s Safety Logic System (SLS). We will refer to it as the “HMI” system for brevity.

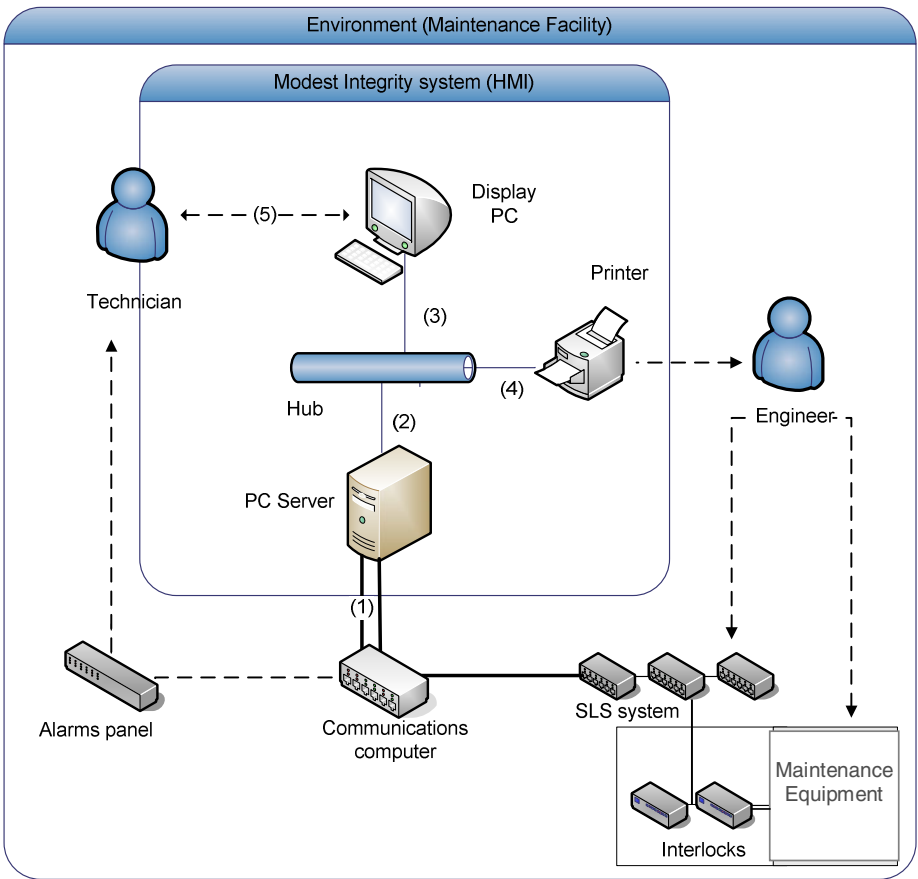


Fig. 3. HMI System Architecture and Environment

The HMI presents information on the plant status (SLS inputs) and state of the SLS outputs. This information is used by the plant operators (technicians) to diagnose plant set-up and configuration problems and also by engineers to identify UMF faults.

The example system and its context are shown in Fig. 3. Note that the links numbered 1 to 5 identify the PC interfaces that are a potential source of hazardous failures.

Operating Context. The system's immediate application environment consists of:

- a high-integrity logic system, known as the SLS, which performs safety interlocking for the UMF
- a communications computer for data acquisition from the SLS
- a hard-wired alarm facia panel driven directly by the SLS
- the UMF Technician, who is the first-line user of the HMI system to diagnose problems in the UMF plant interfaces
- the UMF Engineer, who is the person with the role to diagnose faults on the plant.

The system's broader engineering context is the rest of the UMF system which is responsible for assuring safety (via the use of interlocks) of the maintenance equipment. The HMI does not have any active safety role. However, the UMF technician, who acts partly on the information presented to him by the HMI system, can change the plant set-up and configuration as a result. His actions can therefore affect the state and serviceability of the SLS and of the UMF more generally.

The boundary of the HMI system therefore includes not only the technical elements above but also the technician who is shown straddling the system's boundary.

The Modest Integrity System. The main functions of the HMI are listed below:

- monitor and log time-stamped data from the SLS
- present plant status information, including external panel alarms, gathered by the SLS to diagnose the plant sensors
- present faults reported by the SLS itself, e.g. when a card within that system malfunctions
- produce hardcopy reports of the logged data to use in analysis by the maintenance engineer for confirming the correct operation of plant sensors

The HMI design comprises:

- A PC-based server, connected to the communications computer via dual-redundant RS422 links.
- A display terminal, also a PC, for displaying the data produced by the server to the UMF maintenance technician.
- A printer, for making hardcopies of the logged data.
- An Ethernet network and a hub.

The PC server uses the Microsoft Windows NT operating system. The display PC runs Windows CE 3.0. Application software running on the PC server and the client implement the monitoring and display functions.

NSPCs in the System. As can be seen in Fig. 3, the proposed design for the HMI system makes use of several NSPCs. These non-assured components are:

- the server PC (runs the main application software)
- the display PC (runs the main application software)
- the printer (contains firmware and is configurable to an extent)

It is clearly necessary for the assessment to consider the two PC components in detail, since they provide the essential functionality of the system.

5.2. Step 2: NSPC Feasibility Assessment

For the purposes of this illustration we will focus on one of the NSPCs (the PC server). This step assesses whether the NSPC is “fit for purpose” by considering whether:

- the functions performed are within the capability of the NSPC
- the performances demands are within the capability of the NSPC

This assessment can be supported by NSPC Annex guidance for a PC.

NSPC Functional Suitability. The overall functions provided by the PC components within the system are:

- data acquisition from the communications computer
- data display and logging (i.e. data monitoring)

There is Annex guidance on the types of function that can be performed by PCs. The results of the functionality assessment are shown in Table 1.

Table 1. Function assessment

Function type	Annex Guidance	Assessment
Monitoring interface	PC can be used to monitor non-critical data, i.e. system operation can continue without the displayed data.	Compliant The data collected and logged by the server PC is not essential for UMF operation.
Data acquisition, display, plotting or logging	PC can only be used to acquire and display / plot non-critical data, i.e. the system or environment can continue to operate without the acquired data.	Compliant The UMF can operate without the HMI display.

NSPC Performance Assessment. The performance demands are assessed using guide figures from the Annex as shown in Table 2.

Table 2. Performance assessment

Attribute	Target Level	Guide Limit	Assessment
Availability	> 95%	N/A (depends on MTBF)	Feasible. Based on guide figures for hardware and software MTBF and hardware repair and software recovery times.
Reliability	> 100 hours MTBF	1,000 hours for the Windows operating system 10,000 hours for the hardware	Feasible. The limiting factor in this case is the reliability of the application software.
Safe failure fraction	90%	90% if PC supervised (50% if PC unsupervised)	Feasible. Provided checks can be implemented that are 90% effective at alerting the operator to possible corruption in the displayed data.
Time response	< 5 seconds data request service time < 10 seconds data poll rate	1 second best case 10 seconds worst case	Feasible. Assuming lock-down of the PC server operating system.
Security	Medium	Medium	Feasible.
Usability	Medium	High	Feasible.

NSPC Assessment Conclusions. The overall conclusion is that a PC can be used as a server provided that:

- The operating system is locked-down
- The reliability of the application software meets the target.

5.3 Step 3: Failure Identification and Analysis

A failure analysis can be used to identify hazardous failures of the NSPCs. This could be done using a hazard analysis [1, 5] or failure modes and effects analysis [2]. The analysis must consider potential failures on all the relevant interfaces with the NSPCs (see the links numbered 1 to 5 in Fig. 3). The results of the Hazard analysis of the HMI system are shown in Table 3.

We could also consider any known vulnerabilities of the NSPC at this stage. These would be listed in the relevant NSPC Annex. However the “lock-down” guidance can

Table 3. Hazard Analysis Summary

Ref.	NSPC	Hazardous failure
HF1	Display PC, Server PC	Incorrect logic state data is displayed to the technician. • Wrong status data • Missing status data
HF2	Display PC, Server PC	Out-of-date logic state data is displayed to the technician.
HF3	Server PC	Misleading plant history is logged or reported. The history recorded in the logs, or recovered from them, may be misleading in the following ways: 1. Incomplete history because of missing data. 2. Incorrect history because of wrong data. 3. Incorrect history because of stale data.

be viewed as an alternative because a standard set of countermeasures are identified to address vulnerabilities and so enhance the integrity and security of the component.

5.4 Step 4: Safety Impact Assessment

In this step we consider whether the existing mitigations to the hazardous NSPC failures are adequate. The current mitigations are listed in Table 4.

Table 4. Existing Mitigations

Ref	Description of mitigation
M1	(External to the HMI system) The SLS safety hardware has fail/OK status indicators on each card (reducing the risk of the maintenance technician changing the wrong card).
M2	(External to the HMI system) The equipment is tested after repair – if the wrong board is replaced the HMI would still give the same (misleading) state after repair.
M3	(External to the HMI system) There is a separate alarm panel (driven by the safety SLS equipment) so the control engineer is not relying on the HMI being the primary information source.
M4	(Internal) Display PC detects failure of server to respond within a set timeout period and the display is “greyed-out” to indicate loss of communications with server.

These defences are assessed for adequacy against the identified hazardous failures shown in Table 5.

As the defences are only considered to be partially effective, we need to consider additional mitigation options, specifically enhancements to:

- Assist the detection of data corruption or incorrect processing by the software in the display PC or server PC, or require specific activities to be carried out by the Supplier to demonstrate that the software has the requisite integrity.
- Improve the detection of software hangs.

The options for additional mitigations are addressed in Step 5.

Table 5. Adequacy of Existing Mitigations

Ref	NSPC	Failure	Mitigations	Adequate?
HF1	Display PC, Sever PC	Incorrect display.	M1: SLS show plant status M2: Test after repair	Yes.
HF2	Display PC, Server PC	Out-of-date display	M4: Timeout on loss of communications	Partially. Not effective if the display PC software itself hangs.
HF3	Server PC	Misleading plant history	M3: Alarm data display on separate SLS panel	Partially. Only provides a snapshot of the current status of the plant. The diagnosis requires an accurate history.

5.5 Step 5: Identify Additional Mitigations

The guidance has a checklist of possible mitigations as shown in Table 6. Their applicability has to be interpreted in the context of the particular system design.

Table 6. Mitigation Options

Technical	Procedural
Partitioning	Periodic proof tests
End-to-end integrity checks	Regression tests
External safety checks/interlocks	Fault reporting procedures
Watchdogs	Change control procedures
Clear user interface	Security and access control procedures
Status indication	Procedures for operating under failure conditions
Redundancy	Staff competence and training
Diversity	

For retrospective system assessment, any additional barriers are likely to be procedural. For new systems, there is more scope for incorporating additional technical barriers at an early stage in the design. For brevity we will focus on the applicability of the potential technical barriers to the HMI system. The results are summarised in Table 7.

Table 7. Review of the Applicability of Technical Options to the HMI

Option	Assessment
Partitioning	N/A The system’s environment is already well partitioned.
End-to-end integrity checks	Feasible. Dummy input signals can be added for checking the correctness of the entire data processing done by the HMI system. Each of these dummy signals displays a set known pattern that can be searched for in the logs and observed on the display.
External safety checks/interlocks	N/A The PCs do not perform a control function.
Watchdogs	Uncertain. More applicable to control computers.
Clear user interface	Already covered. HMI display conforms to human factors guidelines.
Status indication	Feasible. A heartbeat indicator should be displayed on the display PC to show that the screen is being updated and that the data acquisition part of the system is active.
Redundancy (communications)	Not required. Dual communications links are already provided between the HMI and the communications computer.
Diversity	Not practicable for PC based systems.

Based on this assessment, the design is modified to include two additional mitigations (see Table 8):

Table 8. Design Enhancements

Ref.	Recommended additional mitigation	Mitigates
M5	End-to-end checks using additional dummy UMF plant input signals. Reveals failures in data acquisition, processing and display / logging functions of the server PC and display PC.	HF1 (wrong data) HF3 (wrong logs)
M6	Status indication. Addition of a heartbeat or liveness indicator, which would enable the operator to detect whether the display screen is being updated.	HF2 (stale data)

5.6 Step 6: Collect and Communicate Evidence Requirements

In step 6 the evidence requirements from the previous steps are collated and assigned for implementation (typically to the system supplier or the plant operator). In Step 2, a

Table 9. Additional Requirements

Source	Requirement	Performed by
Step 2	Lock down the operating system in the server PC	Supplier
Step 2	Demonstrate reliability of the application software	Supplier
Step 2	Demonstrate application conforms to standards	Supplier
Step 5	M5: End-to-end integrity check. Implement dummy plant input scan software change	Supplier
Step 5	M5: Install dummy plant signals in UMF	Plant operator
Step 5	M5: Test of end-to-end check	Operator + supplier
Step 5	M6: Add status indication “heartbeat” software	Supplier
Step 5	M6: Evidence of correct operation	Operator + supplier

need was identified to demonstrate adequate reliability of the application software. This could be covered by compliance to appropriate standards (e.g., [3, 4]) and comprehensive functional testing. Step 5 identified additional mitigations that need to be implemented. We require evidence that these mitigations have been correctly implemented. The additional requirements are shown in Table 9.

5.7 Step 7: Evaluate the Evidence and Produce a Report

Once the changes have been implemented, the evidence is evaluated to assess whether it satisfactorily meets the evidence requirements. Based on the evidence produced and the analyses performed in the previous steps, an assessment report is produced, containing a clear recommendation as to whether the system can be accepted or not.

6 Concluding Remarks

The LowSIL approach has been developed and updated over a number of years and has been applied to a range of control and instrumentation systems used in nuclear power plants as a means of demonstrating adequate safety assurance when non safety-assured programmable components are used. Future developments are under consideration, primarily in the development of new Annexes for different operating systems, middleware and devices.

Acknowledgments. The authors wish to acknowledge the support of the CINIF research programme that funded the research presented in this paper.

References

1. IEC 61882: Hazard and operability studies (HAZOP studies) – Application guide (2001)
2. IEC 60812: Analysis Techniques for System Reliability – Procedure for Failure Mode Effects Analysis (1985)

3. IEC 61508-3: Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements, 1st edn. (1998)
4. IEC 62138: Nuclear power plants – Instrumentation and control important for safety – Software aspects for computer-based systems performing category B or C functions, 1st edn. (2004)
5. Kletz, T.: HAZOP and HAZAN, Identifying and Assessing Process Industry Hazards, 4th edn. Institution of Chemical Engineers (2006)

Development of High-Integrity Software Product Lines Using Model Transformation

Stuart Hutchesson and John McDermid

Aero Engine Controls, Derby,
UK & University of York, Heslington, York, UK
{stuarth, John.McDermid}@cs.york.ac.uk

Abstract. Academic and commercial approaches to software product line development have concentrated on the rapid instantiation of source code assets to minimise product time to market. Generative programming and model-based software engineering approaches have been suggested as effective ways of achieving this. However, for high-integrity software systems the instantiated product source code has to be accompanied by development process assets that demonstrate and support the product assurance arguments. This paper describes an approach to the model-based development of software product lines that is specifically designed to address the needs of high-integrity software systems. The approach consists of a reference architecture model and component-based development style, supported by model transformations to instantiate the project-specific components and associated development assets.

Keywords: UML, SPARK, M2M, Safety Critical, High Integrity, Software Product Lines, Reference Architectures, Decision Models.

1 Introduction

The development of high-integrity software systems differs from general software development primarily in the requirement to provide a compelling argument that the software is fit for purpose **before** the system is deployed. Such arguments are supported by development evidence, and the form of the evidence is often defined by regulation (government body or industry-defined). In addition, the evidence for a particular product is often scrutinized by a regulatory body prior to the product being accepted into service (a process sometimes termed “approval” or “certification”). Civil aerospace is typical of regulated domains, with specific guidance for the development of software (DO-178B/ED-12B [1]), and regulatory bodies who ensure the airworthiness of the aircraft and their constituent avionics systems.

Evidence to support the product approval is typically gathered via the preparation and execution of development and verification plans that define the development processes, analysis and test activities to be undertaken. Analyses typically include independent peer-review inspection of the development artefacts and the automated static analysis of artefacts to demonstrate the existence or absence of certain properties. Dynamic testing is also used to provide a demonstration that the product as built meets its design and requirements for the test scenarios exercised.

For the evidence to be satisfactory to support the certification process it needs to be both **credible** and **compelling**. To argue the credibility of the evidence, the software developers must be able to demonstrate that it is:

- Directly applicable to the system being deployed
- Complete with respect to the requirements of the software system
- Complete with respect to the implementation of the software system

Applicability of evidence is generally argued through rigorous configuration management, whilst completeness is argued via a combination of traceability data (showing the relationships between requirements, design, implementation and verification artefacts) and coverage analysis (demonstrating that the complete structure of the design/implementation has been exercised during verification.)

Organisations developing these types of systems are constantly challenged to reduce overall lifecycle cost of system development, whilst maintaining their declared product lead times. Time-to-market is not generally a commercial advantage in such industries as the products tend to be sub-systems within a larger overall product (flight control systems in an aircraft, or FADEC¹ systems within a gas turbine engine that is then installed on an aircraft for example.) For this type of development, schedule adherence at minimum development cost is the primary project goal. Product line approaches are very attractive in this respect, potentially enabling the organisation's critical engineering resources to be concentrated on designing and developing components that can be used on multiple system instances and reused to meet lead-times for new projects, cost-effectively.

1.1 Software Product Lines

The primary aim of software product line research has been to enable productivity gains in the commercial software development industry. It was recognised that "software reuse" as a concept was not providing the benefits that should be gained from designing a product once and using it many times [2]. The software product line concept was introduced to describe an approach for the development of classes of software systems as opposed to the opportunistic reuse of previously developed software. Product-line approaches make extensive use of commonality and variability analysis to determine which parts of a product line exist in all product variants and which are optionally included in particular product variants. Once the commonality and variability has been identified, techniques such as generative programming [3] and aspect-oriented development [4] can be used to develop product line assets and realise product line instances.

Most current practical approaches to the implementation of software product lines use source-code oriented techniques for the inclusion and exclusion of variable parts of the product line. Indeed, for many commercial software product lines, product instantiation is achieved via the use of pre-processor directives embedded in the product line source code assets. Product line management and transformation tools that manage variability at a more abstract level are starting to become available [5], but these were not commercially available when the work described in this paper

¹ FADEC –Full Authority Digital Engine Controller.

began. Such tools are designed typically to add the concept of variability to environments and representations, e.g. UML, which do not support it natively. Our approach is designed to utilise such capabilities to realise and support the specific needs of high integrity product-line development, including the generation of design and verification assets for product instances.

One of the major issues with the use of a product line approach on high-integrity system development is the need to provide evidence that is applicable to the specific system instance being approved, whilst still developing as a product line. Including verification assets will be crucial to realising an economic return, as verification often accounts for half the total development costs. This can cause a serious conflict between maximising variability to allow a rich and flexible product range, and minimising variability to allow components to be pre-verified and their design data and verification evidence to be reused unchanged across products. The ultimate consequence of minimising variability is, of course, an old style reuse programme which we have already seen does not provide the required business benefits [2].

This paper describes an approach to the model-based development of high-integrity product lines which includes significant variability mechanisms, but is intended to maximise the ability to pre-verify the constituent components (or at least produce verification evidence for use on product instances at minimum cost), and will produce design data which is directly applicable to the instantiated product instances.

2 Architectural Models

The basis for the work described here was the successful application of a model-based approach to the architectural design and implementation of high-integrity gas turbine control systems. This approach used UML class models to describe the software structure, and employed a model-to-text transformation to generate a SPARK [6] implementation. A SPARK profile was used to extend the UML; this allowed the structure of the SPARK program to be fully described at the model level [7]. The UML modelling environment was used to define the architectural framework and the design details for the hosted components. Automatic report generation was used to produce design artefacts from the UML model which were used as configured design artefacts to support the software system approval (certification) process. This approach was successfully applied to a number of projects [8].

To respond to increasing demand for new products, it was decided to launch a software product line initiative and move the focus of the development process from single-products to the design of a product range. It was decided to take the previously successful architectural design approach and use this as the basis for a product line reference architecture. This had the advantage of enabling existing components to be donated (“harvested”) into the product line with minimum rework/refactoring.

2.1 Reference Architecture Approach

The design of the reference architecture was focussed on the development of a framework suitable for the gas turbine control product line, hosting components which contain variation. A parallel feature modelling exercise was undertaken by the

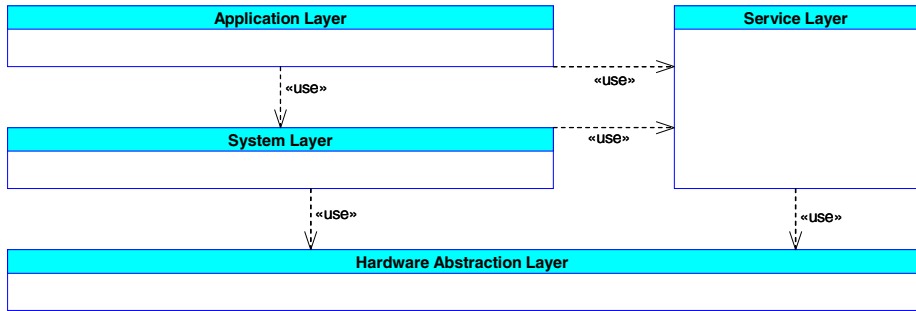


Fig. 1. Reference Architecture – Top Level Layered Structure & Permissions

company’s systems engineering team. The architectural concept was layered to host components at various levels of abstraction; this is shown in Fig. 1.

The reference architecture defines standard abstractions layers in which the product line components are developed and provides a run-time framework supporting the component execution. This includes a standardised scheduling/RTOS approach and a standardised data distribution mechanism to allow multi-processor deployment. The framework and support components are developed and managed by a central architect team, who provide releases of the framework to the component development and product deployment teams.

3 Components and Decisions

3.1 Component Structure

A component in this approach is a functionally cohesive collection of design, specification and implementation information, from which other representations can be generated via transformation. Source code implementations of the components can be generated using model transformations (as described in section 4.) The modelled component uses UML class modelling techniques to provide structure; this is augmented with algorithmic design detail defined using complementary UML notations (for example activity diagrams or state diagrams), or using functional modelling languages and tools such as Matlab/Simulink [9] [10] or SCADE [11]. The problems of interoperability between modelling environments based on a functional/dataflow paradigm and those based on a structural/object paradigm is a significant issue in embedded system design [12]. Currently our approach uses the UML modelling environment as the master, and any design descriptions generated in other environments are imported into the master model as additional annotations (typically) on operations. Currently no syntactic or semantic integration is attempted between the modelling environments, although this will be the subject of further work.

The UML definition of an operation is extended with SPARK as shown in Fig. 2.

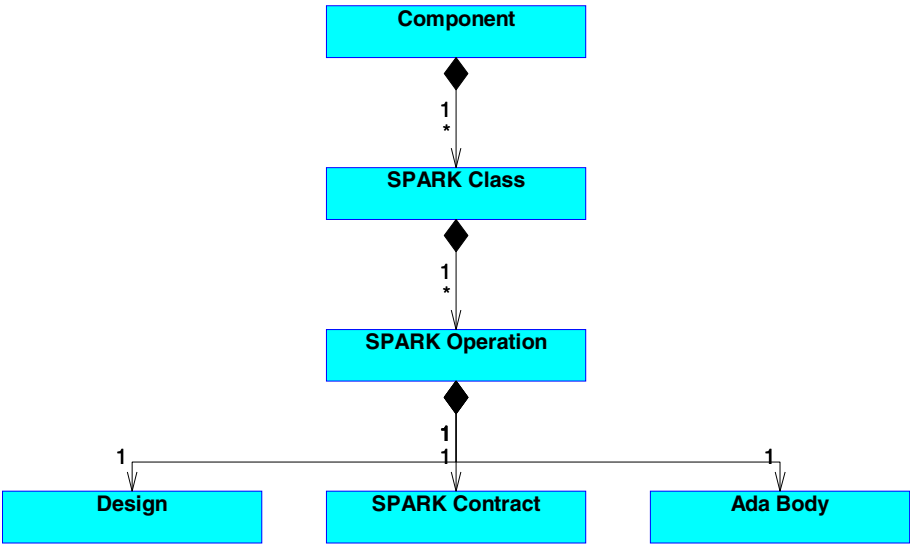


Fig. 2. Model of the information associated with an operation within a component

SPARK Operations introduce the concept of a SPARK Contract [7]. The SPARK mechanism allows a range of operation contract levels to be defined, from data-flow contracts to full pre and post conditions. Our approach uses information flow contracts which define the required input/output relationship of the operation. The implementation of the component can be verified against this information flow contract using the SPARK Examiner tool [6]. The ability to statically verify a component implementation against a contract is fundamental to our approach towards ensuring the correct generation of product line components containing variability. Fig. 2 shows, in addition, that operations contain associated design descriptions and implementations, in the form of SPARK-compliant Ada code bodies.

3.2 Decision Contracts

Feature model-based SPL approaches often maintain a direct relationship between optional features and variation points with the product line assets. Our approach introduces a level of indirection into the variability model via the use of **decisions** [13]. Decisions provide a more granular means of describing variability, and these variability decisions are typically in the context of the implementation rather than the user-oriented view provided by the feature model. This approach can be used to construct components before a complete understanding of the product line scope is available. These component decision points relate directly to **variation points** within the internals of the component. These variation points identify model elements which should be included or removed from the component when given decisions are made. Relationships can be established between features in a feature model and the component decisions, enabling feature-driven selection and traceability to be implemented. The provision of a level of indirection between a feature model and an

implementation via a **decision model** has been demonstrated before [13] and is supported in prototype and commercial software product line environments. However our approach is significantly different in that it makes variability decisions first-class model elements and contains them within the components exhibiting the variability. In this way, variability decisions are prominent in the component designer’s mind at the point of component design, and can be verified alongside the component design and implementation, for example via peer review. In addition, components containing variability can be shared between multiple product lines and the mapping between variation points and variability decisions is maintained.

The approach introduces the concept of **decision contracts**. The component contract is augmented by a model element termed a **decision**. The decision is a public attribute of the component contract. The decision attribute contains a set of possible resolutions to the decision, known as **options**. When a component is deployed, part of the action of deployment is to resolve each decision in the public decision contract. This involves choosing an available option for each of the published decisions.

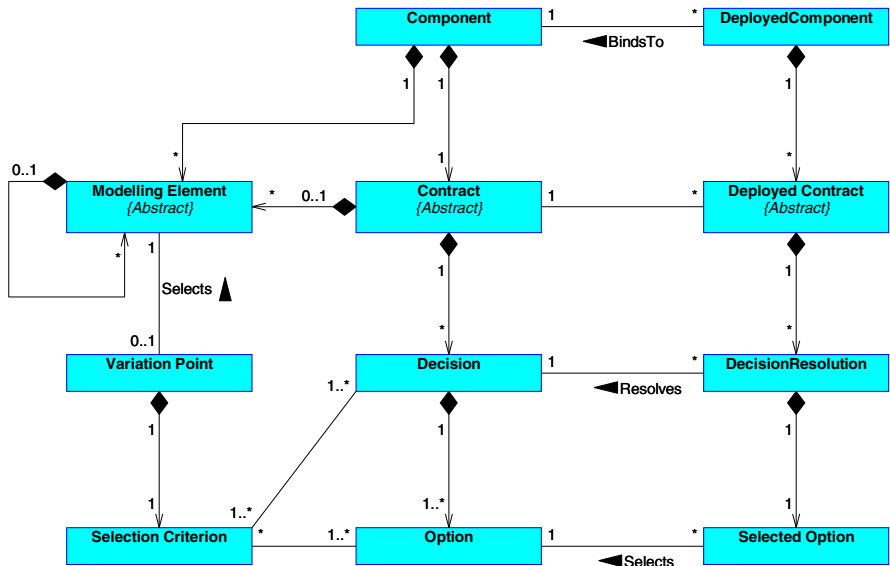


Fig. 3. Component and Decision Contract Meta Model

Component variability is realized via the use of «PL variation point» stereotypes within the component model. Variation point stereotypes can be applied to any relevant meta-model element. Each variation point stereotype contains a “select when” attribute; this attribute holds an expression in terms of component decisions. Evaluation of this expression determines whether the associated meta-model element is included in the product instance model. The model transformation which evaluates these expressions and produces a product instance model is discussed in the next section.

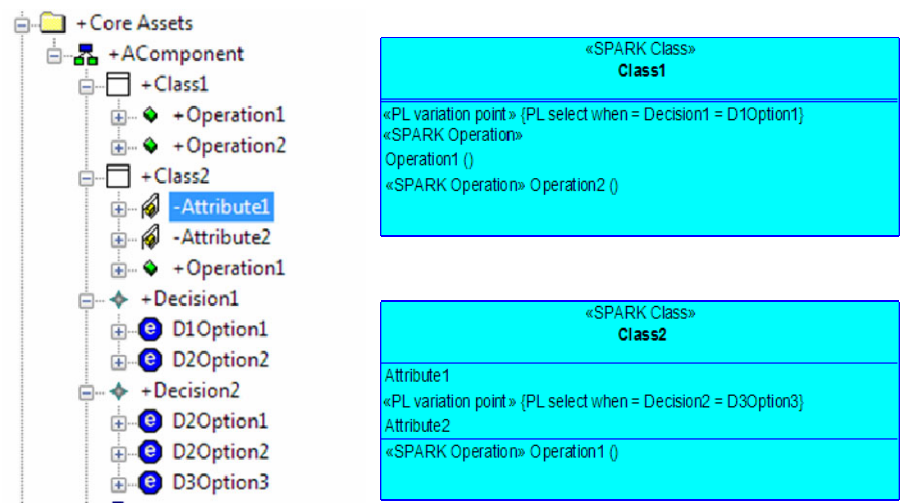


Fig. 4. Example component showing decisions in the component browser and variation points on operations and attributes in the class model

Unfortunately we still have a problem that is common to many model-based product line approaches; to completely specify, design and implement a component we have to utilise model elements which are either non-meta-model compliant, or have no corresponding meta-model. Typically these manifest themselves as attributes containing text fields; the most common example of which is an operation body (e.g. the Ada Body attribute shown in Fig. 2). To address this problem we fall back on a text pre-processing approach common to many SPL implementations. However we maintain the decision contract as the selection mechanism; a simple text mark-up language been defined which uses the same expression language as the «PL variation point» stereotypes to determine the inclusion or exclusion of regions of text in the final product specific model. The longer-term goal of this and related work is to ensure that components can be fully specified designed and implemented using meta-model compliant languages (e.g. specifically designed Domain Specific Languages) to eliminate the need for pre-processing.

4 **Instanting and Model Transformation**

Once the reference architecture and product line components have been developed, product instances can be created. Instantiation of products is achieved by the deployment of the appropriate components in a copy of the reference architecture model and the selection of the appropriate decision options for each component (either directly, or as the result of a higher-level feature model selection). Once the components are deployed and the decision options are resolved, then product-specific assets can be generated using model transformation. Transformations produce product-specific instantiations of all the component assets shown earlier in Fig. 2; that is product specific designs, SPARK contracts and SPARK Ada source code.

4.1 Deploying the Product Line Components

The set of transformations that take the populated deployment (product) model and produces the product source code is shown in Fig. 5 below.

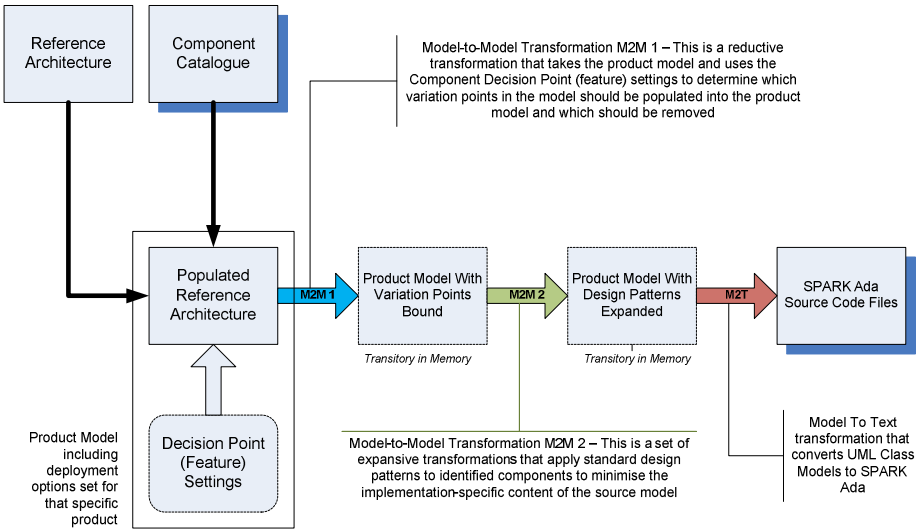


Fig. 5. Product Source Instantiation via Model Transformation – Complete Transformation Chain for Source Code Production

The Model-to-Model transformations take the product line model and transform it to produce a transitory model of the same form as used to describe a single product. From this model the development artefacts can be produced using automatic documentation tools and Model-to-Text transformations. For example, the final Model-to-Text transformation shown in **Fig. 5** is identical to that used to generate SPARK Ada in the single product development process described earlier.

This illustrates one of the fundamental aspects of our approach; where practicable it must be possible to produce a set of development artefacts that support the instantiated product as if it was a single product development; this is essential to meet the certification requirements (cost-effectively). This includes the generation of design and verification artefacts as well as source code.

4.2 Instantiating Products Using Model-to-Model Transformation

The initial M2M transformation takes the deployment model and produces a product specific model which has all the variation points resolved, based upon the selected decision options. This model is then used by downstream transformations to produce the product-specific source code and supporting development artefacts.

The transformation engine used is the ACS/TKD component of the ARTiSAN Studio modelling environment [14]. ACS/TKD provides both M2M and M2T

transformation capabilities, with the declarative transformations described as class models. The instantiation transformation essentially performs the following:

```

For each component included in the deployment model:
  Follow the bind link to the catalogue component;
  For each model element in the catalogue component:
    If it is a variation point then
      If selection expression evaluates True then
        duplicate into deployment model;
      end if;
    Else
      duplicate into deployment model;
    end if;
  end for;
end for;

```

The result of this transformation is a complete product specific model under the deployment model “root” which can be passed to the downstream transformations.

Part of the transformation detail can be seen in Fig. 6. This shows the part of the transformation that duplicates any type definitions that are marked as variation points.

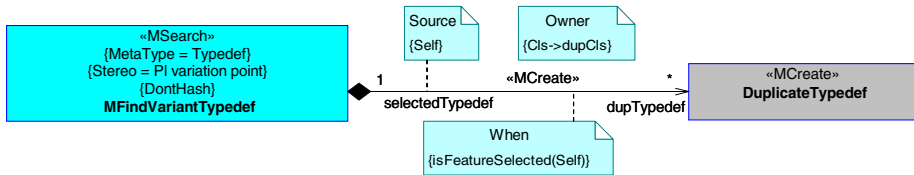


Fig. 6. M2M Transformation Detail – Duplicate Typedef

The class marked as «MSearch» collects model elements that match the search criteria (in this example it will collect meta-model elements of type “typedef” that are stereotyped as «PL variation point»). The «MCreate» class will create a new model element whose form depends upon the annotated «MCreate» association (when the guard specified in the “When” annotation allows). In this case it will duplicate every class in the «MSearch» collection into the deployment model only when the “isFeatureSelected(Self)” guard evaluates True.

As described earlier, the «PL variation point» has an associated “select when” tag which contains an expression in terms of the component **decisions**. The “isFeatureSelected(Self)” guard evaluates that expression using the specific decision options selected for the deployed component.

As discussed previously, a major issue for this approach is that components contain important elements which are not compliant with any defined meta-model. These include many of the operation design elements; these have textual or diagrammatic content that is included from other modelling environments, code inserts generated in IDEs or in text editors and informal content from work processing and diagramming

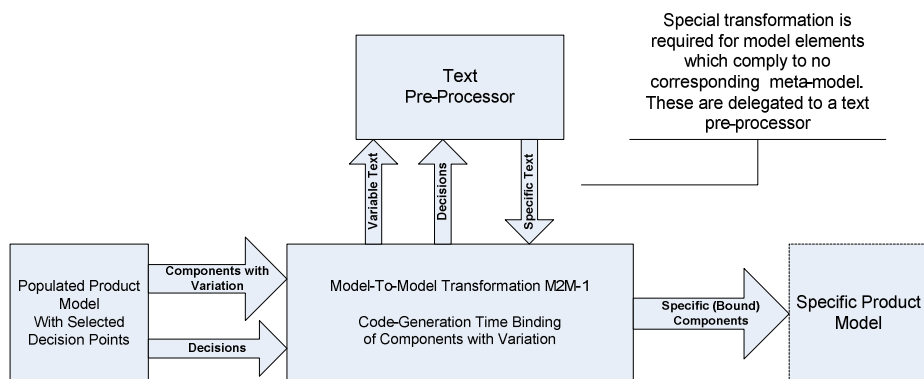


Fig. 7. Delegation of non-metamodel compliant elements to a text pre-processor

tools. However, to successfully deliver a coherent product-specific component, transformations have to be able to manipulate such content to be able to deliver the required variability. Whenever the transformation engine identifies this type of content, the actual transformation is delegated to a pre-processor that deals with the mark-up. This is shown diagrammatically in **Fig. 7**

Any practical approach to product line instantiation using model-transformation must take into account those parts of the product definition that are not meta-model compliant. However, a long-term research aim is to develop and/or integrate a set of modelling environments that are rich enough to capture the full range of specification and design descriptions required for current and future systems.

5 Observations and Evaluation

The component-based design and model transformation approach described in the paper has been in development for about two years, and used on a high-integrity avionics control system development programme for six months. The programme is organised into two separate teams: the product lines (or domain engineering) team is producing product line assets consisting of software requirements and component designs containing explicit variation. The application engineering (or project) team is deploying the product line assets and augmenting them with project specific assets where appropriate. Given the current state of the development programme and given that only a single product instance has been deployed, to date, it is too early to provide a quantitative evaluation of the success of the approach. However, even at this early stage, a number qualitative judgements and observations can be made.

5.1 Project Affordability

The management of electronic component obsolescence, the planning of mid-life upgrades and the in-service replacement of aging and difficult to manufacture equipment is a significant problem in the avionics supply chain. It is the case, however, that equipment upgrades that would make sense from a perspective of

manufacturing (e.g. reliability, yield) are rendered unaffordable by the associated non-recurring engineering costs, frequently due to the software (re-) development and (re-) certification effort. The realisation that product line approaches do not have to just be forward looking, but can also be used to reduce the re-development costs of existing (in service) systems can go a long way to address this serious engineering problem. The requirements of existing systems that will require future upgrade or replacement should be taken into account when specifying product line assets. If this is well-managed and controlled then this can have a significant positive effect on the cost of upgrade and obsolescence programmes.

5.2 Socio-technical Issues

One of the unforeseen consequences of the introduction of a product line approach including variability has been on the attitudes and capabilities of the designers. The development of the approach concentrated on the technical infrastructure and tools to deliver variation into product designs to allow multiple products to be instantiated from a common asset base. There was less effort dedicated to the training and education of the design staff in the “art” of variability, which has resulted in a number of issues in practise. Some of the common issues and misconceptions that have had to be addressed include:

- “Single products can vary at runtime” - One of the most common misconceptions was that in-built modal or state behaviour was actually variability. If the product had different behaviour, say, on the ground and in flight this was initially regarded as variability by some designers. It was surprisingly difficult to ensure all the design team understood that variability **distinguished between different products** and did not represent different states of the same product.
- “The ability to tune the product performance is variability” – This was a subtle but significant point. Closed loop digital control systems usually need to be “tuned” during development to ensure the optimum performance of the plant under control. This is typically done via the inclusion of pseudo-constant data that can be altered without rebuilding the executable program. It was very difficult to get the requirements engineers in particular to understand that the ability to tune and optimise a single product instance was not a variation point (i.e. it does not distinguish between different **product** instances).
- Inclusion of needless variability. It became clear early in the programme that many of the component designers were including variability that was not required to provide the set of products within the scope of the product line. Their argument was that they had been asked to produce a “reusable” component and were therefore catering for all (foreseen and unforeseen) eventualities. However this adds cost and complexity to the product assets with unknown (if any) benefit. The scope of the product line needs to be clear to the development teams, and the component development needs to be closely managed to ensure the variability included is that required to realise the set of products identified.

5.3 Safety and Certification

One of the key success factors for the approach is the ability to generate evidence for certification once, and to reuse it with minimal additional effort. The first system on which this approach is being used has yet to be certified, so there is not yet hard data on the effectiveness of the approach. However the approach enables components to be assessed with all their variability built in, so that the code which results from any valid decision has already been verified – the choices can be thought of as producing code which is within the “assessment basis” hence covered by the evidence collected in component development. By producing the information in such a way that it looks as if it relates to a “single development” (see section 4.1) this evidence can readily be incorporated into the certification data (an accomplishment summary in the context of DO-178B/ED12-B).

There is a need for verification above the component level – for software-software integration and software-hardware integration, for example. This work cannot be done on a product-line basis – at least it is not clear how to do so at present. However the component verification activities typically amount for more than half the verification costs for any one system. The product line approach does not seem to have a material impact on component verification cost, so the verification cost of future developments is likely to reduce to that of the integration and system tests. Also, the component level verification is time-consuming, thus the approach is expected to have a positive impact on project timeframes – another crucial success factor in current projects.

6 Conclusions, Issues and Future Work

This paper describes an approach to using a model-based design approach combined with model transformation to develop a high-integrity embedded product line, and instantiate product instances. The instantiated product instances are supported by development artefacts describing the design and analysis data for that particular product, in a way which makes it simple to use the analysis evidence in support of a certification process which does not recognise product lines. These artefacts are also the result of model transformations. This approach is currently being utilised on high integrity control systems in industry. The degree to which this approach reduces overall development and verification costs is the subject of on-going experimentation and analysis. However, it is expected that, at minimum, it will have a positive impact on component level verification cost – which is a very significant factor in practise.

The value of any pre-verification performed on product-line assets is dependent upon whether the model transformations can be trusted to transform the product assets without error. Our approach uses the SPARK contract to provide a level of assurance that the transformed component matches its transformed contract via static analysis. However this is limited in that the SPARK contract is transformed using the same transformation process as the component source itself; this means that a level of manual verification of the product instantiation is required (with the consequential increase in development cost). However further research work is being undertaken to investigate diverse transformation mechanisms to provide redundancy in the generation of the component source and contracts to address this. A robust means of demonstrating **property preserving transformation** is the primary goal for this

research and is ultimately required to achieve the most economic approach to high-integrity product lines.

Many avionics systems seem natural candidates for product line development as their core requirements are stable, but there is a progressive growth in functionality as user, e.g. airline, requirements evolve. One of the limitations in exploiting the promise of product lines in such high-integrity environments is the difficulty of providing the necessary evidence to support certification in a cost-effective way. We have described an approach, which is currently being used in civil avionics, which addresses this key challenge, and early results show that it should prove cost-effective. We believe the ideas would also prove viable in other, strongly regulated, environments.

One surprising realisation has been that the approach is as likely to be effective in making otherwise unaffordable upgrades to in-service systems possible; this is potentially of significant benefit as electronics obsolescence becomes a greater problem in avionics. It is possible that this benefit may also accrue in other industry sectors, where systems are long-lived, and electronics obsolescence is impacting the ability to maintain in-service systems at appropriate levels of safety and availability.

References

1. DO-178B/ED-12B, Software Considerations in Airborne Systems and Equipment Certification, Radio Technical Commission for Aeronautics / EUROCAE (1992)
2. Bosch, J.: Design and use of software architectures: adopting and evolving a product-line approach. Addison-Wesley, Reading (2000)
3. Czarnecki, K., Eisenecker, U.W.: Generative Programming—Methods, Tools, and Applications, vol. 26, p. 832. Addison Wesley, Boston (2000)
4. Voelter, M., Groher, I.: Product line implementation using aspect-oriented and model-driven software development. In: 11th International Software Product Line Conference (2007)
5. pure::variants integration for IBM Rational Rhapsody [cited 21 March 2010], http://www.pure-systems.com/pure_variants_for_Rhapsody.162.0.html
6. Barnes, J.: High Integrity Software, The SPARK Approach to Safety and Security. Addison-Wesley, London (2003)
7. Amey, P., White, N.: High-Integrity Ada in a UML and C World. In: Llamasí, A., Strohmeier, A. (eds.) Ada-Europe 2004. LNCS, vol. 3063, pp. 225–236. Springer, Heidelberg (2004)
8. Hutchesson, S.: An Architecture-Centric Approach To FADEC Software Development. In: SPARK User Group 2006, Praxis High-Integrity Systems, Bath, UK (2006)
9. Mathworks. Simulink - Simulation and Model-Based Design, <http://www.mathworks.com/products/simulink/> [cited]
10. Yoshimura, K., et al.: Model-Based Design of Product Line Components in the Automotive Domain. In: 12th International Software Product Line Conference (2008)
11. ESTEREL. SCADE Suite, <http://www.esterel-technologies.com/products/scade-suite/> [cited]
12. Muller-Glaser, K.D., et al.: Multiparadigm modeling in embedded systems design. Control Systems Technology, IEEE Transactions on 12(2), 279–292 (2004)
13. Schmid, K., John, I.: A customizable approach to full lifecycle variability management. Science of Computer Programming 53(3), 259–284 (2004)
14. ArtisanSoftware. Artisan Studio, <http://www.artisansoftwaretools.com/products/artisan-studio/> [cited]

On the Safety Implications of E-Governance: Assessing the Hazards of Enterprise Information Architectures in Safety-Critical Applications

Christopher W. Johnson and Stefan Raue

Department of Computing Science, University of Glasgow, Scotland
{johnson, raues}@dcs.gla.ac.uk

Abstract. Governments across Europe and North America have recently reviewed the ways in which they provide both the public and their own departments with access to electronic data. Information service architectures have been proposed as one important component of the new e-Governance visions. These web-based technologies offer huge benefits by defining common interfaces between different information systems, enabling government services to share information with the members of the public and among each other. However, the introduction of e-Governance architectures also creates a number of concerns. Inaccuracies or errors can be propagated well beyond the organizations that are responsible for maintaining the resource. There is also a concern that data, which was originally gathered for general applications, will be integrated into safety-critical systems without the corresponding levels of assurance or data integrity. This paper advocates the creation of a code of practice for the digital dissemination of safety-related information across government departments.

Keywords: e-Governance, Data Integrity, Safety Information, Emergency Planning.

1 Introduction

Relatively little attention has been paid to the safety-related hazards that arise from the integration of government information sources. This is a significant omission given that demographic data and infrastructure information inform the deployment of emergency services as well as the allocation of healthcare resources. Rather than focusing on the safety-related concerns of e-Governance, attention has focused on reducing costs and increasing social inclusion through the provision of networked information services.

1.1 E-Governance and the Focus on Cost Reduction

UNESCO defines e-Governance to be the ‘use of ICT by different actors of the society with the aim to improve their access to information and to build their capacities’¹.

¹ See <http://portal.unesco.org/>, Last accessed June 2010.

The UK government began the sustained development of resources in this area during the mid 1990s. These initiatives were mainly focused on data provision to the public. However, they suffered from a lack of coordination. In consequence, there was a proliferation of web domains that were "disconnected and relatively hard to navigate" [1]. These problems were compounded by political pressure to move more information on-line. In 1997, Prime Minister Tony Blair promised that 25% of government business would be handled electronically by 2002. A key motivation in the program was the perceived need to reduce the costs of central government [2]. It was argued that each year, the Department of Social Security could save £7.7m by moving 2% of its 160 million phone calls to its website. However, this initiative was again marred by a lack of joined-up thinking. For example, some departments included telephone call centers within their interpretation of Blair's "electronic" services. There was little integration between the information provided using conventional sources and the emerging web-based systems. Call centre operators lacked training in the government computer-based applications. Only a dozen of the UK Benefit Agency's 75,000 staff could access their own web site from the computers on their desks. A lack of standardized information exchange technologies as well as missing development standards across government departments led to huge variations in the implementation of these systems [3]. The focus was on reducing costs rather than on accuracy, security or reliability of the proposed government information systems.

1.2 E-Governance and the Focus on Coordination of Local Services

National governments have been keen to ensure that local authorities adopt the use of networked information systems. However, the lack of consistency already seen between the departments of central government is often worse between local government agencies [4]. For example, Swedish information infrastructures were characterized by a diversity that stemmed from the decentralized 'commune experiments' of the 1980s. During the budgetary crises of the 1990s, more and more administrative functions were transferred from state level to the regional administrations. A host of web based services were developed to help members of the public access information about these decentralized services. The sites were developed both by local government and also by local citizens groups. Again, however, the diversity of local needs and local provision created inconsistencies that acted as barriers for the future exchange of information between local government and central agencies [5]. Similar patterns can be seen across Germany. Under the Constitution of 1949, the Federal Government was not allowed to establish regional or local field offices to carry out national policies or legislation. In consequence around 6% of public sector workers were employed at the federal level, 50% were employed by the Länder and 40% by local government. As in Sweden, this created a legacy of sub-regional information services that hinders integration. The examples of Sweden and Germany show how European states have focused on the need to improve the integration of national and regional government information infrastructures rather than considering potential safety implications.

1.3 E-Governance and the Focus on Individual Information Portals

The early proponents of electronic government argued that this technology would revolutionize public access to administrative and financial information [1]. No longer would citizens have to go to government departments during office hours and wait for hours to find that the forms were held in another office. In the future, it would be possible to directly access the required information in a matter of seconds through individual information portals. In contrast, many European states suffered from a proliferation of local and central government web sites. Individuals had to spend increasing amounts of time navigating between web sites for Parents Online; Supporting People Strategies Toolkit; Floor Targets Interactive; Interactive Whiteboards Catalogue; UK Man and Biosphere; Government Decontamination Service; Home Information Pack; Drinking Water Inspectorate; Civil Service Statistics. In the UK this led to a cull of domain names. Fewer than 30 sites were retained from a total of more than 900. The public were redirected through a Directgov portal for most individual information requirements and a business link portal for commercial needs. In France, the *mon.service-public.fr* domain extended the existing Minitel infrastructure. The intention was to provide every citizen with a personal internet portal through which they could pay taxes, register a child for a state school, check the status of car registrations etc. The emphasis on consistency and centralization in other European states can be contrasted with moves towards e-Governance in France. In particular, there was a perceived need to “move away from a traditional mindset of dependency on the central ministries towards one where the field services could exercise greater autonomy in their operational management and be held more accountable for their own actions” [6]. However, as in Sweden, Germany and the UK, the emphasis was on reducing costs and enabling public access. This obscured concerns that the integration of government data services might have implications for public safety.

1.4 E-Governance and the Focus on Social Inclusion

Safety concerns have, however, been raised as part of wider arguments about social inclusion. Individuals may be placed at increased risk if they cannot access electronic information about healthcare services, faulty products, etc. [2]. For example, many government sites still cannot be accessed by those with a visual impairment because they cannot be translated using screen reading software. Other government sites cannot be accessed by linguistic minorities because they are only published in the language of the majority population. The problems of social inclusion also extend to low income groups who often lack the equipment and domestic stability necessary to access on-line information systems. These sections of society often have the greatest need for government information services. The concerns extend well beyond European member states. According to the latest figures published in the Global Information Technology Report 2009-2010 only 4.4% of the Indian population has access to the internet. At the same time, the southern Indian state of Andhra Pradesh has invested some \$5.5m in their SmartGOV initiative. This is intended to put all local government services online. The two main objectives are again to cut ‘red tape’ and reduce costs for the taxpayers.

1.5 E-Governance and the Focus on Security

E-Governance initiatives have been supported by legal innovations, such as the recognition of digital signatures in French law during March 2001. These provisions support the transfer of many financial and administrative services to emerging web-based infrastructures. However, legal changes also reinforce concerns over the security of network transactions. Early denial of service attacks prompted President Clinton to establish a series of public-private partnerships that were designed to prevent an ‘electronic Pearl Harbor’. In 2000, the US Government invested some \$1.75 billion to safeguard the .gov infrastructure. President Obama has continued to increase expenditure in this area through the development of a renewed cyber-security program in 2009 [7]. The focus on cost savings, on regional information dissemination, on social inclusion and on security are instructive because they have arguably obscured the safety threats posed by future plans for the integration of government information services.

2 E-Governance and Concerns over Public Safety

One means of assessing the utility and usability of government information services is to consider the support that they provide for citizens during an emergency. For example, the need to improve government information services for safety critical applications can be illustrated by problems that faced the public and emergency personnel during the UK floods in 2007. Subsequent sections identify potential solutions to these problems through the use of distributed information management between government departments. This is illustrated by a case study in resource allocation for Fire and Rescue Services.

2.1 Problems of Distributed Information Management: UK Floods (2007)

The UK floods of 2007 provide an appropriate case study in the safety concerns associated with e-Governance because many different local and national agencies struggled to provide first responders, planners and individual citizens with information to combat a series of extreme events. The floods were triggered by heavy rainfall that exacerbated high levels of ground water. This combination overwhelmed drains and other forms of flood defense. The UK Meteorological Office recorded 414.1mm of rain across England and Wales; this was more than double the mean expected level of rainfall. The independent report into the subsequent floods, chaired by Sir Michael Pitt [8], argued that these events created “a new level of challenge” for emergency personnel; triggering “a series of emergencies which stretched local resources to the limit” and provided UK civil contingency planners with a “wake-up call”. The floods caused 13 deaths as well as damage to over 40,000 homes and 10,000 businesses. Areas of the UK national rail network were disabled. At the same time, approximately 10,000 motorists were stranded by the closure of part of the M5 motorway.

Confusion, contradiction and inconsistency characterized many aspects of the information interchange that took place between local and national agencies during the floods. The UK Cabinet Office had an almost continual need for information from

local agencies to help form the 'big picture' during these floods. The Cabinet Office is a department of the Government of the United Kingdom responsible for supporting the Prime Minister and Cabinet. It has a coordinating role across different branches of government, in particular via the Cabinet Office Briefing Room (COBR) crisis response committee. This committee guides the government's response to major contingencies. However, their information requests were not always synchronized by regional government so that some key individuals became swamped by requests for information [9]. At the same time, local agencies often did not prioritize these requests from national agencies if they were not perceived to help the people caught up in local flooding. Central government, therefore, found it hard to estimate how many people had been affected by the floods. Initial reports from the Environment Agency suggested that between 3,000 and 4,000 properties were affected. Several days later, Government Offices and local authorities reported that 30,000 houses were flooded. The discrepancy arose because the Environment Agency only counted properties affected by river flooding. It excluded surface water flooding of urban properties even though this was the most significant source of damage.

One reason for the devolution of e-Governance responsibilities during the 1990s from national to local agencies was that they were best equipped to meet the information requirements of the local population. However, local agencies had a 'poor understanding of the location of critical sites; the mapping of their vulnerability to flooding; the consequences of their loss; and dependencies on other critical infrastructure' [8]. There was a need for first responders to have up-to-date flood risk information to coordinate their efforts in helping the public. This data was also important to ensure that emergency personnel did not expose themselves or their vehicles to additional hazards. Local risk assessments created a requirement to integrate national meteorological forecasting, with environmental and urban models that considered critical infrastructures. Responders had to access warnings issued by many other agencies, for example to ensure that they were aware of changes in the level of a water course, or to determine whether or not a power cable was live, or to determine the degree of risk posed by structural damage to a dam. These problems stem from the institutional and organizational barriers to information interchange that are a legacy of the piecemeal manner in which most European and North American governments created their information infrastructures. For example, different UK government agencies use different mapping tools and file formats during the development of Geographical Information Systems. This makes it difficult to share data – for instance about flood levels and the location of 'at risk' members of the public or the location of Fire and Rescue Personnel and the state of local critical infrastructures.

These same problems of information exchange not only affected government agencies, they also had a direct impact on the safety of the general public. During the UK floods, one family saw water pour through the door of their home. They asked the local government agency or council for sandbags, which arrived one week later. This was after their property had sustained significant water damage. When the father called the local Fire and Rescue Service, he was put through to a different county. They were unable to provide any help as he tried to evacuate his family from the rising flood waters. He, therefore, again telephoned the local council and was told to go to a nearby leisure centre. He drove his family at some risk through the flood waters only to find that had been given the wrong information. The leisure centre was

not being used as an evacuation point. One businessman noted that “The websites don’t actually say [this] car park is going to flood – it’s this tributary and that confluence – for people who don’t have a geographical knowledge of rivers, it’s almost impossible to weigh what’s at threat and what’s not” [9]. Individuals were forced to search through dozens of web sites to find information about insurance claims, about whether or not they could drink the water in their mains supply, about the disconnection or restoration of electricity; about the risk of further flooding. These sites were usually overloaded with enquiries and response times were very poor.

Natural disasters such as the 2007 floods provide important insights into the information needs of government agencies and of the general public. They also illustrate the difficulty of identifying whether data is ‘safety-critical’ or not. Information about the capacity and location of supermarket car parks gain importance when it is used to coordinate evacuation activities. Conversely, it becomes very difficult for information providers to identify those members of the public with the greatest needs, as they seek to protect their families, from those individuals who have more mundane requests. A range of government initiatives offer the potential to address these concerns – for instance through the extension of common information architectures.

2.2 Opportunities for Distributed Information Management: Integrated Risk Management Planning

The floods of 2007 illustrate problems in the dissemination of safety-related information between Government Departments. In contrast, the potential benefits of e-Governance can be illustrated by recent attempts to integrate diverse data sources to support the allocation of emergency services. In the UK, much of this work has been driven by a policy decision to use risk assessment to inform strategic planning by the Fire and Rescue Services (FRS). This approach is embedded within the Integrated Risk Management Plans (IRMPs) that document the deployment of FRS resources to fight and prevent fires but also to support the public during natural disasters, including floods, and terrorist attacks. The aim of IRMPs is to improve community safety and make a more effective use of FRS resources by: “reducing the incidence of fires; reducing loss of life in fires and accidents; reducing the number and severity of injuries; safeguarding the environment and protecting the national heritage; and providing communities with value for money”. The development of an IRMP requires data from a range of different government agencies including but not limited to the Department of Communities and Local Government (CLG), the Home Office, and the Office of the Deputy Prime Minister [10, 11 and 12]. For instance, information is required about the population at risk – this implies demographic data from census statistics together with, for instance, information about the occupancy and use of business premises. It is also important to consider whether there are any special hazards within a particular location, including petrochemical storage facilities or manufacturing plants. The allocation of FRS resources must also consider vulnerable locations including hospitals or care homes. These approaches also require information about the likelihood and consequences of future fires, informed by data about previous losses. In addition, risk based planning must draw on government information about the effectiveness of prevention and protection measures including structural fire resistance, means of escape, sprinkler systems, automatic detectors and

alarms, fire doors, ventilation systems etc. Decisions about the deployment of fire resources, vehicles and people, also need to be informed by data on road traffic congestion in order to predict response times.

The complexity of gathering all of this information from various government departments has resulted in the development of software tools to support the Fire and Rescue Services. For example, the Fire Service Emergency Cover (FSEC) tool helps to assess risk, plan response, and model the consequences of different resource allocations for emergency events. Similarly, Figure 1 illustrates a tool to help analyze the risks created by false alarms. This integrates information about previous fires, about the probability of false alarms in a particular region together with the costs of deployment for fire-fighting appliances. The intention is to help FRS planners identify optimum tactics in response to future alarms. Such tools can be used to assess whether or not to send a large number of fire appliances to a location with a known history of previous false alarms. The risk-based approach to planning is important because the answer to such a question depends, in part, upon the people and property that would be threatened by a potential fire.

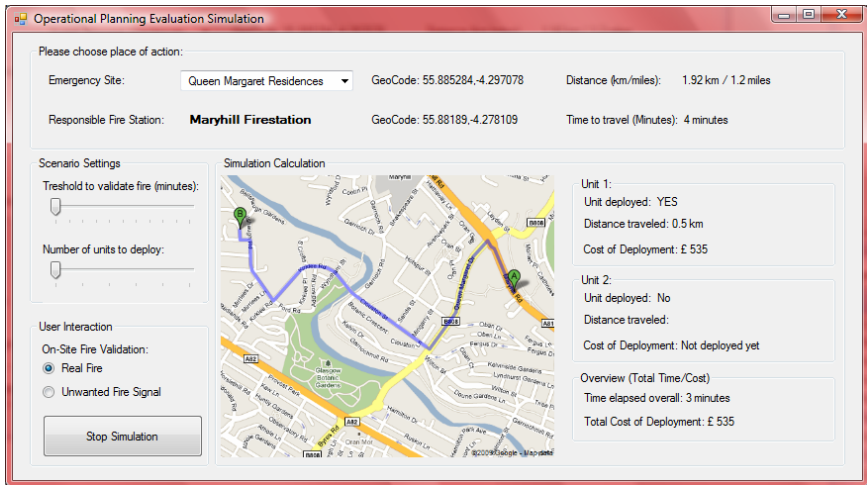


Fig. 1. Possible User-Interface for conducting Operational Planning Evaluations (Raue and Johnson, [11])

These initial steps towards information integration have introduced further research challenges. How can we assess whether the deployment of additional staff and equipment has helped to reduce the number of fatalities/building loss that might otherwise occur? How can we validate the information used to inform our predictions when much of the underlying Government data was never intended to be used in safety-related systems? It is difficult to gather the data required by this new generation of safety-related tools for e-Governance. Each FRS in England collects data in different formats to support their existing systems and processes. This makes it difficult to update the data that is exploited in tools such as that illustrated in Figure 1. The data from each area must first be converted into common formats before the information is introduced into a periodic update.

New copies of the FSEC application are distributed to end users in timescales that are measured in years and not months. In most situations this is not significant; however, it can create problems for instance when industrial units change their operations, when buildings change their occupancy levels or when new housing developments create entirely new demands on the FRS.

3 The Safety of Future Government Information Architectures

Both the U.K. [13] and U.S. governments [14] have recently reviewed their provision of electronic information. Web service architectures have been proposed as an important component within new visions for e-Governance. The W3C define a web service to be a software system that supports ‘interoperable machine-to-machine interaction over a network’. Other systems interact with the web service using a prescribed interface over Simple Object Access Protocol (SOAP) messages based on HTTP with XML serialization. These technologies offer considerable benefits including mechanisms for the integration of government information services. This, in turn, has important implications for safety-related applications, such as those introduced in the previous section.

3.1 The UK Government’s Enterprise Information Architecture (xGEA)

Recent e-Governance initiatives can be illustrated by proposals for the UK Government’s Enterprise Information Architecture (xGEA) [15]. This is intended to provide a reference model that can help to ‘align existing and emerging technical architectures across government’. It was also intended to broaden and deepen the government’s ‘professionalism’ in the provision of information services. The use of this term is significant given the limitations identified with previous public IT procurements [1, 2 and 7]. The xGEA is intended to support three primary objectives:

- ‘To reuse solutions developed for specific issues but which potentially could have a wider value’ [15]. This has clear implications for the manner in which tools such as FSEC have re-used demographic and road traffic data to support safety related decision making in the Fire and Rescue Services.
- ‘To share across public sector organization boundaries to work more efficiently and save resources’ [15]. This again is important given that barriers exist not simply in terms of the hardware and software used across different Government departments but also in terms of the different data formats used, for example by different English FRS. Previous sections have also described similar barriers in many other EU member states hence we would argue that this is a generic aim to be shared across many different countries.
- ‘To be informed of the wider context (other public sector bodies, business and the citizen) in which IT enabled business change is taking place’ [15]. This is a significant aim behind the xGEA initiative because public information systems have tended to lag behind private sector innovations.

These themes of sharing and re-use are critical because they create the opportunities for safety-related decisions to be better informed by the integration of data from across government departments. At the same time, these innovations renew concerns about the integrity and application of this information, when it may not originally have been intended for such uses. An xGEA Reference Model (xGEARM) has been developed in order to support the reuse and integration of information across government, through an agreed set of terms and definitions. The key components of this model are illustrated in Figure 2 [15]. At the time of writing, work is continuing to develop the technical and architectural details of each of the domains mentioned in this diagram. As can be seen, the issues of information assurance and integrity are not explicitly represented at this top level.

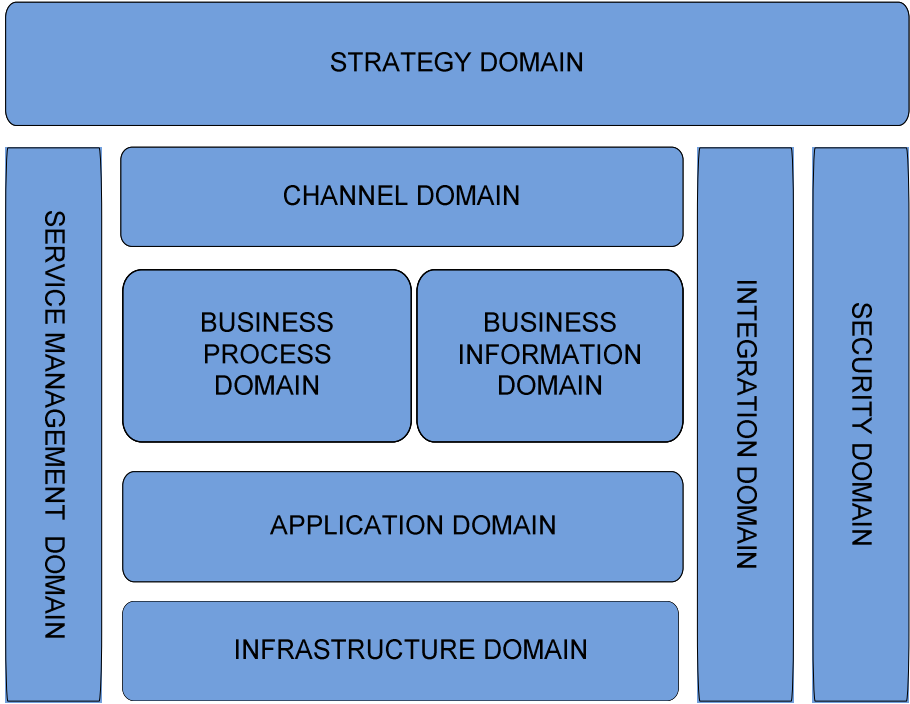


Fig. 2. The UK Government's Enterprise Information Architecture Reference Model (xGEARM)

One of the key concepts in xGEA is a repository that will collect case studies of the ways in which departments can exchange data. More generally, these case studies can also provide examples of the exchange of 'leading practices' or business processes. UK Cabinet Office documentation identifies four initial types of exemplar. These include a Managed Service built using existing staff and technical resources. They also include Solution exemplars. These require additional investments but are based on proven techniques. A third form of 'exemplar' provides patterns that can be followed again. Finally, Lesson Learned provide 'a set of recommendations around a

specific area'. The initial exemplars to be held within the xGEA were selected in terms of their value to government defined in terms of:

- "Cost saving – e.g. investment has already been made and can be reused with little further expense;
- Cost avoidance – e.g. in a future planned program driving down its costs;
- Increase quality – reuse an existing solution/service that has already been tested;
- Time to market – reuse an existing solution/service that has been built
- Increased function to citizen – additional functionality not previously envisaged may be available
- Increase citizen access to government – access to more citizens than first envisaged may be possible" [15]

Safety concerns over data integrity and accuracy are covered within data quality. However, the observation that this will 'reuse an existing solution/service that has already been built' would seem to focus again on the issues of cost that are already listed as the first item in this enumeration of value within the enterprise architecture.

The UK government have also identified a process by which xGEA supports the exchange of information across government. The identification of business needs leads to a sustained search across the repository of previous exemplars to provide a template for exchange. This is then placed within the broader context of the xGEA, for instance by mapping elements of the case study to components of xGEARM in Figure 2. This is important because the exchange of information and processes must, in turn, support further sharing with other departments who might themselves, in turn, benefit from any new application. The final stage is to deliver the service provision within the end user organization. It is, therefore, critical that anyone re-using an exemplar for a safety-related application conduct a formal risk assessment to consider the potential hazards from re-use. These include an over-reliance on data that has not been adequately validated or independently verified. They also include the problems of re-using obsolete information. There are further concerns about whether subsequent users of government information understand the semantics of the data items that are being re-used to inform life critical decisions.

Figure 3 illustrates the top-down functional modeling that has been proposed to identify areas for information sharing across government. The lower box illustrates the corporate functions that support the transformations illustrated by the upper box. Functions A, B and C depend on underlying accounting, procurement, human resource, contract and document management infrastructures. The UK Government's Chief Information Officer argues that "In describing the Business Function model and then comparing it with that from another organization, a number of organizations can be seen to perform a similar function or similar information flow, such as 'Payment Instruction'. This could highlight a potential exemplar that could be used across organizations, which perform similar functions" [13]. However, such an approach requires considerable additional work in order to identify the key constraints that hold over those information flows. In the context of this paper, we might need to ensure that safety-related data was timely, reliable, accurate etc. in addition to the requirement to maintain these functional relationships. It

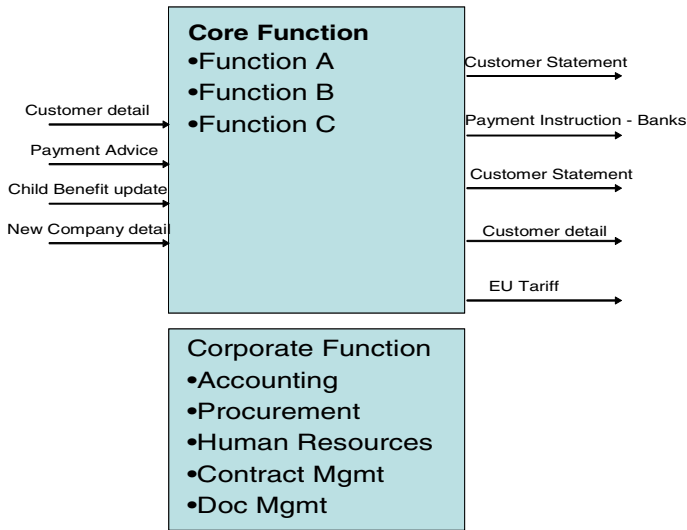


Fig. 3. An Example of a Top-Level Business Process View of Core Functions within xGEA

is important not to view the previous paragraphs as direct criticisms of the xGEA. The intention is to identify generic lessons as many different States extend the integration of electronic data to plan their provision of safety-related services. As we have seen, concerns over data integrity and accuracy are not isolated within the United Kingdom.

3.2 A Proposal for Government Enterprise Integrity Requirements

Many government agencies already operate information assurance guidelines that might inform these proposed architectures for data integration. For example, the UK Statistics Authority, Code of Practice for Official Statistics [16] provides eight principles:

- **Principle 1: Meeting user needs.** The production, management and dissemination of official statistics should meet the requirements of informed decision-making by government, public services, business, researchers and the public.
- **Principle 2: Impartiality and objectivity.** Official statistics, and information about statistical processes, should be managed impartially and objectively.
- **Principle 3: Integrity.** At all stages in the production, management and dissemination of official statistics, the public interest should prevail over organizational, political or personal interests.
- **Principle 4: Sound methods and assured quality.** Statistical methods should be consistent with scientific principles and internationally recognized best practices, and be fully documented. Quality should be monitored and assured taking account of internationally agreed practices.

- **Principle 5: Confidentiality.** Private information about individual persons (including bodies corporate) compiled in the production of official statistics is confidential, and should be used for statistical purposes only.
- **Principle 6: Proportionate burden.** The cost burden on data suppliers should not be excessive and should be assessed relative to the benefits arising from the use of the statistics.
- **Principle 7: Resources.** The resources made available for statistical activities should be sufficient to meet the requirements of this Code and should be used efficiently and effectively.
- **Principle 8: Frankness and accessibility.** Official statistics, accompanied by full and frank commentary, should be readily accessible to all users.

These principles provide a template for the development of a more general set of requirements that might be imposed across the data that is to be provided using government information architectures such as those described in the previous section. In particular, we could build upon principle 4 to identify a number of more detailed requirements for information assurance in areas where data might be used in the aftermath of adverse events, including those identified by the Pitt review of the 2007 floods. Similarly, principle 8 might be developed to enable users to benefit from the 'self descriptive' elements of more recent information architectures. Subsequent users of any information should be warned about the potential risks or limitations of applying that data to inform safety-related decisions [17]. For instance, the following principles might be adopted by many different States as they seek to integrate their e-Governance infrastructures:

- **Safety Information Service Principle 1: Transparency.** It should be possible to identify the original source of data that is derived from other government or external agencies. This is especially important when information may be derived from an information service provided by another department, which in turn is derived from yet another information service. This is a non-trivial issue. For example, information from the Environment Agency about the number of premises affected by a flood may be integrated into FRS planning tools, similar to those described in previous sections. However, the flood data may itself depend upon mapping information and building occupancy data provided by other branches of government. It is difficult, if not impossible, for end users to assess the integrity of this information if they cannot trace these interdependencies.
- **Safety Information Service Principle 2: Applicability.** Ideally, any information provided by a government department should come with the level of assurance that would enable its application to safety-related decisions. However, lack of funding or access constraints can limit the applicability of information services. It may not be possible to conduct detailed site surveys to assess the level of flood protection provided for every water course in the country. Similarly, it may not be possible to accurately measure the traffic delays for every section of road over different times of day. In consequence, the development of national information services often requires the use of extrapolation based on limited sampling techniques. For the end users of this

information, it is critical that they can judge the level of confidence that is associated with the use of these sampling and extrapolation techniques when lives may depend upon a data service.

- **Safety Information Service Principle 3: Recency.** The advent of integrated information services creates complex data dependencies. One of the benefits of this approach is that any updates to information services can be automatically propagated to the different departments who are end users of that service. However, this creates a host of further problems. For instance, the same information request can yield radically different results depending on whether the system was using third-party data provided before or after an update. In many instances, the impact of this update may not be visible to the user unless they understand the many complex ways in which information services interact to support decision making tools. For example, updating demographic information will affect fire risk assessments even though no strategic or operational changes have been made. Additional problems can arise if a third party service changes the format or semantics of data etc.
- **Safety Information Service Principle 4: Triangulation.** Safety-related information should be confirmed by reference to more than one data source. This goes beyond data redundancy because triangulation suggests a complementary data source that is independent of a primary information channel. This increases resilience against the problems of sampling bias. It also provides additional warnings should changes be made in the methodologies used to derive data from any individual source, for instance by comparison of the data derived from two independent sources. The implementation of this principle implies additional costs in cross-checking data sources. However, we would argue that for many safety-related decisions it is worth meeting the additional overheads implied by data triangulation.

It is important to stress that this is a partial list. Additional assurance requirements will be needed as we develop more complex information architectures for the reuse of data between different government departments. For example, the UNOOSA and EGNOS projects are delivering a host of location sensitive information services for emergency response that are beyond the scope of this paper [18]. In the meantime, it is critical to enhance the high-level architectures for government information exchange, such as those illustrated in Figures 2 and 3, if they are to support safety-related services.

4 Conclusion and Further Work

Both the U.K. and U.S. governments have recently reviewed the ways in which they provide access to electronic sources of information. Web service architectures have been proposed as an important component within these new visions for e-Governance. This technology offers huge benefits. In particular, it encourages the provision of joined-up information services that have important implications for a range of safety-related applications. The recent Pitt review into the UK floods of 2007, described how additional risks were created for the public because government agencies

did not use compatible Geographical Information Systems. In contrast, web service architectures help to define standardized interfaces between different government information systems. This provides renewed hope of being able to integrate the various resources held by the Meteorological Office, the Environment Agency, the Fire and Rescue Services etc.

The development of novel architectures for e-Governance also creates a number of concerns. It is important to ensure the integrity of data that is shared between many different agencies. Inaccuracies or errors can be propagated well beyond the organizations that are responsible for maintaining the resource. Data, which was originally gathered for general applications, can be integrated into safety-critical applications without the corresponding levels of assurance or data integrity. This paper has described how these issues have arisen during the development of a web service architecture for emergency planning by Fire and Rescue Services. A range of innovative software helps planners to integrate information about demographics, about transportation infrastructures and about fire risks. These tools help to identify the costs and benefits of moving emergency resources, or of allocating greater attention to fire prevention measures.

The concerns identified in our case studies are increasingly important as governments develop service oriented architectures that resolve many of the technical barriers to data integration. We have, therefore, advocated a code of practice for the exchange of government information in safety-related applications. In particular, we introduce the principles of transparency, applicability, recency and triangulation as means of providing the necessary level of information assurance for critical decision making. Transparency deals with the need to identify the source of third party information. Applicability deals with the need to identify caveats and constraints on the use of information services for applications that are very different from those for which they were originally developed. Recency deals with a host of update problems that can arise, for instance, when safety-critical information is subject to radical changes before and after updates on underlying third party data. Triangulation refers to the need to increase confidence in information sources, ideally by cross-referring data from more than one data sources.

The intention here is not to provide an exhaustive list of data assurance principles but to start a dialogue. Further work is also required to determine whether 'data fusion' and information integration will support safety-critical decision making by European governments. There is a danger that end users will be overwhelmed by a mass of additional information that serves more to confuse than to enlighten. Unless we begin to address the assurance of Government information architectures then there is a danger that many critical decisions will be based on partial or biased information that was never intended for use within safety-related applications.

Acknowledgement

The work described in the paper has been supported by the UK Engineering and Physical Sciences Research Council grant G026076/1; Evaluation of Prevention and Protection Activities On Commercial, Public and Heritage Buildings.

References

1. UK National Audit Office, Government on the Web, London, UK (December 1999), http://www.nao.org.uk/publications/9900/government_on_the_web.aspx (Last accessed March 2010)
2. UK National Audit Office, Government on the Internet: Progress in Delivering Information and Services Online, London, UK (July 2007), http://www.nao.org.uk/publications/0607/government_on_the_internet.aspx (Last accessed June 2010)
3. BBC, Government to Close 551 Websites, Thursday (January 11, 2007), http://news.bbc.co.uk/1/hi/uk_politics/6247703.stm (last accessed March 2010)
4. Lind, M., Östberg, O., Johannisson, P.: Acting Out The Swedish E-Government Action Plan - Mind And Mend The Gaps. *International Journal of Public Information Systems* 2, 37–60 (2009)
5. Löfgren, K.: The Governance of E-government. A Governance Perspective on the Swedish E-government Strategy. *Public Policy and Administration* 22(3), 335–352 (2008)
6. Cole, A., Jones, G.: Reshaping the State: Administrative Reform and New Public Management in France. *Governance* 18, 567–588 (2005)
7. United States Government Accountability Office, Information Security: Concerted Response Needed to Resolve Persistent Weaknesses, Testimony Before the Subcommittee on Government Management, Organization, and Procurement, Committee on Oversight and Government Reform, U.S. House of Representatives, GAO-10-536T (March 2010)
8. Pitt Review learning Lessons from the 2007 Floods (Interim report), Cabinet Office, London, UK (December 2007)
9. Johnson, C.W.: Complexity, Structured Chaos and the Importance of Information Management for Mobile Computing in the UK Floods of 2007. In: Löffler, J., Klann, M. (eds.) *Mobile Response*. LNCS, vol. 5424, pp. 1–11. Springer, Heidelberg (2009)
10. Our Fire and Rescue Service, White Paper, Office of the Deputy Prime Minister (2003)
11. Raue, S., Johnson, C.W.: Using Web Service Architectures and Advanced Simulation Tools to Ensure that Cuts in Strategic Funding for Emergency Services Do Not Jeopardize the Safety of Local Communities. Submitted to the Proceedings of the International Systems Safety Society, Minneapolis, USA (in press)
12. He, Y., Grubits, S.: A Risk-based Equivalence Approach to Fire Resistance Design for Buildings. *Journal of Fire Protection Engineering* 20, 5–26 (2010)
13. UK CIO, Enterprise Architecture for UK Government: An overview of the process and deliverables for Release 1 (2010), http://www.cabinetoffice.gov.uk/cio/chief_technology_officer.aspx
14. US CIO, Enterprise Architecture Assessment Framework v3.0: Improving Agency Performance Using Information and Information Technology, Office of Management and Budget, Washington DC (2008), http://www.cio.gov/Documents/OMB_EA_Assessment_Framework_v3-0_Dec_2008.pdf
15. UK Cabinet Office, Enterprise Architecture for UK Government: An overview of the process and deliverables for Release 1, London, UK (October 2009), http://www.cabinetoffice.gov.uk/media/153627/enterprise_architecture_uk.doc (Last accessed March 2010)

16. UK Statistics Authority, Code of Practice for Official Statistics (January 2009), <http://www.statisticsauthority.gov.uk/assessment/code-of-practice/index.html> (last accessed March 2010)
17. McClatchey, R., Kovacs, Z., Estrella, F., Le Goff, J.-M., Varga, L., Zsenei, M.: The Role of Meta-Objects and Self-Description in an Engineering Data Warehouse, ideas. In: 1999 International Database Engineering and Applications Symposium, p. 342 (1999)
18. Johnson, C.W., Atencia Yepez, A.: Safety Cases for Global Navigation Satellite Systems Safety of Life (SoL) Applications. In: Proceedings of the International Association for the Advancement of Space Safety, Huntsville Alabama, NASA/ESA (2010)

The Right Degree of Configurability for Safety-Critical Embedded Software in Variable Message Signs

Thomas Novak and Christoph Stoegerer

SWARCO FUTURIT, Muehlgasse 86, 2380 Perchtoldsdorf, Austria
{novak, stoegerer}.futurit@swarco.com

Abstract. Traffic control and information systems are used in traffic technology for information propagation from a higher order control unit to the traffic participant. Nowadays, the user interface for the traffic participant is provided via freely programmable signs displaying e.g., traffic jam warnings or speed limits. These signs can be switched on or off and fed with arbitrary data corresponding to the present traffic situation. However, signs are manifold in size, functionality and means to communicate with them. For that reason, a certain degree of configurability of the embedded safety-critical software is needed in order to meet project-specific demands.

Configurability has its advantages in offering a certain degree of flexibility, but poses a risk on safety integrity and requires additional efforts for tools and documentation. This paper focuses on configurability of software in the field of Variable Message Signs (VMS). Possible configuration parameters are considered with regard to its importance during the life-cycle of a VMS and its safety impact. Considering pros and cons, finally an idea of an optimum degree of configurability is being given.

Keywords: Embedded software, configurability, traffic control and information.

1 Introduction

Traffic signaling has its roots in the late 1970s when fiber optic signs for speed limit signalization appeared. At that time, traffic signs based on dot matrices were mainly accessed on-site via specialized hardware and digital I/O and were able to display different, but fixed signalization pictograms (cf. Fig. 1).

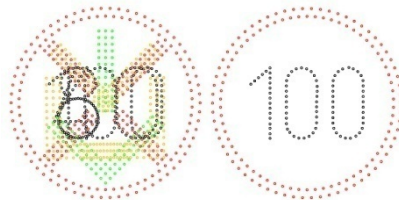


Fig. 1. Traffic sign dot matrix

With the advent of Light Emitting Diodes (LEDs) that make a higher light output possible, reduces power consumption and increases lifetime in contrast to the light sources used in the fiber optic technology, sign manufacturers decided to use LED-technology for signaling issues. At the beginning they were mainly found in the advertising domain where monochrome LED-screens and later on full color technology composing pixels of three LEDs in red, green and blue (RGB) color were used. Shortly after, the concept was introduced to the traffic domain too.

Starting from relatively simple traffic signs (e.g., again for speed limit signalization) with a limited number of aspects (so-called “limited signs”), applications were growing in terms of complexity: “Freely programmable signs” were emerging, providing the possibility to display text and symbols (subsumed by the term aspect) that do not have to be specified in advance. Since then signs displaying a number of various aspects that may be changed or switched off as required, were referred to as Variable Message Signs (VMS) in the context of road applications [1]. Besides VMS, Dynamic Route Information Panels (DRIP) as outlined in [2] or Graphical Route Information Panels (GRIP), as presented in [3], were introduced. Step by step, different combinations and variations of limited and freely programmable traffic signs came up with rising complexity and demands on configurability.

A VMS consists of a graphical part and optionally of a text part below or left/right to the graphical one as shown in [5]. So-called LED modules equipped with LEDs of different colors (white, red, yellow or green) are used for displaying aspects. Those modules are connected serially for larger displays and, in turn, the first module is connected via dedicated cables to a microcontroller. The sign controller runs a firmware that includes functionality to receive commands via various protocols from a Higher Order Control Unit (HOCU), and to process the commands and execute them by activating an aspect and deactivating them, respectively. Data provided from the connected sensors of a VMS is collected and sent to the HOCU. In addition, the controller performs autonomous operations such as proof-tests to check the status of the LEDs on the modules.

Due to the multitude of VMS architectures and protocols used to interface with the VMS from a HOCU, or the various aspects to be shown it is inevitable to make the firmware of the sign controller configurable in order to be able to meet customer’s requirements in a cost-effective way. As a consequence, the paper examines types of software configurations in Section 2. Next, Section 3 deals with VMS related software configuration parameters and stakeholders during the VMS’ life-cycle. Finally, Section 4 outlines a way of assessing the right degree of configurability for a VMS.

2 Types of Configuration

Looking at the configuration of a VMS, it is distinguished between two types of configuration because of the changeability of configuration-parameters as shown in Fig. 2.

In the static configuration, parameters are compiled into software. I.e., those parameters are not subject to be changed after building the executable binary. The dynamic configuration on the other hand can be changed without rebuilding the

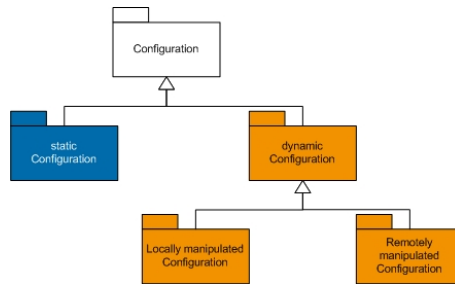


Fig. 2. Configuration types

executable. I.e., parameters might be manipulated either by changing the configuration locally (e.g., by using a service-interface, by changing configuration files) or remotely (e.g., via the user-protocol).

Although dynamic configuration looks much more flexible than static configuration or allows for reusability of large software blocks [4], there are also other arguments to be considered.

In the field of safety, configuration of software is always considered to be failure-prone and might have hazardous effects. Therefore, an “as less as possible”-degree of configurability is preferred. However, dynamic configuration in safety-related automation/process control systems, or also in safety-related traffic management systems is unavoidable (e.g., setting of addresses) in order to provide a suitable degree of usability. In such systems it is a common practice to use a multi-stage process to set parameters as specified in [7] or implemented in a safety-related automation technology called SafetyLon [6].

Practically, static configuration has its advantages because it might not get corrupted (in contrast consider a corrupted configuration file) and it is not subject to be changed or manipulated. Dynamic configuration on the other hand gives great flexibility after finishing development and testing of software. Project-specific configurations as well as adoptions in-field might be realized without involving experienced R&D personnel. Hence, the degree of configurability has also major impact on the VMS life-cycle as outlined on the next section. The pros and cons of configurability can be summarized as follows:

2.1 Pros

- Large software blocks can be reused.
- Functionality can be adjusted to application-specific needs without changing source-code.
- In case of dynamic configuration, software functionality can be adopted very late in the life-cycle by changing the configuration.

2.2 Cons

- Configuration might be failure-prone, especially in case of dynamic configuration performed by departments involved in the in-field process.

- To avoid configuration-failure during the in-field process, sophisticated tools for dynamic configuration are required.
- Increased effort in training of dedicated stakeholders (e.g. commissioning-department, maintenance-personnel).

3 Configuration during Variable Message Sign Life-Cycle

For properly configuring a VMS complying with the customers' demands, a large number of parameters is needed. For being able to clearly show the relation among stakeholders and the configuration and particular configuration parameters, the configuration parameters are grouped into the following subclasses:

1. *External communication interface*: Parameters of the communications interface to the Higher Order Control Unit (HOCU) like RS-485 (e.g., baud-rate, parity) or Ethernet (e.g., IP Port)
2. *External communication protocol*: Parameters like the type of protocol, timeouts or protocol addresses
3. *Debug interface*: Parameters of the debug interface like RS-232 (e.g., baud-rate, parity)
4. *VMS hardware architecture*: Parameters specifying the logical units (LU's) included in the VMS (e.g., a graphical part, text lines or flashers) and its addressing
5. *LED hardware*: Parameters giving information on the type of LED hardware used for every particular LU (e.g., 8x16 mono color board) and physical connection to the controller
6. *Contents*: Symbols to be shown at the VMS (including symbols to be shown in case of a failure)
7. *Fonts* used to show text at the text line of the VMS.
8. *Autonomous operations*: Parameters for autonomous operations like repetition rate of the LED test, CPU temperature and voltage limits
9. *Identification and geographic information*: Parameters to identify the VMS clearly (e.g., serial number) and to be able to locate the VMS within a traffic management system (e.g., GPS coordinates)

Looking on the life-cycle process of the VMS as shown in Fig. 3, it is differentiated between In-House and In-Field processes whereas In-House processes are part of the VMS-vendors value chain and In-Field processes belong to the systems operator- and maintenance-companies.

3.1 In-House Process

The following departments act as stakeholders to the configuration in the In-House process.

- *R&D department* acts as the process-owner of the "Development"-process. It is the most experienced unit relating to configuration with the most influence on the configuration as R&D decides which parameters are realized as static or dynamic configuration-parameters. Furthermore, all static configuration-parameters are applied by R&D.

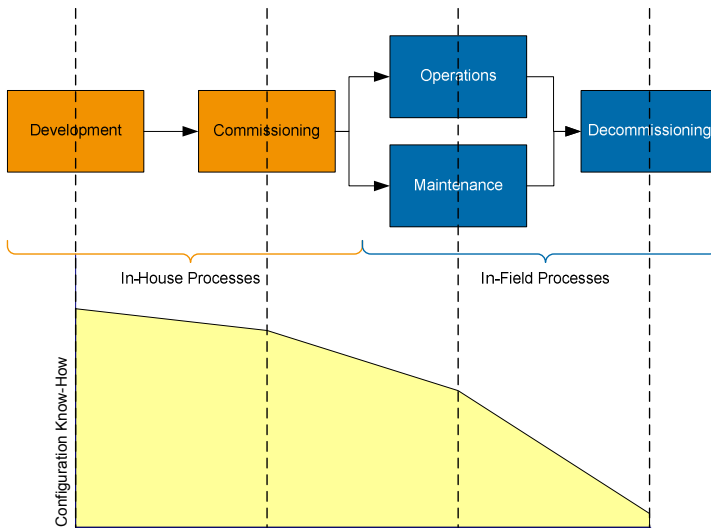


Fig. 3. VMS life-cycle and configuration know-how

- *Commissioning department* acts as the owner to the “Commissioning”-process and partly acts as supporting unit to the “Maintenance”-process. It solely manipulates dynamic configuration parameters locally. Furthermore, a feedback-loop to the R&D-department in case of problems with static configuration might be established.

3.2 In-Field Process

The following departments/companies act as stakeholders to the configuration in the In-Field process:

- *Systems operator* acts as the process-owner of “Operations”-process. It informs service- or maintenance-personnel in case of malfunction or failure. It normally manipulates configuration parameters remotely such as operational parameters like addresses.
- *Maintenance- and service-contractor* company acts as the owner of the “Maintenance”-process. It reacts on failure-notifications from the operator or services facility in defined time-intervals. It normally manipulates configuration parameters remotely such as operational parameters like addresses.
- *Decommissioning* unit acts as the owner of the “Decommissioning”-process. In practice, this is most likely the same unit as the one responsible for the “Maintenance”-process. This is because decommissioning normally only takes place in course of replacement of a broken unit or end of lifetime replacement.

As it is illustrated in Fig. 3, product-specific knowledge decreases throughout the life-cycle process. This is due to the fact that R&D dealing with planning and

developing of the whole product is in touch with the product in much more detail as for instance an operator dealing with the product as a black-box.

With the facts and constraints presented in Section 2 (pros and cons of dynamic configuration) and Section 3 (configuration parameter classes and product know-how throughout the life-cycle) in mind, the next step is to reason about an assessment model that delivers the right degree of configurability of a VMS.

4 Assessment of Configuration Level

In general, configuration of software is unavoidable if on the one hand software shall meet various customer needs and on the other hand shall be maintained with as less as possible resources in a cost-effective way. In detail, what has to be thought about is how to configure the software (static or dynamic as shown in Fig. 2), what classes of configuration parameters are made available and what parties are involved in the configuration of the software (cf. Section 3). Put succinctly, the question is how to find the right degree of configurability.

For that reason a hazard and risk analysis is carried out that is part of standard safety life-cycle models as mentioned in IEC 61508 [10] or MIL-STD-882D [11]. Typical failures and resulting hazards coming from wrong settings of the various classes of configuration parameters are listed and the risk of the hazards is assessed. Finally, the results of the risk assessment together with the facts presented in Section 3 lead to a guideline of how to configure a Variable Message Sign (VMS) properly.

Generally, misconfiguration of parameters can occur for example due to human faults (especially in case of a high stress level [9]), incomplete documentation, faulty specifications, or not or only partially specified project requirements. The effect of faults like the ones mentioned before can be summarized in fault categories and is referred to as configuration fault source:

- The parameter is set out-of-range (e.g., a transfer rate of 225.000 baud is set that is not supported by a converter).
- Not all parameters are set as required (e.g., protocol specific parameters are not set).
- The parameter setting does not fit to the requirements of the project (e.g., a not specified serial protocol is configured).
- Parameter dependencies are ignored (e.g., enabling an autonomous operation like auto-dimming also requires setting a lower and upper limit for the luminosity values).

The configuration fault sources typically result in two failure categories:

1. The software of the VMS and the VMS itself does not work at all.
2. The software of the VMS and VMS itself does not work as expected by the customer's requirements.

4.1 Hazard Analysis

To rate the impact of misconfiguration of software *configuration* parameters on the safety integrity of a VMS, first the hazards have to be identified. Therefore, a Failure

Table 1. FMEA regarding impact of misconfiguration of configuration parameter

	Failure	Mode (Reason)	Effect (Hazard)
1	Communication between HOCU and VMS not possible	Only unidirectional communication (1), wrong protocol selected (2)	Activating or deactivating symbols or text; getting status of sign not possible
2	Other functionality than expected enabled at VMS	Wrong protocol selected where OSI2 layer is equal to the right one (2)	Symbol or text shown unintentionally, blinking of symbols activated
3	Other response than expected received	Wrong protocol selected where OSI2 layer is equal to the right one (2)	Symbol or text activated or deactivated, brightness increased or decreased
4	No debug information available	Wrong baud rate, wrong parity selection (3)	Getting information on internal software details not possible
5	Retrieving status information not possible	Wrong baud rate, wrong parity selection, half duplex communication (3)	Getting information on internal software details not possible
6	Activation of text instead of symbols and vice versa	Wrong assignment of graphical and text part (4)	Symbol or text shown unintentionally
7	Showing symbols or text not possible	Wrong number of modules, symbols or font missing (4), (5), (6), (7)	No text or symbols visible
8	Wrong color of LEDs activated	Module type mismatch (5), wrong format of content (6)	No clear identification of speed limits or text messages by drivers
9	Symbols or text (aspects) corrupted	Symbols or text not defined completely, wrong number of modules (4), (5), (6), (7)	No clear identification of speed limits or text messages by drivers
10	Other symbol or text shown than expected	Numbering of symbols or fonts not matching with specification (6), (7)	Other behavior of drivers than expected
11	Dazzling of sign brightness	Wrong limits of luminosity sensor values (8)	Irritation of drivers on the road
12	Controller in sign shutting down	Wrong temperature limits causing over-temperature (8)	No text or symbols visible
13	LED failure not detected	LED test not activated or less often than required (8)	No clear identification of speed limits or text messages by drivers
14	Response times of protocol not meeting project requirements	LED test or other internal proof test performed in to short time intervals (8)	Symbol or text only partly shown for short time irritating drivers
15	Retrieving information on serial number or geographic position not possible	Serial number or geographic information not set (9)	Identification or location of the VMS not possible

Table 2. Risk analysis and assessment

	Severity class	Probability	Risk classification
1	II	frequent	A (intolerable)
2	I	occasional	A (intolerable)
3	I	occasional	A (intolerable)
4	IV	improbable	D (tolerable)
5	IV	improbable	D (tolerable)
6	I	occasional	A (intolerable)
7	I	occasional	A (intolerable)
8	II	occasional	B (undesirable)
9	I	probable	A (intolerable)
10	I	occasional	A (intolerable)
11	III	remote	C (conditionally tolerable)
12	I	remote	B (undesirable)
13	II	remote	C (conditionally tolerable)
14	III	remote	C (conditionally tolerable)
15	IV	remote	D (tolerable)

Mode and Effect Analysis (FMEA) is exercised (Table 1). A similar more formal approach is suggested in [8] for conceptual software faults. Since the FMEA pursues a bottom-up strategy where the fault sources and the associated failures are analyzed and effects (hazards) are identified, the FMEA is a proper tool for evaluating the impact of configuration fault sources. Thus, the categories of configuration failures and configuration faults are split up into various subcategories. The column “Effect (Hazard)” only includes that hazards that may cause harm to the *user* (e.g., the operator of the sign and/or the driver passing by). The column “Mode (Reason)” lists references to the configuration parameter classes in parentheses presented in Section 3. Within the FMEA it is always assumed that only a *single* failure occurs. A combination of failures is not considered.

4.2 Risk Analysis and Assessment

The second step is to perform a risk analysis including an assessment of the risk coming from every hazard presented in Table 1. The risk analysis is carried out according to the standard MIL-Std. 882 (cf. [11]). It is committed to protecting personnel from accidental death or injury and standardizes a qualitative risk assessment approach. As there is a lack of statistical numbers of the probability of hazards presented in Table 1, the qualitative approach of MIL-Std. 882 is an adequate solution to assess the risk.

Table 2 summarizes the risk analysis and risk assessment. First, the four severity classes give information on the severity of the hazard. Class I is the highest meaning

that the harm resulting from the hazard may be catastrophic (loss of life). Hazards of class II (critical) may cause severe physical injury or permanent partial disability. Class III (marginal) includes hazards that may lead to marginal injuries whereas class IV summarizes hazard that are negligible. Second, the probability ranges from “frequent” i.e., the hazard occurs more than ten times every year, to “probable” (1 to 10 times a year), to incredible where the hazard is considered to be extremely unlikely to occur. Within Table 2 the probability is derived from the occurrence of the failure and its associated fault presented in Table 1. Finally, the risk assessment is shown in column “Risk classification”. The risk categories are according to an informative example in MIL-Std. 882.

Risk class A subsumes all hazards that are intolerable because of severe harm such as loss of life due to a car accident. As shown in Table 1, that class includes hazards where

- it is not possible to intentionally deactivate aspects (i.e., symbols or text), or
- aspects are shown unintentionally, or
- aspects are mixed up, or
- aspects cannot be identified.

Risk class B summarizes all hazards that are undesirable and may lead to incidents such as missing speed limits in case of heavy rain. As shown in Table 1, that class includes hazards where

- no aspects are shown, or
- aspects can hardly be recognized.

Risk class C includes hazards that are conditionally tolerable and may result in service failures like a short irritation of the driver or permanent minor irritation. As shown in Table 1, that class includes hazards where

- the brightness of aspects is too high at night or too low during the day, or
- aspects are incomplete for a short period of time, or
- a LED is not switched on permanently.

Finally, risk class D sums up hazards that are tolerable and may cause degraded availability such as shown in Table 1:

- No information of the internal hardware status of the sign, or
- no information of the internal software status, or
- no identification or location of the VMS.

4.3 Requirements on Software Configuration

The output of the hazard and risk analysis can be linked to the parameter classes (i.e., the reasons of hazards) as shown in the first column of Table 3. It is illustrated that faulty parameter settings especially in classes 1 and 2, but also in classes 4, 5, 6 and 7 may lead to hazards causing severe harm and therefore endanger safety integrity. In contrast, misconfiguration of parameter of classes 8, and in particular 3 and 9 only result in marginal and negligible harm, respectively.

A simple mitigation strategy would be to use only static configuration parameters that are set by skilled staff members of the R&D department. However, reconfiguration requires manipulation of source code to be applied by development personnel with in-depth know-how. In addition, maintenance of the software is resource intensive since all the varying software versions have to be administrated. Finally, changing requirements even at the end of a development are almost inevitable as daily business proves. Consequently, only statically configurable software increases the risk of further delays in the overall-project and is not a suitable solution.

Table 3. Configuration guideline

Parameter class Risk class derived from hazard and risk analysis in parenthesis	In-House		In-Field		
	Development	Commissioning	Operations	Maintenance	Decommissioning
1. External communication interface (A)	✓	✓	✗	✗	✗
2. External communication protocol (A)	✓	✓	✗	✗	✗
3. Debug interface (D)	✓	✓	✓	✓	✗
4. VMS hardware architecture (A, B)	✓	✓	✗	✗	✗
5. LED hardware (A, B)	✓	✗	✗	✗	✗
6. Contents (A, B)	✓	✓	✓	✓	✗
7. Fonts (A, B)	✓	✓	✓	✓	✗
8. Autonomous operations (B, C)	✓	✓	✗	✗	✗
9. Identification and geographic information (D)	✓	✓	✓	✓	✗

The result of the aforementioned facts is to provide possibilities of dynamic configuration to some extent. I.e., the same firmware can be used for various VMS types and the customer needs are satisfied by uploading or setting parameters. A guideline of how to properly and dynamically configure a VMS is shown in Table 3.

Parameters of configuration class 1 and 2 are only manipulated by In-House stakeholders because the know-how about configuration is the highest as illustrated in Fig. 3. The parameters can only be changed locally via a user-friendly command-line interface. And normally they are set by the R&D department because the staff is experienced and familiar with the details of the software. The commissioning department is also authorized to change these parameters (e.g., the address of a VMS). To assure that the parameters are changed correctly, personnel of the commissioning department are trained in a periodic manner by the R&D department.

They receive descent configuration documents and verify the functionality after changing the parameter using test tools that for example verify the response telegram coming from the VMS via the chosen protocol.

According to the risk class of hazards resulting from faulty parameters of configuration class 4, 5, 6 and 7 the same should be valid as written before. That is true for class 4 and 5. Situation is different with class 6 and 7, though. Generally, contents and fonts are provided and verified on a VMS by the R&D department and uploaded locally. However, these two classes are subject to be changed in the field too because during the life-cycle of a VMS (lasting about 10 years) it is likely that further aspects shall be shown due to customer requests.

For that reason, the handling of these classes is twofold. On the one hand the required symbols and contents are handled during the In-House process like the one of class 4 and 5. On the other hand, in case of an alteration during the In-Field process a software upload-tool is provided by the R&D department that verifies the correctness of the remote uploading procedure (e.g., using a 16-bit CRC). Moreover, a test procedure is carried out on the VMS where the new aspect is activated with very little brightness during the day. Finally, the right readout is verified by maintenance personnel and authorized by a national authority.

Furthermore, parameters of class 8 need not to be changed during the In-Field process. So they are handled like parameters of the most critical classes: they are set locally by experienced R&D staff members or trained personnel of the commissioning unit via a user-friendly command-line interface.

Finally, faulty parameters of class 3 and 9 have a negligible negative impact on safety integrity of the VMS software according to the results of the hazard and risk analysis. Consequently, they can be set by members of the In-House or In-Field process without extensive training.

In practice, parameters relating to the debug interface are set locally by R&D personnel to ensure availability of debug information and are typically not changed later in the life-cycle. Parameters of class 9 are often set remotely via an application protocol by the HOCU.

To sum up, the right degree of configurability of a VMS is a trade-off between flexibility and cost-efficiency because of customer requirements and safety integrity. In case of parameter classes 1, 2, 4, and 5 safety integrity is ensured by organizational measures (i.e., only experienced staff members of the In-House processes change the parameters locally). In contrast, parameter classes 6 and 7 must be changed mostly remotely by personnel of the In-Field process also. Thus, a software tool, a test procedure and an organizational measure is required (i.e., verifying the correctness of the aspect by an employee of a maintenance company and a proof of the correctness by a national authority).

5 Conclusion

The paper presented a way of dealing with software configuration parameters of safety-related software for Variable Message Signs (VMS). It started with a discussion of the two different ways of configuring software: statically or dynamically.

Static configuration means to set the parameters in the source code of the VMS firmware by experienced staff members resulting in a reduction of failure probability. In contrast, dynamic configuration allows using a single firmware that provides additional functions to read and process parameters set from a user.

Section 3 presented the VMS related configuration parameter classes, the stakeholders and processes (In-House and In-Field) involved in the VMS life-cycle. It was outlined which stakeholder has the most (R&D: owner of the In-House development process) and the least know-how (Maintenance and Decommissioning: owner of In-Field processes) about the firmware itself. Moreover, the consequences of setting configuration parameters to a defined value (Fig. 3) were stated.

Finally, Section 4 dealt with a hazard analysis using a FMEA where the hazards to users were listed resulting from failures in the configuration parameter classes. Next, a risk analysis and assessment of the hazards was carried out by applying MIL-Std. 882d. The results of the analyses were shown in Table 3. Mostly, the risk resulting from the hazards can be mitigated by changing critical parameters only locally by experienced staff members of the In-Field processes. However, a remote change of some critical parameters like aspects is also required by In-Field process members. Consequently, tool support (e.g., upload tool, use of a CRC) and a verification procedure (e.g., check of the new aspect by a national authority) are implemented.

The result of the paper is a starting point to further improve software safety engineering within SWARCO FUTURIT. It not only presents a clearly structured way of dealing with software parameters regarding their safety criticality for every staff member in the R&D department. It is the base of a common software safety understanding within the company. Additionally, it makes visible for instance what stakeholders have to be trained and what functionality of software tools is required.

References

1. Stainforth, R.: The Application of Variable Message Signs. Widening: Opportunities for New Communications and Lighting, pp. 12/1–12/3 (1991)
2. Schouten, W., et al.: VMS in the Polder. In: Proceedings of the IEE International Conference on Road Transport Information and Control, pp. 222–226 (1998)
3. Alkim, T., et al.: Graphical Route Information on Variable Message Signs. In: Proceedings of the 10th IEE International Conference on Road Transport Information and Control, pp. 32–36 (2000)
4. Ramachandran, M., et al.: Design for large scale software reuse: an industrial case study. In: Proceedings of the Fourth International Conference on Publication Software Reuse, pp. 104–111 (1996)
5. Kulovits, H., Stoegerer, C., Kastner, W.: System Architecture for Variable Message Signs. In: Proceedings of 10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), vol. 2, pp. 903–909 (2005)
6. Novak, T., et al.: Safe Commissioning and Maintenance Process for a Safe System. In: Proceedings of the 7th IEEE International Workshop on Factory Communication Systems, pp. 225–232 (2008)
7. ISO/IEC 13849:2007: Safety of machinery - Safety-related parts of control systems (2007)

8. Denger, C., Trapp, M., Liggesmeyer, P.: SafeSpection – A systematic customization approach for software hazard identification. In: Proceedings of the 27th International Conference on Computer Safety, Reliability and Security (2008)
9. Gruber, T., Althammer, E., Schoitsch, E.: Safety Requirements for a Cooperative Traffic Management System: The Human Interface Perspective. In: Buth, B., Rabe, G., Seyfarth, T. (eds.) SAFECOMP 2009. LNCS, vol. 5775, pp. 159–172. Springer, Heidelberg (2009)
10. International Electrotechnical Commission: IEC 61508 – Functional safety of electric/electronic/programmable electronic safety-related systems – Part 1: General Requirements. IEC (1998)
11. Department of Defense: MIL-STD-882D – Standard practice for system safety. DoD (2000)

INDEXYS*, a Logical Step beyond GENESYS

Andreas Eckel¹, Paul Milbredt², Zaid Al-Ars³, Stefan Schnee⁴, Bart Vermeulen⁵,
György Csertán⁶, Christoph Scheerer⁷, Neeraj Suri⁸, Abdelmajid Khelil⁸,
Gerhard Fohler⁹, Roman Obermaisser¹⁰, and Christian Fidi¹¹

¹ TTTech, ² Audi AG, ³ Delft University of Technology

⁴ EADS Deutschland GmbH, ⁵ NXP Semiconductors Netherlands B.V.

⁶ OptXware Research and Development Ltd., ⁷ Thales Rail Signalling Solutions GesmbH

⁸ Technical University of Darmstadt, ⁹ Technical University of Kaiserslautern,

¹⁰ Vienna University of Technology, ¹¹ TTTech Computertechnik AG

Abstract. Embedded computing systems have become a pervasive aspect in virtually all application domains, such as industrial, mobile communication, transportation and medical. Due to increasing computational capabilities of microcomputers and their decreasing cost, new functionality has been enabled (e.g., driver assistance systems) and cost savings have become possible, e.g., by the replacement of mechanical components by embedded computers.

Conventionally, each application domain tends to develop customized solutions, often re-inventing concepts that are already applied in other domains. It is therefore expedient to invest into a generic embedded system architecture that supports the development of dependable embedded applications in many different application domains, using the same hardware devices and software modules.

INDEXYS targets to pave the way from the European Commission Framework 7 GENESYS Project reference computing architecture approach towards pilot applications in the automotive-, railway- and aerospace industrial domains. INDEXYS will follow-up GENESYS project results and will implement selected industrial-grade services of GENESYS architectural concepts.

The results of laying together GENESYS, INDEXYS and the new ARTEMIS project ACROSS, which will develop multi processor systems on a chip (MPSoC) using GENESYS reference architecture and services, will provide integral cross-domain architecture and platform, design- and verification- tools, middleware and flexible FPGA- or chip- based devices lowering OEM cost of development and production at faster time-to-market of COOPERS.

Keywords: Embedded computing systems, Embedded platform, INDEXYS, GENESYS reference architecture.

1 Introduction

The objective of INDEXYS (INDustrial EXPloitation of the genesYS cross-domain architecture) is to tangibly realize industrial implementations of cross-domain

* INDustrial EXPloitation of the genesYS cross-domain architecture, <http://www.indexys.eu/>

architectural concepts developed in the GENESYS project in three domains: automotive, aerospace and railway, thereby relating to ARTEMIS-JU Industrial Priority: “Reference designs and architectures”, see reference [1]. GENESYS (Generic Embedded System Platform) is developing a cross-domain architecture according to requirements of the ARTEMIS Strategic Research Agenda. The GENESYS architectural style supports a composable, robust, and comprehensible component-based framework with strict separation of computation from message-based communication. So components can be massively reused in differing contexts. In the GENESYS architecture three integration levels of components are distinguished: **chip-level**, where IP cores communicate via a deterministic Network-on-a-Chip (NoC); **device level**, where chips communicate within a device and **system level**, where devices communicate in an open or closed environment. INDEXYS expands the GENESYS approach by implementing and integrating architectural services into prevailing (real-world) platform solutions. A key goal of INDEXYS is legacy integration, for platform providers – by integrating new architectural services into legacy platforms – and for platform users – by supporting legacy applications. INDEXYS addresses robustness w.r.t. design faults and physical faults by diversity and component replication. INDEXYS targets ARTEMIS-JU Sub-Programme 5, see reference [1]: “Computing environments for embedded systems” by developing new concepts for composable component integration, re-usable dependability services, and a cross-domain tool-chain based on OMGs Model Driven Architecture. By fostering development of re-usable, dependable products and associated services, INDEXYS significantly contributes to competitive advantages of European players in the transportation industries.

Proposed in the first call of the ARTEMIS Joint Undertaking, INDEXYS relates to the ARTEMIS SRA industrial priority “Reference Designs and Architectures” and aims at re-using concepts and designs in multiple industrial domains for different types of applications. INDEXYS is conducted by an international consortium of 10 partners, representing SME partners (2), industrial partners (4) and universities (4). The project is partly funded by the ARTEMIS Joint Undertaking and by national governmental funding agencies and the project partners. The effort required for INDEXYS amounts to 667 man months. INDEXYS has started in April 2009 and will be completed in September 2011.

1.1 DECOS, GENESYS, INDEXYS, ACROSS – The Coherent R&D Program Chain

INDEXYS addresses actual instantiations of architectural cross-domain services which are defined through the GENESYS reference architecture template, and over ARTEMIS SRA’s (see reference [2]) requirements and constraints. INDEXYS is further based on results of the DECOS project – a predecessor project of GENESYS. DECOS focused at finding common composable design concepts across multiple application domains. With ARTEMIS as a basis, GENESYS extended the DECOS concepts by developing a cross-domain architectural reference template. INDEXYS builds on the DECOS / GENESYS foundation to realize these concepts by instantiating architectural cross-domain services defined by the GENESYS reference architecture template. INDEXYS architectural service instantiations targets three domains: automotive, aerospace and railway, leading to domain specific upgrades of existing

architectural solutions. Legacy platforms, such as the TAS Control Platform (see reference [3]) in the railway domain or IMA (see reference [4]) in the aerospace domain will serve as the basis for subsequent instantiation of GENESYS' generic platform services.

Complementing and further enhancing the achievements from DECOS, GENESYS and INDEXYS, the ACROSS project has been submitted as a proposal in the ARTEMIS second call in 2009. The project has been selected for funding and is due to start end of Q1/2010. It will develop a multiprocessor system on a chip (MPSoC) providing an on-chip network with flexible composition of components replying to the market's requirement of composable, reliable, embedded networks. In addition, ACROSS will target the development of tailored middleware components and suitable design, development and verification tools to generate a powerful cross-industry platform.

2 R&D Results from Previous Programs Establish INDEXYS Basis

2.1 DECOS Paving the Way for Cross-Domain Architecture Reference Template

As a European Commission Framework 6 Integrated Platform Programme, DECOS (see reference [13]) targeted the development of fundamental technologies facilitating the paradigm shift from federated to integrated design of dependable real-time embedded systems. DECOS provided means for systematic design and development of integrated, electronic sub-systems in embedded applications by (a) cost reduction in electronic hardware, (b) by enhancing dependability by design, (c) by enabling modular certification, (d) special diagnosis and maintenance support, (e) by offering individual intellectual property protection. Applications were deployed in automotive, aerospace and industrial control domains.

The DECOS results offered the first proof of concept for the GENESYS basic idea of developing a cross-domain architecture reference template. The DECOS results fundamentally enhanced the means for design, analysis and tools for integrated, dependable, real time embedded systems (see also Fig. 1).

2.2 Introduction on the GENESYS Project

Conventionally, each application domain tends to develop customized solutions, often re-inventing concepts that are already applied in other domains. It is therefore expedient to invest into a generic embedded system architecture that supports the development of dependable embedded applications in many different application domains, using the same hardware devices and software modules. Furthermore, increasing capabilities of microcomputers and decreasing cost foster a change from mechanically to electronically controlled functions in all industrial domains.

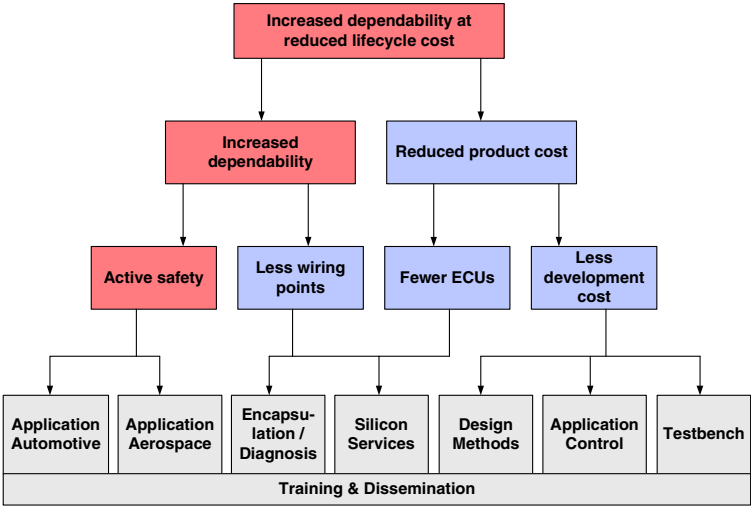


Fig. 1. DECOS Results

Legend:

ECU Electronic Control Unit

Detailed information is available in reference [13].

The world of embedded systems, in particular when respecting various industrial domains with its different application requirements, is broad and diverse. The technological situation is strongly fragmented and the expectations of the increasing number of users are permanently rising.

In order to take advantage of the economy of scale of the semiconductor industry, designing and developing a cross domain architecture reference template for embedded systems to be used in various industrial domains can easily be justified (this chapter frequently makes reference to [5]) due to faster time to market and reduced design and implementation cost resulting from re-using once developed approaches.

H. Kopetz and R. Obermaisser summarise GENESYS as follows (see reference [5]): Mainly three challenges have driven the development of the GENESYS reference architecture: (a) Complexity Management, (b) Robustness, (c) Energy Efficiency.

GENESYS established a platform architecture providing a minimal set of “core services” and a plurality of optional services (i.e. special communication services, diagnostic services, etc.) predominantly implemented as self-contained system components selected due to the industrial domain application requirements (see Fig. 2).

2.3 Results Used from the GENESYS Project

The GENESYS reference architecture template provides architectural services as a baseline for the development of applications. GENESYS distinguishes between core services and optional services.

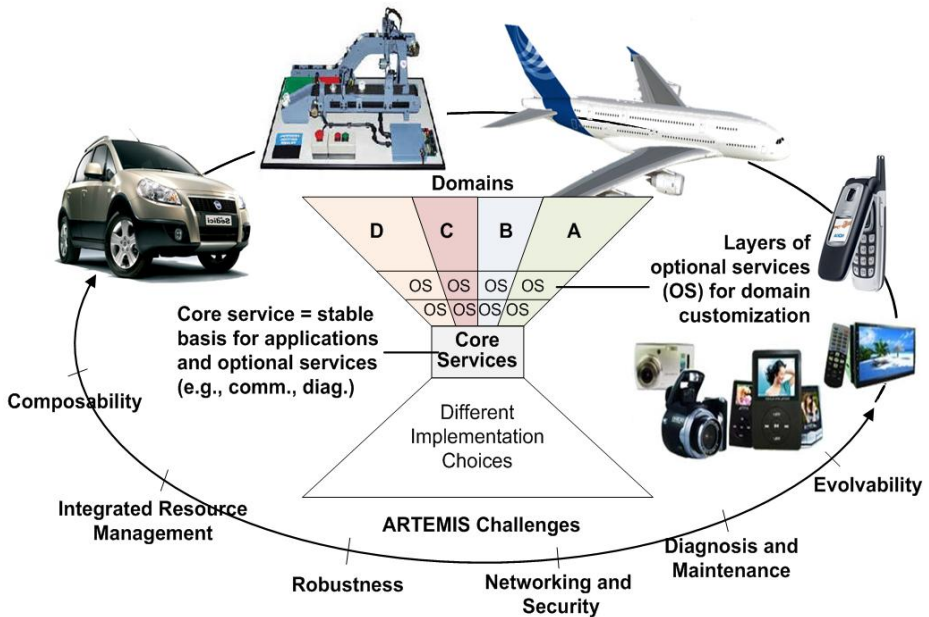


Fig. 2. GENESYS Overview (see reference [9])

Legend:

OS Optional Services

The GENESYS reference architecture targeted to provide (a) a consolidated cross-domain architectural style, (b) a reference architecture template, and (c) a cross-domain development methodology.

The GENESYS Reference Architecture has been described in detail by R. Obermaier and H. Kopetz, Vienna University of Technology, see reference [5]. The book is available free of charge for download from <http://www.genesys-platform.eu/>.

As illustrated in Fig. 3, the GENESYS reference architecture template provides specifications for a comprehensive set of platform services. These platform services can be partitioned into the following three service categories: core services, optional services and domain specific services.

2.4 Core Services

The core services are mandatory and thus part of any instantiation of the GENESYS architecture. The core services are minimal in the sense that only those services which are absolutely indispensable to build higher-level services or to maintain the desired properties of the architecture are included in the set of core services. In GENESYS the core services must be amenable to certification. For this reason they must be deterministic and simple.

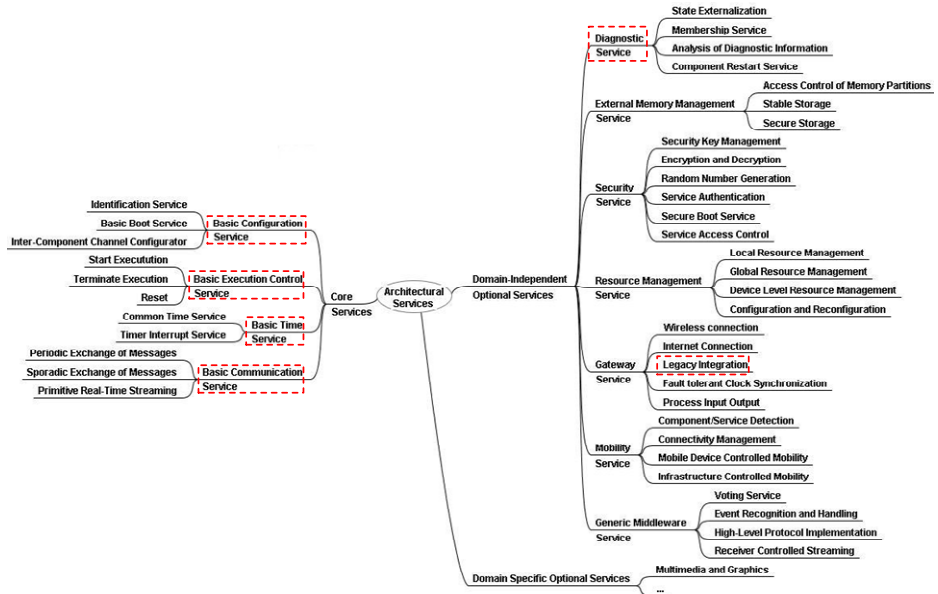


Fig. 3. Services of the Reference Architecture Template

2.5 Optional Services

The optional services are built on top of the core services. Optional services are an open set of services that can be extended as needed. All or only a subset of these optional services can be selected for any particular instantiation of the architecture. Most of the optional services are implemented in self contained system components that interact with the generic middleware (GEM) of the application components by the exchange of messages. In case an optional service is mature and stable, it can be implemented in the form of a hardware component, which leads to a significant improvement in the energy efficiency.

2.5.1 Domain Specific Services

The domain specific services are a sub-set of the optional services and support specific features required by the particular domain or application under consideration.

3 The INDEXYS Approach – A Reply to a Market Pull

Generally, market domains such as automotive, aerospace or railway use their specific processes, development technologies, development tools, verification and validation approaches and test methodologies. Various standards have been established and enormous budgets are required to permanently maintain and improve these technologies with respect to their platform components and performance. Increasing cost pressure and the shortening of “time-to-market” requirements caused the large players in the various market domains to “look over the fence” and check whether processes,

methods and technologies from other market domains wouldn't be suitable for their own field of application as well in order to make use of "economy of scale" and reduce cost and time-to market appropriately for their own competitive advantage.

Thus the idea of establishing a cross domain data communication architecture template and a platform approach, such as offered by INDEXYS, definitely replies to market trends of many domains in parallel. INDEXYS shows the flexibility in integrating domain specific services in combination with template based core services allowing the use of "proven" and "existing" design building blocks and thus resulting in the targeted advantage of reducing cost and time to market by using once developed, cross-industrial platform based technologies and proofs concept in their automotive, aerospace and railway domain demonstrators.

In addition, cross domain industrial use will foster permanent improvements and enhancements resulting from different area of application. Thus the potential community will benefit from a higher maturity of the platform in the long run compared to just developing, maintaining and improving such platform internal of one domain only. The likelihood for being confronted with system based errors is therefore reduced to a minimum and the speed and efficiency of development is optimized.

3.1 Overview of the INDEXYS Project

The objective of INDEXYS is to enable industrial exploitation of GENESYS' cross-domain architectural services, thereby particularly focusing on (but not restraining to) the domains: aerospace, automotive and railway.

The overall strategy of the INDEXYS work plan is defined by three phases, namely Analyze, Implement, and Evaluate, detailed as (see Fig. 4):

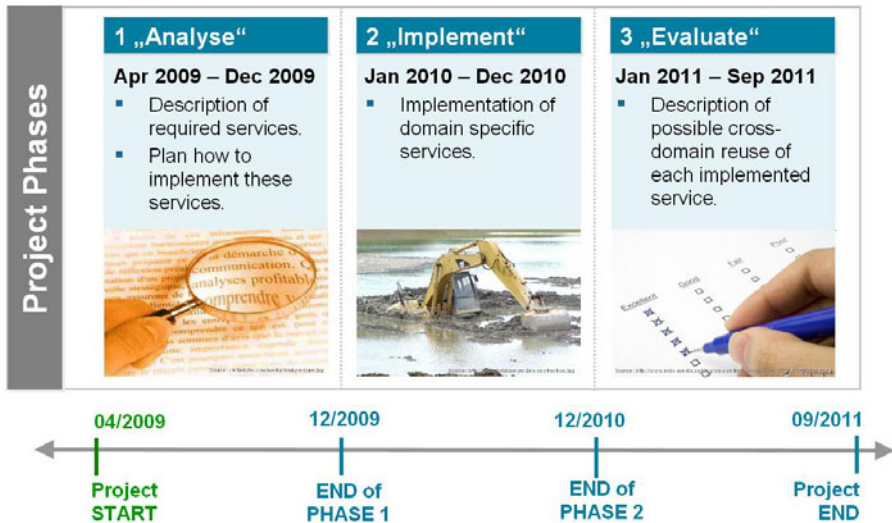


Fig. 4. INDEXYS Project Phases

Phase 1: “Analyse”: Analyze and prioritize GENESYS architectural services and create fundamental paradigms, methods and tools supporting embedded system design for a broad range of applications. This phase will investigate on topics such as: general tool-chain concepts, i.e., development of a comprehensive, customizable tool-chain based on OMG’s Model Driven Architecture (MDA), domain specific modelling based on domain specific modelling languages (DSL), support for advanced synthesis solutions based on COTS tools and configurable transformations, and on-line fault handling that copes with increasing transient fault rates in modern embedded devices. The “Analyse Phase” will evaluate the gap between existing, reusable technology and GENESYS architectural services deemed relevant within INDEXYS and summarize the results in an analysis. All required services will be specified and a plan will be developed how to implement them.

Phase 2: “Implement”: Based on the results of the first phase, three domain specific applications (Aerospace, Automotive and Railway) will be developed. In each of the three domains, a certain set of architectural services will be instantiated. This instantiation will either be by reuse of existing technology or by new developments, where a strong focus will clearly be on reuse of existing components, methods and tools. At that point it should be mentioned that several project partners have closely cooperated in the DECOS project. It is thus planned to (re)use results, i.e., methods, tools and concepts of DECOS wherever appropriate. The “Implementation Phase” will investigate what is required to instantiate GENESYS architectural services so that these services can be exploited for concrete platforms in the automotive, aerospace and railway domains. As result, INDEXYS will develop the domain specific prototype implementations of GENESYS architectural services.

Phase 3: “Evaluate”: As soon as prototype implementations of architectural services are available, an evaluation will take place to find out if the domain specific architectural service implementations can be reused across other domains targeted by INDEXYS. As it is a key objective of INDEXYS to maximize cross-domain reuse, the objective of this phase will be to enable reuse of as many architectural service implementations as possible. INDEXYS will define the required means for enabling reuse of INDEXYS’ architectural service implementations across further domains. The possible reuse of each service implemented during the second phase will be specified.

3.2 INDEXYS Builds on GENESYS and Enhances the Results Achieved

The reference architecture template generated in the course of the GENESYS project offers four core services summarized below (see Tab. 1). These services map to several of the domain specific requirements and features. The core services consist of a couple of sub-services. The INDEXYS Demonstrators make use of these services according to the Tab. 1 below. Please note that throughout all INDEXYS project aspects except the demonstrators all services as identified in the Tab. 1 are considered and supported.

The optional architectural services of the reference architecture template facilitate the establishment of the identified requirements and features are illustrated in Tab. 2 (Again, the table refers to the INDEXYS demonstrators while all optional services are considered and supported in all INDEXYS project aspects except the demonstrators).

Table 1. GENESYS Core Services in INDEXYS Demonstrators

Core Service Name	Sub-Service Name	Automotive	Aerospace	Railway
Basic Configuration	Identification Service	yes	yes	yes
	Basic Boot Service	yes	yes	yes
	Inter-Component Channel Configurator	yes	yes	yes
Basic Execution Control	Start Execution	yes	yes	yes
	Terminate Execution	yes	yes	yes
	Reset	yes	yes	yes
Basic Time	Common Time Service	yes	yes	yes
	Timer Interrupt Service	yes	yes	yes
Basic Communication	Periodic Exchange of Messages	yes	yes	yes
	Sporadic Exchange of Messages	yes	no	yes
	Primitive Real-Time Streaming	no	no	no

Table 2. GENESYS Optional Services in INDEXYS Demonstrators

Service Name	Automotive	Aerospace	Railway
State Externalization	no	yes	no
Membership Service	yes	yes	yes
Analysis of Diagnostic Inform	yes	yes	no
Component Restart Service	yes	yes	yes
Access Control of Memory	no	no	no
Stable Storage	no	no	no
Secure Storage	no	no	no
Secure Key Management	no	no	no
Encryption and Decryption	no	no	yes
Random Number Generation	no	no	no
Service Authentication	no	no	no
Secure Boot Service	no	no	no
Service Access Control	no	no	no
Local Resource Management	yes	yes	yes
Global Resource Management	no	no	no
Device Level Resource Man	no	no	no
Configuration and Reconfig	no	no	no
Wireless connection	no	no	yes
Internet Connection	no	no	yes
Legacy Integration	yes	yes	yes
Fault-tolerant Clock Synchron	yes	yes	yes
Process Input Output	yes	yes	yes
Component/Service Detecti	no	no	no
Connectivity Management	no	no	no
Mobile Device Controlled M	no	no	no
Infrastructure Controlled M	no	no	no
Voting Service	yes	yes	yes
Event Recognition and Hand	yes	yes	yes
High-Level Protocol Implem	no	no	yes
Receiver Controlled Stream	no	no	no

3.3 The INDEXYS Project Consortium

The ten member INDEXYS project consortium coordinated by TTTech Computertechnik AG is well balanced consisting of four industrial partners evaluating the developments in their demonstrators (Audi AG [automotive domain industrial partner], EADS Deutschland GmbH [aerospace domain industrial partner], NXP Semiconductors & Thales Signalling Solutions GesmbH [railway domain industrial partner]), two SMEs providing the technical know-how as the market leaders in the specific technological area (OptXware Research and Development Ltd. & TTTech Computertechnik AG) and four universities representing most relevant academia (Delft University of Technology, Technical University of Darmstadt, Technical University of Kaiserslautern & Vienna University of Technology). The Consortium combines technical excellence of four European countries, Austria, Germany, Hungary and The Netherlands.

3.4 The INDEXYS Project Goals – Progress beyond the State of the Art

Contrary to the approach of many present platform solutions, which are tailored to a specific domain, INDEXYS aims at the development of reusable architectural services that can be exploited across platforms of different domains. INDEXYS' architectural service implementations will support a gradual shift towards higher reusability of services across different domains (particularly across automotive, aerospace and railway domains) due to lower cost by availability of existing solutions, and by existing experience with these solutions in the engineering community, see reference [7].

INDEXYS will build on prevailing platform solutions such as AUTOSAR for automotive systems, IMA for aerospace systems (i.e. AFDX (see reference [9]), see also SPIDER project, see reference [10]), and TAS Control Platform (see reference [8]) for railway systems with the goal to implement selected GENESYS services within these platforms.

INDEXYS will further observe (and potentially take up) related technical approaches of other domains such as telecommunication or consumer electronics.

Technological Aspects

In line with the results of GENESYS, INDEXYS will implement architectural services according to GENESYS' cross-domain architectural style. Thereby, the following **technological aspects** will be considered for complementing existing platform solutions (and thereby progressing beyond the state of the art): (a) Networking and Resource Management (i.e. development of scalable, deterministic communication and design methods), (b) Robustness, Diagnosis and Maintenance (i.e. architectural services ensuring the "capability of a system to deliver an acceptable level of service despite the occurrence of faults"), (c) Composability (i.e. architectural support of the constructive composition of large systems out of components and sub-systems without uncontrolled emerging behavior or side effects) and (d) Methodology and Tools (i.e. multi-paradigm tool/methodology approach based on the Model Driven Architecture (MDA) and on the Model Driven Development).

3.5 The INDEXYS Developments

INDEXYS will first of all develop methodologies (tool chain, modelling, verification and validation, on-line fault handling and complexity management) and then focus on the industrial domains (Automotive, aerospace and railway) and conclude the project in a cross-domain integration phase. As usual in European Commission projects this is accompanied by a dissemination/exploitation and standardization work package and project management activities (see Fig. 5).

3.5.1 Methodology

The objectives within this INDEXYS topic will analyze and prioritize GENESYS architectural services and thereby create fundamental paradigms, methods and tools supporting embedded system design for a broad range of applications. Furthermore, an initial project alignment according to the final GENESYS results is performed. Analysis and implementation of exploitable cross-domain services is provided. INDEXYS Methodology covers (a) general tool-chain concepts, (b) domain specific modeling and synthesis support, (c) design time verification and validation, (d) on-line fault handling and (e) complexity management.

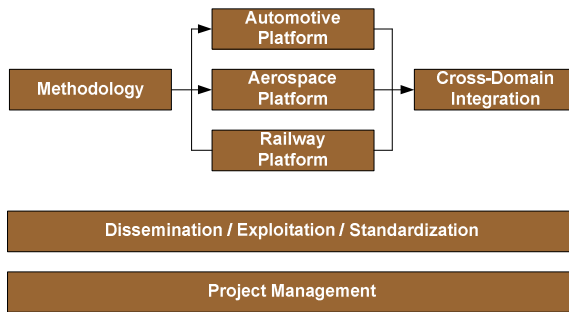


Fig. 5. INDEXYS Development and Work Plan Overview

3.5.2 Industrial Platforms

An overview on the three industrial domains and their innovation is provided in Fig. 6.

Automotive Platform

The objective of the automotive platform takes under consideration the increasing complexity of current and future automotive systems requiring composable subsystem integration. Composable subsystem integration guarantees that properties which have been established at subsystem level are not invalidated through the integration of these subsystems into a larger system. It is the objective of this work package to investigate on and to enable composable integration of real-world in-vehicle network technology of prevailing automotive networks, such as CAN and FlexRay, also considering Functional Safety. The prototype implementation will be validated also for their composability by thorough validation in a target vehicle network.

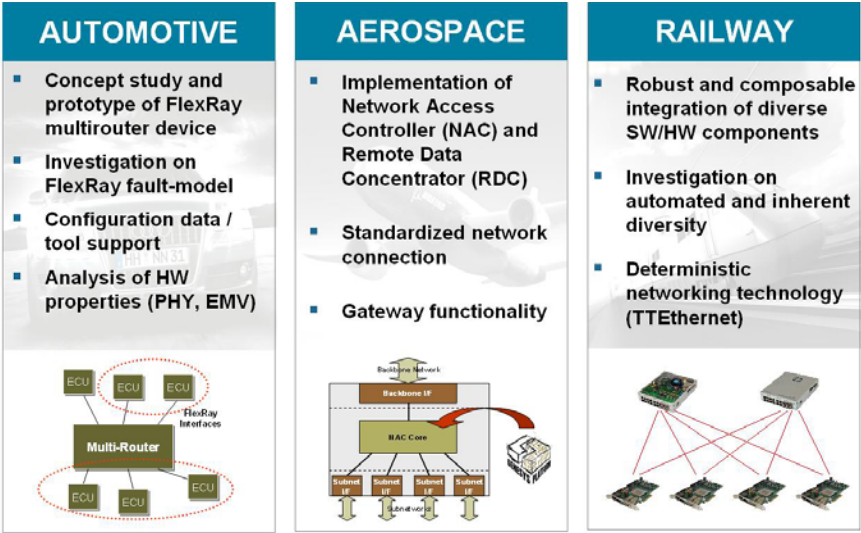


Fig. 6. INDEXYS Innovation per Domain

Legend:

ECU	Electronic Control Unit	I/F	Interface
PHY	Physical Layer	EMV	Electro Magnetic Compatibility
RDC	Remote Data Controller	NAC	Network Access Controller
HW	Hardware	TTEthernet	Time-Triggered Ethernet

With respect to multiple CAN networks required in automotive industry, INDEXYS will elaborate a completely new integration concept. This integration concept will be based on a GENESYS conformant composable CAN interconnect that supports the legacy CAN interface of existing ECUs, increases the CAN network performance and will provide a programmable gateway service between different CAN networks.

In INDEXYS the concept of a FlexRay multi-switch will be investigated and prototypically implemented. A multi-switch device directly connects to FlexRay end-systems in a star topology and allows parallel transmission of messages from different senders at the same time (given that the receivers are disjunctive). The methodology for cross-domain reusable safety modules will be applied in selected areas on the prototypes. This means: (i) early definition and consideration of safety requirements, (ii) ongoing coaching of the development for functional safety needs, and (iii) evaluation of safety, including reachable Hardware Safety Integrity.

Demonstrator Automotive - FlexRay Multi-Switch

The FlexRay “multi-switch” is a cut-through switch for FlexRay networks. It switches a number of FlexRay branches according to a pre-defined static schedule. The communication elements are forwarded with minimal delay and are not stored and forwarded at a later point in time.

A FlexRay multi-switch is a device which is physically similar to the FlexRay Active Star device, but in contrast to the Active Star, it implements a selective switching

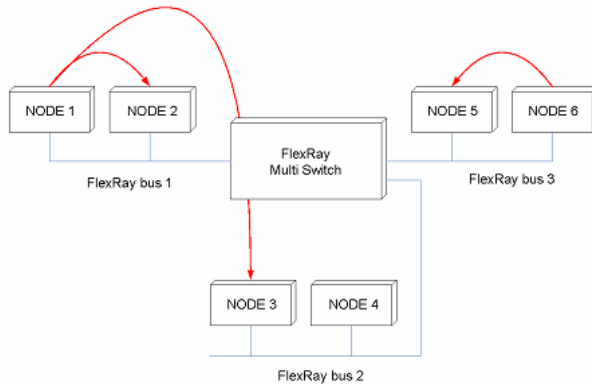


Fig. 7. Automotive Platform FlexRay Cluster with FlexRay Multi-Switch

of the communication paths according to a configured switching schedule. The multi-switch is able to provide additional functionality of complex data traffic paths and also isolation of branches Fig. 7.

Aerospace Platform

The objective of the Aerospace Platform deals with Integrated Modular Avionics (IMA), where distributed functional computations are centralized on a group of Central Processing Modules with powerful CPUs interfacing small, easy to maintain and reliable Remote Data Concentrators. This state-of-the-art platform concept is widely used for safety-critical control functions of the airplane, but currently not for cabin control applications. The objective is to extend the IMA approach due to enhancements of a Remote Data Concentrator and of a Network Access Controller (NAC) for cabin control applications.

INDEXYS will specify and develop a simple System-on-a-Chip solution of a Remote Data Concentrator for transducer interfacing in distributed aerospace applications. The Remote Data Concentrator will offer a standardized network connection to a time-triggered field bus (e.g., TTP).

The task of Network Access Controller (NAC) is to provide Gateway functionality between a high-speed backbone (e.g., Ethernet) network and up to eight linear buses with several passenger oriented devices. A main focus within aircraft cabins is the development of a system wide cabin-communication architecture that incorporates all electronic cabin systems. Currently, there exist three separate cabin systems: CIDS (cabin interconnection data system), ALNA (airline network architecture), and IFE (in-flight entertainment).

A cabin backbone network (e.g., Fast Ethernet, TTEthernet) interconnects the various components of the cabin management system. The Network Access Controller (NAC) with smart wireless interfaces provides redundant and secure links. Moreover, the network must provide sufficient reliability, fault tolerance, guaranteed bandwidth, high-integrity, bounded latency and jitter, security, scalability, reconfigurability, and interoperability for various heterogeneous components. INDEXYS will specify and

develop a Network Access Controller based on architectural and modular principles. Thereby, the focus will be on device level integration.

Several GENESYS principles will be addressed such as complexity management (i.e., reduction of cognitive complexity is achieved through small and easily understood interfaces between core processing modules and associated transducers), component based design (i.e., separation of processing devices and input/output devices and communication over linking interfaces), hard and soft components (i.e., FPGA based implementation of Remote Data Concentrator), message passing (i.e., message based Remote Data Concentrator interface), composability (i.e., deterministic communication over time-triggered network) and the concept of a common time (i.e., global time which is shared across the network).

Demonstrator Aerospace - Network Access Controller (NAC)

Fig. 8 shows the principle structure of a network architecture used to provide connectivity from a central unit like a server to passenger oriented devices like a PSU (Passenger Service Unit) in an aircraft cabin for cabin control applications. This cabin communication architecture should incorporate all electronic cabin systems.

The network consists of one or more Central Units (CU), Network Access Controllers (NAC) and Passenger Service Units (PSU). A high data rate backbone connects up to 12 NACs with the CU. Each NAC provides at least four subnets for connecting to the network up to eight PSUs per subnet.

With this configuration it is possible to connect 32 passenger oriented devices to one NAC. In total up to 384 passengers oriented devices are possible with the use of 12 NACs.

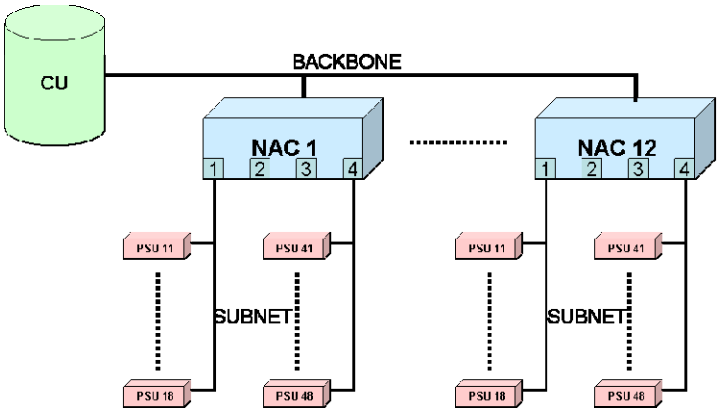


Fig. 8. Cabin communication network architecture

Legend:

CU	Central Unit	PSU	Passenger Service Unit
NAC	Network Access Controller		

The NAC itself connects the high data rate backbone with the sub-networks. Fig. 9 shows the block diagram of a NAC. The NAC has a modular based structure with generalized interfaces to the backbone and to the sub-networks. The NAC Core Module provides gateway functionality between the backbone and the sub-networks. The backbone network is based on Ethernet communication protocol whereas the sub-networks can be implemented to handle protocols like CAN, Ethernet, etc.

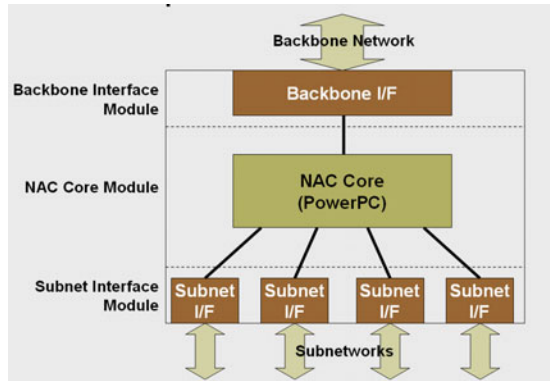


Fig. 9. Block diagram of Network Access Controller (NAC)

Demonstrator Aerospace - Remote Data Concentrator (RDC)

The task of a Remote Data Concentrator (RDC) is to interface transducers (i.e., sensors and actuators) in the aircraft. Remote Data Concentrators are connected via communication systems to the Central Processing Modules. To achieve a highly deterministic behavior for the RDC communication a time triggered communication protocol (TTP) has to be used.

In the past TTTech developed a table driven communication layer (TD-COM Layer, see Fig. 10) for TTP in software to make use of the reduced certification effort during reuse. The TD-COM Layer implements a high-performance communication layer between TTP networks and host applications. The TD-COM Layer can support up to two TTP networks, each being connected by a separate TTP controller.

For simple distributed communication nodes as Remote Data Concentrators are a solution with the TD-COM layer requires a lot of CPU power for executing the packing and unpacking of messages. This hints to a TD-COM implementation based on a FPGA or moreover as ASIC.

Railway Platform

The majority of Thales Rail SIL4 signaling and train protection system products worldwide is based on a generic fault-tolerant computer platform (HW, OS, Middle-ware), which has been developed during the last decade by the competence centre of Thales Rail Signaling Solutions Austria (former Alcatel Transport Automation Solution). After the successful rollout of the first generation of the system (called "TAS Control Platform") in more than 20 countries on 4 continents, the TAS Control Platform is now going for the second generation. Major items on the roadmap of TAS

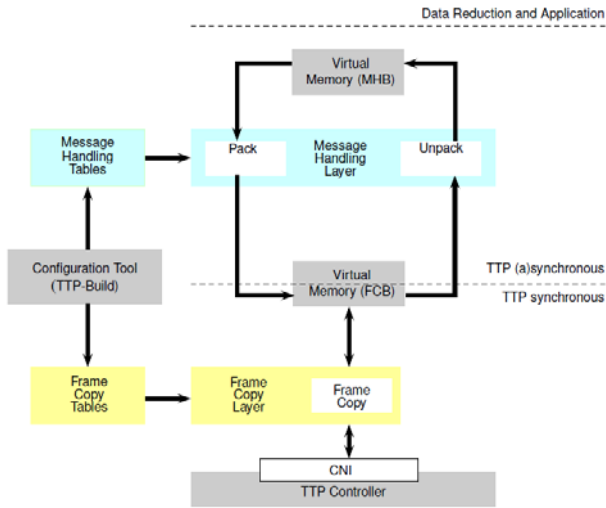


Fig. 10. TD-COM Layer overview

Legend:
CNI Controller Network Interface TTP Time-Triggered Protocol

Control Platform 2.x is the architectural support for application binary diversification and the increasing use of COTS components.

INDEXYS will develop mitigation strategies to cope with unknown faults in re-used components to make the use of off-the-shelf components possible. The CENELEC standards highly recommend the use of diversity to mitigate the risk of undetected faults and to increase robustness in complex components, i.e., the implementation of diverse hardware/software components. In addition, INDEXYS will develop a novel concept for the implementation of fault-containment regions as defined in GENESYS. Conventionally, fault-containment regions w.r.t random hardware faults are formed by active redundant hardware components. INDEXYS seeks to implement fault containment regions on a single hardware channel running diverse application software channels. These software channels are being deduced automatically from legacy source code. The motivation for this approach is to save hardware costs in high volume applications without having the burden of N-Version programming during development.

The integration of these diverse hardware/software components must be composable so that the properties of each independently developed component are not invalidated by its integration into a larger system. The key pre-requisite for composability is deterministic system behavior.

TTethernet is a novel communication protocol that has been designed with the requirements to provide temporal deterministic communication and to support standard Ethernet communication without the need to change or configure higher level communication protocols, like IP, TCP, UDP, FTP. A TTethernet system consists of a set of computer nodes interconnected by a specific switch called TTethernet switch. A

node can be either a standard Ethernet node or a TTEthernet node. A standard Ethernet node consists of a COTS Ethernet controller and a host computer. A TTEthernet node consists of a TTEthernet communication controller that executes the TTEthernet protocol and a host computer that executes the user application.

TTEthernet nodes (end systems) can be implemented either in hardware (FPGA based TTEthernet communication controller is available) or by using a COTS Ethernet controller and by implementation of the TTEthernet stack on top of a COTS Ethernet controller. In the INDEXYS project, it is planned to implement a SW based end system for TAS Control Platform hardware (Intel PC based).

Demonstrator Railway – TTEthernet Integration into TAS Platform

The TAS Control Platform is an open, scalable software architecture oriented towards established industrial computing standards. The communication system offers a number of standard communication services, such as Internet Protocol (TCP/IP family), serial lines and field buses (CAN controller area network, TTP time triggered protocol, PROFIBUS Process Field Bus), as well as specific safe communication services conforming to European Committee for Electro technical Standardization (CENELEC) standards, see reference [12].

At the hardware level, the TAS Control Platform uses commercial off-the-shelf components, which are supplemented by added-value services for railway control systems.

Fig. 11 shows the component architecture of TAS Platform. A “Computing Node” (CN) is the logical target computer. It may consist of 1 up to 3 individual “Computing Elements” (CEs), depending on the application systems replication degree. A CE refers to a physical computer that is synchronized with other CEs of the same CN. A “Task Set” (TS) is a set of tasks forming a logical application software entity.

The synchronization medium serves inter replica synchronization and is implemented as point-to-point network based on Ethernet.

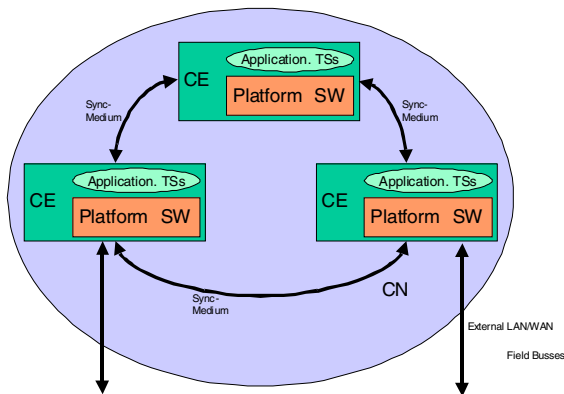


Fig. 11. TAS Platform Component/System Architecture

Legend:

CE	Computing Element	CN	Computing Node
TS	Task Set		

Implementing the Software-based TTEthernet protocol into diverse computing elements of a computing node on TAS Platform will increase the deterministic behavior of the communication system with the advantages of a Time Triggered Communication system.

The Software-based TTEthernet specifies a special implementation of TTEthernet which was created to make use of the time-triggered communication benefit implemented in software without the fault tolerant features of the hardware-based solution enabling also high throughputs for much lower costs. If the use case requires fault tolerance mechanisms then they have to be implemented in the application on top of the TTEthernet API Library.

Fig. 12 shows the layered structure of the software-based TTEthernet implementation on a host without an operating system including the TTEthernet protocol core layer embedded between the hardware layer and the API library.

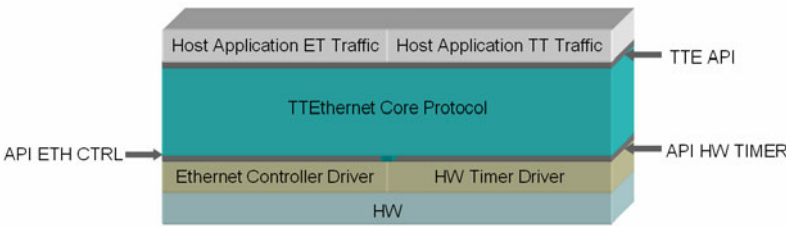


Fig. 12. Software-Based TTEthernet detailed architecture

Legend:
ET Event Triggered
HW Hardware
TTE TT Time Triggered
 Time Triggered Ethernet

Setting up an application with software-based TTEthernet on a platform with an operating system a driver has to be established on top of the TTEthernet core protocol to get access to the core functions (see Fig. 13).

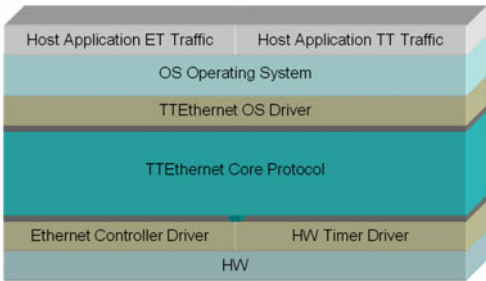


Fig. 13. Software-Based TTEthernet with Operating System Driver

3.5.3 Cross-Domain Integration

A key goal of INDEXYS is to enable cross-domain reusability of architectural service implementations which are instantiated for platforms in the automotive, aerospace and railway domains. Cross-domain reusability is the ability to reuse architectural service instantiation which have been implemented for one of the targeted domains (i.e., automotive, aerospace, railway) in other domains. Cross-Domain Integration will focus on analyzing, steering, and evaluating cross-domain reusability during the actual project work. A particular focus will be the reusability within the other targeted domains of INDEXYS. However, reusability can also apply for domains which are not directly in the scope of INDEXYS such as consumer electronics or industrial control systems.

Reusability of safety-related modules across different domains will be supported by deriving a generic methodology for managing reuse of safety-related modules. The methodology shall be applied to one implementation to test its usability in real life. This will allow improvement of the concept. The so improved concept and the experience gained from applications will be investigated with experts from other domains. If suitable the generic methodology can be published and brought into domain standardization groups for further propagation. A good methodology for reuse will allow sustaining advantages from the considerable investments made by companies in developing safety modules with the appropriate safety evidence. This enables a considerable competitive advantage for safety-related developments.

4 ACROSS – Closing the Gap

The objective of the ACROSS project to develop and implement an ARTEMIS cross-domain architecture for embedded MPSoCs based on the architecture blueprint developed in the FP7 project GENESYS (Generic Embedded System Architecture). ACROSS will result in the design of a generic Multi-Processor Systems-on-a-Chip (MPSoC) and a first implementation in an FPGA. The ACROSS MPSoC will provide a stable set of core services as a foundation for the component-based development of embedded systems with short-time-to-market, low cost and high dependability. The ACROSS-MPSoC will be demonstrated as a universal platform for automotive, aerospace and industrial control systems in order to realize the benefits of the economies of scale of the semiconductor technology. Additionally, the ACROSS-MPSoC platform provides significant potential for being adopted by other industries with safety-critical data communication requirements such as applications in the medical-, power generation -, (i.e. atomic power plant control equipment), space- domains, to mention just a few. Using the core services of the ACROSS-MPSoC, a library of middleware services will be realized in the ACROSS project. Generic middleware will offer services to be used in multiple application domains (e.g., fault-tolerance, diagnosis, security), while domain-specific middleware will implement domain-specific services for specific domains (e.g., AUTOSAR services for the automotive domain, IMA services for the avionic domain). Another significant result of the project will be a general design methodology, supported by appropriate adaptable tools, for the implementation of ACROSS-based applications. The benefits of the cross-domain architecture will be

shown in demonstrators from the targeted application domains. It is planned to set up demonstrators for the automotive-, the aerospace- and the industrial control domains.

5 Conclusion

The introduced projects DECOS, GENESYS, INDEXYS and ACROSS do perfectly complement each other. Initially, the DECOS project demonstrated the advantages of the design, development and validation of an embedded platform for dependable, integrated systems.

The next step beyond was taken in GENESYS providing a flexible, cross domain reference architecture defining the fundamental architectural principles required. GENESYS specified, developed and designed the core services and a set of initial optional services allowing a start in applying the results in industry.

INDEXYS now provides the first implementation and proves the feasibility in establishing demonstrators for the automotive-, the aerospace- and the railway sectors using the GENESYS reference architecture platform approach.

To finally provide means for industrial series production visions, ACROSS designs, develops and verifies a FPGA based device integrating a GENESYS architecture on one chip. This will bring development cost further down and will enable the large scale application of the technological step taken in GENESYS and INDEXSYS.

As a final step in a R&D project family industry will obtain all tools, methodologies and embedded building blocks to efficiently and competitively implement systems based on the GENESYS reference architecture. The fact that the entire platform is available will pave the way and significantly improve the chance that different industries will pick up the technology for some of their visionary projects in close future.

References

1. The ARTEMIS Joint Undertaking Annual Work Programme, ARTEMIS-PAB-17/08 (2008)
2. ARTEMIS Joint Undertaking Strategic Research Agenda, Reference Designs and Architectures, Edition May 2006 (ARTEMIS SRA Working Group)
3. Kantz, H., König, N.: TAS Control Platform: A Vital Computer Platform for Railway Applications. Technology White Paper (2004)
4. Aeronautical Radio Incorporated (ARINC), Annapolis, MD, USA. ARINC Specification 651: Design Guide for Integrated Modular Avionics (November 1991)
5. Obermaisser, R., Kopetz, H.: GENESYS: An ARTEMIS Cross-Domain Reference Architecture for Embedded Systems. In: Südwestdeutscher Verlag für Hochschulschriften Aktiengesellschaft & Co.KG, Saarbrücken, Germany (2009) ISBN 978-8381-1040-0, The book can be downloaded free of charge: <http://www.genesys-platform.eu/>
6. Obermaisser, R.: Presentation on GENESYS, Vienna University of Technology
7. The information refers to the INDEXYS Description of Work (according to the contractual agreed content of project work)

8. Kantz, H., König, N.: As for AUTOSAR and IMA, summary descriptions are contained from the GENESYS description of work (DoW). The summary of the TAS platform is taken from the technology white paper: TAS Control Platform: A Vital Computer Platform for Railway Applications
9. Avionics Full-Duplex Switched Ethernet. Wikipedia, The Free Encyclopedia (October 22, 2008),
http://en.wikipedia.org/w/index.php?title=Avionics_Full-Duplex_Switched_Ethernet&oldid=246938506
(Retrieved 13:13, November 10, 2008)
10. <http://shemesh.larc.nasa.gov/fm/spider/>
11. Atat, Y., Zergainoh, C.f.: Simulink-based MPSoC Design: New Approach to Bridge the Gap between Algorithm and Architecture Design. In: IEEE Computer Society Annual Symposium N.-E. VLSI, 2007. ISVLSI apos 2007, March 9-11 (2007)
12. Refer to the following homepage for further information about CENELEC and related standards: <http://www.cenelec.eu>
13. DECOS homepage: <http://www.decos.at>

Integrating System Modelling with Safety Activities

Bernhard Kaiser, Vanessa Klaas, Stefan Schulz, Christian Herbst, and Peter Lascych

{bernhard.kaiser, vanessa.klaas, stefan.schulz,
christian.herbst}@berner-mattner.com,
peter.lascych@continental-corporation.com

Abstract. Increasing enforcement of safety standards – such as the new ISO 26262 – requires developers of embedded systems to supplement their development processes with safety-related activities, such as hazard analysis or creation of technical safety concepts. Since these activities are often only loosely coupled with core development tasks, their addition reduces efficiency and causes a lack of consistency and traceability. This paper presents an approach to the integration of architectural modelling, modelling of failure nets, allocation safety mechanisms to architectural elements, and finally traceability to requirements and test coverage. The presented methodology gives clear instructions for the comprehensive usage of existing techniques. The process is demonstrated using a real-world example from the automotive sector. In two industrial projects a significant increase of productivity could be achieved, solely using standard tools such as DOORS and IQ-RM. Nevertheless, the paper concludes with some suggestions for further enhancement of the method through formalization, e.g. using SysML, and tool integration.

1 Introduction

Although consideration of safety aspects has a long tradition in the automotive business, the integration into the development process of automotive embedded systems is still not satisfactory. Automotive manufacturers and suppliers are more familiar with mechanical components implementing safety functions, than they are with software-controlled components. Software and system development processes have not yet attained a high level of maturity, and in particular software and hardware interfaces are sometimes poorly specified. In addition, safety processes, such as those defined by the new ISO 26262, are often imposed from “outside”, i.e. by external safety specialists unfamiliar with developers’ daily work, leading to regular misunderstandings and inconsistencies. The use of (semi-)formal system models, where they exist, for safety analyses is not formalized, and also the feedback of safety measures into the development process, which should take place during the requirements engineering phase, is not carried out in practice in the formal way that the standard requires.

This paper presents a methodology that integrates existing techniques into a consistent framework, supporting the whole safety development cycle. The key point is to start hierarchical system modelling and feature allocation early in the project, for example using UML/SysML. The requirements, collected in DOORS and grouped by

features are allocated to these blocks. Next, hierarchical failure chain modelling, as offered by the APIS IQ-RM tool, is carried out, in order to investigate dangerous failures systematically. The failures at the top level model, describing the system in its context, correspond to hazards, which have to be assessed for severity during hazard analysis. The failures of system blocks deeper down the architectural hierarchy are investigated with failure mode and effects analysis (FMEA), augmented by a keyword approach taken from the HAZOP technique for systematic investigation of failure possibilities at block interfaces. The cause-effect chains are modelled from the failures at the lowest hierarchical level (e.g. failures of individual hardware elements) up to the system level hazards. The allocation of failures allows modifications of the system architecture in subsequent iterations by inserting detection and reaction measures for identified failures and by allocating safety functions to architectural elements. Thanks to the clear correlation between FMEA, Safety Concept and system architecture, change management and traceability is easier and consistency issues are reduced. Finally, rephrasing the measures allocated by the Technical Safety Concept produces detailed requirements that are fed back into the standard requirements management system, so that they can be tracked throughout the project implementation and test phases. The aforementioned allocation of low-level requirements to architectural elements helps both the developers and the safety assessors to understand the relationship between safety requirements and the system (safety) architecture.

The rest of the paper is structured as follows: section 2 introduces a simplified electric drive system that serves as an example throughout all the process steps. Sections 3 and 4 describe the initial activities, feature and requirements engineering, as well as hierarchical modelling of the system architecture. The following sections 5 and 6 introduce the safety activities: hazard analysis, investigation of specific malfunctions of the system and the building of cause-effect chains. The interfaces of these activities with system modelling activities are detailed. Section 7, dealing with the creation of the safety concept, puts the pieces together: the safety measures are allocated to the system architecture and safety requirements are derived. The further activities required in order to achieve traceability of the requirements to system implementation and to testing activities are also described. The conclusion in Section 8 lists some of the benefits of this method in an actual automotive project and makes suggestions on how to improve the methodology, e.g. by formal tool integration.

2 Description of the Example System

As a continuing example throughout the rest of the paper, we choose an electric drive system, consisting of a three phase synchronous electrical machine with permanent excitation and a power inverter ECU (electronic control unit), containing the microcontroller with all peripherals and interfaces, analogue circuitry for the acquisition of measurement signals, driver circuitry for the power stage and as the power interface a bridge comprised of six insulated gate bipolar transistors (IGBT) power semiconductors. Drives of this kind are typical for many automotive and industrial applications, such as hybrid or electric vehicles or electrical power steering in passenger cars.

The electrical machine (EM) is equipped with a rotor position sensor, which delivers two analogue signals, called sine and cosine tracks, from which the software determines the rotor angle as a value in degrees. The power inverter is equipped with three-phase current sensors, which deliver the actual current measurements as analogue inputs to the microprocessor. From these values and further physical constants, the software is able to determine the actual torque (rotational force) of the EM (an understanding of the underlying mathematical formulas and algorithms is not required to follow the example). The EM acts both as a motor and as a generator, depending on the direction of the torque. The electrical power originates from or is fed into a DC voltage source called DC Link. The drive is operated in torque closed loop control, i.e. the software algorithm tries to control the IGBTs such that the actual torque matches an externally generated torque reference value. The torque reference, as well as operation mode commands (switch on and off, failure reset etc) are received from a CAN serial bus.

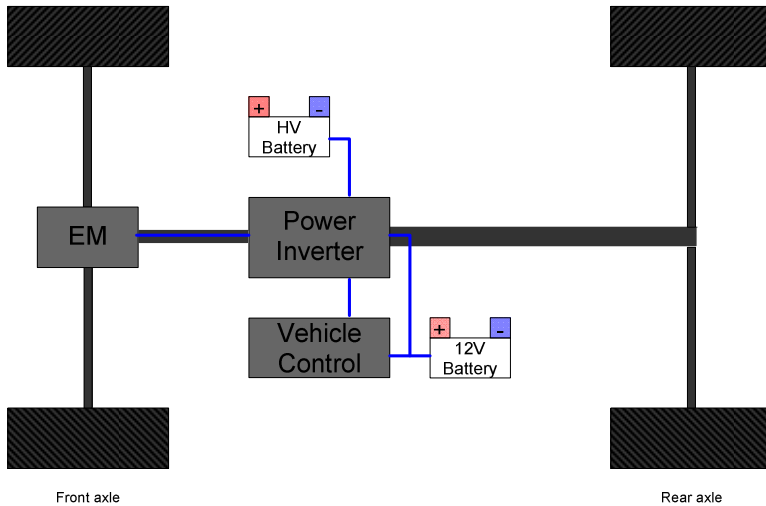


Fig. 1. Overview of the example system

3 Identifying Features and Requirements

Development of these kinds of systems usually starts with an initial product concept and the analysis of the customer requirements. As the wording and structure of the customer requirements often does not correspond with the needs of the supplier, the derivation of system requirements, and later hardware and software requirements, are mandatory parts of a mature development process. Any questions, which arise on the assumptions about the intended use and about the operational environment of the system, are clarified in workshops with developers and the customer.

In our example project, we use DOORS to specify the features and requirements as individual, traceable items. For the sake of efficiency and quality, we use a fixed structure given by templates that have already been proven in use in a series of past

projects. The templates provide stable chapter structures that allow a hierarchical structure according to major functional units (such as interfaces, drive control functions etc.). Within the main chapters, there are subchapters such as for different operation modes and protective features etc. The lowest heading level represents the features (e.g. overvoltage protection), within which the individual requirements are located.

To improve the efficiency and quality of the requirements, we recommend using phraseology templates when specifying system requirements, e.g. “When the CAN signal EME_STOP has the value 1, the system shall switch off the electrical machine current within 20ms” [1]. Tabular data and parameters are stored separately, so that adaptability and re-use in other projects is possible.

Usually, experienced developers are always influenced by past experience, former realisations or first ideas for the safety architecture and therefore design some safety features from the beginning, such as redundant sensors or plausibility checks. As the initial Hazard Analysis shall analyse the system without safety measures, we mark these features in the requirements set and in the initial system architecture, in order to skip them for the first iteration of safety analysis.

During requirements engineering, it is also important to capture assumptions and constraints about the system usage and its environment. For the later safety argument, these explicit assumptions will play an important role, as safety arguments will be based on them.

4 Hierarchical Modelling of the System Architecture

Analysing the requirements for consistency, understanding and feasibility demands that modelling activities begin early together with first conceptual ideas. The model scope in this phase is the system context, i.e. the system as a black box, its environment, the external system interfaces and the signals and energy exchanged with the environment. As this is at the same time the highest level of the static architecture, we also speak of “level 0” of the architecture. Weilkens [2] introduces this type of diagram as a “System Context Diagram”. This diagram is also the starting point for the Hazard Analysis, where dangers in the system environment are investigated, together with the interactions at the external interfaces of our system that may cause or inhibit these hazards.

The interactions with the environment are best represented on this level by sequence diagrams (part of UML and SysML) if discrete interactions such as switching the system on and off are concerned. For continuous signal flows (e.g. electric currents or voltages), SysML also offers modelling stereotypes. However, it is more usual to use informal block diagrams in this stage of the project.

As soon as the broad requirements and constraints and the external interfaces are known, the initial system architecture is developed based on an initial understanding of the system resulting from workshops, preliminary development and experience from the development of similar systems. The design flow is mainly top-down, grouping the system into subsystems and components. The initial “level 0” static architecture (see figure 2) (the system as a black box in its environment) is broken down into subsequent levels, where the inner structure of the black boxes becomes visible.

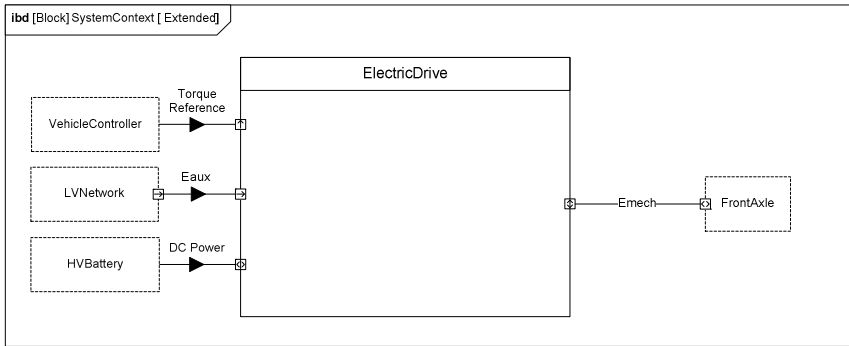


Fig. 2. Overall structure of the example system and its environment

The components of the system are defined iteratively and their interaction is analysed in a similar way to the interaction of the system with its environment, in order to define internal interfaces. These interfaces are not yet specified in detail, rather they are grouped into hardware/hardware interfaces, hardware/software interfaces and software/software interfaces. The system architecture is hierarchically modelled and the components and interfaces are refined in a step-by-step fashion. Common modelling techniques include UML, SysML and Simulink. This decomposition is repeated, until a certain degree of detail is reached in the description of hardware and software components (e.g. hardware block diagram or software architectural diagrams modelled in UML). For the application of this method, the selection of a specific tool is of secondary relevance; first experience in pilot project has been gained using Enterprise Architect; but due to the restrictions of plain UML used in the projects, the data flow models for the application of this method have been drawn by the simple drawing tool Visio and transformed by hand for the subsequent analysis steps.

The results are structural diagrams that describe the system components, the internal and external interfaces as well as the allocation of the functionality to the architectural elements. This allows the specification of the purpose of each port and signal, which will provide important information regarding failure causes and consequences during later cause-effect chain modelling. In our example, we use a SysML internal block definition diagram (ibd, which corresponds with the UML composite structure diagram) in order to illustrate this structure. For sake of readability, we omit the port symbols.

The same structure can be shown as a SysML block definition diagram (denoted bdd), transforming the nesting hierarchy into a tree structure, using the composition relation (similar to UML class diagram), see Fig. 4. This representation, which can be generated automatically by many UML or SysML tools, will serve at the same time as a structure for the hierarchical FMEA.

Having described the static structure in a first draft, we start to specify the dynamic behaviour. Several types of diagrams describe different behavioural aspects. Discrete behaviour at interfaces is illustrated by sequence diagrams whereas state charts describe discrete behaviour in terms of system and software states. If parts of SW or

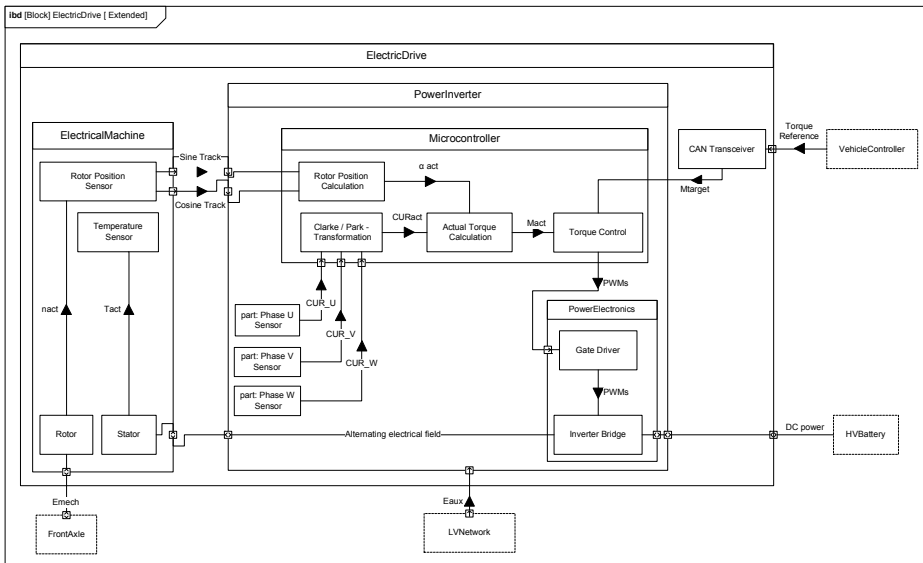


Fig. 3. Inner Structure of the example system (SysML Internal Block Diagram, port symbols partly omitted for better readability)

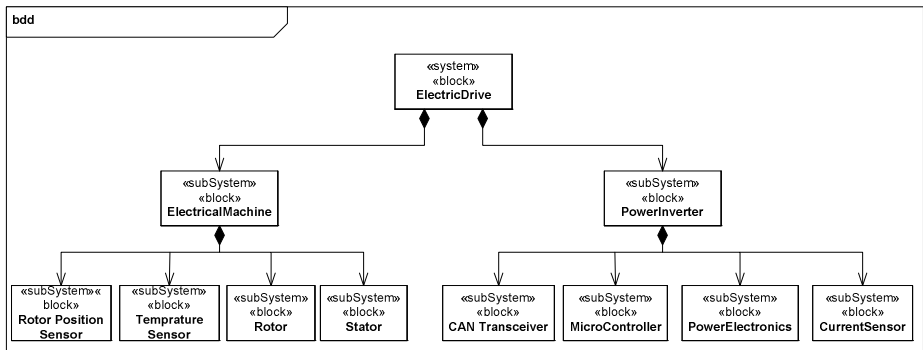


Fig. 4. Representation of the same structure as a composition tree

HW procedures (such as decision about operating conditions) are relevant to show on this level of abstraction they are explained by activity diagrams. Timing diagrams improve the understanding of the system functionality and can be used to spot potential performance problems, synchronisation problems or race conditions. When specifying the safety concept, these timing diagrams can be used to infer failure reaction times. Continuous data streams, which are common in continuous control systems, are difficult to describe in plain UML; therefore SysML (or alternatively Simulink) is a better choice for modelling them, as data flow representations will become essential later for failure propagation modelling.

While refining the architecture and thereby achieving more detailed system understanding, the allocated requirements are also broken down and allocated to the appropriate elements of the system architecture. The relationship between architecture elements on every level and features are documented by means of links to DOORS, which was in the pilot project achieved by the linking capabilities of Enterprise Architect.

5 Hazard Analysis

The objective of the Hazard Analysis is to identify dangers to humans caused by the vehicle, i.e. in the system environment. Therefore it is usually performed in cooperation between the car manufacturer and the electronics supplier. Hazardous failures caused by the system under investigation necessarily involve its external interfaces. Therefore, the initial “level 0” system architecture (see Section 4) with the system, its interfaces and its neighbour systems serves as the input for the Hazard Analysis.

Hazards are identified by systematic investigations on interfaces with the environment with an appropriate combination of analysis methods such as System FMEA [3][4] and HAZOP [5]. FMEA is a structure-focused analysis method that takes the components as the basis for an investigation of functions and corresponding malfunctions, therefore counted among the inductive methods. The FMEA method includes classification of malfunctions with risk priority numbers (RPNs) and definition of measures for avoidance, detection, control or mitigation of malfunctions.

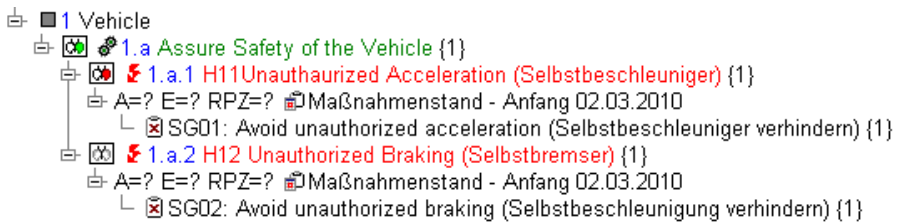


Fig. 5. Excerpt from a FMEA as a means of Hazard Analysis

Whereas normal System FMEA takes the system for root of the decomposition tree, the hazard analysis FMEA starts one hierarchical level above, considering the system environment (vehicle, traffic situation) as well, because this is the place where the hazards occur (see Fig 5).

In order to exploit the interface-centred view provided by the “Level 0” architecture (system with interfaces to its environment), we augment the FMEA method with elements of a HAZOP analysis. The HAZOP method analyses signal flows at interfaces of blocks using keywords like “too high”, “too low”, “too late” or “unexpected”. In our example system, the investigation of the interface “acceleration” would, for instance, reveal the hazards “unauthorized acceleration”.

Unintended interfaces (that are therefore not mentioned in the system architecture) must be identified and examined as well. An example of an unintended system

interface bearing hazard potential could be some bad electrical influence on the vehicle power network, or the release of toxic chemicals by the HV battery.

The risk level corresponding to each hazard is classified with Automotive Safety Integrity Level (ASIL) ratings taking into account the exposure, controllability and severity in different driving situations. ISO26262 provides guidelines for ASIL classification. Finally, safety goals are defined in order to prevent the hazards.

6 Investigating Malfunctions and Modelling Cause Effect Chains

Hazards on vehicle level (i.e. in the system environment) are caused by malfunctions of the system under consideration. These malfunctions do not only comprise violations of specified functions, but also any other behaviour that may lead to any top level hazard. Malfunctions at the outer boundary of the system are in turn caused by failures of subsystems, components and so on to the individual constructive elements. Understanding the failure chains is the foundation of the safety concept and resulting improvement of the system design.

Therefore, the investigation of basic failures and failure chains is performed by hierarchical continuation of the System FMEA from Hazard Analysis, on every finer level of granularity of the system architecture. The System FMEA thereby follows the hierarchy of the system architecture, forming a tree with the environment of the examined system as the root element. The next level is the system, followed by its subsystems and components (e.g. sensors or software components). This FMEA tree structure in Fig. 6 exactly corresponds to the hierarchy in Fig. 4.

As the features have been assigned to the architectural elements on every level, the functions for FMEA are easily derived. The connection to the features (requirements

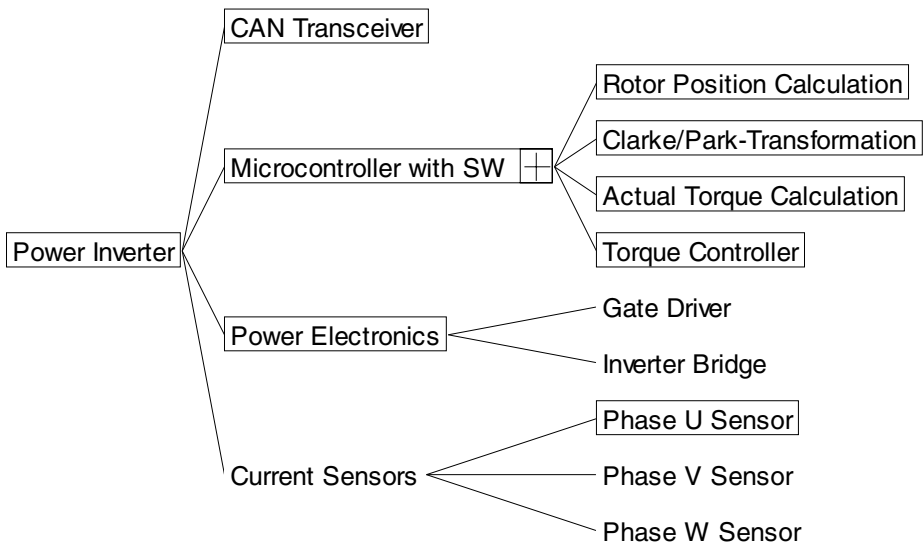


Fig. 6. Excerpt from system structure in IQ-RM

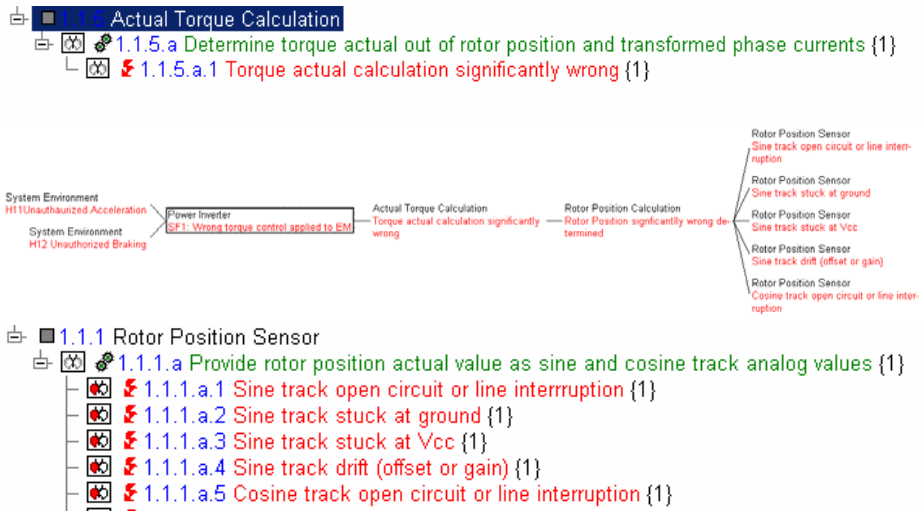


Fig. 7a-c. Definition of function (with input/output relation) and malfunction / Failure Net (excerpt) / Malfunctions on sensor level (rotor position sensor)

headings in DOORS) is maintained by referencing their IDs. As soon as the feature or function assignment has been taken over to the elements of the structural tree in the FMEA, conceivable malfunctions are identified in workshops with experts, including the functional safety expert and the system architect. The system architecture, which explains connections of system components and the functional properties and algorithms, helps understanding the propagation, transformation and mitigation of failure consequences. This is essential to understand the link from individual component failures (like sensor failures) to the top level system failures and finally the vehicle level hazards.

The IQ-RM tool offers the interesting feature of modelling cause-effect chains by connecting malfunctions of system components, thereby forming a network from malfunctions of base elements that lead to top level hazards, which form the malfunction of the root element. Failures of different components can directly be connected in the failure net, even if the components are located on different branches of the structure tree. Following the signal flow as modelled by the internal block diagram (see figure 3), failure propagation if followed along the interfaces between architectural elements on the same level or to the external interfaces of the level above. For a better understanding of the influences between function inputs and outputs it is valuable that the functional correlation has already been recorded during identification of the functions, such as in the example excerpt of a failure net given in Fig. 7a. and Fig 7b.

The search for malfunctions ends at the component level. Here, typical fault assumptions from standards (e.g. SN29500) for electronic components or the standard failure assumptions from ISO 26262 for parts like sensors, busses, microcontrollers etc. are applied to find out the relevant failure modes. The rotor position sensor in our example shows some of the failure modes (truncated list) shown in Fig. 7c.

The malfunctions are then propagated along the signal flow in the failure network. As defining the relation between failures at component outputs and the causes at the component inputs or inside of the component is a structured, but manual process, we are currently investigating the formalisation of the failure propagation modelling by complementary methods of architecture-oriented safety modelling, such as Failure Propagation and Transformation Notation (FPTN) [6], Hierarchically Performed Hazard Origin and Propagation Studies (HiPHOPS) [7] or Component Fault Trees (CFTs) [8].

7 The Safety Concept: Defining and Allocating Safety Measures

The safety concept (divided into the functional safety concept and the technical safety concept) defines the measures to achieve product safety in terms of ISO 26262 with respect to all hazards, as defined in the Hazard Analysis Phase. The measures depend on the identified ASIL.

The starting point for the safety concept is the results of the Hazard Analysis and the preliminary system model with identified cause-effect chains in IQ-RM. The Safety Manager in cooperation with the FMEA moderator extends the system model with safety measures that prevent or control risks or reduce their impact. This includes runtime measures like diagnostics or fallback levels. As the FMEA moderator and the safety manager cooperate in the definition of the measures, the compliance with the ISO 26262 requirements for the applying ASIL is assured. The resulting safety measures are incorporated synchronously into the FMEA and the safety concept. The moderator proceeds with adapting the RPN according to the achieved improvements, whereas the safety manager adds additional safety-related information and obligatory technical target values (e.g. Fault Tolerance Times, hardware-metrics). As quantitative analysis of hazard probabilities is required by ISO26262, Fault Tree Analysis (FTA) is performed in parallel in order to show compliance of the system architecture with these target values. The structure of the fault tree is derived from the hierarchy of the failure nets in the FMEA tool IQ-RM.

Using graphical illustrations is a general recommendation in the Technical Safety Concept in order to allow a better understanding of safety strategies and ASIL decomposition and allocation to system components. Therefore, our methodology uses a combination of ASIL decomposition diagrams, which are tree-like representations of the ASIL decomposition and allocation, and ASIL annotated excerpts from the system architecture. While the first kind of diagram takes profit from the FTA or FMEA structure already produced and synchronised with the system design, the latter one obviously does so by reusing the static system architecture models.

For the Functional and Technical Safety Concept required by ISO26262, we suggest a representation in DOORS for several reasons. Firstly, DOORS supports our approach of hierarchical decomposition: in the same way that failure nets and Fault Trees denote the dependencies of hazards on the component failures, we break down the safety concept from the top level safety goals down to individual technical measures. Secondly, by its linking capabilities, DOORS offers the required

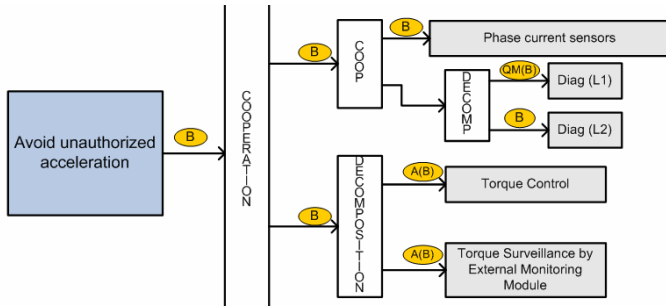


Fig. 8. Schematic decomposition of safety measures with ASIL allocation

traceability from the top level safety goals, associated failures, safety strategies and finally safety requirements on the technical level and further towards test cases. As DOORS forces the user to write atomic statements instead of long phrases of prose text, both traceability and validation are facilitated. Finally, DOORS allows baselining of Safety Concept versions and corresponding requirements sets.

We suggest dividing the functional and the technical safety concept into separate chapters in DOORS and in order to show the relations between corresponding items as DOORS links, as shown in Fig. 9. This allows reviewing the concept just by “clicking along”. The first part of the DOORS module constitutes the functional safety concept and describes safety functions on an abstract level. It relates the safety goals to measures dealing with the top level system failures according to the identified cause-effect chains in IQ-RM. Requirements in this part are rather abstract, claiming for instance measures for certain failure mechanisms, without mentioning the used technical approach in detail. The second part of the DOORS module constitutes the Technical Safety Concept including the detailed Safety Requirements corresponding to the chosen realisation. They are rephrased respecting the common guidelines for requirements wording in order to assure that they integrate smoothly into the rest of the requirements set. For example, most developers are unfamiliar with concepts like “Safe State”, but they usually do understand a requirement like “When the temperature exceeds 80°C, the PWM of the motor shall be switched off within 50ms.” Besides hardware and software requirements, some requirements additionally deal with external requirements (allocated to components in the system environment), measures in “other technologies” or organisational measures.

The safety requirements are finally inserted into the requirements process by copying into the existing set of system, hardware and software requirements (that are usually located in separate DOORS modules), linking them automatically to the Safety Concept and moving them at the appropriate positions. For instance, a requirement for redundant temperature measurement is placed along with other requirements regarding temperature measurement, but marked as a safety requirement by its ASIL attribute.

ISO 26262 requires all safety requirements to be covered by test cases. Monitoring of test coverage can be achieved by tracking test status information in DOORS, e.g. by tables recording the test results together with date, software and hardware version, test equipment etc. of the last test run for each test case, where the test cases in turn

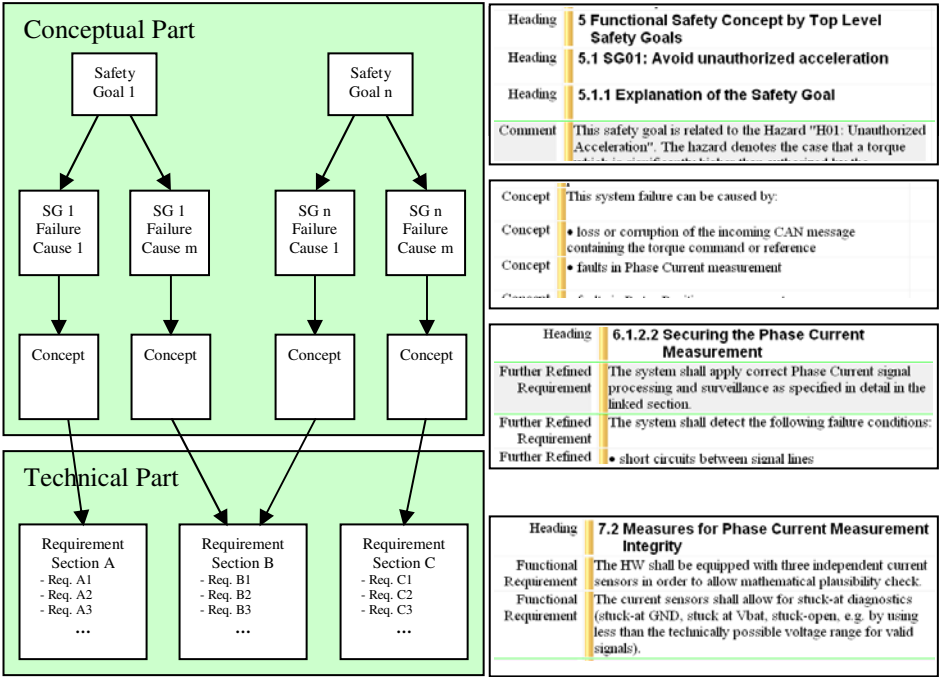


Fig. 9. The safety concept in DOORS (schematic and example)

are linked to the requirements in DOORS. Even traceability between DOORS and textual test specifications located in an external version control system is assured by external links based on URLs. With the help of the links and some scripts, we managed already the automation of most tasks in DOORS, e.g. propagating ASILs along links or calculating metrics for test coverage.

The allocation of the safety requirements to architectural elements of the system is performed as described above for feature allocation, resulting in a new iteration of architectural redesign and feature allocation. The mapping of features to architectural elements and the placing of safety requirements close to other requirements affecting similar features help to quickly identifying the affected architectural elements. In the same way as an ASIL DOORS attribute marks the safety requirements, ASIL tags are applied to elements of the system architecture, showing which components are safety critical. This helps the developers applying the required development process measures to these components.

8 Conclusion and Outlook

The methodology described in this paper has been applied in two development projects in the area of hybrid and electric vehicles at a big automotive electronics supplier. First experiences show that it takes some time to achieve a common

understanding of the process. Throughout the common workshops, lots of inconsistencies and duplicated work became obvious, because in the past, the Safety, FMEA, and System specialists were all creating models for their own purposes. So, the first benefit of the methodology was gaining a common understanding just by gathering the people around one table on a regular basis.

In the pilot projects, the work of aligning the models with each other was a purely manual task, thereby very time-consuming and putting the acceptance of the new method at risk. It is certain that more benefit will be achieved by tool integration, which is currently under preparation. A first piece of tool-based integration has already been achieved by the mere fact that the Functional and Technical Safety Concept were written in DOORS, unlike in former projects, where textual documents were used. The advantage of having navigable links from safety measures to the related hazards and failures, as well as linking product requirements and subsequently test cases to the decided safety measures was a significant advantage at reviews and when changes became necessary. Regarding the tool-based allocation of requirements to architectural artefacts, trials have been made for the software sub domain with DOORS and Enterprise Architect (EA). So far, experience has shown some problems in practice, mainly with assuring consistency after several change cycles and the lack of acceptance for the pure UML notation used in the pilot projects by hardware and system engineers. Furthermore, the behavioural modelling of continuous data flows, which are very common for control algorithms, is a weak point of the UML notation in general. Switching to SysML or using Simulink models for some of the aspects is expected to bring some advantage regarding these issues. We are currently carrying out further research activities in order to formally allocate requirements to architectural elements within the modelling tool, and also to automate the merely mechanical part of transforming the hierarchical block architecture into a tree structure inside IQ-RM, which is on modelling tool side already supported by the XMI export features, but will probably require support from the manufacturer of IQ-RM, e.g. by providing enhanced XML interfacing facilities.

Nevertheless, the applicability of the method could be demonstrated in the pilot projects even by using informal block diagrams drawn in Visio. Yet, most of the involved experts agreed that in the end they had a better understanding of the system and a broader agreement on the safety measures commonly decided. Inconsistencies that popped up in other projects at safety milestone assessments late in the development cycles could be significantly reduced. The work products of the Functional Safety lifecycle were available earlier than in past projects and shared more of the vocabulary used by the developers, so that the influence of safety engineering on the actual product was faster achieved and more systematic, avoiding expensive extra loops for redesign up to now. Due to the clearer structure of the documents, a partial reuse for upcoming projects seems promising.

The final target of our efforts is seamless traceability from input requirements via architecture and implementation to test results, linked by the Safety Concept to analysis methods such as FMEA and metrics, and all managed from an ergonomic common user interface. But even today, following the described approach manually will help the involved companies saving time and costs and reducing the risk of safety issues in the final products.

References

- [1] Hull, E., Jackson, K., Dick, J.: Requirements Engineering. Springer, Heidelberg (2004)
- [2] Weilkens, A.: Systems Engineering mit SysML/UML: Modellierung, Analyse, Design. dpunkt Verlag (2009)
- [3] DIN EN 60812: Analysetechniken für die Funktionsfähigkeit von Systemen – Verfahren für die Fehlzustandsart- und -auswirkungsanalyse (FMEA) (November 2006)
- [4] VDA: Sicherung der Qualität vor Serieneinsatz – System-FMEA, 1. Aufl. (1996) ISSN 0943-9412 (ersetzt durch 2. Auflage 2006)
- [5] Redmill, F., Chudleigh, M., Catmur, J.: System Safety: HAZOP and Software HAZOP. John Wiley & Sons, Auflage: 1. Auflage (April 14, 1999)
- [6] Fenelon, P., McDermid, J.A., Nicholson, M., Pumfrey, D.J.: Towards Integrated Safety Analysis and Design. ACM Applied Computing Review 2(1), 21–32 (1994)
- [7] Papadopoulos, Y., McDermid, J.A.: 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) ISBN 3-540-66488-2, ISSN 0302-9743
- [8] Kaiser, B., Liggesmeyer, P., Mäkel, O.: A new component concept for fault trees. In: Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software (SCS 2003), Canberra Conferences in Research and Practice in Information Technology, vol. 33 (2003)

Aspect-Oriented Implementation of Fault Tolerance: An Assessment of Overhead

Ruben Alexandersson, Peter Öhman, and Johan Karlsson

Department of Computer Science and Engineering,
Chalmers University of Technology
Goteborg, Sweden
{ruben,peter.ohman,johan}@chalmers.se

Abstract. Software-implemented fault tolerance is an attractive technique for constructing fail-safe and fault-tolerant processing nodes for road vehicles and other cost-sensitive applications. This paper investigates the memory consumption and execution time overhead obtained when implementing time-redundant execution and control flow checking in software for an automotive brake controller application. These two mechanisms were implemented at the source code level using three implementations techniques: aspect-oriented programming (AOP), source code transformation and manual programming in C. The results show that AOP generates much higher overheads than code transformation for strictly systematic implementations. On the other hand, when application knowledge is used to optimize the implementations, the overhead of AOP is similar to that of manual programming in C.

Keywords: Fault tolerance; Aspect-oriented programming; Source code transformation; Time-redundant execution; Control flow checking.

1 Introduction

A key concern in the design of safety-critical computer systems for road vehicles and other cost sensitive applications is to minimize the cost of redundancy. Since the system cost largely depends on the number of electronic control units (ECUs) in a system, solutions that rely on distributed voting and massive replication of ECUs to achieve fault tolerance and fail-safe behavior are usually considered too expensive in such systems. The preferred solution is instead to use ECUs that are self-checking and internally fault tolerant.

A cost-effective way of implementing self-checking and fault tolerant ECUs is to use software-implemented fault tolerance. Numerous software techniques for dealing with design faults as well as physical hardware faults have been proposed over the last thirty years. This paper focuses on software techniques for tolerating hardware faults, and especially the memory consumption and execution time overhead imposed when implementing such techniques at the source code level.

In safety-critical systems, where it is imperative to achieve high fault coverage, software techniques are typically used as a second line of defense to deal with errors

that escape fault tolerance mechanisms implemented at the hardware level. (Examples of such hardware mechanisms include the use of fault tolerant latches, protection of caches with error correcting codes, protection of main memory with a memory management units and error correction codes, etc.) To achieve high fault coverage, it is common to use a combination of software techniques such as software assertions, control flow checking and time-redundant program execution.

While the use of these techniques allow a designer to reduce the number of ECU in a system and thereby the hardware cost, they may impose a significant memory consumption and execution time overhead. Many techniques for software fault tolerance require three or four times the performance of the computer node as compared to running the application without fault tolerance support, e.g. [1].

Hence the impact of fault tolerance implementations on system performance can be large. As we will show in this paper, the overhead associated with software-implemented fault tolerance depends not only on the mechanisms themselves, but also their implementation.

In this paper we investigate the overheads associated with three different methods for implementing fault tolerance in software: source code transformation, aspect oriented programming (AOP), and manual implementation in C. The main purpose of this study is to determine the overhead impacts of using AOP in this context and to compare it with the other methods.

In common for all the methods are that they operate on source code level manipulating the target program before it is compiled. Working on source code level has the advantage of the techniques being hardware independent. That is the tools used do not need to be ported to each new hardware platform. Hence the software can be produced using mature and certified compilers. Another advantage that we explore in this study is the possibility to adapt the fault tolerance implementation to the target program at hand. One way to reduce the overhead is to utilize application knowledge in order to use more application-specific fault tolerance mechanisms, or to apply systematic mechanisms only to the parts of the software that are critical. Programmer knowledge about the specific implementation can also be used to reduce overhead.

A short introduction to AOP is given in section 2. Section 3 describes the setup for the evaluation, i.e. the target system, the fault tolerance mechanisms, and the different implementations. Section 4 presents the experimental results obtained. Section 5 gives an overview of related work, and in section 6 the results are discussed. Conclusions are drawn in section 7.

2 Aspect-Oriented Programming

Any normal industrial-size software system is a combined implementation of multiple functionalities. A typical program consist of the basic functionality (the base program) and a number of additional functionalities, such as diagnostics, fault tolerance, data persistence and so on. In the AOP vocabulary both the basic functionality and the supporting functionalities are referred to as concerns.

The system design and modularization are chiefly conducted to reflect the basic functionality of the application. The effect of this is that all other concerns cannot

be modularized in a good way. Hence, the source code implementing other concerns is scattered throughout the program modules. These concerns are therefore called crosscutting concerns, since they crosscut all (or many) of the program modules. As an example, control flow checking [2] affects every function of every module of the application program and is therefore scattered throughout the implementation. Although this mechanism can syntactically be defined with just a few lines of code, object-oriented techniques require that a very large amount of source code be added throughout the entire software.

AOP is an approach that can be used to overcome this problem by separating the implementation of a concern from the base program. Thus AOP allows modularization of functionalities that are difficult to modularize in other programming languages. AOP allows the programmer to implement any (or all) concerns in a loosely coupled fashion and to combine these implementations with the rest of the program to form the final system.

An aspect-oriented implementation of a crosscutting concern consists of two parts, the actual implementation of the functionality associated with the concern and the information on how that code should be integrated in the rest of the program. Any traditional language such as C or C++ is well suited for the first part. However, the traditional languages lack primitives for specifying how the concern-specific code should be composed, or woven, together to form the final system. An AOP language therefore defines a way to specify rules for composing different implementation pieces together. The AOP language is then built as extensions to traditional languages to give them aspect-oriented capabilities, in the same way that C++ was built as an extension to C to provide object-oriented capabilities to the C language. An AOP language compiler is most often implemented as a source-to-source compiler that weaves the different concern-specific sources together, and then the base language compiler is used to compile the result into executable code.

This work is directed towards the domain of road vehicles. The overall predominant programming language for safety-critical applications in this domain is C and to some degree C++. Today there exists only one mature AOP language that extends C and C++, namely AspectC++ [3]. AspectC++ is an extension to C++ but since C is a subset of C++ it can also be used together with C programs. In two earlier studies [4,5] we evaluated AspectC++ for fault tolerance purposes. We also extended the language and optimized the language weaver in order for it to be fully feasible. This extended version of AspectC++ is the one used for the study presented here. Thus when AspectC++ is discussed in this paper, this is the version referred to, rather than the official release.

Some terminology related to aspect-oriented programming is used in this paper. The main characteristic of an AOP language is, apart from what base language it extends, what *joinpoints* it supports. A joinpoint is an accessible point in the application execution where concern-specific tasks can be performed. An AOP language lets the programmer declare a *pointcut* that accesses a set of joinpoints. The pointcut can then be linked to a concern-specific code segment called *advice*.

3 Evaluation

A prototype Brake-By-Wire application developed by Volvo Technology was used as target system for the evaluation. The system is a brake-controller running on a single-board computer using the MPC565 microcontroller from Freescale. The software consists of three parts: an ABS controller, a global controller for wheel node synchronization, and a small scheduler. Apart from the scheduler, the software was produced using Simulink from Mathworks, Inc. and the source code in C was generated with Targetlink from dSPACE GmbH.

Two different mechanisms for fault tolerance were implemented on the target application, namely time-redundant execution [6] and control flow checking [2]. Time-redundant execution is a technique for detecting and sometimes masking transient faults through repeated execution. An error originating from a transient fault can be detected by executing a function two times and comparing the results. If the function is executed a third time, the fault can be masked by voting between the three runs. Here both of these setups are used. In the systematic versions of the program, functions are executed three times in order to tolerate faults. For the versions where application knowledge is utilized, only two runs are used for fault detection since recovery is done through other means. When using time redundancy as an intra thread technique one must ensure that each run start with the same initial state. We achieve this by replicating global data in memory and having each run execute on its own redundant data set. Another approach is to store the initial state in a checkpointing cache before the first run commences and then restore it from the cache prior to each of the subsequent runs. We made initial investigations of this approach and found that it generated higher overheads than our approach. Another disadvantage of this approach is that the checkpointing cache must be fault tolerant.

Time-redundant execution can detect and mask data errors and many program flow errors. However, there are some program flow errors that cannot be handled by time-redundant execution, e.g., those that cause the processor to skip the comparison (or voting) of the redundant results. To provide coverage of such errors, we use software implemented program flow checking as a complement to time-redundant execution. The aim of the program flow checking mechanism is to detect illegal branches, i.e., branches that are not defined in the machine code. It divides a program into a number of blocks and inserts checks which verify that an entry to a block is eventually followed by an exit from the same block. Two identical identifiers that are unique to each block are placed at the beginning and at the end of the block. When the execution enters a block, the first identifier is stored, and when it exits the block, the stored identifier is compared with the second identifier. If they do not match, a control flow error has been detected. A code block can be defined with different granularity. Here function bodies are used as blocks.

The purpose of this study is to compare the overheads caused by using AOP for implementing fault tolerance with that of other alternative techniques. AOP languages bring benefits when implementing fault tolerance from a programmer or software engineering perspective, compared to using pure C or C++. However, when using higher-level languages, what is gained in less programmer effort is normally paid by having more resource-demanding programs. A program written in C++ is generally larger in terms of execution time and memory consumption than the same program

written in C. When using an even higher-level language, like AspectC++, additional overhead compared to C or C++ can be expected. This is due partly to the structure of the code generated by the language compiler, but also to the fact that the programmer has less freedom than when using a lower-level language and therefore might have to use less efficient algorithms. Therefore, when evaluating the cost of using AOP the resource consumption should be compared against standard C implementations.

An alternative to using a higher-level language such as AspectC++ is to use a code transformation tool that introduces fault tolerance automatically. Such a tool does not bring any extra performance cost since it can freely manipulate the code in the best known way. However, being a fully automated tool it is restricted to using predefined transformations and hence does not benefit from knowledge about the specific target application or implementation. Thus, compared to using fully automated transformations, there is both a performance cost and a potential for performance gain associated with using AOP for implementing fault tolerance. Here the question is which is the largest, or whether performance-wise it will ultimately be beneficial or not to use AOP. There can be no truly general answer to this since it depends on the target application. One can certainly construct a program so that knowledge about it will not help in producing more efficient code, and hence there will only be a cost associated with AOP. However, in order to acquire relevant data the method was tested on a real-life application provided to us by our industrial partners. This gave a good insight into the performance impact of AOP in the application domain at hand. See also section 6 for a discussion of how AOP can be a feasible solution both when there are potential performance gains and when there are not.

As an alternative to the above mentioned techniques one can also consider using hybrid solutions such as highly configurable transformation tools or domain specific AOP languages. However, such techniques are outside the scope of this study.

3.1 Source Code Level Transformations

In this study, we have implemented a basic form of triple time-redundant execution with voting using source code level transformation. There are several other approaches for implementing fault tolerance in software via source code transformations that impose lower memory and execution time overhead than our technique. However, these approaches rely on specific fault assumptions to reduce the overhead. For example, the techniques presented in [7] aim to tolerate data errors in memory, but cannot handle data errors occurring in processor registers.

One way to reduce the time overhead of triple time redundant execution is to execute the third run only if there is a mismatch between the results produced by the first and second run. This technique can significantly reduce the time overhead for stateless applications. However, for our brake-by-wire application, which is a stateful application, we found that this approach gave a longer execution time than using three runs. This was mainly due to the cost associated with verifying the correctness of the complete state, rather than only the output. This is needed if the states of the first and second runs are used for updating the state of the third.

Using AOP for fault tolerance is only feasible under the assumption that the code is stored in protected memory. This is because static replication of code is not a feature of programming languages. This is a both common and justified assumption, since hardware level mechanisms for protecting static memory are well known. The cost of

having protected memory is also lower than the alternative of tripling the size of the static memory circuit. Hence, from a cost perspective, this is the preferable approach. However, almost all papers on transformation use code duplication. As far as we know, there exists only one earlier work where time-redundant execution of the same code has been applied as an intra-thread technique using code transformations [8]. This work applies time redundancy only on functions that do not access global variables. Other functions are duplicated statically. Since the Brake-by-wire application uses global data in most functions, a comparison where the transformed implementation duplicates most code would not be just. We therefore also make a contribution to the transformation technique by providing a solution to using time redundancy when global data is used. This is done by having three data sets, one for each run.

There are two things to consider when introducing redundant data in a time-redundant setup. First, since the same code is used for all runs, the data associated with each run should not be accessed by different names. Second, since one set of global data will be copied to another in the case of an error, it helps to have them structured in a way that makes the copy code fast and compact. All global variables of a given type are therefore transformed into one single global array. One axis of the array has the same length as the number of global variables in the original program of the given type, and the other has the length three holding the three sets of data needed for the three runs. Fig. 1 shows an example of the original and transformed code for a small program slice. To index and access the correct data in the array, a new variable called *current_run* is introduced. It holds the information on which run is currently executing and hence which data to access. Apart from this, the transformations used for the time-redundant execution are simple and consist of replicating input data to all data sets, verifying equality of outputs and initiating the three runs. The complete transformation scheme also includes recovery code that cleans up the data sets in case of a mismatch, and code for the control flow checking mechanism that is inserted both at the beginning and end of functions and around function calls. For this study we have not implemented the above scheme in a tool but have done the transformations manually aided by a search and replace tool.

<pre>int a,b; void work(){ a++; b++; }</pre>
<pre>int int_array[2][3]; int current_run; void work(){ int_array[0][current_run]++; int_array[1][current_run]++; }</pre>

Fig. 1. Global data transformations

3.2 Manual Programming in C

In order to get a second reference point to the overhead associated with AOP we also implemented a version by hand using standard C. This was done in order to get an indication of how expensive the use of AOP is, compared to what we thought the optimal way if only hardware resource consumption is considered.

In contrast to code transformation, manual programming allows us to apply application and implementation specific knowledge to reduce overhead. The target program implementation uses 38 global variables. However, we know that only a subset of these actually carries information between different iterations of the control loop, and hence is global from the perspective of the time-redundant execution mechanism. Of the 38 variables, 14 or 38% are either input, or output, or hold a controller state that is preserved till the next iteration. Thus these 14 are the only ones that have to be redundant, and whose read and write operations have to be explicitly handled by the mechanism. The others can be considered local and all runs can operate on the same data set.

Regarding the application, we know that it is a brake-by-wire system that controls a mechanical brake at a wheel node. As such it is inherently robust to some faults. That is, a small deviation from the correct output during a short time period will not lead to a catastrophic failure. It might not even be observable as a change of brake force at the mechanical brake [9]. What must be avoided is that an error remains in the system causing the software to produce incorrect output for a significant time period spanning several control loops. Hence a less costly recovery mechanism known as best effort recovery [10] can be used. Therefore the loop is executed only two times to detect an error. If an error is detected, no new output is produced but the output from the former iteration is kept. This will keep the signal to the brake on the same level and intended changes will be delayed one cycle, which is acceptable. The state of the software is then rolled back to the state it held at the earlier iteration. Thereby one less execution of the loop is needed; however, a third data set must still be maintained and updated in order to support the rollback.

The code transformation scheme moves all global data of a given type into a single array. As the results presented in section 3 shows this way of organizing data helps the compiler to produce very effective code. The downside is that the code also becomes difficult to understand for a human reader. Hence, this technique is suitable for automatic code transformation but not as a coding standard for human programmers. Also the advantage of this approach is reduced when only part of the data is included in the array. This is the consequence of applying our knowledge about the implementation as described above. Therefore for hand written implementation all data is not stored in a single large array but each variable that need to be redundant is replaced by its own array of length two or three.

3.3 Systematic Implementation Using AspectC++

Two different versions of the program were implemented using our extended version of AspectC++. First, AspectC++ was used in the same way as the code transformation scheme. That is, no application or implementation knowledge was utilized in order to optimize the implementation. AOP implementations of several fault tolerance mechanisms including time-redundant execution and control flow checking are presented in [4]. The control flow checking implementation used in this study is similar to the one

presented there with the exception that no stack is used. Since only faults circumventing the time redundancy mechanism need to be detected the mechanism is applied only to that level of the function call tree. Hence, a simple variable is enough to store the function signatures. The time-redundant execution implementation differs more from the one presented in [4] and is thus briefly described below. The figures only show a simplified view of the advice code of the implementation. Aspect definition, pointcut declarations, slices, and error detection and recovery code are omitted. The presented advice code is, however, enough to understand the principle behind the implementation.

Fig. 2 shows the call advice that is executed each time a call to the control loop is made. In case of a detected error in the former, iteration code that cleans up the erroneous state is executed. Otherwise the control loop is called three times and the *current_run* variable, that here is part of the aspect, is updated between each run.

```
advice call(loop()) : around() {  
    if (errorcode > 1) {...} else {  
        current_run = 0;  
        tjp->proceed();  
        current_run = 1;  
        tjp->proceed();  
        current_run = 2;  
        tjp->proceed();  
    }}
```

Fig. 2. Call advice implementation for the time-redundant execution mechanism

The algorithm in Fig. 2 is identical to the one used with the code transformation scheme. However, with AspectC++ one cannot, as when using code transformations, manipulate the code freely and join global variables into large arrays. AspectC++ does have a feature called slicing that allows for introduction of new fields in classes and structs. Our extended version of AspectC++ also supports slicing on primitive variables. Hence, slicing can be used to associate variables holding the redundant data with the variables of the target program. However, this means that read and write operations on global data have to be surrounded by switch statements targeting the operation on the correct data set, instead of using a simple index variable.

Fig. 3 shows the implementation of the get advice that returns data from the correct data set whenever a variable is read. There is also a set advice that handles write operations which is not shown here. However, it follows the same structure as the get advice.

Apart from these advices, the time-redundant execution implementation also requires a check that verifies the equality of the results of the three runs. This is done by a get advice that is triggered when an output variable is accessed outside the control loop. The implementation is similar to the one in Fig. 3, but instead of simply returning the value from the correct data set, the values of all three copies are compared. The correct value is then returned and, if needed, an error code is set in order to signal which data set is erroneous.

```

advice get(variables()): around() {
    switch (current_run) {
        case 0:
            tjp->proceed();
            break;
        case 1:
            *tjp->result() =
            ((JoinPoint::Target*)tjp->target())->value2;
            break;
        case 2:
            *tjp->result() =
            ((JoinPoint::Target*)tjp->target())->value3;
            break; }}

```

Fig. 3. Get advice implementation for the time-redundant execution mechanism

3.4 Application Specific Implementation Using AspectC++

In the application specific version of the AspectC++ program we rely on application and implementation knowledge to reduce the number of replicated variables and to implement best-effort recovery, as we do with the hand written implementation described in section 3.2. In this implementation, the control loop is executed two times. Hence the call advice initiates two runs and the get and set advices handles two data sets. The call advice also contains code for storing the state of the previous program iterations and the recovery code used by the best effort recovery mechanism.

4 Experimental Results

As described in section 3, four fault tolerant versions of the target program were implemented; one using a source code transformations scheme, one written by hand in C, and two using AspectC++. GCC v4.2.2 was used for compiling all different program versions, both with low and high optimization. In low optimization, only the `-finline` option was set. The reason for using this option is that AspectC++ produces extremely ineffective code if the inline statements added are not honored by the compiler. Hence a comparison without this option would not do AspectC++ justice. High optimization equals the GCC option `-O3` except that `strict-aliasing` was turned off. This is because the option caused some versions of the program to behave incorrectly.

We measured the hardware resource consumption for all fault tolerant implementations as well as the unmodified target program. The space in bytes used on the on-board flash memory is labeled code in the tables but includes, apart from program code, also constants, and initial values to dynamic variables. The memory usage for data is measured dynamically and includes data segments, stack, and heap. Execution time is approximated with the number of traced instructions in the longest path through a complete control loop. All values are measured without the presence of faults.

Table 1 shows the resource consumption data for the strictly systematic implementations, i.e. where knowledge about the application and implementation were not used

Table 1. Experimental results for systematic implementations

	Code (byte)		Data (byte)		Trace (#instr)	
	#	%	#	%	#	%
Low optimization						
No fault tolerance	4252	0%	509	0%	790	0%
Code transformation	8900	109%	828	63%	3893	393%
AspectC++	32744	670%	2144	321%	6394	709%
High optimization						
No fault tolerance	3876	0%	333	0%	294	0%
Code transformation	4476	15%	692	108%	976	232%
AspectC++	14168	266%	712	114%	1642	459%

to reduce overhead. First data for the brake-by-wire application without any added fault tolerance is given. Second is the resource consumption for the transformation scheme and the overhead in % compared to the implementation without fault tolerance. Notable is that the compiler optimization is able to significantly reduce the overhead to the level that the extra code only give 32% additional overhead apart from the 200% expected from executing the loop three times. This is due to that the transformation scheme moves all global data into a single array. Structuring the data in this way enables GCC to optimize the code very effectively. Notice also that the data overhead is quite low although the global data is tripled. The reason is that the stack consumption is not significantly higher.

The last rows show the result obtained when using our extended version of AspectC++. The overheads are significantly larger than for the transformation scheme. One reason is the extra glue code inserted at each joinpoint by the aspect weaver. However, most of the glue code is removed by GCC if the high optimization level is used. The difference that remains is because of the different ways of storing and accessing data as described above. The dynamic memory overhead for the aspect-oriented implementation is 321% with the low level of compiler optimization. When using compiler optimization it is reduced to 114% which is nearly as good as for the transformation scheme. The reason is that the code generated by the weaver gives a high stack consumption overhead that GCC can completely remove if allowed to. Hence we recommend having this optimization turned on whenever AspectC++ is used.

Table 2 shows the resource consumption data for the implementations where knowledge about the application and implementation were used to reduce overhead as described in section 3.2. Comparing the data for the two implementations reveals some surprising results. As expected, the hand-written implementation does achieve the best results when no compiler optimization is turned on. However, with the high optimization level the results are different. The number of executed instructions as well as both types of memory consumption is noticeably higher than for the aspect oriented implementation. It seems that the structured way in which the weaver introduces the advice code in the target program helps GCC to optimize the implementation in a more

Table 2. Experimental results when using target system knowledge

	Code (byte)		Data (byte)		Trace (#instr)	
	#	%	#	%	#	%
Low optimization						
No fault tolerance	4252	0%	509	0%	790	0%
Programming by hand in C	6783	60%	596	17%	1865	136%
AspectC++	11632	174%	972	91%	2396	203%
High optimization						
No fault tolerance	3876	0%	333	0%	294	0%
Programming by hand in C	5681	47%	524	57%	746	154%
AspectC++	5344	38%	444	33%	721	145%

efficient way. It could be argued that the hand-written implementation should have been written in the same way and thus achieve the same result. However, although possible, writing the code in this way is not natural for a human programmer. Also this would yield higher overheads in the case where GCC optimization is not used. It is well known that compiled C programs are usually more efficient than human-written assembler programs. What we see here is a case of the same property being observed in higher-level languages. The conclusion that a good AOP weaver produces more efficient code than a human programmer can naturally not be drawn from a single data point. It is, however, interesting to observe that this can be the case and that using AOP can actually be a saving and not a cost in hardware resource consumption even compared with a lower-level language. It is also worth noting that the execution time and dynamic data consumption are lower for the aspect oriented fault-tolerant implementation when using GCC optimization, than for the implementation without fault tolerance when no optimization is used. This means that adding fault tolerance in fact gives less overhead than not using compiler optimization.

When comparing the results from the two tables we see that even though the transformation scheme is much better than the AOP implementation when only using systematic mechanism for fault tolerance, the fact that AOP allows for adaptation to the target program at hand makes it possible to reduce the overhead to significantly lower levels. Another observation that can be made is that compiler optimization affects the results for the systematic implementations more than it does for the application specific ones. This is true regardless if AOP is used or not. This implies that it is important to consider compiler optimization when conducting comparative studies on source code level mechanisms for fault tolerance as we have done here.

5 Related Work

Some work has been done about performance of aspect-oriented programs, also with regard to fault tolerance. In [11] AOP is used to move some functionality from the FT-CORBA middleware to the application while still retaining transparency to the

application programmer. This is shown to reduce the overhead induced by the moved functionality. Apart from the different domain and approach to fault tolerance, this work also differs from the one presented here in that the target application is a Java program and the aspect-oriented language used is AspectJ.

Afonso et al. [12] evaluate the performance impact of using AspectC++ for implementing fault tolerance. This is done by applying thread-level recovery blocks to a target application. In contrast to us, they do not explore the power of AOP to reduce overhead, but apply the technique in a less flexible yet also less costly way. Their results may look contradictory to ours since they measure significantly lower overheads when using AspectC++ than the ones presented here. Some differences are explained by the different way of obtaining data. They measure CPU utilization in a threaded environment running on a real-time OS, while we count number of executed instructions in the application itself. Our relatively higher dynamic memory overheads for the AOP implementations are due to higher stack consumption, which they do not include when calculating memory footprint. However, the big difference comes from our different fault tolerance mechanisms and from the level these are applied at. Those authors use recovery blocks, which is a mechanism that itself cause much lower overhead than the ones we use. Moreover, recovery blocks can be implemented in conceptually the same way whether AOP is used or not. This is true regardless of whether it is applied at thread level or at function level. In our case with the time redundancy mechanism a less efficient algorithm for data access has to be used for the AOP implementation when applied at function level. Choosing to apply a mechanism at function level as done here, or thread level as they do, has a great impact in a number of ways including overhead. This is further discussed in section 6.

There has been quite a lot of different work done on code transformations for fault tolerance. The work most closely related to the transformations presented in this paper is by Oh et al. [8]. To the best of our knowledge this is the only work that uses time redundancy rather than code duplication as the underlying mechanism. In their case this is done as a means to reduce energy consumption in portable devices. We extend their ideas to include functions that operate on global data and present a transformation scheme that solves this problem. Applying our ideas to their work would probably further improve their results.

6 Discussion

Fault tolerance mechanisms when applied at function or statement level have a very high level of code tangling. The systematic implementation presented here pointcuts nearly every single joinpoint in the execution graph supported by the language. This naturally means that performance overheads associated with AOP have a very big impact. But it also means that the benefits, from a programmer perspective, of using AOP are probably greater than for any other domain. In this paper we have shown that although the overheads cannot be neglected, the power of the AOP paradigm enables us ultimately to reduce the run-time cost most significantly. This, however, is based on properties of the target application used for the experiment. Although the ideas presented in this paper are based on the assumption that there are commonly properties in an application that can be used to reduce the overhead, it is safe to assume that this is not always the case. As can be seen in Table I, the cost of applying

the mechanisms with full coverage is still so large that it can be hard to justify. Hence the technique proposed here is most feasible when knowledge can be and is applied. On the other hand, applying fault tolerance mechanisms at thread level as done by Afonso et al. [12] gives much less tangling and hence equally lower overheads due to AOP. For instance, if the time redundancy mechanism used in this paper is instead applied at thread level executing each run in its own thread, global data access does not have to be monitored at all, because each thread has its own data set. This would reduce the extra overhead due to AOP to the same negligible levels as shown by Afonso et al. The drawback is that one cannot, as done here, freely use implementation knowledge to minimize resource consumption. Our conclusion is thus that a flexible fault tolerance framework should allow for both ways. As the default approach, thread level mechanisms could be applied, and when possible or needed one can reduce overhead by replacing them with optimized function level mechanisms. AOP would serve well as the underlying technology for such a multi-layer framework.

7 Conclusion

This paper has compared the memory consumption and execution time overhead caused by three different methods for implementing fault tolerance on the source code level: code transformation, aspect oriented programming (AOP), and manual programming in C. Two fault tolerance mechanisms – time redundant execution [6] and control flow checking [2] – were implemented for a brake-by-wire application.

First the mechanisms were implemented in a strictly systematic manner, thus allowing them to be automatically applied to the target program without programmer intervention. This can be achieved both with code transformation and with AOP. The results showed that code transformation gave much lower overheads than AOP for these implementations. The main reason for this result was that the presented code transformation technique handles replication of global variables very efficiently.

The two fault tolerance mechanisms were also implemented using application and implementation specific optimizations. Utilizing these optimizations requires that the programmer is able to manually fine-tune the code. This is possible in AOP and manual C programming, but not when using code transformation.

In general, the application specific implementations imposed much less overhead than the systematic ones. An important observation is that the overhead for AOP was comparable or even less than that for the hand coded C-implementation.

The main advantages of using AOP for implementing fault tolerance are a reduction of development and maintenance cost through the separation of concerns. Our work shows that it is possible to utilize these advantages without imposing higher overheads than those obtained when implementing the fault tolerance mechanisms directly in C. Further research is however needed to determine if our observations also hold for other applications.

Acknowledgments. This research was conducted within the CEDES (Cost Efficient Dependable Electronic Systems) project, which is funded by the Swedish industry and government joint research program IVSS – Intelligent Vehicle Safety Systems. The

authors thank Volvo Technology and Martin Sanfridsson for providing the Brake-By-Wire application.

References

1. Nicolescu, B., Savaria, Y., Velazco, R.: Software Detection Mechanisms Providing Full Coverage Against Single Bit-Flip Faults. *IEEE Transactions on Nuclear Science* 51(6), Part 2, 3510–3518 (2004)
2. Oh, N., Shirvani, P., McCluskey, E.J.: Control-Flow Checking by Software Signatures, Center for Reliable Computing, Stanford Univ., CA, CRC-TR-00-4 (CSL TR num 00-800) (2000)
3. Spinczyk, O., Gal, A., Schröder-Preikschat, W.: AspectC++: An Aspect-Oriented Extension to C++. In: 40th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific 2002), Sydney, Australia, pp. 18–21 (2002)
4. Alexandersson, R., Öhman, P.: Implementing Fault Tolerance Using Aspect Oriented Programming. In: Bondavalli, A., Brasileiro, F., Rajsbaum, S. (eds.) LADC 2007. LNCS, vol. 4746, pp. 57–74. Springer, Heidelberg (2007)
5. Alexandersson, R., Öhman, P.: On Hardware Resource Consumption for Aspect-Oriented Implementation of Fault Tolerance. In: 8th European Dependable Computing Conference, Valencia, Spain, April 28-30 (2010) (to appear)
6. Damm, A.: The effectiveness of software error-detection mechanisms in real-time operating systems, FTCS Digest of Papers. In: 16th Annual International Symposium on Fault-Tolerant Computing Systems, Washington, DC, USA (1986)
7. Rebaudengo, M., Reorda, M.S., Violante, M.: A new approach to software-implemented fault tolerance. *Journal of Electronic Testing: Theory and Applications* (20), 433–437 (2004)
8. Oh, N., McCluskey, E.J.: Error detection by selective procedure call duplication for low energy consumption. *IEEE Transactions on Reliability* 51(4), 392–402 (2002)
9. Skarin, D., Karlsson, J.: Software Implemented Detection and Recovery of Soft Errors in a Brake-by-Wire System. In: 7th European Dependable Computing Conference (EDCC-7), pp. 145–154 (2008)
10. Vinter, J., Johansson, A., Folkesson, P., Karlsson, J.: On the design of robust integrators for fail-bounded control systems. In: International Conference on Dependable Systems and Networks 2003, San Fransisco, USA, pp. 415–424 (2003)
11. Szentivanyi, D., Nadjm-Tehrani, S.: Aspects for improvement of performance in fault-tolerant software. In: 10th IEEE Pacific Rim International Symposium on Dependable Computing, pp. 283–291 (2004)
12. Afonso, F., Silva, C., Brito, N., Montenegro, S., Tavares, A.: Aspect-Oriented Fault Tolerance for Real-Time Embedded Systems. In: ACP4IS 2008: 7th workshop on Aspects, components, and patterns for infrastructure software (2008)

System of Systems Challenges

Hermann Kopetz

Institute for Computer Engineering
Vienna University of Technology, Austria
hk@vmars.tuwien.ac.at

Abstract. The available technology (e.g., the Internet) makes it possible to interconnect *independently developed embedded systems (legacy systems)* to form new *system-of-systems (SoS)* that promise more efficient economic processes and improved services. Examples of SoSs are *smart power distribution, car-to-car communication, or air-traffic control*. The different subsystems of an SoS are developed according to their *unique architectural style*, are operated by *different* organization and serve their *own* purposes. The integration of the subsystems into an SoS can be static or dynamic. The *emergent properties* that come into existence by the integration of the subsystems can be *predicted* or are, at first, *unforeseen*. There a number of unique challenges in the design of system of systems such as, for example: the alignment of the diverse architectural styles, the control of the emergent properties, information security, and the provision of dependable service in the face of the continuous evolution of the subsystems. This talk will elaborate on the characteristics of SoS and will focus on the mentioned research challenges that must be tackled in order to provide dependable SoS services.

Murphy Was an Optimist

Kevin R. Driscoll

Honeywell International, Inc.

Abstract. Embedded, safety-critical systems often have requirements for incredibly small probabilities of failure, e.g. 10^{-9} for a one hour exposure. One often hears designers of safety-critical systems say: "We have to tolerate *all* credible faults".

However, the word "credible" in this assertion contrasts starkly with the word "incredibly" in the sentence before. In fact, there are faults and failures that most designers think can't happen which actually can and do happen with probabilities far greater than the requirements allow. The well known Murphy's Law states that: "If anything can go wrong, it will go wrong." When requirements limit failure probabilities to one-in-a-million or less, this should be rewritten as: "If anything can't go wrong, it will go wrong anyway."

There are a couple of factors that lead to designers erroneously thinking that certain faults and failures are impossible; when in fact, not only are they possible, but some are actually highly probable.

One factor is that the requirements are outside any designer's experience, even when that experience includes that of colleagues. Using the literature seems like an obvious way of expanding one's (virtual) experience. However, there are two problems with this. The first problem is that people who actually design safety-critical systems are rarely given enough time to keep current with the literature. The second problem is that the literature on actual occurrences of rare failure modes is almost nonexistent. Reasons for this include: people and organizations don't want to admit they had a failure; designers feel that rare failure occurrence aren't worth reporting; and, if designers aren't given enough time to read literature, they certainly aren't given enough time to write it. Take away: Designers should fight their management for time to keep current with the literature and designers should use every report of a rare failure as an opportunity to imagine other similar modes of failure.

The other factor that leads to designers erroneously thinking that certain faults and failures are impossible stems from abstraction. The complexity of modern safety critical systems requires some form of abstraction. However, when designers limit their thinking to one level of abstraction, certain faults and failures can seem impossible, but would clearly be seen as probable if one were to examine layers below that level of abstraction. For example, a designer thinking about electrical components would not include in their FMEA the possibility that one component (e.g. a diode) could transmogrify into another component (e.g. a capacitor). But, at a lower level of abstraction, it can be seen that a crack through a diode die can create a capacitor. And, a crack is one of the most highly probable failure modes at the physical material level of obstruction.

Examples of rare but actually occurring failures will be given. These will include a number of Byzantine faults, component transmogrification, fault mode transformation (e.g. stuck at faults that aren't so stuck), the dangers of self-inflicted shrapnel, component creation via emergent properties, "evaporating" software, and exhaustively tested software that still failed.

Process Control Security: Go Dutch! (United, Shared, Lean and Mean)

Eric Luijff

TNO Defence, Security and Safety
The Hague, The Netherlands

Abstract. International studies have shown that information security for process control systems, including SCADA, is weak. As many critical infrastructure (CI) services depend on process control systems, any vulnerability in the protection of process control systems in CI may result in serious consequences for the safety of our citizens and the security of our society, economy and ecology.

Various critical sectors in The Netherlands like drinking water, energy, multinationals have identified process control security as an important theme to jointly address in the Dutch National Infrastructure against Cybercrime (NICC). A set of activities were started, such as sector-wide benchmarks, awareness raising, development of good practices, sharing of incident information, developing an acquisition standard, and red-blue team training. Mid of 2010, the Dutch Process Control Security Roadmap project took off which comprises a coordinated set of actions to raise the security barriers in the domain where information technology touches the physical world. Rather than re-inventing wheels, the Dutch approach is lean and mean trying to improve and integrate existing efforts and advancements using a united effort by for instance chief information officers, process control users, manufacturers, system integrators, EDP-auditors, education, and R&D. The results are shared with all the participants in order to reach an improved and high level of protection at the short, medium and the long time. Results are shared as well with other nations, international information exchanges and vendors aiming international acceptance and a next, shared improvement cycle.

The keynote session will highlight the approaches and show some of the results.

Author Index

- Abele, Andreas 332
Akella, Ravi 263
Al-Ars, Zaid 431
Alexandersson, Ruben 466
Althammer, Egbert 183
Ambrósio, Ana Maria 112
Aryus, Hans-Jörg 97
- Basir, Nurlida 68
Bishop, Peter 237, 375
Bobbio, Andrea 41
Böhm, Petr 15
Brinkkemper, Sjaak 251
- Castorini, Elisa 291
Cha, Sungdeok 223
Chen, DeJiu 332
Chozos, Nick 375
Csertán, György 431
Cyra, Lukasz 237
- Denney, Ewen 68
Dittel, Torsten 97
Driscoll, Kevin R. 481
- Eckel, Andreas 431
Eriksson, Henrik 210
- Faza, Ayman 277
Feng, Lei 332
Ferdinand, Christian 196
Ferri, Felipe 361
Fetzer, Christof 169
Fidi, Christian 431
Fioriti, Vincenzo 291
Fischer, Bernd 68
Fohler, Gerhard 431
- Gomes, Adriano 361
Gómez, Carolina 28
Granzer, Wolfgang 303
Gruber, Thomas 15, 183
- Haanappel, Suzanne 251
Habli, Ibrahim 82
- Harju, Hannu 55
Hatebur, Denis 317
Heckmann, Reinhold 196
Heisel, Maritta 317
Herbst, Christian 452
Hutchesson, Stuart 389
- Iliasov, Alexei 126
Ilic, Dubravka 126
- Jee, Eunkyoung 223
Jesus, Joabe 361
Johansson, Mika 55
Johansson, Rolf 332
Johnson, Christopher W. 142, 402
- Kaiser, Bernhard 452
Karlsson, Johan 466
Kästner, Daniel 196
Kastner, Wolfgang 303
Kopetz, Hermann 480
Khelil, Abdelmajid 431
Kim, Suin 223
Klaas, Vanessa 452
- Lahtinen, Jussi 55
Laibinis, Linas 126
Lascych, Peter 452
Latvala, Timo 126
Lee, Insup 223
Liggesmeyer, Peter 28
Lönn, Henrik 332
Luijff, Eric 483
Lüth, Christoph 347
- Madeira, Henrique 112
Marchei, Elena 291
Mattes, Tina 155
McDermid, John 389
McMillin, Bruce M. 263, 277
Milbredt, Paul 431
Mota, Alexandre 361
- Nevalainen, Risto 55
Novak, Thomas 418

- Obermaisser, Roman 431
 Öhman, Peter 466

 Palin, Robert 82
 Peleska, Jan 1
 Pontes, Rodrigo P. 112

 Ranta, Jukka 55
 Raue, Stefan 402
 Romanovsky, Alexander 126
 Ruzzante, Silvia 291

 Sandberg, Anders 332
 Scheerer, Christoph 431
 Schiffel, Ute 169
 Schiller, Frank 155
 Schmitt, André 169
 Schneelee, Stefan 431
 Schoitsch, Erwin 183
 Schulz, Oliver 1
 Schulz, Stefan 452
 Sedigh, Sahra 277
 Stoegerer, Christoph 418

 Suri, Neeraj 431
 Stüßkraut, Martin 169
 Sutor, Ariane 28
 Svenningsson, Rickard 210

 Täubig, Holger 347
 Tavakoli-Kolagari, Ramin 332
 Terruggia, Roberta 41
 Torchiaro, Sandra 332
 Törngren, Martin 210, 332
 Turlas, Kostas 375
 Troubitsyna, Elena 126

 Väisänen, Pauli 126
 Varpaaniemi, Kimmo 126
 Vêras, Paulo C. 112
 Vermeulen, Bart 431
 Vieira, Marco 112
 Villani, Emilia 112
 Vinter, Jonny 210

 Walter, Dennis 347
 Watanabe, Edson 361