

Barbara Gallina
Amund Skavhaug
Friedemann Bitsch (Eds.)

LNCS 11093

Computer Safety, Reliability, and Security

37th International Conference, SAFECOMP 2018
Västerås, Sweden, September 19–21, 2018
Proceedings



 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, Lancaster, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Zurich, Switzerland

John C. Mitchell

Stanford University, Stanford, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

C. Pandu Rangan

Indian Institute of Technology Madras, Chennai, India

Bernhard Steffen

TU Dortmund University, Dortmund, Germany

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max Planck Institute for Informatics, Saarbrücken, Germany

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


Barbara Gallina · Amund Skavhaug
Friedemann Bitsch (Eds.)

Computer Safety, Reliability, and Security

37th International Conference, SAFECOMP 2018
Västerås, Sweden, September 19–21, 2018
Proceedings

Editors

Barbara Gallina 
Mälardalen University
Västerås
Sweden

Friedemann Bitsch 
Thales Deutschland GmbH
Ditzingen
Germany

Amund Skavhaug
Norwegian University of Science
and Technology
Trondheim
Norway

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-99129-0 ISBN 978-3-319-99130-6 (eBook)
<https://doi.org/10.1007/978-3-319-99130-6>

Library of Congress Control Number: 2018950937

LNCS Sublibrary: SL2 – Programming and Software Engineering

© Springer Nature Switzerland AG 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

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

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This volume contains the proceedings of the 37th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2018) held during September 19–21, 2018, in Västerås, Sweden. Since 1979, when the conference was established by the European Workshop on Industrial Computer Systems, Technical Committee 7 on Reliability, Safety and Security (EWICS TC7), it has contributed to the state of the art through knowledge dissemination and discussions of important aspects of computer systems of our everyday life. With the proliferation of embedded systems, the omnipresence of the Internet of Things, and the commodity of advanced real-time control systems, our dependence on safe and correct behavior is increasing. Currently, we are witnessing the beginning of the area of truly autonomous systems, perhaps with driverless cars as the most well-known example to the non-specialist, where the safety and correctness of their computer systems are already being discussed in the mainstream media. In this context, it is clear that the relevance of the SAFECOMP conference series is increasing.

The international Program Committee (PC), consisting of 56 members from 15 countries, received 63 papers from 24 nations. Of these, 20 papers were selected to be presented at the conference resulting in an acceptance rate of 31.7%. The review process was thorough with at least three reviewers, which ensured independency, and 20 of these reviewers met in person in Munich, Germany in April 2018 for the final discussion and selection. Our warm thanks go to reviewers who offered their time and competence in the PC work. We are grateful for the support we received from the PC member Mario Trapp, Fraunhofer ESK, who generously hosted the PC meeting.

The conference featured three keynotes: “Software Engineering for Safety in Molecular Programmed Systems” by Robyn Lutz, Professor of Computer Science at Iowa State University; “Reviews?! We Do That! Cross-Domain Reuse of Engineering Knowledge and Evidence” by Uma Ferrell, Software and Airborne Electronic Hardware Designated Engineering Representative for the US Federal Aviation Administration; “Experiences from the Industry, Design and Application of a Control System Platform for Safety of Machinery” by Richard Hendeberg, Specialist in Functional Safety at Epiroc Rock Drills AB.

As in the previous years, the conference was organized as a single-track conference, allowing intensive networking during breaks and social events, and participation in all presentations and discussions. The conference also included a fast abstracts session, giving the opportunity for new ideas and work in progress to bloom in a fertile soil. The Fast Abstracts proceedings are published in the HAL repository.

Finally, the conference also included a panel session, focusing on stimulating an interactive discussion with the audience around the main theme of SAFECOMP 2018, i.e., “Cross- and Intra-Domain Reuse of Engineering and Certification Artefacts: Challenges and Opportunities.”

As has been the tradition for many years, the day before the main track of the conference was dedicated to five regular workshops: DECSoS, ASSURE, SASSUR, STRIVE, WAISE. Papers from these workshops are published in a separate LNCS volume (11094).

We would like to express our gratitude to the many people who helped with the preparations and running of the conference, especially Friedemann Bitsch as publication chair, Erwin Schoitsch as workshop chair, Jérémie Guiochet as fast abstracts chair, Alexander Romanovsky as publicity chair, and not to be forgotten the local organization and support staff, Irfan Sljivo, Lena Jonsson, Martina Pettersson, Elena Rivani, Linda Claesson, and Gunnar Widforss.

For its support, we wish to thank Mälardalen University, represented by the School of Innovation, Design, and Engineering and, more specifically, by the research group Certifiable Evidences and Justification Engineering. We also wish to thank all other supporting institutions.

Without the support from the EWICS TC7 headed by Francesca Saglietti, this event could not have happened. We wish the EWICS TC7 organization continued success, and we are looking forward to being part of this in the future.

Finally, the most important people to whom we want to express our gratitude are the authors and participants. Your dedication, effort, and knowledge are the foundation of the scientific progress. We hope you had fruitful discussions, gained new insights, and had a memorable time in Västerås.

September 2018

Barbara Gallina
Amund Skavhaug

Organization

EWICS TC7 Chair

Francesca Saglietti University of Erlangen-Nuremberg, Germany

General Chair

Barbara Gallina Mälardalen University, Sweden

Program Co-chairs

Barbara Gallina Mälardalen University, Sweden
Amund Skavhaug The Norwegian University of Science and Technology,
Norway

Workshop Chair

Erwin Schoitsch AIT Austrian Institute of Technology, Austria

Publication Chair

Friedemann Bitsch Thales Deutschland GmbH, Germany

Organizing Committee

Irfan Sljivo Mälardalen University, Sweden
Lena Jonsson Mälardalen University, Sweden
Martina Pettersson Mälardalen University, Sweden
Elena Rivani Mälardalen University, Sweden
Linda Claesson Mälardalen University, Sweden
Gunnar Widforss Mälardalen University, Sweden

Publicity Chair

Alexander Romanovsky Newcastle University, UK

Fast Abstracts Chair

Jérémie Guiochet LAAS-CNRS, University of Toulouse, France

Program Committee

Uwe Becker	Draeger Medical GmbH, Germany
Peter G. Bishop	Adelard, UK
Friedemann Bitsch	Thales Deutschland GmbH, Germany
Robin Bloomfield	City University London, UK
Sandro Bologna	Associazione Italiana Esperti Infrastrutture Critiche, Italy
Andrea Bondavalli	University of Florence, Italy
Jens Braband	Siemens AG, Germany
Anna Carlsson	OHB Sweden, Sweden
António Casimiro	University of Lisbon, Portugal
Peter Daniel	EWICS TC7, UK
Ewen Denney	SGT/NASA Ames Research Center, USA
Felicita Di Giandomenico	ISTI-CNR, Italy
Wolfgang Ehrenberger	Hochschule Fulda, Germany
Massimo Felici	Deloitte Consulting & Advisory, Belgium
Uma Ferrell	MITRE Corporation, USA
Francesco Flammini	Linnaeus University, Sweden
Barbara Gallina	Mälardalen University, Sweden
Ilir Gashi	CSR, City University London, UK
Janusz Górski	Gdańsk University of Technology, Poland
Jérémie Guiochet	LAAS-CNRS, France
Maritta Heisel	University of Duisburg-Essen, Germany
Chris Johnson	University of Glasgow, UK
Bernhard Kaiser	Assystem Germany GmbH, Germany
Karama Kanoun	LAAS-CNRS, France
Johan Karlsson	Chalmers University of Technology, Sweden
Phil Koopman	Carnegie Mellon University, USA
Floor Koornneef	Delft University of Technology, The Netherlands
Timo Latvala	Space Systems Finland Ltd., Finland
Bev Littlewood	City University London, UK
Silvia Mazzini	Intecs, Italy
John McDermid	University of York, UK
Frank Ortmeier	Otto-von-Guericke Universität Magdeburg, Germany
Michael Paulitsch	Intel, Austria
Holger Pfeifer	Technical University of Munich, Germany
Thomas Pfeiffenberger	Salzburg Research Forschungsgesellschaft m.b.H., Austria
Peter Popov	City University London, UK
Laurent Rioux	Thales R&T, France
Alexander Romanovsky	Newcastle University, UK
John Rushby	SRI International, USA
Francesca Saglietti	University of Erlangen-Nuremberg, Germany
Christoph Schmitz	Zühlke Engineering AG, Switzerland
Erwin Schoitsch	AIT Austrian Institute of Technology, Austria

Christel Seguin	Office National d'Etudes et Recherches Aérospatiales, France
Amund Skavhaug	The Norwegian University of Science and Technology, Norway
Mark-Alexander Sujan	University of Warwick, UK
Kenji Taguchi	CAV Technologies Co., Ltd., Japan
Stefano Tonetta	Fondazione Bruno Kessler, Italy
Mario Trapp	Fraunhofer Institute for Experimental Software Engineering, Germany
Elena Troubitsyna	Åbo Akademi University, Finland
Fredrik Törner	Volvo Car Corporation, Sweden
Martin Törngren	KTH Royal Institute of Technology, Sweden
Pieter van Gelder	Delft University of Technology, The Netherlands
Marcel Verhoef	European Space Agency, The Netherlands
Jonny Vinter	RISE Research Institutes of Sweden
Helene Waeselyncx	LAAS-CNRS, France

Additional Reviewers

Matthieu Amy	LAAS-CNRS, France
Milan Battelino	OHB Sweden, Sweden
Victor Bos	Space Systems Finland Ltd., Finland
Bill Drozd	Carnegie Mellon University, USA
Sam George	Adelard, UK
Didem Gürdür	KTH Royal Institute of Technology, Sweden
Denis Hatebur	University of Duisburg-Essen, Germany
Dubravka Ilic	Space Systems Finland Ltd., Finland
Lola Masson	LAAS-CNRS, France
Viorel Preoteasa	Space Systems Finland Ltd., Finland
Irum Rauf	Åbo Akademi University, Finland
Behrooz Sangchoolie	RISE Research Institutes of Sweden
Paulius Stankaitis	Newcastle University, UK
Kimmo Varpaaniemi	Space Systems Finland Ltd., Finland
Inna Vistbakka	Åbo Akademi University, Finland
Andrzej Wardziński	Gdańsk University of Technology, Poland
Xinhai Zhang	KTH Royal Institute of Technology, Sweden

Supporting Institutions

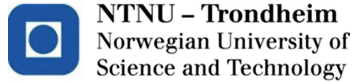
European Workshop on Industrial
Computer Systems Reliability,
Safety and Security



Mälardalen University, Sweden



Norwegian University of Science
and Technology



Austrian Institute of Technology



Thales Deutschland GmbH



Lecture Notes
in Computer Science (LNCS),
Springer Science + Business Media



Austrian Computer Society



ARTEMIS Industry Association



European Network of Clubs
for Reliability and Safety
of Software-Intensive Systems



German Computer Society



Electronic Components and
Systems for European
Leadership - Austria



Verband österreichischer
Software Industrie



XII Organization

European Research
Consortium for Informatics
and Mathematics



IEEE SMC Technical
Committee on Homeland
Security (TCHS)



Invited Talks

Software Engineering for Safety in Programmed Molecular Systems

Robyn R. Lutz

Iowa State University, Ames, IA 50011, USA
rlutz@iastate.edu

Abstract. Molecular programming uses the computational power of DNA and other biomolecules to create nanoscale systems. Many of these envisioned nano-systems are safety-critical, such as diagnostic biosensors that detect contaminants, drug capsules that dispense medicine when they encounter diseased cells, and configurable nano-robots. Challenges to the safety engineering of the nano-systems include their probabilistic behavior, their very small size, the very large number of them that execute at once, and the dynamic environment in which they operate. Designs need to assure safe outcomes from highly fault-prone devices, hampered by the difficulty of defining the limits of their safe operation.

I organize the talk around our interdisciplinary team's development of an essential safety building block for programmed molecular systems – an embeddable, reusable, molecular Runtime Fault Detector. I describe how we harnessed goal-oriented requirements and risk analyses, reaction network modeling, and probabilistic model checking to specify, analyze, and verify the safety requirements and design for this new nano-system. Finally, I suggest that a similar approach also may be helpful in the safety engineering of non-molecular systems composed of highly distributed, autonomous, fault-prone components operating in dynamic environments.

Keywords: Software safety · Molecular programming · Software engineering
Chemical reaction networks

Acknowledgments. This research was supported in part by National Science Foundation Grant 1545028.

Reviews?! We do that! Cross-Domain Reuse of Engineering Knowledge and Evidence

Uma Ferrell

MITRE Corporation, 7515 Colshire Drive, McLean VA 22102, USA

Abstract. Both industry and certification authorities have reason to be excited about the benefits and opportunities of reusing and building products for more than one domain such as aviation and automobiles. Cross-domain reuse in an increasingly complex world can inject novel technologies to conventional domains to increase safety. Such opportunities come with social and ethical responsibilities for the safe use of a product in the target environment, not just whether the product and evidence are acceptable to certification authorities. The evidence may be wrongly presented based only on the equivalency in the use of expected language in pertinent standards. The evidence should be based on the actual accomplishments met and whether those accomplishments are applicable towards design assurance and safety in the target domain and environment.

Cross-domain reuse has many considerations. This talk is focused only on safety and security. Obviously, consideration of reuse must include functionality, use of standards in that domain, and certification concerns. All these considerations have undercurrents of safety as well as security. Let us focus further on three topics:

- **Derivation of risk:** Derivation of risk depends on the target domain and the human/system use of the product. Also, the acceptable level of risk tolerance is inherently different in different domains. Aviation is one of the few domains where safety risk tolerance is codified. As stewards of safety in this society, we need to be aware of the real idea behind certification, and promulgate a safety culture to take responsibility for safe cross-domain use of the product throughout the product life.
- **Appropriate use of evidence:** While acceptability for certification is important, the knowledge and evidence for why a product is acceptable is even more important. Evidence may have been produced in a previous domain that appears to be usable in a target domain. Only the basis for that evidence may have a different interpretation and implication in the target domain because the terminology for even simple terms such as “reviews” may not have the same meaning in different domains. Further, the same functionality may be used in diverse ways in the two domains.
- **Importance of systems engineering:** There are certainly considerations that may be codified and delegated to checklists. But blind use of checklists makes a poor substitute for domain knowledge and engineering. Cross-domain use does not just mean that one could deploy a product. Continued safe use of the product in the target domain has specific implications for maintenance of the product as well as maintenance of the system of which the product is just one component. For example, an electro-mechanical system may need adjustments

to maintenance cycles depending on the characteristics of the component commanding the mechanical actions. In general, we must make sure that component engineering is within the context of system safety and security.

Opportunities of cross-domain reuse indeed come with responsibilities to understand, analyze, and engineer the product. Appropriate reuse considered in the system context can be a powerful tool to introduce newer technologies to solve complex problems.

Experiences from the Industry, Design and Application of a Control System Platform for Safety of Machinery

Richard Hendeberg

Epiroc Rock Drills AB, Örebro, Sweden
richard.hendeberg@epiroc.com

Abstract. Epiroc Rock Drills AB is a global manufacturer of mining and construction machinery. These highly automated machines operates in an incredibly harsh environment where reliability and availability is paramount. In this talk, the focus is on Epiroc's control systems platform and work with safety of machinery. How a modular design, componentization of software and standardization on hardware modules has led to an efficient reuse of engineering efforts and an automation platform, which is used throughout Epiroc's entire range of machinery. In this talk, it is also given an overview of Epiroc's journey with safety of control systems, leading up to the integration of safety functions into the existing control system platform. The challenges of designing safety functions for a harsh environment and why availability of the machine might be as important for the safety of the operator as the reliability of the safety function.

Keywords: Mining machinery · Construction machinery · Safety of machinery
Hardware component reuse

Contents

Automotive Safety Standards and Cross-Domain Reuse Potential

Practical Experience Report: Automotive Safety Practices vs. Accepted Principles	3
<i>Philip Koopman</i>	
A Generic Method for a Bottom-Up ASIL Decomposition	12
<i>Alessandro Frigerio, Bart Vermeulen, and Kees Goossens</i>	
Assurance Benefits of ISO 26262 Compliant Microcontrollers for Safety-Critical Avionics	27
<i>Andreas Schwierz and Håkan Forsberg</i>	

Autonomous Driving and Safety Analysis

Structuring Validation Targets of a Machine Learning Function Applied to Automated Driving	45
<i>Lydia Gauerhof, Peter Munk, and Simon Burton</i>	
Multi-aspect Safety Engineering for Highly Automated Driving: Looking Beyond Functional Safety and Established Standards and Methodologies	59
<i>Patrik Feth, Rasmus Adler, Takeshi Fukuda, Tasuku Ishigooka, Satoshi Otsuka, Daniel Schneider, Denis Uecker, and Kentaro Yoshimura</i>	
A Model-Based Safety Analysis of Dependencies Across Abstraction Layers	73
<i>Christoph Dropmann, Eike Thaden, Mario Trapp, Denis Uecker, Rakshith Amarnath, Leandro Avila da Silva, Peter Munk, Markus Schweizer, Matthias Jung, and Rasmus Adler</i>	

Verification

Formal Verification of Signalling Programs with SafeCap	91
<i>Alexei Iliasov, Dominic Taylor, Linas Laibinis, and Alexander Romanovsky</i>	
Deriving and Formalising Safety and Security Requirements for Control Systems.	107
<i>Elena Troubitsyna and Inna Vistbakka</i>	

Optimal Test Suite Generation for Modified Condition Decision Coverage Using SAT Solving 123
Takashi Kitamura, Quentin Maissonneuve, Eun-Hye Choi, Cyrille Artho, and Angelo Gargantini

Efficient Splitting of Test and Simulation Cases for the Verification of Highly Automated Driving Functions 139
Eckard Böde, Matthias Büker, Ulrich Eberle, Martin Fränzle, Sebastian Gerwinn, and Birte Kramer

Multi-Concern Assurance

Roadblocks on the Highway to Secure Cars: An Exploratory Survey on the Current Safety and Security Practice of the Automotive Industry 157
Michael Huber, Michael Brunner, Clemens Sauerwein, Carmen Carlan, and Ruth Breu

Safe and Secure Automotive Over-the-Air Updates 172
Thomas Chowdhury, Eric Lesiuta, Kerianne Rikley, Chung-Wei Lin, Eunsuk Kang, BaekGyu Kim, Shinichi Shiraishi, Mark Lawford, and Alan Wasssyng

Dependability Analysis of the AFDX Frame Management Design 188
Venesa Watson and Mahlet Bejiga

Fault Tolerance

Efficient On-Line Error Detection and Mitigation for Deep Neural Network Accelerators 205
Christoph Schorn, Andre Guntoro, and Gerd Ascheid

Random Additive Control Flow Error Detection 220
Jens Vankeirsbilck, Niels Penneman, Hans Hallez, and Jeroen Boydens

Fault-Tolerant Clock Synchronization with Only Two Redundant Paths 235
Zoha Moztarzadeh

MORE: MOdel-based REDundancy for Simulink. 250
Kai Ding, Andrey Morozov, and Klaus Janschek

Safety and Security Risk

Diversity in Open Source Intrusion Detection Systems. 267
Hafizul Asad and Ilir Gashi

Inter-device Sensor-Fusion for Action Authorization on Industrial Mobile Robots	282
<i>Sarah Haas, Andrea Höller, Thomas Ulz, and Christian Steger</i>	
Towards a Common Ontology of Safety Risk Concepts for Railway Vehicles and Signaling	297
<i>Bernhard Hulin, Hermann Kaindl, Roland Beckert, Thomas Rathfux, and Roman Popp</i>	
Author Index	311

Automotive Safety Standards and Cross-Domain Reuse Potential



Practical Experience Report: Automotive Safety Practices vs. Accepted Principles

Philip Koopman^(✉)

Carnegie Mellon University, Pittsburgh, PA 15217, USA
koopman@cmu.edu

Abstract. This paper documents the state of automotive computer-based system safety practices based on experiences with unintended acceleration litigation spanning multiple vehicle makers. There is a wide gulf between some observed automotive practices and established principles for safety critical system engineering. While some companies strive to do better, at least some car makers in the 2002–2010 era took a test-centric approach to safety that discounted non-reproducible and “unrealistic” faults, instead blaming driver error for mishaps. Regulators still follow policies from the pre-software safety assurance era. Eight general areas of contrast between accepted safety principles and observed automotive safety practices are identified. While the advent of ISO 26262 promises some progress, deployment of highly autonomous vehicles in a non-regulatory environment threatens to undermine safety engineering rigor.

Keywords: Software safety · Automotive · Unintended acceleration

1 Introduction

Innocent people have died, been severely injured, or gone to jail because of defects or potential defects in computer-based automotive systems. With the deployment of self-driving cars, it is more important than ever to understand the gaps between theory and practice in automotive computer-based system safety.

This paper is based on the author’s personal experiences with unintended acceleration (UA) litigation against car makers (Original Equipment Manufacturers, or OEMs) for 2000–2010 model year vehicles, and additional experiences with multiple recent military and commercial self-driving car (Autonomous Vehicle, or AV) safety assurance projects. These experiences include access to extensive sets of engineering documents, analysis of Electronic Throttle Control (ETC) source code, and vehicle testing to confirm identified safety vulnerabilities. These experiences have revealed common threads that encompass technical, business, regulatory, and litigation aspects of safety. While regulatory environments vary in other countries, the significant role that the United States (US) car industry and US legal system play in the automotive domain ensure that these factors will influence many cars produced worldwide.

Unlike other domains, conformance to international computer-based system safety standards is voluntary for US-sold vehicle OEMs and suppliers. Moreover, some OEMs have not followed industry-specific guidelines such as the MISRA Software Guidelines [1], including vehicles that are the subject of two class action lawsuits. [2] at

30:21–25 and [3] at 78:15–79:15. (Note that [2] is a transcript from a death and injury case involving a vehicle of a type included in the corresponding class action lawsuit).

The US permits OEMs to deploy vehicles that are self-certified to meet provisions of the US Federal Motor Vehicle Safety Standards (FMVSS). FMVSS regulations take the form of a test procedure approach originally intended to ensure that the normal safety-relevant functionality of pre-computer vehicles, such as braking capability, was adequate. While some simplistic failure modes such as detecting the complete loss of a functional subsystem are included, the test procedures are not intended to achieve any defined amount of software testing coverage, are not designed to detect non-deterministic faults, and do not demonstrate fault recovery from non-trivial computational faults. While vehicles commonly use some basic fault tolerance patterns such as redundant CPUs for life-critical functions, it can be the case that redundancy and other fault tolerant computing techniques not used in accordance with accepted practices, such as dual-CPU designs with a single point of failure [4].

At least one death has been officially declared to be due to automotive computer system malfunction [4], and there have been approximately 500 settlements for death and injury alleged to also be due to defective vehicle designs by the same OEM [5]. Another class action against a second OEM alleges similar issues [3]. Additionally, there are instances in which individuals have faced civil or criminal penalties for mishaps they claim were caused by vehicle malfunctions (e.g., [6]). Now that computers have life critical control authority, they must be considered as a credible potential cause of severe mishaps.

Electrified vehicles present additional risks because regenerative braking tends to disable the direct hydraulic connection between the brake pedal and friction-based brake pads [7]. (If this weren't the case, energy could be lost due to friction instead of being used to recharge the battery.) Some drivers have reported loss of brake effectiveness with these vehicles (e.g., [8]) which could potentially be caused by a software defect. Some litigation has involved reported symptoms consistent with such a defect. Increasing levels of autonomy raise the stakes further.

Table 1. Contrasting areas of safety principles and observed automotive practices.

Accepted safety principle	Observed automotive safety practice
Evidence required to show <u>safety</u>	Evidence required to show <u>defect</u>
Safety argument	System-level functional test
Arbitrary failures	“Realistic” failures
Random failures expected	Non-reproducible failures are discounted
Blaming humans is a last resort	Driver error presumed
Engineering rigor and integrity level	All unsafe defects identified and fixed
Independent assessment	Self-certification
ALARP, etc.	Cost effective regulation

Table 1 identifies areas in which some observed OEM practices do not necessarily correspond with accepted safety principles. The scope of this table deals with vehicles produced with ETC in the 2002–2010 era from some Asia, US and European OEMs

selling into the US market. It should be emphasized that some OEMs claim to follow accepted safety practices. And to be clear, the listed OEM practices should not be considered industry-accepted practices for making safe vehicles, but rather should be seen as areas in which some OEMs' observed practices fell short of meeting accepted safety practices. Based on personal experience in a variety of venues, it is clear that portions of the OEM and supplier ecosystem were still stuck in the pre-software safety engineering era at least up until the creation of ISO 26262 [9], and that adoption of that new standard is taking time.

2 Safety Principles vs. Automotive Safety Practices

2.1 Safety Arguments Aren't Specifically Required by Regulators

A general safety principle is that a system is not presumed to be safe until a mishap occurs, but rather must be demonstrated to be safe before deployment. Approaches to demonstrating safety are typically based on some sort of safety argument. That argument might be explicit (e.g., a GSN argumentation structure [10]), implicit in the form of having followed a suitable set of safety practices (e.g., [1]), or some mixture of the two. Common codified safety practices include the generic notions of a Safety Integrity Level (SIL), Design Assurance Level (DAL), or other risk-based approach to identifying and requiring a defined level of engineering rigor.

The US legal system, on the other hand, tends to emphasize the identification of defects. OEMs can attempt to defend themselves simply by asserting that their vehicle is safe because no bugs have been identified that lead to UA [11] at 47:3–10. Injured parties and their experts typically must search for relevant bugs or other design defects such as single points of failure to support a vehicle defect argument.

US regulations do not require vehicles to have a safety argument beyond FMVSS compliance, although using one is not precluded. However, lack of following accepted engineering practices can be a contributing factor to legal outcomes, especially when considering negligence. Additionally, a pattern of mishaps can lead to a mandatory vehicle recall in some cases.

Some European vehicles in the 2000s adopted the E-Gas approach for electronic throttle control ([12] is a newer, publicly available description). In general, the approach involves a primary functional unit that performs control, and monitoring/checking units that disable engine power if a fault is detected. The suitability of this approach for life-critical applications depends upon adequate isolation between doer/checker levels and appropriate fault coverage. In some cases, independent UA mitigation is required, such as a vacuum pump to boost braking force independent of throttle position. The specification also describes required fault handling functionality.

2.2 Argumentation vs. Testing

While general safety principles require some sort of argument based in part on engineering analysis and rigor, the US regulatory system and much common practice is heavily based on vehicle-level testing. It is common for OEMs to practice

non-software-specific techniques for fault analysis such as DFMEAs [13]. However, use of more advanced computer-based system safety techniques is uneven.

As previously discussed, the centerpiece of US automotive safety regulation is the suite of Federal Motor Vehicle Safety Standards (FMVSS). While some testing contemplates simplistic component fault models, FMVSS criteria generally do not involve design processes, code quality, or other accepted computer-based system safety considerations. For example, FMVSS 138 [14] fault injection covers a silent malfunction due to loss of component power in a tire pressure monitoring system. Similarly, US National Highway Traffic Safety Administration (NHTSA) investigations involve vehicle level testing and discussions with the OEM, but emphasize driver error as a cause of UA. For example, [15] blames the driver rather than the ETC for data samples showing a doubling of engine RPM and vehicle speed with unchanged accelerator pedal input.

2.3 Arbitrary vs. “Realistic” Faults and Failures

For safety critical systems, even a single bit flip or other small fault has the potential to cause a catastrophic mishap if not sufficiently mitigated. Well defined and expansive fault models such as transient faults and single event upsets are well known in the areas of safety and fault tolerant computing research. Arbitrary failures of computer-based system components must be considered when designing life-critical systems [16]. Moreover, there is an increasing body of confirmed reports of Byzantine (e.g., two-faced) faults occurring in real systems [17]. However, some OEMs do not embrace these accepted fault and failure models.

Automotive OEM safety analysis is often concerned with simplistic fault models such as electrical wires shorted to power supply voltages, open circuits, or computer crashes. Faults that are subjectively judged not to be “realistic” by designers are often dismissed. However, research has documented subtle real world faults and failures that defy designer intuition about fault realism [18].

Any redundancy often relies upon self-diagnosis and simplistic fault detection mechanisms such as watchdog timers, heartbeats, and input port sanity checks [4]. Such simplistic redundancy management approaches offer only partial fault coverage, and permit dangerous fail-active behaviors [19].

2.4 Failure Reproducibility

Transient faults and resulting failures are generally not reproducible upon demand in ordinary system operation, because the underlying causes can be comparatively infrequent, randomly occurring events. Fault injection experiments reveal vulnerabilities, but are routinely criticized in litigation for involving minor instrumentation modifications to vehicle software such as inclusion of a subroutine to flip memory bits upon command. Such modifications are then claimed to render fault injection results invalid due to involving a variation from the exact software image that would be in a production vehicle, or otherwise not being “realistic” [11] at 84:14–24.

Diagnostic gaps and undiagnosed failures are common. In some – but not all – cases, Trouble Not Identified (TNI) incidents can eventually be traced to systematic

causes with sufficient detective work [20]. Despite less than complete diagnostic coverage, and substantial TNI rates, ETC malfunction is often inappropriately ruled out by OEMs or investigators when no Diagnostic Trouble Code (DTC) has been recorded. This is especially true when problems cannot be reproduced with the subject vehicle – even when a report is made by a source that many would consider credible, such as a dealership employee or police officer [3] at 86:10–87:24.

Automotive safety struggles with non-reproducible faults. NHTSA tends to close investigations of non-reproducible faults rather than investigating potential software defects as root causes of mishaps. Similarly, OEMs can emphasize reproducible faults and undeniable trends of field data, rather than perceived “one-off” events, in part to avoid putting “the company out of business” [21].

2.5 The Driver Error Narrative

It is well known that humans are imperfect. It follows that the heart and soul of a typical UA legal defense is a claim of driver error, typically in the form of pressing the accelerator pedal instead of the brake pedal. Many publications, including those from NHTSA, repeat the refrain of driver error causing UA events [22]. However, these reports fail to consider computer system defects. Rather, reports conclude that in the absence of mechanical defects or concrete physical evidence of a vehicle malfunction the cause of a mishap must be driver error. Situations that provide truly compelling evidence to rule out driver error tend to be attributed to “unknown” causes.

While OEMs and NHTSA typically cite various reports in support of the pedal misapplication narrative, what data can be found on that specific failure mode tends to tell a different story. A pre-ETC analysis of 997 “reasons/excuses” for crashes found only one instance of “hit gas pedal instead of brake” – but 29 instances of “vehicle failure” [23] pp. 293, 296. Thus, contrary to the typical human error narrative, available data provides support for a finding that vehicles malfunction more often than humans press the wrong pedal.

Revisiting the Audi 5000 investigation report reveals that even the veritable poster child of human error producing UA provides incomplete support for the pedal misapplication narrative. Audi vehicle malfunctions produced up to 0.3 g of un-commanded acceleration, having nothing to do with driver error. However, when such a UA event startled the driver, sometimes the driver would press the wrong pedal, resulting in a collision before there was time to self-correct in a tight-quarters situation [24].

Pedal misapplication issues are complicated by problems with data recording strategies, such as potentially missing driver actions due to under-sampling [15]. Moreover, data recordings can be untrustworthy to the extent they rely upon suspect data being provided by the same computer that is potentially causing the UA.

2.6 Engineering Rigor

Developing naked, undocumented code with no substantive safety process can reasonably be expected to result in defects that could cause a catastrophic loss event for life critical systems. This can create a fear that developers will be criticized for the smallest of imperfections. However, the remedy for this fear is well understood: use an

accepted safety approach. If nothing else, a successful independent assessment provides an argument in defense of allegations of negligence. However, a negative assessor report can appear to be adverse in litigation [3] at 78:15–78:21.

Some automotive designers adopted model-based design during the 2000–2010 timeframe. This type of approach can provide tool support for certified code generation and formal proofs of correctness for some aspects of system operation. However, more than this is required for safety, and use of this type of tooling does not by itself ensure good design quality. The two class action cases discussed in this paper did not make any apparent use model based design for the code in question.

2.7 Certification and Deployment of Autonomous Vehicles

Independent assessment of safety standard conformance has been possible for many years in the automotive industry. However, current automotive regulations only require assessment against FMVSS test regimes. The future of AVs currently promises more of the same. A first draft AV policy [25] encouraged some level of accountability for safety arguments via a self-certification signature sheet. However, a later version takes a “non-regulatory” approach to safety, making even self-certification entirely optional for AVs [26]. Current US federal regulatory efforts emphasize modifications or waivers of FMVSS test regimes to accommodate AVs.

Of significant concern in AV deployment is the usual argument for doing so: human drivers make avoidable mistakes; computers won’t make those mistakes; therefore computers will be safer drivers than humans. There is insufficient field data and no robust technical public safety argument upon which to base an assertion that AVs have even achieved safety parity with an “average” human driver (whatever that might actually mean, noting that impaired drivers are part of the human driver population). Perhaps AVs will simply make *different* mistakes. Ensuring AV safety is complicated by the use of novel technologies such as machine learning [27].

Two vendors have commendably published safety brochures [28, 29]. No vendors currently claim rigorous, independently assessed safety arguments.

3 Regulatory and Litigation Considerations

3.1 Cost Effectiveness of Safety Assessment

Accepted safety practices require reducing risk to an acceptably low level, e.g., As Low as Reasonably Practicable (ALARP). However, US government agencies are required to justify that all new regulations, including safety regulations, are cost effective. The existing pedal misapplication narrative surrounding UA makes it difficult to introduce new software safety regulations to avoid software defects, because such defects have not been officially blamed for many mishaps. If there is no apparent carnage from unsafe software, it is difficult to cost-justify improving software safety. However, new laws can create stronger safety requirements without cost justification.

The litigation aspect of cost effectiveness is a bit different. Generally, the questions asked are whether accepted engineering practices were followed, and whether a

reasonable alternative design approach would have prevented a mishap from occurring. However, a defect must first be identified before those questions are asked, and generally some sort of loss or legal violation must occur before legal action can be taken.

3.2 Source Code Availability

Source code is generally unavailable for inspection unless a very large litigation effort is mounted. Government regulators do not have access to source code, nor do any outside assessors unless the OEM decides to voluntarily grant access. Even if litigation source code access is granted, it is often done under onerous conditions such as via a dedicated non-networked secure room with a metal detector wand procedure before entrance. In one case, a judge found that OEM “misrepresentations caused Plaintiffs to incur unnecessary costs” due to requiring overly burdensome source code security measures [30]. All things considered, source code analysis can easily turn into a million-dollar-plus effort including the cost of litigating to gain access, the cost of operating a secure room, and expert witness costs. This makes source code analysis impractical for most litigation, especially criminal defense, unless it can piggy-back on a class action lawsuit that has deep pockets financial backing.

The expense and difficulty of source code analysis provides a perverse incentive for poor code quality, skimpy design information, and opaque configuration management practices. The more difficult to understand the software system is, the more difficult and expensive it will be for experts to access it and identify specific defects that could have caused UA or other dangerous vehicle behaviors.

3.3 The Importance of Academic Rigor in Publication

Academics need to be aware that litigation uses peer-reviewed academic papers as evidence to support expert testimony. Even a well-intentioned paper that reaches a flawed or poorly stated conclusion can do significant damage to practical safety if a lawyer can find a way to interpret it as providing protective cover for an unsafe system. Researchers and reviewers should be mindful of ways in which a paper might be used to support an opinion that accepted safety practices are deficient unless that is truly the finding of the research data. A particularly important point is that old techniques should not be identified as defective simply because new techniques are better. Studies should disclose threats to validity so that conclusions are not applied in inappropriate situations. Finally, reviewers and editors should ensure that authors who attempt to discredit previous publications fully disclose potential conflicts of interest that might potentially result in bias, such as involvement in pending litigation adverse to the previous publication’s findings or authors [31].

4 Conclusions

Automotive-specific safety guidelines and standards have existed for more than two decades. Yet adoption is not required, and not is universal. Recent findings of industry cover-ups regarding sticky gas pedals, floor mats, ignition switches, air bags, and

emission defeat devices do not inspire confidence. One can hope that the significant costs paid by OEMs for these transgressions will motivate better behavior in the future. Litigation historical outcomes notwithstanding, it remains to be seen whether AV designers will adopt robust safety engineering practices, or will succumb to pressure and take shortcuts in the rush to market.

While it would be best if all OEMs actually adopted well understood accepted safety practices, a more pragmatic approach is to perform research that will meet the automotive industry where it is instead of where it should be. To that end, additional work on the following topics could help improve practical automotive safety (this list should not be interpreted as criticism of currently accepted safety practices):

- Studies that explicitly differentiate between driver error and computer faults
- Studies that measure how well specific safety techniques reduce mishap risk
- Fault injection techniques tailored to production vehicle deployment
- System-level testing approaches that validate safety
- Safety measurement approaches suitable for FMVSS test procedure codification
- Forensically valid automotive data recorders
- AV-specific safety validation (e.g., machine learning safety validation)
- Better understanding of the factors that support a robust safety culture.

More generally, anything that the safety community can do help educate regulators, lawmakers, and non-specialist automotive practitioners appreciate the importance of adopting safety techniques proven in other domains can also help.

Threats to Validity: Reported experiences are based on previous-generation vehicle designs due to the retrospective nature of the litigation and regulatory system. There is a significant variation in OEM attitudes and practice of safety, and certainly some OEMs try hard to adopt and even go beyond basic accepted safety practices.

Disclosure: The author is involved in ongoing litigation concerning multiple OEMs, including Toyota and Ford, and is a principle in an autonomous vehicle safety company. He is not a lawyer. No external support funded this research.

References

1. MISRA: Development Guidelines for Vehicle Based Software, November 1994
2. Bookout v. Toyota Trial Transcript, 11 October 2013. <https://goo.gl/MP8w3w>
3. Charles Johnson et al. v. Ford Motor Company, US Dist. S. WV, Huntington, 3:13-CV-06529, 1 Feb 2018 PM. (Lawyer summaries of expert testimony and evidence)
4. Koopman, P.: A case study of toyota unintended acceleration and software safety. Carnegie Mellon University, 18 September 2014. Presentation slides
5. Kennedy, J.: Toyota has reached deals in 496 cases in acceleration MDL. Law360, 15 November 2017. <https://goo.gl/T4TaLs>
6. Manganis, J.: Cop's fatal-crash trial underway; defense appears to abandon long-touted 'sudden acceleration' theory. Salem News, 17 March 2008. <https://goo.gl/jiZ9rN>
7. Toyota, 2005 Prius Repair Manual (RM1130U), pp. 05–951
8. Marosi, R., Olivarez-Giles, N.: Runaway prius driver: I was laying on the brakes but it wasn't slowing down, 10 March 2010. <https://goo.gl/aZK7BM>
9. ISO: Road vehicles-Functional Safety-Management of functional safety, ISO 26262 (2011)

10. GSN Community Standard Version 1, November 2011
11. Bookout v. Toyota Trial Transcript, 22 October 2013. <https://goo.gl/hh47vg>
12. EGAS Working Group, Standardized E-Gas Monitoring Concept for Gasoline and Diesel Engine Control Units, Version 5.5 (2013)
13. SAE: Potential Failure Mode and Effects Analysis in Design (Design FMEA), J1739_200901, 15 January 2009
14. GPO: Section 571.138, Standard No. 138; Tire pressure monitoring systems. 49 CFR Ch. V (10-1-11 Edition)
15. NHTSA: Denial of a petition for a defect investigation. Federal register vol. 80, no. 93, pp. 27835–27844, 14 May 2015
16. Lala, J., Harper, R.: Architectural principles for safety-critical real-time applications. Proc. IEEE **82**(1), 25–40 (1994)
17. Driscoll, K., Hall, B., Sivencrona, H., Zumsteg, P.: Byzantine fault tolerance, from theory to reality. In: Anderson, S., Felici, M., Littlewood, B. (eds.) SAFECOMP 2003. LNCS, vol. 2788, pp. 235–248. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39878-3_19
18. Driscoll, K.: Real system failures (2012). <https://c3.nasa.gov/dashlink/resources/624/>
19. Hammett, R.: Design by extrapolation: an evaluation of fault-tolerant avionics. In: 20th Conference on Digital Avionics Systems. IEEE (2001)
20. Thomas, D., et al.: The ‘trouble not identified’ phenomenon in automotive electronics. Microelectron. Reliab. **42**, 641–651 (2002)
21. Gladwell, M.: The engineer’s lament: two ways of thinking about automotive safety. The New Yorker, 4 May 2015
22. Lococo, K., et al.: Pedal Application Errors, DOT HS 811 597, March 2012
23. Wierwille, W., et al.: Identification and evaluation of driver errors: overview and recommendations. Federal Highway Administration; McLean, VA, FHWARD-02-003 (2002)
24. Walter, R., et al.: Study of mechanical and driver-related systems of the Audi 5000 capable of producing uncontrolled sudden acceleration incidents, DOT-TSC-NHTSA-88-4, December 1988
25. US DoT: Federal Automated Vehicles Policy: Accelerating the next revolution in roadway safety, September 2016
26. US DoT: Automated Driving Systems 2.0: a vision for safety, September 2017
27. Koopman, P., Wagner, M.: Autonomous vehicle safety: an interdisciplinary challenge. IEEE Intell. Transp. Syst. Mag. **9**, 90–96 (2017)
28. Waymo: On the Road to Fully Self-Driving (2018). <https://goo.gl/3GwP2T>
29. GM: 2018 Self-Driving Safety Report. <https://goo.gl/2d5PTM>
30. Johnson, C., et al. v. Ford Motor Company, US Dist. S. WV, Huntington, 3:13-CV-06529, order granting sanctions, 27 December 2017
31. Koopman, P.: Letter to editor. IEEE Consum. Electron. Mag. **7**(1), 6 (2018)



A Generic Method for a Bottom-Up ASIL Decomposition

Alessandro Frigerio¹(✉), Bart Vermeulen², and Kees Goossens¹

¹ Eindhoven University of Technology, Eindhoven, The Netherlands
a.frigerio@tue.nl

² NXP Semiconductors, Eindhoven, The Netherlands

Abstract. Automotive Safety Integrity Level (ASIL) decomposition is a technique presented in the ISO 26262: Road Vehicles - Functional Safety standard. Its purpose is to satisfy safety-critical requirements by decomposing them into less critical ones. This procedure requires a system-level validation, and the elements of the architecture to which the decomposed requirements are allocated must be analyzed in terms of Common-Cause Faults (CCF). In this work, we present a generic method for a bottom-up ASIL decomposition, which can be used during the development of a new product. The system architecture is described in a three-layer model, from which fault trees are generated, formed by the *application*, *resource*, and *physical* layers and their mappings. A CCF analysis is performed on the fault trees to verify the absence of possible common faults between the redundant elements and to validate the ASIL decomposition.

Keywords: ADAS · ASIL decomposition · Automotive architecture
Common-Cause fault analysis · Fault trees · Functional safety
ISO 26262

1 Introduction

Automotive Safety Integrity Level (ASIL) decomposition is a standardized practice presented in *ISO 26262: Road Vehicles - Functional Safety* [8]. This technique is used to reduce the criticality of safety requirements. It is generally applied during the allocation of the ASIL values to the safety requirements. The ASIL value of a requirement corresponds to a minimum ASIL that the system, which consists of a given mapping of applications, resources, and locations, must be able to achieve. When sufficiently independent architectural elements are present, the safety requirements can be split into less critical ones and mapped to the independent elements.

Figure 1 shows an example of a simple application and its mapping to the resources (Fig. 1a) and a corresponding version in which the processing part *proc* is implemented by two different functional nodes, *proc1* and *proc2*, and executed by different processors, *ecu1* and *ecu2* (Fig. 1b). The *split* and *merge* nodes provide the safety mechanisms to obtain the correct application functionality

with high reliability. They are implemented in this example by the sensor and the actuator respectively. The application layer contains the ASIL related to a safety requirement, while the resource layer has ASIL specifications that must satisfy the application requirements. The implementation resources are then mapped to the physical layer.

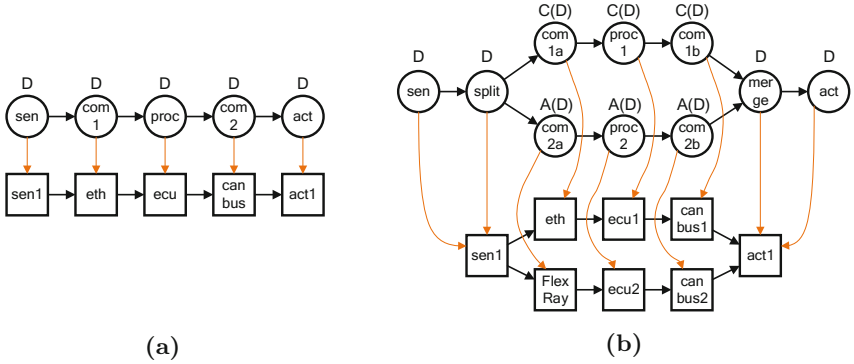


Fig. 1. Example application mapped on a resource graph (1a) with redundancy (1b). The ASIL requirements for the application are shown above the application nodes. The notation $X(Y)$ refers to a decomposed requirement in which X is the new value and Y the original.

To validate the ASIL decomposition shown in Fig. 1, according to the ISO 26262 standard, the redundant elements must be independent, meaning that they cannot have Common-Cause Faults (CCFs) that could result in a system failure [8]. The independence must be analyzed in terms of software and hardware design and implementation, failures of adjacent elements, environmental factors, failure of common external resources, etc. The ASIL decomposition can be approved only after the analysis of the CCFs.

In this paper we approach the ASIL decomposition in a bottom-up fashion. Compared to a top-down approach, where the ASIL requirements are allocated to an existing architecture, we modify the architecture introducing independent elements on which the redundant requirements can be allocated.

To this end, we present a three-layer model of automotive Electrical and Electronic (E/E) architectures. It is used to analyze the system with automated tools, validate the ASIL requirements from the mapping of the applications, and introduce system redundancy by modifying the structure of the architecture. From the architecture model we generate fault trees for each application that is executed in the vehicle. We use application, resource, and physical space model to analyze the independence of the redundant parts of the system. A model implemented since the early stages of the development phase helps the system architects to maintain proper documentation and to trace the requirements on the implementation. When comparing different solutions, a model-based approach

helps in making the trade-offs between safety, availability of the products, costs, and performance of the implementation.

An inspection of the fault trees allows the detection of a CCF that will cause breaks in the modules's independence assumptions, exposing the situations in which the ASIL decomposition would not be valid.

The novel contribution of this work is:

- A three-layer model that consists of *application*, *resource*, *physical* layers, and their mappings, that explicitly expresses redundancy with specific application and resources elements, to perform the ASIL tailoring process on the implementation level;
- Automated fault trees generation from the model, which are used in the CCF analysis. This validates the independence of redundant elements, as required by the ASIL decomposition process described in the ISO 26262 standard;
- Model transformations to modify the degree of redundancy of the system and lower the ASIL requirements for single elements, while maintaining the ASIL of the system as a whole.

The rest of the paper is organized as follows: Sect. 2 provides an overview of the ISO 26262 safety standard. Section 3 describes the architecture model that is used in this work, and Sect. 4 discusses redundancy in terms of model transformations. Section 5 introduces fault trees and the generation algorithm to synthesize them from the architecture model. Section 6 presents the related work and Sect. 7 concludes the paper by summarizing our results.

2 ISO 26262: Road Vehicles - Functional Safety

The ISO 26262: *Road Vehicles - Functional Safety* standard, published in 2011, addresses the safety aspects of automotive E/E architectures, considering both random and systematic system failures. It is an automotive-specific adaptation of the IEC 61508 standard [7], which focuses on functional safety of general electronic systems.

The ISO 26262 standard is divided into 10 parts, analyzing safety requirements during all the product life-cycle. It provides guidelines on the management of safety requirements, as well as which safety requirements are necessary for the concept phase of the product, its hardware and software development, the production and the validation of the system. Moreover, it provides guidance on ASIL-oriented requirements and decomposition. A second edition of the standard will be published in 2018 focusing on motorbikes and providing guidance on the application of the standard and ASIL definition to hardware components.

2.1 Automotive Safety and Integrity Level

Safety can be measured with the Automotive Safety Integrity Level (ASIL) concept, which is similar to the Safety Integrity Level (SIL) of IEC 61508.

The ASIL system uses a risk-based approach that takes into account the *Severity*, *Exposure*, and *Controllability* of a potential harm. There are five possible levels: from the most critical ASIL D to the least critical ASIL A and a QM (quality management) level that refers to non-safety-critical items. Figure 2 shows how the ASIL values are calculated based on the three risk parameters. The highest level D corresponds to all the risk parameters being at their maximum: S3 corresponds to life-threatening or fatal injuries, E4 to a high probability of exposure and C3 to a difficult to control or uncontrollable risk.

2.2 Requirement Decomposition

The ISO 26262 standard “provide(s) rules and guidance for decomposing the safety requirements into redundant safety requirements to allow ASIL tailoring at the next level of detail” [8]. Lower ASIL requirements for the implementation resources on which the application nodes are mapped on could be necessary for three main reasons during the product development:

1. Elements with the maximum criticality level are not available. Creating ASIL D compliant devices is a difficult task, and often the highest safety level can be achieved only by exploiting the knowledge of the application that the device will support. This is not possible for general purpose elements or resources that are shared by many applications.
2. High-ASIL software is difficult and expensive to develop and test. The same holds for the software development tools used;
3. The production process used to create a safety-critical component is expensive. Decomposing the system into less-critical elements may be the most cost-efficient solution.

		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Fig. 2. ASIL determination table

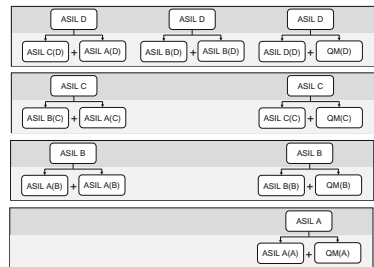


Fig. 3. Possible decomposition schemes

Figure 3 shows the acceptable ASIL decomposition schemes defined by the standard, which follow the rule of Eq. 1. To the ASIL values, QM to D, we assign a number from 0 to 4, and the following relation must be satisfied:

$$ASIL_{orig} \leq \sum ASIL_{decomp} \quad (1)$$

The standard uses the notation $ASIL_{decomp}(ASIL_{orig})$ to mark which elements have been decomposed and trace the original requirement. Additional procedures must be carried out when decomposing ASIL D requirements, for example, the test and the integration of each decomposed element shall be implemented in compliance with ASIL C. In particular, when a requirement is decomposed into redundant elements, it is necessary to establish independence between them for the original safety requirement to be correctly satisfied. For example, the redundant elements should not depend on a common resource, such as a shared battery, that could cause them to fail simultaneously, i.e. a CCF.

3 Three-Layer Architecture Model

When describing Advanced Driving Assistance Systems (ADAS) or Autonomous Driving (AD) related applications, the *sense-think-act* paradigm is generally used. It is a common concept used in Robotics, which separates an application into three main domains:

- (a) *Sense*: an application will always start by collecting information about the surrounding environment or the vehicle status from one or more sensors.
- (b) *Think*: the collected data is then processed. Different design approaches can be used to determine if it will happen, for example, in a centralized architecture, where a single module will analyze the data, or in a distributed fashion, in which multiple modules will analyze the different sensor data.
- (c) *Act*: the final part of an application involves the actuators, which modify the status of the vehicle.

In this work we assume that all applications follow this paradigm, and in the application graph a path from each actuator to at least one sensor always exists.

3.1 Model Description

Modeling the automotive E/E architecture is necessary to analyze the system. To validate ASIL decomposition it is necessary to include both the descriptions of the applications, the implementation resources used, and the physical space of the vehicle.

A three-layer approach is used: the architecture is described in terms of *application*, *resource*, and *physical* layers. The *application* layer contains disjoint application graphs, while the *resource* and *physical* layers contain one graph.

The *application layer* can contain multiple graphs, each describing a different application. Each application is related to a specific safety requirement, for example availability of the system for a certain task, derived from the safety goals, analyzed during the Hazard Assessment and Risk Analysis (HARA) phase. The *application layer* describes the functional architecture of the vehicle by defining the relationships between the software nodes via a directed cyclic graph

$G = (N, E)$ for each application, where N is the set of software nodes and E is the set of edges that connect the nodes. Each node has an ASIL requirement, which is originally inherited from the initial safety requirement, but can be lowered by the ASIL decomposition procedure. The edges indicate information flow between the nodes, but do not have any capacity or timing properties, which are expressed by explicit communication nodes. Each node has a specific type:

- (a) *Functional*: the computational aspects of an application;
- (b) *Communication*: the communication aspects of an application;
- (c) *Sensor*: the data source of an application;
- (d) *Actuator*: the data sink of an application;
- (e) *Splitter*: node that replicates the input data to its output ports;
- (f) *Merger*: node that compares the redundant inputs and ensures only correct outputs are forwarded.

Note that the *splitter* and *merger* nodes are necessary to describe the redundant elements of the system, and will be discussed in the following sections.

The *resource layer* describes the implementation architecture of the vehicle, comprising of hardware and software elements. The resources are expressed with a directed cyclic graph $H = (R, L)$, in which R is the set of resources and L is the set of links that connect them. Each resource can provide multiple types e.g.:

- (a) *Functional*: a resource on which the application functional nodes can be mapped on, like a processor or a controller;
- (b) *Communication*: resources that represent the different types of automotive networks (LIN, CAN, FlexRay, MOST, Ethernet) or direct connections;
- (c) *Sensor*: a resource that collects data, like a camera or a wireless receiver;
- (d) *Actuator*: a resource that interacts with the physical environment by executing the desired operations, for example the braking actuator;
- (e) *Splitter*: a resource capable of forwarding the data received on an input ports to multiple output ports;
- (f) *Merger*: a resource capable of deciding which input data is correct and forwards it to its output ports.

We model generic resource-resource dependencies in the *resource* layer. To show one example, we use the power supply, but any other shared resource can be modeled in the same way and included in the CCF analysis.

- (g) *Power Source*: a resource that provides the power supply for other resources, for example a battery;
- (h) *Power Line*: a resource that distributes the power supply to other resources.

Each resource has a set of types, and the application nodes with that type can be mapped on that resource. Hybrid resources can be described properly by the model, for example a gateway would be both a *Functional* and a *Communication* resource, and might have *Splitter* or *Merger* capabilities too. Each resource has

an ASIL value, representing the maximum ASIL value that it can satisfy for a specific safety requirement, usually referred to as a ASIL-X ready resource.

The *physical layer* is described similarly by a cyclic graph $F = (P, C)$, where P is the set of physical locations and C the set of connections. The description of the physical space is inspired by [10], in which the authors focus on the study of the wiring costs in an E/E architecture and they model the system to analyze wire routing and splice allocations. In our proposed approach, the physical locations can describe: the areas of the vehicle in which the ECUs and hardware components can be placed, which have a limited available space, and the paths in which the communication wires can be positioned and their length.

The interconnections between the different graphs show the mapping of the applications to the hardware resources, and of the hardware resources to the physical locations of the vehicle. Figure 1 is an example of the first two graphs and the relationships between them; it does not show the physical graph to which the resources would be mapped.

4 Model Transformations

Reducing the criticality of each module as much as possible apparently lowers the cost of the product while maintaining high safety. In practice more complications are introduced in the design: safety mechanisms must ensure that the proper functionality is preserved, new communication interfaces are added to the architecture, and a system-level analysis must be performed to ensure that the redundant elements are sufficiently independent.

As a base example, Fig. 4 shows the new elements that are introduced in the architecture after the duplication of a single node n , which has only one input and one output for simplicity's sake:

- n_s has a *splitter* type, it collects the inputs and redirects them to the redundant paths;
- c_{1a} and c_{1b} are the new communication nodes that describe the channels between the splitter and the functional nodes;
- n_1 and n_2 are the redundant functional nodes;
- c_{2a} and c_{2b} are the new communication nodes that describe the channels between the two functional nodes and the merger;
- n_m has a *merger* type, it checks the input correctness of the data from the redundant paths and forwards only correct data.

Both the *splitter* and the *merger* nodes are single points of failure for the applications, which means that they will be safety-critical elements that must have at least the same ASIL requirements as the original node n . They perform generic operations on the inputs and outputs of the redundant blocks, for example a merger could be a comparator of a classic k-out-of-n model [1] or part of an health monitoring system which decides which output to use. The other elements of the two branches instead follow the rule presented in Eq. 1.

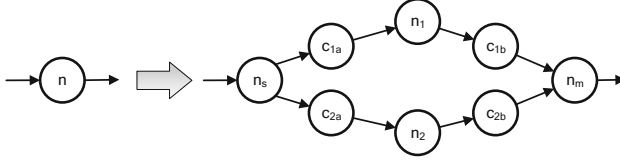


Fig. 4. Node duplication

A transformation of the resources can be applied in a similar way. High reliable *splitter* and a *merger* resources will manage the redundant independent resources. The replicated application nodes do not always have a one-to-one relationship with the transformed resources. In a bottom-up approach, the designer will make this kind of transformations to the applications and resources layers to create redundant architectures.

Even from a simple transformation, it is clear that replication will introduce a lot of complexity in the system. From a single safety-critical node we introduce at least two nodes, the *splitter* and the *merger*, with the same ASIL value as the original one. In this case, since their functionality is very specific, it is possible to obtain these elements with contained costs compared to a more generic safety-critical one. Moreover, new connections are created and additional latency is introduced by the extra communication and the splitter and merger functionality. New constraints for the design are introduced to meet the independence requirements: redundant application nodes must be mapped on independent resources, and independent resources must be positioned in independent locations. In this work we consider only the safety aspect of these modifications.

5 Common-Cause Fault Analysis for ASIL Validation

The information provided by the *application*, *resource*, and *physical* layers allows us to compute the ASIL value obtained with the implementation of each application. If the obtained value is lower than the requirements, it means that the resources cannot satisfy them, and either a different mapping or a different implementation must be used.

The computed ASIL value requires a Common-Cause Fault analysis performed on the three layers of the model to be valid. This analysis can either be manual or automated.

In this work, we generate a fault tree for each application, which is used for an automatic CCF analysis. This analysis recognizes redundancy in the model by searching for *splitter-merger* combinations, and uses the nodes and resources dependencies to determine any possible CCF.

This analysis can also be used to validate a new model after a transformation.

5.1 Fault Trees in Automotive Systems

Fault Tree Analysis (FTA) is a common top-down Safety Analysis, in which an undesired top-level event is identified and then its causes are considered.

In this work we consider as the top-level event the failure of an application for a specific safety goal, for example the system availability, that manifests itself through the failure of at least one of the actuators. Each node, starting from the actuators, can fail because of different reasons:

- Internal failure of the hardware resource on which the application node is mapped or of the software component that implements the functionality;
- Failure of the location on which the used resource is mapped;
- Dependent resource failure. The resource on which the application node is mapped may depend on other resources, such as the power supply;
- Input Failure. Failure of node A that provides data to node B leads to the failure of the node B .

Figure 5 shows the fault tree generated for each node. The same structure is generated for each input application node, until the final sensors are reached, according to the assumption of *sense-think-act* applications. This type of fault tree is based on [6], in which the authors use Dynamic Fault Trees to describe ADAS related applications and perform a FTA.

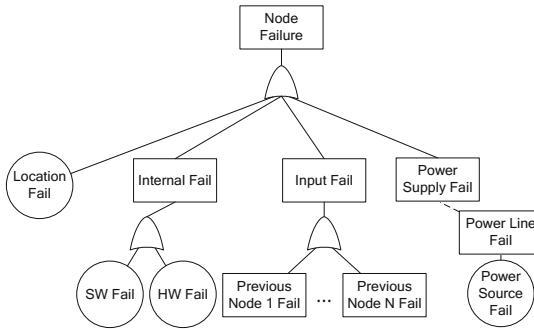


Fig. 5. Subtree for each application node

The internal failure base event could be further developed as in [6], where the hardware and the safety mechanisms implemented in the resources are considered.

5.2 Fault Tree Generation

The fault tree generation algorithm is based on [11]. We assume that the failure of a safety requirement corresponds to the failure of at least one of the related

application’s actuators. All the actuators are assumed to have the same importance for the success of each application. Assuming a *sense-think-act* paradigm, we can always expect to find a path from an actuator to a sensor.

Algorithm 1 accepts an application graph G and the top-level event e_T as inputs, and then calls the recursive procedure *DevelopSubTree* for each of the application actuators. The function *MappedResource* returns the resource on which an application node is mapped, while the function *MappedLocation* returns the physical location on which the resource is positioned.

For each node, a top fault event is created, to which the four possible fault events described previously in this section are connected via an OR gate. The function *Predecessor* finds all the inputs of an element in its graph. It is used to find all the inputs of the current application graph node, for which a new sub-tree will be generated with the *DevelopSubTree* procedure and connected to the input failure event of the parent node. In case of a *merger* type node, this connection is made through an AND gate, meaning that an input failure is acquired only when all the different input branches of a redundant part of an application fail. In case of a *sensor* type node, no input nodes can be found, the input failure event is deleted and the sub-tree is returned. For all the other nodes, a failure of any of its input leads to a fault, so they are connected with an OR gate.

The power supply event is developed in the *DevelopResourceSubTree* procedure, which is similar to *DevelopSubTree*, but works on the resource and physical layers only. This procedure is generic for resource-resource dependencies, and in this example we use it to generate the power supply fault tree. Each resource can be connected to one or more power source via power lines, which are modeled as resources in the graph. This recursive function travels through the graph, from a resource to each of its power supply, instantiating three types of events: a fault in the power line or power source resource, a fault in the physical location and a fault from the parent power supply resource.

Each element of the fault tree graph is related to the relevant architecture node. The fault trees are saved as graphs, but also exported in the text based Galileo format [5], which is a generic format supported by commercial FTA tools. By adding to the graph information related to the fault rates of each element and the fault probabilities of external events, it is possible to analyze them with the commercial tools and compute reliability metrics for the design.

5.3 Example Scenarios

In this section we discuss three possible scenarios in which our analysis can provide important information to the system architect. In Fig. 6a we see the application showed previously in Fig. 1b, its mapping to the hardware resources and their positioning in the physical space. The redundant communication resources are correctly placed in different parts of the vehicle so that they will not suffer from CCF related to the environment, while *ecu1* and *ecu2* are placed in the same location *f2*. This is an example in which a CCF related to a common location is found, and a warning is issued to the designer by our analysis tools. Note that in this situation, *proc1* and *proc2* are two different function. If, for

Algorithm 1. Fault Tree Generation

Inputs: Application graph G , Resource graph H , Physical graph P , Top-Level Event e_T

Output: Fault Tree F

```

1: procedure GENERATEFT( $G, e_T$ )
2:   for  $n_i \in N$  s.t. NodeType( $n_i$ ) = actuator do
3:      $e_i$  = DevelopSubTree( $n_i$ )
4:   CreateGateOR( $e_T, \forall e_i$ )
5: procedure DEVELOPSUBTREE( $n$ )
6:    $F_{sub}$  = CreateNodeEvent( $n$ )
7:    $r$  = MappedResource( $n$ )
8:    $p$  = MappedLocation( $r$ )
9:    $e_1$  = CreateResourceBasicEvent( $r$ )
10:  if NodeType( $n$ ) != sensor then  $e_2$  = CreateInputFaultEvent( $n$ )
11:   $e_3$  = CreatePowerSupplyEvent( $r$ )
12:   $e_4$  = CreateLocationBasicEvent( $p$ )
13:  CreateGateOR( $F_{sub}, (e_1, e_2, e_3, e_4)$ )
14:  if NodeType( $n$ ) = sensor then Return  $F_{sub}$ 
15:  for  $n_j \in$  Predecessor( $n$ ) do
16:     $e_j$  = DevelopSubTree( $n_j$ )
17:  for  $s_k \in$  Supply( $r$ ) do
18:     $e_k$  = DevelopResourceSubTree( $r_j$ , powerSource, powerLine)
19:  CreateGateOR( $e_3, \forall e_k$ )
20:  if NodeType( $n_j$ ) = merger then
21:    CreateGateAND( $e_2, e_j$ )
22:  else
23:    CreateGateOR( $e_2, e_j$ )
24:  Return  $F_{sub}$ 
25: procedure DEVELOPRESOURCESUBTREE( $r, types$ )
26:    $F_{resSub}$  = CreateResourceDependencyEvent( $r$ )
27:    $p$  = MappedLocation( $r$ )
28:    $e_1$  = CreateResourceBasicEvent( $r$ )
29:    $e_2$  = CreateLocationBasicEvent( $p$ )
30:   if ((Predecessor( $r$ ) != NULL) and (ResourceType(Predecessor( $r$ ))  $\in$ 
      types)) then  $e_3$  = CreateResourceInputEvent( $r$ )
31:   CreateGateOR( $F_{resSub}, e_1, e_2, e_3$ )
32:   if Predecessor( $r$ ) = NULL then Return  $F_{resSub}$ 
33:   for  $r_j \in$  Predecessor( $r$ ) do
34:     if ResourceType( $r$ )  $\in$  types then
35:       DevelopResourceSubTree( $r_j$ )
36:   CreateGateOR( $e_3, \forall r_j$ )
37:   Return  $F_{resSub}$ 

```

example, the function *proc1* was used in both redundant branches, then the possibility of a CCF derived from a systematic fault in the function would have been highlighted by the tool.

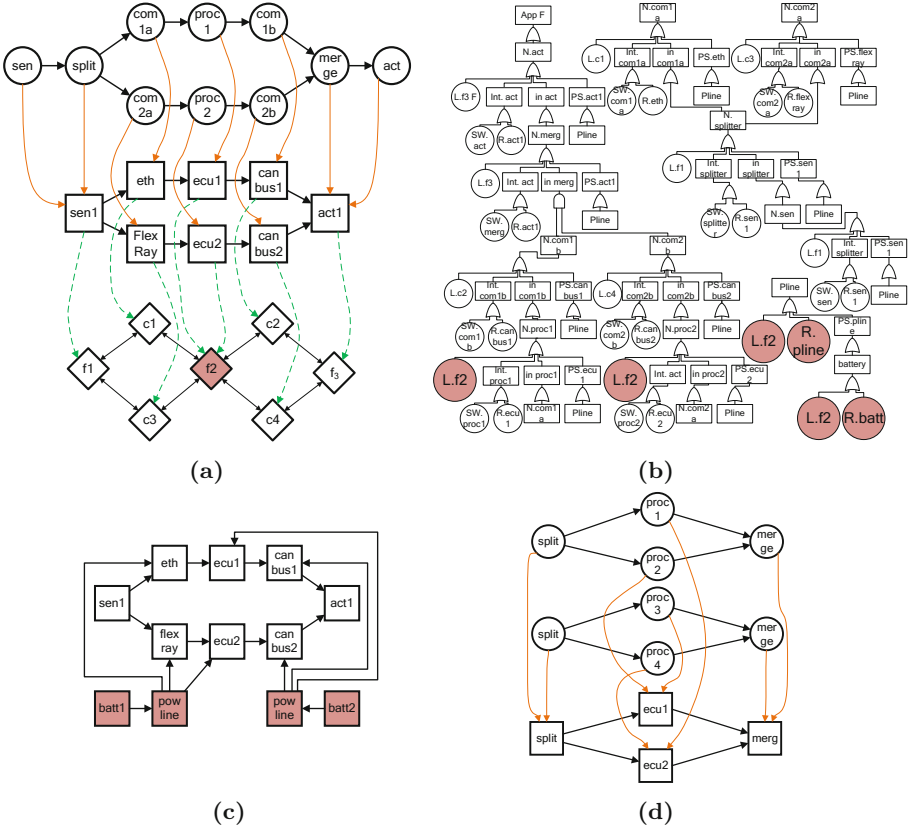


Fig. 6. Example of different scenarios and the CCF analysis results

A second scenario in which a CCF is found is shown in Fig. 6c. Two different power supplies are used for the redundant paths of the first example. Since both the power lines are connected to elements of both the branches, a failure of a single supply will lead to a system failure. The designer is again warned about the possible CCF, which invalidates the ASIL decomposition.

Figure 6d shows an example in which two applications have redundant elements. Since the two applications are independent from each other, it is possible to map them on the same independent resources. The ASIL decomposition assumptions will be valid, in this scenario no CCFs are present.

Figure 6b shows the generated fault tree for the first scenario, when considering a single power line and a battery to supply all the resources. The base events marked in red represent the CCF related to the placement of the redundant ECUs to the same locations and their common power supply. For this simple illustrative scenario the fault tree contains 59 events, 32 OR gates and one AND gate. In a realistic automotive system the number of nodes for each safety requirement is higher, and in combination with the high number of safety

requirements the resulting graph will contain thousands of nodes. Without an automated framework, the safety engineers would have to manually create and maintain those graphs, resulting in a more time consuming safety analysis and validation of the system, with possibilities for human errors.

6 Related Work

In [12] the authors present a method to allocate the Safety Integrity Levels (SIL) in a top-down automated process, supported by the commercial tool HiP-HOPS. The system architecture is analyzed to find which elements affect the different safety requirements and a top-down allocation of the minimum SIL values is performed. Our approach differs from theirs since we modify the architecture with model transformations in order to satisfy redundant safety requirements with new independent resources. However, the two methods can be used in combination during different phases of the design to validate each other's results.

Additional information about ASIL decomposition are given in [4, 17], where it is made clear that introducing redundancy in safety-critical systems is a difficult task and must be taken care of by making appropriate considerations: adding an additional resource without considering its position inside the system and its dependencies is not enough for a valid decomposition.

In [6] Dynamic Fault Trees and their analysis are used to provide information about the reliability of automotive systems using the STORM tool. Since our fault trees are generated in the common Galileo format, they can be analyzed with commercial or open-source tools like STORM to obtain parameters such as the Mean Time To Failure of the system. An equivalent model that can be used for the safety analysis instead of the Fault Tree is the Reliability Block Diagram [3], but we decided to use Fault Trees to focus on the failure of the safety requirement.

In [10] the authors introduce a model for the wires used in an automotive system, optimizing the wire routing to minimize the harness expenses. This model contains more details related to these than our model, and could be used to describe the physical connections in a more refined way.

The authors in [14] describe an approach that supports the mapping of software elements to hardware resources, using the AutoFocus3 tool, in an AUTOSAR and ISO 26262 context. We currently perform the mapping step manually. A similar Design Space Exploration that considers tasks scheduling, latency, and costs is necessary to automate this process.

The work presented in [16] takes a step into the direction of a generalized functional architecture for autonomous vehicles: currently there is no standard Autonomous Driving system, but a step towards a common solution is necessary to speed up the development and validation parts, included the safety case analysis. With our work provide an environment that allows a system architect to describe generic automotive systems to compare them and decide on the most efficient solutions. It will help determine which will be the trends and most appropriate decisions for the future automotive systems. For example, the discussion between Centralized [15] versus Distributed [9] architecture design, but

also more types of architectures like Domain-based [13] or more recent ideas like Zonal [2], makes more sense when compared on a real system. They all have their pros and cons. By modeling the system and compare the same applications with different topologies it is possible to determine the efficiency of each solution.

7 Conclusions

In this work we presented a system-level analysis that validates an ASIL decomposition according to the ISO 26262 standard. We focus on the validation of the decomposition applied on a transformed system architecture, in which the designer has introduced redundancy via specific elements, hence a bottom-up method for the development of the redundant parts of the system.

Our validation is based on a CCF analysis performed on fault trees generated from the system architecture model. The model describes the system in terms of *applications*, *resources*, and *physical* layers and their mappings. The model, the fault tree generation, and the CCF analysis are implemented in Python, using the *graph-tool* library.

Our results show how a structured method to the ASIL decomposition process is necessary for a formal validation of the redundant system. We have seen that, even for a simple and artificial scenario, the generated fault trees contain a high number of events, making the CCF analysis a complex task to perform manually. By automating part of the safety analysis, the development of a safety-critical product becomes faster and less prone to human error.

Acknowledgements. The work in this paper is supported by TU/e Impuls program, a strategic cooperation between NXP Semiconductors and Eindhoven University of Technology. The authors thank all reviewers for their helpful comments and suggestions that helped improve and clarify this paper.

References

1. Boland, J.P., Proschan, F.: The reliability of K out of N systems. *Ann. Probab.* **11**(3), 760–764 (1983)
2. Brunner, S., Roder, J., Kucera, M., Waas, T.: Automotive E/E-architecture enhancements by usage of ethernet TSN. In: 13th Workshop Intelligent Solutions in Embedded Systems, pp. 9–13, June 2017. <https://doi.org/10.1109/WISES.2017.7986925>
3. Čepin, M.: Reliability block diagram. *Assessment of Power System Reliability*, pp. 119–123. Springer, London (2011). https://doi.org/10.1007/978-0-85729-688-7_9
4. D’Ambrosio, J.G., Debouk, R.: ASIL decomposition: the good, the bad, and the ugly. Technical report, SAE Technical Paper (2013)
5. Galileo User’s Manual & Design Overview. <https://www.cse.msu.edu/~cse870/Materials/FaultTolerant/manual-galileo.htm>
6. Ghadhab, M., Junges, S., Katoen, J.-P., Kuntz, M., Volk, M.: Model-based safety analysis for vehicle guidance systems. In: Tonetta, S., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2017. LNCS, vol. 10488, pp. 3–19. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66266-4_1

7. IEC 61508 Edition 2.0. Principles and Use in the Management of Safety (2010)
8. ISO 26262–2011: Road vehicles - Functional safety - Part 9: ASIL-oriented and Safety-oriented Analyses (2011)
9. Jo, K., Kim, J., Kim, D., Jang, C., Sunwoo, M.: Development of autonomous car - part II: a case study on the implementation of an autonomous driving system based on distributed architecture. *IEEE Trans. Ind. Electron.* **62**(8), 5119–5132 (2015). <https://doi.org/10.1109/tie.2015.2410258>
10. Lin, C.W., Rao, L., D’Ambrosio, J., Sangiovanni-Vincentelli, A.: Electrical architecture optimization and selection-cost minimization via wire routing and wire sizing. *SAE Int. J. Passeng. Cars-Electron. Electr. Syst.* **7**(2014–01–0320), 502–509 (2014). <https://doi.org/10.4271/2014-01-0320>
11. McKelvin Jr, M.L., Eirea, G., Pinello, C., Kanajan, S., Sangiovanni-Vincentelli, A.L.: A formal approach to fault tree synthesis for the analysis of distributed fault tolerant systems. In: Proceedings of the 5th ACM International Conference on Embedded Software, pp. 237–246. EMSOFT 2005, ACM, New York (2005). <https://doi.org/10.1145/1086228.1086272>
12. Papadopoulos, Y., et al.: Automatic allocation of safety integrity levels. In: Proceedings of the 1st Workshop on Critical Automotive Applications: Robustness and Safety, pp. 7–10. ACM (2010)
13. Reinhardt, D., Kucera, M.: Domain controlled architecture - a new approach for large scale software integrated automotive systems. *Pervasive Embed. Comput. Commun. Syst.* **13**, 221–226 (2013)
14. Schtz, B., Voss, S., Zverlov, S.: Automating design-space exploration: optimal deployment of automotive SW-components in an ISO26262 context. In: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (June 2015). <https://doi.org/10.1145/2744769.2747912>
15. Sommer, S., et al.: RACE: a centralized platform computer based architecture for automotive applications. In: IEEE International Electric Vehicle Conference, pp. 1–6. IEEE (2013)
16. Ulbrich, S., et al.: Towards a functional system architecture for automated vehicles. arXiv preprint [arXiv:1703.08557](https://arxiv.org/abs/1703.08557) (2017)
17. Ward, D.D., Crozier, S.E.: The uses and abuses of ASIL decomposition in ISO 26262. In: 7th IET International Conference on System Safety, Incorporating the Cyber Security Conference, pp. 1–6. IET (2012)



Assurance Benefits of ISO 26262 Compliant Microcontrollers for Safety-Critical Avionics

Andreas Schwierz¹(✉) and Håkan Forsberg²

¹ Research Center: Competence Field Aviation, Technische Hochschule Ingolstadt,
85049 Ingolstadt, Germany

Andreas.Schwierz@thi.de

² School of Innovation, Design and Engineering,
Division of Intelligent Future Technologies, Mälardalen University,
721 23 Västerås, Sweden
hakan.forsberg@mdh.se

Abstract. The usage of complex Microcontroller Units (MCUs) in avionics systems constitutes a challenge in assuring their safety. They are not always developed according to the assurance requirements accepted by the aerospace industry. These Commercial off-the-shelf (COTS) hardware components usually target other domains like the telecommunication branch, because of the volume of sales and reduced liability. In the last years MCUs developed in compliance to the ISO 26262 have been released on the market for safety-related automotive applications. The avionics market could profit taking credit for some of the activities conducted in developing these MCUs. In this paper we present evaluation results based on comparing assurance activities from ISO 26262 that could be considered for compliance to relevant assurance guidance for COTS MCU in avionics.

Keywords: Microcontroller · DO-254 · Assurance · Reuse · Avionics
ISO 26262 · COTS

1 Introduction

COTS hardware components are ubiquitous in Airborne Electronic Hardware (AEH) designs as execution platform for airborne software. In safety-critical avionics systems this task is dominantly performed by Microprocessor Units (MPUs). Due to the high demand for more integrated semiconductor products the market share of “pure” MPUs went down in favour of MCUs solutions. Thus avionics system suppliers are encouraged to embed MCUs today.

In contrast to MPUs, MCUs predetermine many features like memory, bus architecture or peripherals that were formerly designed by the AEH manufacturer in a rigorous manner adequate for the quality and safety needs in the aviation industry. This process-based and product-based approach is called

Development Assurance (DA) and for AEH it is described in the guidance document RTCA/DO-254 [1] released in 2000, which should be used within the context of SAE ARP4754A safety considerations. This industry consensus standard was devoted to address all kinds of complex hardware: Line Replaceable Units (LRUs), Circuit Board Assemblies (CBAs), Programmable Logic Devices (PLDs), Application-Specific Integrated Circuits (ASICs) and COTS components. This resulted in a very abstract description of process objectives which enables a broad spectrum of interpretation in industrial practise. Therefore, the Federal Aviation Administration (FAA) reduced the scope of the RTCA/DO-254 to PLDs or ASICs and concentrated their guidance on this topic [2]. As more AEH manufacturers envisaged to embed complex COTS components as Multi Core Processors (MCPs), Graphics Processing Units (GPUs) and highly integrated MCUs, Certification Authorities (CAs) also published guidance to emphasize aspects for considerations to clarify their usage in safety-critical aircraft functions [3–5]. This approach is called COTS hardware component assurance.

The DA of AEH described in RTCA/DO-254¹ and the COTS hardware component assurance have the same objective, which is to assure that a hardware safely performs as intended in its operational context. But the method is inevitably distinct because of the nature that COTS hardware components may not have been developed according to the RTCA/DO-254 or that COTS manufacturers do not disclose required development artefacts to be able to demonstrate compliance afterwards. So process-based evidence of the design life cycle could not be claimed as aircraft systems concerns may not have been regarded during the development of the COTS product.

Avionic manufacturers already employ COTS components actually intended for other domains. Hardware is requested with a long market availability and operable under harsh environmental conditions. These component properties are characteristic for the automotive domain. Functional safety is at least since 2011, when the ISO 26262 standard [6] has been released, a major concern for Original Equipment Manufacturers (OEMs) and also for many suppliers of automotive parts like integrated circuits. MCUs developed in compliance with ISO 26262 are designed for safety-critical applications. AEH manufacturers want to utilize the fact that safety plays an essential role in the automotive domain [7].

In a previous work [8] we evaluated in an objective by objective comparison between the domain standards RTCA/DO-254 and ISO 26262 their design integrity. The initial purpose was to examine similarities of both standards in order to enable an AEH manufacturer to argue about the omission of already satisfied RTCA/DO-254 objectives by the semiconductor manufacturer. But during the evaluation it became obvious that determined gaps in the ISO 26262 cannot be closed by the AEH manufacturer. Further access to detailed design data would be necessary to accomplish the DA activities and to conduct a safety analysis since failure consequences within the context of system operation could

¹ Including CA recommendations that supplement this DA approach.

be different in different domains. However, the identified similarities justified further research.

In this article we present an analysis that shifts the focus from examining the DA of the ISO 26262 to how the ISO 26262 compliance statement of an MCU can be beneficial in the COTS hardware assurance process. In order to increase the relevance of this research for safety-critical avionics, only MCUs are considered that targeted the highest Automotive Safety Integrity Level (ASIL)² and operates in Lock-Step Mode (LSM). The European Aviation Safety Agency (EASA) Certification Memorandum (CM) section nine [5] currently describes in the greatest extent which COTS assurance activities for MCUs shall be conducted. Some of these activities are inherited from RTCA/DO-254. EASA's CM therefore serves as analysis basis to identify aspects of the ISO 26262 compliance statement that can be beneficially reused. In addition, some artefacts required for MCUs being compliant to ISO 26262 overlap with compliance results required for RTCA/DO-254, and as such may be reused for aviation purposes.

This paper is structured as following: Sect. 2 describes how assurance is achieved for avionics systems in general and how it differs if complex MCUs shall be embedded. The application of CA documents are discussed in detail in Sect. 2.2. Evaluation of ISO 26262 compliant MCU benefits in COTS hardware component assurance is performed in Sect. 3. Conclusions together with limitations of the reuse argument are given in the last section.

2 Assurance Methods for Avionics

The term assurance methods is currently often used in the avionics domain [9–11]. In general, assurance can be defined as the actions that provide appropriate confidence and evidence that a product or process meets its requirements [12]. It delivers reasons why the confidence on achieving a claim is so justifiable [13]. Most assurance activities are conducted to establish a convincing confidence [14]. So assurance intends to reduce the uncertainty about the correct realisation of the product. In a requirements-based product development this means, that the requirements specification meets the real-world needs (*validation of requirements*)³ and that the product is a correct implementation of the requirements specification (*verification of requirements*).

For avionics systems the airworthiness requirements are on the top of their requirements specification. Summarized, it has to be assured that the avionics system design is appropriate for the intended function and that its function is provided as defined in its operational context (environmental and operating conditions of the aeroplane). These are for example prerequisites to ensure that it is extremely improbable that safety-critical AEH contributes to a catastrophic

² This should not imply an equivalence or comparability between ASIL and Development Assurance Level (DAL).

³ Including all derived requirements that have to be validated to ensure that they not cause any hazardous condition.

failure condition at aircraft level that harms human life. For safety-critical systems, assurance methods are necessary to deliver enough credit to justifiably state that the system is safe in its context.

The assurance approach varies depending on the context and which aspects should be assured. The first approach is DA and can be applied for avionics systems that mainly comprise components manufactured alongside the avionics development life cycle see Sect. 2.1. The second approach is COTS hardware assurance and shall be used if already developed components like complex MCUs shall be used in AEH see Sect. 2.2. Both approaches have to be coordinated with each other to enable e.g. an safety assured integration of an MCU in self-developed CBA.

2.1 Development Assurance

Modern avionics systems are too complex in order to provide the requested level of confidence by exhaustive tests which fully characterise the system. Hence, the method of *Development Assurance* was defined to cope with this issue in different areas, system (ARP4754A, ARP4761), software (RTCA/DO-178C) and hardware development [1]. These DA areas aim to accomplish the development in a sufficiently rigorous and disciplined way so that development errors do not impact safety. DA is characterised by techniques that are applied during the whole development process in order to identify and correct errors that could occur at various steps within the development life cycle.

The DA approach described in the RTCA/DO-254 is connected to the DA approach of the system development. To ensure safety, a strong collaboration has been established between aircraft system designer and hardware level developers, where information such as requirements are exchanged and validated. System engineers need to understand correlations and consequences of hardware properties on system level. With this approach, AEH provides the functionality or behaviour as specifically requested for the selected aircraft type. The principle is a stringent requirements-driven work flow down to detailed design tightly connected to the system development process. This allows the visibility of the hardware design for all participants (engineers and CAs) and beside this, a diligent planning or control of processes shall assure that the design is a correct implementation of the requirements describing the system level objectives about safety and other product attributes.

For AEH embedded MCUs, DA on device level cannot be claimed or used as assurance method [15]. The reason is that most DA techniques are based on an ongoing development and the accessibility of development-time artefacts down to a level detailed enough to realize the hardware component and to assure its safety aspects on device level. For MCUs this is not possible as the development has already been accomplished and detailed development-time artefacts are not available. Thus, other assurance methods have to be determined which can reduce uncertainty in a similar magnitude creditable by the CAs.

2.2 MCU COTS Hardware Component Assurance

In case of DA, a component is developed as part of a certain system in contrast to COTS component that is created without a dedicated system in mind. This fact together with the broad variety of COTS components makes it a challenging task to create a guidance for COTS hardware component assurance [16]. Such guidance shall support the industry in realization of certifiable AEH, embedded with COTS components, in a practical way so that certification costs do not explode and safety can be sufficiently assured.

The latest initiatives for MCU COTS assurance by CAs in this direction resulted in the following documents⁴ and reflect the status quo:

- *CAST*⁵:
 - CAST-32A: Multi-core Processors [3].
 - CAST-29: Use of COTS Graphical Processors (CGP) in Airborne Display Systems [4].
- *EASA*:
 - Certification memorandum CM SWCEH-001: Development Assurance of Airborne Electronic Hardware [5].

This document represents the current attitude of the EASA about several certification aspects of AEH and in Sect.9 especially to COTS MCUs. The content is based on experience in COTS hardware component assurance in many certification projects gathered in the years before and funded research activities as [17].

- *FAA*:
 - Commercial Off-The-Shelf Airborne Electronic Hardware Assurance Methods - Phase 3 - Embedded Controllers [9].
 - Assurance of Multicore Processors in Airborne Systems [18].

These technical reports are the results of funded research by the FAA to develop proposals for assurance approaches for different COTS hardware.

Notable is that for different hardware component categories the assurance methods were separately considered. This faces the fact, that each technology has its own issues which shall be incorporated to provide methods that are useful in practice. Especially small companies and market newcomers are interested in guidelines as concrete as possible [19].

Guidance provided by the CAST working group describes considerations that should be followed in a COTS hardware component assurance process if a MCP or GPU is part of an MCU. In [3] the interface to the software assurance process and in [4] specific hazards for display applications are explained. They are important aspects in an assurance process but are not sufficient for MCUs assurance activities. Further for the case study in this paper, all ISO 26262 developed

⁴ None of these listed documents are binding guidance material (unless so specified by CA for the project), although they are usually used by CAs to query the safety of COTS usage before approvals are granted.

⁵ In the Certification Authorities Software Team (CAST) working group representatives from EASA and FAA working together.

MCUs must be run in LSM for safety-critical applications. Dual cores that are synchronized and operating in LSM are excluded as stated in [3]. A GPU is also not integrated in examined MCUs so [4] is not further respected.

The reports [5, 9, 17, 18] share one similarity: COTS assurance should be managed from system level in parallel or within the AEH design process⁶. However, they lack in formulating a framework that brings them all together in a coherent approach that can be related as an deployable COTS hardware component assurance process. From these reports, the EASA CM [5] can be considered as most relevant to identify necessary COTS assurance tasks for MCUs. It represents the current position of a CA and defines assurance activities as an Electronic Component Management Plan (ECMP)⁷ extension.

FAAs research report [9] does not explicitly cover activities but has identified issues to keep track of during assurance. Also, findings and recommendations can be found in this research report. Some similar to the activities in EASAs CM, e.g. usage domain analysis, integration aspects, errata handling, and configuration management, and some similar to typical ISO 26262 implementations, e.g. robustness verification. These issues, findings and recommendations have been analysed but were not identified as obvious COTS assurance tasks and therefore not included in this paper. In addition, the other FAA research report [18] concerns MCPs and is therefore excluded as explained for the CAST document [3] before.

The CM contains sixteen recommended activities, numbered with brackets from 1 to 16 (e.g. Activity [1]). These activities are referenced in this article with the same numbering but are emphasized in round brackets instead to avoid confusion with other references in the article. These activities should be considered depending on the DAL associated by a higher level safety assessment, the magnitude of Product Service Experience (PSE) traceable from different domains and the complexity of the MCU. In the subsequent text, activities are discussed for DAL A which apply for components with the highest possible safety impact. This will extend the value of scope of this article, because targeting DAL A means that all activities have to be conducted if the PSE is inadequate. The argumentation behind these additional assurance activities is not further stated in the document but is essential for the understanding on how they contribute to COTS hardware component assurance. Thus the assumed argumentation was reconstructed. Two top arguments were extracted that have to be assured:

Argument 1. The component performs as described by the manufacturer without anomalous behaviour.

Argument 2. The component as used satisfies the AEH requirements assigned within the system context.

It has to be differentiated between those arguments as the MCU was not developed according to the requirements of the AEH. How these arguments can be

⁶ Recommended also by RTCA/DO-254.

⁷ The term ECMP in the CM is misleading because typically such a process does not perform profound functional assurance activities.

supported depends on the complexity of the COTS component. For MCUs with a functional architecture classified as simple, the arguments can be fulfilled as following:

– *Argument 1*

- Verification of component behaviour on device level as specified by the manufacturer.

The simplicity of the COTS component allows to verify all requirements on the physical device.

- Substantiate the confidence of a design free from anomalous behaviour by demonstrating device maturity or quality.

Most of the confidence on device quality is already supported by the comprehensive verification effort. However, additional errata management in activity (6) and (7) shall be considered to state that the device design is stable enough. This can be demonstrated by errata decreasing over the service time on the market. Also the errata publishing policy of the manufacturer shall be adequate to be always informed about revealed problems and to achieve that errata with potential safety impacts can be handled.

– *Argument 2*

- Verification of AEH requirements on LRU or CBA level during equipment design.
- As requested by certification requirements, no single point of failure should lead to a catastrophic failure condition. This is also valid for COTS components in general. Activity (15) requests the implementation of an adequate architectural mitigation technique like dissimilar redundancy or monitoring.
- An ECMP e.g. as described in IEC TS62239.

Most of the available MCUs on the market are complex or even highly complex components. For these devices, exhaustive tests on device level can not be achieved to adequately substantiate argument 1 as for simple components. Therefore additional activities for complex or highly complex hardware are necessary, which are depicted as following for arguments 1 and 2:

– *Argument 1*

- Verification of component behaviour on device level as specified by the manufacturer.

The concept of usage domain as described in activity (4) resp. (5.1) suggest to bound the scope of device level verification only on component behaviour that is relied on or is really used by the AEH manufacturer⁸.

The determined usage domain (4) consists of requirements from the avionics manufacturer that are compliant to the COTS component specification and verified in activity (5.1) on device level. In this activity all technical

⁸ We separated the activity (5) in (5.1) usage domain verification and usage domain validation (5.2).

aspects have to be substantiated like deterministic behaviour e.g. worst case execution time, shared resources in case of MCPs or performance requirements. If the MCU is part of a partitioning concept, an analysis has to be performed as described in activity (16) to claim the robustness of this mechanism at device level⁹.

- Substantiate the confidence of a design free from anomalous behaviour by demonstrating device maturity or quality.

The verification on device level limited to usage domain aspects is not enough to mainly support argument 1. In comparison to simple COTS components, the correct behaviour assumption of complex hardware is more based on other activities like:

- * COTS manufacturer quality management and production process has to be assessed in activity (3).
- * Errata management as for simple components in activity (6) and (7). Additionally, activity (8) requests that the AEH manufacturer has to document own made experience with the hardware during the development (e.g. errata workarounds).
- * COTS manufacturers configuration management including a change process has to be assessed in activity (9) to make sure that changes are appropriately controlled and communicated. Activity (10) additionally requests a change impact analysis to identify potential extra verification effort.
- * The PSE has to be documented by activity (13) in order to determine if it is sufficient¹⁰ to omit certain assurance activities. PSE should be used conservatively for dissimilar domain treatments of accident investigations. Specifically for DAL A and B, a minimum amount of PSE has to be reported in order to exclude really novel designs to be embedded in AEH systems. Usually it is very difficult to use PSE at DAL A and B. Also, it is a good idea to use PSE as supplemental argument in addition to conducting the necessary verification and validation activities within the target system. Activity (14) further increases the confidence on the maturity and stability of the MCU by requesting evidence on the rate and fact of past modifications.

– *Argument 2*

- Usage domain validation in activity (5.2) ensures that the usage domain is consistent to system, software and hardware requirements.
- For complex COTS it is not adequate to verify requirements allocated to the MCU at equipment level as for simple components. Activity (11) requests verification and validation of these requirements coming from other hardware or software components on device level in order to get confidence about its correct integration.

⁹ Actually, we consider partitioning aspects as a specific part of the usage domain analysis, because MCU properties shall be verified on device level.

¹⁰ The metric to determine a PSE as sufficient is also defined in the CM.

- For highly complex MCUs activity (12) has to be conducted to have a clear understanding of possible device failure modes and rates depending on its configuration. Descriptions in [20] provides support for this task.
- Architectural mitigation technique external to the MCU as requested by activity (15) shall also be applied.
- An ECMP e.g. as described in IEC TS 62239.

Activity (1) and (2) were not mapped to a top argument, since determining or classifying the MCU characteristic (1) and archiving public available device data (2) are required for both top arguments. It does not matter if the MCU is classified as simple, complex or highly complex.

All these explained assurance activities of the CM are only applicable for the peripheral subsystem and other functions which are not part of the processing core. The DA of the processing core is based on the software development process compliant to RTCA/DO-178C that includes software testing on the target hardware platform. This separation is based on the assumption that other MCU functions do not interfere with the software execution on the processing core [9].

The explanations about complex COTS hardware component assurance established the basis on which in the next section the potential benefits from ISO 26262 compliant complex MCUs can be examined.

3 Benefits from ISO 26262 Compliant MCUs in AEH COTS Assurance

Derived from the introduction the following research question is asked: *How can the avionics industry benefit from ISO 26262 compliant MCUs in the course of COTS hardware components assurance?* Before starting to evaluate an ISO 26262 compliant MCU against the assurance approach from Sect. 2.2, the differences to other MCUs on the market have to be identified first. What makes these MCUs so special? These are the aspects on which COTS assurance can probably profit in comparison to other MCUs e.g. from the telecommunication domain.

3.1 Determination of ISO 26262 Specifics for Reuse

The special characteristics of interest come from the development approach defined by the process requirements from ISO 26262. During previous research we made a comparison between the DA method of RTCA/DO-254 and ISO 26262-5¹¹, which concludes that the ISO 26262 does not reach the same level of design integrity [8]¹². The reason is that only safety requirements are considered in the development life cycle of the MCU, whereas the traceability down to detailed design level is not required. For manufacturers the main focus is on

¹¹ Part five of the standard is about product development at the hardware level.

¹² This demonstrates reasonableness of a dedicated COTS assurance process see Sect. 2.2.

the safety architecture to handle random hardware failures by adequate safety mechanisms to achieve the targeted diagnostic coverage and to be able to enter a safe state if necessary or indicate failures to external components. On the device level the characteristic of a very high diagnostic coverage makes these products something special on the market and manufacturers are very encouraged in the realization and verification of the MCU's safety architecture.

The MCU development approach has to adhere to ISO 26262-5 and referenced parts. ISO 26262-10:2012 does not define conformance requirements but gives guidance especially on MCU development. It explains the safety element out of context method and describes in appendix A how it can be applied for MCUs. This concept allows the realization of a component like an MCU which is deployable to different application contexts: it is *built for reuse*. Therefore the manufacturer first assumes the safety requirements that could be allocated from the system level and architecture around the component. These assumptions are necessary to develop the MCU internal safety architecture. The system integrator has to follow the manufacturers assumptions and recommendations to preserve the integrity of the MCU safety architecture in the final system context. For ISO 26262 compliant MCUs typically an additional document type is released in order to inform the integrator about the ISO 26262 related information essential for system integration activities: the *safety manual* or *safety application note*. In ISO 26262-10:2012 section A.3.10 an example on the content of the safety manual is given.

As only suggestions for the safety manual content is provided, it is still worth to examine which aspects have been realized in published documents. In order to assess the potential benefits of the safety manual in an avionics COTS component assurance process, the content of a representative probe of three manuals from three different vendors was analysed. The selected MCUs target ISO 26262 ASIL D. They have been chosen to increase the value of the scope of this article and not if they are really suitable for the avionics industry. Thus, no analysis has been performed to check the suitability of these devices for avionics due to e.g. cosmic radiation or other environmental or functional issues such as *correct* set of interfaces. Together with a performance analysis these aspects should be checked early in the project during a selection and evaluation process before investigating a lot of effort in the other COTS assurance activities. The selected MCUs with respective safety manuals are: NXP MPC5744P [21], ST SPC56ELx [22] and TI TMS570LC4x [23]. The content analysis of these manuals resulted in the following two major topics of interest that can be found in each of the examined safety manual in different level of detail:

- MCU safety architecture: It describes how random hardware fault management is separated between internal hardware diagnostics and additional software diagnostics. The examined MCUs employ a three layered approach:
 1. All hardware blocks required for software execution are equipped with the highest degree of diagnostic coverage by hardware safety mechanisms. Two cores operate in delayed lock step and data transfers between memory and the processing cores are protected by end-to-end error-correcting

code. This shall assure, that the software execution is not impacted by random hardware faults.

2. Based on the integrity of software execution, peripheral functions are mainly assured by software safety mechanism e.g. informational redundancy on application layer protocols.
3. Debug functions should not be used in an operational safety-related system, thus no diagnostics are provided and recommended respectively.

Worst case fault recognition times of hardware diagnostics are stated together with the failure indication and handling by entering safe states of the MCU.

- Hardware and software requirements on system level: Here the assumptions are explained which have to be followed by the system integrator. Hardware requirements define the functionality of external hardware safety mechanism like supervision of the power supply. Software requirements describe the correct way to utilize the internal hardware safety mechanisms and how software can improve the diagnostic coverage depending on the used MCU hardware functions in the safety-related system.

The avionics manufacturer could benefit from the same aspects as the automotive system integrator: At first from the ISO 26262 certified development process of the manufacturer and the process-requirements documented in the ISO 26262 respectively. At second, the additional information from the safety manual may be used. It can be assumed that the AEH supplier may get further support from the MCU manufacturer only in a limited scope, if necessary. However, these are the only public available information that can be additionally reused in particular for ISO 26262 compliant MCUs in the COTS assurance evaluation process described in the next section.

3.2 Evaluation of COTS Component Assurance Benefits by ISO 26262 Compliant MCUs

In Sect. 2.2, COTS assurance activities were outlined on the basis of recommendations from [5] for simple and complex/highly complex MCUs. The presented selection of ISO 26262 compliant MCUs in Sect. 3.1 cannot be classified as simple¹³ and MCUs aiming at an even lower ASIL level like ASIL A or B are often based on more complex architectures. For that reason and to examine all benefits from the ISO 26262 compliance statement for every assurance activity, a classification of highly complex is assumed. The COTS component assurance activities have to be conducted by the AEH supplier and some of them are achievable with minimal or no additional support by the MCU manufacturer. These activities have to be excluded from the evaluation because they can be accomplished with MCUs in general and to claim these as ISO 26262 specific benefits would falsify the assessment results. Even if some activities can be performed without a special support of the MCU manufacturer, establishing a partnership with him

¹³ It is assumed that the full functional scope of the MCU is used and in that case it will be not practical to verify it on that extent on device level.

is meaningful to prevent misunderstandings and facilitate the assurance tasks. Thus the following activities were omitted from the evaluation:

- (1) Describing the COTS component characteristics in order to classify the MCU as simple/complex/highly complex is feasible on basis of the usual public available hardware documentation.
- (2) Archiving of collected device data like errata notes or user manuals.
- (5.2) For usage domain validation the avionics system developer is responsible. Validation in this context means, that a determined usage domain has to be checked such it does not contradict any higher level requirements from system/hardware/software. It is like requirements validation, to check if a low level requirement is a valid refinement of a higher level requirement.
- (8) Documentation of past experience made with the MCU during the AEH development shall substantiate the robustness and maturity in the field.
- (15) Architectural mitigation techniques addressing MCU common modes on system level. These are means external to the MCU on a higher level and have to be implemented during system development. They are also known under the term *safety net*.

Table 1 gives an overview of the evaluation results. The considered assurance activities can make use of additional MCU artefacts in particular. They are assigned according to the identified top level arguments of Sect. 2.2 and arranged in two groups resp.: *Yes* if a COTS component assurance activity benefits from the ISO 26262 compliance statement and *no* if that is not the case.

Table 1. Evaluation results overview

Top level argument	Assurance activity	Benefits by ISO 26262
1. The component performs as described by the manufacturer without anomalous behaviour	(3): Quality management and production	No
	(13), (14): PSE	
	(4), (5.1), (16) ^a : Usage domain	Yes
	(6), (7): Errata management	
2. The component as used satisfies the AEH requirements assigned within the system context	(9), (10): Configuration management	
	(11), (12): Integration	Yes

^aPartitioning considerations were allocated to the usage domain analysis.

For top level argument 1 no benefits can be directly asserted for activities (3), (13) and (14). Quality management and production process requirements in (3) can not be claimed to be defined by the ISO 26262 for MCUs. However, in a comprehensive ISO 26262 assessment process by a third party these aspects should also be checked.

Activities (13) and (14) require the documentation of the PSE and assurance that the PSE is considered similarly in the two domains. The ISO 26262 also introduces a *proven in use argument* to claim a sufficient safety integrity, but no activities are defined that the MCU manufacturer has to document the usage of their products in the automotive field. In [5] it is mentioned that also PSE is creditable from the safety-critical automotive sector if it can be adequately demonstrated. To gain credit for usage experience from applications in this domain is a debatable point, because in contrast to the avionics industry there is no regulated approach that objectively examines the causes of every accident. So the exposure of design errors by automotive field experience is therefore not guaranteed and weakens this argument. In general PSE should be demonstrated as part of the assurance process, but credit should be mainly based on the other activities.

The determination (4) and verification (5.1) of the usage domain profits from detailed data descriptions in the safety manuals including disabling on chip functions, test of activated functions, implementation hints, mandatory requirements, assumptions, and initial configurations. Safety mechanisms described in the safety manual can also be utilized in usage domain verification tasks. Taking into account errata documents during system integration is demanded in the examined safety manual [21–23]. They are published and sufficiently prepared in order to allow the system integrator to determine possible safety implications. Therefore, the errata management activities (6) and (7) should have an advantage by using a ISO 26262 compliant MCU. Assurance activities (9) and (10) request an adequate configuration management or change description approach by the MCU manufacturer and additional change impact analysis by the AEH developer. According to ISO 26262 part 8 a configuration management and change management plan shall be provided by the MCU manufacturer. In the safety manuals or errata documents the applicable device revision or product configurations are clearly stated. It is therefore assumed that COTS manufacturers configuration management is available and in good shape.

Top level argument 2 in Table 1 shows that only two assurance activities can benefit from an ISO 26262 compliant MCU. Actually, most AEH requirements are already determined and verified on device level in activities (4) and (5.1). Usage domain determination is a mapping of AEH requirements on basis of the adequate configuration and usage of the MCU. So the actual function and properties on device level designed by the manufacturer are reused as AEH requirements. In activity (11) AEH requirements from a higher level like LRU or CBA level allocated to the COTS component have to be verified and validated in avionics system context. The device level description in the safety manual for I/O functions and software requirements may support in the validation and verification process for correct integration of the MCU in an AEH. The assurance activity (12) demands a clear understanding of possible device failure modes and rates depending on its configuration. The safety manuals will help in this activity. Several failure scenarios are covered in these documents and failure rate calculations are one of the main topic of ISO 26262 hardware development. Note,

however if a COTS component has previously been approved with specific safety architectures to mitigate propagation of faults from its failures, its reuse in a new system may not have the same benefits of protection.

4 Conclusion

In this article, we presented the differences of assurance approaches for AEH including complex COTS MCUs. We examined which COTS assurance activities in the avionics domain benefit from using an ISO 26262 compliant MCU. To understand the advantages of selecting such a cross-domain COTS component is helpful for an AEH developer. Based on [5] a new structured overview was presented for the COTS hardware component assurance activities. Currently, no industry consensus standard or recommendation from CAs is available that brings all necessary COTS assurance aspects together in an integrated approach [16]. Therefore the presented assurance activities are supposedly not complete. However, the selected assurance tasks provide an adequate foundation for the evaluation of possible benefits of ISO 26262 compliant MCUs during the assurance process. Specifics of ISO 26262 compliant MCUs were described to identify the aspects that can be reused. It was demonstrated where further support by the MCU manufacturer is required even in the case that the MCU was developed according to ISO 26262. The evaluation concentrates on assurance activities where additional support by the MCU manufacturer is most helpful. It could be demonstrated that an ISO 26262 compliant MCU is beneficial in the execution of certain assurance activities. This information can be used by AEH manufacturers to better plan the required COTS assurance activities and to estimate their required effort or possible savings.

However, the reuse analysis result does not allow to make a generic statement that the ISO 26262 approaches are in any case transferable to the avionic domain. An MCU developed according ISO 26262 should be treated like any COTS component in the assurance process. No activity should be skipped even if it was demonstrated in this article that a benefit from the ISO 26262 statement is expected. The AEH manufacturer has to go more into detail by selecting a certain MCU and perform the assurance activities in a specific system context. On that level it is possible to show which aspects can be reused. The manufacturer has to agree with the CA on that approach.

Acknowledgment. This paper is sponsored by the Airbus Defense and Space endowed professorship “System Technology for safety-related Applications” supported by “Stifterverband für die Deutsche Wissenschaft e.V.”. MDHs work in this paper is supported by the Swedish Knowledge Foundation within the project DPAC.

Disclaimer. Although this paper contributes to a reuse argumentation aligned to the regulatory position of CAs, it does not represent them. Only one way to formulate a reuse argument is suggested which has to be finalized in a project context by specific considerations of safety risks and an evaluation of functional or performance requirements in respect to the required integrity level of the avionics systems.

References

1. RTCA: DO-254 Design Assurance Guidance for Airborne Electronic Hardware (2000)
2. FAA: AC 20-152, June 2005
3. CAST: CAST-32A: Multi-core Processors, November 2016
4. CAST: CAST-29: Use of COTS Graphical Processors (CGP) in Airborne Display Systems, February 2007
5. EASA: EASA CM - SWCEH - 001 Development Assurance of Airborne Electronic Hardware, March 2012
6. ISO: ISO 26262 Road vehicles - Functional safety (2011)
7. Schwierz, A., Seifert, G., Hiergeist, S.: Funktionale Sicherheit in Automotive und Avionik: Ein Staffellauf. In: Proceedings of the Automotive - Safety & Security. GI-Edition - Lecture Notes in Informatics, LNI, pp. 13-25 (2017)
8. Schwierz, A., Forsberg, H.: Design assurance evaluation of microcontrollers for safety critical avionics. In: 2017 IEEE/AIAA 36th Digital Avionics Systems Conference, DASC, pp. 1-9. IEEE (2017)
9. Mutuel, L.: Electronic DOT/FAA/TC-17/50: Commercial Off-The-Shelf Airborne Hardware Assurance Methods—Phase 3—Embedded Controllers (2017)
10. DeWalt, M., McCormick, G.F.: Technology independent assurance method. In: 2014 IEEE/AIAA 33rd Digital Avionics Systems Conference, DASC, pp. 8A1-1-8A1-14. IEEE (2014)
11. Jean, X., Mutuel, L., Brindejone, V.: Assurance methods for COTS multi-cores in avionics. In: IEEE (eds.) 35th DASC - Digital Avionics Systems Conference. IEEE (2016)
12. SAE Aerospace: ARP4754A: Guidelines for Development of Civil Aircraft and Systems (2010)
13. ISO: ISO 15026-1: Systems and software engineering - Systems and software assurance - Part 1: Concepts and vocabulary (2013)
14. Holloway, C.M.: Explicate '78: uncovering the implicit assurance case in DO-178C. In: Parsons, M., Anderson, T. (eds.) Engineering Systems for Safety, pp. 205-225. Safety-Critical Systems Club (2015)
15. Mahapatra, R.N., Bhojwani, P., Lee, J.: DOT/FAA/AR-08/14: Microprocessor Evaluations for Safety-Critical, Real-Time Applications: Authority for Expenditure No. 43 Phase 2 Report, June 2008
16. Condra, L., Horan, G., Forsberg, H., et al.: DOT/FAA/TC-16/57: Commercial Off-The-Shelf Airborne Electronic Hardware Issues and Emerging Solutions: Authority for Expenditure No. 75 Report (2017)
17. Faubladiere, F., Rambaud, D.: EASA.2008/1: Safety Implications of the use of system-on-chip (SoC) on commercial-of-the-shelf (COTS) devices in airborne critical applications (2008)
18. Mutuel, L., Jean, X., Brindejone, V., Roger, A., Megel, T., Alepins, E.: DOT/FAA/TC-16/51: Assurance of Multicore Processors in Airborne Systems (2017)
19. Strasburger, J.: FAA Status on Multi-Core Processors (2014)
20. Bieth, P., Brindejone, V.: EASA.2012.C15: COTS-AEH -Use of complex COTS (Commercial-Off-The-Shelf) in airborne electronic hardware - failure mode and mitigation, April 2014
21. NXP: Safety Manual for MPC5744P, June 2014
22. ST: Safety application guide for SPC56ELx family, January 2018
23. TI: Safety Manual for TMS570LC4x Hercules ARM Safety MCUs, September 2016

Autonomous Driving and Safety Analysis



Structuring Validation Targets of a Machine Learning Function Applied to Automated Driving

Lydia Gauerhof^(✉), Peter Munk, and Simon Burton

Corporate Research, Robert Bosch GmbH, Robert-Bosch-Campus 1,
71272 Renningen, Germany

{lydia.gauerhof,peter.munk,simon.burton}@de.bosch.com

Abstract. The validation of highly automated driving vehicles is an important challenge to the automotive industry, since even if the system is free from internal faults, its behaviour might still vary from the original intent. Reasons for these deviations from the intended functionality can be found in the unpredictability of environmental conditions as well as the intrinsic uncertainties of the Machine Learning (ML) functions used to make sense of this complex input space.

In this paper, we propose a safety assurance case for a pedestrian detection function, a safety-relevant baseline functionality for an automated driving system. Our safety assurance case is presented in the graphical structuring notation (GSN) and combines our arguments against the problems of underspecification [9], the semantic gap [3], and the deductive gap [16].

Keywords: Safety · Intended functionality · Functional insufficiency
Nominal performance · Automated driving · Machine learning
Assurance case · GSN

1 Introduction

Highly automated driving vehicles will potentially ease the daily life of millions of commuters, increase the mobility of elderly and disabled people, and enable numerous new business cases. A highly automated driving system can be defined as a vehicle that monitors its driving environment and executes steering, acceleration and deceleration without permanent human monitoring or intervention.

In order to achieve the safety goals of a highly automated driving system, e.g., do not harm pedestrians, the propagation of internal faults must be prevented as is standard in today's automotive systems. However, we must additionally deal with situations where the automated driving system is free from internal faults but behaves in a manner that nevertheless leads to a hazard. For example, a cold and foggy environment can result in blurred camera images and the radar sensors becoming iced such that a safe automated driving operation cannot be ensured

and the probability of the system violating its safety goals is unacceptably high. In contrast to today’s driver assistance systems, an immediate non-technical fallback solution reliant on a human driver is not an option for highly automated driving.

In the following, we refer to such deviations from the intended functionality of a system as *functional insufficiencies*. Note that functional insufficiencies are also known as performance limitations. We also like to point out that if a functional insufficiency occurs, no guarantees about the behaviour of the system can be made. In other words, there is a remaining probability that a functional insufficiency causes the violation of safety goals. Therefore, we regard functional insufficiencies as contributions to system hazards.

In order to prevent functional insufficiencies, the specification of the system must reflect its intended functionality. Furthermore, the system’s specification must be appropriate for any environment within which the system potentially operates. We consider it an insufficient validation strategy to simply drive a specific distance in automated driving mode, since a safely driven route does not necessarily include all environmental conditions and hence does not indicate the absence of failures. Instead, we consider this test driving approach only as supplementary for a structured and validated specification of the intended functionality of the system in all potential environments. In other words, we argue that the absence of unreasonable risk due to hazards caused by functional insufficiencies has to be achieved by a rigorous overall development approach - from the specification of intended functionality, through derivation of subsystem functionality, the implementation and integration of functions and runtime monitoring with the possibility for updates to improve the system based on real-world observations.

In addition to the inherent uncertainty and complexity of the environment, functional insufficiencies can stem from intrinsic uncertainty within the functional implementation itself. Machine learning (ML) is a prominent example for a function with an intrinsic uncertainties, since ML has the ability of learning without being explicitly programmed [15]. A highly automated driving system may contain various ML functions, e.g., for detecting objects from video images.

In order to analyse functions with intrinsic uncertainties, their intended functionality has to be well understood and specified. However, to correctly specify functions with an intrinsic uncertainty, we require either expert knowledge about the conditions under which they usually tend to fail or we need a ground truth reference from which we can determine remaining uncertainty. In the context of ML applied to automated driving (AD), we argue that neither the expert knowledge nor the ground truth reference is perfect.

In order to able to nevertheless build safe systems with ML functions, we are interested in how the potentially unknown functional insufficiencies at the system level can be mitigated despite this intrinsic uncertainty. This includes the question which validation targets have to be achieved to demonstrate that a ML function fulfils its intended functionality. Furthermore, it requires the ability to argue about the validity of these specification and validation targets themselves.

Different challenges in AD development have been already described: under-specification [9], semantic gap [3] and deductive gap [16]. In this paper we review these contributions by means of an ML-specific case study for the safety relevant function “pedestrian detection”. Based on this case study, we propose approaches to answer the following questions: (i) Underspecification: What is the intended functionality and what are its limits? (ii) Semantic gap: How can the intended functionality be described? (iii) Deductive gap: Which requirements on the functional layer (here: ML) can be deduced?

We present a safety assurance case that supports the argument of the absence of unreasonable risk due to hazards caused by functional insufficiencies by structuring the validation targets. Our safety assurance case broadens the approach of Burton et al. [4] and is visualised by a graphical notation, namely the goals structuring notation (GSN).

The remainder of this paper is structured as follows: In Sect. 2, we detail the function pedestrian detection of our case study and derive validation targets for it. In Sect. 3, we present the safety assurance case of the pedestrian detection function. In Sect. 4, we summarize our results, discuss our approach, and present future work.

2 Validation Targets for Pedestrian Detection

In this section, we first introduce the pedestrian detection function and present its functional specification. Then we discuss reducing the risk of hazards caused by underspecification and semantic gap. Finally, we discuss the deductive gap and propose functional modifications to achieve the intended functionality.

2.1 The Pedestrian Detecting Function

A typical automated driving system is comprised of the following parts: sensors, perception, behaviour and trajectory planning, trajectory control, and actuators. The perception part acquires and processes data from sensors, e.g., cameras, lidars, and radar sensors, and other data sources, e.g., car2X-communication, and creates an environmental model of the surroundings of the vehicle. For our case study, we focus on the ML function that detects pedestrians based on video analysis.

This pedestrian detection function is typically realized by a Convolutional Neural Network (CNN), since CNNs are regarded to have a high potential for classification tasks [10]. CNNs are a class of feed-forward neural networks (NN) that consist of a large number of connected neurons - computational units that calculate a weighted sum of their inputs and apply a nonlinear activation function on this sum. The weights are determined by minimizing a loss function of the network over a given set of training data (labelled images) and backpropagating the respective error terms through the network. In this manner, CNNs allow a classification annotated with a confidence level for each class and a localisation of an object within a given image (e.g., frames of a video).

Incorrect functioning of the pedestrian detection function can cause hazards such as “unnecessary emergency braking or steering” and “too late or no emergency braking when necessary”. These hazards potentially violate the safety goal “do not harm pedestrians” of the automated driving system. Thus, we consider the pedestrian detection function as safety relevant. In the following, we consider the general safety goal “ML function meets its intended functionality” as the overall safety goal for the pedestrian detection function.

2.2 Functional Specification

In this case study the pedestrian detection function is divided into two subtasks: 1. classification and 2. localisation of the pedestrian. The specification of each task is derived from the driving task (e.g., ego speed, distance to object) and system boundaries (e.g., braking distance). For example, for the first subtask, the specification is derived from the need to detect persons of a minimum height from a particular distance travelling with a maximum relative velocity which results in a minimum amount of pixels inhabited by the object within a single image frame from the camera.

We propose the following requirements for the first subtask to be defined for each pedestrian class:

- Pedestrian of height ($X1$ pixels) and of width ($X2$ pixels) are classified.
- Pedestrians are detected if $Y\%$ of the person is concealed.
- There are less than $W1$ False Positives per 1000 frames.
- There are less than $W2$ False Negatives per 1000 frames.
- There are less than $B1$ misclassified detections.
- Confidence level shall reflect the actual uncertainty of correctness of a classification.

Both subtasks are required for an adaptation of behaviour and trajectory planning. If the confidence level is not high enough to result in an unambiguous decision, defensive measures are taken (e.g., increased safety distance). This results in the following additional requirements:

- Vertical deviation less than $C1$ pixels to ground truth.
- Horizontal deviation less than $C2$ pixels to ground truth.

The validation data used to confirm these requirements must cover those characteristics of the environment relevant to the task and hence be representative of real-world situations. Note, this does not necessarily mean that the validation data is representative in terms of the frequency of occurrence of certain situations. Critical situations may occur only rarely but must be adequately trained and tested. Furthermore, the validation data must include sufficient variants of pedestrians. The data must also allow for targeted validation of certain attributes, such as non-discrimination between age, gender or race. This requires that these attributes are represented and labelled in the validation data.

In this paper, known challenges of automated driving, in particular when applying ML, are reviewed as sources of functional insufficiencies. These sources

are structured into validation targets, so that a systematic approach is introduced to arguing the goal “ML functions meets its intended functionality”. A cumulative argumentation based on a diverse set of validation targets and statistical evaluation (here formulated as sub-goals of a safety assurance case) including the associated evidence must be discussed. In the following, a thorough investigation is made into how to analyse the intended functionality and how to set the validation targets. This approach is later used to build a safety assurance case.

First of all, the risk due to hazards caused by two sources of functional insufficiencies shall be reduced: underspecification and semantic gap. Later the implementation specific deductive gap will be reviewed. These validation targets must be determined iteratively during development.

2.3 Reduction of Risk Due to Hazards Caused by Underspecification

Underspecification might occur if the intended functionality is more diverse than what is specified [9]. Consequently, defined use cases are only part of the intended functionality. Addressing underspecification by means of a generalization can lead to a inadequately defined safety requirements. In order to reduce risk of underspecification, we suggest the following validation targets. Note that this list might need to be extended and evaluated for each specific task.

- Environment is sufficiently well known.
 - ⇒ Evidence: The hazard analysis and risk assessment (HARA) in the scope of ISO 26262 should be extended to determine properties of the environment that lead to critical situations, e.g., low-angled sunlight, fog and reflective surfaces, etc. Systems safety approaches such STAMP [11, 12]) can be beneficial here. While specifying the intended functionality, unintended use cases must be excluded explicitly in order to highlight the system boundaries. Assumptions on the environment shall be made explicit in order that they can be validated through review, analysis and monitoring.
- Task is sufficiently well known.
 - ⇒ Evidence: Requirements shall be specified including task specific attributes. In the case of generalization abilities, attributes such as colour invariances and translations invariances might be required.
- Sensitivity against unpredictable or unspecified impact of environmental attributes is sufficiently low.
 - ⇒ Evidence: Sensitivity to environmental changes shall be investigated. Moreover, influence due to distributional shift over time or due to geographic changes shall be reviewed. Requirements on invariance and generalization attributes shall be reviewed according their appropriateness to the intended functionality. Run-time monitoring of assumptions and field-based validation shall be used to investigate discrepancies between the real environment and the assumptions as well as sensitivity to these changes. Moreover, statistical extrapolation shall be used generalise the results of acquired data.

2.4 Reduction of Risk Due to Hazards Caused by Semantic Gap

Semantic gap refers to using implicit knowledge on the satisfaction of Safety Goals [3]. In the context of ML, semantic gap refers to making claims on the relevance of references used for training, test and validation data sets. We propose the following sub-goals to support the argument of “Reduction of risk due to semantic gap”. Note that these might have to be extended and evaluated:

- Pedestrian classes are sufficiently accurately described.
 ⇒ Evidence: Functional specification of several validation data sub-sets shall include all variants of classes that can be derived from the environment. Moreover, safety requirements shall be transferred into task specific requirements, e.g. informal textual specifications shall be transferred into formal specifications as far as it is possible, at least for safety-relevant requirements. Evaluation of specific influences and appropriate object variations shall be specified beyond statistical evaluation.
- Location accuracy is sufficiently well described.
 ⇒ Evidence: Training and validation data shall be specified. Evaluation of specific influences shall be specified. Additionally, evaluation of compliance with tolerances, of size and of location variation shall be specified.
- Discrepancy between real and described environment is sufficiently small.
 ⇒ Evidence: Evaluation of similarity between reality and specification of validation data shall be specified. Functional modifications, such as run-time monitoring, degradation modes, pre-processing of ML input etc., shall be specified and documented.

Although systems engineering approaches to ensure a rigorous and complete derivation of the requirements reduce the risk due to hazards caused by underspecification and semantic gap, further evidence must be collected through targeted field-based observations and used to iteratively improve the specifications.

2.5 Reduction of Risk Due to Hazards Caused by Deductive Gap

Deductive gap refers to using invalid assumptions on different abstraction levels causing an unintended functionality [16]. In the context of ML, features might be wrongly learnt or erroneously implemented. The deductive gap for ML differs from the deductive gap for non ML functions due to its intrinsic uncertainty. Note that reduction of underspecification and semantic gap is a prerequisite for the implementation of the intended function and as important as the avoidance of deductive gap.

The following validation targets can be defined for reducing the risk due to hazards caused by deductive gap before and during training:

- Data set is sufficient for the intended functionality.
 ⇒ Evidence: Transfer of system-level requirements to ML-specific requirements as well as the attribute distribution within training, test and validation data sets shall be evaluated. Moreover, independence from unintended object

relations shall be highlighted. For example, synthesised data can be used to broaden recorded data by special attributes.

- Overfitting is sufficiently reduced.
 - ⇒ Evidence: Overfitting measures, such as pretraining on diverse data set, regularisation methods, Dropout or DropConnect, data augmentation shall be documented and evaluated.
- Underfitting is sufficiently reduced.
 - ⇒ Evidence: Underfitting measures (e.g., a minimum amount of training data for each class variant) shall be documented and evaluated.

Self-learning algorithms are difficult to understand, since parameters are set not by an engineer, but by the learning method, e.g., back propagation. Moreover, the learnt features do not have to represent physical properties, but may occur arbitrarily. within the data To achieve the following validation targets, it is essential not only to consider ML in a black box manner (as it is handled during statistical analysis) but also to understand the essential influences for deviation from the intended functionality. In order to evaluate weaknesses of the ML function, the following methods are currently known:

Feature Visualization. One possibility of uncovering features which have activated a class is to visualise the part of the image that contributed to the classification result. Saliency methods belong to such techniques. Zeiler and Fergus [17] present a visualisation technique mapping various layers of a CNN to an image. The pixels within the image can be highlighted according to the scale of influence on each layer. Visualizing features might help to understand which patterns of the training set activates the feature map.

Structuring of the Input Space. By annotating known attributes of the images that are independent from the classification task (e.g., weather conditions, light conditions, contrast), the input space can be further structured for training and validation purposes. Either these additional attributes are chosen by developers or appropriate clusters are identified by algorithms. Equally, sub-classes of task classifications and their properties can be defined manually or automated and offers opportunities to further optimize the classification process [13]. Then, task-specific misclassification and mislocalisation are investigated (e.g., correlations are visible in confusion matrices [1]). If correlations exist, either training data can be broadened appropriately or the confidence level can be adapted appropriately during post-processing.

Formal Verification. Under certain conditions, some functions allow the application of formal verification to investigate whether certain constraints are met across the complete input space.

One approach based on formal verification that is applied on a neural network is provided by [7]. The investigated neural network makes flight control decisions. Katz et al. investigate a fully connected neural network. After rewriting inputs and outputs as Boolean formula, a linear real arithmetic, the authors proof with the help of an Satisfiability Modulo Theories (SMT) solver that the formula is satisfiable in the sense of SMT.

Nevertheless, in our case study, a formal verification of CNNs is not realistically feasible, since it is difficult to describe the input and output space (images with all kinds of variations in appearance etc.) or to formulate linear real arithmetic for this purpose.

Uncertainty. The confidence level of each class output does not express a probability of existence of the object itself. Therefore, uncertainty calculation might be used to measure the reliability of the classification result. Uncertainty quantification can be used for further measures (e.g., in plausibility checks and sensor fusion algorithms [8]), thus improving the overall robustness and reliability of the subsequent trajectory planning tasks [14].

With the help of these methods, the following validation targets must be proven in order to reduce the risk of hazards caused by deductive gap. We suggest the following incomplete list of evidences:

- Essential influences on the ML function are sufficiently understood.
⇒ Evidence: The application of feature visualization, adaptation of confidence level and uncertainty calculation shall be documented. Furthermore, correlations of errors to features shall be investigated and reduced by appropriate training. Evaluation of these correlations shall be documented.
- ML function is sufficiently robust.
⇒ Evidence: Tolerance against distributional shift, adversarial and faulty input shall be evaluated. Statistical evaluation shall be documented. An integrity test of ML function shall be documented.
- Learnt features are sufficient for function.
⇒ Evidence: Learnt features and correlations between these and detection results shall be analysed and documented (e.g., by feature visualization).

A well-known weakness of ML functions is the sensitivity to adversarial attacks. In this case an object, e.g., a road sign [5,6], is slightly modified such that a human would not recognize a manipulation but it is misclassified with a high confidence by a machine learning function. ML functions trained with different data subsets or with adversarial examples are more robust against adversarial attacks but unfortunately, to the best of our knowledge, there is no general solution to avoid adversarial attacks as of today. Hence, special caution is necessary while system engineering and when creating validation targets for adversarial attacks.

Furthermore, the following validations targets regarding any changes made during the development of the system (e.g., in its parameters or in the computing platform) must be proven.

- Changes to parameters do not inviolate safety requirements.
⇒ Evidence: Verification specification for any changes shall be documented.
- Differences between the training and target platforms do not lead to a violation of the safety requirements.
⇒ Evidence: Verification specification for any changes shall be documented.
- Changes in target platform comply with safety requirements.
⇒ Evidence: Verification specification for any changes in target platform shall be documented.

2.6 Functional Modifications to Achieve the Intended Functionality

The validation targets for the deductive gap defined in the previous section cannot guarantee that hazards at a system level will not occur. In the following, we provide a brief overview of potential safety measures to reduce the risk induced by functional insufficiencies at a functional and system level.

Measures at the Functional Level. Besides demonstrating the performance of the ML functions themselves the following measures can be introduced to reduce the risk associated with functional insufficiencies:

- Pre-processing of ML-input might be conducted according to known factors that significantly influence performance. If the performance of the ML function, for example, is decreased for pictures with very low contrast, a classification might be suspended if such conditions are detected.
- Post-processing of the ML-output might include adjustment of confidence levels based on factors known to influence performance, so that decisions about driving behaviour and trajectory planning are adapted to the reliability of the perception function. Influences might include object size (due to resolution problems), ego travel velocity (due to blurring effect) or image quality e.g., contrast and light conditions.

Measures at the System Level. To reduce the propagation of errors throughout the system and therefore to reduce the risk of hazards induced by functional insufficiencies at the system level should be identified by applying rigorous systems engineering approaches. This can include the introduction of the following measures:

- Diverse redundancy increases the dependability of a function. For pedestrian detection, several possibilities exist, e.g., Lidar, Radar and traditional computer vision algorithms.
- Operating modes, also called degradation modes, depend on the vehicle's environmental model. As long as the environmental model is reliable, decisions are taken within a wider range of possible trajectories. In contrast a degradation mode is chosen according to a cautious and defensive driving strategy, if an object is detected with a low confidence level.
- Transition between operating modes ensures a continuous driving behaviour.
- Run-time monitoring of assumptions allows the validation of whether assumed attributes about environment are still valid. The detection of discrepancies between distribution of environmental attributes and design assumptions at run-time could indicate either errors in the trained function or that the system is operating within an environment for which it was not adequately trained.
- Established driver assistance systems (e.g., Emergency brake assist) applied to AD must be reviewed from a system engineering perspective. It must be clarified to what extent measures must be taken at the system level to reduce the integrity requirements on the individual functional components.

3 Safety Assurance Case for Pedestrian Detection

The previous section introduced validation targets that must be addressed during development. Nevertheless, the argumentation that a ML function meets its intended functionality has to be summarised and structured. In this section we propose a safety assurance case using the goal structuring notation (GSN) that structures the presented validation targets and associated evidence. The aim of our safety assurance case is to argue the safety of ML with respect to the absence of unreasonable risk due to hazards caused by functional insufficiencies.

The safety assurance case includes all validation targets as sub-goals and evidence of mitigating against weaknesses in the ML function at the system as well as functional level. An evaluation of the validation targets is not conducted in this paper, since it is not representative for a general validation approach due to environmental-, task-, and system-specific dependencies. Instead we propose and discuss an approach to structure validation targets for a safety relevant ML function applied to automated driving and to combine diverse sources of evidence.

Figure 1 graphically represents our approach to arguing that the ML function meets its intended functionality. In order to support the goal “Machine Learning function meets all of its functional requirements”, the strategy “Argument over sufficient reduction of root causes of functional insufficiencies” is given. This argument is associated with the three sub-goals to reduce risk due to underspecification, semantic gap and deductive gap. Their sub-goals in turn are not depicted, but stated in the following.

In the right upper context symbol all information about the ML function, its tasks, requirements and the CNN are stated (usually as a link to an appropriate document). The use of a CNN for the task is argued by the statement that CNNs are currently the most successful classifier (right lower context symbol) and this statement is justified by the contests that are won by CNNs [10]. Therefore, this justification also depends on the assumption that classification performance from benchmark contests are transferable and therefore highlight the potential of CNNs for classification in general.

The goal “ML function meets its intended functionality” is stated within the context “Pedestrian detection is a safety-relevant function, so that this ML function is also safety-relevant.” which might be linked to a hazard and risk analysis. Furthermore, the assumptions on the environment are also stated (usually linked to an appropriate document).

The strategy to achieve the main goal “ML function meets its intended functionality” is argued by a sufficient reduction of root causes of functional insufficiencies. This argument is reflected in the three sub-goals reducing risk due to underspecification, semantic gap and deductive gap with the arguments over appropriate specification, description, deduction. Note that the justification that these three sub-goals are sufficient is missing here, but shall be stated in general.

In Fig. 2, the GSN is refined for the two arguments over appropriate specification and description. The sub-goal “Reduction of risk due to semantic gap” is achievable for the given specification that reflects the sub-goal “Reduction of

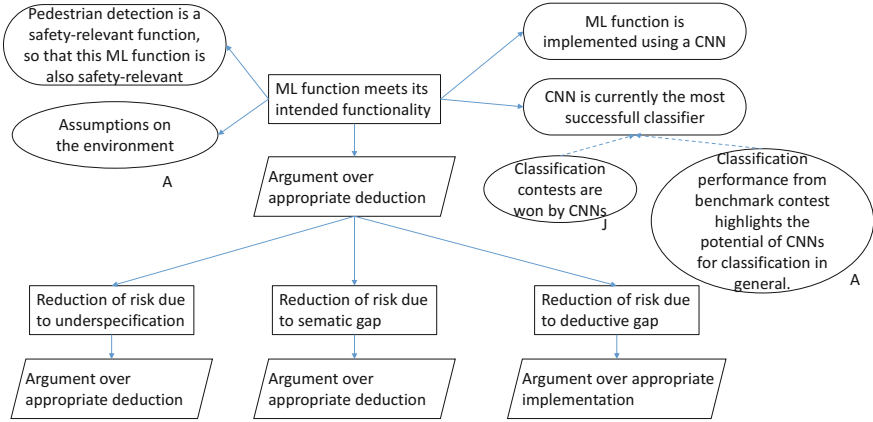


Fig. 1. GSN for the goal machine learning function meets its intended functionality.

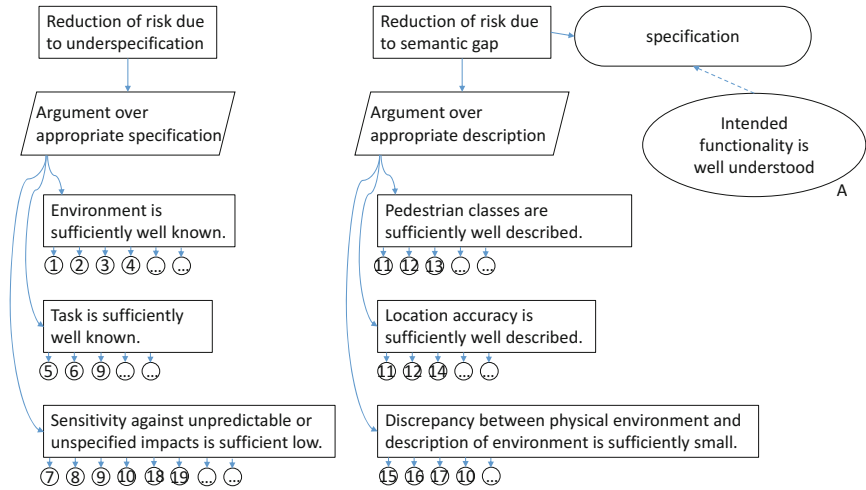


Fig. 2. GSN for the Goal: “Reduction of risk due to underspecification and semantic gap”. The following evidence is suggested to support the claims: (1) hazard and risk analysis, (2) accident database, (3) explicit exclusion of unintended use cases, (4) explicit assumptions on environment, (5) requirements specification (colour invariances, translations invariances), (6) documentation of functional modifications, (7) sensitivity investigation, (8) review of distributional shift, (9) specification of invariance and generalization attributes, (10) run-time monitoring, (11) specification of training and validation data, (12) evaluation of specific influences, (13) evaluation of appropriate object variations, (14) evaluation of compliance with tolerances, of size and location variation, (15) evaluation of similarity between reality and specification of validation data, (16) degradation modes, (17) pre-processing of ML input, (18) field-based validation, (19) statistical extrapolation

risk due to underspecification”. Moreover, all validation targets reducing the risk associated with underspecification and semantic gap lead to a refined specification that is task-specific, but not implementation-specific.

Figure 3 depicts the sub-goal “Reduction of risk due to deductive gap” focusing on the implemented function. Its refined specification is stated in the context element. The specification holds, since it is assumed that intended functionality is well understood and described. A justification is missing here, but shall be stated in general.

Especially the first validation target “Data set is sufficient for intended function” of the argument over appropriate implementation points out that the data set (that is the basis for training, test and validation data set) shall reflect the goals “Reduction of risk due to underspecification” and “Reduction of risk due to semantic gap”. Here the dependencies between all three goals become clear. If underspecification or a semantic gap increases the risk of unintended functionality, then this will also affect the data set. For example, if a firefighter was not explicitly included in the specification, the data set might also lack this kind of variant of a person. Consequently, a ML function is not trained properly and might not be able to recognize the firefighter.

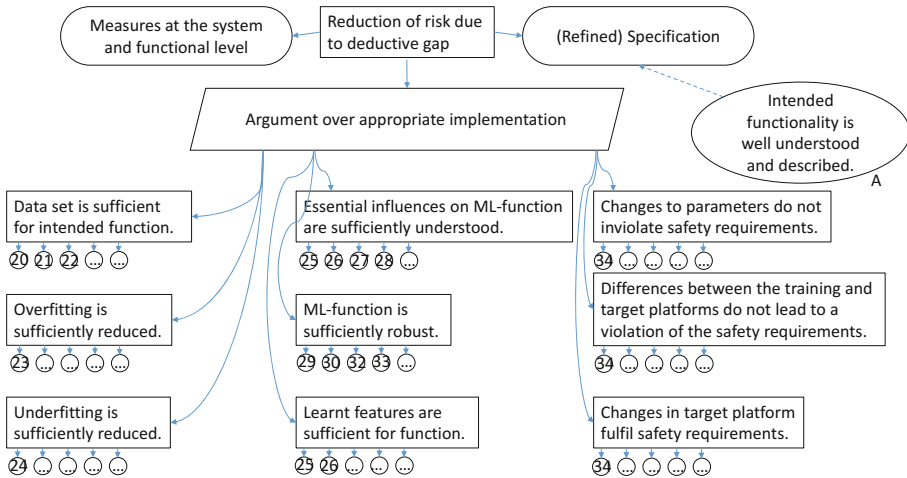


Fig. 3. GSN for the Goal Reduction of risk due to deductive gap. The following evidence is suggested to support the claims: (20) evaluation of transfer of requirements to ML-specific requirements, (21) evaluation of attribute distribution within training, test and validation data sets, (22) independence from unintended object relations, (23) overfitting measures (pretraining on diverse data set, regularisation methods, Dropout or DropConnect, data augmentation), (24) underfitting measures, (25) feature visualization, (26) correlations to features, (27) adaptation of confidence level, (28) uncertainty calculation, (29) evaluation of tolerance against distributional shift, (30) adversarial attacks, (31) integrity test of ML function, (32) statistical evaluation, (33) evaluation of faulty inputs, (34) verification specification for any changes

Additionally, the refined specification must also approach the integrity requirements on the individual functions. The awareness of attributes that might be adverse to the intended functionality under particular conditions, might be used to introduce a data fusion with further information processing methods and/or redundant information sources (e.g., sensors, digital maps, server). Therefore, the deductive gap might be mitigated by other measures than within the function itself. These measures are stated in the context element on the left hand side (usually linked to an appropriate document).

4 Conclusion and Future Work

We investigated sources of functional insufficiencies and derived validation targets in order to demonstrate the intended functionality of a machine learning (ML) function in an automated driving system, namely the pedestrian detection function. Our approach is based on the reduction of the well-known root causes of functional insufficiencies: underspecification [9], the semantic gap [3], and the deductive gap [16]. From the validation targets we outlined a safety assurance case in GSN for the safety goal “Machine learning function meets its intended functionality” of the pedestrian detection function.

When we derived and structured the validation targets for our case study we included well-known methods and measures. While statistical evaluation offers a good basis to investigate improvements at a functional and system level, other methods, such as feature visualization, are used for analysis and for reaching a better overall understanding of the learnt functionality. However, the effectiveness of these methods and measures must still be evaluated in detail. Therefore, further research on the contribution of each method and measure to the validation targets of automated driving is necessary. We argue that only with an industry consensus on an established set of methods, can a convincing and accepted argument for the safety of automated driving be made. This includes an agreement on a sufficient “weight of evidence” and abort criteria for each validation activity, which has not yet been reached [2]. As part of future work we aim to systematically evaluate the effectiveness and contribution of methods and measures discussed in this paper to derive and structure validation targets and evidence for series development projects. This includes the question how the effectiveness of the individual methods and measures can be measured and demonstrated based on quantifiable key performance indicators.

References

1. Alsallakhl, B., Jourabloo, A., Ye, M., Liu, X., Ren, L.: Do convolutional neural networks learn class hierarchy? *IEEE Trans. Vis. Comput. Graph.* **24**(1), 152–162 (2018). <https://doi.org/10.1109/TVCG.2017.2744683>
2. Amarnath, R., Munk, P., Thaden, E., Nordmann, A., Burton, S.: Dependability challenges in the model-driven engineering of automotive systems. In: 2016 IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW, pp. 1–4, October 2016

3. Bergenheim, C., et al.: How to reach complete safety requirement refinement for autonomous vehicles. Technical report, CARS 2015 - Critical Automotive Applications: Robustness & Safety, Paris, France, September 2015
4. Burton, S., Gauerhof, L., Heinzemann, C.: Making the case for safety of machine learning in highly automated driving. In: Tonetta, S., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2017. LNCS, vol. 10489, pp. 5–16. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66284-8_1
5. Evtimov, I., et al.: Robust physical-world attacks on deep learning models. *Cryptography and Security* (2017). <https://arxiv.org/abs/1707.08945>
6. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2015). <https://arxiv.org/abs/1412.6572>
7. Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M.: Reluplex: an efficient SMT solver for verifying deep neural networks. Technical report. Stanford University, USA (2017). <https://arxiv.org/pdf/1702.01135.pdf>
8. Kendall, A., Gal, Y.: What uncertainties do we need in Bayesian deep learning for computer vision? (2017). <http://arxiv.org/abs/1703.04977>
9. Koopman, P., Wagner, M.: Challenges in autonomous vehicle testing and validation. *SAE Int. J. Trans. Saf.* **4**, 15–24 (2016). <https://doi.org/10.4271/2016-01-0128>
10. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems*, pp. 1097–1105 (2012)
11. Leveson, N.: <http://psas.scripts.mit.edu/home/2016-stamp-workshop/>
12. Leveson, N.G.: *Engineering a Safer World: Systems Thinking Applied to Safety*. The MIT Press, Cambridge (2011)
13. Nguyen, A.M., Yosinski, J., Clune, J.: Multifaceted feature visualization: uncovering the different types of features learned by each neuron in deep neural networks. *CoRR* abs/1602.03616 (2016). <http://arxiv.org/abs/1602.03616>
14. Tas, Ö.Ş., Kuhnt, F., Zillner, J.M., Stiller, C.: Functional system architectures towards fully automated driving. In: *2016 IEEE Intelligent Vehicles Symposium, IV* (2016). <http://ieeexplore.ieee.org/document/7535402/>
15. Samuel, A.L.: Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.* **3**(3), 210–229 (1959). <http://ieeexplore.ieee.org/document/5392560/>
16. Wilhelm, U., Ebel, S., Weitzel, A.: Functional safety of driver assistance systems and ISO 26262. In: Winner, H., Hakuli, S., Lotz, F., Singer, C. (eds.) *Handbook of Driver Assistance Systems*, pp. 109–131. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-12352-3_6
17. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) *ECCV 2014*. LNCS, vol. 8689, pp. 818–833. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10590-1_53



Multi-aspect Safety Engineering for Highly Automated Driving Looking Beyond Functional Safety and Established Standards and Methodologies

Patrik Feth¹(✉), Rasmus Adler¹, Takeshi Fukuda², Tasuku Ishigooka²,
Satoshi Otsuka², Daniel Schneider¹, Denis Uecker¹, and Kentaro Yoshimura²

¹ Fraunhofer Institute for Experimental Software Engineering,
Kaiserslautern, Germany
patrik.feth@iese.fraunhofer.de

² Hitachi, Ltd., Tokyo, Japan

Abstract. Highly automated and autonomous driving is a major trend and vast amounts of effort and resources are presently being invested in the development of corresponding solutions. However, safety assurance is a concern, as established safety engineering standards and methodologies are not sufficient in this context. In this paper, we elaborate the fundamental safety engineering steps that are necessary to create safe vehicles of higher automation levels. Furthermore, we map these steps to the guidance presently available in existing (e.g., ISO26262) and upcoming (e.g., ISO PAS 21448) standards and point out open gaps. We then outline an approach for overcoming the identified deficiencies by integrating three different safety engineering disciplines. This includes (1) creating a safe nominal behavior specification; (2) dealing with functional insufficiencies, and (3) assuring the related performance wrt. functional safety. We exemplify our proposed methodology with a case study from industry.

1 Introduction

In many embedded systems domains we presently see a trend towards higher levels of automation up to the point of autonomy. Highly automated and autonomous driving, for instance, is a major trend, and vast amounts of effort and resources are presently being invested in the development of corresponding solutions. Significant progress has already been made and results have been very promising; for instance, numerous demonstrator vehicles have impressively shown the technical feasibility of advanced automated driving features up to the point of fully autonomous driving. However, before such features can become actual products, it is absolutely mandatory to ensure that they do not introduce unacceptable levels of risk. This is the domain of safety engineering, where we presently face major challenges due to the insufficiency of established methods and standards, which are mostly designed with respect to aspects of functional safety, omitting other aspects that are now gradually moving into the limelight.

Considering only functional safety is not enough because systems are no longer fully controlled by human operators; rather, they increasingly incorporate their own extended perception, reasoning, and decision capabilities. In the automotive domain, this trend manifests in the introduction of vehicles with increasing levels of automation. According to the SAE classification [9], the currently available Advanced Driver Assistance Systems (ADAS) can be classified as level 2 or partially automated systems. However, Audi is the first manufacturer [2] claiming to be technically ready for automation level 3 as soon as the corresponding regulations are available. The transition from automation level 2 to automation level 3 is a remarkable change, especially from a safety point of view. This is due to the fact that starting from level 3, the system is actually responsible for rendering safe nominal behavior. In contrast to that, for lower automation levels, it is still the driver who is responsible for guaranteeing safe system behavior and the safety scope is consequently limited to functional safety, i.e., to hazards caused by (random and systematic) faults. This change in responsibility changes the way we need to perform the overall engineering and especially the safety engineering of those systems.

Thus, for lower automation levels, the driver is responsible for choosing an appropriate vehicle behavior in a given driving situation and the vehicle is responsible for supporting the driver in terms of situation awareness and correct implementation of the driving decisions. Any electric or (programmable) electronic system (E/E system) that affects controllability by the driver is consequently considered safety-critical. This obviously includes also ADAS, as the goal of any ADAS is to assist the driver and contribute to his ability to control the vehicle. The topic of safety assurance of such E/E systems is well studied and corresponding guidance is provided by the existing safety standard ISO 26262 and the upcoming safety standard ISO PAS 21448 “Road vehicles - Safety of the intended functionality” (SOTIF). ISO 26262 focuses on functional safety, which means managing risks emerging from malfunctioning behavior (due to random hardware failures or systematic failures) of E/E systems. It does not, however, cover safety issues emerging from functional insufficiencies. This means that it assumes that the performance limits of all functions are specified in a reasonable and safe manner, so that it is sufficient to focus on critical deviations of the specified functionality. Particularly for ADAS, with their complex situation awareness, this assumption becomes very difficult to handle and hence leads to a gap in the established field of safety assurance. The upcoming SOTIF standard is meant to close this gap. For example, a camera without a night filter can only work during daytime or a LIDAR sensor might not work in heavy snowfall or even in rain. Thus, any creation of situation awareness based on these sensors will fail in these respective critical situations. SOTIF addresses this problem by supporting the systematic selection of an appropriate sensor concept.

However, the basis for conceiving an adequate sensor concept and related performance limits is knowing which situations need to be detected with which level of confidence. Furthermore, it is necessary to define appropriate responses if the current situation cannot be determined/classified at all or not with sufficient

confidence. In the case of automation levels 1 and 2 and partly also in the case of level 3, the response can be a shutdown of the functionality and transition to manual driving. This strategy is not applicable at automation level 4, as any fallback to manual driving is excluded by definition.

However, even if we assume that the correct detection of situations is not a problem, we still have to deal with a huge number of driving situations if we want to define which vehicle behavior is appropriate in which situation. We use the term **safe nominal behavior specification** to refer to such a specification that describes which driving behavior is safe in which situation and that abstracts from all technological challenges of situation awareness. Existing standards, including the upcoming SOTIF, provide no guidance for engineering such a safe specification even though the name SOTIF seems to indicate at least some support in this regard.

We think that being aware of these different dimensions within the overall notion of safety is very important to foster a structured discussion and to organize further research in these important fields. Furthermore, it helps to avoid over- or misinterpretations of existing safety standards in the context of safety engineering for higher automation levels.

The remaining article is structured as follows: Sect. 2 highlights the contribution of this paper and places it in the context of related work. Section 3 gives a brief overview of the solution and points out some of its risks and limitations. Section 4 presents the proposed solution and a holistic approach that achieves the above-mentioned goals. Section 5 relates this process to SOTIF and other safety standards. Section 6 exemplifies the proposed solution before Sect. 7 concludes the paper.

2 Related Work

To the best of the authors knowledge, the state of the art is still lacking wrt. precise identification and characterization of the gaps in current safety methods and standardization regarding highly automated and autonomous driving (and systems in general). Existing work in this field rather provides experience reports on the usage of ISO 26262 for vehicles of higher automation levels, for example in [10] Spanfelner et al. identify insufficient models as the major problem of using ISO 26262 for driver assistance systems. We agree with this line of argumentation and add a contribution to their work, as we additionally consider the ISO PAS 21448 “Safety of the Intended Functionality” standard and propose a holistic process that goes beyond ISO 26262 instead of enforcing the use of existing standards (which clearly have not been designed for highly automated or autonomous systems). Higher-level thoughts on the topic of safety for vehicles of higher automation levels can be found, for example, in [4]. In this work, Koopman and Wagner also mention the shortcomings of ISO 26262, but do not provide clear concepts or methods on how to overcome these problems.

In contrast to this, one major aspect of our work in this paper is the identification of the need to specify safe system behavior, i.e., to create a safe nominal

behavior specification. Although we propose the usage of state machines to this end, we do not intend to argue that this is the best way to do it. We argue that the non-existence of dedicated methods and standards for this aspect indicates that there is currently no commonly accepted best practice, and it will be the task of future experience to find such a practice. In earlier work, Leveson also used state machines to describe the higher-level behavior of a safety-critical system [7]. In more recent work, Leveson uses control structure diagrams in the Systems-Theoretic Accident Model and Processes (STAMP) [5] and the related Systems-Theoretic Process Analysis (STPA) [6] approach for hazard identification. These approaches build on a systems engineering foundation for analysis and, just like ISO 26262 and the SOTIF standard, focus on deviations from this specification of the intended behavior, which is assumed to be safe. Some earlier work developed at Fraunhofer IESE is systematizing and automating hazard and risk analysis [3]. This might also help in analyzing highly automated or autonomous systems, where degrees of freedom and uncertainties lead to a very high complexity, which is generally hard to tackle without systematic, tool-supported and ideally (semi-) automated approaches.

3 Safety Aspects Relevant to Autonomous Systems

As illustrated in Fig. 1, we argue that guidance (by means of methods, techniques and maybe explicit standardization) is required for the creation of a safe behavior. The aspect of creating a safe nominal behavior specification has been out of scope for existing safety standards and ongoing activities in standard creation initiatives (e.g., SOTIF), but is becoming very important for highly automated and autonomous systems. The safe nominal behavior specification defines which behavior is safe in which situation and is therefore the basis for reasoning about functional performance limits and which limits are sufficient and which are insufficient.

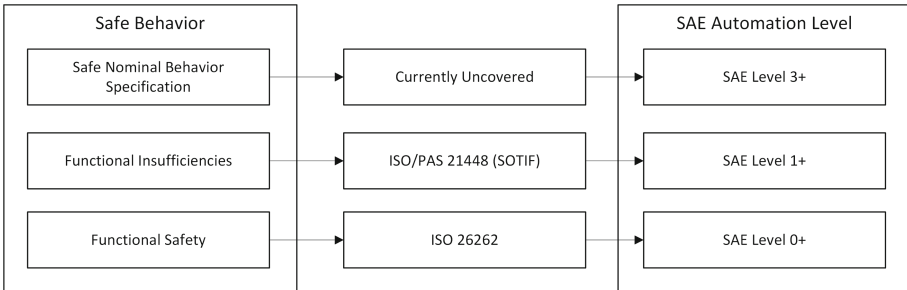


Fig. 1. Elements to reach safe behavior: standard coverage

The second layer provides guidance for dealing with functional insufficiencies, which is likewise the focus of the upcoming SOTIF standard. The performance

limits in the SOTIF wording are of a more technical nature than the functional performance limits that we consider as part of the safe nominal behavior specification. Both functional and technical performance limits are a prerequisite to reasoning about functional safety and considering violations with respect to these performance limits.

The third important safety aspect for autonomous systems is functional safety, which is the predominant aspect considered in the engineering of today's (more or less) operator controlled systems. Consequently, there is already a broad established and proven range of techniques, methods, standards and tools available which can be used (or be at least a starting point) also for autonomous systems. Of course, all three relevant safety aspects should be integrated and harmonized on a conceptual and methodological level and corresponding standardization shall be tightly interlinked as it is, for instance, already the case for the ISO 26262 and the ISO PAS 21448.

The three safety aspects elaborated above can be further illustrated in mapping them to the standard architecture of embedded systems (Monitor - Plan - Actuate). Doing this leads to the following observation: The SOTIF safety aspect (and thus the ISO PAS 21448) focuses on the monitoring part and provides guidance for creating a safe situation awareness. The functional safety aspect (and thus the functional safety standard ISO 26262) focuses on random and systematic software and hardware failures that might impact any element of the cycle, i.e. monitoring, planning, and actuation. A significant gap exists regarding the assurance of safety with respect to automated or autonomous planning. The planning determines which vehicle maneuvers and trajectories are safe in which (perceived) situations. It is thus closely related to what we call the aspect of safe nominal behavior specification in this article. The planning needs to implement the safe specification as well as possible, particularly considering the uncertainties that are dynamically induced by the monitoring element due to inherent challenges in assessing the current context situation.

In summary, we argue that the engineering of a safe highly automated or autonomous system requires the consideration of functional safety, functional insufficiencies, and a safe nominal behavior specification. Today, only the first aspect of functional safety is well understood, supported by established methods, techniques and tools and addressed by an existing and established standard: the ISO 26262. So it is known only for this aspect what is considered necessary to claim sufficient coverage, i.e. to be able to argue a sufficient level of functional safety to release a (non-automated) car as a product. For the topic of functional insufficiencies, a draft version of an upcoming standard, ISO PAS 21448 "Road vehicles - safety of the intended functionality", was used as input for deriving the recommendations in this article. The requirements concerning this topic might change in the future; best practices are not known yet and need to be established over time. The current draft version of the SOTIF standard is explicitly only intended to cover vehicles up to automation level 2. On top of the aspects considered by SOTIF and ISO 26262, the aspect of how to create a safe nominal behavior specification needs to be addressed for higher automation levels.

At the time of this writing, this aspect is not being considered yet at all by existing standards or by standard creation initiatives. One reason for that might be, that in non-automated system a safe nominal behavior is typically pretty straight forward and commonly agreed upon. In case of a car, it is clear how the user interface looks like and how a driver operates it. And it is also clear, that monitoring and planning are tasks of the driver and the driver is thus responsible for the driving behavior, leaving only functional safety as important safety aspect. Now, given the trend towards ever higher levels of automation across domains, this area requires more consideration in the future to allow the development and also validation (i.e. creation of sufficient evidence wrt. safety) of safe autonomous vehicles.

4 Multi-aspect Safety Engineering for Autonomous Systems

The proposed approach still generally aligns with the established principles of how safety engineering works and interacts with “normal” system engineering. Safety engineering builds upon the initial results from system engineering and analyzes them with respect to safety. Based on the analysis results, safety requirements are elicited, a safety concept is compiled, corresponding safety measures are selected, implemented and validated and a related safety argumentation is created. This procedure (or at least parts thereof) occurs at different development stages (or abstraction levels) in parallel (and interaction) with the system engineering activities. However, compared to the established approach focused on functional safety, the boundaries between safety engineering and “normal” system engineering, i.e. the engineering of the nominal system behavior, are softened. In particular with respect to the engineering of a safe nominal behavior (of the automated behavior) we obviously have a tighter integration between the disciplines.

As illustrated in Fig. 2, we consider three (horizontal) abstraction layers that directly relate to the three safety aspects identified in the previous section.

The System in Its Usage Context layer is related to the aspect of defining a safe nominal behavior specification. It represents an abstraction layer on which high-level concepts of the system and the requirements on the system are described independent of their technical realization.

The System Realization Concept layer is related to the aspect of handling functional insufficiencies. It contains first technical information on the future system, such as sensor concept and algorithms.

Finally, the System Functional View layer is related to the aspect that addresses functional safety. It represents the more detailed functional view on the system and describes how the requirements are functionally realized by the system. It is the basis for conducting safety analyses and deriving a functional safety concept in terms of ISO 26262.

The first fundamental research question concerns the notations and modeling languages used for representing the system at the different abstraction layers.

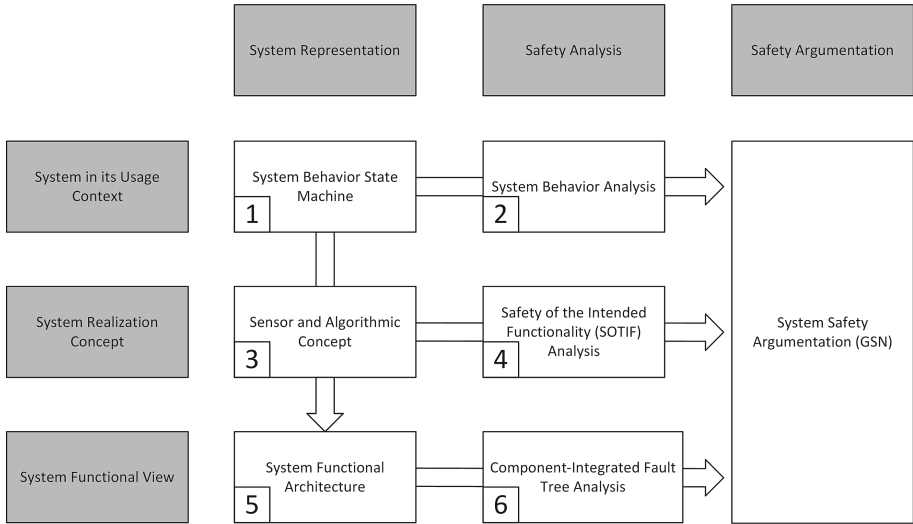


Fig. 2. Multi-aspect safety engineering process

At the highest level of abstraction, we aim at a definition of the vehicle motion. In a variety of industry projects, we have observed that this is often specified via images showing trajectories of the ego-vehicle in a certain driving situation. The problem with this approach is the assurance of completeness. All possible sequences of valid trajectories need to be specified. To solve this completeness problem, we propose state machines, as state machines are a common notation for specifying complete sets of sequences. Furthermore, this specification technique supports modeling the activation, deactivation, and degradation of the automation functions. At the more detailed abstraction level, we recommend using the familiar notation of functional architectures. However, creating these system representations is not within the actual scope of safety engineering and should be performed, or at least strongly assisted, by domain and system engineering experts.

The safety analysis is conducted on the basis of the system representation. At the requirements level, this is done by analyzing the behavior of the system and deriving possible hazards of that behavior independent of the technical realization. The next layer, the System Realization Concept layer, already contains initial information on the technical realization and considers this information in the safety analysis. The SOTIF analysis additionally and explicitly takes into account functional insufficiencies of the technology used. More details on how to perform the safety analysis in these two top layers will be given with the example in Sect. 6. At the functional level, we propose using component-integrated fault trees (CFT) to refine high-level safety goals into more detailed functional safety requirements. CFTs have proven their benefit in multiple industry projects and are an accepted approach for systematic and model-driven safety analysis [1].

The Safety Argumentation layer is the bonding element among the three abstraction layers. The safety argumentation can also be considered as a safety documentation as it stores the refinement of top-level safety goals into safety requirements and provides a structured argument regarding how the fulfillment of low-level requirements implies fulfillment of top-level goals. As a notation, we recommend the Goal-Structuring Notation (GSN) for this activity [8].

The order for the outlined activities in the previous section depends on their dependencies. Activities at higher levels of abstraction depend on activities at lower levels of abstraction and vice versa. Furthermore, safety analyses depend on the engineering of the system representation being analyzed, and the selection of safety measures along with the related safety argumentation depends on the safety analyses. Any process that takes these dependencies into account is valid. We assume the following waterfall-like process:

1. Model high-level system concept
 - Output: State machine model of the behavior of the system independent of any implementation details
2. Consider the high-level system behavior and possible hazards of this behavior
 - Output: Safe system specification including safety goals derived by a systematic state space analysis
3. Provide information on the sensor and algorithmic concept
 - Output: Sensor and algorithmic concept as a basis for SOTIF analysis
4. Conduct analysis of system limitations and functional insufficiencies
 - Output: Safe system specification extended with consideration of system limitations and functional insufficiencies and their derived safety goals
5. Model functional architecture consistent with safe system specification and realization concepts
 - Output: System functional view as a basis for ISO 26262 analysis
6. Perform ISO 26262 analysis to investigate the contribution of component failures to the violation of safety goals
 - Output: Functional safety requirements assigned to components in the architecture.

Steps 1 and 2 take place at the highest abstraction levels and are independent of any implementation details. Steps 3 and 4 take place at the SOTIF level and consider realization concepts. Steps 5 and 6 take place at the ISO 26262 functional safety level.

5 Relation to SOTIF and Other Standards

In the following, we relate the six steps of the solution proposed in the previous section to the current world of safety standardization. To this end, we look at the scope definition in the current draft of the SOTIF standard. This scope definition considers different causal factors of hazards, providing the relation to SOTIF and other standards for each factor. The causal factors are as follows:

1. E/E system failures
2. Unintended behavior without fault or failure (including E/E system performance limit)
3. Foreseeable user misuses
4. Security violation
5. Impact from active infrastructure and/or vehicle to vehicle communication
6. Impact from car surroundings
7. Unsafe nominal behavior specification.

We enhanced this overview with respect to the safe nominal behavior specification and structured it using the Goal Structuring Notation (GSN). Due to space limitations, we cannot show the full GSN here. From the top-level goal “Perform such safety engineering activities that guarantee the absence of unreasonable risk for the automation level 4 driving system”, we derived “the safe specification” by creating subgoals related to the aforementioned causal factors in the SOTIF scope and the scope of this work. The defined safety goals were then defined as “Perform such safety engineering activities that guarantee the absence of unreasonable risk for the automation level 4 driving system caused by [element from the enumeration before]”.

The second level of refinement argues over existing safety standards for the particular source of unreasonable risk. For security violations (safety goal 5), the “Cybersecurity Guidebook for Cyber-Physical Vehicle Systems” (J3061) has been an initial step (which is presently being integrated in other activities), and for vehicle-to-infrastructure and vehicle-to-vehicle communication (safety goal 6), the “Road Vehicle - Extended Vehicle” ISO 20077 standard is about to be published. Performing safety engineering activities according to the recommendations in these standards leads to an absence of unreasonable risk caused by the particular aspect addressed by this standard. In the project we conducted, it was assumed that the vehicle is not connected to its environment. Because of that, both causes were out of scope in the conducted project and there was no further refinement of the goals to perform safety engineering activities according to the safety standard.

The cause of E/E system failures (safety goal 1) is addressed by the ISO 26262 standard. The SOTIF standard claims to address the causes of unintended behavior without fault or failure, foreseeable user misuse, and impact from car surroundings (safety goals 2–4).

The final level of refinement contains the steps of the proposed solution in Sect. 4. The performance of activities belonging to the concept phase from ISO 26262 and the SOTIF standard is given as the goals at this refinement. The activities of the standard are mapped to the activities in the solution. Note that not every step in the suggested process can be mapped to an existing standard. This is due to the fact that no standards are currently available for the development of systems with higher automation levels. Even the SOTIF standard is only suited up to automation level 2 to date. Considering this background, the suggested process might need to be revised once appropriate standards have been established.

6 Methodology Example

This section exemplifies the solution presented in Sect. 4 for a highway assistant system. The system under consideration is classified as SAE automation level 4 and shall operate without interruption and without relying on a human driver as fallback performance of the dynamic driving task but with the limited system capability to operate on highways only. This system will be used as an ongoing example in this section.

6.1 Model High-Level System Concept

The first step in the proposed process is the modeling of the high-level system concept. The goal of this step is to capture the system concept at a high abstraction level. This includes the concept for activation, deactivation, degradation (e.g., from level 4 to level 3), and for handling emergency situations. In addition, it includes the general vehicle behavior in these different cases. As mentioned above, we propose using state charts to represent these high-level concepts. In the conducted project, we created such a state chart for an automation level 4 highway assistant system.

6.2 Consider the High-Level System Behavior and Possible Hazards of This Behavior

After the system's behavior has been captured at a high abstraction level, it can be analyzed regarding its safety in different driving situations. Whether a behavior is safe or appropriate depends on the current situation. For example, the operating mode "passing" is a type of behavior that is not safe or unsafe independent of the situation. It is a safe and appropriate behavior if there is a slower vehicle in front and the left lane is free; but it is an unsafe behavior if a vehicle is currently approaching on the left lane. This analysis of the safety of behavior in different situations has to be conducted for systems of higher automation levels. For each operating mode, i.e., for each state in the above state chart, one needs to argue on the preconditions that must be fulfilled to enter a mode and the circumstances under which a mode needs to be deactivated. **A mode shall only be activated if the risk of this mode in the current driving situation is acceptable; if the risk of the mode becomes unacceptable, the mode needs to be deactivated.** Deactivation obviously requires us to define a mode that is less risky. If there is no other alternative how the vehicle can drive "safer" in the considered situation and if this "safest" solution is not acceptable, then we have to think about external measures that can serve to avoid the occurrence of the situation (e.g., external infrastructure, new driving laws, etc.). As the introduction of external measures goes beyond the scope of this report, we focus on patterns for annotating the operating modes with safety conditions and assumptions.

This general line of thought directly gives us a pattern for deriving a safe nominal behavior specification. To make this step systematic, we recommend performing a systematic analysis of the state space that the system can encounter. A possible way to do this is the usage of tables describing environmental factors and possible values for these factors. For each value or combination of these values, a classification is performed as to whether it is acceptable to allow the operating mode or whether the situation requires deactivation of this mode.

For the operating mode “Passing” of the AL 4 driving system, an analysis of the behavior in different situations has been conducted. An excerpt of this analysis shows that it may yield the following safety goals for passing:

- Passing must not be performed at an intersection (merge) area of a highway
- Passing must not be performed if there is a vehicle on the adjacent left lane.

6.3 Provide Information on the Sensor and Algorithmic Concept

Up to this point, the process steps have abstracted from the implementation concepts. To conduct the SOTIF analysis, these concepts need to be added to the information available about the developed system. In particular, the standard focuses on sensors and algorithms for the creation of situation awareness. Concepts about this part of the system are necessary to perform an analysis on the limitations of situation awareness. In its current version, the SOTIF standard mainly focuses on functional insufficiencies: situations in which sensors and algorithms are operating outside their intended state space. It needs to be specified how the sensors are used to create the needed situation awareness. Which situations the system needs to be aware of from a safety perspective can be derived from the analysis conducted in the step before. From the safety goals derived above for the operating mode “Passing”, the following requirements on situation awareness can be derived:

- Detect intersection (merge) area of a highway
- Detect vehicle on the adjacent left lane.

The resulting sensor concept for the automation level 4 system might state that the intersection (merge) area of a highway shall be detected with GNSS and 3D maps and the presence of vehicles on the adjacent left lane shall be detected with radar, lidar and camera.

6.4 Analysis of Limitations and Functional Insufficiencies

Based on the sensing concept, the performance limits are derived. This shall be done for each sensor used. Reaching the performance limits of a sensor can again trigger a transition in the system’s functional concept. An example is the usage of lidar in situations where there is heavy snowfall. Under such conditions, a lidar sensor usually does not work anymore. If the lidar sensor is the only way to determine the distance to objects in front of the vehicle, then this automation

level 4 functionality cannot be provided without this sensor. Thus, the environmental situation of heavy snowfall demands the deactivation of the AL 4 driving mode. This step refines the step of creating a safe system specification by adding implementation-specific information to the system specification.

Above, we derived a sensor concept from the safety goals related to the operating mode of “Passing”. As part of the system limitations and functional insufficiencies analysis, we will detail this sensor concept. Let us assume that the sensors that are used come with the following limitations:

- Camera: Limited performance during nighttime
- Lidar: Limited performance in heavy rain and snow
- Radar: Limited performance in heavy snow
- 3D Maps: Information is usually delayed by at least 10 min
- GNSS: Limited performance inside tunnels.

Table 1 gives resulting limitations from the sensor concept.

Table 1. Sensor concept limitations table

Situation	Sensor concept	Resulting limitation
Intersection (merge) area of a highway	GNSS + 3D maps	Not possible to detect if currently at an intersection (merge) if currently driving in a tunnel due to missing GNSS reception
Vehicle on the adjacent left lane	Radar + lidar + camera	Not possible to detect vehicle on the adjacent left lane during nighttime with heavy snow due to sensor limitations

From the resulting limitations, we can derive the following functional improvements:

- Passing must not be performed while driving in a tunnel (not able to detect if currently at intersection (merge) area of a highway)
- Passing must not be performed during nighttime with heavy snow (not able to detect vehicle on the adjacent left lane or tail vehicle at traffic jam or obstacles on the road).

These safety goals become part of the safe nominal behavior specification.

6.5 Model Functional Architecture Consistent with Safe Nominal Behavior Specification and Realization Concepts

After the system behavior has been defined in a safe nominal behavior specification containing both implementation-independent information from steps 1 and 2 of the suggested solution in Sect. 4 and implementation-specific information from steps 3 and 4, this specification shall be translated into a functional

architecture as a basis for ISO 26262 analysis. Again, we do not see this step as a genuine safety engineering step but as a step to be conducted as part of the engineering process. The functional architecture shall use hierarchy and make intensive use of ports. In industry, components are often modeled only with one input- and one output-port. This is not enough to support component-integrated fault tree analysis. The information that is exchanged between the functions in the functional architecture needs to be defined in more detail. For every information with a unique character, a special port has to be created.

6.6 Perform ISO 26262 Analysis to Investigate the Contribution of Component Failures to the Violation of Safety Goals

In order to achieve the requirements of functional safety, which is of course still important for systems with a high automation level, the ISO 26262 standard is the corresponding reference in the automotive domain. This step is already standardized and mature methodologies exist to support it. We argue that the problems encountered when applying the standard to higher automation levels, which are mentioned in other publications, originate mainly from the imprecise definition of the intended function. If the steps recommended in this work are followed and a functional architecture is created that realizes a safe nominal behavior specification, ISO 26262 can be applied.

7 Conclusion

In the automotive domain, as well as in other domains of embedded systems, we see a significant trend towards ever higher levels of automation up to the point of autonomy. The economical and societal potential is huge, but several challenges need to be tackled before such systems can actually become products and a business success. One important challenge is ensuring safety, whereas established methods and standards have been designed with manually controlled systems in mind and need to be augmented to actually cover all relevant aspects for highly automated and autonomous systems.

Accordingly, in practice, safety engineering is currently mainly concerned with ensuring functional safety and the corresponding fulfillment of normative requirements from standards and regulations. Regarding systems with high automation levels, this limitation of safety engineering is not appropriate anymore. In this paper, we propose a multi-aspect safety engineering approach for highly automated driving which incorporates additional relevant safety aspects beyond functional safety and thus beyond established methods and standardization. Most importantly, we introduce the additional aspect of engineering a safe nominal behavior specification with the help of state machines and a systematic state space analysis. This puts an additional layer on top of the safety aspects tackled by ISO PAS 21448 and ISO 26262, i.e. safety of the intended functionality (actually focused on functional insufficiencies and assuming the availability of a specification of safe nominal behavior as a starting point) and functional

safety. The overall approach has been illustrated based on an industrial case study of an advanced driver assistance system. I.e. we briefly described how the safe nominal behavior specification was created, how it has been used as a starting point for the analysis of causes and consequences of deviations from the intended functionality as per the SOTIF standard and, finally, how functional safety can be tackled.

We see the core contribution of this paper in the discussion of the necessary safety considerations for highly automated systems and the explicit identification of the three required safety aspects. In doing so, we point out the current gaps in the established safety engineering state of the practice and standardization. The proposed solution is in its details still relatively premature and has only been applied in few occasions. However, the experiences made have been promising and we think that the described approach can contribute as a basis for discussion and be a starting point for further work to facilitate systematic engineering of safe highly automated and autonomous systems.

References

1. Adler, R., Schneider, S., Hoefig, K.: Evolution of fault trees from hardware safety analysis to integrated analysis of software-intensive control systems. In: International Conference on Engineering Sciences and Technologies (2004)
2. Audi (2017). <https://www.audi-mediacycenter.com/en/press-releases/the-new-audi-a8-future-of-the-luxury-class-9124>
3. Kemmann, S.: SAHARA: a structured approach for hazard analysis and risk assessments. Dissertation. TU Kaiserslautern, Kaiserslautern (2015)
4. Koopman, P., Wagner, M.: Autonomous vehicle safety: an interdisciplinary challenge. *IEEE Intell. Transp. Syst. Mag.* **9**(1), 90–96 (2017)
5. Leveson, N.: A new accident model for engineering safer systems. *Saf. Sci.* **42**(4), 237–270 (2004)
6. Leveson, N.G.: An STPA primer. <http://sunnyday.mit.edu/STPA-Primer-v0.pdf>
7. Leveson, N.G., Heimdahl, M.P., Hildreth, H., Reese, J.D.: Requirements specification for process-control systems. *IEEE Trans. Softw. Eng.* **20**, 684–707 (1994)
8. Limited, O.C.Y.: GSN community standard version 1 (2011)
9. SAE: J3016: Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles (2016)
10. Spanfelner, B., Richter, D., Ebel, S., Wilhelm, U., Branz, W., Patz, C.: Challenges in applying the ISO 26262 for driver assistance systems. *Schwerpunkt Vernetzung, 5. Tagung Fahrerassistenz* (2012)



A Model-Based Safety Analysis of Dependencies Across Abstraction Layers

Christoph Dropmann¹(✉), Eike Thaden², Mario Trapp¹,
Denis Uecker¹, Rakshith Amarnath², Leandro Avila da Silva¹,
Peter Munk², Markus Schweizer², Matthias Jung¹,
and Rasmus Adler¹

¹ Fraunhofer IESE, 67663 Kaiserslautern, Germany
{christoph.dropmann, mario.trapp, denis.uecker,
leandro.avila, matthias.jung,
rasmus.adler}@iese.fraunhofer.de

² Robert Bosch GmbH, 71272 Renningen, Germany
{eike.thaden, rakshith.amarnath, peter.munk,
markus.schweizer}@bosch.com

Abstract. Identifying and mitigating possible failure propagation from one safety-critical application to another through common infrastructural components is a challenging task. Examples of such dependencies across software-stack layers (e.g., between application and middleware layer) are common causes and failure propagation scenarios in which a failure of one software component propagates to another software component through shared services and/or common computational resources. To account for this, safety standards demand freedom from interference in order to control failure propagation between mixed-critical software components. Safety analysis is typically focused on one abstraction layer, while robustness tests try to find failure propagation paths across abstraction layers. To this end, this paper presents a model-based failure propagation analysis combining failure propagation within and across abstraction layers. A classification of dependencies in combination with fault trees is used to perform a model-based dependency analysis. In addition, a novel modeling technique for integrating failure propagation aspects resulting from shared services and resources is presented. The analysis was used to carry out an early safety assessment of a real-world automotive redundancy mechanism within an integrated architecture. The results show that the method improved reusability and modularity, and made it easier to estimate failure propagation issues, including possible violations of freedom from interference within an integrated system.

Keywords: Software and system safety · Interferences · Safety analysis

1 Introduction

Today's cars are equipped with up to 100 Electronic Control Units (ECUs). The continuous growth of electronics in safety-critical systems of automotive vehicles, however, means that the trend is now towards architectures with only 8–12 very

powerful ECUs or Domain Controllers instead of today's 80–100 federated ECUs [1]. Such an integrated architecture allows multiple software applications to share services provided by these centralized platforms. Consequently, such an integrated architecture increases deployment flexibility and reduces weight, size, power consumption, and costs.

However, a major disadvantage of such an integrated system is its lack of “inherent fault containment barriers” between different vehicle functionalities, which is a side effect of using federated ECU architectures with several ECUs for several functionalities [2]. Within an integrated system with only a few ECUs, each hosting many applications, a failure can propagate from one software component to another through a shared service or resource. Examples of such failure propagation paths, which are also called interference paths, are potential CPU contentions and memory corruptions.

Safety assessment assigns different safety integrity levels through a safety-critical system's functionality. In order to avoid safety-critical interferences across integrity boundaries, existing standards such as ISO 26262 [3], IEC 61508 [4], or DO-178C [5] demand ‘freedom from interference’ or ‘robust partitioning’, respectively. To this end, modern real-time operating systems such as PikeOS [6], QNX [7], or AUTOSAR OS [8] provide protection features ranging from execution time monitoring and memory protection to hypervisors. As these concepts are domain-independent, we only discuss approaches from automotive and the approach presented in this paper is applied to an example from the automotive domain.

A safety engineer has to show that a safety-critical integrated system is free from interference. Particularly, the decomposition of a component's safety requirement into multiple safety requirements for redundant components with lower safety integrity levels (following the idea of ASIL decomposition of ISO 26262) requires independence of the components implementing these requirements, and thus freedom from interference even within the same ASIL level. Simulation and fault injection experiments, e.g., with FAIL* [9], generate robustness evidences to support an independence claim. Lists of known issues such as those described in [3, 10, 11] and systematic interference analysis for new types of shared resources as proposed by us in [12] provide guidance for identifying interferences and arguing freedom from interference.

Neither current analysis nor robustness tests can answer the question of whether or not an integrated system setup provides sufficient protection against interference. This is because the application and configuration setup of a system is beyond the scope of current analysis, and testing can only show the absence of the tested failure. In order to enable getting more complete results, we must consider testing the dependencies across abstraction layers (e.g., between application and middleware layer) in a multidirectional analysis together with the dependencies within an abstraction layer (e.g., sensor-actuator layer) in addition. This will tell us whether an interference is safety-critical and has to be mitigated.

To this end, this paper presents a failure propagation analysis that combines our analysis from [12] with our multidirectional contract-based system description approach from [13]. We applied the analysis to an early safety assessment of a real-world automotive redundancy mechanism within an integrated multi-layer architecture. The results show that our method improved the reusability, modularity, and completeness of the dependency analysis. In addition, it enables safety engineers to identify

dependent failures before the system is implemented or complex simulation models are built. Thus, the presented method is a solution for a dependent failure analysis in the context of the trend towards more powerful ECUs. The contribution of this paper is a model-based analysis of dependencies across abstraction layers. In detail, the contribution is three-fold: (1) We present a dependency classification that guides safety analysts in identifying concrete dependent failures. The classification can be used separately from the presented method. (2) The proposed component-based approach modularizes the composition of components across abstraction layers, and thus enables reuse of analysis artifacts within an integrated system with shared services. (3) We propose a novel modeling technique for platform services that are part of the platform infrastructure and support the development of application software, e.g., an actuator driver or a communication service. The modeling technique aims to simplify failure propagation analyses compared to a traditional fault tree for, e.g., a communication service and is complete with respect to the defined classification.

This paper is structured as follows: In Sect. 2, we provide a basic overview of related embedded software modeling and corresponding safety analysis methods, as well as related approaches. Section 3 introduces the dependency classification, which is the basis for the actual analysis method presented in Sect. 4. Before concluding the paper in Sect. 6, we perform and illustrate an example analysis in Sect. 5.

2 Related Work

In this section, we will provide a short introduction to model-based embedded software engineering along with corresponding component-oriented safety analysis methods. In the field of software engineering, several model-based description languages have evolved over time. A well-known language is the *Unified Modeling Language* (UML). In embedded software engineering, the *Systems Modeling Language* (SysML) provides language elements for considering the technical system as a whole [14]. UML and SysML do not cover safety as a quality attribute. The *Vertical Safety Interface Language* (VerSaI) [15] and *Conditional Safety Certificates* (ConSerts) [16] are contract-based and component-oriented languages used for annotation by safety aspects. They define a system as a composition of components that exchange information, energy, and/or mass flow at their interfaces. The contracts allow the formulation of black-box specifications of the corresponding components. VerSaI provides language elements for describing the dependencies of an application's safety demands and the safety guarantees of the platform on which it is designated to run. ConSerts defines a system as a hierarchical composition of application components. The contracts are post-certification artifacts equipped with variation points bound to formalized external dependencies that are meant to be resolved at runtime. The *Architecture Analysis & Design Language* (AADL) [17] from the avionics domain and EAST-ADL [18] from the automotive domain cover the embedded domain from real-time software to hardware. Both AADL and EAST-ADL focus on system design and real-time aspects. Error annexes available as a kind of extension partly cover safety aspects. Current research such as [19] focuses on optimizing the mapping from software to hardware resources in

the context of a safety-critical system. However, the aspect of safety analysis within an integrated system is yet to be covered systematically.

The error annex of EAST AADL provides a logic for describing a known case of failure propagation using state machines. *Hierarchically Performed Hazard Origin and Propagation Studies* (HiP-HOPS) [20] and *Component-Integrated Component Fault Trees* C²FTs [21] combine component-based design with fault tree failure propagation analysis. The authors of [22] extend the approach of component-based analysis with formal contract-based design. Failure propagation via infrastructural services is the focus of [23]. The authors present the worst-case assumption of combining all the basic events from a shared device into one software component. The focus of the approach is on common causes. The effect of temporal interferences known at an early stage of design time can be analyzed with the approach presented in [24]. The authors model the system across several abstraction layers, from the functionality layer to the physical layer. The paper [25] presents a dependency check based on ASIL consistency over all modeled components. The authors of [26] describe a formal specification of an interference and derived rules for achieving freedom from interference. Systematic guidance and an analysis covering failure propagation within and across abstraction layers, which would support safety engineers and enable comprehensive dependency analyses, is still missing. As a contribution to this end, a dependency classification and a novel analysis approach will be proposed in the following section.

3 Dependency Classification

The related work shows a trend towards model-based safety analysis to achieve alignment between nominal behavior and safety aspects, and towards support for the usage of commercial-off-the-shelf components or reuse of components. However, current analyses focus on failure propagation within one abstraction layer of a model. For instance, a component-based fault tree analysis focuses on a sensor, a controller, and an actuator component, or on the underlying realization components (DMA, CPU, memory). It is important to note that failure propagation caused by interference is an unavoidable challenge in an integrated system. The chaining within an interference is a failure propagation between shared service users via the components that realize the service. In addition, the realization components can trigger further interferences that affect further services. Consequently, dependencies within and across abstraction layers are important for an integrated system.

As a contribution towards mastering this challenge, we will first provide a description of the types of dependencies between multiple users of a shared service and the dependencies that a model-based safety analysis can address. Then we will present a classification for dependencies within an abstraction layer. The classification is an extension of the one we presented in [12] and provides systematic guidance for manual interference analysis of shared services. In [12], we focused on hardware resources such as DMA. With the classification of this paper, we expand the focus to cover dependencies across abstraction layers in general, i.e., between an application and the application infrastructure software.

The classification focuses on multi-user failure propagation to support model-based analysis at design time. Propagation along other dependencies, caused, e.g., by design processes or tools, are out of scope and covered by diversity measures. For the scope of failure propagation along modeled dependencies, the well-known keywords from Hazard and Operability Studies [27] generally provide high-level guidance. These keywords are omission failure, commission failure, time failure, and value failure. As a result, a modeler will define failure type demands along usage relations within an abstraction layer, e.g., “A 250 ms delay of signal X must be avoided with ASIL B for a specific safety goal”. ASIL B is the automotive safety integrity level B. An ASIL defines acceptable failure rates as well as techniques and safety mechanisms that engineers should apply to achieve a given safety goal. In short, the ASIL represents the tolerable likelihood that the announced failure type (250 ms delay of signal X) will contribute to the related safety goal violation. In contrast to propagation along usage relations, propagation along hidden dependencies is caused by the services that realize a user’s functionality; for example, a shared communication service that realizes the signal flow between components. The concept of hidden dependencies is described in [28], where the authors label them as pseudo functionality. We distinguish between two different kinds of hidden dependencies: (1) failure propagation via dependencies that are not modeled. Robustness evidences, generated, e.g., with fault injection or simulation, support independence guarantees that focus on such dependencies that are not modeled; (2) failure propagation via invisible dependencies across abstraction layers. The dependencies are invisible because in model-based safety analysis, e.g., in C²FTs, there is no view that indicates these dependencies. The dependencies are defined within mappings between hardware and software components, deployments, sometimes called bindings.

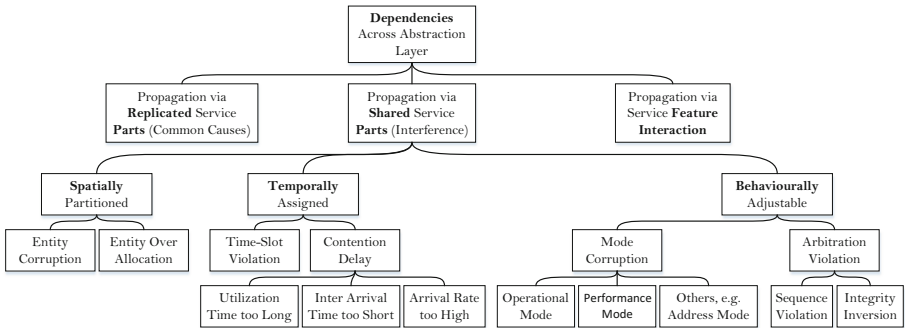


Fig. 1. Dependency classes across abstraction layers

The classification presented in Fig. 1 guides safety engineers through a logical interference analysis to identify invisible dependencies across abstraction layers. We distinguish between three main classes: failure propagation via replicated services parts, respectively via copied model components: failure propagation via shared service parts, respectively interferences; and failure propagation via service feature interaction.

Replicated service parts invalidate the independence of the replicated failure causes (called basic event within a component fault tree). If one cause occurs, the replicated ones may occur in addition. In other words, failure propagation via replicated service parts refers to common causes. In the case of an integrated system, this is essential if, for instance, the modeler copies the component that represents a task including the failure model (e.g., C²FT). The classification for shared service parts (Fig. 1) is based on our classification presented in [12].

The proposed services, i.e. spatially partitioned, temporally assigned, and behaviorally adjustable, deal with the question of whether, when, and how a service part could be accessed by a given service user. This tripartite interference classification divides the temporal interferences of the traditional interference classification from [10] into temporal and behavioral aspects in order to support more fine-grained guidance for safety engineers ([10] subsumes within the temporal aspect all temporal effects and not only the accessibility of a service). The subclasses of the spatial, temporal, and behavioral aspects are depicted in Fig. 1. The feature interaction class is the last aspect that a safety engineer should investigate for a multi-user service, especially in the case of users with different integrity levels. This class covers the propagation along the usage relations between services and within a service itself. The presented classification is the basis for the model-based dependency analysis presented in the next section.

4 Dependency Analysis

This section presents a system model covering dependencies within and across abstraction layers that forms the basis for the analysis. In addition, this section introduces a novel modeling technique for platform services that supports application development. Based on the modeling technique, we will explain our analysis procedure using an example in the next section. The concept of dependencies within and across abstraction layers is inspired by the interface type definition of [15], where the authors distinguish between the vertical interface between the application and the underlying platform, and the horizontal interface between applications. In contrast to [15], our aim is a failure propagation analysis instead of a modular requirements description for semi-automated integration (Fig. 2).

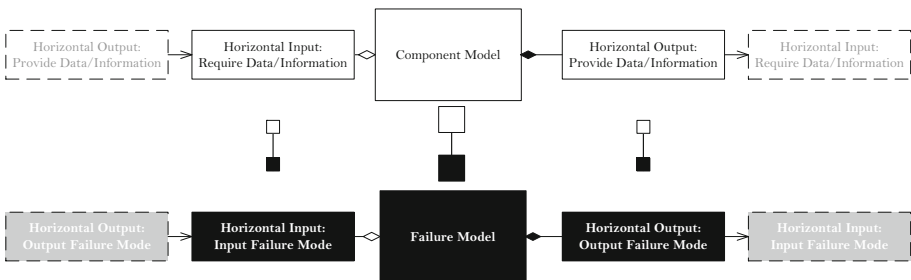


Fig. 2. Horizontal dependency model

4.1 Dependencies Within an Abstraction Layer

As stated in the related work section, there are established methods, such as C^2 FTs and HiP-HOPS, for specifying a modular failure logic within an abstraction layer. Therefore, our solution uses the established method of C^2 FTs. Based on a component-integrated component fault tree (C^2 FT), failure rates and cut sets (combinations of component failures that can cause a system failure) can be calculated. The modules, e.g., sensor, controller, and actuator, can be structured hierarchically, e.g., a controller can contain several sub-controllers. However, a shared service or resource, such as a computation unit that is used by two components, e.g., by a sensor component to scale its signal and by a PID controller to calculate a set value, is out of scope. Yet this is a relevant scenario for an integrated architecture. Therefore, we propose developing the relations between the components within an abstraction layer along its data and information dependencies. In addition, we propose modeling the realization dependencies that come with the challenge of shared services within another abstraction layer. Finally, we propose connecting the abstraction layers. The next paragraph discusses this connection.

4.2 Dependencies Across Abstraction Layers

The dependencies across abstraction layers are a generalization of the dependencies within an abstraction layer. We define dependency across abstraction layers along required and provided services. Such a service can be information, similar to a dependency within an abstraction layer, e.g., if an application is required to ‘send a signal’, but it can also be an operating system service, like a demand for memory or execution time.

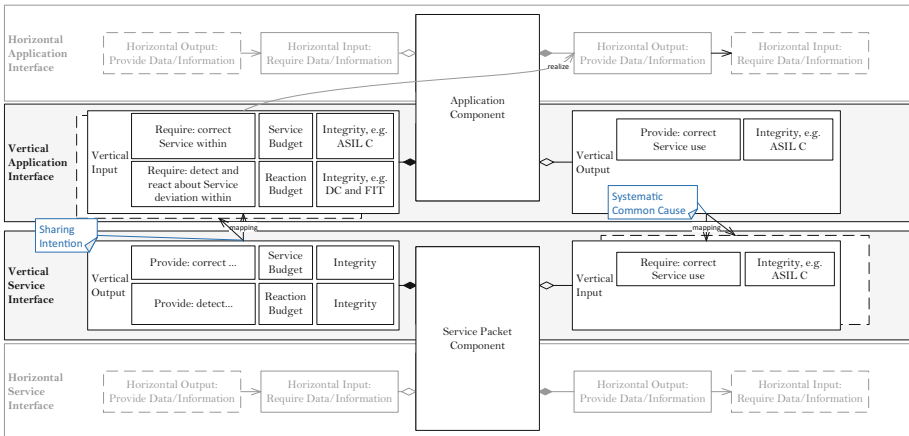


Fig. 3. Holistic dependency model with focus on the vertical interface

The authors of [29] distinguish between application services and basic services. An application service can be experienced by the user, whereas a basic service realizes the

application service. Please note that we consider humans, software services, and hardware services as users. Based on this distinction, we call the upper-layer component *Application Component* and the lower one *Service Packet Component*.

The vertical interface has a direct influence on the performance of the application since it realizes the application. Consequently, as shown in Fig. 3, a vertical interface consists of a *Required* and *Provided Service* in combination with a *Service Budget*. The *Integrity*, e.g., ASIL C, of the *Vertical Application Interface* is inherited from the *Horizontal Output* that the *Vertical Application Interface* is realizing. Please note that we focus on dependency across abstraction layers, which is why the *Horizontal Output* is not depicted in detail (it should also contain an integrity level and a link to the safety goal that it satisfies). A *Vertical Output* is an option for specifying a requirement from a *Service Packet Component* to an *Application Component*. Conservative embedded systems do not need this option. Consequently, vertical description languages, such as VerSaI, do not contain any *Vertical Output*. However, a modern system based, e.g., on POSIX, will need this option; for example, dynamic memory management with the `malloc()`-service requires correct use of the `free()`-service, or a waiting service requires correct passive waiting. Additionally, in order to require a service, the vertical interface allows specification of detection and reaction mechanisms for handling and detecting failures. This option comes with *Reaction Budget* and *Integrity*, e.g., diagnostic coverage (DC) and *Failure in Time Rate* (FIT). Mapping links the *Vertical Application Interface* with the *Vertical Service Interface*. Please note that a *Vertical Output* mapped to multiple inputs indicates a sharing intention (if one *Service Packet Component Output* is used by multiple *Application Components*) or a systematic common cause (if an *Application Component* does not provide correct behavior to multiple *Service Packet Components*).

4.3 Dependencies Within the Platform Service Domain

In this paragraph, we will describe the elements of our novel modeling approach for platform services. When we applied C²FTs to a POSIX-based system including the platform services, we concluded that the C²FTs approach is not directly applicable. The reason is that the complexity (the number of dependencies and failure mode transformations within and between the services) results in a large and confusing set of fault trees. Consequently, reviewing and reusing are hard to achieve. Therefore, we identified the design components of platform services that are needed to provide interference analysis and automated fault tree generation. Please note that our intention is not to come up with an additional modeling approach for service design. Instead, we propose modeling with the minimum of information needed for a safety engineer to perform a dependency analysis and create a fault tree.

Typical platform service packets are actuator drivers, operating system services, memory management services, timing and interrupt services, and communication services [15, 28]. Each service packet consists of various services. [30] defines a service as the behavior of the provider as its users perceive it. The behavior is described by internal and external states. In our case, the service packet is the provider and the perceived behavior is the service action. An example of such a service action (service for short) is a thread wake-up service (the implementation could be, for example, sending a

corresponding POSIX signal) from a thread-handling packet (e.g., a library or a module). An internal or external state manipulation of a service is, for example, the manipulation of a running queue (from where a scheduler fetches the next thread to schedule) or the data structure with the process context information. Figure 4 presents our modeling elements for platform services. The two central elements are the *Service* (as a service above) and the *Entity* (which can be used to model the objects for the state information, e.g., a data object). Both elements, *Service* and *Entity*, as well as the *service packet component* can contain themselves in a hierarchy. For reasons of legibility, we skipped the composite pattern in Fig. 4. The *Entity* can be a *User Entity* if it represents an object for an external state, or a *Management Entity* if it represents an object for an internal state. The relations between *Vertical Service Interface*, *Services*, and *Entities* allow transformation into a fault tree and interference traces. We created the failure modes of the *Vertical Output* manually. However, [31] has shown that failure types can be generated automatically via a predefined failure type taxonomy for a domain.

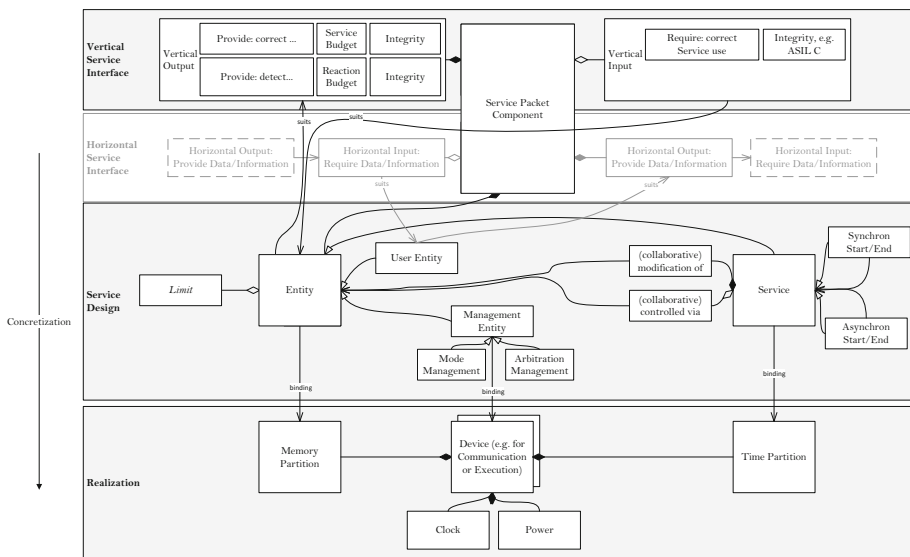


Fig. 4. Platform service dependency meta-model

In addition, we integrated the interference aspects for dependencies across abstraction layers (Fig. 1) within a meta-model (Fig. 4). This allows identifying failure propagation via shared service parts. Please note that a simple and automated comparison of modeled elements can be used to identify replicated service parts and that the propagation via service interaction is solved via the *Horizontal Service Interface* or the proposed fault tree generation.

The meta-model contains the interference aspects as follows: A potential corruption of a *User Entity* (if it is not a *Service*) is a *Spatial Partition* violation, e.g., a message corruption. A potential corruption of a *Limit* is an *Over-Allocation* in the case of an

allocation *Limit*, e.g., limited memory allocation size, and a *Behavioral Mode Corruption* in the case of a wear-out *Limit*, e.g., a limited number of write accesses to a flash memory before it precipitates. A potential *Management Entity Corruption* represents an unwanted *Behavioral Adjustment* and a delayed *Synchro Start/End* represents a temporal time-slot violation, while an *Asynchrony Start/End* that demanded is too frequently represents a *Contention Delay*. To find out whether an interference exists or not, the traces are evaluated via the relations depicted in Fig. 4. An interference potentially exists if there is a trace from one user (application component) via the *Vertical Service Interface* to another user.

Another concretization of the *Service Design* is the *Realization* of the *Service*. Each *Entity* has to be bound to a *Memory Partition*. Please note that a *Service* is a specialization of an *Entity*. In addition, each *Service* has to be bound to a *Time Partition*. *Memory-* and *Time Partitions* belong to a device that needs a *Clock* and *Power*. A similar *Realization* view can, for example, be found in [32], namely a hardware-software meta-model to support performance analysis. The *Realization* allows more modularity for the *Service Packets* and thus the *Service Design*, as well as a common cause analysis. More modularity is achieved because the basic events that represent random hardware failures can be shifted from the *Service Design* to a *Device*. Common causes can be found because a shared *Partition, Device, Clock, or Power* is a random common hardware cause.

5 Example Analysis

Following the presentation of the dependency analysis method in Sect. 4, this section shows parts of the execution of the method for an example application with corresponding services. The example is a power window application that the authors of [33] described for a timing benchmark. We deploy an example power window application on a POSIX-based multicore system with a software-based redundancy mechanism in-between as a service packet.

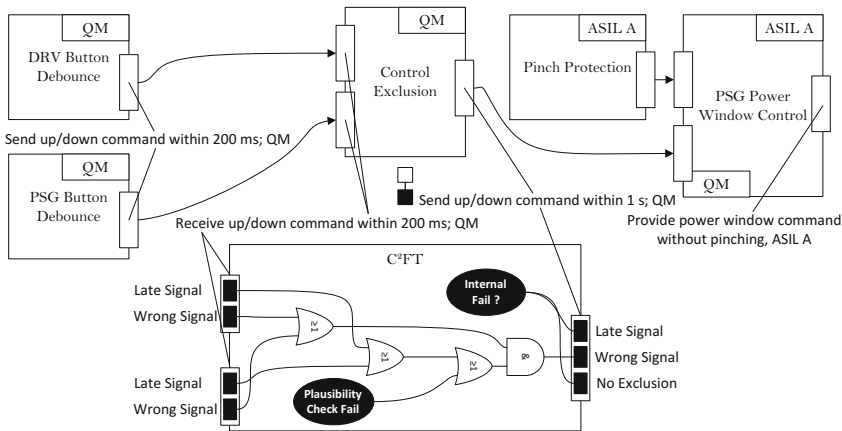


Fig. 5. Example dependencies within a power window application

The analysis process summarizes the previous chapter and contains six steps. First, the application components and the application interfaces should be modeled along the data/information flow. The starting point is the application component that provides the data/information that meets the safety goal. Second, the failure propagation within the horizontal application layer should be modeled, e.g. with C²FTs. Figure 5 depicts these two steps. The application is a power window. It can be automatically raised and lowered by pressing a button via the *PSG Power Window Control*. *Pinch Protection* detects whether an obstacle is present between the top of the passenger window frame and the glass pane when the window closes. *Control Exclusion* assigns lower priority to the passenger when control inputs come from the driver (DRV) and the passenger (PSG) simultaneously. *DRV Debounce* and *PSG Debounce* rebound the mechanical button when it is pushed [33]. In this example, the requirement “*Provide power window command without pinching, ASIL A*” fulfills the safety goal “*Unintentional closing of windows must be avoided, ASIL A*”.

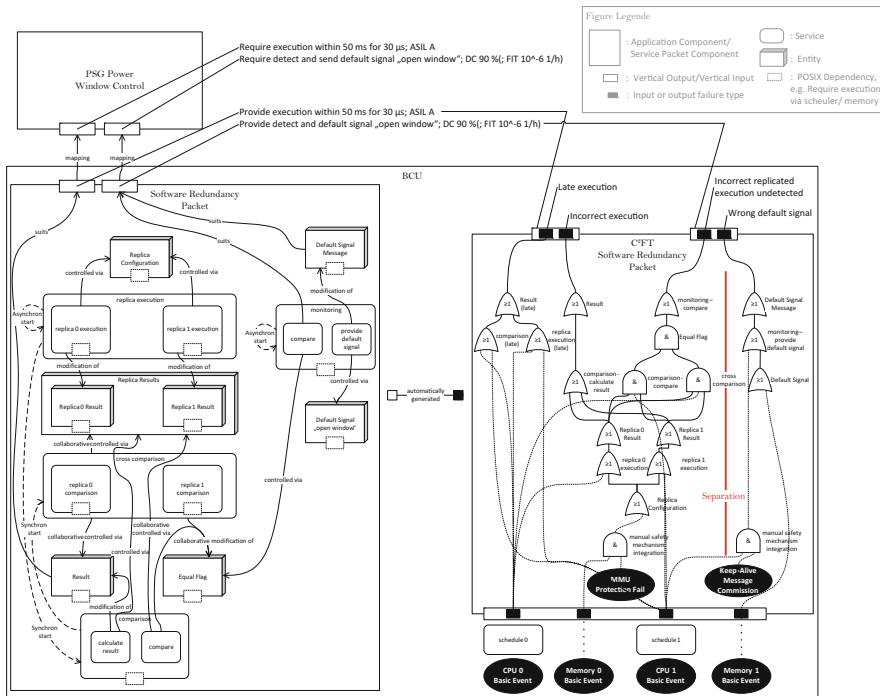


Fig. 6. Example platform service model. The red line indicates the separation between the default signal and the replicas that execute the application to achieve independence. (Color figure online)

However, if, for instance, *Control Exclusion* shares infrastructure with other application components and should be reused, the basic event ‘*Internal Failure*’ has to

be adapted in order to guarantee probabilistic independence. Furthermore, due to the different integrity levels (QM and ASIL A), an interference analysis is needed. To this end, we concretize the system across the abstraction with the remaining four analysis procedure steps. In the third step, the engineer models the vertical interface for the application component. This is exemplified in Fig. 6 for the *PSG Window Control* component. Please note that vertical inputs realize horizontal outputs and replace internal-failure basic events. In the fourth step, an engineer can then select or define the service components with their interfaces and map them to the application interfaces (in Fig. 6, a *Software Redundancy Packet* of a Body Control Unit (BCU) is illustrated). The fifth step is the creation of the service design (left part of Fig. 6). The example shows a mechanism that provides redundant execution via two replicas, including comparison, cross-comparison, and monitoring. The illustrated relations describe how the services influence the data objects (entities). In the case of a ‘*collaborative modification of*’-relation, the collaborating services can only modify the entity together. This results in a logical AND-gate within the fault tree. In the case of a ‘*modification of*’-relation, the service can always modify the corresponding entity. This results in a logic OR-gate within the fault tree. The ‘*controlled via*’-relation indicates that an entity influences the entity by which a service makes the modification. This relation is used to get the traces to create the fault tree and the interferences. The ‘*asynchronous start*’- and ‘*synchronous start*’-relations are used for temporal traces within the interference analysis and for fault tree generation (see, e.g., the *Late execution* failure mode in Fig. 6).

In the case of a composition, the containing entities get all the relations of the superordinate component. This improves clarity (see, e.g., cross-comparison in Fig. 6). Finally, with the sixth step, the result is created. The right part of Fig. 6 shows what a generated fault tree would look like. The fault tree is then enriched with two manually integrated protection mechanisms, “*MMU Protection Fails*” and “*Keep-Alive Message Commission*”. However, these mechanisms could also be integrated via a service and a corresponding device. The modeling technique allows automated generation of the fault tree (named gates and their relations). Safety mechanisms, e.g., MMU or keep-alive messages, as well as the description of the failure modes have to be added manually in the current version.

The interference analysis results are failure propagation scenarios listed in a table similar to what we presented in [12]. An example is, in case of a wrong assignment of the *setDefaultSignal*-service to set the *Default Signal Message*:

‘The application *Control Exclusion* overrides the *Default Signal Message* via the service *setDefaultSignal*. → The *Default Signal* of the service packet *software redundancy* is corrupted. → The application *PSG Power Window Control* receives a wrong *Default Signal Message* provided by the application *Control Exclusion*. → The application *PSG Power Window Control* receives a wrong window command. → The safety goal is violated.’

The modeling technique shown in Fig. 6 allows creating compositions and hierarchies via sub-services and sub-entities and can thus be applied for larger systems.

6 Conclusion

In this paper, we presented an approach for model-based safety analysis across abstraction layers. The analysis is based on modeling the dependencies within and across abstraction layers. Through the application of our new method, we achieved improved reusability compared to component-based safety analyses like C²FTs, which focus only on one abstraction layer. The method achieves completeness for the dependency analysis since all dependency aspects are covered for a given system model. However, robustness tests are still required to check the implementation. In addition, a safety engineer can perform a dependent failure analysis of the design before the system is implemented or complex simulation models are built. Our meta-model enables automated generation of fault trees and thus facilitates the safety analysis and its auditability. We applied the analysis to a real-world automotive redundancy mechanism within an integrated architecture. In this paper, we demonstrated our approach with a simplified example. During the application of our method, we also detected limitations of this method. Modeling the complete system with our approach results in nearly the same effort as creating a fault tree analysis. However, our approach detects common causes via shared devices and interference via platform services in addition to fault trees. We identify common causes iterating over modeled devices (Fig. 4) and interferences via modeled dependencies from the classification (Fig. 1). A second limitation is that fault tree analysis always considers the worst case of a temporal dependence. Yet our model and existing design languages contain temporal information. Currently, we are using temporal information only to identify interference sequences. In future work, we will develop a better representation for the interference results or use other techniques, e.g., Markov Chains, instead of fault trees. To reduce the effort for the analysis, we will work on integration of our language into SysML in order to automate as many steps of the model generation as possible. Another area of potential future work is to combine our analysis systematically with fault injection analysis and experiments.

Acknowledgments. We acknowledge financial support for this work from the German Federal Ministry of Education and Research (BMBF) in the projects “ARAMiS II” (01IS16025) and “Software Campus” (01IS12053). All responsibility for the content remains with the authors.

References

1. QNX Auto Blog. <http://qnxauto.blogspot.de>. Accessed 22 Feb 2018
2. Kopetz, H., Obermaisser, R., El Salloum, C., Huber, B.: Automotive software development for a multi-core system-on-a-chip. In: Proceedings of the 4th International Workshop on Software Engineering for Automotive Systems. IEEE Computer Society, May 2007
3. ISO: ISO 26262 - Road vehicles - Functional safety (2011)
4. IEC: IEC 61508 - functional safety of electrical/electronic/programmable electronic safety-related systems (2010)
5. RTCA: DO-178C: Software Consideration in Airborne Systems and Equipment Certification (2012)
6. SYSGO Homepage. <https://www.sysgo.com>. Accessed 22 Feb 2018

7. BlackBerry Homepage. <http://blackberry.qnx.com/en/sdp7>. Accessed 22 Feb 2018
8. AUTOSAR development partnership, Specification of Operating System (v 5.3.0) (2014)
9. Schirmeier, H., Hoffmann, M., Kapitza, R., Lohmann, D., Spinczyk, O.: Fail*: towards a versatile fault-injection experiment framework. In: ARCS Workshops (ARCS) 2012, pp. 1–5. IEEE, February 2012
10. John, R.: Partitioning in avionics architectures: requirements, mechanisms, and assurance (1999)
11. Kotaba, O., Nowotsch, J., Paulitsch, M., Petters, S.M., Theiling, H.: Multicore in real-time systems—temporal isolation challenges due to shared resources. In: Workshop on Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems, March 2013
12. Zimmer, B., Dropmann, C., Hänger, J.U.: A systematic approach for software interference analysis. In: Software Reliability Engineering (ISSRE) 2014. IEEE, November 2014
13. Dropmann, C., Amorim, T., Ruiz, A., Schneider, D.: Towards safe mixed critical embedded multi-core systems in dynamic and changeable environments. CPS Week EMC2, Vienna, Austria, April 2016
14. OMG SysML Website. <http://www.omg.sysml.org>. Accessed 05 Mar 2018
15. Zimmer, B., Bürklen, S., Knoop, M., Höfflinger, J., Trapp, M.: Vertical safety interfaces – improving the efficiency of modular certification. In: Flammini, F., Bologna, S., Vittorini, V. (eds.) SAFECOMP 2011. LNCS, vol. 6894, pp. 29–42. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24270-0_3
16. Schneider, D., Trapp, M.: Conditional safety certification of open adaptive systems. *ACM Trans. Auton. Adapt. Syst. (TAAS)* **8**(2), 8 (2013)
17. Feiler, P.H., Gluch, D.P., Hudak, J.J.: The architecture analysis & design language (AADL): an introduction (No. CMU/SEI-2006-TN-011). Carnegie-Mellon University, Pittsburgh, Software Engineering Institute, PA (2006)
18. EAST-ADL Association: EAST-ADL Domain Model Specification. Version V2.1.12. EAST-ADL Association, Göteborg (2013)
19. Hilbrich, R., Behrisch, M.: Improving the efficiency of dislocality constraints for an automated software mapping in safety-critical systems (2018)
20. Papadopoulos, Y., Walker, M., Parker, D., Rüdte, E., et al.: Engineering failure analysis and design optimisation with HiP-HOPS. *Eng. Fail. Anal.* **18**(2), 590–608 (2011)
21. Höfig, K., Trapp, M., Zimmer, B., Liggesmeyer, P.: Modeling quality aspects: safety. In: Pohl, K., Hönninger, H., Achatz, R., Broy, M. (eds.) *Model-Based Engineering of Embedded Systems*, pp 107–118. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34614-9_8
22. Kaiser, B., Weber, R., Oertel, M., Böde, E., Nejad, B.M., Zander, J.: Contract-based design of embedded systems integrating nominal behavior and safety. *Complex Syst. Inf. Model. Q.* **4**, 66–91 (2015)
23. Höfig, K., Zeller, M., Heilmann, R.: ALFRED: a methodology to enable component fault trees for layered architectures. In: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 167–176. IEEE, August 2015
24. Vitali, E., Palermo, G.: Early stage interference checking for automatic design space exploration of mixed critical systems. In: *Proceedings of the 9th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, p. 3. ACM, January 2017
25. Sari, B., Reuss, H.C.: A model-driven approach for dependent failure analysis in consideration of multicore processors using modified EAST-ADL (No. 2017-01-0065). SAE Technical Paper (2017)
26. Di Vito, B.L.: A model of cooperative noninterference for integrated modular avionics. In: *Dependable Computing for Critical Applications* 7, 1999. IEEE, January 1999

27. Dunj3, J., Fthenakis, V., V3lchez, J.A., Arnaldos, J.: Hazard and operability (HAZOP) analysis. A literature review. *J. Hazard. Mater.* **173**(1–3), 19–32 (2010)
28. Auerswald, M., Herrmann, M., Schulte-Coerne, V.: Entwurfsmuster f3ur fehlertolerante softwareintensive Systeme (Design Patterns for Fault-Tolerant Software-Intensive Systems). *at-Automatisierungstechnik Methoden und Anwendungen der Steuerungs-, Regelungs-und Informationstechnik*, 50(8/2002), 389 (2002)
29. Feth, P., Adler, R.: Service-based modeling of cyber-physical automotive systems: a classification of services. In: *Workshop CARS 2016-Critical Automotive Applications: Robustness & Safety*, September 2016
30. Avizienis, A., Laprie, J.C., et al.: Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secure Comput.* **1**(1), 11–33 (2004)
31. M3hrle, F., Bizik, K., Zeller, M., H3fig, K., Rothfelder, M., Liggesmeyer, P.: A formal approach for automating compositional safety analysis using flow type annotations in: component fault trees. In: *Risk, Reliability and Safety: Innovating Theory and Practice: Proceedings of ESREL*. Taylor & Francis, CRC Press, Portoroz, Slovenia, June 2017
32. Amalthea Project Homepage. <http://www.amalthea-project.org/>. Accessed 01 Mar 2018
33. Li, H., De Meulenaere, P., Hellinckx, P.: Powerwindow: a multi-component TACLeBench benchmark for timing analysis. *Advances on P2P, Parallel, Grid, Cloud and Internet Computing. LNDECT*, vol. 1, pp. 779–788. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-49109-7_75

Verification



Formal Verification of Signalling Programs with SafeCap

Alexei Iliasov¹(✉), Dominic Taylor², Linas Laibinis³,
and Alexander Romanovsky¹

¹ Newcastle University, Newcastle upon Tyne, UK
alexei.iliasov@ncl.ac.uk

² Systra Scott Lister, London, UK

³ Institute of Computer Science, Vilnius University, Vilnius, Lithuania

Abstract. SafeCap is a modern toolkit for modelling, simulation and formal verification of railway networks. This paper discusses the use of SafeCap for formal analysis and fully-automated scalable safety verification of solid state interlocking (SSI) programs – a technology at the heart of many railway signalling solutions. The focus of the work is on making it easy for signalling engineers to use the developed technology and thus to help with its smooth industrial deployment. In this paper we explain the formal foundations of the proposed method, its tool support, and their application to real life railway verification problems.

1 Introduction

Effective signalling is essential to the safe and efficient operation of a railway network. It enables trains to travel at high speeds, run close together, and serve multiple destinations. Whether by mechanical semaphores, colour lights or electronic messages, signalling only allows trains to move when it is safe for them to do so. Signalling locks moveable infrastructure, such as the points that form railway junctions, before trains travel over it. Furthermore, signalling often actively prevents trains travelling further or faster than it is safe and sometimes even drives the trains. At the heart of any signalling system there are one or more *interlockings*. These devices constrain authorisation of train movements as well as movements of the infrastructure to prevent unsafe situations arising.

The increasing complexity of modern digital interlockings, both in terms of the geographical coverage and that of their functionality, poses a major challenge to ensuring railway safety. Even though formal methods have been successfully used in the railway domain (e.g. [2,3]), their industry application is scarce. In spite of a large body of academic studies addressing issues of formal verification of railway systems, they typically remain an academic exercise due to a prohibitive cost of initial investment for their industrial deployment. The following are some of the reasons. First, signalling engineers need to learn mathematical notations to apply them. Second, the tools often cannot be applied for analysing large real stations due to their poor scalability. Third, the companies need to drastically change the existing development processes in order to use them.

This paper proposes a formal tool-based approach that addresses these issues by (i) verifying the signalling programs and layouts developed by signalling engineers in the ways they are developed by industry, (ii) ensuring fully-automated verification of safety properties using a family of the state of the art verification techniques (in particular, automated theorem provers and solvers), and (iii) providing diagnostics in terms of the notations used by the engineers. All together, this affirms that the developed methods and tools can be easily deployed to augment the existing development process in order to provide extra guarantees of the railway safety.

The paper is structured as follows. Section 2 presents the work background by overviewing the SafeCap toolkit, the role of SSI programs in railway signalling, and the key safety principles that these programs must follow. In Sect. 3 we discuss the SafeCap verification core, including its underlying modelling language and essential verification techniques. The proposed verification method is illustrated by a case study of a real railway station in Sect. 4. Finally, Sect. 5 concludes the paper by summarising the achieved results.

2 Background

SafeCap Platform. The SafeCap platform is a toolkit for modelling railway capacity and verifying railway network safety [11]. It allows signalling engineers to design stations and junctions relying on the provided domain specific language (SafeCap DSL), as well as to check their safety properties and evaluate potential improvements of capacity by using a combination of theorem proving, SMT solving and model checking [12]. The platform has been substantially extended by adding new simulators, solvers and provers, as well as the support for representing a wide range of the existing signalling frameworks [14, 15].

This paper also takes our work on SafeCap further by developing a set of new tools for importing, analysing and proving safety of railway data in standard SSI and SSI-based technologies such as Smartlock¹ by Alstom and WESTLOCK² by Siemens. The overall SafeCap architecture is presented in Fig. 1. Verification of SSI is our first experience with constructing and verifying large (i.e., containing tens of thousands of state transitions) models of the system dynamic behaviour. Previously, our industrial experience was concerned solely with verification of static data. The current work extends the SafeCap framework with advanced capabilities for reasoning about dynamic (i.e., transition-based) systems.

The developed SafeCap verification and proof back-ends enable automated reasoning about static and dynamic properties of railways or their signalling data. Our two principal verification routes are the built-in symbolic prover backed by a SAT solver, a range of external provers provided via the Why3 framework [4], and the ProB model checker [17] (used just as a constraint solver).

¹ For more details, see <https://www.mobility.siemens.com/mobility/global/en/interurban-mobility/rail-solutions/rail-automation/electronic-interlockings/pages/electronic-interlockings.aspx>.

² For more details, see <http://www.alstom.com/products-services/product-catalogue/rail-systems/signalling/products/smartlock-interlocking-products/>.

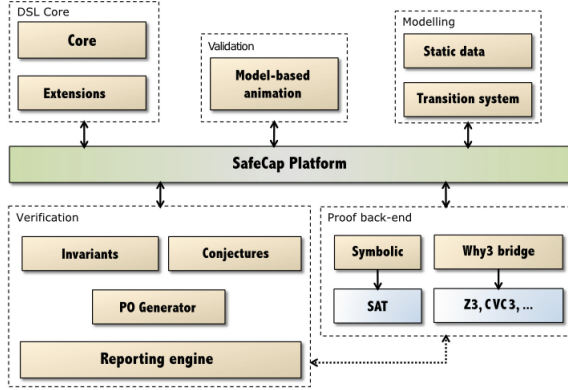


Fig. 1. SafeCap architecture

The SafeCap DSL provides a formal, graph oriented way to capturing railway schemas and some aspects of signalling [13]. In the DSL, a railway schema is a mathematical object consisting of data structure definitions (namely, types and constants) as well as required logical constraints on the defined data (axioms and lemmata). We can distinguish two main parts of the SafeCap DSL – the Core and its various extensions. The Core provides means to mathematically describe the physical topology of a railway schema (or, in fact, any graph-based structures). Its first-class concepts are graphs and subgraphs. These typically represent track topology, track circuits, routes or axle counters. As such, a model of a railway schema in the SafeCap DSL Core is independent of any given signalling solution.

Once a railway schema model is created (or imported from an external format) in the Core, it is checked for its validity or well-definedness. For that, a number of graph theoretical statements are automatically generated and verified, including isomorphism properties between constituent subgraphs, path validity within a given graph, connectivity, acyclicity, node degree and so on.

Various concepts of a railway schema such as signals and signalling solutions, speed limits, stopping points and so on can be incorporated into via DSL extension plug-ins. Such plug-ins introduce new data (as custom annotations) and supporting logic (as additional logical constraints or relationships). Such a tool architecture allows us not to commit to any regional technology and thus to offer a broadly similar approach for a range of legacy and current technologies.

In this paper we focus on the fixed-block signalling prevalent in the UK and common to many other countries. In doing that, we rely on a dedicated SafeCap DSL plug-in for incorporating this type of signalling data.

Computerised Signalling and SSI. The first signalling interlockings were mechanical devices that constrained the movement of levers, were connected to points and semaphore arms, and were contained in mechanical signalling boxes. During the twentieth century these devices were superseded by electrical relay-based interlockings that switched electrical current to motorised points, colour

light signals and other lineside devices. The safety conditions applied by relay interlockings included tests of track circuits or axle counters – the devices that automatically determine whether a section of line has a train on it. The advent of computer technology brought the opportunity to replicate and build on the relay interlocking functionality within a computer based interlocking.

```

*QR117B(M) / route request block for route R117B(M)
if R117B(M) a / route R117B(M) is available
    USD-CA f,OSV-BA f,OSV-BA f / sub-route and sub-overlaps are free
then if OSL-AC l, / sub-overlap is OSL-AC locked
    P223 fr , P224 fr / points P223, P224 free to move reverse
    then @P223QR \ / call subroutine P223QR
if OSD-BC f / sub-overlap is OSD-BC is free
    LTR04 xs / latch (boolean flag) not set (false)
    P224 crf / point P224 commanded reverse or free to move reverse
then R117B(M) s / set route set flag for R117B(M)
    USD-AC l , USC-AB l , USB-AB l , OSA-AB l / set sub-routes/overlaps
    P224 cr / command point P224 reverse
    LARR xs / clear latch LARR
    S117 clear bpull / clear signal button pull flag
    if P223 xcr , P223 rf then / check point states
        @P223QR / point command subroutine
        EP230 = 0 \ / reset timer EP230

```

Fig. 2. SSI example: route request code for route R117B(M) in the PRR module

One of the earliest forms of computer based interlocking was the Solid State Interlocking (SSI), developed in the UK in the 1980s through an agreement between British Rail (the then nationalised railway operator) and two signalling supply companies. Running on bespoke hardware, SSI software consists of a core application (common to all signalling schemes) and site specific geographic data. The original SSI has now been superseded by more powerful, modern hardware platforms running software developed in accordance with modern standards for safety critical software. Nonetheless, the functionalities of the core application and the Geographic Data Language (GDL) remain largely unchanged.

SSI GDL data configures a signalling area by defining site specific rules, concerning the signalling equipment as well as internal latches and timers that the interlocking must obey. Despite being referred to as data, a GDL configuration resembles a program in a procedural programming language. The configuration is iteratively executed in three major stages: reception of input state messages from signalling equipment, followed by execution of rules, followed by construction and transmission of output command messages to the signalling equipment.

There are two main modules defining the signalling behaviour – the route and point request (the PRR module) and formation of output telegrams (the OPT module). An example of route request code (a part of logic that reacts to an external route request) is given in Fig. 2. Notice that the parts between **if** and **then** are atomic predicates combined with implicit conjunction, while everything between **then** and a slash character is a command (made of a sequence of atomic commands). For instance, `USD-CA f` stands for *test that subroute USD-CA f is free*, while `USD-CA l` commands the subroute to be freed.

Safety Principles. There are two main safety principles shared by all signalling operations that employ the SSI technology. From these a large number of operational constraints can be derived, that consequently become verification conditions to check against given signalling data.

A schema must be free from collisions. A collision happens, potentially, when two trains may occupy the same part of a track at the same time. In route-based as well as speed-based signalling, the principal mechanism to address this property is that of route locking and holding. A train is given permission to enter an area of a railway, once there is a continuous and safe path through the area assigned exclusively to this train. Such a path is normally called a *route* and is delineated by *signals* – either physical track-side signals with lamps or conceptual signals displayed to a driver on a computer screen. The extent between successive signals defines the smallest train separation.

For a route to be locked, all the movable equipment such as *points* or level crossings must be set and detected in a position that would let a train safely travel on its desired route. They must remain locked in such a state and their position must be positively confirmed before a train enters the route.

A schema must be free from derailments. A derailment may happen when a train moves over a point that is not set in any specific direction and thus may move under a train. To avoid this, a point must be positively confirmed to be *locked* before a train may travel over it. Typically, a signaller must define conditions under which point reconfiguration is considered safe.

Related Work. There have been a number of studies focusing on formal verification of SSI programs. The majority of works (e.g., [10, 16, 20]) use various forms of model-checking in an attempt to verify safety of *train run scenarios*, with interlocking rules derived manually or via an automated translation from SSI data. With few exceptions, the proposed techniques actually scale up to only toy examples, or cover a small subset of functionalities, or both. For instance, the approach presented in [10] uses NuSMV to model check a small subset of safety properties for a selected subset of SSI data based on real-life signalling data. In the face of sheer number of train run scenarios, one way to avoid the state explosion problem might be statistical simulation of train runs [6]. However, this approach has non-trivial implications on result interpretation.

We see a fundamental flaw in all such scenario exploration techniques: by introducing train runs and assuming certain traffic patterns they cannot find, even if they were to scale up, serious signalling mistakes that do exist in real-life implementations and only manifest themselves when a combination of several rare conditions happen [8]. Our approach does not suffer from this limitation as we do not need consider train runs (and thus limit verification to few assumed possibilities). Instead we check the worst case safety implications for all possible train run scenarios. Another problem with these solutions is their poor diagnostics, where the feedback on safety violations is not given in terms that signalling engineers could understand (i.e., SSI and the schema language).

In [5] the authors build a model of railway operation constrained by imported signalling data. A model checker automatically explores train movement

scenarios (i.e., model states) and reports on violation of safety properties. The technique does not support generic safety properties (which have to be written separately for a specific layout) and the reported result indicate it is unlikely to scale to the industrial scale. In cases where a track graph can be cut with a very spectral ratio (i.e., two stations connected by a straight graph), it is sound to conduct verification of subparts separately [18, 19]. This is not often found in practice as SSI is traditionally limited to 64 or 256 controlled pieces of equipment and it is impractical to wire equipment at significant distance from a control box.

Verification and validation of a fragment of *safety logic* for European Railways Train Management System (ERTMS), ensuring also interoperability of different signalling solutions, is described in [7]. ERTMS specifications (written in a structured programming language) are automatically translated into formats of the employed external verification tools. Paper [9] presents an ongoing work on automatic model generation and verification of Railway Markup Language (RailML) formatted data, which also include route tables and interlocking information. Interlocking programs are defined in RailML using route scheduling and route automata. Neither of these approaches however could be applied for verification of SSI programs. Moreover, contrary to our work, they heavily rely on model checking techniques and tools for verification of railway safety properties.

3 Modelling and Verification in SafeCap

In this section we present our main contribution – the integrated generic framework for modelling and verification in SafeCap that we rely on to verify safety properties of railway signalling.

3.1 SafeCap Data Analytics

In our earlier works [12, 14, 15] we have proposed a formal model to capture and verify concrete signalling constraints by enforcing a certain standard of input data representation. However, industrial applications do not easily fit into the proposed view and there is a wide variation in the kind, rigour and comprehensiveness of the data defining existing signalling designs.

To be able to deal with varying forms of signalling input data, we complement SafeCap DSL with a generic modelling framework. The framework, called SafeCap Data Analytics (SDA), offers modelling concepts similar to that of a state-based modelling language. It is not meant to be used by an end user but rather as an intermediary tool bridging signalling input data and generated verification conditions. The SDA approach allows us to incorporate any extensions that require non-trivial reasoning (in particular, specific signalling solutions) with the DSL Core in an uniform and mathematically consistent way.

An SDA model comprises the static part, defining model constants, axioms, as well as verification statements (called conjectures), and the dynamic part, defining state transitions over model variables and thus expressing possible state

evolution. Overall, it can be seen as a characterisation of a state transition system with discrete time (SSI timers are seen purely as integer counters enabling causation reasoning) and state transitions are assumed to fire in an atomic fashion. As the focus is exclusively on static proof, we only define the proof semantics and do not consider construction of model states or traces.

We employ first-order logic equipped with the Zermelo-Fraenkel version of set theory and arithmetics to write predicates defining system axioms, conjectures as well as pre- and post-conditions of state transitions. Relational and functional model structures are expressed as special kinds of sets (i.e., sets of mappings between associated elements) and variable values can be drawn from finite or infinite sets. There are also the predefined sets of integers, booleans and reals. The notation and underlying formal semantics of a transition system (a variation of the weakest precondition semantics) are adopted from the B Method [1].

In relation to the railway domain, for each format of input data representing signalling data, there is a dedicated importing plug-in translating it into an SDA model: a collection of constants, axioms and state transitions. The number of such formal elements for a real life example is quite large – from several thousands to tens of thousands. The resulting formal model is a solid foundation for formal reasoning about the properties, in particular operational safety, of a chosen signalling design. At the moment SafeCap supports two schema formats – LDL (proprietary) and RailML – and two signalling data formats – SSI and XML-based (a proprietary schema).

There are three main classes of signalling models distinguished by the mix of axioms (static constraints) and state transitions (system dynamics):

- a purely static model reasoning about data with no model variables or state transitions. An example is a model derived from a set of *control tables* – signalling design data represented in a tabular form. For a verification tool, its is a collection of conjectures (lemmata) expressing data properties;
- a purely dynamic model where signalling is defined by state transitions. An example is SSI signalling data that we see as piece of code to be transformed into a state transition system. Such models are verified via safety invariants;
- a mixture of the two. An example is verification of equivalence between a control table and its implementation SSI data.

Next we consider how verification of signalling data differs depending on the class of a considered SDA model.

3.2 SDA Verification: A Static Model

For a static SDA model, its verification involves proving a set of logical conjectures expressing the required data consistency properties. By a conjecture we understand a predicate (logical condition) constraining the model constants. Such a conjecture must be proven in the context of model axioms, i.e.:

$$\text{ctx}(c) \vdash \text{conj}(c), \tag{1}$$

where c are model constants. Here $\text{ctx}(x)$ represents a set of axioms from the DSL Core such as the railway topology data definitions as well as all incorporated extensions. Predicate $\text{conj}(c)$ stands for an expected data property or a constraint to be implied by such core definitions.

As one example, we might wish to check that a route setting control table includes all the points necessary to be set reverse to enable the given path of the route. This statements translates into the following conjecture:

$$\begin{aligned} \forall r \in \text{Route} \cdot r \in \text{dom}(\text{Routes.Point}) \\ \text{Node.base}[\text{schema.reversepoints}[\{r\}]] \subseteq \\ (\text{Points.base}[(\text{Points.base}^{-1}[\{r\}] \cap \text{ran}(\text{Routes.Point}[\{r\}]))]) \end{aligned} \quad (2)$$

Here $\text{schema.reversepoints}[\{r\}]$ is the topology derived set of points to be set reverse to enable the route r , while $\text{Routes.Point}[\{r\}]$ defines an ordered list of required points. The (topology-derived) constant relations Node.base and Points.base map between the physical and logical points and between the point names and the point states respectively. Finally, $[\cdot]$ and $(\cdot)^{-1}$ are relational image and inverse operators. For more details on the used notation, see [1].

Depending on the kind and form of $\text{ctx}(c)$, a property $\text{ctx}(c) \vdash \text{conj}(c)$ can be handled by a constraint solver, a symbolic prover (SMT solver), or a satisfiability (SAT) solver. There are several provers available within SafeCap, such as Why3, ProB, or Minisat. In the production version of the tool, a conjecture is always checked by at least two distinct provers, one of which must be external.

There are a number of requirements to satisfy for a conjecture to be deemed logically meaningful and well-formed. Overall, a conjecture must not be a contradiction or tautology. The reason for these checks is to avoid conjectures that are logically inconsistent or those that are true or false irrespectively of a verified schema or signalling data. The latter cases, while logically consistent, in practice indicate serious mistakes in the formulation of a conjecture predicate.

3.3 SDA Verification: A Dynamic Model

For a dynamic SDA model, its verification boils down to proving an inductive system invariant expressed as set of predicates. For simplicity, we refer to each such a predicate as a safety invariant. A safety invariant represents a property on the system state (variables) to be maintained during the system functioning. Safety invariants formalise the established principles of interlocking operation. They are formulated manually with the help of domain experts and translated into a formal notation. This is done once for any given technology (e.g., SSI).

To show that an invariant property is indeed preserved by the system, one must prove a logical sequent (theorem) of the following form:

$$\text{ctx}(c) \wedge \text{inv}(c, s) \wedge \tau(c, s, s') \vdash \text{inv}(c, s'), \quad (3)$$

where $\text{ctx}(c)$ are all the defined axioms constraining the model constants c , $\text{inv}(c, s)$ is a safety invariant over the constants c and the current state (variables) s , and $\tau(c, s, s')$ is some state transition producing a new state s' . $\tau(c, s, s')$ is

usually defined by a conjunction of transition pre- and post-conditions: $\text{pre}(c, s) \wedge \text{post}(c, s, s')$. Finally, $\text{inv}(c, s')$ is an invariant over the new state s' .

It is convenient to generalise the above statement to also account for some historic (previous) model state. We refer to such a state as s_h and understand it as the state observed prior to the current state s :

$$\text{ctx}(c) \wedge \text{inv}(c, s, s_h) \wedge \tau(c, s, s') \vdash \text{inv}(c, s', s) \quad (4)$$

Historic states are not manipulated in state transitions. The only source of information about a historic state is the invariant inv . Conceptually, when a transition happens, the old state (s) takes the place of the historic state (s_h) and the new state s' replaces the old state s . Since we are doing symbolic proof, this is all we need to know about historic states. The principle can be generalised to arbitrary deep historic trace although we did not encounter a need for this.

As an example, the following concrete invariant checks that the minimal conditions of point switching are met:

$$\forall p \in \text{Node} \cdot \text{point_c}(p) \neq \text{point_c_h}(p) \Rightarrow \text{schema.pointcleartracks}[\text{Node.base}^{-1}[\text{Node.base}(p)]] \cap \text{track_o} = \emptyset \quad (5)$$

Here model variables are given in italic, while all the other identifiers are constants originating from the underlying model railway schema. The model variable point_c_h is a historic version of the current-state variable point_c .

The verification conditions for such a model are generated by instantiating model invariants. Namely, a separate verification condition (proof obligation) is created for each pair of invariant $\text{inv}(c, s, p)$ and state transition $\tau(c, s, s')$. Thus, for 10 invariants and 2000 transitions, there are up to 20000 conditions to prove.

Transition system proofs are not as well suited for constraint solving as conjectures about control tables. The primary reason is the abundance of complex abstract relations. Off the shelf provers, such as E, SPASS, Z3, have proven themselves capable but are unable to return any valuable feedback on failed proofs and are generally quite slow (typically about 20 s per proof obligation) and memory demanding (some proofs require up to 64 GB memory). To address this, we have developed a custom symbolic prover, which is described next.

The built-in SafeCap symbolic prover is used as the primary means for verifying safety invariants. Unlike conjectures for a static SDA model that are typically proven for concrete constant values, generated safety invariant proof obligations are stated over all permissible state values. The complexity of constraints and, to much less degree, the scale of state space make it an inordinately difficult verification task for a constraint solver.

The symbolic prover starts with the invariant statement as the top goal and tries to simplify, split or rewrite this goal until it becomes trivially true. It relies on a number of *tactics* – functions that implement goal transformations (like splitting a conjunctive goal into several subgoals).

Since the prover is supposed to be used fully automatically, a special attention in its implementation is given to the cases when it fails to prove its goal. The prover is designed to stop in a state best suited to the subsequent interpretation.

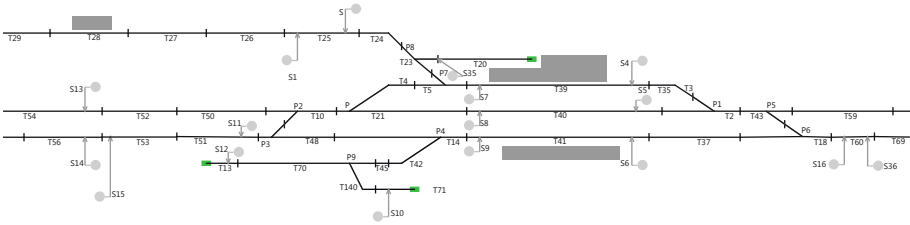


Fig. 3. Case study railway schema (an excerpt).

To achieve this, once the prover detects a failed proof branch (due to a time out, an absence of applicable tactics, etc.), it backtracks while looking for a historic undischarged goal matching one of the predefined templates. For each of these templates, there is its defined interpretation in a natural language to be shown to an engineer to assist with the understanding of the nature of an error.

The symbolic prover is not designed to be ever used interactively or even outside of SafeCap. However, it is possible to write dedicated tactic scripts for each safety invariant. Such scripts can reorder, remove and parametrise proof tactics as well as define different proof branches using applicability tests. In particular, the form and nature of SSI data allows us to easily recognise repeating patterns in the usage of SSI commands and, even without seeing a specific SSI data instance, we are able define efficient tactics to support a safety invariant.

We use two complementary techniques to demonstrate that the built-in prover is sound. First, all the rewrite rules are known to be valid lemmata in first order logic and set theory³. Second, for each instance of a rewrite rule, SafeCap can generate many thousands of theorems originating from successful or failed proof obligation and check them in an external prover. This technique is a form of automated mass testing to guard against programming mistakes. It can also be used to automatically recheck all proof scripts for extra reassurance.

Much attention is given to the presentation of verification results. It is imperative that the formal verification core operates autonomously and presents its findings in a clear and useful manner. To achieve this, together with every static or invariant verification condition one must write one or more reporting templates that define the mapping of verification output into a report coherent to a domain expert. The examples of such templates are given in Sect. 4.

4 Case Study

As a case study, we consider a railway schema and the accompanying SSI signalling program for a real medium-size station. The SSI data for the case study were developed by an industrial company using the existing process. Verification was conducted against the properties mandated by the national railway authority (Network Rail).

³ With the unfortunate exception of arithmetics that is handled as a black-box rewrite.

```

pre
LTR04  $\notin$  latch_s  $\wedge$  OSD-BC  $\notin$  overlap_l  $\wedge$  R117B(M)  $\in$  route_a
OSC-BA  $\notin$  overlap_l  $\wedge$  OSV-BA  $\notin$  overlap_l  $\wedge$  USD-CA  $\notin$  subroute_l
LTR117  $\notin$  latch_s  $\wedge$  LTR119  $\notin$  latch_s
(REVERSE = point_c(P224))  $\vee$  ((TSD  $\notin$  track_o)  $\wedge$  (USD-BC  $\notin$  subroute_l)  $\wedge$  ...
request = QR117B(M)
post
route_s' = route_s  $\cup$  {R117B(M)}

```

Fig. 4. An example of a translated SSI transition: route locking

A small part of a diagrammatic (not to scale) representation of its layout is given in Fig. 3. To give a sense of its actual size, the area consists of 117 train detection track circuits, 12 points and 42 routes. The labels in the diagram had to be obfuscated for the purposes of this publication. The signalling data (following the SSI standard) are defined in 14 separate modules, summing up to 274 KB of disk space. The modules contain plain text source of SSI signalling in the SSI format described above.

The case study data were loaded into SafeCap in two stages. First, the railway schema was imported and represented in the SafeCap DSL Core. Second, the SSI signalling program was added (using the dedicated plug-in) as a DSL Core extension based on the SafeCap SDA. When a digital version of a railway schema exists, it can be imported directly into SafeCap. If it is only available as a paper or digital scan representation, it has to be manually drawn in the SafeCap schema editor. This takes about half a day for an experienced railway engineer.

A railway schema typically contains the track topology, track joints and signals. From this, the platform generates the necessary derived information (such as track circuits, points, routes, subroutes, overlaps, etc.) that is represented and stored in the SafeCap DSL. SafeCap also attempts to automatically decode route names to match paths on the schema. For instance, R117B(M) would normally refer to a route starting from the signal S117 and taking the path B.

We treat SSI signalling data as a program made of large number of independent units (essentially event handlers). Furthermore, every such unit can be translated into a number of state transitions (one per each command). The result is a completely flat structure made of thousands of individual state transitions.

Figure 4 illustrates one such translated state transition. The transition describe route setting resulting from the route request presented (in SSI GDL) in annotated Fig. 2. The route set update is specified in the transition postcondition. All the transition preconditions (apart from the last two) can be traced directly to the conditions of respective **if** blocks. The penultimate precondition is the result of expanding the SSI GDL expression P224 **crf**, testing whether point P224 is already commanded reverse or is free to move into the reverse position. The last precondition associates this transition with the specific request type.

To conduct verification of an SSI data set, the underlying schema model and the generated dynamic SDA model (transition system) are integrated together.

Additionally, the overall system model also includes safety invariants to be verified against the schema data and system state transitions.

The following is one example of a safety invariant. The invariant is concerned with route setting protocol. In particular, it ensures that the conflicting routes going in opposite directions cannot be set at the same time, which can be formulated as specific conditions on the current free and locking subroutes.

$$\begin{aligned} \forall ra \in \text{Route} \cdot ra \in \text{route_s} \wedge ra \notin \text{route_s_h} \Rightarrow \\ \forall rb \in \text{Route} \cdot rb \in \text{routeopposing}[\{ra\}] \wedge \text{routedir}(ra) \neq \text{routedir}(rb) \wedge \\ \text{routelast}(rb) \in \text{ran}(\text{routetracks}[\{ra\}]) \Rightarrow \\ \text{subroute_l} \cap \text{LastSubRoute}[\{rb\}] = \emptyset \end{aligned} \quad (6)$$

In the above, *route_s* is a model variable of type $\mathbb{P}(\text{Route})$, representing a set of routes. *route_s_h* is its historic counterpart. The identifiers *routeopposing*, *routedir*, *routelast*, *routetracks* are schema-derived constant relations. Specifically, the above invariant requires that, for any route *ra* and its opposing route *rb* such that the *rb* exit is within the *ra* extent ($\text{routelast}(rb) \in \text{ran}(\text{routetracks}[\{ra\}])$), the last shared sub-route of *rb* must be checked free.

Overall, for route setting alone, we define 14 different invariants corresponding to 6 distinct safety principles. There are more invariants addressing telegram formation, flag operations and point commanding. Currently, we do not check timeliness conditions (on system reactions within a certain number of cycles). We also do not consider any liveness conditions as progress and the absence of livelocks and deadlocks is not part of interlocking safety requirements.

(the model constants and axioms (implied))
(the safety invariant INV6)
 $\forall ra \in \text{Route} \cdot ra \in \text{route_s} \wedge ra \notin \text{route_s_h} \Rightarrow \dots$
(the transition preconditions)
 $\text{LTR04} \notin \text{latch_s} \wedge \text{OSD-BC} \notin \text{overlap_l}$
 $(\text{REVERSE} = \text{point_c}(\text{P224})) \vee ((\text{TSD} \notin \text{track_o}) \wedge (\text{USD-BC} \notin \text{subroute_l}) \wedge \dots$
 $\text{R117B(M)} \in \text{route_a}$
(and the remaining preconditions)
 \dots
(the transition postcondition defining a new state)
 $\text{route_s}' = \text{route_s} \cup \{\text{R117B(M)}\}$
 \vdash
(the safety invariant over the new state)
 $\forall ra \in \text{Route} \cdot ra \in \text{route_s}' \wedge ra \notin \text{route_s} \Rightarrow \dots$

Fig. 5. An example of an invariant preservation proof obligation

The verification process consists of generating verification goals to be proved (proof obligations) and attempting to dispatch them. The hypothesis list of the generated proof obligation combines declarations of the model constants and axioms, the current state version of the verified invariant as well as the pre- and post-conditions of the verified transition, while its goal states that the invariant

in question must be preserved in any resulting transition state. Figure 5 shows an abbreviated example of the invariant preservation proof obligation, generated for the route locking state transition (see Fig. 4) and the invariant presented above.

Once all obligations are generated, the built-in symbolic prover attempts to discharge every one of them. Each failed case is reported as a potential error in signalling data. By design, there is no provision to assist with automatic proof.

Table 1 gives SSI data verification summary of the conducted case study. Here transitions are all the state transitions derived from the data, while invariants are formalisations of various safety principles. Non-trivial p.o.'s (proof obligations) is the overall number of proof obligations after ignoring trivially correct ones (e.g., when a transition does not involve the variables mentioned in an invariant). Failed proof obligations indicate potential problems. Note that we do not attempt to distinguish between properties that are too hard to prove and those genuinely incorrect – they are all reported as potential errors. Finally, *Rejected* is the number of error reports rejected as false positives after manual inspection of a generated error report.

Table 1. Verification statistics

Transitions	Invariants	All p.o.'s ^a	Failed p.o.'s	Unique errors	Rejected
2248	9	1451	46	12	0

^aExcluding trivial proof obligations

In addition to the built-in symbolic prover, the framework supports discharging a proof obligation via a number of different external provers. In practice they turned out to be much slower and not as capable overall. Below Table 2 gives the performance times for discharging all the proof obligations of the case study for different external tools. The external prover Why3 [4] is relying on the integrated Alt-Ergo and CVC3 SMT solvers⁴ and eventually arrives at exactly the same result as the built-in prover albeit it takes several hours. Moreover, it turns out to be very sensitive to the available amount of RAM, e.g., restricting RAM to only 8 GB leads to 52 undischarged proof obligations. The built-in SAT solver is unable to discharge a number of proof obligations proven by the symbolic prover but agrees on the set of proven conditions. It is fast and can be used to confirm the result of the symbolic prover in a production setting. Finally, ProB [17] (run in the constraint solver mode) leaves a number of additional proof obligations undischarged and takes rather long time to complete the proof.

The experiments were conducted on Intel I7-4790K @ 4.0 Ghz with 64 Gb RAM. The built-in prover has used a number of custom tactic scripts tuned to the invariants defined. The end result of a verification exercise in SafeCap is an automatically generated verification report in a PDF format. A sample subsection of such a report is given in Fig. 6. A report briefly describes the nature of the failed conditions, points to the problem source code location, and, if applicable, generates a part of the schema diagram with the key elements

⁴ For more details, see <http://alt-ergo.lri.fr/> and <https://cs.nyu.edu/acsys/cvc3/>.

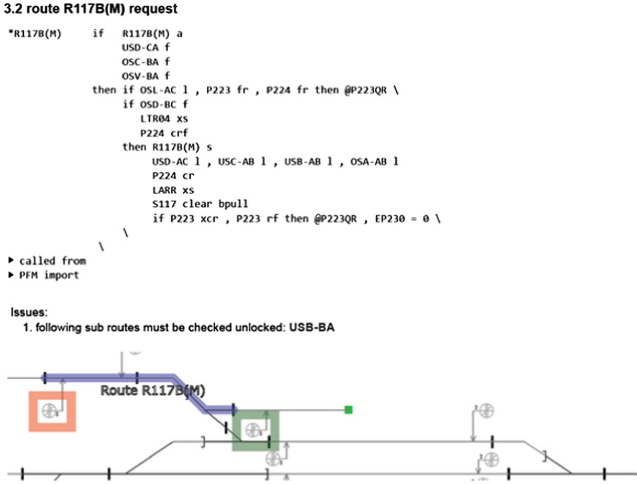


Fig. 6. Verification report sample

related to the failed proof. In the case of the displayed report in Fig.6 the problem is a missing sub-route test and the diagram shows the offending route location.

Table 2. Prover comparison

Prover	Run time	Undischarged proof obligations
Built-in symbolic	12 s	46
Built-in SAT	12 m	207
Why3 + Alt-ergo + CVC3	4 h 15 m	46
ProB	2 h 46 m	101

From the user perspective, the whole process consists of only two steps – providing input data and analysing the generated output. The actual model construction, generation of proof obligations and proving of them: all this happen behind the scenes. Invariant construction is perhaps the most intricate and demanding part of the process that we are going to discuss in our future papers.

5 Conclusions

In this paper we presented the SafeCap approach to verifying railway signalling, in particular, signalling data with program-like representation called SSI. As a number of attempted case studies have demonstrated, the approach proved to scale well. Moreover, although only a subset of safety principles is currently encoded, we are confident that the approach is capable to effectively capture and formalise different formats of signalling data as well as required safety properties.

While SSI is a rather simple notation, it is still liable to state explosion. With all possible modules defining controllers for equipment such as signals and points connected, the state space grows to about 10^{1204} states. Also, one should notice that in industry safety principles are not designed or discussed in terms of train movements – something we commonly see in research papers applying simulation or state exploration techniques – but rather as constraints on signalling rules.

A combination of set theory and first order logic as the underlying mathematical language is the result of experiments over the course of several years. It appears to deliver the optimal combination of a terse, efficient notation for expressing conjectures and safety invariants, while, at the same time, also enabling effective symbolic automated proofs. Two other alternatives we have also explored are pure predicate logic and first order logic with functions and equality.

A custom made symbolic prover might seem a dangerous direction to take for an industry-oriented tool. Indeed, the prover we have developed is not anywhere as powerful or comprehensive as many state-of-the-art provers. However, it has a decisive advantage of being highly customisable via per-invariant tactic scripts. At such a level of fine-tuning it showed to be able to outrun any competition. The prover is also carefully designed to backtrack and terminate in a state facilitating helpful end user feedback.

The approach developed offers immediate industry benefits as it can be used within the existing SSI GDL production processes. The rapid, automated verification that it offers enables errors to be identified earlier in these processes, thereby reducing time consuming and expensive re-work. Furthermore, the Safe-Cap formal approach to verification provides additional assurance over the scenario based testing that is traditionally used in railway signalling. As the safety case underpinning SafeCap develops, and the range of safety properties that it verifies expands, further industry benefits become possible as the manual testing and checking activities are replaced by automated verification by SafeCap.

References

1. Abrial, J.-R.: *The B-book: Assigning Programs to Meanings*. Cambridge University Press, New York (1996)
2. Badeau, F., Amelot, A.: Using B as a high level programming language in an industrial project: Roissy VAL. In: Treharne, H., King, S., Henson, M., Schneider, S. (eds.) *ZB 2005*. LNCS, vol. 3455, pp. 334–354. Springer, Heidelberg (2005). https://doi.org/10.1007/11415787_20
3. Behm, P., Benoit, P., Faivre, A., Meynadier, J.-M.: Météor: a successful application of B in a large project. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) *FM 1999*. LNCS, vol. 1708, pp. 369–387. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48119-2_22
4. Bobot, F., Filliâtre, J.-C., Marché, C., Paskevich, A.: Why3: Shepherd your herd of provers. In: *Proceedings of Boogie 2011*, pp. 53–64 (2011)
5. Busard, S., Cappart, Q., Limbrée, C., Pecheur, C., Schaus, P.: Verification of railway interlocking systems. In: *Proceedings of ESSS 2015*, pp. 19–31 (2015)

6. Cappart, Q., Limbrée, C., Schaus, P., Quilbeuf, J., Traonouez, L.-M. Legay, A.: Verification of interlocking systems using statistical model checking. In: Proceedings of HASE - High Assurance Systems Engineering, pp. 61–68 (2017)
7. Cimatti, A., et al.: Formal verification and validation of ERTMS industrial railway train spacing system. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 378–393. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_29
8. Department for Transport: RAIB review of the railway industry’s investigation of an irregular signal sequence at Milton Keynes (2008). <https://www.gov.uk/raib-reports/review-of-the-railway-industry-s-formal-investigation-of-an-irregular-signal-sequence-at-milton-keynes>
9. Gonschorek, T., Bedau, L., Ortmeier, F.: Automatic model-based verification of railway interlocking systems using model checking. In: Proceedings of ESREL (2018)
10. Huber, M., King, S.: Towards an integrated model checker for railway signalling data. In: Eriksson, L.-H., Lindsay, P.A. (eds.) FME 2002. LNCS, vol. 2391, pp. 204–223. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45614-7_12
11. Iliasov, A., Lopatkin, I., Romanovsky, A.: The safecap platform for modelling railway safety and capacity. In: Bitsch, F., Guiochet, J., Kaâniche, M. (eds.) SAFE-COMP 2013. LNCS, vol. 8153, pp. 130–137. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40793-2_12
12. Iliasov, A., Lopatkin, I., Romanovsky, A.: Practical formal methods in railways - the safecap approach. In: George, L., Vardanega, T. (eds.) Ada-Europe 2014. LNCS, vol. 8454, pp. 177–192. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08311-7_14
13. Iliasov, A., Romanovsky, A.: SafeCap domain language for reasoning about safety and capacity. In: Proceedings of PRDC - Pacific-Rim Dependable Computing, pp. 1–10. IEEE (2012)
14. Iliasov, A., Romanovsky, A.B.: Formal analysis of railway signalling data. In: Proceedings of HASE - High Assurance Systems Engineering, pp. 70–77 (2016)
15. Iliasov, A., Stankaitis, P., Adjepon-Yamoah, D.: Static verification of railway schema and interlocking design data. In: Lecomte, T., Pinger, R., Romanovsky, A. (eds.) RSSRail 2016. LNCS, vol. 9707, pp. 123–133. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33951-1_9
16. James, P.: Verification of solid state interlocking programs. In: Counsell, S., Núñez, M. (eds.) SEFM 2013. LNCS, vol. 8368, pp. 253–268. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-05032-4_19
17. Leuschel, M., Butler, M.: ProB: A model checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 855–874. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45236-2_46
18. Limbrée, C., Cappart, Q., Pecheur, C., Tonetta, S.: Verification of railway interlocking - compositional approach with OCRA. In: Lecomte, T., Pinger, R., Romanovsky, A. (eds.) RSSRail 2016. LNCS, vol. 9707, pp. 134–149. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33951-1_10
19. Macedo, H.D., Fantechi, A., Haxthausen, A.E.: Compositional verification of multi-station interlocking systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9953, pp. 279–293. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_20
20. Morley, M.J.: Safety Assurance in Interlocking Design. PhD thesis, University of Edinburgh (1996)



Deriving and Formalising Safety and Security Requirements for Control Systems

Elena Troubitsyna^{1,2} and Inna Vistbakka²(✉)

¹ KTH, Stockholm, Sweden
elenatro@kth.se

² Åbo Akademi University, Turku, Finland
inna.vistbakka@abo.fi

Abstract. Safety-critical control systems become increasingly open and interconnected. However, there is still a lack of the techniques that enable an integrated analysis of safety and security requirements. In this paper, we propose an approach that allows the designers to derive and formalise safety and security requirements in a structured systematic way. To elicit both types of the requirements, we adapt and integrate traditional safety and security analysis techniques. To formally specify and verify them, we rely on Event-B framework. The framework allows us to develop a complex specification of system behaviour in presence of both accidental faults and security attacks and analyse mutual interdependencies between safety and security requirements.

Keywords: Formal modelling · Safety analysis · Data flow
Event-B · Refinement · Safety-critical systems · Security

1 Introduction

Modern control systems increasingly rely on networking technologies. While offering greater flexibility and possibility to provide richer functionality, the increased system openness also introduces security threats. Security vulnerabilities can be exploited to undermine safety, e.g., by tampering with sensor data or hijacking the controlling functions. To ensure safety, we have to integrate the mechanisms for coping with both accidental component failures and malicious security attacks. However, since traditionally safety and security engineering have been considered to be two different disciplines, there is a lack of approaches supporting an integrated analysis of safety and security requirements.

In this paper, we propose an integrated approach to deriving safety and security requirements by applying safety analysis to the systems data flow. To analyse the intricate interdependencies between the requirements, we rely on formal modelling in Event-B [1]. Event-B is a rigorous approach to correct-by-construction system development by refinement. Development typically starts

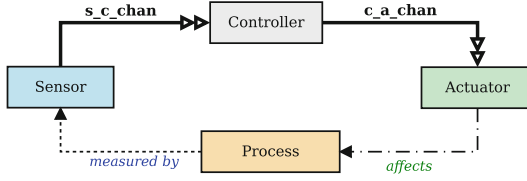


Fig. 1. Generic architecture of a control system

from an abstract specification that models the most essential system functionality. In the refinement process, the abstract model is transformed into a detailed specification. While refining the system model, we can explicitly represent both nominal and failure behaviour of the system components as well as define the mechanisms for error detection and recovery. Moreover, we can also explicitly represent the effect of security vulnerabilities such as tampering, spoofing and denial-of-service attacks and analyse their impact on system safety.

The formal specification mimics the data flow. The security failures are modelled by their effect on the system – altering or blocking messages sent over the communication channels. Formal modelling allows us also to understand the limits of programmable means of achieving safety and suggest the corresponding modification of the system architecture. The proposed approach is illustrated by a case study – a battery charging system.

We believe that the proposed approach facilitates an integration of the security consideration into the safety-driven design of control systems. It allows us to capture the dynamic nature of safety and security interplay, i.e., analyse the impact of deploying the security mechanisms on safety assurance and vice versa.

2 Safety-Critical Networked Control Systems

Currently safety-critical systems – the systems whose failures might cause loss of human lives or environmental damage [10] – are increasingly relying on networked technologies in their functioning. In this section, we analyse a generic architecture of a networked control system and discuss the problem of assuring system safety in a security-aware way.

Figure 1 depicts a generic architecture of a control system. The aim of the system is to control a certain physical process. Lets assume that the state of the process is characterised by some physical value p_real . While the system is operational, the physical value should be maintained within certain safe boundaries, i.e., we can define the following safety invariant over the operational states:

$$Safety_{op} = p_real \in [min_safe_threshold, max_safe_threshold]$$

In this paper, we focus on the analysis of failsafe systems, i.e., the systems that can be put into a safe non-operational state if safety in an operational mode can no longer be maintained. Formally, it can be described as follows:

$$Safety_{nonop} = shutdown = TRUE$$

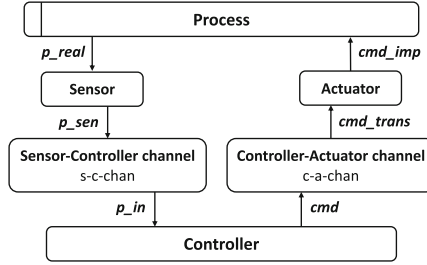


Fig. 2. A data flow diagram

Overall system safety is a disjunction of two invariants:

$$Safety = Safety_{op} \vee Safety_{nonop}$$

The value of the physical parameter is measured by a (physical or logical) sensor. The sensor has a certain imprecision or might experience failures, i.e., the readings p_{sen} produced by the sensor are in general do not exactly match p_{real} .

Since we consider a networked control system, sensing can be remote, i.e., the sensor transmits its readings to the controller via the channel s_c_chan . In general, due to random channel failures or malicious attacks, the transmission is unreliable. Therefore, the value p_{in} received by the controller can also be different from the sensed value.

Upon receiving p_{in} , the controller performs error detection, i.e., evaluates veracity of p_{in} , and either adopts it as the current estimate of the process state or ignores and relies on other alternative means to evaluate the state of the process. Then the controller computes to which state the actuator should be put to achieve the desired functional behaviour and guarantee safety and issues the corresponding command cmd .

The commands issued by the controller are sent to the actuator over the channel c_a_chan . Similarly, to s_c_chan , the channel c_a_chan is considered to be unreliable (due to the random faults or attacks). Hence, the command cmd_trans received by the actuator might differ from the command cmd issued by the controller. Finally, the actuator itself might fail and hence cmd_imp – the impact produced by the actuator on the process – might also deviate from the one associated with the command cmd .

The system behaviour is cyclic. At each cycle, the system goes through the sequence of steps as represented by a data flow diagram (DFD) shown in Fig. 2. DFDs give a graphical representation of the flow of information for a given system [3]. They are often used as a basis for the security analysis. Since at each cycle the overall control system alternates between the control actions and physical process reaction, the physical process constitutes the source and the sink of the data flow. Rectangles depict the system components and the communication channels that consume and produce data attributes shown by the corresponding incoming/outgoing arrows.

To analyse the impact of the various failure modes and security attacks, we need to rely on a systematic analysis process that would allow us to consider safety and security in an integrated way. Traditionally, security analysis is data flow oriented, while safety analysis often focuses on determining the impact of component failures on safety-related functions.

In this paper, we propose to combine these views by integrating HAZOP [12] and data flow analysis as we explain next.

3 Deriving Safety and Security Requirements

To analyse safety-security interplay, we adopt a systems theoretical model [25] shown in Fig. 3. It guides us in defining the main goals of our integrated analysis. Namely, we should ensure that

- G1:** The feedback allows the controller to build a sufficiently accurate process model.
- G2:** The intended control actions ensure safety.
- G3:** An implementation of control actions preserves safety.

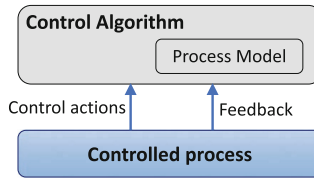


Fig. 3. Abstract system representation

Let us informally analyse the implications that the high-level safety goals have on the data flow of the system given in Fig. 2. Essentially, the goal **G1** postulates that the part of the data flow that produces the input to the controller either does not alter the information about the process state or the alterations are detectable. The goal **G2** implies two subgoals. Firstly, the controller's estimate of the current process state is sufficiently accurate, i.e., either the feedback is sufficiently precise or the deviations in the feedback are detected and alternative means to evaluate process state are employed. Secondly, the computed control commands guarantee safety, i.e., safety can be maintained in a programmable way. Finally, **G3** implies that the intended control actions are properly implemented or alternative non-programmable actions are enabled to enforce safety.

The informal analysis above shows that we need to establish a systematic way to study the causes of deviations in the systems data flow. To achieve this, we propose to tailor the well-known safety analysis technique HAZOP [12] and integrate it with the data flow analysis.

NO	Data is missing
MORE/LESS	Value of data increased/decreased
REVERSE/INSTEAD	Substitution of intended data value to logical opposite
EARLY/LATE	Data available before/after intended

Fig. 4. Interpretation of standard HAZOP guidewords

HAZOP – HAZard and OPerability Analysis [12] systematically identifies the possible causes and consequences of deviations within the system to analyse their impact on safety. The analysis is performed using a set of guidewords and attributes. The original set of guidewords includes *no*, *more*, *less*, *as well as*, *part of*, *reverse*, *other than*, *early*, *late*, *before* and *after*. The guidewords are applied to the system attributes, which allows a team of experts to identify the deviations that might affect safety. HAZOP has been extensively used in safety-critical domain and extended to specifically target security vulnerabilities [24]. However, to the best of our knowledge, an integrated analysis of safety and security interdependencies with DFDs and HAZOP has not yet been attempted.

To facilitate an integrated derivation of safety and security requirements, we need to rely on a well-structured description of the system and a systematic procedure for selecting the attributes to be analysed. To describe the system and identify the attributes, we propose to use DFDs. Indeed, DFD clearly defines the relationships between the system components and the data that they produce and consume. The analysed attributes are the data attributes defined in DFD. To achieve a seamless integration of DFD and HAZOP, we propose an interpretation of standard HAZOP guidewords presented in Fig. 4.

The majority of the guidewords are self-explanatory. The guidewords *REVERSE/INSTEAD* represent the deviations over the discrete data types. If they are applied to booleans then *REVERSE* would represent replacements of *TRUE* by *FALSE* and vice versa; if they are applied to sets then *INSTEAD* would represent replacing one constant of the set with another (e.g., *ON* by *OFF*).

HAZOP performed over the data attributes defined in DFD provides us with a structured methodology to analyse causes and consequences of the possible deviations of the data attributes. Per se, it enables a compositional integrated analysis of the impact of accidental and malicious failures on system safety. The methodology, that we propose, consists of the following steps:

- Select a data attribute and identify the component(s) responsible for producing or modifying the selected attribute
- Apply relevant guidewords and identify possible deviations
- For each deviation identify the causes by analysing failure modes (accidental and malicious) of the associated components
- Analyse the impact on system safety.

The cause-consequence analysis performed over the data attributes using HAZOP results in building cause-consequence trees for each data attribute in

Attribute	Guideword	Consequence	Causes
p_sen	MORE/	the value p_sen is greater (or smaller)	Sensor stuck at (high/low)
	LESS	than the value p_real	Loss of sensor precision
	LATE	Delay in sensing process state	
Attribute	Guideword	Consequence	Causes
cmd_trans	NO/	No command received by the actuator	DOS attack on c_a_chan Omission channel failure
	INSTEAD	Actuator receives opposite command	Spoofing controller identity Tampering with channel data
	LATE	Delay in transmission	Network congestion

Fig. 5. Analysing the attribute p_sen and cmd_trans

the data flow. It allows us to systematically derive both safety and security constraints that should be fulfilled to guarantee safety. Figure 5 presents a few examples of applying the proposed approach.

For the deviation “The value p_sen is greater (or smaller) than the value p_real ”, we can define the requirements addressing the causes of this deviations as follows: “Controlling software should check reasonableness of received input data and use indirect data or calculated predicted value if the received data is outside of reasonable range”.

For the deviation “No command received by the actuator” it is clear that we need to rely on a non-programmable means to achieve safety. Hence, we will need to augment system architecture with a reliable non-programmable channel that shuts down the system to achieve safety. For the deviation “Actuator receives opposite command” we need to install a secure gateway for the channel c_a_chan to prevent the attacks on it.

The proposed methodology allows us to derive safety and security requirements for controlling software and overall system architecture. However, often such requirements are mutually interdependent. For instance, while introducing a secure gateway for the controller-actuator channel, we also introduce a delay in transmitting the control commands. Such a delay should be taken into account while calculating the state of the actuator.

4 Modelling and Refinement in Event-B

Formal modelling and verification allow us to systematically analyse the interdependencies between the requirements and provides us with the mathematical evidences – proofs – of system safety. Event-B [1] is a state-based framework that promotes the correct-by-construction approach to system development and formal verification by theorem proving.

In Event-B, a system model is specified using the notion of an *abstract state machine* [1]. An abstract state machine encapsulates the model state, represented as a collection of variables, and defines operations on the state, i.e., it describes

the dynamic behaviour of a modelled system. A machine also has an accompanying component, called *context*, which includes user-defined sets, constants and their properties given as model axioms.

The dynamic behaviour of the system is defined by a set of atomic *events*. Generally, an event has the following form:

$$e \hat{=} \mathbf{any} \ a \ \mathbf{where} \ G_e \ \mathbf{then} \ R_e \ \mathbf{end},$$

where e is the event's name, a is the list of local variables, the *guard* G_e is a predicate over the local variables of the event and the state variables of the system. The body of an event is defined by a *multiple* (possibly nondeterministic) assignment over the system variables. The guard defines the conditions under which the event is *enabled*, i.e., its body can be executed. If several events are enabled at the same time, any of them can be chosen for execution nondeterministically.

The consistency of Event-B models, i.e., verification of well-formedness and invariant preservation as well as correctness of refinement steps, is demonstrated by discharging a number of verification conditions – proof obligations. The Rodin platform [14] provides an automated support for formal modelling and verification in Event-B. In particular, it automatically generates the required proof obligations and attempts to discharge (prove) them automatically. It also provides a support for an interactive proving.

Event-B employs a top-down refinement-based approach to system development. Development typically starts from an abstract specification that nondeterministically models the most essential functional requirements. In a sequence of refinement steps, we gradually reduce nondeterminism and introduce detailed design decisions.

In the next section, we demonstrate how to use Event-B framework to formalise the derived safety and security requirements by refinement.

5 Refinement of Safety-Critical Networked Systems

Let us start by describing the strategy of the refinement process. In our refinement-based development, we aim at gradually unfolding the system data flow according to DFD. Our initial specification non-deterministically models the behaviour of the controlled physical process and abstractly represents the data flow by specifying the input and output of the controller and the applied control actions.

The overall data flow is modelled as a sequence of phases – at each phase a component produces or modifies a corresponding data attribute (its output in DFD). The refinement steps capture these phases and model the impact of nominal and off-nominal behaviour on the data attributes. We rely on the proposed HAZOP analysis to identify the possible deviations. The refined model also introduces the specifications of the error detection and recovery procedures that allow us to eventually prove the desired safety invariant, i.e., to demonstrate that the system safety can be guaranteed in the presence of the identified accidental faults and security attacks – the causes of the deviations of data attributes.

```

Machine CS_Abs
Variables phase, cmd, p_in, p_real, failsafe
Invariants  $phase \in \text{PHASES} \wedge cmd \in \text{CMD} \wedge p\_in \in \mathbb{N} \wedge p\_real \in \mathbb{N} \wedge failsafe \in \text{BOOL} \wedge \dots$ 
Events ...
Process  $\hat{=}$ 
  when  $phase=ENV \wedge failsafe=FALSE$ 
  then  $p\_real : \in \mathbb{N} \parallel phase:=EST$  end
Inp_estimation  $\hat{=}$ 
  when  $phase=EST \wedge failsafe=FALSE$ 
  then  $p\_in : \in \mathbb{N} \parallel phase:=CONT$  end
CONT_act  $\hat{=}$ 
  when  $phase=CONT \wedge failsafe=FALSE$ 
  then  $cmd : \in \{ON, OFF\} \parallel phase:=OUTPUT$  end
...

```

Fig. 6. A generic structure of the abstract specification

The general structure of the abstract specification is shown in Fig. 6. The specification abstractly models the control cycle. In the phase *ENV*, the process non-deterministically changes its state, i.e., produces *p_real*. In the phase *EST*, we model the part of the data flow that is responsible for sensing the state of the process and transmitting it to the controller, i.e., producing *p_in*. In the phase *CONT*, the controller computes the control action *cmd*, which is then applied (probably altered) to the process in the consequence phase *OUTPUT*.

Since in our specification we focus on defining the programmable means of ensuring safety, the flag *failsafe* is set to *FALSE*. An event *SHUTDOWN* can change the flag to *TRUE* to model that non-programmable means are deployed to ensure safety.

Our consequent refinement steps focus on unfolding the phases of the data flow and making the specification progressively more deterministic and detailed. To facilitate an identification of all possible deviations of the corresponding data attributes, we use HAZOP, as described in the previous section.

We propose the following steps to represent the requirements derived via HAZOP in Event-B:

- Select a data attribute in the data flow and apply one of the relevant guidewords, to define the deviation and the causes
- In the formal specification, perform a refinement step that introduces the variable representing the attribute and events modelling the deviation and corresponding error detection and recovery mechanisms
- By defining the corresponding invariants, establish the relationships between the more abstract representation of the data flow and its refined (unfolded) specification
- Iteratively repeat the process for the other guidewords and all other attributes.

For instance, the analysis of the attribute *p_sen*, shown in Fig. 5 defines the consequence “the value of *p_sen* is smaller than the value *p_real*” and a cause of this “a sensor failure stuck at low”. In our formal specification, such a behaviour

can be modelled by introducing the variables p_sen and $sensor\ failure$. An event modelling a non-deterministic occurrence of sensor failures can change the value of the variable $sensor\ failure$ to model failure occurrence. Correspondingly, the event modelling the sensor reading phase, would result in assigning p_sen the value representing the saturated low. In case of cause “sensor lost precision”, the value p_sen would be assigned a value smaller or greater than p_real .

Correspondingly, in the specification of the controller behaviour, we will introduce the variable modelling the predicted state of the process. Moreover, we introduce the events modelling error detection (checking validity of the input) and events modelling error recovery (using alternative means of assessing the state of the process).

To model the consequence “Data is missing”, we add a constant NIL to the definition of the types of the corresponding variables. An occurrence of such a deviation, i.e., caused by the denial-of-service (DOS) attack on the channel transmitting the corresponding data, is represented by assigning the variable modelling the corresponding data attribute the value NIL . This allows us to formally model that the corresponding phase of the data flow has not produced the expected data.

To demonstrate an application of the proposed methodology, in the next section, we present a case study – a battery charging system.

6 Case Study: Battery Charging System

A battery charging system [16] controls charging of a battery of an electric car. Figure 7 shows the main system components: the battery, the battery management system, the Controller Area Network (CAN bus), the charging station (with the associated charging interface and the external charging unit). When the charging station detects that an electrical vehicle got connected to its external charging unit, it starts the charging procedure. While charging, the battery management system (BMS) – the controlling software of the system, monitors the measurements received from the battery and issues the signal to the charging station to continue or stop charging. BMS and the charging station communicate via the CAN bus. The system behaviour is cyclic: at each cycle the charging station receives the command from BMS to continue or stop charging. Correspondingly, it either continues or stops to supply the energy to the car battery.

The main hazard associated with the system is overcharging of the car’s battery, which might result in an explosion. Hence, the top-level safety goal of the system is to keep a battery level bl_real within the safe boundaries, i.e., the system safety goal can be formulated as follows: $0 \leq bl_real \leq bl_max_crit$, where 0 and bl_max_crit denote the lowest and highest boundaries.

The battery charging system is a typical example of a *control system* discussed in Sect. 2. Indeed, the BMS acts as a *controller*, the charging station (with its associated charger unit) – as an *actuator* and the battery unit as the *process* that the system controls. The battery level parameter can be directly measured by the *sensor* of BMS or computed on the basis of the alternative

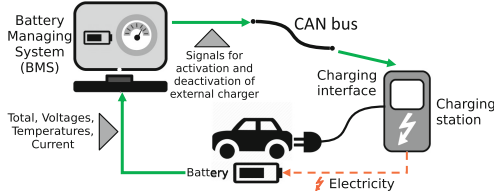


Fig. 7. Architecture of battery charging system

measurements obtained from the battery. At each cycle, BMS assesses the value of the battery level and sends the corresponding control command.

The charging station and in-car CAN bus are linked by the corresponding communication channel that could be possibly vulnerable to the security attacks. In particular, the attacker can use the in-car charging interface as an entry point by compromising the external charger interface or tampering with the communication between the interfaces to inject a malicious content into the CAN bus. Therefore, while reasoning about the behaviour of such a system, we should also reason about the impact of security threats on its safety.

The analysis presented in Sect. 3 shows that safety cannot be guaranteed by the programmable means when the controller-actuator channel is attacked or the actuator fails to execute the required control actions. Thus, the battery charging system should include an additional hardware component – a non-programmable channel – that should be installed in the car to break the charging circuit if the battery charge level becomes dangerously high. Such a non-programmable switch can put the system in the failsafe state to guarantee safety.

Next we present an abstract Event-B specification of our case study that focuses on modelling programmable means.

Abstract Specification. We follow the methodology outlined in Sect. 5, i.e., in the initial Event-B specification `BatteryCharging_Abs`, we introduce an abstract representation of the control cycle.

We introduce the variable *phase*, where $phase \in PHASES$. Here the set $PHASES = \{BAT, EST, BMS, TRANSM, CHARGST\}$. The variable *phase* is used to enforce the pre-defined cyclic execution of the data flow:

Battery \rightarrow BMS_estim \rightarrow BMS_act \rightarrow CAN_bus \rightarrow ChargStation \rightarrow Battery \rightarrow ...

The event `Battery` (see Fig. 8) models the changes of the battery parameter *bl_real* while charging. The event `BMS_estim` models the BMS estimation of this parameter (that is defined by *bl* variable). The event `BMS_act` specifies the BMS actions (i.e., sending the signal to continue or stop charging) and the event `CAN_bus` models transmission of the corresponding command to the charging station. Finally, the event `ChargStation` models the required actions from the charging station upon receiving the signal from BMS.

In addition to modelling the control cycle, we also define the event `Connect` that represents the beginning of the charging procedure (i.e., when a vehicle

```

Machine BatteryCharging_Abs Sees BatteryCharging_c0
Variables phase, signal, bl, bl_real, status, failsafe
Invariants phase ∈ PHASES ∧ signal ∈ SIGNALS ∧ status ∈ STATUSES ∧ bl ∈ ℕ ∧
           bl_real ∈ ℕ ∧ failsafe ∈ BOOL ∧ failsafe=TRUE ⇒ status=IDLE ∧ ...
Events ...
Battery ≐
  when phase=BAT ∧ status=CHARGING ∧ failsafe=FALSE
  then bl_real := ∈ ℕ || phase:=EST end
BMS_estim ≐
  when phase=EST ∧ failsafe=FALSE
  then bl := ∈ ℕ || phase:=BMS end
BMS_act ≐
  when phase=BMS ∧ failsafe=FALSE
  then signal := {CONT, STOP} || phase:=TRANSM end

```

Fig. 8. The machine `BatteryCharging_Abs`

connects to the charging station) and the event `ChargingComplete` that models its completion.

Let us note that the BMS estimate of the battery level value bl is not necessarily equal to the measurements produced by the battery sensor that monitors bl_real . In our model, bl is an abstraction representing the input to BMS, i.e., it accounts for any possible deviations (e.g., due to the sensor imprecision or failure) from bl_real , as we explained in Sect. 5. The value bl_real is updated in the event `Battery`, while bl is estimated in the event `BMS_estim`.

Finally, the transition into a safe but non-operational state is modelled by the event `FailSafe`. An execution of this event results in an immediate aborting of charging, which is modelled by assigning the variable $status$ the value `IDLE`.

Refinement Steps. In a sequence of model refinements, we aim at unfolding the detailed data flow. Our first refinement step (Fig. 9) aims at introducing a detailed specification of the BMS logic. We define the control algorithm, i.e., model the behaviour of the controller. The controller calculates the commands to be send to the charging station using the current estimate of the battery level. Moreover, at this refinement step, we also elaborate on the dynamics of the controlled process, i.e., define the changes in the real battery level bl_real and model different cases of the behaviour of the charging station.

At each control cycle, the controller receives the current estimate of the battery level from the sensor. The controller checks whether the battery is still not fully charged and it is safe to continue to charge it or charging should be stopped. The decision to continue to charge can be made only if the controller verifies that the battery level at the end of the next cycle will still be in the safe range $[0..bl_max_crit]$. The event `BMS_estim` modelling estimation of the battery parameter made by the BMS is refined. We perform HAZOP and model the impact of sensor imprecision of the data flow. Consequently, the variable bl gets any value from the range $(bl_real - bl_delta .. bl_real + bl_delta)$, where bl_delta is the maximal imprecision value for the battery sensor introduced as a constant in the model context.

We also refine the abstract event `BMS_act` to represent different alternatives. The first alternative defines a reaction to the monitored parameter exceeding bl_max . The second alternative models continuing the charge when the monitored parameter is in the completely safe range $[0..bl_max)$. Note that the monitored value `bl` that BMS relies on here is different from the actual value of the physical process (bl_real) updated by the event `Battery`.

We can formulate correctness of the BMS logic by the following invariants:

$$\begin{aligned} phase = TRANSM \wedge bl \geq bl_max &\Rightarrow signal = STOP \\ phase = TRANSM \wedge bl < bl_max &\Rightarrow signal = CONT. \end{aligned}$$

The invariants postulate that the BMS issues the signal to stop when the parameter bl is approaching the critically high value (bl_max_crit), and vice versa. To give the system a time to react, BMS sends the stopping command to the station whenever the value bl breaches the predefined value bl_max .

```

Machine BatteryCharging_M1 refines BatteryCharging_Abs Sees BatteryCharging_c1
Variables phase, signal, bl, bl_real, status, failsafe
Invariants ...  $\wedge (phase = TRANSM \wedge bl \geq bl\_max \Rightarrow signal = STOP) \wedge$ 
                $(phase = TRANSM \wedge bl < bl\_max \Rightarrow signal = CONT) \wedge \dots$ 
Events...
Battery refines Battery
  when phase=BAT  $\wedge$  status = CHARGING  $\wedge$  failsafe=FALSE
  then bl_real := bl_fnc(bl_real) || phase := EST end
BMS_estimation  $\hat{=}$  refines BMS_estim
  when phase=EST  $\wedge$  failsafe = FALSE
  then bl := bl_real - bl_delta .. bl_real + bl_delta || phase := BMS end
BMS_cont  $\hat{=}$  refines BMS_act
  when phase=BMS  $\wedge$  failsafe=FALSE  $\wedge$  bl < bl_max
  then signal:=CONT || phase := TRANSM end
ChargStation  $\hat{=}$  refines ChargStation
  any sg
  where phase=CHARGST  $\wedge$  sg  $\in$  {CONT, STOP}  $\wedge$  failsafe=FALSE
  then status : | (sg = CONT  $\Rightarrow$  status' = CHARGING)  $\vee$ 
               (sg = STOP  $\Rightarrow$  status' = CHARGED)
               phase := BAT end
...

```

Fig. 9. The machine `BatteryCharging_M1`

Moreover, in this refinement step, we elaborate on the behaviour of the charging station. Upon receiving the command from BMS, the charging station either deactivates the charging unit or continues to supply energy to the battery. Such a behaviour is defined by the refined event `ChargStation` (see Fig. 9).

To represent the fact that the charging station reads the signal from the CAN bus, which might be attacked, at the next refinement step we add several new events and new variables into the refined system specification (see Fig. 10). Firstly, we introduce a new event `Attack` to model a possible attack on the system. The attack can happen anytime while transmitting the signal to the charging interface. The variable $attack \in \text{BOOL}$ indicates whether the system

is under attack. If the event **Attack** is triggered, the value of *attack* becomes *TRUE*, otherwise it equals to *FALSE*.

Secondly, we introduce a new event **SecurityGateway** and a new variable *charg_in* that specifies the input buffer of the charging interface. It might obtain values from the set of possible signals, i.e., $charg_in \in SIGNALS$. If no attack happens, then signal transmission results in copying the signal from one-place output buffer of the CAN bus (represented by *bus_out* variable) to the input buffer *charg_in* of the charging interface. If a security failure occurred (e.g., the system has been under attack) then the output signal would differ from the sent signal. The DOS attack (or in general channel unavailability) results in no values being transmitted over the channel. For the sake of simplicity, we model it by introducing the *DOS* constant that the input buffer of the charging interface will get in this case. However, we also could have modelled it by defining a behaviour of a watchdog process triggering the timeout signal.

```

Machine BatteryCharging_M2 refines BatteryCharging_M1 Sees BatteryCharging_c2
Variables phase, signal, bl, bl_real, status, failsafe, attack, bus_out, charg_in
Invariants ... (phase = CHARG  $\wedge$  bl  $\geq$  bl_max  $\Rightarrow$  bus_out=STOP)  $\wedge$ 
                (phase = CHARG  $\wedge$  bl < bl_max  $\Rightarrow$  bus_out=CONT)  $\wedge$ 
                (attack = FALSE  $\wedge$  phase = CHARG  $\wedge$  bus_out=STOP  $\Rightarrow$  signal=STOP)  $\wedge$ 
                (attack = FALSE  $\wedge$  phase = CHARG  $\wedge$  bus_out=CONT  $\Rightarrow$  signal=CONT)  $\wedge$ 
                (phase=BAT  $\wedge$  status=CHARGING  $\Rightarrow$  bl_real  $\leq$  bl_max + bl_delta)  $\wedge$ 
                (bl_real  $\in$  0 .. bl_max_crit)  $\wedge$  ...

Events ...
SecurityGateway  $\hat{=}$ 
  when phase=CHARG  $\wedge$  failsafe=FALSE  $\wedge$  charg_in=S0
  then charg_in : | (attack=FALSE  $\Rightarrow$  charg_in'=bus_out)  $\vee$ 
                (attack=TRUE  $\Rightarrow$  charg_in'=DOS)
  end
ChargStation_stop  $\hat{=}$  refines ChargStation
  when phase=CHARG  $\wedge$  charg_in=STOP  $\wedge$  failsafe=FALSE
  with sg=STOP
  then status := CHARGED || phase := BAT || charg_in:=S0
  end
...

```

Fig. 10. The machine BatteryCharging_M2

As a result of this refinement step, we arrive at a sufficiently detailed specification to define and prove the desired safety invariant:

$$bl_real \in 0..bl_max_crit.$$

7 Related Work and Conclusions

Related Work. The problem of safety and security interactions has recently received a significant research attention [22, 23]. It has been recognised that there is a clear need for the approaches facilitating an integrated analysis of safety and security [15, 22, 23, 25]. This issue has been addressed by several techniques

demonstrating how to adapt traditional safety techniques like FMECA and fault trees to perform a security-informed safety analysis [6, 15]. The techniques aim at providing the engineers with a structured way to discover and analyse security vulnerabilities that have safety implications. Since the use of such techniques facilitates a systematic analysis of failure modes and results in discovering safety and security requirements, these approaches provide a valuable input for our modelling.

There are several works that address formal analysis of safety and security requirements interactions [2, 8]. Majority of that works demonstrate how to find conflicts between them. In our approach, we treat the problem of safety-security interplay at a more detailed level, i.e., we analyse the system architecture, investigate the impact of security failures on safe implementation of system functions and demonstrate how fault tolerance required for safety leads to discovery of additional security requirements. Such an approach allows us to analyse the dynamic nature of safety-security interactions.

The distributed MILS approach [4, 5] employs a number of advanced modelling techniques to create a platform for a formal architectural analysis of safety and security. The approach supports a powerful analysis of the properties of the data flow using model checking and facilitates derivation of security contracts. Since our approach enables incremental construction of complex distributed architectures, it would be interesting to combine these techniques to support an integrated safety-security analysis throughout the entire formal model-based system development.

The work presented in this paper, adopts the approach to explicit modelling of failure behaviour in formal specification proposed in [9, 11, 17, 21]. Such an approach allows us to analyse the impact of failures on the system safety defined by its safety invariant. By relying on the extensions of Event-B proposed in [7, 18–20], we can also model the impact of accidental and malicious faults on system reliability and real-time behaviour.

Conclusions. In this paper, we have proposed a novel integrated approach to deriving and formalising safety and security requirements. To derive the requirements, we have combined data flow analysis, traditionally used in security domain, and HAZOP – a widely used safety analysis technique. HAZOP applied to data flow allowed us to identify possible deviations that are caused by accidental failures and security attacks.

To formalise complex requirements derived through an integrated analysis, we relied on modelling and refinement in Event-B. Our approach supported an analysis of interdependencies between the architectural patterns and mechanisms required for safety and security assurance. Instead of separating the safety and security requirements, we have considered them as the interdependent constraints required for achieving safety of a networked control system.

As a future work, we are planning to build a domain-specific framework that would allow us to analyse and visualise the impact of accidental failures and security attacks on the overall system architecture. Moreover, we will also

demonstrate how to generate an integrated safety case from formal models by extending the approach proposed in [13].

References

1. Abrial, J.R.: *Modeling in Event-B*. Cambridge University Press, New York (2010)
2. Brunel, J., Rioux, L., Paul, S., Fauconney, A., Vallée, F.: Formal safety and security assessment of an avionic architecture with alloy. In: *ESSS 2014*, pp. 8–19 (2014)
3. Bruza, P., van der Weide, T.P.: *The Semantics of Data Flow Diagrams*. Technical report 89-16, University of Nijmegen, The Netherlands (1989)
4. Bytschkow, D., Quilbeuf, J., Igna, G., Ruess, H.: Distributed MILS architectural approach for secure smart grids. In: Cuellar, J. (ed.) *SmartGridSec 2014*. LNCS, vol. 8448, pp. 16–29. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10329-7_2
5. Cimatti, A., DeLong, R., Marcantonio, D., Tonetta, S.: Combining MILS with contract-based design for safety and security requirements. In: Koornneef, F., van Gulijk, C. (eds.) *SAFECOMP 2015*. LNCS, vol. 9338, pp. 264–276. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24249-1_23
6. Fovino, I.N., Masera, M., Cian, A.D.: Integrating cyber attacks within fault trees. *Rel. Eng. Sys. Saf.* **94**(9), 1394–1402 (2009)
7. Iliasov, A., Romanovsky, A., Laibinis, L., Troubitsyna, E., Latvala, T.: Augmenting Event-B modelling with real-time verification. In: *Proceedings of the FormSERA 2012*, pp. 51–57. IEEE (2012)
8. Kriaa, S., Bouissou, M., Colin, F., Halgand, Y., Pietre-Cambacedes, L.: Safety and security interactions modeling using the bdmp formalism: case study of a pipeline. In: Bondavalli, A., Di Giandomenico, F. (eds.) *SAFECOMP 2014*. LNCS, vol. 8666, pp. 326–341. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10506-2_22
9. Laibinis, L., Troubitsyna, E.: Refinement of fault tolerant control systems in B. In: Heisel, M., Liggesmeyer, P., Wittmann, S. (eds.) *SAFECOMP 2004*. LNCS, vol. 3219, pp. 254–268. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30138-7_22
10. Leveson, N.G.: *Safeware: System Safety and Computers*. Addison-Wesley, Boston (1995)
11. Lopatkin, I., Iliasov, A., Romanovsky, A., Prokhorova, Y., Troubitsyna, E.: Patterns for Representing FMEA in formal specification of control systems. In: *HASE 2011*, Boca Raton, FL, USA, pp. 146–151. IEEE Computer Society (2011)
12. Ministry of Defence: Interim Defence Standard 00–58/1: Hazop Studies on Systems Containing Programmable Electronics. In: Directorate of Standardization (1994)
13. Prokhorova, Y., Laibinis, L., Troubitsyna, E.: Facilitating construction of safety cases from formal models in Event-B. *Inf. Softw. Technol.* **60**, 51–76 (2015)
14. Rodin: Event-B platform. <http://www.event-b.org/>
15. Schmittner, C., Gruber, T., Puschner, P., Schoitsch, E.: Security application of failure mode and effect analysis (FMEA). In: Bondavalli, A., Di Giandomenico, F. (eds.) *SAFECOMP 2014*. LNCS, vol. 8666, pp. 310–325. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10506-2_21
16. Schmittner, C., Ma, Z., Puschner, P.: Limitation and improvement of STPA-sec for safety and security co-analysis. In: Skavhaug, A., Guiochet, J., Schoitsch, E., Bitsch, F. (eds.) *SAFECOMP 2016*. LNCS, vol. 9923, pp. 195–209. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45480-1_16

17. Sere, K., Troubitsyna, E.: Safety analysis in formal specification. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) FM 1999. LNCS, vol. 1709, pp. 1564–1583. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48118-4_33
18. Tarasyuk, A., Pereverzeva, I., Troubitsyna, E., Latvala, T., Nummila, L.: Formal development and assessment of a reconfigurable on-board satellite system. In: Ortmeier, F., Daniel, P. (eds.) SAFECOMP 2012. LNCS, vol. 7612, pp. 210–222. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33678-2_18
19. Tarasyuk, A., Troubitsyna, E., Laibinis, L.: Towards probabilistic modelling in Event-B. In: Méry, D., Merz, S. (eds.) IFM 2010. LNCS, vol. 6396, pp. 275–289. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16265-7_20
20. Tarasyuk, A., Troubitsyna, E., Laibinis, L.: Integrating stochastic reasoning into event-B development. *Form. Asp. Comput.* **27**(1), 53–77 (2015)
21. Troubitsyna, E.: Stepwise Development of Dependable Systems. Technical report (2000)
22. Troubitsyna, E., Laibinis, L., Pereverzeva, I., Kuismin, T., Ilic, D., Latvala, T.: Towards security-explicit formal modelling of safety-critical systems. In: Skavhaug, A., Guiochet, J., Bitsch, F. (eds.) SAFECOMP 2016. LNCS, vol. 9922, pp. 213–225. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45477-1_17
23. Vistbakka, I., Troubitsyna, E., Kuismin, T., Latvala, T.: Co-engineering safety and security in industrial control systems: a formal outlook. In: Romanovsky, A., Troubitsyna, E.A. (eds.) SERENE 2017. LNCS, vol. 10479, pp. 96–114. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65948-0_7
24. Winther, R., Johnsen, O.-A., Gran, B.A.: Security assessments of safety critical systems using HAZOPs. In: Voges, U. (ed.) SAFECOMP 2001. LNCS, vol. 2187, pp. 14–24. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45416-0_2
25. Young, W., Leveson, N.G.: An integrated approach to safety and security based on systems theory. *Commun. ACM* **57**(2), 31–35 (2014)



Optimal Test Suite Generation for Modified Condition Decision Coverage Using SAT Solving

Takashi Kitamura¹(✉), Quentin Maissonneuve^{1,2}, Eun-Hye Choi¹,
Cyrille Artho³, and Angelo Gargantini⁴

¹ National Institute of Advanced Industrial Science and Technology (AIST),
Osaka, Japan

{t.kitamura,e.choi}@aist.go.jp

² University of Nantes, Nantes, France

quentin.maisonneuve1@etu.univ-nantes.fr

³ KTH Royal Institute of Technology, Stockholm, Sweden

artho@kth.se

⁴ Università di Bergamo, Bergamo, Italy

angelo.gargantini@unibg.it

Abstract. Boolean expressions occur frequently in descriptions of computer systems, but they tend to be complex and error-prone in complex systems. The modified condition decision coverage (MCDC) criterion in system testing is an important testing technique for Boolean expression, as its usage mandated by safety standards such as DO-178 [1] (avionics) and ISO26262 [2] (automotive). In this paper, we develop an algorithm to generate optimal MCDC test suites for Boolean expressions. Our algorithm is based on SAT solving and generates minimal MCDC test suites. Experiments on a real-world avionics system confirm that the technique can construct minimal MCDC test suites within reasonable times, and improves significantly upon prior techniques.

1 Introduction

Boolean expressions occur frequently in descriptions of computer systems, and are prevalent in software and hardware artifacts, such as programs, specifications, and hardware descriptions. On the other hand, they tend to be complicated and thus error-prone. *Boolean expression testing* is a technique to effectively test such complicated Boolean expressions. Modified Condition Decision Coverage (MCDC), a.k.a., Active Clause Coverage (ACC), is one of main coverage criteria for testing Boolean expressions. The MCDC criterion requires that each condition in a decision is shown by execution to independently affect the outcome of the decision, where “decision” means “Boolean expression” [3].

To demonstrate the notion, take the following Boolean expression:

$$\phi_1 = s \wedge (t \vee u). \quad (1)$$

The test suite in Table 1 consisting of the four test cases satisfies the MCDC criterion for this Boolean expression. Observe that, for example, the pair of the first and third test cases confirms that condition s independently affects the outcome of the expression: the value of condition s changes the value of ϕ_1 while the values of the other conditions, i.e. t and u , remain unchanged. Observe similarly that the first and second test cases confirm that for t , and the second and fourth confirm that for u . Boolean expressions such as (1) often represent abstract versions of actual logical expressions used in software artifacts; e.g., the following logical expression, $(z > 1) \wedge (f(x) \vee (x < y + 1))$, can be represented by (1), where the conditions $(z > 1)$, $f(x)$, and $(x < y + 1)$ are abstracted by s , t , and u respectively.

The usage of the MCDC criterion is motivated by several rationales.

1. MCDC can detect an erroneous use of *and* for *or* (or vice versa) in a Boolean expression; this is called ‘‘Operator Reference Fault (ORF)’’ [4].
2. MCDC is a stricter form of decision coverage than decision coverage, which stipulates each decision must evaluate to *true* and *false*.
3. The size of a test suite to satisfy the MCDC criterion (MCDC test suite, for short) is reasonably small, and hence testing with MCDC incurs a reasonable test cost. For a Boolean expression ϕ with n conditions, the size of its minimal MCDC test suite is $n + 1$ (e.g., [4–6]), instead of 2^n required for exhaustive testing. (This will be explained more in details in Sect. 2.2.)

Thus, MCDC effectively detects faults at reasonable cost. Due to such practical rationales, MCDC has been widely used. Various safety standards mandate its use in testing safety-critical components, such as DO-178 [1] (avionics) and ISO26262 [2] (automotive).

Driven by such industrial demands, MCDC has been actively studied from various aspects, e.g., development of variants of MCDC [4, 7], model-based testing with MCDC [7], coverage-driven test generation (a.k.a., CDTG) for program codes [8], empirical studies on its effectiveness [9–11], and testing Deep Neural Networks (DNNs) with MCDC [12]. Among them, test case generation for MCDC for Boolean expressions, which, given a Boolean expression, to find its MCDC test suite, is a basic function for such testing techniques with MCDC, and thus has been an important subject. Jones and Harrold developed algorithms for MCDC test reduction [13]. Arcaini et al. [14] developed a Boolean expression testing framework to construct a small test suite including MCDC. Bloem et al. [7] proposed yet another approach for MCDC test generation in developing a model based testing technique using MCDC.

In this paper, we develop an algorithm to construct small or even minimum MCDC test suites; i.e., given a Boolean expression ϕ with n conditions, generate

Table 1. A MCDC test suite for $\phi_1 = s \wedge (t \vee u)$.

No.	s	t	u	ϕ_1
1	1	1	0	1
2	1	0	0	0
3	0	1	0	0
4	1	0	1	1

an MCDC test suite whose size is equal to or smaller than $n + 1$. To realize such an algorithm, we use SAT solving as the key technique. To evaluate the proposed technique, we conduct experiments where we apply it to a real-world avionics system in comparison with state-of-the-art techniques. The experiments confirm that the technique can construct minimal MCDC test suites within reasonable times, and improves significantly upon prior techniques.

This paper is organized as follows. The next section states the problem that we tackle. Section 3 describes technical details of the proposed technique. Section 4 reports our evaluation of the proposed technique via experimental results. Section 5 states the significance of the work w.r.t., existing studies. Section 6 mentions possible future directions of this work.

2 Preliminaries

2.1 Definitions and Problem Formulations

In this subsection, we define the notion of ACC more formally. It is important to note that there have been proposed several variants of ACC, including General Active Clause Coverage (GACC), Restricted Active Clause Coverage (RACC), Correlated Active Clause Coverage (CACC), General Inactive Clause Coverage (GICC), Restricted Inactive Clause Coverage (RICC) [4, 15], Reinforced Condition Decision Coverage (RCDC) [11, 16], and Observable Modified Condition Decision Coverage (OMCDC) [17].

RACC is the classical and traditional form of ACC testing. On the other hand, recently, CACC has become known to be more practical in industrial usage. In this paper, we thus focus on RACC and CACC, but note that our proposed techniques in this paper can be applied to the other variants of ACC.

In the following, we define RACC and CACC. First, as in this paper we aim to develop ACC test generation techniques, we handle logical formulas as the target object that our techniques process; however, we mainly deal with them in the form of Boolean expressions by abstracting conditions in logical formulas by Boolean variables. This is as exemplified by the example in Sect. 1. We call Boolean variables in Boolean expressions *conditions*. We also use the term *predicate* as a shorthand for Boolean expression.

Next we introduce the notion of *determination*:

Definition 1 (Determination). *Let ϕ be a predicate and c be a condition of ϕ . We say that condition c determines ϕ , if all the other conditions in ϕ have values so that changing the truth value of c changes the truth value of ϕ .*

ACC stipulates that each condition of predicate ϕ independently determines the result of ϕ . Thus, it is convenient to name the condition on which we are focusing as the *major condition*, and all of the other conditions are *minor conditions*. Formally, for a predicate ϕ with n conditions c_1, c_2, \dots, c_n , and a condition c_i in ϕ , i.e., $i \in \{1 \dots n\}$ called the *major condition*, conditions c_j such that $j \in \{1 \dots n\} \wedge (i \neq j)$ are called *minor conditions*.

Definitions of test cases and test suites are also provided as follows:

Definition 2 (Test case and Test suite). *A test case of a predicate ϕ is a possible truth assignment for all the conditions in ϕ . A test suite is a set of test cases.*

The definitions of RACC and CACC are now ready as follows:

Definition 3 [RACC and CACC]. *Let ϕ be a predicate and T be a test suite. A CACC pair of a condition c of ϕ is a pair of test cases of ϕ , satisfying that (1) condition c determines ϕ as the major condition in both test cases and (2) the values of c are different. An RACC pair is a CACC pair such that the values for the minor conditions are same in the two test cases. We say that T covers a condition of ϕ in RACC (resp. CACC), if it includes an RACC (resp. CACC) pair of the condition. The test requirement of RACC (resp. CACC) for ϕ is that T should cover all the conditions of ϕ . RACC (resp. CACC) is a measure to describe the degree to which test requirements of RACC (resp. CACC), i.e., how many conditions, are covered by T .*

Note that RACC only differs from CACC in that the former requires the minor conditions of a test case pair have to have the same values, while the latter does not. Therefore, we can say RACC is stricter criterion than CACC; i.e., a RACC test suite for a predicate ϕ is also a CACC for it, but not vice versa. We call a test suite satisfying RACC (resp. CACC) test requirements a *RACC (CACC) test suite*.

2.2 Sizes of Minimum ACC Test Suites

The size of a test suite often plays a critical factor for the feasibility and success of real-world (software and hardware) system testing, and has been a main concern in theory and practice of testing techniques. A main reason for it is that “full automation” of test case execution, despite of a large body of research on it, is still largely limited in practice due to problems such as the *oracle problem*.¹ Another reason is that execution of one test case, even if automated, is often too expensive for allowed time and/or financial constraints; and thus the number of test cases that can be executed is limited.

Regarding properties on the ACC test suite size, it is often mentioned in literature that the size of a minimum ACC test suite is “around” $n + 1$, for a predicate with n conditions:

1. “Achieving MCDC requires, in general, a minimum of $n + 1$ test cases for a decision with n inputs.”, by Hayhurst et al. [5], where “decision” here means “predicate” in our setting.

¹ The challenge of distinguishing the corresponding desired and correct behavior from potentially incorrect behavior given a test case for a system under test.

2. “MCDC requires for n input variables a test suite at least of size $n + 1$.”, by Felbinger et al. [6], where ‘variables’ here mean ‘conditions’ in our setting.
3. “It turns out that for a predicate with n clauses, $n + 1$ distinct test requirements, rather than the $2n$ one might expect, are sufficient to satisfy active clause coverage.”, by Amman and Offutt [4], where ‘clause’ here means ‘condition’ in our setting.

These claims are similar but not equivalent: the first two claims state that at least $n + 1$ test cases are required to for an ACC test suite for a predicate with n conditions, while the third one states $n + 1$ test cases are sufficient. Although it can be conjectured from these that the minimum sizes of ACC test suites exist around $n + 1$, according to our best knowledge, a formal proof of this property is still an open problem.

2.3 Boolean Satisfiability Problem (SAT) and SAT Solvers

The *Boolean satisfiability problem (SAT)*, known to be an NP-complete problem, is the problem of determining if there exists an interpretation, a.k.a., model, that satisfies a given Boolean expression. *SAT solvers* are software tools that automatically solve SAT problems. As many NP-complete problems can be reduced to SAT, the development of efficient SAT solvers is an important and active research subject. We use SAT solvers for our ACC test generation.

3 Algorithm for Optimal MCDC Test Generation

In this section, we develop an algorithm for generating an optimal test suite for MCDC. We explain our algorithm by first introducing the notion of Predicate Determining Condition (PDC), based upon which we develop our algorithm.

3.1 Predicate Determining Condition (PDC)

We explain the notion of *predicate determining condition (PDC)*, introduced in [4], which we use in developing our ACC test generation algorithm.

Definition 1 provided the definition of determination. PDC, given the target predicate ϕ and the major condition of ϕ , provides the condition, as a logical formula for minor conditions, under which the major condition c determines ϕ .

Definition 4 (Predicate determining condition: PDC). *Let ϕ be a predicate ϕ and c be a condition in ϕ . The predicate determining condition (PDC) of a predicate ϕ and a condition c , denoted by ϕ_c , is a predicate under which the value of c determines that of ϕ .*

Since a predicate expresses a set of test cases, the PDC of condition c can be also interpreted as all test cases for which c determines ϕ .

An advantage of PDCs is that we can compute them. Given a predicate ϕ and a condition c , the PDC is computed as follows:

Proposition 1. *Let $\phi_{c=\text{TRUE}}$ (and $\phi_{c=\text{FALSE}}$) represent the predicate ϕ with every occurrence of c replaced by ‘TRUE’ (and ‘FALSE’), respectively. Neither $\phi_{c=\text{TRUE}}$ nor $\phi_{c=\text{FALSE}}$ has any occurrences of condition c . The PDC of condition c for a predicate ϕ is obtained by connecting the two expression with exclusive-or:*

$$\phi_c = \phi_{c=\text{TRUE}} \oplus \phi_{c=\text{FALSE}} \quad (2)$$

This proposition holds, as the requirement that condition c independently affects the outcome of predicate ϕ can be described as the following formula, which corresponds to (2): $(\phi_{c=\text{TRUE}} \wedge \neg\phi_{c=\text{FALSE}}) \vee (\neg\phi_{c=\text{TRUE}} \wedge \phi_{c=\text{FALSE}})$.

Example 1. Let ϕ be $\phi_1 = s \wedge (t \vee u)$ of Boolean expression (1). The PDC of predicate ϕ and condition s , i.e., ϕ_s , can be derived as follows:

$$\phi_s = \phi_{s=\text{TRUE}} \oplus \phi_{s=\text{FALSE}} \quad (3)$$

$$= (\text{TRUE} \wedge (t \vee u)) \oplus (\text{FALSE} \wedge (t \vee u)) \quad (4)$$

$$= (t \vee u) \oplus \text{FALSE} \quad (5)$$

$$= (t \vee u) \quad (6)$$

This means that for the major condition s to determine predicate ϕ , there are three choices for truth assignments, (t, u) , $(t, \neg u)$, and $(\neg t, \neg u)$.

We remark that if the PDC of a predicate and a condition is not satisfiable, then this means that it is not feasible to create an ACC test pair for that condition. We call such a condition an *infeasible condition*.

3.2 SAT-encoding

The key device to develop our algorithm is the SAT-encoding of the following problem: “For a predicate ϕ and the number of test cases k , is it possible to construct an ACC test suite?”. We explain about the SAT-encoding of this problem in this subsection. The encoding uses a *matrix model* with p columns and n rows as Boolean conditions x_p^n to encode the test suite, as follows:

$$X = \begin{pmatrix} x_1^1 & \dots & x_p^1 \\ \vdots & & \vdots \\ x_1^k & \dots & x_p^k \end{pmatrix} \quad (7)$$

This matrix model of a test suite expresses that the i -th row represents the i -th test case, and the p -th column of the row represents the value of the p -th parameter of the i -th test case.

Using the matrix model to represent a test suite, we construct the SAT encoding to construct an ACC test suite with the size k , as follows:

$$\begin{aligned}
 \text{encode}(\phi, \text{conds}_\phi, k) = & \bigwedge_{c \in \text{conds}_\phi} \bigvee_{i=1}^k \left(\underbrace{(x_c^i \wedge \phi_c^i)}_{(2)} \wedge \right. \\
 & \left. \bigvee_{j=1, i \neq j}^k \left(\underbrace{(\neg x_c^j \wedge \phi_c^j)}_{(3)} \wedge \bigwedge_{d \in \text{conds}_\phi \text{ s.t. } c \neq d} \underbrace{(x_d^i \Leftrightarrow x_d^j)}_{(4)} \right) \right) \quad (8)
 \end{aligned}$$

where ϕ_c^i is PDC of condition c whose Boolean conditions are those in the i -th row of the matrix model, and conds_ϕ is a set of feasible conditions in ϕ . This SAT-encoding specifies the following: Sub-formula (2) specifies that for a condition of the predicate, say c , in at least one row of the matrix table, i.e., in one test case of the test suite, the value of the condition in the test case, x_c , must be ‘TRUE’, while the PDC of the condition c must be also true. On the other hand, sub-formula (3) specifies that the same condition must be ‘FALSE’, while the PDC of c must be ‘TRUE’, in at least one test case. The $i \neq j$ in formula (3) clarifies that the value of condition c cannot be ‘TRUE’ and ‘FALSE’ in the same row. This sub-formula can be omitted, but carefully placed redundant SAT formula can speed up the solving process. The outermost conjunction (1) indexed by conditions of ϕ imposes that the above sub-formulas (2) and (3) are to be applied to all the conditions in the predicate under test ϕ .

Sub-formulas (1), (2), and (3) are encoding for CACC, since sub-formulas (2) and (3) together satisfy condition (1) and (2) of Definition 3, and sub-formula (1) guarantees it for all the conditions. Sub-formula (4) specifies the condition for RACC that stipulates that values of all the minor conditions are equivalent. For RACC, we may also omit ϕ_c^j in (2); this is possible, since if constraints (2) and (4) hold and $x_c^j = \text{FALSE}$, then the i -th test case and the j -th test case differ only in the value of x_c so that ϕ_c^j and ϕ_c^i should be equivalent.

The encoding for a given test suite size induces a SAT formula, such that ‘SAT’ for the evaluation of the formula entails the existence of an ACC test suite with that size; in that case, the solution contains all the information on the ACC test suite. On the other hand, ‘UNSAT’ means the refutation of the existence of such a test suite.

Example 2. The following snippet of SAT-formula is the SAT-encoding of Boolean expression ϕ_1 with the test suite size 4. The sub-formulas (i), (ii), and (iii), respectively, express that conditions s, t and u are confirmed to independently affect the outcome of the predicate.

$$\begin{aligned}
& \left(\begin{array}{l} (s^1 \wedge \phi_s^1) \wedge \left(((\neg s^2 \wedge \phi_s^2) \wedge (t^1 \Leftrightarrow t^2) \wedge (u^1 \Leftrightarrow u^2)) \vee \right. \\ \quad \left. ((\neg s^3 \wedge \phi_s^3) \wedge (t^1 \Leftrightarrow t^3) \wedge (u^1 \Leftrightarrow u^3)) \vee ((\neg s^4 \wedge \phi_s^4) \wedge (t^1 \Leftrightarrow t^4) \wedge (u^1 \Leftrightarrow u^4)) \right) \\ \vdots \\ \vee (s^4 \wedge \phi_s^4) \wedge \left(((\neg s^1 \wedge \phi_s^1) \wedge (t^4 \Leftrightarrow t^1) \wedge (u^4 \Leftrightarrow u^1)) \vee \right. \\ \quad \left. ((\neg s^2 \wedge \phi_s^2) \wedge (t^4 \Leftrightarrow t^2) \wedge (u^4 \Leftrightarrow u^2)) \vee ((\neg s^3 \wedge \phi_s^3) \wedge (t^4 \Leftrightarrow t^3) \wedge (u^4 \Leftrightarrow u^3)) \right) \end{array} \right) \quad (i) \\
\wedge \left(\begin{array}{l} (t^1 \wedge \phi_t^1) \wedge \left(((\neg t^2 \wedge \phi_t^2) \wedge (s^1 \Leftrightarrow s^2) \wedge (u^1 \Leftrightarrow u^2)) \vee \right. \\ \quad \left. ((\neg s^3 \wedge \phi_s^3) \wedge (t^1 \Leftrightarrow t^3) \wedge (u^1 \Leftrightarrow u^3)) \vee ((\neg s^4 \wedge \phi_s^4) \wedge (t^1 \Leftrightarrow t^4) \wedge (u^1 \Leftrightarrow u^4)) \right) \\ \vdots \end{array} \right) \quad (ii) \\
\wedge \underline{\left((u^1 \wedge \phi_u^1) \wedge \dots \right)} \quad (iii)
\end{aligned} \tag{9}$$

3.3 Algorithm

We proceed with the actual algorithm on how to find a smaller ACC test suite using SAT-encoding. For the discussion, we first argue about properties on the size of ACC test suites. From our literature review in Sect. 2.2, we conjecture that the minimum size of ACC test suites should be around $n + 1$ for a predicate with n conditions. However, because no formal proof for this lower bound exists yet, we use the following proposition, which is a weaker form of the conjecture, but which can be proved easily:

Proposition 2. *For a predicate with n conditions, the minimum size of a CACC/RACC test suite is at most $2n$.*

Proof. From Definition 3, an RACC test suite should include at least one RACC pair for each condition in ϕ . Since we can make an RACC pair with two test cases, $2n$ is enough for the size of an RACC test suite for ϕ with n conditions. Since a RACC test suite is also an CACC one, this property also holds for CACC.

Using Proposition 2, the algorithm is designed as in Algorithm 1. The algorithm, at the beginning (in Line 1), computes feasible conditions of ϕ . Computing all the feasible conditions can be done by collecting conditions whose PDCs are satisfiable (possibly, using a SAT solver). Based on the computed feasible conditions ($conds_\phi$) and their number ($|conds_\phi|$), the algorithm starts searching for a minimum test suite. It iteratively attempts to generate an ACC test suite by decrementing the size of a test suite by one, starting from $2 * |conds_\phi|$. In each iteration, the algorithm applies the SAT-encoding of the current test size, and applies a SAT solver to the encoded formula. The decremental iteration continues while the encoded formula is satisfiable, and terminates when it encounters ‘unsatisfiable (UNSAT)’ (Line 6). The test suite found in the last satisfying iteration is minimal. Line 2 handles a corner case, where no feasible conditions are contained in ϕ , by throwing an exception.

Algorithm 1. The main part of our algorithm for CACC

Input: A Boolean Expression (ϕ)
Output: A CACC test suite

- 1 Compute the feasible conditions of ϕ , and store them in $conds_\phi$;
- 2 if ($|conds_\phi| == 0$) **return** (“Exception: No feasible conditions exist.”);
- 3 Set $size \leftarrow 2 * |conds_\phi|$, for the initial test size;
- 4 **while true do**
- 5 (isSAT, model) \leftarrow checkSat(encode(ϕ , $conds_\phi$, size));
- 6 if (isSAT == UNSAT) **return** suite;
- 7 Make test suite from model;
- 8 size \leftarrow size - 1
- 9 **end**

The encoded formula in each iteration is a SAT problem over Boolean variables, and the test size. The size starts at ‘ $2 * |conds_\phi|$ ’ and is reduced by one in each iteration. Thus, the algorithm is guaranteed to terminate with a minimum ACC test suite, in principle. In practice, however, limited computational resources may prevent us from finding the minimum one. The SAT solver may take a lot of time and/or a lot of memory, as the attempted search approaches the optimal size. Thus, a concern is scalability of the algorithm, which we examine by experiments in Sect. 5.

The algorithm uses $2n$ for the initial test size based on Proposition 2. We have two main reasons to adopt this proposition, instead of the conjecture for $n + 1$. First, our algorithm is certainly correct using the proved proposition with $2n$, instead of using the (unproved) conjecture with $n + 1$. Second, our algorithm with $2n$ for the initial size performs well in practice, outperforming existing techniques, as shown in Sect. 5. Moreover, if the worst-case bound of $n + 1$ is proven in the future, we can easily adapt that result in our algorithm, by setting $n + 1$ as the initial size.

4 Related Work

Test suite reduction and minimization is one of central subjects in software testing. To the best of our knowledge, the earliest work which discusses the test suite sizes in MCDC testing is by Jones and Harrold [13]. They develop a test suite reduction technique for MCDC, which, given a predicate ϕ and an MCDC test suite T , finds an MCDC test suite T' such that $T' \subset T$. The basic approach of the technique is to construct such an MCDC test suite T' , by iteratively choosing or removing one test case from the given MCDC test suite T based on the *contribution* weight computed for each test case in T . For example, their ‘build-up’ approach constructs an MCDC test suite by iteratively adding a test case in T with the highest contribution weight to T' , starting from the empty set as the initial test suite of T' . Note that the JH-algorithm is a test reduction technique, rather than MCDC test generation; however, by complementing the

Table 2. The benchmark set consists of 20 logical expressions, retrieved from “*Traffic Collision Avoidance System (TCAS)*” of an avionics system [8]. The logical expressions in the benchmark set contain 5 to 15 (feasible and infeasible) conditions, as indicated by #conds. The table also shows the infeasible conditions (I.C.) for each benchmark; ACC pairs cannot be made for such infeasible conditions.

No	Boolean expression	#conds	I.C.
1	$a(!b+!c)d + e$	5	
2	$!(ab)(d!e!f+!de!f+!de!f)((ac(d+e)h) + (a(d+e)!h) + (b(e+f)))$	7	
3	$!(cd)(!ef!g!a(bc+!bd))$	7	
4	$ac(d+e)h + a(d+e)!h + b(e+f)$	7	
5	$!ef!g!a(bc+!bd)$	7	
6	$(!ab + !ab)!(cd)!(gh)((ac + bd)e(fg+!fh))$	8	
7	$(ac + bd)e(fg+!fh)$	8	
8	$(a((c + d + e)g + af + c(f + g + h + i)) + (a + b)(c + d + e)i)!(ab)!(cd)!(ce)!(de)!(fg)!(fh)!(fi)!(gh)!(hi)$	9	
9	$a(!b+!c + bc!(!fgh!i+!ghi)!(!fglk+!g!ik)) + f$	9	
10	$a((c + d + e)g + af + c(f + g + h + i))(a + b)(c + d + e)i$	9	b, h
11	$(ac + bd)e(i+!g!k+!j!(h+!k))$	10	
12	$(ac + bd)e(i+!g!k+!j!(h+!k))(ac + bd)e(i+!g!k+!j!(h+!k))$	10	
13	$(!ab + !ab)!(cd)(f!g!h+!f!g!h+!f!g!h)!(jk)((ac + bd)e(f + (i(gj + hk))))$	11	
14	$(ac + bd)e(f + (i(gj + hk)))$	11	
15	$(a(!d+!e + de!(!fgh!i+!ghi)!(!fglk+!g!ik)) + !(fgh!i+!ghi)!(!fglk+!g!ik)(b + c!m + f))(a!b!c+!ab!c+!a!bc)$	12	
16	$a + b + c+!c!def!g!h + i(j + k)l$	12	
17	$a(!d+!e + de!(!fgh!i+!ghi)!(!fglk+!g!ik)) + !(fgh!i+!ghi)!(!fglk+!g!ik)(b + c!m + f)$	12	
18	$a!b!c!d!e!f(g+!g(h+i))!(jk+!jl+m)$	13	
19	$a!b!c(!f(g+!f(h+i))) + f(g+!g(h+i)!d!e)!(jk+!jl!m)$	13	
20	$a!b!c(f(g+!g(h+i)))(!e!n+d)+!n(jk+!jl!m)$	14	

algorithm with a function to prepare initial MCDC test suites, it can be used as a test generation technique for MCDC.

Arcaini et al. [14] developed a general framework for test generation for Boolean expression testing with various coverage criteria, including MCDC. The technique constructs an MCDC test suite with the basic approach of accumulating an MCDC pair (a pair of test cases) for each condition in turn. Offutt et al. [18], in developing a model-based testing technique with MCDC testing, discuss a similar technique to construct an MCDC test suite. However, both techniques require a test suite of size $2n$ for a predicate with n conditions, which is usually larger than the test suites that our algorithm generates.

Bloem et al. [19] devised an algorithm to construct MCDC test suites, also in developing their model-based testing technique using MCDC. The technique is similar to those by Arcaini et al. [14] and Offutt et al. [18], in the respect

that its basic procedure is to accumulate an MCDC pair for each condition in turn; however, they apply an improvement to this basic approach to reduce the size of generated test suites. The improvement is that the algorithm, for every condition, checks if the current test suite already includes an MCDC pair for the condition, when adding an MCDC pair for each condition. If the MCDC pair already exists in the test suite, the algorithm skips adding an MCDC pair for that condition. They also use SAT-solving to realize the technique.

Our algorithm is inspired by the SAT-based test generation technique for combinatorial interaction testing (CIT) by Hnich et al. [20]. The key of their technique is a SAT-encoding of the problem of finding a CIT test suite with a specified size. It is confirmed that the technique can construct CIT test suites with reasonably small sizes, compared with other approaches. Due to the significance, this work is followed by a number of studies for extension or acceleration (e.g., [21]). Our work in this paper thus can be seen as a new application direction of the SAT-based test generation technique to MCDC testing, and also as an import of new technical element to MCDC testing field.

5 Experimental Evaluation

In this section, we conduct experiments to evaluate our proposed technique. Due to the space limitation, we only evaluate the proposed algorithm for the CACC case, as it is the most basic, practical, and interesting case among ACC variants. To clarify the purpose of the experiments and evaluation, we set the following research questions:

- RQ1: Does our algorithm perform better than existing techniques, with respect to the sizes of generated test suites and computation times?
- RQ2: Can our algorithm find minimal CACC test suites?

For investigating these RQs, we implemented our algorithm in Scala. Also to conduct the experiments, we prepared a benchmark set of logical expressions retrieved from “*Traffic Collision Avoidance System (TCAS)*” of a real-world avionics system [8]. The details about benchmark data are shown in Table 2.

For RQ1, we also implemented the MCDC test generation technique based on the JH-algorithm and the technique by Bloem et al., as the the state-of-the-art techniques by ourselves, since implementations of these techniques are not available. Since JH-algorithm is an MCDC test reduction technique instead of test generation, we complemented it with the function to randomly generate MCDC test suites so that JH-algorithm can reduce them as initial test suites. The function is realized to randomly generate m ACC pairs for each condition of a given formula. Therefore, the most basic form of $m = 1$ means $2n$ test cases, where n is the number of conditions in the given formula. We prepared several variants that vary in the sizes of initial MCDC test suites, as follows: $m = 1, 50, 200$, and 400 , respectively denoted by JH_1 , JH_{50} , JH_{200} , and JH_{400} . Also for RQ1, the timeout is set to 60s for our algorithm for a proper comparison and evaluation.

Table 3. The results of our experiments. This table shows the sizes of the generated CACC test suites ($\#tests$) and the computation times in seconds ($time$) of the algorithms (JH₁, JH₅₀, JH₂₀₀, JH₄₀₀, by Bloem et al., and our algorithm) for the benchmark set in Table 2. To properly answer RQ1 and RQ2, we use two timeout settings for our algorithm, 60 s ($time_{<60}$), and 1800 s ($time_{<1800}$). A ‘—’ in columns $time_{<1800}$ means a minimum test suites is found within in 60 s, and the time is given in column $time_{<60}$. The bold font in the columns for $\#tests$ denote instances in which the test suite size is smaller than or equal to those found by other algorithms, while that in the columns for $time$ means the algorithms returned the test suite within 60 s. For columns of our algorithms, a ‘*’ denotes a minimal test suite is found with the guarantee of finding ‘UNSAT’ by the SAT solver. $\#wins$ denotes the number of cases where the algorithm generates the smallest test suite among all algorithms within 60 s; multiple winners may exist for each benchmark data. $impr(\%)$ denotes the reduction rate on the test suite size by our algorithm compared with that by the corresponding algorithm of the column; e.g., the test suite size by our algorithm is smaller than that by JH₄₀₀ by 19.7% in total. The experiments were performed on a machine with a Quad-Core Intel Xeon E5 3.7 GHz and 64 GB Memory running on Mac OS High Sierra 10.13.3. For running the programs, Scala option “-Xmx8g -Xms1024m” is used. As the back-end SAT solver, we used SMT solver Z3 [22].

No	JH ₁		JH ₅₀		JH ₂₀₀		JH ₄₀₀		Bloem et al.		Our algorithm (2n for the initial size)			
	$\#tests$	$time$	$\#tests$	$time$	$\#tests$	$time$	$\#tests$	$time$	$\#tests$	$time$	$\#tests$	$time_{<60}$	$\#tests$	$time_{<1800}$
1	6	0.9	6	4.2	6	10.4	6	10.8	6	0.9	6*	0.5	—	
2	9	1.7	8	11.1	8	11.3	8	12.1	8	1.5	5*	1.3	—	
3	7	1.6	7	2.9	7	2.8	7	2.9	7	1.3	6*	1.0	—	
4	12	1.7	7	18.1	6	52.5	6	98.4	9	1.4	6*	0.9	—	
5	9	1.4	8	5.1	8	5.1	8	5.0	9	1.4	8*	0.9	—	
6	9	1.8	8	13.7	7	49.5	7	118.6	9	1.7	3*	1.5	—	
7	10	1.7	8	20.7	7	69.4	7	89.0	10	1.7	7*	1.0	—	
8	15	2.7	12	24.7	12	96.9	10	208.6	11	2.5	4*	2.6	—	
9	11	2.6	10	16.9	10	27.2	10	44.5	11	2.2	9*	3.0	—	
10	9	2.1	8	18.8	7	71.7	7	138.1	9	1.8	7*	1.0	—	
11	12	2.6	10	27.3	10	125.5	9	290.0	12	2.5	8*	1.8	—	
12	12	2.6	10	27.4	10	126.7	9	289.1	12	2.5	8*	2.5	—	
13	15	3.8	13	27.0	12	99.0	13	243.4	13	3.6	9*	3.5	—	
14	12	3.3	13	30.8	12	148.7	12	369.9	13	3.0	9*	1.8	—	
15	18	4.4	17	34.5	16	150.6	16	414.1	17	4.3	9*	5.8	—	
16	13	3.5	13	35.4	12	142.8	11	304.3	13	3.2	11*	2.0	—	
17	18	4.9	15	35.9	13	165.6	13	435.5	16	4.4	11	4.6	11*	80.7
18	15	4.1	13	37.9	13	127.4	13	286.7	14	3.8	13	2.6	13	>1800
19	20	4.2	18	38.2	16	175.0	16	481.1	14	4.0	12	3.3	12*	197.7
20	18	4.6	18	42.0	16	194.8	15	540.5	16	4.4	12	3.6	12*	167.3
$\#wins$	2		3		4		3		2		20	—		
$impr(\%)$	32.0		23.4		18.2		16.2		25.7		—		—	

For RQ2, we measure the time required for our algorithm to find minimal test suites by finding UNSAT for the benchmark set, thus proving that no test suite smaller than the previously found satisfiable assignment exists. For this, the timeout is set to 1800 s for our algorithm.

Table 3 shows the experimental results. We answer the research questions, by observing the experimental results, in the following.

Answer to RQ1. From the experimental results, we conclude that our algorithm performs better than the state-of-the-art techniques in both the size of generated test suites and computation times. First, our algorithm wins against the other algorithms for all the benchmark data. On other hand, the algorithm variants based on the JH-algorithm or the technique by Bloem et al., only perform equally well to ours only in a couple of cases. Moreover, our algorithm can build such smaller test suites fairly quickly, i.e., within a few seconds for all the benchmark data, which is much faster than the other techniques. The improvement rate achieved by our algorithm is also significant; it can generate test suites that are from 16.2% up to 32.0% smaller than the the other techniques.

Answer to RQ2. From the experiments, we confirm that the resulting test suite is of size equal or less than $n + 1$ in 17 out of 20 cases. We can also confirm that in 19 cases our algorithm can find the guaranteed minimum sizes by finding ‘UNSAT’ within 1800s. For the case where the algorithm cannot guarantee the minimum size of the test suite (i.e., for benchmark #18), there may exist a smaller CACC test suite.

6 Conclusion and Future Work

We believe our algorithm to construct small or even the minimum MCDC test suites will play a significant role broadly in the software testing field, since (1) MCDC has become a de-facto standard coverage criterion in testing safety-critical systems, and (2) the technique, as a basic testing technique, can be used in combination with other testing techniques, such as model-based testing and program-code-level testing.

Model-based testing, a sub-discipline of model-based development (MBD), is another key technique in safety-critical domains such as in avionics and automotive. A number of studies have attempted to introduce MCDC testing in model-based testing techniques, to enhance the accountability as well as its bug-detecting ability [7, 18, 23, 24]; also see Sect. 4. Generally, these techniques deal with state transition systems as models, and regard a sequence of states as a test case. To derive effective test cases, these techniques apply the MCDC criterion to the *transition guards*, which specify as a logical formula the conditions for transitions to take place. This approach is reasonable since real-world transition systems often become complex and error-prone, however, their test generation components are not efficient w.r.t. the test suite size and computation time. Our algorithm can complement such model-based testing techniques.

Another application area of our algorithm is *Coverage-Directed Test case Generation (CDG/CDTG)*. CDTG is a white-box testing technique, working at program code level to generate test cases, to achieve a higher code coverage. CDTG and its adaptation for MCDC have been actively studied [25–27]. Such CDTG techniques targeting MCDC analyze the structure of the given program, especially logical formulas embedded in the program, and try to find test inputs that reach each logical formula in the program. Although these techniques differ

in several dimensions such as search techniques (e.g., dynamic symbolic execution [25], model checking [26], search-based optimization [27]), they have in common that they prepare MCDC test suites for embedded logical formulas before starting the search. Similar to the situation of model-based testing, our proposed algorithms can effectively complement such CDTG techniques.

We consider several directions for future work. One direction is to extend our algorithm to deal with general logical formulas, e.g., $x = 0 \wedge (x > 0 \vee y = 0)$, instead of their abstracted forms in Boolean expressions, i.e., $s \wedge (t \vee u)$. It is since our algorithm may produce unusable tests for such an abstracted predicate. For example, recall Table 1 is an MCDC test suite for $s \wedge (t \vee u)$. Note, however, that the first test case, i.e., $\{s = \text{TRUE}, t = \text{TRUE}, u = \text{FALSE}\}$, is unusable if $s \wedge (t \vee u)$ originates from $x = 0 \wedge (x > 0 \vee y = 0)$ by abstraction, since there is no assignment that makes $s = \text{TRUE}$ (i.e., $x = 0$) and $t = \text{TRUE}$ (i.e., $x > 0$) at the same time. One approach for this would be to impose additional constraints among abstracted variables, to specify hidden relations between s and t . In the example, a constraint like $\neg(s \wedge t)$ would encode that. The PDC of s for this predicate would be constructed as follows: $\phi_{s=\text{TRUE}} \oplus \phi_{s=\text{FALSE}} \wedge \neg(s \wedge t)$. We need careful investigations on this approach including how to specify such additional constraints automatically and how it affects the size of test suite.

Another direction is to further accelerate or scale our algorithm. Although it was shown by the experiments that our algorithm works fairly well for a real-world system in the avionics domain, it may need to be prepared for larger and more complex systems. Toward this technical improvement, we think the technique using incremental SAT solving [21] can be effectively applied. Thirdly, we consider to apply our technique to other ACC variants explained in Sect. 2. We are also interested in applying it to improve the testing technique for Deep Neural Networks based on MCDC proposed by [12].

Acknowledgment. We thank H el ene Waeselynck and anonymous reviewers whose comments have greatly improved this paper.

References

1. RTCA: DO-178C: Software considerations in airborne systems and equipment certification (2011)
2. ISO26262: International Organization for standardization, road vehicles - functional safety (2009)
3. Chilenski, J.J., Miller, S.P.: Applicability of modified condition/decision coverage to software testing. *Softw. Eng. J.* **9**(5), 193–200 (1994)
4. Ammann, P., Offutt, J.: *Introduction to Software Testing*. Cambridge University Press, Cambridge (2008)
5. Kelly J.H., Dan S.V., John J.C., Leanna K.R.: A practical tutorial on modified condition/decision coverage. Technical report NASA/TM-2001-210876, NASA (2001)
6. Felbinger, H., Pill, I., Wotawa, F.: Classifying test suite effectiveness via model inference and ROBBDs. In: Aichernig, B.K., Furia, C.A. (eds.) TAP 2016. LNCS, vol. 9762, pp. 76–93. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41135-4_5

7. Bloem, R., Hein, D., Röck, F., Schumi, R.: Case study: automatic test case generation for a secure cache implementation. In: Blanchette, J.C., Kosmatov, N. (eds.) TAP 2015. LNCS, vol. 9154, pp. 58–75. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21215-9_4
8. Weyuker, E.J., Goradia, T., Singh, A.: Automatically generating test data from a boolean specification. *IEEE Trans. Softw. Eng.* **20**(5), 353–363 (1994)
9. Dupuy, A., Leveson, N.: An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software. In: Proceedings of the 19th Digital Avionics Systems Conference (DASC 2000) (2000)
10. Vilkomir, S., Starov, O., Bhambroo, R.: Evaluation of T-wise approach for testing logical expressions in software. In: Proceedings of the Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013 Workshops Luxembourg, pp. 249–256 (2013)
11. Kapoor, K., Bowen, J.P.: A formal analysis of MCDC and RCDC test criteria. *Softw. Test., Verif. Reliab.* **15**(1) 21–40 (2005)
12. Sun, Y., Huang, X., Kroening, D.: Testing deep neural networks (2018). [arXiv:1803.04792](https://arxiv.org/abs/1803.04792)
13. Jones, J.A., Harrold, M.J.: Test-suite reduction and prioritization for modified condition/decision coverage. *IEEE Trans. Softw. Eng.* **29**(3), 195–209 (2003)
14. Arcaini, P., Gargantini, A., Riccobene, E.: How to optimize the use of SAT and SMT solvers for test generation of boolean expressions. *Comput. J.* **58**(11), 2900–2920 (2015)
15. Chilenski, J.J.: An investigation of three forms of the modified condition decision coverage (MCDC) criterion. Technical report DOT/FAA/AR-01/18, Office of Aviation Research (2001)
16. Vilkomir, S.A., Bowen, J.P.: From MC/DC to RC/DC: formalization and analysis of control-flow testing criteria. In: Formal Methods and Testing, an Outcome of the FORTEST Network, Revised Selected Papers, pp. 240–270 (2008)
17. Whalen, M.W., Gay, G., You, D., Heimdahl, M.P.E., Staats, M.: Observable modified condition/decision coverage. In: 35th International Conference on Software Engineering ICSE 2013, San Francisco, CA, USA, pp. 102–111 (2013)
18. Offutt, A.J., Liu, S., Abdurazik, A., Ammann, P.: Generating test data from state-based specifications. *Softw. Test., Verif. Reliab.* **13**(1) pp. 25–53 (2003)
19. Bloem, R., Greimel, K., Koenighofer, R., Roeck, F.: Model-based MCDC testing of complex decisions for the Java Card Applet Firewall. In: Proceedings of the Fifth International Conference on Advances in System Testing and Validation Lifecycle (VALID 2013), pp. 1–6 (2013)
20. Hnich, B., Prestwich, S., Selensky, E.: Constraint-based approaches to the covering test problem. In: Faltings, B.V., Petcu, A., Fages, F., Rossi, F. (eds.) CSCLP 2004. LNCS (LNAI), vol. 3419, pp. 172–186. Springer, Heidelberg (2005). https://doi.org/10.1007/11402763_13
21. Yamada, A., Kitamura, T., Artho, C., Choi, E.H., Oiwa, Y., Biere, A.: Optimization of combinatorial testing by incremental SAT solving. In: Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST 2015), pp. 1–10 (2015)
22. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
23. Heimdahl, M.P.E., Devaraj, G.: On the effect of test-suite reduction on automatically generated model-based tests. *Autom. Softw. Eng.* **14**(1), 37–57 (2007)

24. Gargantini, A., Heitmeyer, C.L.: Using model checking to generate tests from requirements specifications. In: Proceedings of the 7th European Conference of Software Engineering, Held Jointly with the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 1999), pp. 146–162 (1999)
25. Pandita, R., Xie, T., Tillmann, N., de Halleux, J.: Guided test generation for coverage criteria. In: Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM 2010), Timisoara, Romania, pp. 1–10 (2010)
26. Bokil, P., Darke, P., Shrotri, U., Venkatesh, R.: Automatic test data generation for C programs. In: Proceedings of the Third IEEE International Conference on Secure Software Integration and Reliability Improvement SSIRI 2009, pp. 359–368 (2009)
27. Awedikian, Z., Ayari, K., Antoniol, G.: MC/DC automatic test input data generation. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2009), Canada, pp. 1657–1664 (2009)



Efficient Splitting of Test and Simulation Cases for the Verification of Highly Automated Driving Functions

Eckard Böde¹(✉), Matthias Büker¹(✉), Ulrich Eberle², Martin Fränzle¹, Sebastian Gerwin¹, and Birte Kramer¹(✉)

¹ OFFIS - Institut für Informatik, Escherweg 2, 26121 Oldenburg, Germany
{boede,bueker, fraenzle, gerwin, kramer}@offis.de

² Opel Automobile GmbH, Bahnhofplatz, 65423 Rüsselsheim am Main, Germany
ulrich.eberle@opel.com

Abstract. We address the question of feasibility of tests to verify highly automated driving functions by optimizing the trade-off between virtual tests for verifying safety properties and physical tests for validating the models used for such verification. We follow a quantitative approach based on a probabilistic treatment of the different quantities in question. That is, we quantify the accuracy of a model in terms of its probabilistic prediction ability. Similarly, we quantify the compliance of a system with its requirements in terms of the probability of satisfying these requirements. Depending on the costs of an individual virtual and physical test we are then able to calculate an optimal trade-off between physical and virtual tests, yet guaranteeing a probability of satisfying all requirements.

Keywords: Verification · Simulation · Highly automated driving
Statistical verification · Testing · Advanced driver assistant systems
Optimal trade-off

1 Introduction

Advanced driver assistant systems (ADAS) and highly automated driving functions (HAD) are increasingly complex and their dependency on the environmental situation is increasing. An important step in bringing such systems into the market is to guarantee their safe operation. To this end, the reaction of these systems to all potential inputs needs to be verified. Due to their complexity not only individual inputs but sequences of inputs need to be checked. An analytical verification which exhaustively checks all input combinations and sequences is infeasible¹. Therefore, it is not only important to develop these functions in a safe way but also use testing for their verification. During testing the system is probed at specific points (input sequences) from which the safety

¹ Besides the prohibitively large computational complexity, this also requires an accurate, formal description of possible environments.

of the system, or more generally the compliance of the system with the elicited requirements, can be inferred. Using statistical arguments, one can estimate the necessary number of tests (number of test-kilometers to drive) in order to guarantee a level of safety which is most likely as high as the level to be achieved without the ADAS under test [1]. For HAD this number will presumably be even higher. As determined in [1,2] the scale of such physical tests is also prohibitively large as the costs for such tests (which need to be performed with every newly developed ADAS) amount to the order of hundreds of millions of Euros. Thus, on the one hand, the complexity of the systems forces one to use tests but, on the other hand, physical tests are not sufficiently cost-efficient. A potential solution to this dilemma is to replace physical components with virtual ones mimicking the behavior of their physical counterparts denoted *Virtual Integration*. Depending on which part is replaced with a virtual substitute, these tests are called for instance model-in-the-loop (MIL), software-in-the-loop (SIL), hardware-in-the-loop (HIL), or vehicle-in-the-loop (VIL), see [3] or [4]. Such a virtual setup can not only be used for safety assessment, but also for early-phase development, thereby achieving a much more cost-efficient development cycle. However, when replacing parts of a real operational environment with virtual components one has to guarantee a sufficiently realistic behavior of the virtual components such that results obtained from a simulation can be transferred to a non-virtual situation.

Although physical tests are again necessary to estimate the accuracy of the models used, such validation of models would only need to be performed once for each model. Hence, the overall costs for virtual and physical tests could still be feasible. In this paper, we address the question of feasibility by optimizing the trade-off between virtual tests for verifying safety properties of highly automated driver assistant systems and the physical tests for validating the models used for such verification. To this end, we follow a quantitative approach based on a probabilistic treatment of the different quantities in question. That is, we quantify the accuracy of a model in terms of its probabilistic prediction ability. Similarly, we quantify the compliance of a system with its requirements in terms of the probability of satisfying these requirements—obtained as an average across all possible uncertainties. As these probabilities are often unknown but have to be estimated based on the finite amount of test-samples, we additionally account for the statistical estimation uncertainty. Depending on the costs of an individual virtual and physical test we are then able to calculate an optimal trade-off between physical and virtual tests, yet guaranteeing a probability of satisfying all requirements.

Note that such a probabilistic treatment is mainly for practical reasons, similar to the arguments in [1]. We expect a human driver (one of several other traffic participants, from the perspective of a system-under-test) to be subject to randomness. That is, even if the initial situation is identical, the reaction of a traffic participant can be different between repetitions. In order to still quantify the level of compliance with the requirements we merely require the system to satisfy the requirement up to a pre-specified confidence level. For the same reasons we also can only measure the current level of safety with a similar uncertainty, see [1].

The results presented in this paper are meant to be of a generic nature i.e., applicable from a test-process perspective in general. As such, we do not analyse particular test instances, but investigate more the general conditions in which such a framework is applicable. Furthermore we are using an abstract notion of validity, thus we are not giving concrete checkpoints that would allow a test engineer to decide whether a given model is a valid replacement of the real world. Despite this, it allows us to make predictions about the test processes that have a practical impact. In particular, we illustrate how many tests would be needed (both physical and virtual) under the assumptions of an optimal split. Additionally, we can directly calculate the potential savings in costs compared to a pure physical test as illustrated by Winner *et al.* [1] while achieving the same quantitative guarantee of safety (see Sect. 3.5).

2 Related Work

To use simulation for the verification of ADAS/HAD is a commonly proposed solution for the problems stated above. There are several approaches that offer ideas on how to integrate simulation and test in the verification process.

Virtual Integration. In [5] an overview of virtual integration methods with their current use in practice, as well their limitations is given. Already for ADAS with environmental perception safety cannot be shown economically only by real test drives due to the high complexity of the systems and tests. Finding the right balance between real test drives and virtual integration tests can be considered as an optimization problem with respect to the effort of building and parameterizing simulation models and the efficiency gain won by simulation techniques. Beside the repeatability and efficiency the additional value of simulation techniques is in particular the possibility to perform tests of the whole system already in early design phases. With regard to the V-model the four virtual integration techniques relevant for the development of ADAS are MIL, HIL, SIL and VIL, also compare [4]. The current limitations of virtual integration are stated as the simulation models a) do not always meet the necessary realism or b) do not have the real-time capability needed for virtual integration methods (or both).

Taxonomy for Testing of Advanced Driver Assistance Systems. In the survey [3] a taxonomy of approaches for testing advanced driver assistance systems is presented. This concerns different characterizations of test criteria and metrics to quantify the quality of observations or models. Further, different methods to determine the test reference (ground truth) are discussed i.e. either measurement-based, by simulation or by a mixture of both. Finally, the definition of test scenarios is regarded where actual tests are performed to be checked against the reference. In the present paper, we follow the suggested approach by comparing the virtual model with a reference, similar to [6], where the special case of vision based systems is considered.

Combining Design Time Testing and Runtime Monitoring. In [7] a scenario-based approach is presented that combines testing at design time and monitoring

during runtime of an ADAS. This allows to identify the set of relevant scenarios by simulation and thus reducing field testing to these instead of testing all possible scenarios, which would be infeasible for complex ADAS systems. Furthermore, missing scenarios identified at field tests may be fed back into the set of scenarios for design time simulation to improve test coverage.

Assigning Test Cases to Test Methods. A method for assigning test cases for automated driving functions to X-in-the-Loop test methods is proposed in [8]. The authors make use of their virtual modular test kit, which is a concept to systematically test automated driving functions in virtual environments. Its goal is to reduce the overall number of necessary tests by a systematic test case generation while keeping the test coverage at the same level. Depending on the test case different requirements arise concerning the set of applicable X-in-the-loop methods. The assignment method has two steps. First, the X-in-the-loop methods are characterized by a Kiviat diagram. On the z-axis of the diagram different assessment scales are plotted like quality of results, operational costs, etc. Second, the requirements of the test cases to the X-in-the-loop methods are represented in a Kiviat diagram as well. By matching the diagrams the set of applicable X-in-the-loop methods can be determined. By defining assessment functions describing the quality of the models, operational costs, etc. the best rated method with respect to the defined assessment functions can be identified.

3 Stochastic Methods for Splitting Simulation and Testing

Although simulation offers a thorough investigation and verification of a system under test, testing real components against real environments will always be part of the verification process to guarantee the possibility of a transfer of results obtained in a simulation environment to the deployment phase. To optimize the cost efficiency of the overall verification process reducing and shifting effort towards virtual simulation is of major interest. In this chapter we will present the quantitative basis for an optimal trade-off between real world tests and virtual simulations to achieve a desired level of dependability.

3.1 Preliminaries

As mentioned in the introduction, our approach relies on a probabilistic argument which aims at quantifying the degree to which we can guarantee that the system requirements are fulfilled by the system across all possible situations the system under test might encounter throughout its lifetime. As shown by Winner *et al.* [1], such guarantee can be obtained with purely physical tests. However, these physical tests can also be used to validate a surrogate model of the reality, which in turn can then also be used to verify a system against this model of reality. Before going into details of the approach we first fix the notation of the stochastic variables that we will use in the following.

By the **real system under test** (S_r) we understand the system under test as it will be implemented. Analogously to the **virtual model** (S_v) it receives an input (which could be either provided by a model or the real world) and generates a corresponding response. The **reality or ground truth world model** (\mathcal{W}) is considered to be the desired environment for the system under test. The reality can be observed via measurements from a reference system, which provides sequences of measurements. These traces (sequences of measurements) can be compared with the sequences of the **simulation model** (\mathcal{M}) which can generate traces of virtual inputs for a system under test. Based on the generated traces, two models or a model and its real-world counterpart can be exchangeable. In that case we say the model \mathcal{M} is a **valid** model of the real world \mathcal{W} , denoted by $\mathcal{M} \equiv_{\mathcal{R}} \mathcal{W}$ to check whether a system under test fulfills some **requirements** (\mathcal{R}). These are a set of logical formulae, which should be **satisfied** (\vdash) for all relevant situations of real world scenes in which the system under test operates.

Furthermore, we are looking at **samples drawn from the reality** (X^w). These are discrete sequences of measurements taken from a system equipped with a reference sensor system. This reference system is able to provide sequences of accurate measurements comparable to **samples generated from the co-simulation of the models** (X^s) which are sequences of virtual measurements of the co-simulation of the virtual environment and the system model.

Definition 1. We write $S_{\mathcal{W}} = (S_r, \mathcal{W})$ for the real system under test with input from the real world.. Analogously we write $S_{\mathcal{M}} = (S_v, \mathcal{M})$ for the virtual model with input generated from a simulation model.

Thus read e.g., $P(S_{\mathcal{W}} \vdash \mathcal{R})$ as: the probability that the real system under tests satisfies a set of requirements. If a quantity cannot be directly assessed but has to be inferred from other observations or measurements we annotate this with a hat symbol. For example, if we have no access to a probability p , but have a method to estimate this probability, we denote the estimate by \hat{p} .

For the verification of a system under design, we are interested in the probability that the designed system will satisfy all requirements when facing environments generated from reality, i.e., the true world model. This probability can be written in terms of conditional probabilities assuming a particular model used for simulation and the probability that this model is an accurate description of the real world. With the law of total probability we arrive at:

$$\begin{aligned}
 P(S_{\mathcal{W}} \vdash \mathcal{R}) &= P(S_{\mathcal{W}} \vdash \mathcal{R} \mid S_{\mathcal{M}} \equiv_{\mathcal{R}} S_{\mathcal{W}}) P(S_{\mathcal{M}} \equiv_{\mathcal{R}} S_{\mathcal{W}}) \\
 &\quad + P(S_{\mathcal{W}} \vdash \mathcal{R} \mid S_{\mathcal{M}} \not\equiv_{\mathcal{R}} S_{\mathcal{W}}) P(S_{\mathcal{M}} \not\equiv_{\mathcal{R}} S_{\mathcal{W}}) \\
 &\geq P(S_{\mathcal{W}} \vdash \mathcal{R} \mid S_{\mathcal{M}} \equiv_{\mathcal{R}} S_{\mathcal{W}}) P(S_{\mathcal{M}} \equiv_{\mathcal{R}} S_{\mathcal{W}}) \\
 &\approx P(S_{\mathcal{M}} \vdash \mathcal{R} \mid S_{\mathcal{M}} \equiv_{\mathcal{R}} S_{\mathcal{W}}) P(S_{\mathcal{M}} \equiv_{\mathcal{R}} S_{\mathcal{W}}).
 \end{aligned} \tag{1}$$

That is, we first split the probability that the real system under test in its desired environment will satisfy the requirements into two cases depending on whether composition of the model and the virtual environment can be regarded as a valid replacement. If this is indeed the case, we can replace the pair $S_{\mathcal{W}}$ with

its virtual counterpart. As a result we have split the overall verification effort into a part which can be evaluated purely in a virtual environment (first term in Eq. (1)) and a part which evaluates the validity of the virtual model.

3.2 Validity of a Virtual Model

Validating a virtual model against its real counterpart is a challenging task (see [3]). In Eq. (1), we deduced from the validity of a model that we can use the model as a replacement for the real system within the satisfaction of the requirements. However, there are different notions of validity. We could classify a model to be valid with respect to a particular environment, if such replacement is allowed for a particular requirement. The latter interpretation, for example, is the basis for determining a test-method (including the selection of a virtual model) according to the method described in [8]. In the present paper we would like to follow a more generic approach. That is, we would like to define a notion of validity such that the replacement of the virtual model is valid for all requirements.

For this to hold, we have to at least ensure that all sequences of measurements from the real world X^w could be generated within the virtual environment, i.e., finding corresponding X^s . Please note that we assume all traces to be discrete. To avoid that the virtual model dominantly explores part of its sample space, which are not possible to observe in the real world, a stronger notion of validity would also require that for all traces in the virtual model, there exists a corresponding trace (sequence of measurements) in the real world. Even if traces are possible to generate for both systems, virtual model and real world, it could happen that different kinds of traces are differently favoured. That is some traces might be more likely to be generated in the real world compared to the likelihood of generating them using the virtual model. To summarise, we have the three (increasingly stronger) notions of validity:

1. All sequences of real world observations are also possible within the virtual model
2. Additionally, for each possible sequence within the virtual model, there exists an identical sequence of measurements within the real world
3. For each sequence of measurements there exists an identical sequence within the respective other model. Additionally, the likelihood of generating such sequence is also equal.

For simplicity, we only consider the first notion of validity within this paper. It should also be noted that all notions can be further relaxed by not requiring the existence of an identical sequence, but the existence of a sequence which is close to the required one.

3.3 Splitting Simulation and Testing for Ubiquitous Requirements

Given these notations, we can formulate the following properties as conditional probabilities. These are modeled as probabilities as the models used to describe the environment and potentially for the system model as well are likely to contain

stochastic variables and hence the satisfaction of requirements is potentially also subject to this encoded variability. For example, with the abbreviations introduced above, we can write down the probability that the system model satisfies the requirements, given that the simulation model is an accurate description of the (needed aspects of the) true world model. Note that in this section we assume that the requirements can be validated both via simulation as well as via physical testing. That is, we assume that the environmental model used for simulation provides all necessary information to evaluate whether a single sample generated from the model satisfies the requirements.

Definition 2 (Satisfaction of requirements for a given simulation model). *Let $S_{\mathcal{M}}$ denote the virtual model of the system under test with input generated from a virtual model \mathcal{M} and \mathcal{R} denote the requirements we would like the system to satisfy. We write the conditional probability of the system satisfying the requirements under the assumption that the simulation model \mathcal{M} is an accurate description of the real world \mathcal{W} as:*

$$P_{\vdash}^M := P(S_{\mathcal{M}} \vdash \mathcal{R} \mid S_{\mathcal{M}} \equiv_{\mathcal{R}} S_{\mathcal{W}}). \quad (2)$$

Due to the potential stochastic variability encoded into the simulation model, this probability is a property of the simulation model. As simulation models are typically too complex to be analyzed symbolically, this probability cannot be calculated exactly but can be approximated or bounded by means of a statistical analysis. To this end, samples from the simulation model can be generated and estimates of the probability can be obtained. This is the main goal of simulation based verification. In order to rigorously quantify the level of certainty associated with such a verification process it is important to keep track of the sample uncertainty incurred by the simulation based verification.

Assume we have generated m samples X_1^s, \dots, X_m^s using the simulation model. For each of these samples we can test whether the requirements are satisfied for the particular trace, $X_i^s \vdash \mathcal{R}$. Based on these results, we can, for example, estimate the probability (2) by the relative frequency of the samples satisfying the requirements (ideally, all traces satisfy the requirements, i.e., the relative frequency will be 1). If we denote this estimate \hat{P}_{\vdash}^M , we can statistically bound the probability that this estimate will deviate from the true probability P_{\vdash}^M by more than any given ϵ_s . The resulting bound on the probability depends on three variables: the confidence δ_s , the accuracy ϵ_s and the number of samples m used for this estimate. If two of these are given the others can usually be calculated based on the other two (see below for some specific examples).

$$P_{X_1^s, \dots, X_m^s} \left(\left| \hat{P}_{\vdash}^M - P_{\vdash}^M \right| \geq \epsilon_s(\delta_s, m) \right) \leq \delta_s. \quad (3)$$

In words, such a formula bounds the likelihood of the results being a fluke (judged by the estimation being further than $\epsilon_s(\delta_s, m)$ from the true value apart) as a result of unlucky observed data. Here, we have written $\epsilon_s(\delta_s, m)$ to stress the fact that the accuracy ϵ can be calculated from the other two parameters δ_s, m .

Now we have to combine this probability with the probability that the model represents the relevant information sufficiently well. Here, we assume that we

represent all relevant information which is needed to answer whether the specified requirements are satisfied on a single trace basis. In other words, we say that the simulation model represents the ground truth model if a trace of observations of the real world is considered possible in the simulation model and testing this trace with respect to the requirements on both models leads to the same answer.

$$P_{\equiv_{\mathcal{R}}} := P(S_{\mathcal{M}} \equiv_{\mathcal{R}} S_{\mathcal{W}}). \tag{4}$$

As we do not have access to the mathematical description of the world model, we need to estimate this probability based on observations of the real world, similar as we have estimated (2) via sampling from the model. Again, we need to keep track of the residual uncertainty associated with such an empirical estimation procedure. Specifically, for n observations X_1^w, \dots, X_n^w of the real world, we have:

$$P_{X_1^w, \dots, X_n^w} \left(\left| \hat{P}_{\equiv_{\mathcal{R}}} - P_{\equiv_{\mathcal{R}}} \right| \geq \epsilon_w(\delta_w, n) \right) \leq \delta_w. \tag{5}$$

These different estimates can be used to bound the overall probability of interest (see Eq. (1)). In fact, for the first term in Eq. (1), as we have no access to the true probabilities, we can use their sample-based estimates from Eqs. (3) and (5) to obtain:

$$P(S_{\mathcal{W}} \vdash \mathcal{R}) \geq \left(\hat{P}_{\vdash}^M - \epsilon_s \right) \left(\hat{P}_{\equiv_{\mathcal{R}}} - \epsilon_w \right) \quad \text{with } P \geq (1 - \delta_s)(1 - \delta_w). \tag{6}$$

If all physical tests, i.e., observations of the behavior of the system, satisfy the requirements and could also have been generated by the simulation model both estimates $\hat{P}_{\vdash}^M, \hat{P}_{\equiv_{\mathcal{R}}}$ are 1. In this case the equation simplifies to:

$$P(S_{\mathcal{W}} \vdash \mathcal{R}) \geq (1 - \epsilon_s)(1 - \epsilon_w) \geq 1 - \epsilon_s - \epsilon_w \tag{7}$$

with $P \geq (1 - \delta_s)(1 - \delta_w)$.

For simplicity, we have omitted the dependence of δ_s, δ_w, m, n on the accuracies ϵ_w, ϵ_s . The above equation suggests that the effort to spend on either simulation or physical tests amount to the same contribution to the overall safety guarantee (satisfaction of the requirements). However, due to the multiplication of residual uncertainties, i.e., confidences δ_s, δ_w , we might want to allow for a smaller confidence in the simulation model thereby requiring a larger confidence in the simulation analysis while obtaining the same level of overall confidence and safety estimate. In other words, we can achieve the same safety guarantee with different splits between physical and simulation tests. This degree of freedom can therefore be exploited to obtain an optimal trade-off with respect to the resulting costs.

Assuming a fixed cost c_s for each simulation run and c_w for each physical test to validate the simulation model, we therefore can solve the following constrained optimization problem for a given overall confidence level X and a safety level Y :

$$\begin{aligned} \min_{n,m} (c_s m + c_w n) \quad & \text{s.t.} \\ (1 - \delta_s)(1 - \delta_w) \geq X \text{ and } (1 - \epsilon_s(\delta_s, m))(1 - \epsilon_w(\delta_w, n)) \geq Y. \end{aligned} \tag{8}$$

Although we optimize the costs within the above optimization problem, we only do so under the constraint that a certain level of safety has to be guaranteed. Such optimization problem can be solved (at least numerically) if the estimation accuracy functions ϵ_w, ϵ_s are given. For the particular situation in which we are aiming at estimating a probability - which is specified in terms of a binary indicator variable (satisfaction of the requirements) - we can use a Bernoulli bound to obtain a specific form of the accuracy functions, for example a Clopper-Pearson bound (see [9] and Sect. 3.5).

If the simulation model is only used for generating the environment of the system under test and is therefore independent of the system under test the physical tests to validate that model need be performed only once for a model. The model in turn can be used to verify more than one system without requiring additional physical tests for model validation provided that the samples that were generated from the model were generated for each system under test. However, if the model is allowed to change or adapt to the physical test data that has been acquired, validation of the model corresponds to bounding the prediction performance of a learning system, as the model *learns* from the physical test data. Although this is possible, the calculations are more involved and we therefore postpone this discussion.

3.4 Splitting Simulation and Testing Based on Type of Requirements

In the previous section, we assumed that all requirements can be tested either using simulation or physical tests. Additionally, we also measured the quality of a simulation model based on its ability to generate traces and leading to the same answer regarding the satisfaction of the requirements. In practice, there are certain requirements which are outside the scope of the simulation model. For instance, the model might not include certain variables within its representation that are relevant for some requirements. That is, we might have a model of the vehicle dynamics at hand, but would like to test a requirement which specifies how a route-planning component should work. In these situation Schuldts *et al.* [8] proposed a method to judge which type of test-method (for example HIL or MIL) should be applied. In particular, they also suggested that the quality of the provided simulation model should be taken into account when selecting a suitable test-method. Using the results from the previous sections, we can provide a quantitative measure which supports their method.

Also, we can use similar calculations as above to provide an overall measure in satisfying the desired requirement. Specifically, assume we have given two types of requirements $\mathcal{R}_1, \mathcal{R}_2$ each of which specifies the desired behaviour for different parts of the system under test. Assume further that we have two simulation models \mathcal{M}_1 and \mathcal{M}_2 , each modelling the respective part of the system under test and their respective inputs. Then we can write the overall probability of satisfying the requirements as

$$P(S_{\mathcal{W}} \vdash \mathcal{R}_1 \wedge S_{\mathcal{W}} \vdash \mathcal{R}_2) = P(S_{\mathcal{W}} \vdash \mathcal{R}_1 | S_{\mathcal{W}} \vdash \mathcal{R}_2) P(S_{\mathcal{W}} \vdash \mathcal{R}_2) \quad (9)$$

$$\stackrel{\text{ind.}}{=} P(S_{\mathcal{W}} \vdash \mathcal{R}_1) P(S_{\mathcal{W}} \vdash \mathcal{R}_2).$$

Here we have assumed that the satisfaction of the second requirement does not affect the satisfaction of the first one. If both requirements restrict different parts of the system under test, this might be reasonable, however, it should be noted that all components within the system under test are likely to be connected via a certain computation path. Therefore, the independence assumption might be too strong. Similar to Eq. (1), we can now resolve each of the remaining terms in Eq. (9) using the respective models.

If for one of the requirements there is no model available, we can simply perform physical tests to estimate the corresponding probability. In this case, assuming all tests have been passed, we can rewrite Eq. (7) to obtain

$$P(S_{\mathcal{W}} \vdash \mathcal{R}_1 \wedge S_{\mathcal{W}} \vdash \mathcal{R}_2) \geq (1 - \epsilon_s^1) (1 - \epsilon_w^1) (1 - \epsilon_w^2) \quad (10)$$

$$\text{with } P \geq (1 - \delta_s^1)(1 - \delta_w^1)(1 - \delta_w^2).$$

For the first requirement, we have $(1 - \epsilon_s^1) (1 - \epsilon_w^1)$ representing the accuracy of checking the satisfaction of the requirement times the probability of the model being valid. For the second requirement, we can omit the model validation probability, as we assume to perform real-world tests. Using such formulation, we can again optimise the costs under safety constraints. In the above formulation, we perform real-world tests for checking the validity of the model and checking the satisfaction of \mathcal{R}_2 . However, we can re-use the same real-world test for both objectives. Therefore, the number n of real-world tests within (8), can be used for both ϵ_w^1 and ϵ_w^2 .

3.5 Practical Considerations

In this section we investigate the potential of the approach as outlined in the previous sections from a more practical perspective. Applying the procedure outline in the previous section, one has to first set up a model for simulation then collecting independent observations of the real world, which allows to check whether these observations can also be generated by the simulation model, and finally performing the simulation-based tests. We therefore use the real-world observations only for validating the model here, although a double use, i.e., validating the model and checking requirements would be possible as well and would further strengthen the guarantees.

Validating a simulation model using a reference sensor system. In the previous sections we assumed that the validity of the simulation model can be checked on a single observation basis. In the simplest case, this can be achieved by verifying that a (sequence of) measurements can be reproduced within the virtual environment used as a simulation model. For example, the simulation model could consist of several modules integrated into a co-simulation platform. To be able to generate a simulation run, further parameters such as road topology,

behavior of traffic participants, etc., have to be specified within the co-simulation platform. By choosing a suitable set of these parameters, one can try to mimic the sequence of measurements. If the observed sequence can be reproduced, the necessary check can be considered passed. If all measurement data can be reproduced, the corresponding estimate of the probability that the virtual model is an accurate description of the real world $\hat{P}_{\equiv_{\mathcal{R}}}$ in Eqs. (5) and (7) is 1.

More precisely, one has shown that the virtual model is capable of reproducing the sensor measurements of the (potentially inaccurate) sensor setup used for recording. Therefore, if one aims at validating a system which should serve as a generator of ground truth data, one should use a sensor setup which can act as a reference, i.e., has the desired accuracy. With the help of an applied co-simulation platform, one could measure the (relative) positions of all objects and then reproduce the trajectories of all detected objects within the simulation. If the measurements also contain a visual component, one needs to show that the rendering procedure is capable of generating the recorded video sequence.

It should be noted, that the same procedure can also be used to validate components, such as sensor models, against their hardware counterparts. To this end, one would discretely measure pairs of signals, input and output signal, where input signals could be obtained via a reference sensor system and the output would be measured from the sensor one would like to model. To validate the virtual model of the sensor it would be checked whether it can reproduce all observed sequence of input-output pairs. In fact, having validated a sensor model would also mean that all inaccuracies of the sensor are captured within the model. By combining different validated models for components, one can then also conclude that the combined model is validated. However, the confidence in the combined model is reduced as the overall confidence is given by the product of the confidences for the individual components.

Exemplary Optimization for a Cost-Efficient Trade-Off. To evaluate the practical impact and associated costs (savings) we calculate the optimal trade-off between simulation and tests as outlined in Eq. (8). To this end, we assume that both physical tests as well as simulation based tests have not revealed any violation of the requirements and model validation, respectively. Otherwise we assume that the underlying problem has been addressed and the corresponding tests have been successfully repeated. For the costs of a physical test we assume here $10 \frac{\text{€}}{\text{km}}$. Relative to these costs we assume a virtual kilometer within a simulation environment to cost a fraction of 0.01, that is here $0.1 \frac{\text{€}}{\text{km}}$. Note that these values are only illustrative figures, but are easily replaceable by more accurate values. For the desired overall confidence we are using 0.99 and our desired accuracy is set at $1 - 1.375 \cdot 10^{-7}$ which roughly corresponds to half of the current empirical probability of no accident per km^2 .

As mentioned in Sect. 3.3, we can use the Clopper-Pearson confidence interval to determine the accuracy ϵ_W, ϵ_S for the simulation and physical test specific accuracies. Specifically, we have for both ϵ_W, ϵ_S :

² www.adac.de/_mmm/pdf/statistik.7.1_unfallrisiko_42782.pdf.

$$\epsilon_W(\delta_W, k) = 1 - (\delta_W)^{\frac{1}{k}}, \quad \epsilon_S(\delta_S, k) = 1 - (\delta_S)^{\frac{1}{k}}. \quad (11)$$

Therefore, in the case of no violations of any requirements, we have:

$$\begin{aligned} \min_{n,m} (c_S m + c_W n) \quad & s.t. \\ (1 - \delta_S)(1 - \delta_W) \geq 0.99 \quad & \wedge \quad (\delta_S)^{\frac{1}{m}} (\delta_W)^{\frac{1}{n}} \geq 1 - 1.375 \cdot 10^{-7}. \end{aligned} \quad (12)$$

As the costs for simulation and physical tests might vary between different systems, models, and companies, we illustrate the achievable trade-off for a range of possible costs. We plotted the results in Fig. 1.

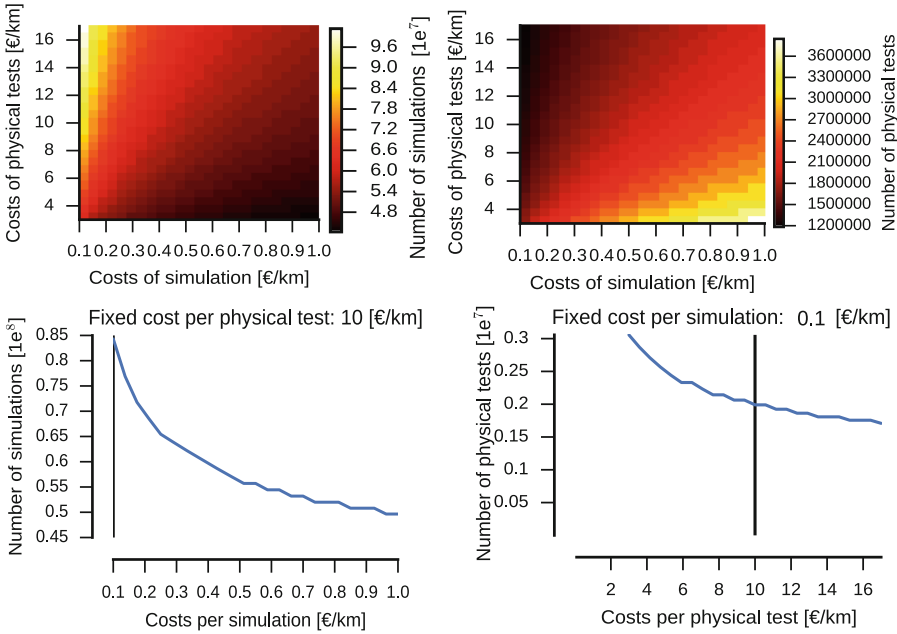


Fig. 1. The achievable trade-off between simulation-based and physical tests.

Here the upper two three dimensional diagrams belong together, meaning that if you choose a fixed cost for a physical test and a fixed cost per simulation you can estimate the number of needed simulations (left diagram) and physical tests (right diagram) for an optimal trade-off from the color depicted in the diagrams. In the lower two diagrams we each fix one dimension of the diagrams above to observe how the needed number of simulations/physical tests changes compared to the cost per simulation/physical test. The intersection with the straight lines drawn into the diagram thus mark the optimal trade-off from the example given above.

It can be seen from the lower panel in Fig. 1 that if a physical test is more costly the optimisation procedure will increase the number of simulations, as

expected. However, the overall number of tests (both physical and virtual) is still quite high, which is due to the high safety targets and no additional assumptions.

Comparison with Purely Physical Testing. We also investigated the potential savings in costs using this approach compared to a setting using only purely physical tests. In Fig. 2 we plotted the relative savings in Euros when comparing a setting in which only real-world tests are performed to check the requirements to an optimal split. Thus this diagram does not say anything about the amount of tests necessary but only about the relative savings of performing an optimal split compared to pure physical tests. Although we only use real-world observations for model validation and not for testing requirements, the potential savings amount to over 90% of the costs associated with the several hundred million kilometers that were estimated to be sufficient using only real-world testing [1]. The savings are particularly dramatic in settings where the costs of a physical test are much higher than for an individual virtual test (upper left corner in Fig. 2).

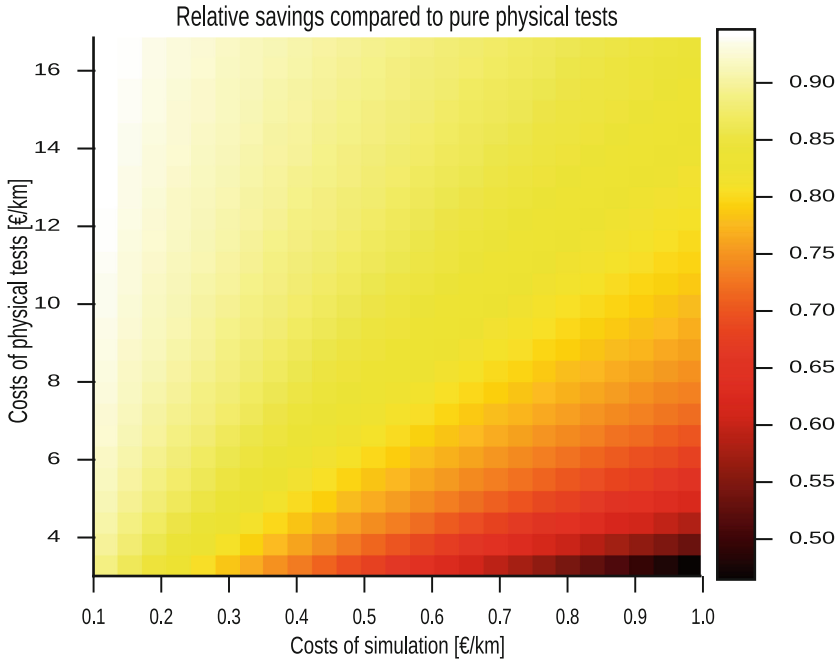


Fig. 2. Comparison of the optimal split and a purely physical testing setting.

4 Conclusion

In this paper, we focused on the foundations for a quantitative analysis of splitting test-cases into virtual and physical tests, thereby taking into account the

difference in costs for these two types. Although we did not use any further assumptions on the regularity of tests (e.g., nearby scenarios are more likely to produce similar satisfaction results with respect to requirements), the results show that the total savings in costs can be quite substantial ($\approx 90\%$ in the given example) when compared to the setting of testing all requirements purely in real situations. Additionally, such savings are likely to be multiplied, as the models used for simulations can be re-used, once they have been validated using the physical tests. Even when we have made slight changes in the simulation model we could include a prior belief about the quality of the simulation into our approach. With the help of a prior quality belief we could reduce the needed real-world observations even further.

As mentioned in the introduction, the results presented in this paper are meant to be of generic nature. In fact, from a very abstract perspective, the overall test-process is unchanged, but incorporates the quality metric as proposed by Winner *et al.* [1] and can be integrated into methods for selecting appropriate test-methods such as [8]. Once a decomposition of the system (e.g. via [10]) has been identified in terms of which parts of the system can be safely replaced by virtual counterparts, this also provides guidance for a X-in-the-loop test setup. Furthermore, if one can identify critical scenarios either as done in PEGASUS³ or simulation based as in [11] one can further reduce the overall needed effort.

The main challenge for our approach to hold is that a tool for model validation is missing. Given a model: how can we find out whether a sequence of measurements could be reproduced with this given model? This becomes even more difficult if we assume a different notion of validity as discussed in Sect. 3.2. To use simulation for the verification of highly automated driving functions we need to be able to decide how valid the simulation is. What are the important aspects? Which deviations from reality are allowed? Thus, rigorous model validation is needed to bring such systems into the market.

Moreover, we have seen that a substantial amount of simulation is necessary. Such simulations have a high computational complexity. Thus one would wish for possibilities to further reduce the simulation effort, e.g. with the use of Multilevel Monte Carlo Methods [12].

Although the results presented in this paper are of quite general nature, we believe that the quantification and necessary formalisation of the different aspects might lead to not only a more efficient test process but also to a safer overall system.

Acknowledgments. This study was partially supported and financed by Opel Automobile within the context of PEGASUS (Project for the Establishment of Generally Accepted quality criteria, tools and methods as well as Scenarios and Situations for the release of highly-automated driving functions), a project funded by the German Federal Ministry for Economic Affairs and Energy.

³ <https://www.pegasusprojekt.de/en/about-PEGASUS>.

References

1. Winner, H.: Quo vadis, FAS? In: Winner, H., Hakuli, S., Lotz, F., Singer, C. (eds.) *Handbuch Fahrerassistenzsysteme. ATZ/MTZ-Fachbuch*, pp. 1167–1186. Springer, Wiesbaden (2015). https://doi.org/10.1007/978-3-658-05734-3_62
2. Kalra, N., Paddock, S.M.: Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *RAND Corp.* **94**, 182–193 (2016)
3. Stellet, J.E., Zofka, M.R., Schumacher, J., Schamm, T., Niewels, F., Zollner, J.M.: Testing of advanced driver assistance towards automated driving: a survey and taxonomy on existing approaches and open questions. In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1455–1462. IEEE (2015)
4. Hallerbach, S., Eberle, U., Köster, F.: Absicherungs- und Bewertungsmethoden für kooperative hochautomatisierte Fahrzeuge. In: *AAET 2017*, Braunschweig (2017) 369
5. Hakuli, S., Krug, M.: Virtuelle integration. In: Winner, H., Hakuli, S., Lotz, F., Singer, C. (eds.) *Handbuch Fahrerassistenzsysteme. A*, pp. 125–138. Springer, Wiesbaden (2015). https://doi.org/10.1007/978-3-658-05734-3_8
6. Nentwig, M.: Untersuchungen zur Anwendung von computergenerierten Kamerabildern für die Entwicklung und den Test von Fahrerassistenzsystemen. Ph.D. thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (2014)
7. Mauritz, M., Rausch, A., Schaefer, I.: Dependable ADAS by combining design time testing and runtime monitoring. In: *10th International Symposium on Formal Methods, FORMS/FORMAT 2014*, pp. 28–37 (2014)
8. Schuldt, F., Menzel, T., Maurer, M.: Eine Methode für die Zuordnung von Testfällen für automatisierte Fahrfunktionen auf X-in-the-Loop Verfahren im modularen virtuellen Testbaukasten. In: *10. Workshop Fahrerassistenzsysteme*, p. 171 (2015)
9. Clopper, C.J., Pearson, E.S.: The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika* **26**(4), 404–413 (1934)
10. Ammersbach, C., Winner, H.: Functional decomposition: an approach to reduce the approval effort for highly automated driving. In: *8. Tagung Fahrerassistenz* (2017)
11. Hallerbach, S., Xia, Y., Eberle, U., Koester, F.: Simulation-based identification of critical scenarios for cooperative and automated vehicles. In: *SAE International WCX World Congress Experience*, April 2018
12. Giles, M.B.: Multi-level Monte Carlo path simulation. *Oper. Res.* **56**(3), 607–617 (2008)

Multi-Concern Assurance



Roadblocks on the Highway to Secure Cars: An Exploratory Survey on the Current Safety and Security Practice of the Automotive Industry

Michael Huber^{1(✉)}, Michael Brunner¹, Clemens Sauerwein¹, Carmen Carlan², and Ruth Breu¹

¹ Department of Computer Science, University of Innsbruck, Innsbruck, Austria
{michael.huber,michael.brunner,clemens.sauerwein,ruth.breu}@uibk.ac.at

² fortiss GmbH, Munich, Germany
carlan@fortiss.org

Abstract. With various advances in technology, cars evolved to highly interconnected and complex Cyber-Physical Systems. Due to this development, the security of involved components and systems needs to be addressed in a rigorous way. The resulting necessity of combining safety and security aspects during the development processes has proven to be non-trivial due to the high interference between these aspects and their respective treatment. This paper discusses the results of an exploratory survey on how organizations from the automotive industry in the Euroregion tackle the challenge of integrating safety and security aspects during system development. The observed state of practice shows that there are significant deficits in the integration of both domains. The results of the exploratory survey enabled us to identify the most common challenges of realizing an integrated approach in a practical setting and discuss implications for future research.

Keywords: Automotive · Cyber-Physical Systems · Safety · Security Integration · Industrial survey

1 Introduction

The upcoming generation of Cyber-Physical Systems (CPSs) will be characterized by fragmentation, heterogeneity, short release cycles, cross-organizational nature and safety criticality [6]. Due to technological advances, safety-critical CPSs like modern vehicles become security-sensitive, with high interference between safety and security requirements that need to be addressed [1, 17, 22]. These – and many more – new conditions pose a specific challenge for the development and ongoing operation of CPSs: The integrated treatment of safety and security aspects [10]. Within this paper, the definition of safety and security is in accordance with [7], that is, safety is concerned with protecting valuable assets

by preventing, detecting and properly reacting to accidental harm. Security, in contrast, is concerned with protecting these assets by preventing, detecting and properly reacting to malicious harm [7]. The automotive domain is particularly affected, since innovation-related challenges are transforming the traditional automotive industry [11]. Unlike the nuclear or avionics industry, where certification of products or systems usually follows a process-oriented approach (i.e., a system is considered safe when developed in accordance to processes mandated by industry standards), manufacturers in the automotive industry need to show that they achieved certain safety objectives using safety assurance cases as required by the ISO 26262 [12] standard. Assurance cases are an established method within certification processes of embedded systems. They trace safety goals down to safety solutions and provide arguments supported by evidence for the satisfaction of relevant types of system properties within a certain context and under certain assumptions [3, 14]. When assurance cases offer argumentation and evidence for the correct implementation of a system's safety requirements, they are called safety cases.

In recent years, a considerable amount of research has been done on safety and security assurance in the automotive domain [10, 19, 27]. To the best of our knowledge, the perspective of industry regarding this matter has hardly been investigated. In order to address this gap, we explored how industry deals with potentially interrelated safety and security aspects during development of CPSs and components.

We conducted an exploratory survey in the automotive domain with organizations which have their headquarter in the Euroregion. By means of in-depth interviews with system development experts, we were able to observe the current state of practice and prevalent challenges. In addition, we evaluated our previously proposed conceptual model [4] for safety and security aspects of CPSs.

The remainder of this paper is structured as follows: Sect. 2 describes the applied research methodology. Section 3 presents the results of the survey and discusses threats to validity. Section 4 presents key findings from the survey, their implications for future research and motivates the use of a holistic model. Section 5 presents related work. Finally, Sect. 6 concludes the paper and provides an outlook on future work.

2 Research Methodology

The main goal of our research is to better understand how the challenge of treating safety and security assurance in an integrated manner during the development and operation of CPSs is confronted by the automotive domain. Our research objective is to analyze the current real-world difficulties of realizing an integrated approach in order to elicit challenges that occur in practical settings. In the pursuit of achieving this objective, we investigate the current state of practice by answering the following research questions:

RQ1: What is the state of practice to unify or synchronize methods and processes of the safety and security domain?

- RQ2:** How is the safety and security domain differentiated regarding definitions, requirements, processes and utilized tools?
- RQ3:** How are interdependencies between the safety and security domain identified and treated?

The remainder of this section is dedicated to the design of the conducted exploratory survey, the applied procedure for data collection and a thorough description of survey participants and their selection procedure.

2.1 Survey Design

A survey is a comprehensive research method for collecting information to describe, compare, or explain knowledge and behavior [15]. In order to observe the relevant aspects regarding the safety and security assurance of CPSs in a practical setting, we followed the paradigm of exploratory research. We collected data using expert interviews. This allowed for a flexible research design and quick adaptation to changes in the observed phenomenon [30].

The survey design followed a three-step process with (1) an initial survey design proposal, (2) a subsequent refinement of central questions and finally, (3) a pilot interview to validate the survey design in a practical setting. In order to structure and define the initial survey design, we utilized a conceptual model which was previously developed to document security and safety requirements in an integrated manner to support certification processes during design and run-time phases of CPSs [4]. This model unifies relevant documentation artifacts from four main domains: *Requirements Engineering*, *System Modeling*, *Risk Assessment*, and *Evidence Documentation*. Requirements are modeled in a hierarchical fashion distinguishing between functional and non-functional requirements (primarily concerning safety and security aspects). System Modeling is represented as the interrelated composition of hardware and software components. Risk Assessment is primarily derived from vulnerabilities and corresponding threats. Evidence Documentation is modeled based on various kinds of assurance artifacts.

Guided by the structure of this model, we derived questions for the survey that aligned with our research questions. The initial proposal, as well as the final survey, comprised a single key question and three sets of additional structural questions to guide the interview process. All questions were formulated in German and later translated into English for interviewees not speaking German. The key question was closely related to our research objective and formulated to approach the subject as broadly as possible in order to prevent the introduction of an initial bias to the interview. Subsequently we defined the following key question:

“What are the three main challenges for an integrated consideration of security and safety aspects in the development and operation of cyber-physical systems in the automotive domain?”

All structural questions were intended to pinpoint and further refine our understanding of the participants’ response to the key question and helped us answer our research questions (i.e., RQ1, RQ2 and RQ3). The aforementioned conceptual model was used as a basis for the structural questions and as a visual representation of potentially relevant structures, processes, and interfaces in order to encourage discussions during the interviews. Figure 1 illustrates the final survey design and all relevant documentation artifacts that were prepared.

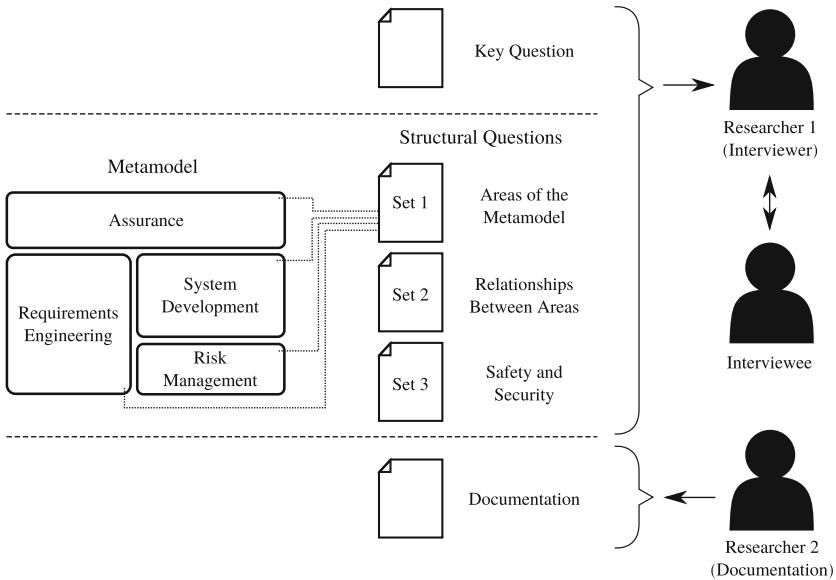


Fig. 1. Structure of the survey

The first set of structural questions covered the *four areas of the conceptual model independently*, addressing activities involving requirements engineering, system development, risk management, and evidence documentation. These questions were aimed at identifying involved stakeholders, utilized tools, relevant data sources and the most prominent challenges occurring in the respective area. For each of the four areas, the same set of questions was used. For example, we asked “*What tools are used for requirements engineering?*” and “*What stakeholders are involved in the assessment of risks?*”.

The second set of structural questions was concerned with the *relationships between the different areas of the conceptual model* considering exchanged information, nature of communication, utilized tools, methods, processes and the most prominent challenges. For example, we asked “*How is information between stakeholders performing risk analysis and stakeholders responsible for requirements engineering exchanged?*” and “*Do you use any software solutions to facilitate this communication?*”.

The third set of structural questions were aimed at understanding the respective organization's *differentiation between safety and security requirements* concerning definition, methods, processes, tools, their identification, assessment and implementation including interactions with stakeholders. For example, we asked "How do you differentiate between safety and security requirements?" and "Do you use different methods/processes/tools for the identification of safety and security requirements?".

An initial pilot interview was used to test and validate our survey design with an expert from the automotive industry. The expert was contacted within the context of the research project SALSA¹ and its industrial network. Criteria for the selection of the expert are described in Sect. 2.4. In a follow-up meeting the pilot expert gave feedback regarding the content and structure of the interview and its delivery. The pilot expert confirmed the alignment of the survey with our research objective. Subsequent changes to the survey design involved time management only.

2.2 Data Collection

All interviews were conducted face-to-face, allowing for a more complex interaction between the interviewers and the participants. Each of the interviews was conducted by two researchers – one taking notes and one interviewing the participants as depicted in Fig. 1. After an initial presentation of the general procedure, purpose, and specific goal of the survey, the participants were asked if they consented to the recording of the interview in audio format. With participants who did not give consent, the process of documenting the interview was conducted in a handwritten format only. This was followed by collecting quantitative data about the participants, their roles and organizations. Finally, the key question was discussed with the help of the aforementioned structural questions and the conceptual model. All questions and illustrations of the model were available in printed format, logically grouped and presented as needed during the interviews. If the interviewee was able to directly respond to the key question, the interviewer chose to discuss and analyze challenges with the appropriate sets of structural questions and the conceptual model in order to gain a detailed understanding. If the interviewee was unable to directly state any challenges, the interviewer presented the conceptual model and followed the first set of structural questions in order to detect problematic areas through deviating data sources, responsibilities, tools or processes. The time required for individual interviews averaged 60 min with only minor deviations. The study was conducted within a time frame of 8 months, starting in February 2017 and ending in September 2017.

2.3 Analysis Procedure

Directly after conducting the interview, the two researchers discussed and documented the obtained data during a debriefing followed by writing a summary

¹ <https://salsa.q-e.at/> (Accessed: 02/12/2018).

for each interview. Audio recordings were transcribed and all produced documentation was further analyzed. We followed the guidelines set out by Mayring et al. [21] to produce qualitative summaries from the collected data in order to extract the facets relevant to our research questions [5]. We grouped the results inductively by reducing, paraphrasing, and generalizing relevant text passages.

2.4 Participants

In the context of the research project SALSA and its industrial network, the participants were contacted by e-mail and provided with a brief summary of the survey goals and procedure. Criteria for the selection of participants were (1) employment in a leading role in development and operation of CPSs in the automotive domain or a closely related safety-critical domain, (2) at least 4 years of professional experience and (3) employment at a company with at least 150 employees. Upon agreement to an appointment, the interviews were conducted on site. All participants offered to take part in the interview voluntarily. An overview of all participants is presented in Table 1. The majority of participants held a degree in Computer Science or another engineering discipline. The encountered roles of the participants within the organizations were predominantly titled *Safety Team Leader* or *Safety Manager* with an average of 17 years of experience in their field. All participants represented companies based in, or having their headquarters in the Euroregion. While the majority of these companies were active in the automotive domain, one was active in avionics. Three of these companies were considered small companies with 150 to 1000 employees and the remainder large companies with more than 1000 employees.

Table 1. Participants of the survey

ID	Operational role	Education	Experience	Type of organization	# Employees
A	Safety team leader	University	25 years	Supplier	150–1000
B	Expert SW	University	10 years	OEM	>1000
C	Safety manager	University	4 years	Supplier	150–1000
D	System engineer	Technical apprenticeship	24 years	Consulting	>1000
E	Director safety management	University	25 years	Supplier	>1000
F	Safety team leader	University	11 years	Supplier	>1000
G	Team leader	University	21 years	Supplier	150–1000
H	Chief expert safety, security, reliability, availability	University	22 years	OEM and supplier	>1000

3 Results

In this section we present the results of our survey by addressing the research questions depicted in Sect. 2. Subsequently, we highlight identified challenges and discuss threats to validity.

3.1 State of Practice Regarding the Unification of the Safety and Security Domain

We encountered three distinct cases concerning the integration of the safety and security domains. Interviewees G and H followed an (1) integrated approach where both domains were considered from the beginning of the system life cycle. All remaining interviewees A-F either (2) treated security as an afterthought, where existing concepts, functionalities, and components were analyzed for their security relevance or (3) did not consider the security domain.

Participants G and H stated that their companies treat security and safety requirements in an integrated way. The company interviewee G works for employs dedicated security teams. The company follows a system development process which involves these security teams from initial phases. Furthermore, there exists a set of internal security guidelines and specifications, however, their source, content, and application, as well as the security teams' interactions with the safety processes or other entities during development or operation were kept confidential by the interviewee. There are dedicated security teams in the company participant H works for also. Processes like HAZOP [16] in the safety domain and threat analysis in the security domain were said to be executed in parallel with defined points of synchronization to treat interdependencies between both domains. Participant H mentioned the organization's approach to be in an early stage, thus, when and how to synchronize both domains was not clearly stated. The approach addresses economical interests by keeping adaptations or changes to established processes of the safety domain to a minimum with processes of the security domain being decoupled, at least during the elicitation of requirements.

When treating security as an afterthought only, a truly bidirectional consideration of the influences of the safety and security domains of the System Under Development (SUD), including treatment, is only possible with considerable and often economically unviable effort, as stated by participant A and D. In case of occurring conflicts between both domains, late treatments might entail costly changes to prevalent system designs. As stated by participant D, this is due to the potential of the safety and security considerations to influence the architecture of a system. Participant A compared this situation to requirements which are supplied by the customer in a late phase of the system life cycle and require changes to the prevalent system design. This circumstance was stated to not be economically desirable. However, participant A and H pointed out that in some cases knowledge of the system which is only available in later phases of the system life cycle or only within the context of the system of a subordinate organization of the supply chain is required. The statement emphasizes the necessity

to synchronize the safety and security life cycles within the overall system life cycle and to properly orchestrate, distribute and communicate safety and security objectives which span multiple organizations of a supply chain.

Classical software engineering problems were observed as a challenge complicating the integration of security aspects. Mastering the current complexity of existing work flows was a frequently mentioned problem. The amount of artifacts accumulating during development processes (for example processes compliant with the V-model XT [8]) were stated to be difficult to manage. Artifacts explicitly named by the participants were: *Process documents* required by ISO 26262 [12] which are necessary for subsequent processes and the establishment of evidence traces through assurance procedures. *Requirements*, imposed on development processes and the SUD which originate from standards, internal documents, and customers. The volume and complex structure of these artifacts were stated to result in difficulties in *traceability*, a recognized problem in software engineering [13].

The problem to establish links and traces throughout the aforementioned artifacts is further exacerbated by the *heterogeneous and diverse tool landscape*, as stated by all participants. Besides popular software solutions, like IBM Rational DOORS and PTC-Integrity for e.g. requirements engineering purposes, Enterprise Architect and Visio for e.g. system development and Microsoft Word and Microsoft Excel for e.g. risk management and assurance purposes, all participants use a variety of proprietary tools developed by the companies. Their purpose is to accommodate for missing functionalities, provide adaptations in highly customized processes and to support intercommunication between different tools. The result is a *complex tool chain*, sometimes unique to even a particular project within an organization. This has consequences for the integration of safety and security as well due to rigid and time intensive *change management*. Participant G stated a case of obsolescence management where the removal of a tool from the tool chain made subsequent changes or any kind of maintenance impossible.

Another frequently mentioned challenge were *economical aspects*. It is well known that the costs of software engineering projects may rapidly escalate [2]. As stated by all participants, the amount and quality of treating security properties will always be limited by available resources within a project's budget and prioritized by the severity of consequences.

3.2 Differences Between the Safety and Security Domain

All interviewees exhibited a uniform understanding of the distinction between the two types of system qualities, citing generally accepted definitions for the safety and security domains [7], respectively. However, concerning the requirements engineering domain, we observed no distinction between safety and security requirements in conducted processes or utilized software solutions for participants A-F, who treat security as an afterthought. As an example, interviewee A described the combined administration of safety and security requirements in the software solution *PTC Integrity* where security requirements are assigned

the Automotive Safety Integrity Level (ASIL) of Quality Management (QM). This is normally used to declare the risk associated with a safety requirement as not being unreasonable and therefore would not require any dedicated safety treatment as declared in the ISO 26262 standard [12]. The definition of safety requirements was stated to be conducted in accordance with the ISO 26262 standard, whereas the origin of security requirements was limited to requirements imposed by customers. As for the combined approach depicted by interviewee H, there was no distinction between requirements from both domains after their definition, except for testing procedures. The definition of security requirements was conducted within a threat analysis which was decoupled from the definition of safety requirements.

Regarding risk assessment, no safety and security co-analysis was mentioned to be applied by participants A–H. Furthermore, all interviewees acknowledged the fact that while they are able to rely on years of experience, standards, and guidelines in the safety domain, they are unable to do so in the security domain. This holds especially true for risk assessment, as stated by many participants. Interviewees A, C, and F stated that they are not obliged to comply with any security standard and thus security problems are only dealt with if the customer demands it and if it is within the limit of the project budget.

The difference between both domains on a process level was stressed by participants A, E, and G. One example given by interviewee A was that while tasks concerning the safety assurance of a vehicle are completed by the Start Of Production (SOP), the scope of the security domain extends into the operational phase where new security incidents have to be dealt with until decommission. Furthermore, due to the nature of security, the time frame in which a security measure remains effective is unpredictable. This contradicts the scope and resource allocation of the classical safety domain. Interviewees A, E, and G gave this circumstance as a reason for why prevalent safety processes are unfit to deal with security properties of a system.

Interviewees A, C, and E mentioned that there are no dedicated employees for the security domain, even when security issues are taken into account and addressed. These responsibilities are integrated into roles like safety managers, system engineers, or system architects.

3.3 Elicitation and Treatment of Interdependencies Between the Safety and Security Domain

We encountered two organizations, G and H, that follow an integrated approach. Both representatives of these organizations described the classification of occurring interdependencies between the safety and security domain as published in [18], namely: *conditional*, *reinforcement*, *antagonism*, and *independence*. Due to confidentiality concerns, we are unable to provide details about the integrated approach followed by participant G. As for participant H, the elicitation and treatment of interdependencies between safety and security requirements is conducted within defined points in the system life cycle, where both domains were

synchronized. The interviewee stated the approach to be in an early stage. Challenges were said to be (1) the meaningful definition of points in the system life cycle where it makes sense to jointly consider artifacts of both domains and (2) develop efficient, holistic and systematic approaches for the elicitation and resolution of these interdependencies within the points of synchronization. Participant H stated that their approach utilizes the concept of risk as a common ground between the safety and security domains in order to harmonize processes for the mitigation of risk. In order to improve the maturity of their approach, processes and methods within these synchronization points were stated to be the main focus of current internal research.

Participant E treats conflicting safety and security requirements by conducting risk assessments in order to determine and prioritize requirements which have more severe consequences. No further method for the elicitation or treatment of interdependencies between the safety and security domain were encountered in the course of the survey.

3.4 Identified Challenges

Concerning our research objective, we identified the following challenges for an integrated consideration of security and safety aspects in the development and operation of CPSs in the automotive domain: (1) Coping with the complexity of prevalent development processes and overcoming traceability issues to enable appropriate change management and thus timely responses to security incidents. (2) Dealing with economic limitations regarding the increased complexity due to interdependencies between domains, the extended time frame in which security has to be treated and the possibly, timely restricted, viability of measures taken. (3) Dealing with the current lack of experience, standards, and guidelines concerning the combination of the safety and the security domain.

3.5 Threats to Validity

Our survey might be limited by certain threats to validity that we are aware of and, to the best of our knowledge, counteracted. The following argumentation is based on the guidelines set out by Runeson et al. [26].

Concerning construct validity, a major objective of the survey was to understand the participants' respective understanding and practice of the subject under investigation. We argue that the nature of our survey inherently counters threats to construct validity. Furthermore, in order to overcome limitations regarding language barriers, we offered interviews in English and German language. The interviews were always moderated by a researcher proficient in the interview language, all handouts, interview guidelines, and questions have been carefully translated and double-checked by native speakers.

Threats to internal validity were avoided by peer debriefing. Concerning external validity, our survey is highly focused and can not be generalized to other domains without considering potential differences.

In order to counter threats to reliability, we avoided influences of moderators through a predefined protocol including a preset pool of questions and dedicated interview guidelines. Moreover, all interviews were conducted by two researchers with fixed roles which were asking the participant questions and documenting the course of the interview, respectively. We further avoided influencing the participants before interviews by only providing broadly formulated contextual informations beforehand.

In order to counteract a biased selection of study participants, we selected interviewees based on criteria as described in Sect. 2.4. Finally, limitations from biased opinion of interviewees were avoided by comparing the transcripts and results of different interviews.

4 Discussion

In this section we derive four key findings from the previously presented answers to the research questions alongside their implications for future research. The section concludes by motivating the use of a holistic model.

We observed that (KF1) *the majority of organizations not actively take interdependencies between safety and security requirements into account*. The majority of participants stated that they do not follow an integrated approach and treat security only if explicitly requested. The current state of practice was claimed to be due to the novelty of the security domain within the automotive industry, lack of standards, guidance and experience, and economic limitations. Further research is necessary in order to be able to synchronize the safety and security life cycle, facilitating efficient and holistic elicitation and treatment of interdependencies.

In addition, (KF2) *prevalent problems concerning complexity, traceability, change management and availability of recourses complicate the integration of security*. The most common consequences inherent to the complexity faced in prevalent development processes are difficulties in traceability and the resulting inefficient change management. The average time to re-certify a system as a consequence of applied changes, due to a security incident, was stated to be too long for the volatile security domain. Future research needs to investigate how to reduce and/or manage the complexity of prevalent development processes in order to facilitate traceability and change management which is applicable for the security domain. Economic limitations further hamper the integration of security, especially maintaining traceability for effective change management is expensive [13]. Developed approaches need to be economically viable, despite the extended time frame in which security has to be treated, the elicitation and resolution of conflicts between requirements and the possibly time-restricted viability of measures taken to mitigate risks.

Participants stated that (KF3) *objectives of the security domain, as well as the safety domain span across multiple organizations*. Further research needs to investigate how to orchestrate, distribute and communicate responsibilities concerning these objectives within an inter-organizational context in order to

facilitate synchronization of the safety and security life cycle regarding processes and process artifacts within the system life cycle of a single organization.

We observed a (KF4) *uniform understanding and general awareness concerning the differences between the safety and security domain*. All interviewees cited generally accepted definitions for the safety and security domain, according to [24]. Participants following an integrated approach described the classification of occurring interdependencies between the safety and security domain as published in [18].

The challenges identified in Sect. 3.4 and the key findings presented above emphasize the necessity for a holistic model which unifies documentation artifacts, e.g. process documents of the system life cycle, in order to reduce complexity and facilitate efficient change management. Our conceptual model [4], which was validated during the course of the survey, was perceived as correct and suitable by interview partners. The model unifies relevant documentation artifacts from requirements engineering, system modeling, risk assessment and evidence documentation. It further establishes dependencies between documentation artifacts of different areas (e.g., between individual requirements, the system components they are defined for, the associated risks and available evidence showing the correct implementation of said requirements). The model constitutes a base for future research by enabling cross-domain documentation of safety and security requirements, and unifying design- and runtime aspects while supporting (re-)certification in accordance with prevalent security and safety standards.

5 Related Work

In recent years, the integrated handling of safety and security has gained more and more interest in the research community. While the research community is concerned with the importance of integrating safety and security and proposes various approaches [18], there are no insights into how this problem is currently treated in industrial practice. To the best of our knowledge, no survey has been performed regarding the integrated consideration of safety and security for CPSs in the automotive industry.

Kriaa et al. [18] provide an overview of a number of industry reports on approaches integrating safety and security treatment. While their work shows that various industries are interested in an integrated treatment of the safety and security domains, our survey focuses on the automotive industry and identifies real world challenges which prohibit a trivial integration of these approaches into prevalent development processes. Glas et al. [10] investigate the integration of the safety and security domain by discussing conflicts between safety and security mechanisms, whereas we explore the perspective of the industry in order to elicit challenges emerging from the current state of practice as perceived by representatives of organizations from the automotive industry. An industrial survey conducted in the area of safety engineering by Jose Luis de la Vara et al. [29] gives an overview on practices for safety evidence change impact analysis. Notander et al. [23] report on a survey regarding challenges in

the development of safety-critical systems. Martins et al. [20] conducted expert interviews and studied literature concerning requirements engineering for safety-critical systems. Ray et al. [25] discuss the current state of practice in automotive security architecture, investigating trade-offs between security countermeasures, real-time requirements, and in-field configurability needs. Sojka et al. [28] conducted a case study on testing and validating safety- and security-related properties of control software in the AUTOSAR [9] architecture. They show that the combination of procedures from the safety and security domain can bring economic benefits.

6 Conclusion

We conducted a survey of experts in the automotive domain in order to gain an understanding of real-world challenges occurring when combining safety and security for CPSs during development and operation. We observed significant deficiencies in the integration of both domains. Identified challenges are: (1) Coping with the complexity of prevalent development processes and its consequences, (2) dealing with economic limitations and (3) the current lack of experience, standards and guidelines concerning the combination of the safety and the security domains. We conclude that the utilization of a conceptual model unifying relevant documentation artifacts from requirements engineering, system modeling, risk assessment and evidence documentation can address these issues. Future research will be conducted in alignment with derived criteria in order to investigate how change management can be facilitated by introducing state-machine based automation capabilities to this model. Means to enable state propagation, the definition of accommodating work flows and a prototypical implementation is planned for the near future. A quality and cost model will be developed to assess the economic viability of our approach addressing aforementioned concerns of interviewees.

Acknowledgments. This work was partially supported by the Austrian Federal Ministry of Science, Research and Economics (BMWFW), FFG Project 855383 SALSA (ICT of the Future).

References

1. Almeida, J.R., Camargo, J.B., Cugnasca, P.S.: Safety and security in critical applications and in information systems-a comparative study. *IEEE Latin Am. Trans.* **11**(4), 1127–1133 (2013)
2. Baheti, R., Gill, H.: Cyber-physical systems. *Impact Control Technol.* **12**, 161–166 (2011)
3. Bloomfield, R., Bishop, P.: Safety and assurance cases: past, present and possible future-an adelard perspective. In: Dale, C., Anderson, T. (eds.) *Making Systems Safer*, pp. 51–67. Springer, Heidelberg (2010). https://doi.org/10.1007/978-1-84996-086-1_4

4. Brunner, M., Huber, M., Sauerwein, C., Breu, R.: Towards an integrated model for safety and security requirements of cyber-physical systems. In: 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), pp. 334–340. IEEE (2017)
5. Campbell, J.L., Quincy, C., Osserman, J., Pedersen, O.K.: Coding in-depth semistructured interviews problems of unitization and intercoder reliability and agreement. *Sociol. Methods Res.* **42**(3), 294–320 (2013)
6. Derler, P., Lee, E.A., Vincentelli, A.S.: Modeling cyber-physical systems. *Proc. IEEE* **100**(1), 13–28 (2012)
7. Firesmith, D.G.: Common concepts underlying safety security and survivability engineering. Carnegie-mellon University, Pittsburgh, PA, Software Engineering Institute, Technical report (2003)
8. Friedrich, J., Kuhrmann, M., Sihling, M., Hammerschall, U.: *Das V-Modell XT*. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-76404-5_1
9. Fürst, S., et al.: AUTOSAR—a worldwide standard is on the road. In: 14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden, vol. 62, p. 5 (2009)
10. Glas, B., et al.: Automotive safety and security integration challenges. In: *Automotive-Safety & Security 2014* (2015)
11. He, W., Yan, G., Da Xu, L.: Developing vehicular data cloud services in the IoT environment. *IEEE Trans. Ind. Inform.* **10**(2), 1587–1595 (2014)
12. ISO/TC 22: ISO/DIS 26262-1 - Road vehicles functional safety Part 1–10. Technical report, Technical Committee 22, Geneva, Switzerland, July 2009
13. Kannenberg, A., Saiedian, H.: Why software requirements traceability remains a challenge. *CrossTalk J. Defense Softw. Eng.* **22**(5), 14–19 (2009)
14. Kelly, T.P.: *Arguing safety: a systematic approach to managing safety cases*. Ph.D. thesis, University of York (1999)
15. Kitchenham, B.A., Pfleeger, S.L.: *Guide to advanced empirical software engineering*. Springer, London **46**, 48–49 (2008)
16. Kletz, T.A.: *HAZOP and HAZAN: Identifying and Assessing Process Industry Hazards*. IChemE, Boca Raton (1999)
17. Kornecki, A.J., Subramanian, N., Zalewski, J.: Studying interrelationships of safety and security for software assurance in cyber-physical systems: approach based on Bayesian belief networks. In: 2013 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 1393–1399. IEEE (2013)
18. Kriaa, S., Pietre-Cambacedes, L., Bouissou, M., Halgand, Y.: A survey of approaches combining safety and security for industrial control systems. *Reliab. Eng. Syst. Saf.* **139**, 156–178 (2015)
19. Macher, G., Höller, A., Sporer, H., Armengaud, E., Kreiner, C.: A combined safety-hazards and security-threat analysis method for automotive systems. In: Koornneef, F., van Gulijk, C. (eds.) *SAFECOMP 2015*. LNCS, vol. 9338, pp. 237–250. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24249-1_21
20. Martins, L.E., Gorschek, T.: Requirements engineering for safety-critical systems: overview and challenges. *IEEE Softw.* **34**, 49–57 (2017)
21. Mayring, P., Gläser-Zikuda, M.: *Die Praxis der Qualitativen Inhaltsanalyse*. Beltz Weinheim (2008)
22. Nostro, N., Bondavalli, A., Silva, N.: Adding security concerns to safety critical certification. In: 2014 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 521–526. IEEE (2014)

23. Pedersen Notander, J., Höst, M., Runeson, P.: Challenges in flexible safety-critical software development – an industrial qualitative survey. In: Heidrich, J., Oivo, M., Jedlitschka, A., Baldassarre, M.T. (eds.) PROFES 2013. LNCS, vol. 7983, pp. 283–297. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39259-7_23
24. Piètre-Cambacédès, L., Bouissou, M.: Cross-fertilization between safety and security engineering. *Reliab. Eng. Syst. Saf.* **110**, 110–126 (2013)
25. Ray, S., Chen, W., Bhadra, J., Al Faruque, M.A.: Extensibility in automotive security: current practice and challenges. In: 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE (2017)
26. Runeson, P., Host, M., Rainer, A., Regnell, B.: *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley, Hoboken (2012)
27. Schoitsch, E., Schmittner, C., Ma, Z., Gruber, T.: The need for safety and cybersecurity co-engineering and standardization for highly automated automotive vehicles. In: Schulze, T., Müller, B., Meyer, G. (eds.) *Advanced Microsystems for Automotive Applications 2015*. LNM, pp. 251–261. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-20855-8_20
28. Sojka, M., Krec, M., Hanzálek, Z.: Case study on combined validation of safety & security requirements. In: 2014 9th IEEE International Symposium on Industrial Embedded Systems (SIES), pp. 244–251. IEEE (2014)
29. de la Vara, J.L., Borg, M., Wnuk, K., Moonen, L.: An industrial survey of safety evidence change impact analysis practice. *IEEE Trans. Softw. Eng.* **42**(12), 1095–1117 (2016)
30. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: *Experimentation in Software Engineering*. Springer, Heidelberg (2012). <https://doi.org/10.1007/978-3-642-29044-2>



Safe and Secure Automotive Over-the-Air Updates

Thomas Chowdhury¹(✉), Eric Lesiuta¹, Kerianne Rikley¹, Chung-Wei Lin²,
Eunsuk Kang², BaekGyu Kim², Shinichi Shiraishi³, Mark Lawford¹,
and Alan Wassying¹

¹ McMaster Centre for Software Certification,
Department of Computing and Software, McMaster University,
Hamilton, ON, Canada

{chowdt2,lesiutej,rikleykn,lawford,wassying}@mcmaster.ca

² Systems and Software Division, Toyota InfoTechnology Center U.S.A. Inc.,
Mountain View, CA, USA

{cwlin,ekang,bkim}@us.toyota-itc.com

³ Software Systems Group, System Architecture Research Division,
Toyota InfoTechnology Center Co., Ltd., Tokyo, Japan
sshiraishi@jp.toyota-itc.com

Abstract. Over-the-air updates have been used for years in the software industry, allowing bug fixes and enhancements to desktop, laptop, and mobile operating systems and applications. Automotive vehicles now depend on software to the extent that manufacturers are turning to over-the-air updates for critical vehicle functionality. History shows that our software systems are most vulnerable to lapses in safety and dependability when they undergo change, and performing an update over a communication channel adds a significant security concern. This paper presents our ideas on assuring integrated safety and security of over-the-air updates through assurance case templates that comply with both *ISO 26262* (functional safety) and *SAE J3061* (cyber-security). Wisely, the authors of *SAE J3061* structured the guidebook so that it meshes well with *ISO 26262*, and we have been able to use principles we developed for deriving an assurance case template from *ISO 26262*, to help include compliance with *SAE J3061* in the template. The paper also demonstrates how a specialization of the template helps guide us to pre-emptively mitigate against potential vulnerabilities in over-the-air update implementations.

1 Introduction

The original motivation for over-the-air (OTA) updates to automotive software seems to have been a realization that customers view a trip to the dealership to install a software patch, as an avoidable waste of their time. This is true even when the patch introduces a new feature that they are pleased to install. An update can take place without the presence of the owner. Whether the update

is installed automatically or needs approval before driving depends on the criticality of the update. For example, if the update is for parts of the infotainment system, perhaps it can be installed automatically. If the update is for a critical component of the vehicle then it may be necessary to have driver approval. In all cases, the update will be installed when the car is at home and is stopped in park mode. In addition, original equipment manufacturers (OEMs) hope that OTA Updates will be a lot more cost effective than paying dealerships to install the updates.

However, with the implementation of OTA firmware updates come new entry points for hackers to tamper with a vehicle’s software. Not only do we introduce the potential for hacking, but we also remove a trained technician from the process. These trained professionals help validate that the installation of the new firmware is successful, and ensure that there are no safety hazards resulting from the update. For example, even a simple update to an infotainment system caused cycles of rebooting the heads-up display, accompanied by distracting bright purple flashes, thus resulting in a serious safety concern [4].

It is important to note that we are primarily interested in the final safety of the vehicle. To this end we have to consider the safety aspects of OTA Updates, independent of security concerns, as well as the effect of security issues on vehicle safety – and even the adverse effect of safety mitigation on security. Most current research seems to be heavily focused on security, as though it is an end in itself, although we are starting to see significant work on the interplay between safety and security.

The primary contribution of the paper is the development of an *Assurance Case Template (ACT)* that applies to OTA Updates. Our template complies with both *ISO 26262* [10] and *SAE J3061* [24], and applies in general to functional safety (we have not always prefixed safety by “functional”, but that is the focus of this paper) and cybersecurity of the “connected car”. An important contribution is the demonstration that the template for OTA Updates can be used to guide development (not just to document assurance after/during development) in a way that helps to avoid/mitigate OTA Update vulnerabilities. The scope of cybersecurity in this paper is limited to protecting the download of the updates. We believe that not protecting such downloads is equivalent to a systematic design fault, and is thus of importance to functional safety. Cybersecurity also deals with financial loss, personal identity theft, data loss, etc. These concerns do not have immediate functional safety implications and are thus not considered in this paper.

This paper is organized as follow. A brief introduction to concepts discussed in the paper is provided in Sect. 2. This includes an introduction to relevant standards, assurance cases, and ACTs. This is followed by a brief discussion on relevant literature in Sect. 3. Section 4 is the heart of the paper. It presents an overview of the methodology we used to arrive at an ACT that assures both safety and security for vehicles that may be maintained using OTA Updates. This section extends earlier work [7], in which we developed and used principles for transforming clauses in *ISO 26262* into claims and evidence in an ACT.

We have now used those same principles applied to *SAE J3061*, to develop an ACT that integrates safety and security for the connected car. The section includes a very brief description of threats and threat analysis that we need to include OTA-specific assurance. We added this OTA assurance to the template, based on an open source OTA Update design, Uptane [13,15]. In Sect. 5 we show how this template can be used by examining a potential vulnerability in an implementation of Uptane, that was discovered by instantiation of the ACT. Finally, we present our conclusions and future work in Sect. 6.

2 Preliminaries

2.1 Relevant Standards

There are two standards that are specifically relevant to this work. The first is *ISO 26262* which has become the de facto functional safety standard for electric and software components in automotive vehicles. The second is a newer “standard”, *SAE J3061*, which is an SAE guidebook specifically targeting automotive security. It is intended as a companion standard to *ISO 26262*, and has been organized to mesh well with *ISO 26262*, but its written structure differs significantly from *ISO 26262*. There is an unpublished standard *ISO/SAE 21434* [11] for cybersecurity of automotive vehicles. The standard defines requirements for cybersecurity risk management for road vehicles throughout the development process [3]. The standard is currently under development.

2.2 Assurance Cases

An assurance case is a living document assuring a system’s critical properties. According to Bloomfield et al., “An assurance case is a documented body of evidence that provides a convincing and valid argument that a specified set of critical claims about a system’s properties are adequately justified for a given application in a given environment” [5]. An assurance case starts with a top-level claim which is decomposed into sub-claims supported by other sub-claims or evidence. Each (sub-)claim must be supported by its sub-claims and/or evidence. The argument (reasoning) should be explicit. Other artifacts are used in the assurance case to provide *context*, *assumptions*, *justification*, etc. There are various notations for documenting assurance cases, the most popular one currently being *Goal Structuring Notation (GSN)* developed by Kelly [14]. We have used a GSN tool to draw the assurance cases used in this paper.

2.3 Assurance Case Templates

An ACT provides the structure for a family of assurance cases for a particular product-line. Given a specific product in that product line, one instantiates the ACT to create a complete assurance case for that product. An ACT is a complete assurance case for a particular product-line. The template consists of optional

argument paths corresponding to various features of different products from a specific product-line. A major benefit of templates is that they are developed before the systems/products are built, and thus an evidence node in the template contains a guideline for the specific evidence required to support a sub-claim, along with acceptance criteria for that evidence [29]. The original motivation for a product-specific ACT was that if every assurance case uses a unique structure and argument, regulators will be overwhelmed by the task of evaluating these assurance cases [30]. Another motivating factor was the work by Graydon, Knight and Strunk on Assurance Based Development [9].

Making these templates specific to a product line may yield the following additional benefits:

- Facilitation of incremental certification;
- Robustness with respect to likely changes (reminiscent of information hiding);
- Playing the role of a safety plan with regard to what needs to be produced, for example:
 - Directing developers as to what evidence should be provided to support specific sub-claims, as well as acceptance criteria for that evidence;
 - Structured using arguments built by people with appropriate expertise;
 - Avoiding confirmation bias, due to the template being constructed before development begins; and
 - Providing a publicly accessible example of an assurance case argument and structure.

A skeleton for the claims, sub-claims and evidence in an ACT is shown in Fig. 1.

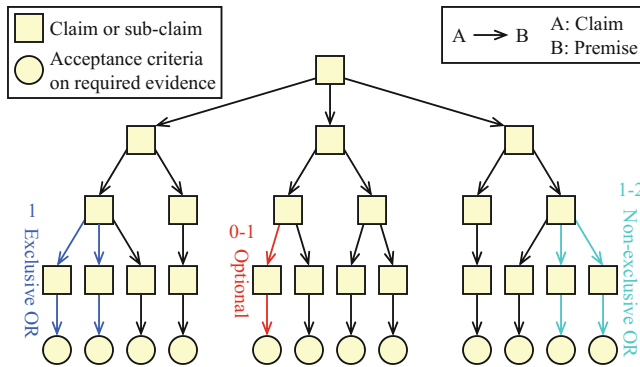


Fig. 1. Basic structure of an ACT [7]

2.4 Principles for Developing the ACT

The starting point in developing the ACT is the decision to make it compliant with *ISO 26262* and *SAE J3061*. We previously developed and published principles for constructing an ACT from a standard such as *ISO 26262* [7]. The 10

Table 1. Principles for developing an ACT from a standard

Id	Principle	Description
1	Model the standard	Understand terms, relationships & requirements
2	Model system variability	Model detailed enough to guide development
3	Flip-It	Reverse process flow for claim dependency
4	Conjunctive claims	Multiple claims support a parent claim
5	Optional pattern	Standard defined alternative processes
6	Evidence acceptance criteria	Standard specifies attributes/characteristics
7	Evidence classification	Determines type of evidence required
8	Completeness	Arguments that depend on completeness
9	Argument options	Optional paths motivated by alternative arguments
10	Feature options	Optional paths motivated by alternative features

principles are shown in Table 1. We originally used these principles to build an ACT that assures safety and is compliant with *ISO 26262*. We have now used these principles to include security in the ACT, compliant with *SAE J3061*. Although *SAE J3061* is not written in the style of *ISO 26262*, we did manage to apply those principles sufficiently well to help in the derivation of the new ACT. We supplemented compliance with the standards with our knowledge of safety and software engineering principles. More detail about this process is in Sect. 4.2.

3 Related Work

Various threat analysis methods are described in [17–19, 25, 31]. OTA Update specific security is discussed in [13, 15, 26]. In particular, the *Uptane Project* [13, 15] defines an open source software security system with a flexible design, allowing it to be adapted easily to various systems. The Uptane project presents a comprehensive look at common types of attacks that an unsecured vehicle will be vulnerable to, specifically when updated remotely. The attacks described in [13] are: Read Attacks, Replay Attacks, Denial-of-Service (DoS) Attacks (including Drop Attacks, Slow Retrieval Attacks, Flood Attacks, Freeze Attacks), Roll-back Attacks, Modify Attacks (including Partial Bundle Attacks, Mixed Bundle Attacks, Mix-and-match Attacks), Spoof Attacks and Control Attacks.

Long-term, we believe that the best way of integrating safety and security is to use an integrated hazard/threat analysis and risk management. Implementing safety and security requirements derived separately from independent hazard analysis and threat analysis may lead to conflicting requirements which result in new hazards and/or vulnerabilities, and also may miss hazards/threats resulting from combined security/safety concerns. There are some early attempts at this in the literature. Unfortunately, this is not yet a common approach, simply because the relevant “standards” *ISO 26262* and *SAE J3061* deal with the two aspects separately in order to limit their scope during their initial development. Our

own work currently is based on compliance with *ISO 26262* and *SAE J3061*. However, we suggest more comprehensive integration in our section on “Future Work” (Sect. 6.1). In the meantime, we have included a brief look at the literature on integrated hazard and threat analysis in this section.

Systems-Theoretic Process Analysis (STPA), developed by Nancy Leveson, is a well-regarded hazard analysis technique that is focused strictly on ensuring safety [16]. *STPA-Sec* [32] developed by Leveson and Young, is a derivative of STPA in the security domain. STPA-Sec is an extension of STPA considering security aspects in a top-down fashion. However, in striving to integrate safety and security analysis, separate analysis of safety and security does not seem to adequately cover the integrated effects of safety and security. Another method, *STPA-SafeSec* [8] based on STPA, proposes a more unified analysis technique for safety and security. To support the unified approach, STPA-SafeSec defines the component layer diagram and extends the causal factors of security domains. This method considers the cyberattacks on integrity and availability at the component layer. The authors do not show the relationship between safety and security, and how conflicts can be resolved is not explicitly defined.

In [23], the authors developed a method called *SAFE (Systematic Analysis of Faults and Errors)*. In order to combine safety and security, SAFE considers a semantic framework of error “effect” that integrates an adversary model used in security analysis with fault/error categorization used in hazard analysis. This method is a heavily modified form of STPA. Safety and security analysis are also combined in [22]; namely, STPA and NIST SP800-30 [2] are considered to derive the safety constraints and security constraints respectively. The authors use an automatic scheme to detect conflicts and reinforcement. However, they do not define the automatic scheme precisely which is the key mechanism in detecting conflicts. In [17], the *SAHARA (Security Aware Hazard Analysis and Risk Assessment)* method derives a measure of the security impact on the “Automotive Safety Integrity Levels” (*ASILs*). This approach uses STRIDE (**S**poofing identity, **T**ampering with data, **R**epudiation, **I**nformation disclosure, **D**enial of service, **E**levation of privilege) to derive “Security Levels” to combine with the *ASILs* based on *ISO 26262*’s HARA (Hazard analysis and risk assessment). Amorim et al. [1] use patterns to interlink safety and security in the development process. Some of the authors of that paper were also involved in creating SAHARA, described above.

4 An ACT for Safety and Security of OTA Updates

The primary aim of this work was to develop an ACT that can be used in general to assure safety and security for automotive vehicles, and especially to deal adequately with OTA Updates. We divided the task into two:

1. Step 1 – Develop an ACT for safety and security of automotive vehicles, compliant with both *ISO 26262* and *SAE J3061*;
2. Step 2 – Specialize the previously developed ACT, to include assurance when maintenance is performed using OTA Updates.

This approach was used since both of the relevant standards do not include specific guidance for OTA Updates, and we believe that there is some general guidance we can provide that covers both maintenance implemented at a dealership, or through OTA Updates.

4.1 Assurance of Integrated Safety and Security

Generally, security and safety are considered separate disciplines because of their own regulations, standards and methodologies [6]. A concept is gaining momentum that security and safety are closely interconnected. Nowadays it is not acceptable to assume that a cyber-physical system is immune to threats and it is not feasible to assure the safety of the cyber-physical system independent of security. In this regard, a safety case is incomplete and unconvincing without consideration of the impact of security. In [6], the authors emphasize that the impact of security on the safety case should be explicitly mentioned to make the system safe and secure. In [21], the authors describe a layered assurance approach that combines safety and security.

4.2 Step 1 – An Automotive ACT for Safety and Security

Figure 2 shows the top-level of a security informed safety ACT for “<X> considered as an ISO 26262 item/SAE J3061 feature, delivers the behaviour required and does not adversely affect the safety of the vehicle, nor does it create security vulnerabilities in the vehicle, over its expected lifetime in its intended environment”. (We have not included “context”, “assumptions” and the content of “strategy” nodes in the diagrams, in the interest of saving space.) Six sub-claims support the top level claim. All six sub-claims deal with safety and security issues together with consistent interaction. The tabs on the top left of a claim node indicate that this is a *module*, and the remainder of that argument path can be seen by “opening” that module (in the tool we use, achieved by double clicking the tab). The relevant ISO and SAE clauses/sections are indicated inside a smaller text box within the claim. In terms of software engineering, four argument paths could be shown to adequately support a top claim of safety of a specific system. An informal description of the four top level claims supported by these arguments, would be:

1. The system’s requirements are “correct” [GS in Fig. 2].
2. The system is implemented to meet its requirements [GR in Fig. 2].
3. The system is safe even when maintenance is performed [GPM in Fig. 2].
4. The system is operated within its operational assumptions [GA in Fig. 2].

ISO 26262 and SAE J3061 take a similar approach, and add two more claims:

5. Compliance with configuration management requirements [GC in Fig. 2].
6. Compliance with change management requirements [GCM in Fig. 2].

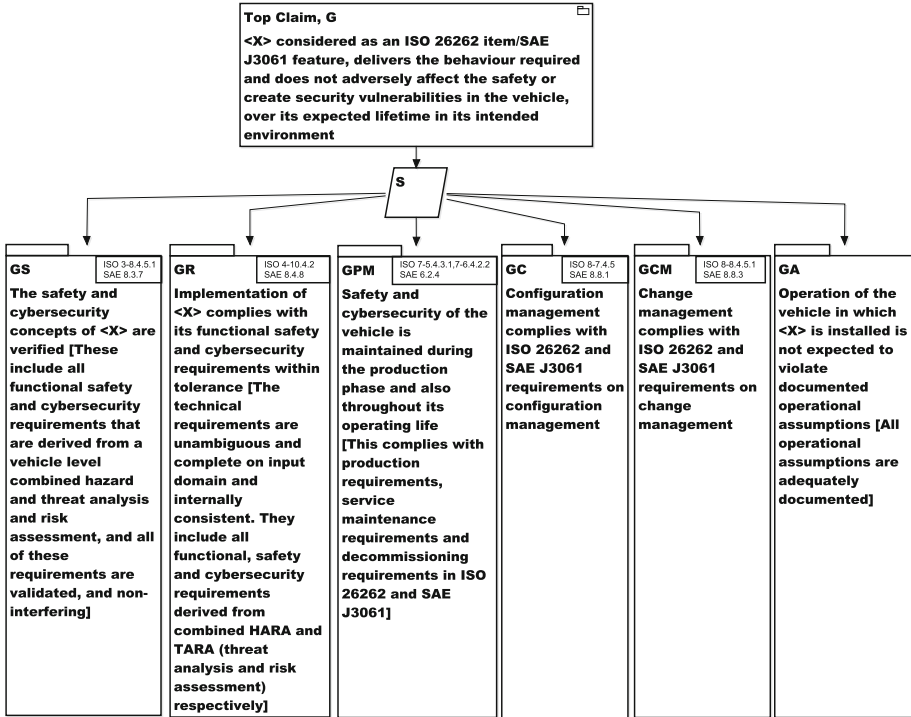


Fig. 2. Top-level of a safety and security ACT

Roughly, the argument that the conjunction of 1, 2, 3, 4 implies safe and secure, follows from the fact that we can think of these claims as:

1. validation,
2. verification,
3. safe/robust with respect to change,
4. operated within known bounds, respectively.

Of particular interest in this template is the argument path leading off the claim “GPM” in Fig. 2. This path shows how we, through compliance with the standards, argue the safety of *maintenance* throughout the life of the vehicle. Part 7 of *ISO 26262* and Sect. 6 of *SAE J3061* define maintenance requirements on production, and operation. We have highlighted this path because it is of central importance in arguing the safety and security of OTA Updates. Figure 3 shows a slice of “GPM” developed from *ISO 26262* and *SAE J3061*. The structure is largely dictated by the structure of the standards. For example, the safety argument is contained in the GPM1 branch, and the security argument in the GPM2 branch. *ISO 26262* describes requirements on production, maintenance, and decommissioning. One option would have been to split these at the sub-claim level shown in Fig. 2. We chose to combine them in a single

claim, and so the premises for GPM1 are GPM1.1 (production), GPM1.2 (maintenance) and GPM1.3 (decommissioning). Similarly, the premises for GPM2 are GPM2.1 (production), GPM2.2 (maintenance) and GPM2.3 (decommissioning). The decommissioning claim (in module-GPM2.3) was further decomposed into several sub-claims to assure the required cybersecurity properties during decommissioning. They are not shown in the diagram as they are not relevant within the context of this paper. There are no decommissioning aspects related to cybersecurity. Figure 3 shows compliance with the ISO and SAE standards **before** any specialization for OTA Updates. It is reasonably obvious that OTA Updates will affect claim GPM1.2 and its argument path (safety) and claim GPM2.2 and its argument path (cyber-security) – primarily the GPM2.2.1 argument path.

4.3 Step 2 – An Automotive ACT for Safety and Security that Includes OTA Updates

In order to include OTA Updates explicitly in the ACT, we have to analyze exactly what is different between traditional at the dealership maintenance, and OTA Update maintenance. This involves both hazard and threat analyses. OTA Updates introduce both safety and security vulnerabilities. In terms of safety, OTA Updates are performed remotely, without the aid of a knowledgeable technician who would be responsible for testing the update. Clearly, the update will have been thoroughly tested by the manufacturer, but there are significant issues of *completeness* that complicate this task. An obvious example is the malfunctioning heads-up display discussed in the Introduction. In terms of security, *SAE J3061* describes in general how to protect the vehicle from cyber-security attacks. The guidebook does not explicitly consider what is necessary when maintenance is performed OTA. We want to include the option of OTA Updates in our ACT. To do this, we used the work reported in the design of Uptane [13, 15] as the basis of the OTA-specific arguments in the ACT, as far as security is concerned. Once we have a design in mind (and Uptane is sufficiently generic in terms of identification of communication channels), we are in a position to generate threats and mitigations that can be used as a base for the assurance case argument.

When analyzing OTA Updates, not only must the security of data be considered, but the protocols that handle this data must also be considered. We note that relevant attacks consistently target and exploit weaknesses of four main security properties: *confidentiality*, *integrity*, *availability*, and *authenticity* [26]. By adequately protecting these four main properties, which have been at the root of all known attacks, it is possible to provide security assurance for the system.

The first three of these properties are widely considered to be the most crucial components of information security [27], and are known as the *CIA triad*, and CIA is currently being used to analyze the security requirements of more than one hundred use cases of the connected vehicle proposed by the ARC-IT project funded by the U.S. Department of Transportation [28].

Confidentiality: Confidentiality of communicated information is put at risk by read attacks. Data encryption is suggested to mitigate these attacks [26].

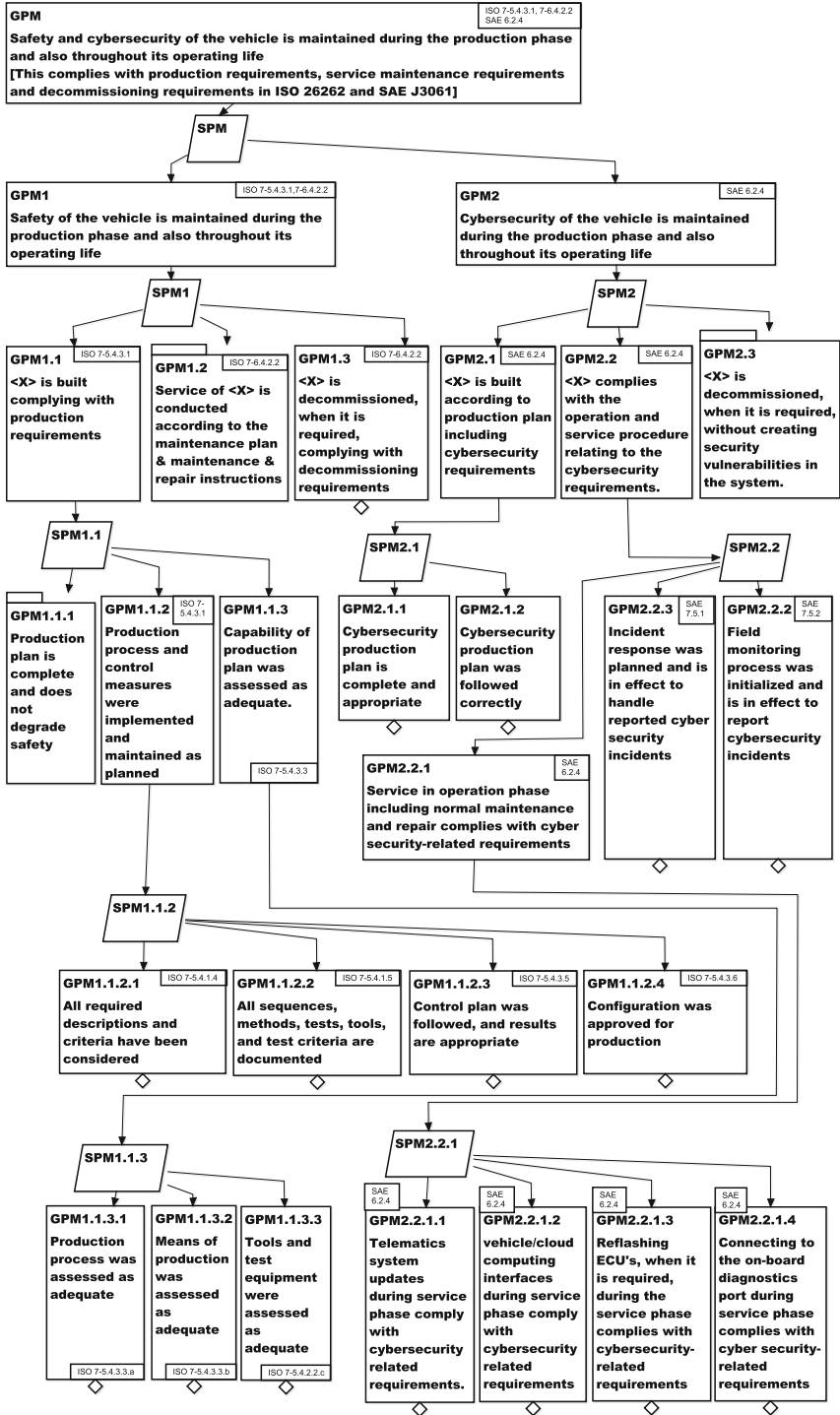


Fig. 3. Extract of assurance case for maintenance of automotive vehicles (GPM)

Integrity: The integrity of data can be protected through the use of hashing, cyclic redundancy checks (CRC) and signatures, preferably used in combination.

Availability: A comprehensive backup strategy, anomaly detection, and time-outs are recommended to mitigate these attacks [26].

Although the CIA triad are considered the most crucial components of information security, they are not enough to completely secure the system. The STRIDE Threat Model from Microsoft [19] recommends protection of Authenticity, Authorization, and Non-repudiation.

Authenticity: Authenticity ensures that the data received comes from a trustworthy source. This protects against man in the middle (MITM) and spoofing attacks [26].

Authorization: Authorization prevents unprivileged parties gaining access [31].

Non-repudiation: Maintaining secure logs of activities and the entities to which they are attributed protects non-repudiation scenarios [31].

We also need to consider two generic security measures – **private key protection** and **version control**. Private key protection can help prevent “key extraction” [26], and version control is essential in general, but can also help protect against installation of an older version of software.

There exist a number of tools and methodologies for classifying and managing security related threats. Many of these are outlined in *SAE J3061*. We chose to use Microsoft’s threat modelling tool which performs threat analysis using STRIDE [19] and a data flow diagram of the system [18]. STRIDE classifies attacks (threats) into six categories – *Spoofing identity*, *Tampering with data*, *Repudiation*, *Information disclosure*, *Denial of service*, and *Elevation of privilege*.

For each type of threat presented by STRIDE, Microsoft suggests a security property countermeasure.

The NCC group [20] created a customized template for the automotive domain to perform threat analysis using STRIDE, and we created a data flow diagram that describes the communication flow as input to STRIDE. We modelled our data flow diagram on an Uptane design (Fig. 1 in [13]). Our data flow diagram of a partial vehicle network illustrating OTA Updates is shown in Fig. 4. We used the NCC template together with the data flow diagram in Fig. 4 to analyze OTA Updates for the connected car, to generate threats and corresponding mitigations to include in our ACT.

A slice of GPM specialized for OTA Updates using the results from the STRIDE analysis, is shown in Fig. 5.

5 Example Usage of the ACT for OTA Updates

We used this slice of the ACT to explore what we would need to do to develop a safe and secure OTA Update design. Thanks to the extensive work in the Uptane project, we could use their design and a python implementation as an

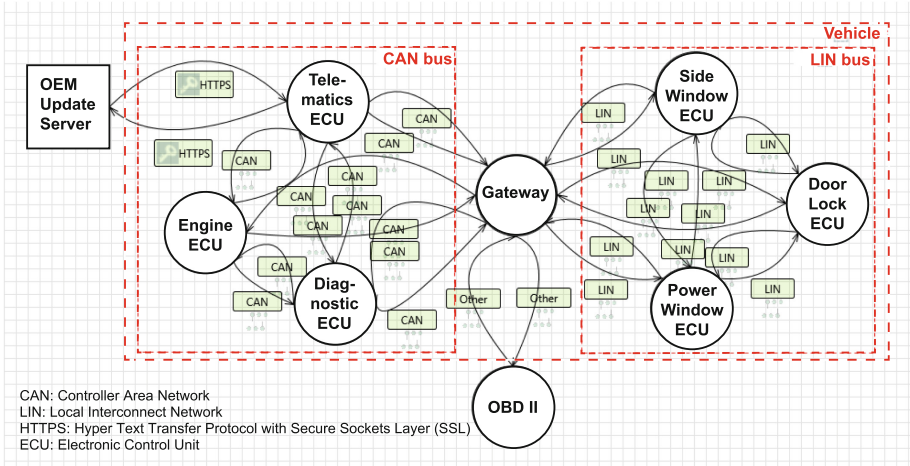


Fig. 4. Data flow diagram of a partial vehicle network based on Uptane [13]

example. We found that part of the implementation did not satisfy one of the threat mitigation requirements in the acceptance criteria of the ACT (see Fig. 5).

In particular, Threat 2 in EPM 2.2.1.1.1.a.2.1 refers to a MITM threat. This may lead to a vulnerability in the Uptane implementation. The suggested mitigation strategy is that communication must be secured (using TLS or cryptographically signed). In the sample implementation, requests from the primary ECU to the OEM's time server for an updated timestamp are sent as unsigned plain text. (The OEM time server is included in the OEM Update Server in Fig. 4). Although the communication is just a pseudorandom nonce from each secondary ECU, this allows MITM agents to alter the communication as they see fit, and force the system into an unexpected state. Depending on a vendor's implementation, attacks such as a buffer overflow could be possible. In this case, editing the packet to contain no nonces, then allowing it to go through, causes the primary ECU to ignore the updated time. However, it will then make its next request to the time server without sending any nonces, at which point the MITM can inject a subset of the previously blocked nonces, and the primary ECU will accept the reply from the time server. The primary ECU will then pass the message from the time server along to all the secondary ECUs, but since the MITM manipulated the exchange to only contain a subset of nonces, only secondary ECUs in this selected sub set will accept the updated time. If a vendor decides to implement a check for a recent timestamp from the time server on each secondary ECU before installing an update, a Mixed Bundle Attack could be possible. The ACT suggests mitigating this vulnerability by signing the packet of nonces from the primary ECU to the time server. If the developer does this, this specific MITM attack can be mitigated.

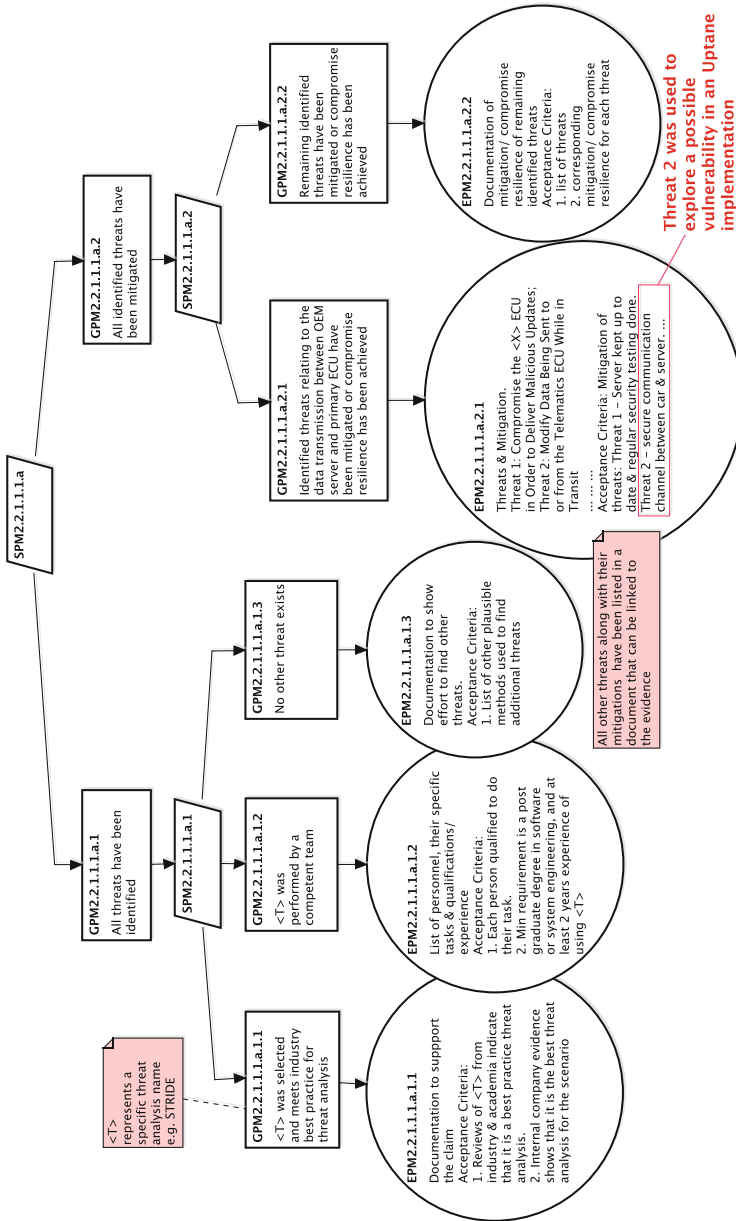


Fig. 5. Slice of OTA update ACT

6 Conclusion

We have demonstrated that ACTs can be designed to integrate functional safety and security for automotive vehicles. As a basis for such a template, we start with claims and evidence derived from *ISO 26262* and *SAE J3061* using principles we developed for just such a process. We then integrate into that ACT, claims and evidence based on our knowledge and derived from additional analyses. In particular, we specialized the ACT to include specific arguments that apply when maintenance is performed using OTA Updates. To illustrate the value of using these templates in automotive development, we used the OTA specific template to show that it may help developers mitigate security threats, during development of the vehicle.

6.1 Future Work

We are exploring how to better integrate functional safety and security. As mentioned earlier, one of the best ways is to integrate the hazard and threat analyses. If we achieve this, we will then re-evaluate the argument in the assurance case. This should eventually result in changes to *ISO 26262* and *SAE J3061* – or even better, the integration of cyber-security into *ISO 26262*. We intend to derive more examples of evidence, and especially acceptance criteria for that evidence. In this paper we only considered the integration of security of OTA and functional safety in a combined ACT. Currently ISO is developing a new standard, ISO 21448, on “Road vehicles – Safety of the intended functionality” [12]. Rather than the safety of the system in the presence of failures, this standard concerns itself with the safety of systems when they are functioning correctly. Once the standard becomes available we intend to develop an explicit ACT for it applying the same method used for *ISO 26262* [7]. We could then integrate the SOTIF ACT with the ACT proposed in this paper. Finally, although we have made quite a lot of progress in working out *how* to develop these ACTs, we believe there is still more work to be done in this regard, and we are starting to look at syntax and semantics for ACTs in order to build effective tools.

References

1. Amorim, T., et al.: Systematic pattern approach for safety and security co-engineering in the automotive domain. In: Tonetta, S., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2017. LNCS, vol. 10488, pp. 329–342. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66266-4_22
2. Aroms, E., et al.: NIST Special Publication 800–30 Risk Management Guide for Information Technology Systems (2012)
3. Barber, A.: Status of work in process on ISO/SAE 21434 Automotive Cyber-security Standard. <https://www.sans.org/summit-archives/file/summit-archive-1525889601.pdf>. Accessed 28 May 2018
4. BBC News: Faulty update breaks Lexus cars’ maps and radio systems. <http://www.bbc.com/news/technology-36478641>. Accessed 08 Jun 2016

5. Bloomfield, R., Bishop, P., Jones, C., Froome, P.: ASCAD. Adelard Safety Case Development Manual, Adelard (1998). ISBN 0-9533771-0 5
6. Bloomfield, R., Netkachova, K., Stroud, R.: Security-informed safety: if it's not secure, it's not safe. In: Gorbenko, A., Romanovsky, A., Kharchenko, V. (eds.) SERENE 2013. LNCS, vol. 8166, pp. 17–32. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40894-6_2
7. Chowdhury, T., Lin, C.W., Kim, B., Lawford, M., Shirraishi, S., Wassyng, A.: Principles for systematic development of an assurance case template from ISO 26262. In: IEEE International Symposium on Software Reliability Engineering, pp. 69–72, October 2017
8. Friedberg, I., McLaughlin, K., Smith, P., Laverty, D., Sezer, S.: STPA-SafeSec: safety and security analysis for cyber-physical systems. *J. Inf. Secur. Appl.* **34**, 183–196 (2017)
9. Graydon, P., Knight, J., Strunk, E.: Assurance based development of critical systems. In: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007, pp. 347–357, June 2007
10. ISO: 26262: Road vehicles-Functional safety. International Standard ISO 26262 (2011)
11. ISO/SAE AWI: 21434: Road vehicles-Cybersecurity Engineering [Under development]
12. ISO/WD PAS: 21448: Road vehicles - Safety of the intended functionality [Under development]
13. Karthik, T., Brown, A., Awwad, S., McCoy, D., Bielawski, R., Mott, C., Lauzon, S., Weimerskirch, A., Cappos, J.: Uptane: securing software updates for automobiles. In: International Conference on Embedded Security in Car, pp. 1–11 (2016)
14. Kelly, T.: Arguing safety - a systematic approach to managing safety cases. Ph.D. thesis, University of York, September 1998
15. Lauzon, S.: Secure software updates for automotive systems: introduction to the Uptane SOTA solution, May 2017
16. Leveson, N.: Engineering a Safer World: Systems Thinking Applied to Safety. MIT Press, Cambridge (2011)
17. Macher, G., Armengaud, E., Brenner, E., Kreiner, C.: Threat and risk assessment methodologies in the automotive domain. *Procedia Comput. Sci.* **83**, 1288–1294 (2016)
18. Microsoft: Microsoft threat modeling tool. <https://www.microsoft.com/en-us/download/details.aspx?id=49168>. Accessed 20 Sept 2017
19. Microsoft: The STRIDE threat model. [https://msdn.microsoft.com/en-us/library/ee823878\(v=cs.20\).aspx](https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx). Accessed 20 Sept 2017
20. NCC Group: The Automotive Threat Modeling Template. <https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2016/july/the-automotive-threat-modeling-template/>. Accessed 20 Sept 2017
21. Netkachova, K., Müller, K., Paulitsch, M., Bloomfield, R.: Security-informed safety case approach to analysing MILS systems (2015)
22. Pereira, D., Hirata, C., Pagliares, R., Nadjm-Tehrani, S.: Towards combined safety and security constraints analysis. In: Tonetta, S., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2017. LNCS, vol. 10489, pp. 70–80. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66284-8_7
23. Procter, S., Vasserman, E.Y., Hatcliff, J.: Safe and secure: deeply integrating security in a new hazard analysis. In: ARES, p. 66. ACM (2017)
24. SAE International: SAE J3061-Cybersecurity Guidebook for Cyber-Physical Automotive Systems. SAE-Society of Automotive Engineers (2016)

25. Shostack, A.: Threat Modeling: Designing for Security. Wiley, Hoboken (2014)
26. Spaan, R., Batina, L., Schwabe, P., Verheijden, S.: Secure updates in automotive systems, pp. 1–71. Radboud University, Nijmegen (2016)
27. Summers, A., Tickner, C.: What is Security Analysis. <http://www.doc.ic.ac.uk/~ajs300/security/CIA.htm>. Accessed 20 Sept 2017
28. US Department of Transportation: Architecture reference for cooperative and intelligent transportation. <https://local.iteris.com/arc-it/>. Accessed 24 Feb 2018
29. Wassyng, A., Joannou, P., Lawford, M., Maibaum, T.S., Singh, N.K.: Chapter 13 new standards for trustworthy cyber-physical systems. In: Trustworthy Cyber-Physical Systems Engineering, pp. 337–368. CRC Press (2016)
30. Wassyng, A., Maibaum, T., Lawford, M., Bherer, H.: Software certification: is there a case against safety cases? In: Calinescu, R., Jackson, E. (eds.) Monterey Workshop 2010. LNCS, vol. 6662, pp. 206–227. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21292-5_12
31. Webtrend: threat modeling with STRIDE. <https://www.webtrends.com/blog/2015/04/threat-modeling-with-stride/>. Accessed 20 Sept 2017
32. Young, W., Leveson, N.: Systems thinking for safety and security. In: Proceedings of the 29th Annual Computer Security Applications Conference, pp. 1–8. ACM (2013)



Dependability Analysis of the AFDX Frame Management Design

Venesa Watson^{1,2(✉)} and Mahlet Bejiga^{1,3}

¹ Framatome GmbH, Erlangen, Germany
{venesa.watson, mahlet-ermias.bejiga}@framatome.com

² University of Siegen, Siegen, Germany
venesa.watson@uni-siegen.de

³ Friedrich-Alexander University Erlangen-Nürnberg, Erlangen, Germany
mahlet.bejiga@studium.fau.de

Abstract. Avionics Full Duplex Switched Ethernet (AFDX) is an implementation of the ARINC 664 specification, which defines the electrical and protocol specifications for data exchange between Computer Systems. AFDX implements extensions on standard Ethernet to achieve a deterministic and fault-tolerant network, which is demonstrated through its frame management design. AFDX, like other emerging time-critical Ethernet-based standards, has potential for use in other critical industries, such as nuclear power plants. This would provide an additional option by which industry players can leverage the speed and ubiquity of Ethernet, with the added benefit of services to support highest safety requirements. However, considering that the nuclear industry continues to be a prime target for advanced security threats, it is imperative to demonstrate what protection AFDX offers, as well as what additional attack surface it may introduce. For this paper, the basic taxonomy of dependable and secure computing is used to conduct a dependability analysis of the AFDX frame management design. An OMNeT++ model simulation of an AFDX network is used to demonstrate potential attacks. Considerations for solutions for a robust AFDX specification are proposed for future research.

Keywords: AFDX · Deterministic · Critical industries
Dependable and secure computing

1 Introduction

ARINC 664 is a multi-part specification that defines a network that provides deterministic, secure and reliable communications data exchange, with redundancy management [1, 2]. AFDX is an implementation of ARINC 664, as defined in part 7 of this specification, and specifically addresses the electrical and protocol conditions for data exchange between Avionics Computer Systems. AFDX is based on commercial 10/100 Mbit switched Ethernet, but with extensions to support determinism and fault-tolerance [1, 2]. There are three components of an AFDX network, namely, the Avionics Subsystem, the End System, and the AFDX Interconnect (Fig. 1) [1, 3, 4]. The core services are provided by the End System and the AFDX Interconnect, each having subcomponents to support their functions. For instance, Virtual Links (VLs) are

subcomponents of the AFDX Interconnect. These VLs are logical, unidirectional paths from one source End-System to one or more destination End-Systems and are used to reserve link capacity and to guarantee reliable and deterministic transmissions [1, 3]. As surmised from [1], the functioning of an AFDX network relies on static, pre-defined configurations, which are implemented by the system integrator. In fact, the specification of AFDX only defines the required network performance without stipulating the methods by which to achieve this performance. Configurations implemented at the discretion of the system integrator include the VL routes for frame transmission, the transmission rate allowances for the VLs, and the size of the receiving and forwarding buffers in the AFDX switch. Determinism is then guaranteed through sufficient queuing capacities and the static routes of reserved link capacity. Fault-tolerance is achieved through redundant AFDX switches and through the Redundancy Manager (RM) of the End System.

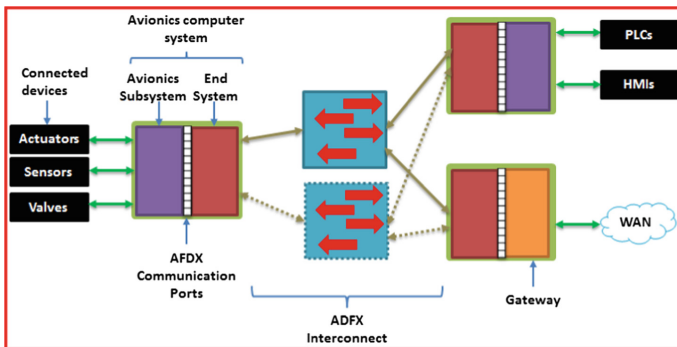


Fig. 1. Typical AFDX network with redundant switch [1, 3]

Comparable counterparts to AFDX are the Time-Triggered Ethernet (TTE), defined in SAE AS6802; and Audio Video Transport Protocol (AVTP), defined in IEEE 1722-2016. Like AFDX, TTE and AVTP are all based on standard Ethernet, with services added to provide deterministic and fault-tolerant data exchange [1, 5, 6]. The differences among the three are observed in how they achieve determinism. However, each has the capacity for use in critical infrastructures. On one hand, TTE and AVTP use services that support flexible performance. For instance, AVTP uses the Generalized Precision Time Protocol (IEEE 802.1 AS), for time synchronization; the Stream Reservation Protocol (IEEE 802.1Q), for bandwidth reservation; and the Forwarding and Queueing Enhancements for Time-Sensitive Streams (IEEE 802.1 qAV), for forwarding time-critical data [5]. Such implementations allow TTE and AVTP networks to be dynamic, allowing on-the-fly route reservation for data exchange. On the other hand, AFDX uses preset configurations to create static routes with fixed link capacity, which achieves determinism in terms of quality of service (QoS), and not necessarily in terms of deterministic transmission. Through these fixed configurations, an AFDX network can be regarded as a dependable system. In that, upon verification of its initial configuration, an AFDX network should reliably route frames, prevent the intrusion of unverified systems/routes, and ensure conformance to network allowances. However,

whilst these highlight quantifiable strengths of an AFDX network, weaknesses arise from the fact that AFDX offers no mechanism to detect or prevent unauthorized or unfavorable manipulations of these configurations. Therefore, the dependability of the data exchange is debatable.

The original definition of dependability states that it is the ability to deliver service that can justifiably be trusted; whilst the alternate definition offers that the dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable [7]. A service failure refers to an event that occurs when the delivered service deviates from the correct service [7]. Therefore, to be truly dependable, an AFDX network must deliver service that can be justifiably trusted and is resilient to the effects of threats that can result in frequent and/or severe service failures. This paper ascertains the dependability of AFDX in relation to its intrinsic protection against attacks and in relation to its vulnerable points. Discussions of future research highlight prospective solutions and the considerations for their effect on the deterministic property of AFDX. The paper is organized as follows: Sect. 2 addresses related works; whilst Sect. 3 provides a summary of the AFDX frame management design, with an analysis against the basic taxonomy of dependable and secure computing [7]. Section 4 uses the AFDX OMNeT++ model to demonstrate the results of three attack test cases. Section 5 discusses the perspective and considerations of future research, and Sect. 6 summarizes the main points of the paper.

2 Related Work

Both references [8, 9] use a model of an AFDX network to provide a reliability analysis of its frame management design. Frame management is used to refer to the flow of messages from the source End System to the destination End System and the supporting processes for a dependable message exchange. As presented in these works, the reliability analyses concerned delivery of sound frames and the reliable (QoS-compliant) delivery of the same. References [8, 9] indicate that the AFDX frame management is vulnerable to the effects of a babbling End System and network or channel errors that can result in dropped frames and unexpected End System resets. Reference [8] proposes a modification of the original frame management design to include a priority queue at the receiver for storing the frames, which would allow the identification of babbled frames. An additional proposal was to communicate redundant copies of the reset message to prevent unwarranted resets from a babbled reset frame or from a channel error. As indicated in [9], these solutions will induce unfavorable delays at the destination End System in its delivery of frames to the upper layers of the protocol stack, as well as increased delay before the destination End System reset. In response, [9] suggests the introduction of a new field in the AFDX frame, which will hold the signature (hash value) of the frame sequence number. Message authentication is performed by the destination End System to verify the frame legitimacy. A second proposal includes the implementation of a queue at the destination End System, and the use of a variable *psn* (previous sequence number). These are used together to determine the legitimacy of incoming frames.

Whilst the proposals from [9] are seemingly more suitable compared to [8], neither paper addresses the reliability of the system from the perspective of maintaining trustworthy configurations. In that, like the original AFDX design, the proposed extensions accepted frames based solely on their sequence number, whilst also making additional allowances to accept delayed frames. This paper considers the dependability of the frame design, rather than simply the reliability, which is just one attribute of dependability.

3 AFDX Dependability Analysis

3.1 AFDX Frame Management

Traffic transmitted between Avionics Computer Systems is shaped and policed by the End System and the AFDX switch. The End System is supported by its subcomponents: the integrity checker, the Redundancy Manager, and the VL Scheduler (comprised of bandwidth allocation gap (BAG) regulators and a multiplexer (MUX)). For the AFDX switch, its subcomponents are the receiving (Rx) and transmitting (Tx) buffers, memory bus, CPU, and the forwarding network table [1].

A simplified overview of the AFDX frame management design is as follows. The sending Avionics Subsystem communicates a message through a predetermined AFDX communication port to its connected End System. The End System encapsulates the message in an Ethernet frame, in preparation for transmission on an also pre-defined VL. The VLs have three properties. The BAG, which, despite its name, is not a measurement of bandwidth, but is a time measurement to represent the minimal interval (milliseconds) between frames transmitted on the virtual link. This value ranges in powers of 2 from 1 to 128 ms [1]. The second is the Lmax, which is the largest Ethernet frame, in bytes, that can be transmitted on the virtual link. The third is the limit of the data transmission rate, which is a function of Lmax and BAG, and represents the maximum transmission/link capacity for a given VL. These properties are enforced by the VL Scheduler, which selects frames for transmission, based on the implemented scheduling algorithm. Once a frame is scheduled, a sequence number is appended to the frame. Inside the VL Scheduler, the BAG regulator paces the frames from the VL queues to create zero-jitter output streams, whilst the MUX multiplexes the BAG regulator outputs into the Redundancy Manager for replication and transmission on to the physical links. The primary frame and the replicated frame are transmitted on different networks towards an AFDX switch. Not all frames are accepted at the AFDX switch, as its policing function checks for and drops non-conformant frames (based on the VL properties). The AFDX switch uses a store-and-forward approach in delivering the frames in a First-in-First-Out (FIFO) order [1]. Incoming frames arrive at the Rx buffers and wait to be transferred to the output ports Tx buffers. The switch CPU uses the VL identifier (VLID) found in the frame header, in conjunction with the forwarding table of the switch, to retrieve the pre-defined destination(s) for the frame. Once the route is retrieved, the frame is copied from Rx buffer to Tx buffer, through the memory bus, then transmitted in a FIFO order on the outgoing link to the destination End System(s) or to another switch. Once received at the destination End System(s), the primary and replicated frames go through integrity

checking, which is done by the Integrity Checker. The sequence number of a frame is used to determine its validity. Valid frames are submitted to the Redundancy Manager of the destination End System, which accepts the first correct frame and drops any duplicates [1]. The frame is now transmitted to the receiving Avionics Subsystem(s). Figure 2 provides a summary of this message flow [10].

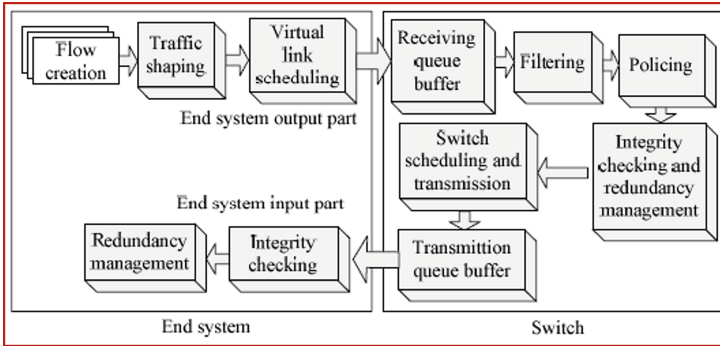


Fig. 2. The AFDX network simulation system [10]

3.2 Dependable and Secure System

The correct functioning of an AFDX network relies on the configurations implemented by the system integrator, as such, there is an inherent dependence on these configurations being trustworthy. The AFDX switch is the central point of this dependence – the End Systems trust that the switch will: (1) reliably route their messages, (2) detect and prevent non-conformance, and (3) prevent transmissions from unauthorized systems. Therefore, transmission from the AFDX switch is perceived as trustworthy, which brings into question the dependability of the frame management design.

Reliability, safety, maintainability, availability and integrity are all attributes of dependability. In addition to confidentiality, availability and integrity are also considered as attributes of security. However, whilst dependability and security share similar attributes, these are associated with different objectives, as expressed in [7]. The dependability and security specification of a system is a function of the requirements and the use environment, and as such, not all attributes may be required [7]. For instance, given that AFDX is built to support determinism and fault-tolerance, the most relevant attributes are availability (*readiness for correct service*; available for authorized actions only), reliability (continuity of correct service) and integrity (*absence of improper system alterations*; absence of unauthorized alterations) [7]. The bracketed objectives correspond to dependability (*in italics*) and security.

Reference [7] postulates four major categories of means by which the attributes of dependability and security can be attained. These are summarized in Table 1 along with example mechanisms used by AFDX [1, 7]. As gleaned from Table 1, AFDX offers at least one protection mechanism in each category. These mechanisms suggest

that AFDX can negate some network attack vectors, such as packet injection, system intrusion, and network enumeration. For instance, network enumeration and system intrusion are made more difficult with the static paths – the pre-configured forwarding table in the switch negates the simplicity of an attacker attaching a device on the network and being able to communicate on the network or interact with any trusted system. However, the threat of an insider can reduce the strength of these intrinsic protection mechanisms – a common threat in all IT and OT environments.

Table 1. Means for achieving dependability and security, with AFDX example mechanisms.

Means	Definition	AFDX mechanisms
Fault prevention	Means to prevent the occurrence or introduction of faults	The AFDX specification provides a number of mathematical formulae for reference by the system integrator, to ensure that the configurations are conducive to an effective implementation By using static routes, unintended and/or unauthorized communication paths or unintended trust relationships between systems can be avoided
Fault tolerance	Means to avoid service failures in the presence of faults	In the event of a babbling end system, traffic policing and shaping services ensure that the switch is not flooded with babbled frames Redundancy management provides communication continuity, in the presence of a faulty communication link
Fault removal	Means to reduce the number and severity of faults	AFDX specifies services at the Switch to verify the legitimacy of incoming messages, to prevent the introduction and proliferation of malformed or malicious data. This filtering process includes an integrity check, frame size inspection, and traffic policing Filter services are also provided at the receiving End Systems, to prevent the proliferation of faulty or replicated data
Fault forecasting	Means to estimate the present number, the future incidence, and the likely consequences of faults	A Management Information Base (MIB) is implemented in each AFDX component, to store information about the same. MIB objects use fault indicators to signal errors in these components. Users are responsible for deriving the future incidence and likely consequences of the signaled fault(s)

3.3 Attack Path Analysis

As previously mentioned, the main dependability and security attributes of the AFDX frame management design include availability, reliability and integrity. As such attack path analysis should consider the vulnerabilities in the mechanisms used by AFDX to prevent, tolerate, remove and forecast faults. The inside attacker point-of-view is used in this attack path analysis, as research continues to show that insiders are one of the biggest threats to cybersecurity. In their analysis of the insider, some writers have chosen to distinguish the level of knowledge and the level of awareness as even greater factors in determining the extent of the threat posed by an insider [11–14]. For this paper, the insider model as proposed by Saglietti et al. [14] will be referenced (Fig. 3), as this work considers a similar context, that is, a critical infrastructure environment.

levels of network, application, control knowledge			knowledge / information domains
high	medium	low	cable location
			bit streams
			protocol and protocol meta-data
		process behaviour and data	
	control system behaviour and specification		
			control system code and potential vulnerabilities

Fig. 3. Levels of knowledge with corresponding domains of knowledge [14]

To address the insider threat issue, it is useful to determine the threat potential of the insider. Factors to determine this includes the degree of knowledge the insider possesses and the role of the insider (i.e. the level of privileged access). In their paper, Saglietti et al. [14] categorized levels of knowledge of an insider as low, high and full. This paper is written from a network security perspective and concerns the development and operation phase of a power plant. ‘*Lowly informed*’ insiders are then described as having enough knowledge to locate network cables and to perceive bit streams and access protocol metadata [14]. ‘*Highly informed*’ insiders have a more technical knowledge of plant applications and can interpret, remove, change and insert messages sent over the network. Whereas, ‘*fully informed*’ insiders possess even further knowledge, such as of the underlying code, which could have been obtained from their involvement in the system platform development phase or the automation and electrical systems engineering phase. After classification of the insider, the next step is to define the possible faults in the target network and formulate example attack trees to realize these paths.

Using the fault taxonomy as presented in [7], two classes of faults have been identified as suitable for the context of the AFDX network. These are interaction faults and malicious faults. Both are human-made faults, but whilst malicious faults can occur during the development and the operation phases, interaction faults occur in the operation phase. Additionally, whilst both can result in similar service failures, malicious faults are goal-based, and interaction faults are considered as unintentional. Malicious faults can be classified as *malicious logic faults* – involve the use of malicious code (worms, virus, logic bombs); or *intrusion attempts* – involve privilege escalation attacks by internal or external attacker. Interaction faults arise from the elements of the use environment interacting with the system. This includes configuration faults (e.g. wrong parameter settings) and reconfiguration faults (occurs during configuration changes, upgrade and maintenance) [7]. Referring the model in Fig. 3, it can then be said that the aforementioned faults can only be realized in an AFDX network by medium to highly informed insiders. With this, the identified fault categories can be used to create an attack tree for an AFDX network, where the resultant service failure is a denial of service (DoS) event (Fig. 4). The actions in the blue boxes can be intentional (malicious fault) or unintentional (interaction fault). The attack tree considers an attack on the configuration settings of the AFDX switch (the core of the system dependability). As indicated by the attack tree, erroneous changes to VL information (network allowances), forwarding table (traffic routes), and Rx and Tx buffers (queue size), can result in a DoS event.

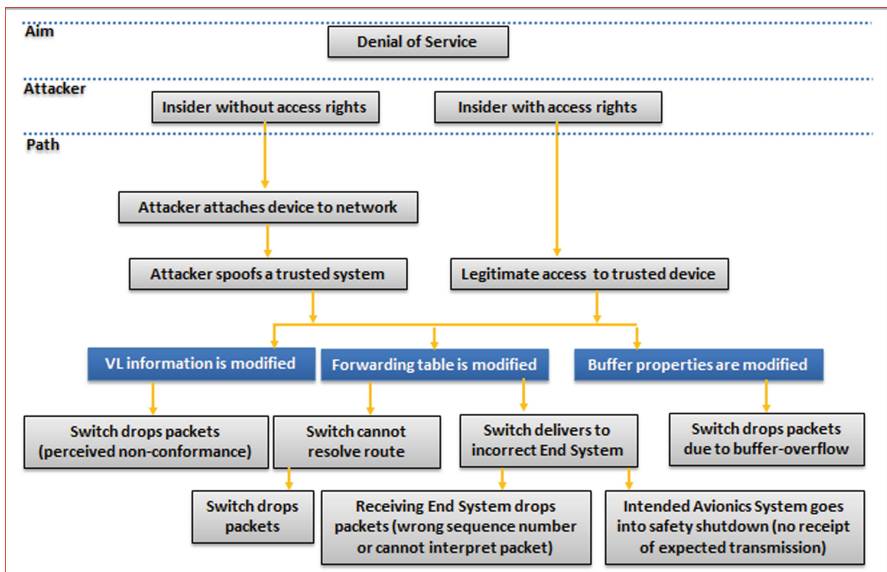


Fig. 4. AFDX DoS attack tree model example (Color figure online)

4 Attack Analysis with OMNeT++ Model of AFDX

As concluded by reference [15], the use of the AFDX network design based on the OMNeT++ TTE model, as well as the simulation approach, provided results that indicate that both “*present a constructive tool and a great value in AFDX network evaluation, particularly on real-time network performance analysis and design verification*”. Furthermore, [15] states that this OMNeT++ simulator represents the only open source simulator of use and is advantageous in facilitating modeling complex case studies. These validate the use of the OMNeT++ simulator for this paper, the source code of which was retrieved from reference [16]. The AFDX network layout used is seen in Fig. 5 [16]. This simulator uses the process modeling library, as described by OMNEST, which is the commercial version of OMNeT++ [17]. The process modelling library is suitable for building queueing and resource reservation models, such as an AFDX network. Models built using this library can be used in testing system performance, as well as in serving as the foundation for the execution of further simulations [18]. The modules of this library are described in reference [18].

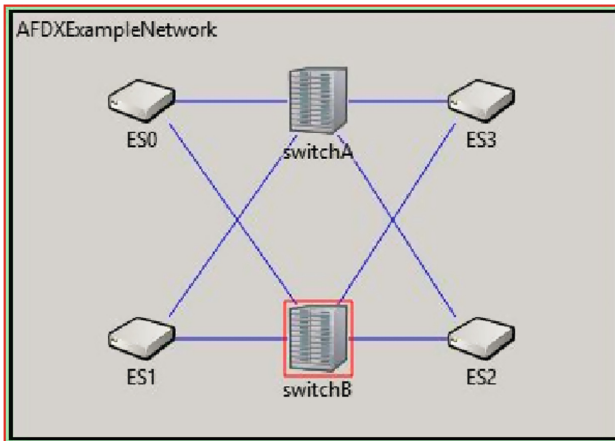


Fig. 5. AFDX network layout in OMNeT++ model

Using the attack tree model as guidance, this OMNeT++ model is modified to test and demonstrate the three security events at the AFDX switch: (1) erroneous VL properties, (2) erroneous routes in the forwarding table, and (3) erroneous buffer sizes. To simulate these events, specific variables of the OMNeT++ model were manipulated, the results of which are discussed below.

4.1 Erroneous VL Properties

As mentioned, traffic policing function is conducted at the AFDX switch. Incoming frames are checked for conformance based on their VLID, which indicates their BAG, Lmax and transmission capacity properties. For this test case, the paper considers an unchecked/unverified event where the corresponding VL information is modified at the

switch – the VLs are conformant, however, the information used to confirm this at the switch is incorrect. It is expected then that the switch will drop these frames. A screenshot of the OMNeT++ simulator log depicts the result of this event (Fig. 6). As observed, a frame is dropped as it is “too old”, to indicate that it has exceeded its overall time on the network – a measure based on inconsistent VL properties at the switch. If left unchecked, sound frames will continue to be dropped, causing a DoS event, which, based on the attack tree (Fig. 4), can cause the intended message recipient(s) to go into safety shutdown, as it is being denied updated data.

```

** Event #1 t=0.795874504566 AFDXExampleNetwork.ES0.trafficSource1 (AFDXMessage:
** Event #2 t=0.795874504566 AFDXExampleNetwork.ES0.regulatorLogic (RegulatorLo
** Event #3 t=0.795874504566 AFDXExampleNetwork.ES0.redundancyController (Redund
** Event #4 t=0.795874504566 AFDXExampleNetwork.ES0.txQueueA (PassiveQueue, id=1
** Event #5 t=0.795874504566 AFDXExampleNetwork.ES0.txQueueB (PassiveQueue, id=1
** Event #6 t=0.795874504566 AFDXExampleNetwork.ES0.macA (MAC, id=153) on (afd
INFO (MAC)AFDXExampleNetwork.ES0.macA: dropping frame. It is too old: 0.00001088s
    
```

Fig. 6. Simulator log showing dropped packets at the switch due to perceived non-conformance

4.2 Erroneous Routes in the Forwarding Table of the Switch

The modules for the switch contain parameters for the allowed routes, which are represented in the format:

```

ES0.ethPortA <--> Eth100 <--> switchA.ethPort[0]
ES2.ethPortA <--> Eth100 <--> switchA.ethPort[0]
ES0.ethPortB <--> Eth100 <--> switchB.ethPort[1]
ES2.ethPortB <--> Eth100 <--> switchB.ethPort[1]
    
```

This example represents the valid paths from End System zero (ES0) to End system two (ES2). Essentially, the paths displayed can be assumed as extracts of the forwarding table in the OMNeT++ model, and a boolean variable, “isPathOK”, is used to determine the validity of the route for incoming packets at the switch. If a connected path/route cannot be detected, then the frame is dropped at the switch. This is observed in Fig. 7, which displays a screenshot of the simulator log depicting the results of an event where a connected path is not found. For this test case, the allowed routes were replaced with invalid ones. As observed, frames are dropped because a route cannot be determined based on the forwarding table at the switch. If left unchecked, this can cause the intended message recipient(s) to go into safety shutdown, as it is being denied up-to-date data.

```

INFO (MAC)AFDXExampleNetwork.switchA.switchPort[1].mac: dropping frame. switchPort[1] is not connected.
** Event #66 t=0.898571268484 AFDXExampleNetwork.switchA.switchPort[3].mac (MAC, id=63) on (afdx::A
INFO (MAC)AFDXExampleNetwork.switchA.switchPort[3].mac: dropping frame. switchPort[3] is not connected.
** Event #67 t=0.898571268484 AFDXExampleNetwork.switchB.switchPort[1].mac (MAC, id=225) on (afdx::A
INFO (MAC)AFDXExampleNetwork.switchB.switchPort[1].mac: dropping frame. switchPort[1] is not connected.
** Event #68 t=0.898571268484 AFDXExampleNetwork.switchB.switchPort[3].mac (MAC, id=233) on (afdx::A
INFO (MAC)AFDXExampleNetwork.switchB.switchPort[3].mac: dropping frame. switchPort[3] is not connected.
** Event #69 t=0.898571268484 AFDXExampleNetwork.switchA.switchPort[1].mac (MAC, id=55) on selfmac
    
```

Fig. 7. Simulator log showing dropped packet at the switch due to erroneous routes

4.3 Erroneous Buffer Size at the Switch

The modules for the switch Tx and Rx buffer queues have a “*capacity*” integer variable, which denotes the queue size. Where, if the capacity is greater than or equal to zero AND the queue length is greater than or equal to the “*capacity*”, the queue becomes full. All incoming packets are dropped until the queue has space for storing frames. This capacity was reduced significantly to simulate the effect of an insufficient buffer size. A screenshot of the simulator log depicts the results of this event (Fig. 8). As observed, the queue is detected as full, as such, frames are dropped at the switch. If left unchecked, the resulting service failure is as described in the previous cases.

```

** Event #1 t=0.795874504566 AFDXExampleNetwork.ES0.trafficSource1 (AFDXMess
** Event #2 t=0.795874504566 AFDXExampleNetwork.ES0.regulatorLogic (Regulato
** Event #3 t=0.795874504566 AFDXExampleNetwork.ES0.redundancyController (Re
** Event #4 t=0.795874504566 AFDXExampleNetwork.ES0.txQueueA (PassiveQueue,
INFO (PassiveQueue)AFDXExampleNetwork.ES0.txQueueA: Queue full! Job dropped.
** Event #5 t=0.795874504566 AFDXExampleNetwork.ES0.txQueueB (PassiveQueue,
INFO (PassiveQueue)AFDXExampleNetwork.ES0.txQueueB: Queue full! Job dropped.
** Event #6 t=0.898560388484 AFDXExampleNetwork.ES1.trafficSource1 (AFDXMess
** Event #7 t=0.898560388484 AFDXExampleNetwork.ES1.regulatorLogic (Regulato
** Event #8 t=0.898560388484 AFDXExampleNetwork.ES1.redundancyController (Re
** Event #9 t=0.898560388484 AFDXExampleNetwork.ES1.txQueueA (PassiveQueue,
INFO (PassiveQueue)AFDXExampleNetwork.ES1.txQueueA: Queue full! Job dropped.
** Event #10 t=0.898560388484 AFDXExampleNetwork.ES1.txQueueB (PassiveQueue,
INFO (PassiveQueue)AFDXExampleNetwork.ES1.txQueueB: Queue full! Job dropped.
** Event #11 t=1.255930758047 AFDXExampleNetwork.ES2.trafficSource1 (AFDXMes
** Event #12 t=1.255930758047 AFDXExampleNetwork.ES2.regulatorLogic (Regulat
** Event #13 t=1.255930758047 AFDXExampleNetwork.ES2.redundancyController (R
** Event #14 t=1.255930758047 AFDXExampleNetwork.ES2.txQueueA (PassiveQueue,
INFO (PassiveQueue)AFDXExampleNetwork.ES2.txQueueA: Queue full! Job dropped.
** Event #15 t=1.255930758047 AFDXExampleNetwork.ES2.txQueueB (PassiveQueue,
INFO (PassiveQueue)AFDXExampleNetwork.ES2.txQueueB: Queue full! Job dropped.

```

Fig. 8. Simulator log showing dropped packets at the switch due to flooding

4.4 Limitations of the OMNeT++ Model

Although the OMNeT++ model provided a useful environment for testing the performance of the AFDX network under the three test cases, it was found to have the following limitations. Not all the variables were initialized in a manner to generate the test cases. For instance, even though the logs show traffic being assigned parameters such as a VLID, changing this value does not have any effect on the simulation. Additionally, a single variable was used for more than one component. For example, changing the queue size “*capacity*” variable affected both switches instead of one, as this variable was a single instance that was shared by both. This restricted the extent to which the test cases reflected true events, as the manipulation on one switch should be an independent action. However, it must be noted that it is declared in the model documentation that the simulator does not implement the entire AFDX protocol, only the MAC layer, as such, some limitations are to be expected.

5 Considerations for Future Improvements AFDX

To remain a trustworthy system, AFDX networks require additional controls on the network or integrated as a part of the specification. However, as AFDX is used in critical Avionics industry and is being considered for further critical industries, consideration must be given to the safety requirements of these industries. For instance, the time-sensitivity of the message delivery must not be negatively impacted by the control. As such, resource-intensive controls are not feasible.

This paper describes a scenario where the integrity of the AFDX network is to be preserved as a prime means by which to ensure a dependable network. One example from the nuclear industry that focuses on this requirement is observed with OPANASec® from AREVA. OPANASec® is implemented as a network component that seamlessly integrates with and protects the integrity of programmable logic controllers (PLCs), through access control and integrity monitoring [19]. Of special note is that OPANASec® is a real-time solution, which provides an additional advantage of also preserving availability. Changes to the PLCs are identified through checksums of the monitored data, and once detected, are stored in a non-volatile diagnostics buffer. An alert (normally a signal light for the operators) is then created [19]. Further examples are observed with industrial switches that have built-in security modules that provide detective and preventive controls to protect unauthorized network events.

However, additional requirements are necessary for AFDX, as the solution must address security on a network-wide basis, and not per-system. Integrity controls must seek to ensure:

- known and authorized communication paths and trust relationships between systems
- known and authorized data frame properties (e.g. BAG, Lmax, etc.)
- observation of delay bounds, as prescribed by the formulae in the AFDX specification, Eqs. (1) and (2) [1].

$$\max_jitter \leq 40 \mu s + \frac{\sum_{j \in \{set\ of\ VLs\}} ((20 + Lmax_j) \times 8)}{Nb_w} \quad (1)$$

$$\max_jitter \leq 500 \mu s \quad (2)$$

As such, a small subset of controls is suitable for use with AFDX. However, this does not eliminate all categories of controls. For instance, cryptographic controls are considered resource intensive, and are usually not considered for use in critical industries. In fact, controls for confidentiality (primarily cryptography) are typically excluded from such industries. Nevertheless, with the introduction of and continued efforts to standardize light-weight cryptographic solutions, such as seen with IEC 29192 *Information technology – Security techniques – Lightweight cryptography*, cryptography is becoming a viable option for time-sensitive infrastructures. In fact, several researchers have demonstrated the minimal impact and significant benefits of introducing light-weight cryptographic controls into resource constrained and critical environments such as the smart grid, e-health systems, smart-vehicles and other IoT infrastructures

[20–22]. The results of these publications demonstrate that the integration of light-weight cryptography satisfied desirable security requirements of the environments, whilst also preserving the performance requirements of the same. This provides additional motivation for a similar integration of cryptographic controls in AFDX to guarantee trustworthy data exchange, that is, to preserve network integrity. With this intent, Message Authentication Codes (MACs) are considered as the prime option to preserve data integrity. Like digital signatures, MACs provide oversight for authentication and integrity. However, as MACs are typically generated using hash-functions or block cipher and use a symmetric key, this makes them faster than digital signatures [23].

To develop an integrity scheme for a time-sensitive switched Ethernet network, it is necessary to assess the available MACs in terms of their speed and security (robustness). Viable MACs are described in ISO/IEC 9797 *Information technology – Security techniques – Message Authentication Codes (MACs)* and the aforementioned ISO/IEC 29192. In this scheme, it is intended that the End Systems submit data across the network along with a corresponding checksum, whilst key management services (key generation, distribution and other processes in the lifecycle of a key) are provided at the AFDX switch. Further research is necessary to test and demonstrate this scheme for AFDX. As a guide to inform the performance of this new security scheme, results from works that analyze conventional AFDX implementations should be used as a baseline. For instance, Fig. 9 indicates suitable AFDX configuration data based on the results of worst-case end-to-end delay analysis [24]. These should serve as baselines to ascertain the impact of the data integrity scheme on an AFDX network. In that, following this implementation, the resultant changes must have minimal impact on a conventional AFDX implementation.

Bag (ms)	Number of VL	Frame length (bytes)	Number of VL
2	20	0-150	561
4	40	151-300	202
8	78	301-600	114
16	142	601-900	57
32	229	901-1200	12
64	220	1201-1500	35
128	255	> 1500	3

Fig. 9. AFDX configuration as an indicator of its performance

Further, concerning the maintenance of trustworthy configuration data, additional monitoring services can be implemented at the switch, where a MIB is already present. This can be expanded to consider existing integrity solutions such as whitelisting. In that, in typical AFDX implementations the switch hosts configuration tables for each End System – this can be expanded to include two copies, where one is used as a baseline to detect changes. This baseline configuration can be whitelisted to protect its integrity and should be further stored in a separate partition. Through SNMP, inconsistencies can be detected, and signaled in a similar fashion to typical errors described

in the AFDX specification. Again, comprehensive modelling and testing is required to ensure that AFDX dependability and security attributes are sufficiently protected, with the integration of these solutions.

6 Conclusion

The design of the Avionics Full Duplex Switched Ethernet (AFDX) realizes mechanisms to prevent, tolerate, remove and forecast faults, to preserve its deterministic and fault-tolerant properties. However, as with IT and OT environments, insider threats can weaken these intrinsic protection mechanisms, and allow the proliferation of attacks. To detect and deter insider attacks, proposed solutions include an integrity monitoring solution integrated as a part of the AFDX switch MIB, the use of light-weight cryptography and of a whitelisting solution. However, consideration must be given to the AFDX specifications that should be met to realize a compliant and viable configuration. Furthermore, consideration must be given to safety and security requirements of the critical environments where AFDX will be deployed. The AREVA SMARTEST project aims to also contribute to the on-going effort of ‘smart’ testing and ‘smart’ security, to ensure reliable and resilient systems.

Acknowledgements. Some of the addressed topics are being elaborated as part of AREVA GmbH’s participation in the “SMARTEST” R&D (2015–2018) with German University partners, partially funded by German Ministry BMWi.

References

1. Aeronautical Radio Inc. (ARINC). Specification 664: aircraft data network, part 7 – deterministic networks, 23 September 2009
2. Thirumeni, P., Ghoshhajra, M., Ananda C.M.: Lessons learned in software implementation of ARINC 664 protocol stack in Linux. In: Proceedings of International Conference on Circuits, Communication, Control and Computing (I4C) (2014)
3. AIM GmbH. AFDX training: AFDX workshop, October 2010. http://www.afdx.com/pdf/AFDX_Training_October_2010_Full.pdf. Accessed 25 Feb 2018
4. GE Fanuc. Embedded systems AFDX/ARINC 664 protocol tutorial, January 2011. http://www.cems.uwe.ac.uk/~a2-lenz/n-gunton/worksheets/AFDX_Tutorial_WP.pdf. Accessed 25 Feb 2018
5. IEEE. IEEE Std 1722-2016: (revision of IEEE Std 1722-2011) - IEEE standard for a transport protocol for time-sensitive applications in bridged local area networks, 16 December 2016
6. TTEch. TTEthernet theory and concepts, 27 August 2015. http://etr2015.irisa.fr/images/presentations/TTEthernet_ETR_2015_Rennes.pdf. Accessed 22 Feb 2018
7. Avizienis, A., Laprie, J., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Comput.* **1**(1), 11–33 (2004). <http://ieeexplore.ieee.org/document/1335465/>. Accessed 01 Mar 2018
8. Anand, M., Dajani-Brown, S., Vestal, S., Lee, I.: Formal modeling and analysis of the AFDX frame management design. In: Proceedings of 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2006), pp. 393–399 (2006)

9. Saha, I., Roy, S.: A finite state modeling of AFDX frame management using spin. In: Brim, L., Haverkort, B., Leucker, M., van de Pol, J. (eds.) FMICS 2006. LNCS, vol. 4346, pp. 227–243. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70952-7_15
10. Song, D., Zeng, X., Ding, L., Hu, Q.: The design and implementation of the AFDX network simulation system. In: Proceedings of International Conference on Multimedia Technology (ICMT) (2010)
11. Tripwire: Insider threats as the main security threat in 2017. <https://www.tripwire.com/state-of-security/security-data-protection/insider-threats-main-security-threat-2017/>. Accessed 22 Feb 2018
12. Dury, S: Employees still the biggest threat to enterprise security. <https://www.digicert.com/blog/employees-still-the-biggest-threat-to-enterprise-security/>. Accessed 22 Feb 2018
13. van Zadelhoff, M.: The biggest cybersecurity threats are inside your company. <https://hbr.org/2016/09/the-biggest-cybersecurity-threats-are-inside-your-company>. Accessed 22 Feb 2018
14. Saglietti, F., Meitner, M., von Wardenburg, L., Richthammer, V.: Analysis of informed attacks and appropriate countermeasures for cyber-physical systems. In: Skavhaug, A., Guiochet, J., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2016. LNCS, vol. 9923, pp. 222–233. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45480-1_18
15. Rejeb, N., Ben Salem, A.K., Ben Saoud, B.: AFDX simulation based on TTEthernet model under OMNeT++. In: Proceedings of 2017 International Conference on Advanced Systems and Electric Technologies (IC ASET), pp. 423–429 (2017)
16. Varga, A., Hornig, R.: Avionics full-duplex switched Ethernet model for OMNeT++, 20 February 2012. <https://github.com/omnetpp/afdx>. Accessed 05 Mar 2018
17. Simulcraft, Inc.: OMNEST - OMNeT++ comparison. <https://omnest.com/comparison.php>. Accessed 05 Mar 2018
18. Simulcraft, Inc. Performance modeling library. <https://omnest.com/queueinglib.php>. Accessed 05 Mar 2018
19. Parekh, M., Gao, Y., Gupta, D., Luschmann, C.: OPANSec – security integrity monitoring for controllers. In: Proceedings of 46. Jahrestagung der Gesellschaft für Informatik, pp. 547–557 (2016)
20. Khemissa, H., Tandjaouiy, D.: A lightweight authentication scheme for e-health applications in the context of Internet of Things. In: Proceedings of 9th International Conference on Next Generation Mobile Applications, Services and Technologies, pp. 90–95 (2015)
21. Fouda, M.M, Fadlullah, Z.M., Kao, N., Lu, R., Shen, X.: Towards a light-weight message authentication mechanism tailored for smart grid communications. In: Proceedings of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 1018–1023 (2011)
22. Mundhenk, P., Steinhorst, S., Lukasiewicz, M., Fahmy, S., Suhaib, A., Chakraborty, S.: Lightweight authentication for secure automotive networks. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1–4 (2015)
23. Paar, C., Pelzl, J.: Understanding Cryptography: A Textbook for Students and Practitioners, pp. 319–330. Springer, Heidelberg (2010). <https://doi.org/10.1007/978-3-642-04101-3>
24. Charara, H., Scharbarg, J.-L., Ermont, J., Fraboul, C.: Methods for bounding end-to-end delays on an AFDX network. In: Proceedings of the 18th Euromicro Conference on Real-Time Systems, Washington, D.C., USA, pp. 193–202 (2006)

Fault Tolerance



Efficient On-Line Error Detection and Mitigation for Deep Neural Network Accelerators

Christoph Schorn^{1,2}(✉), Andre Guntoro¹, and Gerd Ascheid²

¹ Corporate Research, Robert Bosch GmbH, Renningen, Germany
{christoph.schorn, andre.guntoro}@de.bosch.com

² Institute for Communication Technologies and Embedded Systems,
RWTH Aachen University, Aachen, Germany
gerd.ascheid@ice.rwth-aachen.de

Abstract. The use of deep neural network accelerators in safety-critical systems, for example autonomous vehicles, requires measures to ensure functional safety of the embedded hardware. However, due to the vast computational requirements that deep neural networks exhibit, the use of traditional redundancy-based approaches for the detection and mitigation of random hardware errors leads to very inefficient systems. In this paper we present an efficient and effective method to detect critical bit-flip errors in neural network accelerators and mitigate their effect at run time. Our method is based on an anomaly detection in the intermediate outputs of the neural network. We evaluate our method by performing fault injection simulations with two deep neural networks and data sets. In these experiments our error detector achieves a recall of up to 99.03% and a precision of up to 97.29%, while requiring a computation overhead of only 2.67% or less.

1 Introduction

Deep neural networks (DNNs) have surpassed human-level performance in some very challenging decision tasks (e.g. [13, 24, 27]). In the field of machine vision, DNNs are outperforming traditional approaches with hand-crafted feature extractors [20]. This makes them a key component for autonomously operating systems. The computational complexity of DNNs for perception tasks calls for the development of dedicated accelerators which meet the energy efficiency and speed requirements of these applications [5]. At the same time, since environmental perception is at the heart of autonomous systems, its associated hardware components are regarded as highly safety-critical. For example, an environmental perception system for highly automated driving in complex traffic scenarios will presumably have to fulfill the requirements of the highest automotive safety integrity level (ASIL) D, as defined in the ISO 26262 standard for functional safety of road vehicles [17]. This corresponds to an overall random failure rate of less than 10 FIT¹ for the responsible hardware. Yet the ongoing trend towards

¹ 1 failure-in-time (FIT) = one failure in one billion device operating hours.

shrinking structure widths and lower operating voltages increases the susceptibility of modern integrated circuits to random hardware errors [1, 14]. Furthermore, the growing memory size of electronic systems increases the per device FIT rate due to radiation induced soft errors [16]. Since DNN accelerators require large amounts of buffer memories and are extensively employing data reuse, efficient and effective soft error mitigation techniques are needed to avoid silent data corruption (SDC) [22].

The contribution of our research is a novel error detection and mitigation method for DNN accelerators. This method has three major advantages compared to the state-of-the-art. Firstly, it achieves very high detection recall rates for random bit-flip errors, and thus it can effectively lower the risk of SDC in DNN accelerators. Secondly, it requires only a small fraction of additional multiply accumulate (MAC) operations, while redundancy-based methods often have a computation overhead of more than 100%. And thirdly, it not only detects critical bit-flip errors, but also is able to recover the correct output of the DNN accelerator in most cases.

The remainder of this paper is structured as follows. The next section gives an overview about DNNs, especially those for image classification and the basics of DNN accelerators. Then we introduce our framework for the error detection and mitigation. In the experiments section, we evaluate our method with two image classification DNNs and discuss our findings. Finally, we compare our method with existing approaches from the related literature and round off our paper with a summary and conclusions.

2 Preliminaries

2.1 Deep Neural Networks

A DNN can be described as a directed graph that is mainly constituted of a set of interconnected *neurons*. Neurons are the fundamental computational units in DNNs. Each neuron calculates a weighted sum of its inputs and applies a nonlinear activation function on this sum. Neurons are grouped in layers, which share a common set of inputs and outputs. Although our methods are in principle not limited to a specific kind of network architecture, we focus on (convolutional) feed-forward networks here, since these are widespread in the computer vision domain [9]. In a feed-forward network with N layers, the i^{th} layer output \mathbf{y}_i is a function of the layer weights $\boldsymbol{\theta}_i$ and the preceding layer output

$$\mathbf{y}_i = \mathbf{f}_i(\boldsymbol{\theta}_i, \mathbf{y}_{i-1}). \quad (1)$$

The first layer input \mathbf{y}_0 equals the input data to the network (e.g. an image) and the last layer output \mathbf{y}_N represents the task result (e.g. predicted class-probabilities for the input image). The weights of the network are determined in an initial *training phase*, by minimizing a loss function of the network over a given set of training data and back-propagating the respective error terms through the network. After the weights have been determined, the DNN is used

in the *inference phase* to perform predictions on new data samples, e.g. to classify sensor data in an autonomous vehicle.

Our goal in this paper is to detect errors in the computation of a DNN on-line, during the inference phase, and to mitigate their effect on the network output. We do not consider errors during the training phase, since training usually takes place off-line, on different hardware and before the network is applied in a safety-critical context. Furthermore, SDC in the network weights caused by errors during training can be detected with existing validation methods, e.g. by measuring the network accuracy on a test data set.

2.2 Convolutional Neural Networks

Vision-based environmental perception is a key application of DNNs in autonomous vehicles, which is why we focus on image processing neural networks in the following. In this area, convolutional neural networks (CNNs) [21] have, since their first success on large-scale visual recognition challenges [19], become the gold standard approach [9]. CNNs are a specific type of DNN. The main ideas behind them are the use of learned filters with local connectivity, weight sharing, and a deep network architecture with intermediate pooling stages [20].

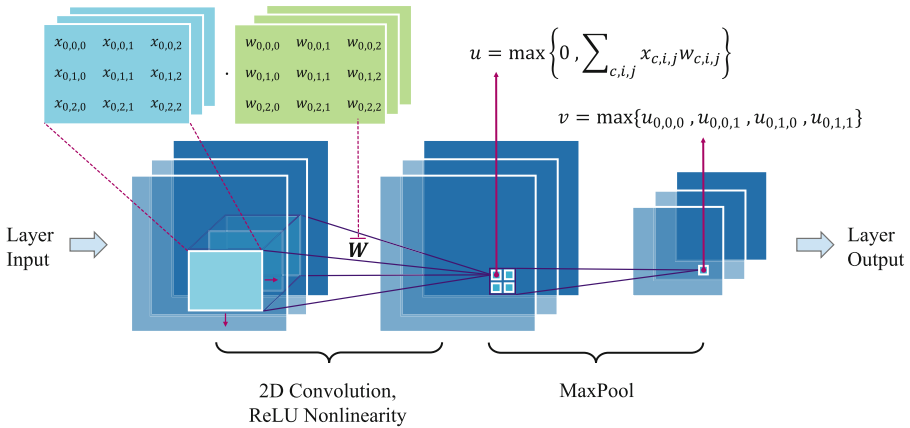


Fig. 1. Schematic depiction of the individual steps taking place in the convolutional and pooling stages of a CNN.

Each convolutional layer of a common CNN performs two-dimensional convolutions of multiple filter kernels over the layer input (see Fig. 1). Layer input and output are three-dimensional, with two spatial image dimensions and one feature dimension. In the first layer input, the features correspond to the color channels (e.g. RGB) of the image. The convolutions are performed along the two spatial axes. In this way, for each filter kernel a two-dimensional feature map is generated. The pixels in the feature maps can be regarded as neuron outputs,

since they represent the weighted sum of an input subset, followed by a nonlinear activation function. State-of-the-art CNNs typically use rectified linear units (ReLUs) [32] as activation function. The filter kernel coefficients are learned in a training phase, just like the weights of any other neural network architecture. As shown in Fig. 1, a maximum-pooling stage is appended to some of the convolutional layers in order to reduce the spatial dimensions. The last few layers of a CNN are often realized as fully connected layers instead of convolutional layers. In a fully connected layer each neuron has weighted connections to all the outputs of the preceding layer. These layers usually also use ReLU activations, except for the final output layer. For classification tasks the output layer typically has a softmax activation function that transforms the output into class probabilities.

2.3 Deep Neural Network Accelerators

Architecture. DNNs are often computed on graphics processing units (GPUs), which are well suited for the highly parallel structure of the underlying operations of a neural network. However, when embedded applications are regarded, commercial off-the-shelf GPUs often do not fulfill the energy-efficiency, throughput and reliability requirements. Energy-efficiency and throughput have been addressed by some recent publications of dedicated DNN accelerator architectures (e.g. [6, 8, 10, 25]). The common ideas behind these accelerators are to exploit parallelism and to optimize dataflow and local data reuse, since external memory access is costly, both in terms of power consumption and latency [7].

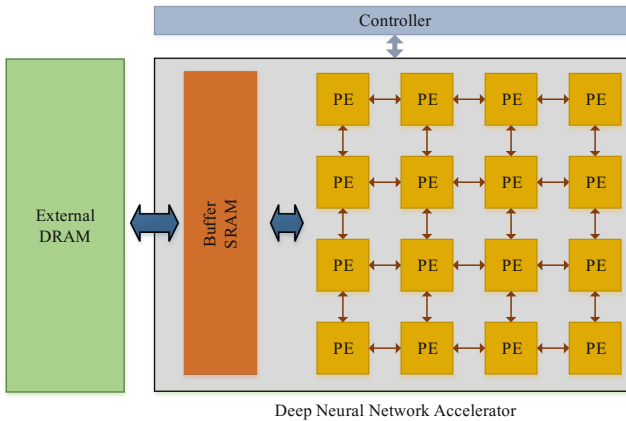


Fig. 2. Generic architecture of a dataflow-based DNN accelerator.

Figure 2 shows the high-level view of a generic DNN accelerator. Parallelism is achieved by using an array of interconnected processing elements (PEs), which perform the mathematical operations of the neural network. These are mainly

MAC operations. PEs usually have local memories, in which they accumulate partial results. A shared on-chip static random-access memory (SRAM) buffer is used to distribute data to the PEs and store intermediate results. For the DNN input and output, as well as larger intermediate results, a transfer to an external dynamic random-access memory (DRAM) is required.

Quantization. Because GPUs have been historically optimized for precise graphical computations, DNNs on GPUs typically use the 32-bit IEEE 754 floating-point (FLOAT) arithmetic. However, the bit-width of the data representation has a large effect on the required design area and power consumption, which is why dedicated DNN accelerators tend to use a lower precision quantization for the weight and neuron activation representation of the network [12]. In fact, it has been shown that by adapting the training procedure, classification networks can be quantized even down to a 1-bit data representation for weights and activations in the inference phase without a significant degradation of the DNN output accuracy [15,31].

The internal data representation has a strong influence on the propagation of bit-flip errors through the DNN accelerator [22]. This is why we test our methods with a realistic data representation for dedicated DNN accelerators. Since very low bit-width quantization requires a careful adjustment of the training procedure, we instead employ an approach similar to [11], which lets us quantize pre-trained DNNs from FLOAT to a variable fixed-point format with little to no output accuracy degradation. Throughout our experiments, we use an 8-bit variable fixed-point representation.

3 Error Detection and Mitigation Framework

3.1 Fault Model

We focus on transient faults in the form of random bit-flips in the buffer memories and data path of a DNN accelerator. These faults can result in errors in the values computed by the DNN and consequently lead to SDC. We define *critical errors* as those errors, which change the originally correct classification output of the CNN for a given input image. Our goal is to detect these critical errors. Furthermore, we want to mitigate their effect, i.e. recover the correct classification output. We limit our analysis to a single bit-flip hardware fault per inference of the accelerator, since the inference time of a DNN accelerator typically amounts to only a few milliseconds or less [5]. Thus, we neglect the probability of two independent bit-flips during one inference.

The efficient detection of random memory and data path bit-flips in DNN accelerators is an important concern for several reasons:

- DNN accelerators possess a high vulnerability to these errors, since they typically transfer considerable amounts of data between memory buffers and PEs for each single classification. This can lead to FIT rates which violate international safety-standards, such as ISO 26262 [22].

- The fact that these errors occur at any random point in time, while the accelerator is operating, makes them difficult to detect by self-test methods. Moreover, an immediate detection and mitigation is required for autonomous vehicles, since their action planning relies on the correct output of the DNN. The misclassification of a stop sign can for example lead to an accident with another vehicle.
- Existing error detection and correction schemes for DNN accelerators either result in large area and computation overheads, or are unsuited for DNN accelerators with low bit-width quantization [22].

3.2 Anomaly Detection in Intermediate Feature Activations

A CNN produces in each layer a set of feature activations for a given input image. The level of activation (i.e. the respective neuron output value) indicates how present a certain feature is in the image. In the convolutional layers, the feature activations for each filter kernel are two-dimensional maps, i.e. the presence of features is indicated for different locations in the input image. In the fully connected layers, each neuron represents a certain feature. Which features the CNN extracts is determined by the network itself during the training phase.

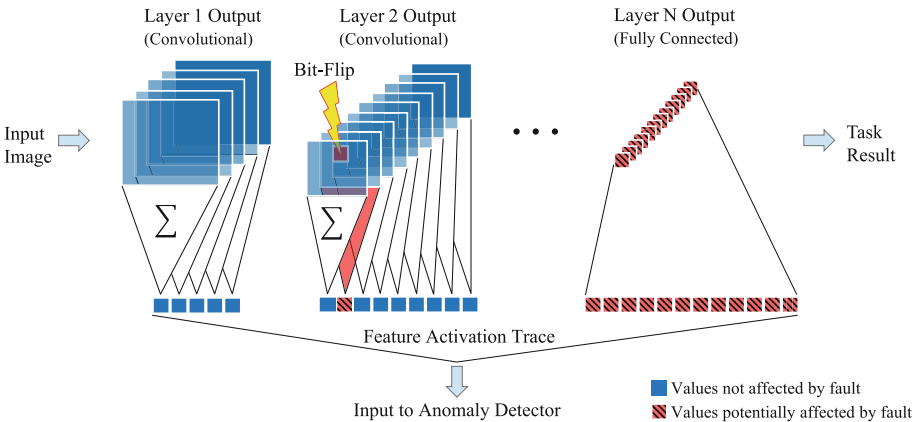


Fig. 3. Anomaly detection in intermediate feature activations.

A single bit-flip fault in the memory or data path of a DNN accelerator can result in a change of one feature activation in the layer which is currently being processed by the accelerator. Furthermore, all feature activations of the following layers are potentially affected as well, since the error can propagate through the network. This property is the motivation for our error detection concept, as visualized in Fig. 3. We generate a *feature activation trace* for a given input image of the CNN by concatenating the feature activations of all layers of the network. In the convolutional layers, we summarize the activation values of the

feature maps along the spatial axes to generate a single value for each feature. The central idea behind our method is, that a critical bit-flip error results in an anomaly in the feature activation trace that can be detected by an anomaly detector. Moreover, the hypothesis is that even under the influence of a bit-flip error, enough information about the input image features remains present in the feature activation trace to allow for a recovery of the desired task result (i.e. the correct image classification).

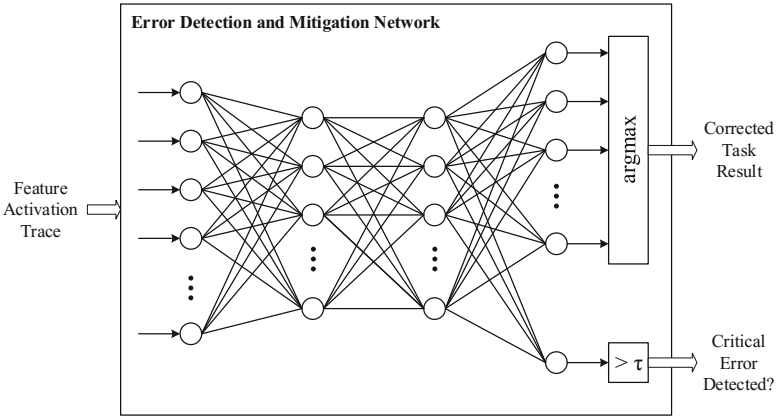


Fig. 4. Architecture of the error detection and mitigation network.

Neural networks are well-suited for detecting anomalies in high-dimensional data. This is why we choose to employ a small feed-forward neural network for detecting critical errors in the feature activation traces of the CNN that performs the actual classification task. As shown in Fig. 4, the network is designed to predict both, if a critical error is present or not, as well as a corrected task result for the image classifier. It would also be possible to use two separate neural networks for error detection and error correction respectively, but the combined network can solve the two tasks more efficiently. The error detection and mitigation network (EDMN) has two common fully connected layers with ReLU activation functions. The number of input neurons of the network corresponds to the length of a feature activation trace of the supervised CNN. The output of the second fully connected layer is connected to two different output layers, one for the error detection and the other one for the correction prediction. The detection part has only a single output neuron with a sigmoid activation function. This outputs a value between 0 and 1, indicating the probability for a critical error being present. Based on the comparison with a threshold τ , it is decided if a critical error is present or not. Throughout our experiments we set $\tau = 0.5$. The correction prediction output layer has as many output neurons as the image classifier CNN, with a softmax activation function to assign probabilities to each of the possible image classes. An argmax function is used to select the predicted

class based on the output neuron that gives the highest probability score. The correction output is only taken, if the detector indicates that a critical error was detected.

3.3 Training the Error Detection and Mitigation Network

Training the EDMN requires a set of labeled training data, consisting of feature activation traces and the appropriate target outputs for the error detection and error correction. For generating a training data set, random bit-flips are simulated in different neuron outputs of the quantized CNN for different input images. These bit-flips mimic random hardware faults that can occur when the neuron output values are stored in buffer memories or transmitted over a data bus. The corresponding feature activation traces are recorded and taken as input data to the EDMN. Since we consider only single bit-flips, not more than one fault is injected for each inference of the CNN. Our method is not limited to single bit-flips though, and can be adapted to a different fault model for a given DNN accelerator, if necessary.

To speed up training data generation, faults are preferably injected in the least fault tolerant neurons of each layer, based on a neuron resilience prediction [26]. The input images are randomly drawn from the training data set of the CNN. Additionally, random data augmentations [28] are applied to the input images to further increase the variability of the input data. This helps the EDMN to better distinguish between benign variations in the feature activation traces, caused by input data variation, and a malicious feature activation variation, caused by a random hardware fault.

In order to successfully predict faults and corrected task results, the EDMN has to learn to distinguish between three types of feature activation traces:

1. Feature activation traces, where no hardware fault is present.
2. Feature activation traces, where a hardware fault is present, but uncritical (i.e. the CNN output stays the same).
3. Feature activation traces, where a hardware fault is present and results in a critical error (i.e. the CNN output changes from correct to wrong).

We generate equal shares of training samples for each of the three cases. For each training sample, we also generate target labels for the error detection and correction outputs. The error detection target label is 0 for the first two cases and 1 for the third case. The error correction target labels correspond to the labels of the input images of the CNN.

The EDMN performs both, an error detection, as well as the prediction of an error corrected output, using a single neural network. This means that the network parameters have to be jointly optimized for the two tasks during the training phase. Each task represents a classification problem, for which a cross-entropy cost function can be used.² For the training, we add both cost functions together and minimize the combined cost function on the generated training data using stochastic gradient descent.

² A basic introduction to neural network training concepts can be found e.g. in [3].

3.4 Safeguarding the Error Detection and Mitigation Network

In order to achieve a reliable error detection and mitigation for a DNN accelerator with the proposed second neural network, the EDMN itself has to be safeguarded against random hardware faults. However, since the additional network is much smaller and requires far fewer operations than the image classifier CNN, it can be computed on a separate hardware module that is protected with classical measures, such as triple modular redundancy (TMR). Though the cost for computing the EDMN is considerably increased this way, the overall overhead is much smaller compared to a fully redundant computation of the large CNN.

Besides safeguarding against hardware faults, it also has to be ensured that the error detection and correction performance of the EDMN measured at test time is a good estimator for the performance in the field. Two main aspects are important in this regard. Firstly, the fault model used for generating the training data of the EDMN should as accurately as possible describe the types and distribution of faults in the real hardware. And secondly, the training data of the underlying monitored DNN, which is also the basis for the EDMN training data generation, should be sufficient in terms of scenario coverage and distribution. The second requirement is closely related to ensuring the safety of the intended functionality (SOTIF) of the monitored DNN [4].

4 Experiments

We evaluate our methods on two different classification networks. The first one is an All-CNN [28], which only uses convolutional and no fully connected layers. This network is trained on the popular CIFAR-10 classification benchmark, which consists of 32×32 pixel RGB images divided into ten different classes [18]. The second DNN is a Road Sign Recognition (RSR) network trained on the German Traffic Sign Recognition Benchmark (GTSRB), which contains RGB images of 43 different types of traffic signs [29]. The images are preprocessed to have a resolution of 48×48 pixels, before they are fed into the classifier. Additionally, we perform a fixed-point quantization for both networks with layer-wise variable fixed-point scaling factors, similar as in [11]. A summary of the properties of the two DNNs is given in Table 1.

For both networks we use an EDMN with 256 neurons in each of the first two fully connected layers. The correction output layer of the EDMN has 10 neurons for the CIFAR-10 All-CNN and 43 neurons for the RSR network. In each experiment the EDMN is optimized using stochastic gradient descent with a mini-batch size of 256 samples, 20 epochs, Nesterov momentum of 0.9, initial learning rate of 0.01 and a learning rate decay of 10^{-6} per batch. All experiments are conducted with our own fault injection simulation environment for deep neural networks, which makes use of GPU acceleration for the simulations.

Table 1. Networks used for the evaluation.

Network	Dataset	Layers	Features per layer	Accuracy (8-bit)
All-CNN	CIFAR-10	9 conv.	96, 96, 96, 192, 192, 192, 192, 192, 10	88.43%
RSR	GTSRB	6 conv., 2 fully conn.	32, 32, 64, 64, 128, 128, 512, 43	98.57%

4.1 CIFAR-10

The CIFAR-10 training data set consists of 50 000 images. With these training images and the All-CNN, we create training set of 711 999 feature activation traces, by randomly choosing several bit-flip error positions in the intermediate feature activations for each image. For each fault position, a random data augmentation, consisting of an image shift, zoom, rotation, color channel shift and random horizontal flip, is applied to the input image, to increase the feature activation trace variety. The training set has equal amounts of samples for the three cases described in Sect. 3.3. Each trace sample has a length of 1258, which corresponds to the sum of all layer features of the All-CNN.

In order to evaluate the performance of our EDMN after the training, we create a separate test set. This is created similarly to the training set, with the difference that it is based on the separate 10 000 test images of the CIFAR-10 dataset and no data augmentations are applied. We create a total number of 70 368 test samples.

4.2 Single Image Road Sign Recognition

For the RSR network we consider two different cases. The first one is the classification of single, independent road sign images by the DNN accelerator (RSR-single). In this case, the feature activation trace training and test sets are generated similarly to the CIFAR-10 data, with the only difference that horizontal flips are removed from the random data augmentations, since these would change the meaning of some traffic signs. Out of the 39 210 training images and 12 630 test images of the GTSRB data set, we generate 135 648 training and 47 013 test feature activation traces with the RSR network.

4.3 Sequential Road Sign Recognition

The second case, which we consider for the RSR network, is the processing of sequential input data (RSR-sequence). This is a realistic scenario for real-world applications, since a driving car typically captures several sequential images, while approaching a road sign. We expect, that if the EDMN sees a sequence of two feature activation traces from similar input images, it can more easily detect an error in one of them and recover the correct classification result.



Fig. 5. Example of two consecutive road sign image captures from the GTSRB dataset.

The GTSRB training dataset contains more than 1700 traffic sign instances and for each instance a sequence of 30 consecutive images that were recorded while approaching the traffic sign [29]. Figure 5 shows an example of two sequential images. The small differences between the two images will lead to slightly different feature activation traces. However, we expect that the EDMN can distinguish between the changes resulting from slight modifications of the input image and the changes resulting from a random bit-flip.

For our experiment, we randomly sample multiple sequences out of this dataset, each containing two consecutive images, and feed them into the RSR network. The resulting two feature activation traces per sequence are then concatenated before they are given to the EDMN. The input layer of the EDMN consequently has twice as many input neurons as before. The number of neurons in the other layers is kept constant. Bit-flips are only injected during the inference of the second image of the sequence, which represents the most current video frame captured by the camera. As before, equal shares of traces without hardware faults, traces with a bit-flip in the second sequence image that do not lead to a misclassification and traces with a bit-flip that lead to a misclassification of the originally correctly classified image are generated.

4.4 Results

The results of our experiments are summarized in Table 2. To evaluate the detection performance of the EDMN, we compute three different metrics:

1. Recall: The number of correctly detected critical errors divided by the total number of samples with critical error present in the test data set.
2. Precision: The number of correctly detected critical errors divided by the total number of detected critical errors (including false positives).
3. False positive rate (FPR): The number of samples incorrectly classified as critical errors divided by the total number of samples without critical error.

All three experiments result in a high detection recall rate. A similar behavior can be seen for the detection precision. In general, the results are better for the

Table 2. Experimental results.

	All-CNN	RSR-single	RSR-sequence
Detection recall	96.16%	97.64%	99.03%
Detection precision	90.29%	96.32%	97.29%
Detection FPR	5.17%	1.86%	2.75%
Resulting classification accuracy on test data without critical error	89.00%	98.75%	98.79%
Resulting classification accuracy on test data with critical error	62.01%	71.90%	85.43%
Computation overhead for EDMN with TMR (MAC operations)	0.41%	1.49%	2.67%

RSR experiments. This can be attributed to the fact that the RSR network itself shows a better classification performance than the All-CNN on CIFAR-10. Moreover, as expected, the EDMN performs better, if it gets the information from two sequential feature activation traces. Considering the FPRs, one might be concerned that too many of the valid outputs of the monitored network are rejected. However, samples classified as erroneous are in fact not rejected in our approach, but instead the EDMN tries to mitigate the error by providing a corrected task output for the monitored network.

To evaluate the error correction performance, we compute the resulting accuracies on the image classification tasks with the EDMN in place. As shown in Table 2 we consider two different cases. In the first case, we take only test samples that do not have a critical error. In this case, the EDMN incorrectly predicts critical errors for some of the samples (false positives) and assigns a corrected task output to these samples. Nevertheless, it can be observed that in all three experiments the overall resulting classification accuracies on test data without critical errors lies slightly above the original classification accuracies of the quantized CNNs. This indicates that despite the false positives in the error detection, the image classification accuracy is not degraded by the error correction. In the second case, we take only test samples that have a critical error. Without the EDMN in place, these samples would all lead to a wrong image classification and consequently the classification accuracy would be zero. Our results show, that the EDMN can recover the correct classification for at least 62% of the test samples in this case.

The computation overheads for the EDMNs in Table 2 are computed in terms of additional MAC operations in relation to the MAC operations of the CNN. Complete TMR is assumed for the computations of the EDMN, which offers a high protection of the EDMN itself. Still, the computation overheads are only a small fraction of the computations needed for the CNNs.

5 Related Work

Neural networks have already been used for on-line error detection and diagnosis in complex systems for a while (e.g. [2]). However, to the best of our knowledge, there is no previous work that considers the use of a neural network to detect hardware faults in another neural network.

A slightly related concept can be found in [23]. The authors connect a second neural network to a DNN, to detect whether there were adversarial perturbations [30] in the input image. Nevertheless, this context is different from ours in two ways. Firstly, adversarial perturbations only occur in the input data of the DNN, while random hardware errors can occur at any position in the network. Secondly, adversarial perturbations are not random, but in fact result from an optimization procedure that aims at fooling the DNN.

The closest related work to our research is the paper by Li et al. [22]. They study the error propagation in a modern DNN accelerator architecture by performing bit-flip fault injections in the memory and data path. Furthermore, they also propose an error detection as well as an error mitigation method. However, their approaches have some significant drawbacks compared to our method. Their detection method is based on the simple detection of activations that are below or above a certain value range that is considered to be normal for the activation values. As they have shown, this method works well, if the activation values utilize only a minor fraction of the dynamic range that the data type provides. However, it is unlikely for a dedicated DNN accelerator to not fully utilize the dynamic range of its data types. On the contrary, only a full utilization of the data type range allows for a maximum quantization and thus an energy efficient design of the accelerator. But in this case, their error detection method fails, since erroneous values are not distinguishable from regular values through a simple threshold comparison anymore. In contrast, our detection method also works for neural networks with low bit-width quantization. Their error mitigation method is based on hardening the data path against random hardware faults. This is orthogonal to our method and can be additionally used. However, the significant overheads resulting from the hardening might not be necessary, since our detector already offers a high detection recall and is able to recover from errors.

6 Conclusion

We have proposed a novel method for an efficient and effective on-line error detection and mitigation in DNN accelerators. To the best of our knowledge, other existing methods either require larger computational overheads or do not achieve the same level of error detection and mitigation performance. We tested our method with random bit-flip injections in the activations of two neural networks, since memory bit-flip errors are critical for DNN accelerators. Nevertheless, we expect that our method can be extended to other types of hardware errors. Moreover, we have seen that the usage of sequential information from a

time series of images can boost the performance of our method. We think that temporal information can be further exploited, both from a safety and efficiency perspective, and regard this as an interesting topic for future research.


References

1. Aitken, R., Fey, G., Kalbarczyk, Z.T., Reichenbach, F., Sonza Reorda, M.: Reliability analysis reloaded: how will we survive? In: Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 358–367 (2013)
2. Bernieri, A., Betta, G., Liguori, C.: On-line fault detection and diagnosis obtained by implementing neural algorithms on a digital signal processor. *IEEE Trans. Instrum. Meas.* **45**(5), 894–899 (1996)
3. Bishop, C.M.: *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York (2006)
4. Burton, S., Gauerhof, L., Heinzemann, C.: Making the case for safety of machine learning in highly automated driving. In: Tonetta, S., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2017. LNCS, vol. 10489, pp. 5–16. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66284-8_1
5. Canziani, A., Paszke, A., Culurciello, E.: An Analysis of Deep Neural Network Models for Practical Applications. CoRR abs/1605.07678 (2016)
6. Chen, Y.H., Emer, J., Sze, V.: Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks. In: Proceedings of the 43rd International Symposium on Computer Architecture, ISCA 2016, pp. 367–379 (2016)
7. Chen, Y.H., Emer, J., Sze, V.: Using dataflow to optimize energy efficiency of deep neural network accelerators. *IEEE Micro* **37**(3), 12–21 (2017)
8. Du, Z., et al.: ShiDianNao: shifting vision processing closer to the sensor. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture, ISCA 2015, pp. 92–104 (2015)
9. Gu, J., et al.: Recent advances in convolutional neural networks. *Pattern Recogn.* **77**, 354–377 (2018)
10. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: Proceedings of the 32nd International Conference on Machine Learning, pp. 1737–1746 (2015)
11. Gysel, P.: Ristretto: hardware-oriented approximation of convolutional neural networks. Master Thesis, University of California, Davis (2016)
12. Hashemi, S., Anthony, N., Tann, H., Bahar, R.I., Reda, S.: Understanding the impact of precision quantization on the accuracy and energy of neural networks. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 1474–1479 (2017)
13. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. In: IEEE International Conference on Computer Vision (ICCV), pp. 1026–1034 (2015)
14. Henkel, J., et al.: Reliable on-chip systems in the nano-era. In: 50th Annual Design Automation Conference, pp. 695–704 (2013)
15. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks. In: Advances in Neural Information Processing Systems (2016)
16. Ibe, E., Taniguchi, H., Yahagi, Y., Shimbo, K.I., Toba, T.: Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule. *IEEE Trans. Electron Devices* **57**(7), 1527–1538 (2010)

17. ISO: Road vehicles—Functional safety (2011)
18. Krizhevsky, A.: Learning multiple layers of features from tiny images. Master Thesis, University of Toronto (2009)
19. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **25**, 1097–1105 (2012)
20. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**(7553), 436–444 (2015)
21. LeCun, Y., et al.: Handwritten digit recognition with a back-propagation network. In: *Advances in Neural Information Processing Systems*, pp. 396–404 (1990)
22. Li, G., et al.: Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2017)
23. Metzen, J.H., Genewein, T., Fischer, V., Bischoff, B.: On detecting adversarial perturbations. In: *International Conference on Learning Representations (ICLR)* (2017)
24. Mnih, V., et al.: Playing Atari with Deep Reinforcement Learning. CoRR abs/1312.5602 (2013)
25. Peemen, M., Setio, A.A.A., Mesman, B., Corporaal, H.: Memory-centric accelerator design for convolutional neural networks. In: *IEEE 31st International Conference on Computer Design (ICCD)*, pp. 13–19 (2013)
26. Schorn, C., Guntoro, A., Ascheid, G.: Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators. In: *Design, Automation and Test in Europe Conference and Exhibition (DATE)* (2018)
27. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. *Nature* **529**(7587), 484–489 (2016)
28. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: the all convolutional net. CoRR abs/1412.6806 (2014)
29. Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C.: Man vs. computer: benchmarking machine learning algorithms for traffic sign recognition. *Neural Netw.: Off. J. Int. Neural Netw. Soc.* **32**, 323–332 (2012)
30. Szegedy, C., et al.: Intriguing properties of neural networks. In: *International Conference on Learning Representations (ICLR)* (2014)
31. Vogel, S., Schorn, C., Guntoro, A., Ascheid, G.: Efficient Stochastic Inference of Bitwise Deep Neural Networks. CoRR abs/1611.06539 (2016)
32. Zeiler, M.D., et al.: On rectified linear units for speech processing. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3517–3521 (2013)



Random Additive Control Flow Error Detection

Jens Vankeirsbilck¹(✉) , Niels Penneman² , Hans Hallez¹ ,
and Jeroen Boydens¹ 

¹ Department of Computer Science, KU Leuven,
Sporoorwegstraat 12, 8200 Brugge, Belgium

{jens.vankeirsbilck,hans.hallez,jeroen.boydens}@kuleuven.be

² Televic Healthcare NV, Leo Bekaertlaan 1, 8870 Izegem, Belgium
n.penneman@televic.com

Abstract. Today, embedded systems are being used in many (safety-critical) applications. However, due to their decreasing feature size and supply voltage, such systems are more susceptible to external disturbances such as electromagnetic interference. These external disturbances are able to introduce bit-flips inside the microcontroller's hardware. In turn, these bit-flips may also corrupt the software. A possible software corruption is a control flow error. This paper proposes a new software-implemented control flow error detection technique. The advantage of our technique, called Random Additive Control Flow Error Detection, is a high detection ratio with a low execution time overhead. Most control flow errors are detected, while having a lower execution time overhead than the considered existing techniques.

Keywords: Fault tolerance · Resilient software
Software-implemented control flow error detection · Erroneous bit-flips

1 Introduction

Embedded systems are being used more and more in our everyday life. Therefore, their reliability in ever harsher working environments is becoming more important. Today, these systems are, however, more vulnerable to external disturbances. These external disturbances range from electromagnetic interference and temperature fluctuations to high energy particles. Sierawski et al. show that particles such as neutrons and muons now have the energy to strike microprocessor components, such as memory cells, and change their state [4, 12]. Baffreau et al. present how electromagnetic interference (EMI) accumulates charges on PCB traces, on transistors, etc., and changes their state [3]. Jagannathan et al. demonstrate that the sensitivity of such components increases with temperature. They show that a higher temperature increases the charge generated by a striking particle or EMI and decreases the drive strength of the transistors used in the components [7]. Recent research shows that external attackers can

also introduce bit-flips in order to extract critical data. De Keulenaer discusses how attackers use lasers to introduce bit-flips in smart cards which lead to the leakage of critical data such as PIN codes [5]. Tang et al. show it is possible for external attackers to extract cryptographic keys and other data by influencing the energy management systems of microprocessors [13]. To conclude, external disturbances or attackers can affect a processor's internal state, introducing bit-flips into its hardware. These bit-flips can, in turn, cause invalid behaviour such as erroneously controlling an actuator, corrupting data or corrupting the execution order of instructions.

The corruption of the execution order of instructions is better known as a control flow error (CFE). A CFE is a violation against the control flow graph (CFG) of the program. The CFG is a representation of the program, in which the program is divided into basic blocks and edges. A basic block is a sequence of consecutive instructions with exactly one entry and one exit point. An edge is an intentional path between basic blocks. CFEs are typically partitioned into two categories: inter-block CFEs and intra-block CFEs. An inter-block CFE is an invalid jump through the program between two different basic blocks, while an intra-block CFE is an invalid jump within one basic block. Both types of CFE can cause the affected program or system to halt, to crash or to provide erroneous output, potentially leading to hazardous situations.

To protect embedded systems against CFEs, software-implemented fault tolerance measures have been proposed [8–10]. Inter-block CFEs are typically detected through signature monitoring, while intra-block CFEs are typically detected via instruction monitoring. In both cases, extra control variables are inserted at compile time. At run time, these control variables are calculated and compared to the expected compile-time value. A mismatch between both values indicates that an error has occurred. The main difference between inter-block and intra-block CFE detection is the frequency of updating the control variables.

In reality, CFEs can also jump into unused code space. These out-of-program jumps are detected by filling the unused code space with branches to an error handler. This paper, however, focuses on intra- and inter-block CFEs and their software-implemented counter measures.

To have full CFE detection, both signature monitoring and instruction monitoring have to be implemented. Combining two existing techniques, one of each category, leads to extra execution time overhead. This paper aims to resolve this issue by proposing a new CFE detection technique with reduced execution time overhead. Our new technique is developed to detect both inter-block and intra-block CFEs and builds on our previously developed Random Additive Signature Monitoring (RASM) technique [14]. In a previous study, we concluded that RASM outperformed other signature monitoring techniques. However, RASM aims at detecting inter-block errors, while some of the investigated techniques, such as RSCFC and SIED, are also used to detect intra-block CFEs.

The structure of this paper is as follows. First, some background is given with respect to combined signature and instruction monitoring techniques. Here, we will discuss commonly used techniques, RSCFC [8] and SIED [9], which will serve

as a reference for our new proposed algorithm. Next, a new algorithm RACFED, based on RASM, is proposed and is compared with RSCFC and SIED.

2 Background

This section discusses the principles behind the two considered techniques. For each of the techniques, we discuss their needed variables and the run-time updates they perform to detect CFEs. Figure 1 shows both techniques applied to an example CFG and indicates the added instructions in bold.

2.1 Relationship Signatures for Control Flow Checking

RSCFC relies on three compile-time variables per basic block: the compile-time signature s_i , the CFG locator L_i and the cumulative signature m_i [8]. The compile-time signature is a bit sequence that indicates the successor basic blocks of the current basic block. The CFG locator shows where the current basic block is located in the CFG. Finally, the cumulative signature is a bit sequence that indicates how much instructions must be executed for each basic block.

At the beginning of each basic block, the run-time signature S is updated and verified. In case of an error-free run, S contains the s_i of a valid predecessor. Considering basic block 0 of Fig. 1a, S contains a bit pattern in which the bit indicated by L_0 is set. The update thus updates S to L_0 in an error-free run. In case of an error, the update results in 0 and the CFE is detected.

Next, the intra-block updates are executed. First, the run-time cumulative signature N is updated with the m_i of the current basic block. Next, N is updated after each instruction. N is updated using an EXCLUSIVE OR operation with a bit pattern indicating which instruction has executed. In an error-free run, N is updated to 0 once all instructions have executed.

Finally, at the end of each basic block, S is updated using Eq. (1).

$$S = s_i \& (S \overline{\oplus} L_i) \& (!N) \quad (1)$$

Breaking it down for an error-free run:

- $!N = !0 = -1$ or an all 1 bit pattern;
- $S \overline{\oplus} L_i = L_i \overline{\oplus} L_i = -1$ or an all 1 bit pattern;
- $S = s_i \& -1 \& -1 = s_i$. Thus, in an error-free run, S is updated to s_i . In other words, S is updated to show the valid successors of the current basic block.

2.2 Software-Implemented Error Detection

SIED uses several variables to detect CFEs [9]. At compile time, each basic block gets assigned a unique identifier IDB , a list containing the compile-time

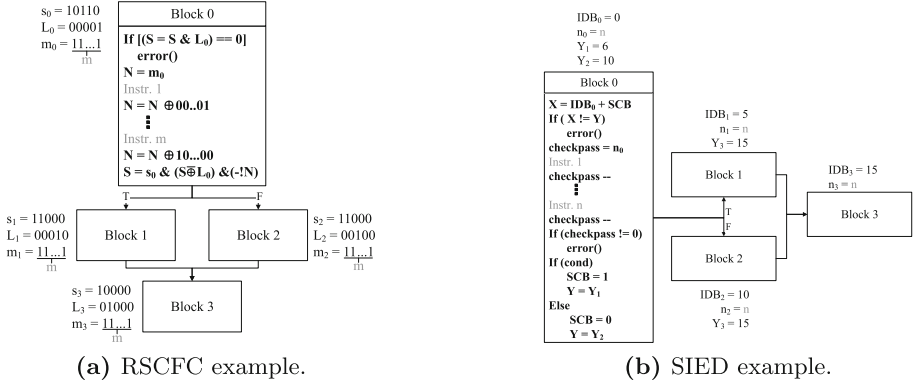


Fig. 1. RSCFC and SIED applied to an example CFG.

signatures of all successor blocks $\{Y_j, \dots, Y_k\}$ and a variable n_i that shows how many instructions must be executed in the basic block.

At run time, the signature X is calculated by performing an addition between the unique identifier of the current block, IDB_i , and the Status Condition Branch, SCB , which is updated each time a conditional instruction is executed. This addition is the first instruction the basic block executes.

Next, the run-time signature X is compared to the compile-time signature of the current block, Y_i , which is held in the run-time variable Y . A mismatch between X and Y indicates a CFE has occurred.

The intra-block detection uses a run-time variable, called *checkpass*. Once the run-time signature has been verified, the *checkpass* variable is updated with the n_i variable of the current basic block. After each instruction, *checkpass* is decremented. At the end of the basic block, a verification instruction is inserted that validates the run-time value of *checkpass* is now zero. If is not the case, a CFE has occurred. SIED thus uses two verification instructions, one for each type of CFE.

Finally, the run-time variables SCB and Y are assigned their new values after the intra-block verification.

3 RACFED

This section presents our new RACFED method. First, we describe the general CFE detection principle of the method. Next, we discuss the compile-time process to implement the technique and show an example of protected code. The section concludes with the theoretical fault detection capability of RACFED.

3.1 Principle

RACFED is an extension of RASM. As mentioned, RASM is a signature monitoring technique that uses two gradual signature updates and one signature

verification per basic block. Using gradual updates means that all updates on a specific intentional path are linked together, acting as one update. Skipping one gradual update implies that the run-time signature can never hold the correct value again.

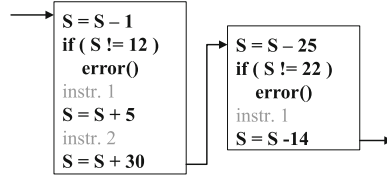


Fig. 2. Principle of the RACFED technique.

RACFED extends this functionality by inserting gradual signature updates after each instruction, as shown in Fig. 2. As can be seen, all updates now update the run-time signature S . This implementation of instruction monitoring assures a high CFE detection ratio. By updating the run-time signature as part of the intra-block CFE detection method, we can use the signature verification instruction already in place. This is the same verification method used by RSCFC and helps to lower the execution time overhead of the technique.

3.2 Compile-Time Process

The compile-time process to implement RACFED is shown in Algorithm 1. It consists of four steps. The first step is a global step that assigns the needed variables to all basic blocks. The second step assures the implementation of the instruction monitoring. The third and fourth step implement the signature monitoring part of RACFED and are executed for each basic block in the CFG.

Step 1: Assign the needed variables for all basic blocks. The process starts by assigning two random values to each basic block. The first random value is the compile-time signature and is unique for each basic block. The second value, called `subRanPrevVal`, will be used in a later step of this process to update the run-time signature. To uniquely identify each basic block, the sum of the compile-time signature and the `subRanPrevVal` has to be unique. A new `subRanPrevVal` is assigned until this is the case.

Step 2: Implement the instruction monitoring. Next, the instruction monitoring part of RACFED is implemented. The instruction monitoring consists of run-time signature updates with random values. After each original instruction, an extra run-time signature update is inserted.

Not all basic blocks have this countermeasure implemented. Only basic blocks with more than two original instructions are processed in this step. In a basic

block with only one original instruction, an intra-block CFE cannot occur. All CFEs that occur in this type of basic block are inter-block jumps, which are detected via the signature monitoring instructions of RACFED. In a basic block with two instructions, an intra-block CFE can occur, by prematurely stopping the execution of the first instruction and starting the execution of the second instruction. Such an intra-block CFE, however, cannot be detected via instruction monitoring, because no update would be skipped. Therefore, the intra-block CFE detection countermeasure is only implemented in basic blocks with more than two instructions.

Step 3: Insert the first signature update in each basic block. The third and fourth step of RACFED are the same as in the process to implement our RASM technique. The third step inserts the first update of the run-time signature and the only verification instruction per basic block. The update is a subtraction between the signature and the `subRanPrevVal` of the current basic block. The update should result in the signature having the value of the compile-time signature. If not, a CFE has occurred and is detected.

Step 4: Insert the last signature update in each basic block. The last signature update that is inserted into each basic block, is the update that assures all intentional paths in the CFG can still be taken in error-free runs, without causing a false positive CFE detection. In order to keep intentional paths valid, the signature is updated with an adjustment value. This value is calculated as the difference between the signature updates of this block and the first update of the next basic block, as shown by lines 23–27 of Algorithm 1. First, the current value of the signature is calculated by computing the total impact of the inserted intra-block updates. Next, the expected value per successor is calculated by summing up the `subRanPrevVal` and the compile-time signature of each successor. The adjustment value is the difference between the current run-time value and the expected value. The inserted update adds the adjustment value to the run-time signature. If the basic block ends with a conditional branch, this last update is executed conditionally, so the run-time signature has the correct value for the corresponding successor.

This last step changes if the basic block ends with a return instruction. A return instruction is an instruction that exits the current function and returns to its caller. Depending on the number of instructions in such a basic block, either an extra verification is inserted or no instructions are inserted. If the basic block has more than one instruction, an extra verification is added in front of the return instructions. This instruction verifies whether or not the run-time signature matches a number chosen at random, called `returnVal`. To avoid false positives, the last signature update is inserted in front of the exit verification. The adjustment value is the difference between the signature updates of the current basic block and the `returnVal`, as described by lines 15–21 of Algorithm 1. This extra verification allows to detect CFEs that would cause a premature exit out of the program.

Algorithm 1. Pseudo-code describing the compile-time process to implement RACFED.

```

1: for all Basic Block (BB) in CFG do
2:   repeat  $compileTimeSig \leftarrow random\ number$ 
3:   until  $compileTimeSig$  is unique
4:   repeat  $subRanPrevVal \leftarrow random\ number$ 
5:   until  $(compileTimeSig + subRanPrevVal)$  is unique
6: for all BB in CFG do
7:   if  $NrInstr_{BB} > 2$  then
8:     for all original instructions insert after
9:        $signature \leftarrow signature + random\ number$ 
10: for all BB in CFG insert at beginning
11:    $signature \leftarrow signature - subRanPrevVal$ 
12:   if  $signature \neq compileTimeSig$  ERROR()
13: for all BB in CFG do
14:   if Last Instr. is return instr. and  $NrInstr_{BB} > 1$  then
15:     Calculate needed variables
16:      $returnVal \leftarrow random\ number$ 
17:      $adjustValue \leftarrow (compileTimeSig_{BB} + \sum instrMonUpdates_{BB}) -$ 
18:        $returnVal$ 
19:     Insert signature update before return instr.
20:      $signature \leftarrow signature + adjustValue$ 
21:     if  $signature \neq returnVal$  ERROR()
22:   else
23:     for all Successor of BB do
24:        $adjustValue \leftarrow (compileTimeSig_{BB} + \sum instrMonUpdates_{BB}) -$ 
25:          $(compileTimeSig_{succs} + subRanPrevVal_{succs})$ 
26:       Insert signature update at BB end
27:        $signature \leftarrow signature + adjustValue$ 

```

If the basic block only has a return instruction and no other instruction, this step inserts no extra instructions. This kind of basic block only has the extra instructions inserted by step 3 of this process. Otherwise, such a basic block would have two signature verifications following each other. Not inserting this second run-time verification reduces the execution time overhead of RACFED.

3.3 RACFED Applied

An example of code protected by RACFED is shown in Fig. 3, where the run-time signature value is held in register r11. The example uses the ARMV7-M instruction set and indicates the instructions added by RACFED in bold. As with RASM, the first three instructions each basic block executes are a signature update and a verification of the run-time value against the compile-time value. If the two values do not match, control is transferred to the error handler. In the example, our error handler is located at address `0x234`, thus when a mismatch occurs, control is transferred to that location.

Next, this example shows that only a basic block with more than two instructions updates the run-time signature after each original instruction executed. As mentioned in the previous subsection, the intra-block updates are either an addition or a subtraction with a random value.

Each basic block has a signature update inserted that uses the needed adjustment value to ensure intentional paths in the CFG can still be taken. The adjustment value of 480 for the instruction at address 0x1dc is calculated as follows. First, the current value of the run-time signature is computed. At the instruction located at 0x1da, the run-time value is 130(= 129 + 1). Next, the expected value of the successor is calculated. The instruction at address 0x1dc is executed in the true case, thus the expected value of the true successor is computed. In this case, the value is 610(= 421 + 189). Finally, the adjustment value is the difference between the two previously calculated values, resulting in 480(= 610 – 130). As shown in the two-previous and upper-right basic blocks, this last update is executed conditionally if the basic block ends with a conditional branch. This allows RACFED to detect whether the correct branch is taken or not.

Finally, the two bottom basic blocks indicate how basic blocks with a return instruction are treated. The return instruction is represented by the BX lr instruction. Because these two basic blocks only contain the exit statement, no extra exit verification is inserted. This is the special case discussed in the fourth step of the implementation process.

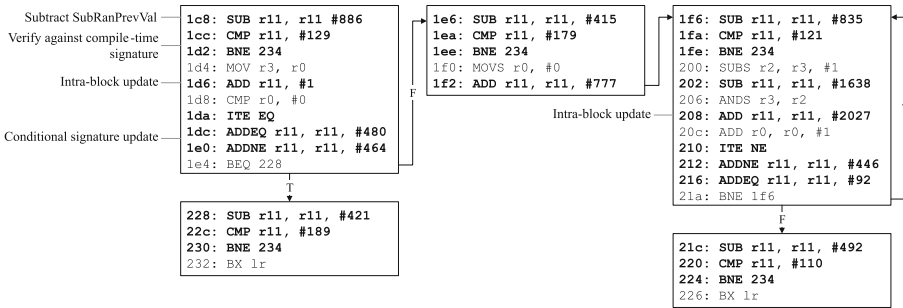


Fig. 3. RACFED implemented in an example program, with the run-time signature value held in register r11.

3.4 Theoretical CFE Detection

RACFED uses gradual updates along intentional paths. This means that all CFEs that skip at least one update are detected. Skipping an update results in an unintended run-time signature value and results in the detection of the CFE.

The exceptions are CFEs that jump to a return instruction. Although these jumps skip at least one run-time update, they do not have a successor basic block with verification instructions. Such jumps can only be detected at the place where the exit statements jumps to. An example of a return instruction is

given in Fig. 3 at the address 0x232. The `BX lr` statement instructs the processor to leave this program and branch to the address contained in the `lr` register. To detect erroneous jumps to this instruction, verification instructions must be inserted at all locations the `lr` register can point to.

Another category of undetected CFEs are intra-block jumps that skip an original instruction but no update instruction. Considering the example of Fig. 3, an erroneous jump from address 0x202 to 0x208 is not detected by RACFED, because no signature update is skipped. A possible solution to detect this kind of jump, is to insert a duplicate instruction of the original instruction. When this duplicate is inserted after the intra-block update, this assures that at least one version of the original instruction is executed [9].

The final category of CFEs that cannot be detected are shared-signature jumps. Because random values are used as intra-block updates, one run-time signature value can be valid at different points in the protected program. If a CFE jumps between two such points, it will not be detected. Since the landing point gets the expected value, the following updates will calculate the expected run-time signature value, thus masking the CFE.

4 Experimental Setup

Next to a theoretical analysis, we validated our RACFED technique through fault injection experiments. This section discusses which case studies and fault injection process were used.

4.1 Case Studies

The selected case studies are the same as used in the comparative study. They are an implementation of the following algorithms: bubble sort (BS), quick sort (QS), matrix multiplication (MM), bit count (BC), Dijkstra (DIJ), fast fourier transform (FFT) and cyclic redundancy check (CRC). The first three case studies use our own implementation, the last four were selected from MiBench version 1.0 [6].

These case studies were not only selected because they are highly used in the domain of embedded systems, but also because they have different CFGs. These differences assure a thorough and broad evaluation of RACFED.

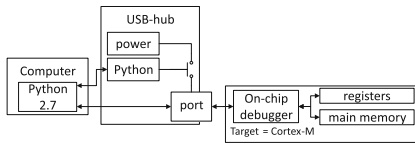
4.2 Hardware

The case studies were executed on an ARM Cortex-M3-driven microcontroller running at 96 MHz, including 512 kB FLASH and 32 kB RAM. We selected the ARM architecture because ARM is the global leader with an 85% market share for mobile devices and a 25% market share for embedded intelligence applications such as Internet of Things [2]. The ARM Cortex-M3 was selected because it is an industry-leading 32-bit processor used in many different embedded application domains [1].

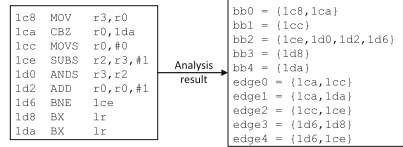
4.3 Fault Injection

To make a valid comparison between RACFED, RSCFC and SIED, and to inject CFEs deterministically, we use our own software-implemented fault injection (SWIFI) tool. SWIFI is a well-known method to validate software-implemented fault tolerance measures. This methodology was also used to validate the considered techniques in their respective publications.

Our SWIFI tool injects both inter-block and intra-block CFEs in a deterministic way at run time. As shown in Fig. 4a, this is possible by using the on-chip debugger of the target. The Python-controllable USB-hub allows to issue a hard reset when the on-chip debugger fails.



(a) SWIFI tool architecture.



(b) Example program with its corresponding CFG.

Fig. 4. Fault injection setup

The used fault injection process is our own control-flow-aware injection process [15]. It uses the CFG of the program to inject the desired intra- or inter-block CFE. The following is an example of an inter-block CFE injection for the program shown in Fig. 4b.

1. The program counter (PC) is read out: its value is $0x1d2$;
2. Create a list of all possible single-bit bit-flip possibilities: $\{0x1d3, 0x1d0, \dots, 0x9d2\}$. We use the single-bit bit-flip fault model because research shows it is an accurate fault model and only few programs benefit from multi-bit bit-flips [11];
3. Discard all non-existing values: $\{0x1d0, 0x1d6, 0x1da\}$;
4. Discard all intra-block values: $\{0x1da\}$;
5. Write the new value to the PC: it now has the value $0x1da$.

While injecting faults in other registers or memory words could also result in a CFE, we chose to only inject in the PC because that is the easiest way to assure a CFE would occur. This research and its fault injection experiment serve to find out whether CFEs are detected. We are less concerned with the origin of the CFE. We want to know, if a CFE occurs due to whatever reason, whether the implemented technique detects it or not.

4.4 Criteria

To validate our RACFED technique, we determined the effect of the faults and grouped them into categories and we measured the execution time overhead.

Result Categories. We subjected each combination of case study and technique to 4000 CFEs, divided as 2000 inter-block and 2000 intra-block CFEs. Each cycle of 2000 CFEs was injected in 10 batches of 200 single-bit bit-flips to be able to average out the results of the 10 batches. Per batch of faults, we determined the effect of each and every fault. We grouped the effects in four categories:

- **Detected (Det.):** This percentage of faults was detected by the implemented countermeasure. In other words, this category is the desired result.
- **Hardware Detection (HD):** The Cortex-M3 already has several internal fault handlers that are able to detect specific hardware faults, such as improper bus usage or memory access violations. This category represents the faults detected by such fault handler.
- **Silent Data Corruption (SDC):** These are the faults that were not detected by the implemented technique and caused the algorithm to produce a wrong result. These faults can be faults as described in Sect. 3.4, but could also be faults that we anticipated to detect. In this case, these faults expose a flaw in the implemented detection technique.
- **No Effect (NE):** Finally, this is the percentage of the faults that were not detected and did not affect the outcome of the target algorithm.

Execution Time Overhead. Erroneous bit-flip detection does not come cheap, as each technique inserts extra instructions and thus an execution time overhead. Execution time overhead expresses the extra time it takes for the algorithm to execute. The introduced overhead, for an error-free run, is calculated using Eq. (2). In this paper, the shown overhead is the extra amount of execution time in an error-free run introduced by the technique.

$$overhead = \frac{exec.time\ protected - exec.time\ unprotected}{exec.time\ unprotected} \quad (2)$$

5 Results

First, we discuss the fault injection results of our RACFED technique. Secondly, we present the execution time overhead, the unavoidable cost of any error detection countermeasure. The results presented in this paper are averaged over the 10 executed batches per technique.

5.1 Fault Injection Results

Figure 5 shows the fault injection results of RACFED compared to the results of RSCFC and SIED. To be complete, we also compare the results against the results obtained with our RASM technique implemented. Each bar represents a case study and can be broken down in two pieces, a green piece and a red piece. The green piece indicates the amount of detected CFEs while the red

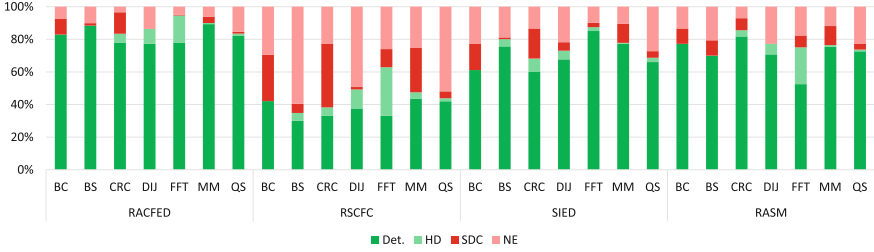


Fig. 5. Results of the fault injection campaign. (Color figure online)

piece indicates the number of undetected CFEs. The following is a more detailed discussion of the results.

When only considering the faults detected by the three implemented techniques, thus the Det. category, it can be seen that RACFED has the highest ratio for six of the seven case studies. Only for the FFT case study SIED has a higher Det. ratio. However, when considering all detected faults, the sum of the Det. and HD categories, RACFED has the highest ratio for all seven case studies.

Regarding the undetected faults, it’s important to know what percentage are malicious, in other words the SDC category (indicated in dark red) must be taken into account. For this category, RACFED also has the lowest ratio for six of the seven case studies. Only for the BS case study applying SIED results in a smaller SDC ratio. The difference with RACFED is, however, only 0.4%. These results show that RACFED better protects embedded systems against CFEs.

When comparing the results of RACFED with RASM, Fig.5 shows that RACFED detects more CFEs. On average, RACFED has an increase of 11% for the Det. category compared to RASM. Secondly, the results show that RASM has a higher SDC ratio than RACFED. On average, RACFED has a decreased SDC ratio of 3%. These results show that the intra-block updates are indeed necessary to decrease the number of SDC errors. RACFED is thus an improvement of RASM when considering fault detection.

The remaining CFEs, indicated in light red, represent the NE category. These are undetected CFEs that did not affect the outcome of the target program. These show that not all CFEs necessarily affect the targeted program. In other words, this category shows that every algorithm is resilient against certain CFEs, even without having any protection applied. This is better known as the inherent masking capability of the targeted program.

5.2 Execution Time Overhead

Since extra instructions have to be executed, an execution time overhead is introduced. Figure6 shows the imposed execution time overhead of RACFED compared to the one imposed by RSCFC, SIED and RASM.

On average, RACFED imposes an execution time overhead of 108.4%. This is 60% lower than the average execution time overhead of RSCFC and SIED.

RACFED has a lower overhead due to two factors. A first factor is the usage of the signature for the intra-block updates instead of an additional control variable. This eliminates the need to insert an instruction to set up a second variable and eliminates the need to insert extra verification instructions. Using the signature as control variable for the intra-block detection allows to use the signature verification instruction to validate the inter-block and intra-block updates. The second factor is that the intra-block updates are only inserted in basic blocks with more than two instructions.

Because RSCFC inserts intra-block updates in all basic blocks, it has a higher average execution time overhead of 166.0%. SIED uses a second control variable to detect intra-block CFEs and thus inserts more instructions. Those extra instructions result in an average execution time overhead of 184.7%.

RASM does impose a lower overhead than RACFED. RASM is designed to detect inter-block CFEs only and has no intra-block updates, which results in a lower overhead. On average, RACFED has an increase of 30% in execution time overhead due to its intra-block updates. The increase in fault detection capabilities does come at a price of extra execution time overhead.

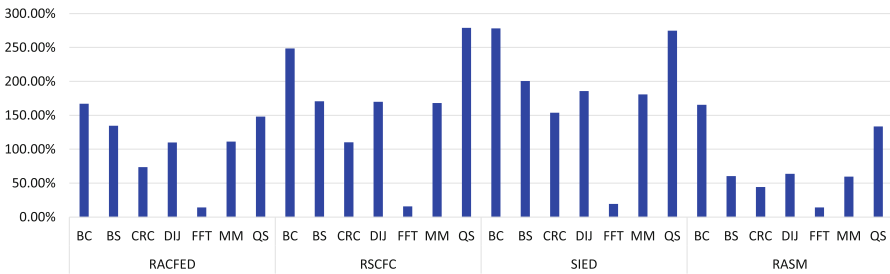


Fig. 6. Execution time overhead of the three considered techniques.

6 Future Work

Although RACFED has an average Det. ratio of 82.35% there is room for improvement. As mentioned in Sect. 3, one type of undetectable CFE is the shared-signature jump. This is an erroneous jump between two program points that expect the same run-time value of the signature to produce a correct signature update. Such jumps exist within RACFED because random values are used as intra-block signature updates. These undetectable CFEs can be eliminated by improving the implementation algorithm provided in Algorithm 1. An extra compiler check could be inserted each time an intra-block signature update has been inserted that verifies whether the updated run-time signature is unique throughout the entire program. When this is not the case, a new random value could be chosen until the verification holds.

Due to the one-register implementation, the allowed random values and basic block identifiers are limited. Most instruction set architectures allow limited

values to be added immediately to or be compared with a register. These limited values could mean that for certain algorithms RACFED is not implementable. To solve this issue, a two-register implementation should be developed. Using two registers allows to reserve one to hold the run-time signature value and another one to hold larger random values that can be added to the run-time signature. The downside of needing two registers is that the target program will need more memory accesses since it has two less registers. The increase in memory accesses could increase the execution time overhead.

While analyzing the data generated during the fault injection campaign, we noticed that most of the CFEs were injected in the inner loops of the case studies and only a small percentage of the CFEs were injected in the remainder of the target code. To improve the code coverage of the SWIFI tool, we are currently developing a CFE injection process that would allow to step through the target program and inject all possible CFEs for each program point. Another improvement of the fault injection experiments would be to use different input vectors and ease the use of different input vectors. Currently, each fault injection experiment has to be set up with a fixed input vector, meaning that the entire experiment needs to be reset and re-executed for a different input vector. This could be improved to make it easier to use different input vectors while only having to set up the experiment once.

7 Conclusions

This paper presented our new CFE countermeasure RACFED. Our technique is developed to detect almost all inter-block and intra-block CFEs, while having a lower overhead compared to RSCFC and SIED. To detect inter-block CFEs, an implementation of signature monitoring is used. Per basic block, two signature updates and one signature verification is added. The intra-block CFE detection is realized via an implementation of instruction monitoring for selected basic blocks. After each original instruction, a signature update is added. This intertwined inter-block and intra-block CFE detection results in an average detection ratio of 82.35 %, with an execution time overhead of 108.4%.

We compared RACFED to two established techniques: RSCFC and SIED. Considering RSCFC, our technique has an increase in detection ratio of 45% with a decrease in execution time overhead of 58%. Compared to SIED, RACFED has an increase of 12% in detection ratio and a decrease in execution time overhead of 76%. This shows that our RACFED technique is the better of the three, proving that a higher detection ratio with a lower execution time overhead compared to RSCFC and SIED is possible.

Acknowledgement. This work is supported by a research grant from the Baeke-land program of the Flemish Agency for Innovation and Entrepreneurship (VLAIO) in cooperation with Televic Healthcare NV, under grant agreement IWT 150696.

References

1. Cortex-M3, December 2017. <https://developer.arm.com/products/processors/cortex-m/cortex-m3>
2. Integrating IoT and advanced technology design, application development and processing environments (2017). <https://m.eet.com/media/1246048/2017-embedded-market-study.pdf>
3. Baffreau, S., Bendhia, S., Ramdani, M., Sicard, E.: Characterisation of microcontroller susceptibility to radio frequency interference. In: Proceedings of the Fourth IEEE International Caracas Conference on Devices, Circuits and Systems, pp. 1031.1–1031.5. IEEE (2002)
4. Baumann, R.: Soft errors in advanced computer systems. *IEEE Des. Test Comput.* **22**(3), 258–266 (2005)
5. De Keulenaer, R.: Softwarebeveiliging van smartcards tegen laseraanvallen. Master's thesis, Universiteit Gent (2013)
6. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: MiBench: a free, commercially representative embedded benchmark suite. In: 2001 IEEE International Workshop on Workload Characterization, WWC-4, pp. 3–14. IEEE (2001)
7. Jagannathan, S., et al.: Temperature dependence of soft error rate in flip-flop designs. In: 2012 IEEE International Reliability Physics Symposium (IRPS), pp. SE.2.1–SE.2.6. IEEE (2012)
8. Li, A., Hong, B.: Software implemented transient fault detection in space computer. *Aerosp. Sci. Technol.* **11**(2), 245–252 (2007)
9. Nicolescu, B., Savaria, Y., Velazco, R.: SIED: software implemented error detection. In: 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Proceedings, pp. 589–596. IEEE (2003)
10. Oh, N., Shirvani, P.P., McCluskey, E.J.: Control-flow checking by software signatures. *IEEE Trans. Reliab.* **51**(1), 111–122 (2002)
11. Sangchoolie, B., Pattabiraman, K., Karlsson, J.: One bit is (not) enough: an empirical study of the impact of single and multiple bit-flip errors. In: Proceedings of 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 97–108, June 2017
12. Sierawski, B.D., et al.: Effects of scaling on muon-induced soft errors. In: 2011 IEEE International Reliability Physics Symposium (IRPS), pp. 3C.3.1–3C.3.6. IEEE (2011)
13. Tang, A., Sethumadhavan, S., Stolfo, S.: CLKSCREW: exposing the perils of security-oblivious energy management. In: 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, pp. 1057–1074. USENIX Association (2017). ISBN 978-1-931971-40-9
14. Vankeirsbilck, J., Penneman, N., Hallez, H., Boydens, J.: Random additive signature monitoring for control flow error detection. *IEEE Trans. Reliab.* **66**(4), 1178–1192 (2017)
15. Vankeirsbilck, J., Thati, V.B., Waes, J.V., Hallez, H., Boydens, J.: Control flow aware software-implemented fault injection for embedded CPUs. In: Proceedings of XXVI International Scientific Conference on Electronics (ET), pp. 1–4, September 2017



Fault-Tolerant Clock Synchronization with Only Two Redundant Paths

Zoha Moztarzadeh^(✉)

ICB – Institute for Computer Science and Business Information Systems,
University of Essen-Duisburg, Essen, Germany
moztarzadeh@dc.uni-due.de

Abstract. Many safety-relevant real-time systems require a reliable time source, which leads to the requirement of fault-tolerant clock synchronization. This paper proposes a fault-tolerant synchronization protocol for networks where the bridges are connected via point-to-point links (like Ethernet or Time-Sensitive Network) and the number of redundant point-to-point links is kept small for cost reasons, like in ring topologies. This new protocol “single initiator forward and collected answer” (SFC), can tolerate all failures of one faulty bridge though it needs only two disjoint paths between any pair of bridges.

Keywords: Distributed real-time systems
Fault-tolerant clock synchronization · Ring topology · Byzantine failure

1 Introduction

Many safety-critical real-time applications require accurate and precise clock synchronization to keep the local clocks in a distributed system synchronized. Some of the methods are based on a master-slave hierarchy, where a master broadcasts its current time at a predefined period, used in protocols like Network Time Protocol (NTP) [1] and Precision Time Protocol (PTP) [3] which is described in the standard IEEE-1588 [4]. PTP is appropriate for Ethernet-networks, where NTP is mostly used to synchronize the clocks of hosts connected via internet. PTP uses master-slave hierarchy and its accuracy and precision can get worse if a link or node fails. Providing a secondary master clock as a spare does not solve the fault-tolerance problem. If the primary master fails by distributing wrong time values rather than failing silently, changing to the secondary master clock cannot ensure a consistent time in the presence of arbitrary failure types [5, 6]. Therefore fault-tolerant behavior becomes a key requirement. The relative synchronization of clocks must be achieved by a fault-tolerant algorithm. Internal clock synchronization algorithms guarantee a bounded maximum deviation among the clocks, which may differ from the external global time.

The fault-tolerant mid-point algorithm (FTMA) [7] is one example of a distributed convergence algorithm, which tolerates up to f faulty clocks with Byzantine behavior by at least $n = 3f + 1$ nodes with $2f + 1$ disjoint paths between any pair of nodes. In FTMA each node maintains an offset vector with the differences of its clock to all other clocks. Then the f lowest and the f highest offset values are discarded and the arithmetic mean of the minimum and the maximum of the remaining offset values are

calculated as correction term of the local clock. The provision of three disjoint paths between any pair of nodes for $f = 1$ leads to a costly topological demand requiring a considerable number of links.

Some researchers propose an improvement of the robustness by integration of IEEE-1588 with network redundancy protocols like RSTP [8] or high-availability seamless redundancy (HSR) [9]. RSTP is not suitable for real-time applications. On occurrence of a link failure or a node failure, the network must be reconfigured to rebuild the logical path (this recovery time can be in the range of a millisecond to a second depending on the network topology). HSR is a redundancy protocol used in ring topologies, which utilizes the idea of Parallel Redundancy Protocol (PRP). HSR and PRP both need zero-recovery time in the faulty case. In the absence of faults HSR causes half of the worst-case delay over the ring compared to RSTP, whereas the worst-case delays are equal in the presence of faults. HSR does not require the duplication of any node or link but provides duplicated frames to transmit them via separate paths. HSR tolerates a single message loss due to a link failure by sending data via both directions in the ring. Arbitrary malfunctions of a faulty node are not tolerated [10]. Other approaches use a more redundant topology like a braided ring with additional guardian functionality [11] to tolerate a Byzantine failure. However, when the cost imports, the redundancy of the links should be kept low. The absolute minimum is $n - 1$ links to connect n nodes by a tree structure. A ring requires just one additional link (a total of only n links for n nodes) and guarantees two disjoint paths between any pair of nodes. Unfortunately, two disjoint paths are not sufficient for the tolerance of Byzantine behavior according to the theory in [7]. Since three disjoint paths increase the cost by far, we aim at a protocol that is based on a ring and tolerates as many malfunctions as possible. As we will see later, our solution as “single initiator forward and collected answer” (SFC) protocol presented in this paper is resilient to one faulty bridge which may exhibit any type of failure such as fail-omission, delay and corruption, – with only a slight restriction: A very special type of Byzantine behavior may somewhat reduce the precision.

The remainder of this paper is structured as follows: After this introduction the principles of the protocol (Sect. 2) are described. Then the SFC protocol development is presented in Sect. 3, followed by its analysis (Sect. 4) and a summary.

1.1 System Model

In this paper we consider a network consisting of a ring of bridges, where each bridge is connected to a node. Every bridge owns a clock that is either located within the bridge or in an adjacent node.

1.2 Timing

The clocks are synchronized periodically, where each synchronization interval (from one synchronization to the next) has equal duration T_{nextsync} . Within a synchronization interval the time is subdivided as shown in Fig. 1. During T_{protocol} synchronization messages are exchanged (the sub-protocols FP and SP and their durations T_{FP} and T_{SP} are explained later). When too frequent synchronization should be avoided to limit the

overhead, T_{protocol} can be preceded by some period T_{sep} to achieve a greater temporal separation of the synchronizations. After execution of the synchronization protocol the bridges execute FTMA on their local data and adjust their clocks according to the calculated correction term. Since clock adjustment must neither overlap with the message exchange nor with the subsequent synchronization interval, small pauses T_{waitA} and T_{waitP} have been inserted. Their minimum duration is β , which is the maximum deviation among fault-free clocks. After adjustment, fault-free clocks differ only by $\alpha < \beta$, which means the synchronization becomes effective. Then they drift apart again due their oscillator’s drifts until they reach their maximum distance of at most β , which will be reduced to α by the next clock adjustment. The relation between α and β expresses the quality of the synchronization, also called precision which refers to the relative difference between fault-free clocks.

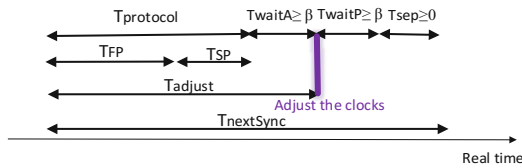


Fig. 1. Timing of synchronization interval

1.3 Forwarding Delay Compensation

Nearly all synchronization protocols apply delay compensation. Whenever a bridge B_i receives a message containing time t and forwards it after some delay, it measures the duration d_i of the message’s stay in the bridge, and adds d_i to the content of the message. A later receiver will then interpret time t as $t + d_i$ to compensate the delay. Due to unavoidable deficiencies in delay measurement we have to consider an inaccuracy with an assumed upper bound τ . Formally: For a true delay θ_i the indicated delay is d_i with a small $\Delta_i = |d_i - \theta_i| < \tau$.

Detection of Wrong Compensation. Appending the forwarding delay information to the message also leads to detection of wrong behavior of a faulty bridge that indicates its delay wrongly. A bridge is able to check if the absolute difference between the time point of sending a message (t_{send}) and the time point of receiving the message back (t_{receive}) is approximately equal to the sum of the indicated delays in the message. Only a small unavoidable difference of τ per hop must be accepted. Naturally the acceptable difference linearly increases with the number of hops the message has passed until it is received back (see Eqs. (1) and (2) in Sect. 2.2).

2 Principle of the Protocol

The developed protocol is composed of two sub-protocols “Forward Protocol” (FP) (see Fig. 2(a)) and “Secondary Forward Protocol” (SP) (see Fig. 2(b)), though the SP is executed only if a failure is detected. Both sub-protocols are launched by the

initiator, the bridge that starts the synchronization. The initiator sends its time information in both directions along the ring and receives it back later. Then it can check if its time information has been correctly forwarded by all bridges. Moreover, the other bridges piggyback their time information to the initiator’s message. If an error is detected SP is launched. Finally, the time information has been exchanged correctly between all fault-free bridges under the assumption of a single faulty bridge.

2.1 Sub-protocols

Forward Protocol (FP). By starting the FP sub-protocol, the initiator sends its time information in a so-called “*time-message*” in both clockwise and counterclockwise along the ring, which is forwarded by other bridges in the network (called forwarders). The forwarders that get the time-message take the received time information to calculate the offset to their local clock, and store it in their offset-vector. Then they add their forwarding delay information to the time-message and forward it in the same direction. Some forwarders (not necessarily all) add also their time information to the message. These forwarders are called sources. The bridge opposite to the initiator is called *merger*. After it has received the time messages from both sides it unites them and sends it as an “*answer-message*” both clockwise and counterclockwise back to the initiator. Again, the sources among the forwarders add their time information.

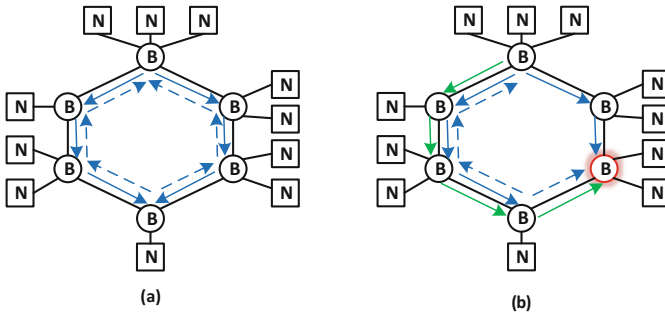


Fig. 2. (a) Forward Protocol (FP), (b) Secondary Forward Protocol (SP), blue solid arrows are time-messages, blue dashed-arrows are answer-messages, green solid arrows are replacement-message, B denotes bridges (Bridge in red circle is faulty), N denotes nodes. (Color figure online)

Secondary Forward Protocol (SP). If one bridge has added wrong delay information to the time-message or corrupted the time-message or applied some other malfunction, the initiator can tolerate these failures by sending the time information again as a so-called *replacement-message*. Details on fault detection will be discussed in Sect. 3.2. The initiator determines the direction and the bridge up to which the replacement message is forwarded. The selection of this so-called *destination-bridge* depends on the fault location the initiator has identified, see Fig. 2b.

The two sub-protocols guarantee that the bridges' time information are received correctly at least via one path. If a bridge obtains some time information twice (both via FP and SP) the time information of SP is preferred. With the time information of all bridges each bridge can build its offset vector as the basis its clock adjustment.

2.2 Self-compensating Error

Whenever an answer-message is received by a bridge it is checked whether the message is syntactically correct and also if the sum of the indicated forwarding delays in the message fits to the difference between the point in time when a bridge forwards a time-message (t_{send}) and the point in time when it receives the time information back in the answer-message (t_{receive}), see Eqs. (1) and (2).

$$|t_{\text{send}} - t_{\text{receive}}| < \delta_i^+ = (d_i + \dots + d_{h_{FP}}) + (d_{h_{FP}} + \dots + d_i) + 2(h_{FP})\tau(1 + 2\rho) \quad (1)$$

$$|t_{\text{send}} - t_{\text{receive}}| > \delta_i^- = (d_i + \dots + d_{h_{FP}}) + (d_{h_{FP}} + \dots + d_i) - 2(h_{FP})\tau(1 + 2\rho) \quad (2)$$

The values of δ_i^+ and δ_i^- are calculated according to Eqs. (1) and (2) such that not more than the maximum inaccuracy of the local forwarding delay measurements τ are accepted. Thus the bounds δ_i^- and δ_i^+ are dependent on the indicated forwarding delays d_i of the bridges in the time- and answer-message, the number of hops between the respective bridges, multiplied with the maximum inaccuracy τ of the local forwarding delay measurements (factor 2 because the inaccuracy can be either positive or negative) and the drift ρ (which can also be either positive or negative).

If any faulty bridge adds wrong forwarding delay information, the failure can be detected by checking (1) and (2). In case of violation the initiator initiates SP to transmit the time information via a replacement-message up to the destination-bridge. However, if the faulty bridge has added a wrong forwarding delay to the time-message in FP and compensates this error in the answer-message by an opposite wrong forwarding delay, this failure cannot be detected. Such undesired self-compensation can occur when a faulty bridge B_i delays the time-message by θ_i but indicates $d_i = \theta_i + u$, where u is the error, and later delays the answer message by θ'_i but indicates $d'_i \approx \theta'_i - u$. The two errors of $+u$ and $-u$ prevent fault detection. Therefore we define the *NSC-assumption (no self-compensation)* that a faulty bridge does not compensate its own error by a complementary error.

Fortunately, the self-compensating error is not unlimited, because the indicated delay information in the message cannot be negative, of course. Consequently the worst-case self-compensated error occurs, if the true delay θ_i is the maximum $\theta_i = T_{\text{forw}}$ and the indicated delay d_i is 0, or vice versa. In any case the absolute difference between the two values is the maximum forwarding delay T_{forw} , which limits the effect of an undetectable self-compensating error: $|u| < T_{\text{forw}}$.

When a time-message is forwarded along h_{FP} hops, where one of the bridges is faulty and commits a self-compensating error u , then this error is detectable if:

$$|u| > (2h_{FP}\tau + T_{forw})(1 + 2\rho) \quad (3)$$

It should be noticed that the proposed protocol SFC works correctly both with and without the NSC-assumption, but without the NSC-assumption the worst-case inaccuracy between the clocks is higher (limited by T_{forw}). Thus, the faster the network the smaller the effect of undetected self-compensating errors.

2.3 Protection by Signatures

SFC uses signatures for message authentication. This allows each bridge to conclude the path of information flow in time-, answer- and replacement-messages. Protection of messages by signatures allows detecting corruption of message content, format, etc. (to the extent of the coverage of the signature test). The signatures used for clock synchronization should be different from other signatures to prevent confusion. The signatures must withstand technical faults, not human attacks. Hence, they can be rather short to limit the computational overhead. 16, 32 or 64 bits are sufficient in most cases [12, 13].

3 Protocol Development

The first solution to efficient fault-tolerant clock synchronization in a ring was the “*Ring forward and Answer Protocol (RFA)*” [14] in which all bridges act as initiators by executing their protocol individually. The time-message is sent completely along the ring up to the last bridge of the network, and each bridge that gets the time-message sends an answer-message back to the initiator. This leads to a higher over-head and precision (because the messages are sent via more hops), but based on the contents of answer-messages, the faulty bridge can be localized. An optimization of the RFA protocol led to a reduced overhead and increased precision by sending the time-message not completely along the ring, but half of the ring in both directions. Moreover, only the merger bridge at the half of the ring issues an answer message, which finally obtains the collected time information from all bridges. Collecting information in a message instead of utilizing separate messages leads to a lower number of exchanged messages. The increased message length is not significant as each node only adds a few bytes.

3.1 Algorithm of SFC

In SFC there is only a single initiator (initiator is a source). The merger bridge opposite to the initiator (the $\lfloor \frac{n}{2} \rfloor$ th bridge) unites the information from both half-rings. The initiator sends its time-message addressed to the merger both clockwise and counter-clockwise. All other bridges are forwarders some of which are also sources (we need s sources according to $s \geq 3f + 1$). They add their forwarding delay d_i (and their time information t_i in the case of sources) whenever they forward a message (Fig. 3).

The way how they extend the message content when forwarding it as pure forwarder (not being a source) is expressed by the extension function $f_i(\text{Msg})$, see (4),

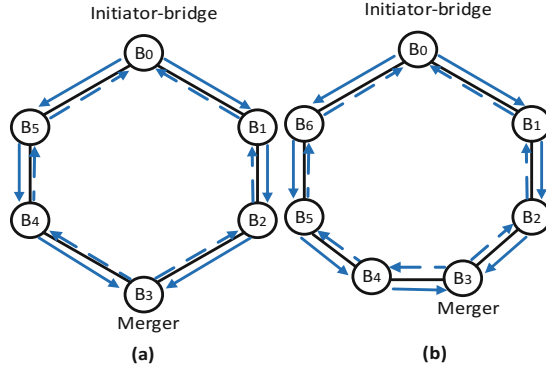


Fig. 3. Protocol execution in the fault-free case (a) for the network with even and (b) with odd number of bridges. In both cases B_0 is the initiator and B_3 is the merger. Blue solid arrows depict the time-messages (sent by the initiator) and blue dashed arrows depict answer messages (sent by the merger). (Color figure online)

whereas sources use the extension function $g_i(\text{Msg})$, see (5). t_i denotes the current clock-time, d_i the delay. The digital signature is expressed by “: B_i ”.

$$f_i(\text{Msg}) := (\text{Msg}, d_i) : B_i \quad (4)$$

$$g_i(\text{Msg}) := (\text{Msg}, t_i, d_i) : B_i \quad (5)$$

As soon as both messages have been received by the merger, it unites the time-messages received from both half-rings, and extends the united content by the extension function $g_i(\text{Msg})$. This new, extended message is sent as an answer-message back to the initiator both clockwise and counterclockwise. On the way back, the messages are again extended by $f_i(\text{Msg})$ or $g_i(\text{Msg})$ in each bridge it passes.

Now we describe the temporal protocol behavior in detail with respect to the timeouts utilized by bridges in different roles. The durations of the timeouts will be calculated later in Sect. 4. The intention behind the timeouts is the triggering of bridges’ actions in case of fail-silence or omission of a faulty bridge. Even then the flow of message forwarding along the ring must be continued (although the information gathered so far is lost).

The initiator uses the timeouts: T_{nextSync} , T_{FP} and T_{adjust}
 The forwarders use the timeouts: T_{timeMsg} , $T_{\text{answerMsg}}$ and T_{adjust}
 The merger uses the timeouts: T_{timeMsg} and T_{adjust}

After entering the synchronization interval (duration T_{nextSync}) the initiator waits the durations T_{waitP} and T_{Sep} . These durations specify the duration after clock adjustment up to the next execution of the FP sub-protocol. Then the initiator starts FP by sending the time-message containing its local time and signature clockwise and counterclockwise. The time-message should be received by each bridge before its respective timeout T_{timeMsg} has expired (to detect any missing time-message). The bridges take the time

information they need to calculate the offset value required by FTMA later. They also extend the time-message either by $f_i(\text{Msg})$, (if pure forwarder, see Eq. (4) or $g_i(\text{Msg})$ (if source, see Eq. (5)). If T_{timeMsg} of any bridge expires and the time-message is missing then the respective bridge creates a new time-message with its own time information and sends it to the merger.

The merger waits for the time-messages coming from both directions via the ring. These messages should have been received before its timeout T_{timeMsg} . If T_{timeMsg} expires and the merger has not yet received the time-message either from one or both directions, then it creates an answer-message containing its local time as well as all time information the merger has received so far. The answer message is then sent to the initiator in both directions of the ring. The bridges should receive the answer-message from the merger before their timeout $T_{\text{answerMsg}}$ to detect a missing answer-message. If $T_{\text{answerMsg}}$ expires in any bridge before the answer-message has been received, then the respective bridge creates a new answer-message containing its local delay information (and local time if it is a source) and sends it to the initiator. Finally, the initiator should have received an answer-message from the merger from both half-rings before timeout T_{FP} .

Besides timeouts the generation of error indications also triggers some protocol actions. The conditions that lead to an error-flag added to a message forwarded along the ring are as follows.

Each bridge checks an incoming message with respect to syntax, plausibility (non-negative delays, for example) and signatures. If this check fails, the message is discarded. Furthermore the conformance of the sum of indicated delays with the bounds δ_i^+ and δ_i^- according to Eqs. (1) and (2) is also checked. If this check fails, it can be concluded that one bridge on the message path is faulty although it cannot always be identified exactly. The bridge which detects a violation, flags an error in the answer-message. The flagging bridge is called “*reporter*”. In the error-flag the reporter indicates its neighbor bridge from which the answer-message has been received as the suspicious faulty bridge (called “*reported bridge*”).

If any failure is reported, the initiator starts SP by creating a replacement-message and sending it to the destination-bridge, which is determined according to the following localization function (explained for a faulty bridge located in the right half-ring, the explanation for a faulty bridge in the left half-ring is symmetric):

- The case where the initiator misses the time information from bridges B_{i+1} to B_j in the answer-message received from the right half-ring can be interpreted by the following alternatives the initiator cannot distinguish:
 - Bridge B_i is faulty and has not forwarded the answer-message correctly. Note that the hierarchical signature scheme (every bridge signs the complete message it forwards) makes it impossible for a faulty bridge to delete only a subset of the time information in the message content.
 - Bridge B_i is faulty, has forwarded the time-message correctly but omitted to forward the received answer-message and has sent its newly created answer-message containing its time information t_i .
 - Bridge B_{i+1} is faulty and has not forwarded the time-message and answer-message correctly (the bridges fails in both directions).

Then both B_i and B_{i+1} are suspicious and chosen as destination bridges, to which the replacement is sent, clockwise to B_i and counterclockwise to B_{i+1} . This is sufficient in either case: If B_i is faulty in fact, then B_{i+1} will be informed by the replacement-message counterclockwise during SP. If B_{i+1} is faulty, then B_i has already obtained some time information during FP and will get the remaining time information during SP clockwise.

- The case where bridges have reported an error has to be dealt with as follows: The initiator must identify which bridge has flagged the error first (the first reporter that signed an error in the signature sequence, where the signature sequence is considered separately in the left and the right half-ring). Let B_i be the first reporter of wrong delay compensation by bridge B_{i+1} (reported-bridge). This can be interpreted by the following alternatives the initiator cannot distinguish:
 - Bridge B_i is faulty and has erroneously acted as reporter.
 - Bridge B_{i+1} is faulty and has wrongly compensated its forwarding delay.

Again, there are two suspicious bridges, B_i and B_{i+1} , and both are chosen as destination bridges, to which the replacement message is sent clockwise to the reporter (B_i), and counterclockwise to the reported-bridge (B_{i+1}). This is sufficient in either case for the same reason mentioned before.

A special case arises if both neighbor-bridges of the merger flag an error (both are reporters), or if the merger does not send an answer-message at all. Then the merger can be localized as the faulty bridge. In this case the replacement-message is sent clockwise to the right neighbor, and counterclockwise to the left neighbor of the merger. Finally, when T_{adjust} expires at the end of SP, the bridges, the merger and the initiator execute FTMA on their local offset vector and adjust their clocks.

3.2 Fault Tolerance

Fail-Omission and Corruption of Content. If any bridge does not forward the time-message, its neighbor in direction towards the merger detects the failure after its timeout T_{timeMsg} has expired. It then creates a time-message, which is forwarded to the merger. Once the answer-messages of the merger have been received by the initiator, it will notice that they don't contain the time information of all bridges. Thus it starts SP to send the replacement-message containing all available time information up to the destination-bridge.

If any faulty bridge corrupts the message (format, signature, etc.) or writes a wrong forwarding delay in a time- or answer-message (more than the maximum forwarding delay $d_i > T_{\text{forw}}$) the next bridge in the same direction detects this failure and the message will be discarded. The tolerance is then similar to the tolerance of the fail-omission because the corrupted message is rejected and not forwarded further.

Figure 4(a) shows an example where B_5 has failed and doesn't forward the time-message in counterclockwise direction to B_4 . After the timeout T_{timeMsg} in B_4 has expired, it creates a time-message with B_4 's clock time and sends it to the merger B_3 . After both time-messages from the right and the left half-ring have been received by B_3 , it unites them and sends the united message in both directions as the answer-message.

Now the faulty bridge B_5 may fail again and skip the forwarding of the answer-message. The initiator notices that the answer-message from the left half-ring and also B_5 's time information in the answer-message received from the right half-ring are missing. From its view there are different interpretations of this case: B_5 could be faulty as described before, or B_4 is faulty and hasn't forwarded the time-message. The initiator sends the replacement-message containing all available time information (The time information of B_5 is not included, but this will be tolerated by FTMA) clockwise to B_4 and counterclockwise to B_5 . All non-faulty bridges take the time information from the replacement-message (bridges in green circles).

Figure 4(b) shows an example where the merger B_3 is faulty and doesn't send the answer-message to the right half-ring. After timeout $T_{\text{answerMsg}}$ of B_2 has expired, the merger creates an answer-message (containing its time information) and sends it to the initiator. The merger (B_3) has sent its answer-message to the left half-ring before. Thus the initiator receives an answer-message from both half-rings, but in the answer-message from the right half-ring the time information of the merger (B_3) is missing. From the view of the initiator, bridges B_2 and B_3 could be faulty:

- B_2 may have not forwarded the answer-message of B_3 , but have sent its own answer-message.
- B_3 may have missed to send the answer-message in the left half-ring.

Therefore, the initiator sends the replacement-message containing the available time information clockwise to B_2 and counterclockwise to B_3 . All the bridges take the time information from the replacement-message (bridges in green circles).

Fail-omission or fail-silence of the initiator is tolerated by its neighbor bridges. After timeout occurrence they send time-messages along the ring. The sub-protocol SP is not need, because the merger exchanges all time information between the two half-rings.

Wrong Delay Measurement. When the true forwarding delay and the indicated forwarding delay differ, every bridge can detect the failure after having received the answer-message, provided the error was not self-compensating, see Sect. 2.2.

Figure 5(a) shows an example where B_4 adds wrong delay information to the answer-message and also flags an error. The wrong delay measurement is detected by B_5 and also confirmed by B_0 . However, the initiator can't localize the failure precisely. Both B_4 and B_3 are suspicious. B_4 is the reporter, because it has first flagged an error (B_3 is the reported-bridge). The initiator sends a replacement-message clockwise to B_3 and counterclockwise to B_4 .

Figure 5(b) addresses a fault of the merger B_3 , which writes wrong delay information to both answer-messages it sends in the half-rings. This is first detected by its neighbors B_2 and B_4 , and also confirmed by B_1 , B_5 and B_0 . In this case the initiator can localize the failure exactly, because the two neighbor bridges of the merger noticed that (and both are reporters). Consequently the merger is identified as the faulty bridge. The replacement-message is sent clockwise up to the right neighbor B_2 and counterclockwise up to the left neighbor B_4 .

The wrong delay measurement (in the time-message) of the initiator is tolerated locally in each bridge by FTMA. If the initiator adds wrong delay information to the

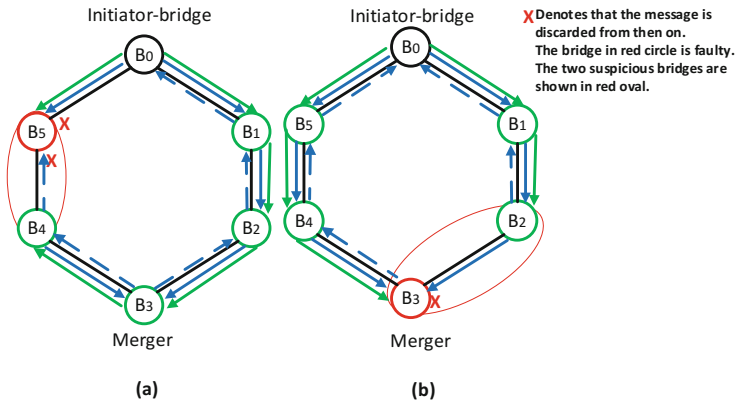


Fig. 4. Tolerance of fail-omission of (a) B₅ and (b) B₃ (the merger). Blue solid arrows are time-messages, blue-dashed arrows are answer-messages, and green arrows are replacement-messages. (Color figure online)

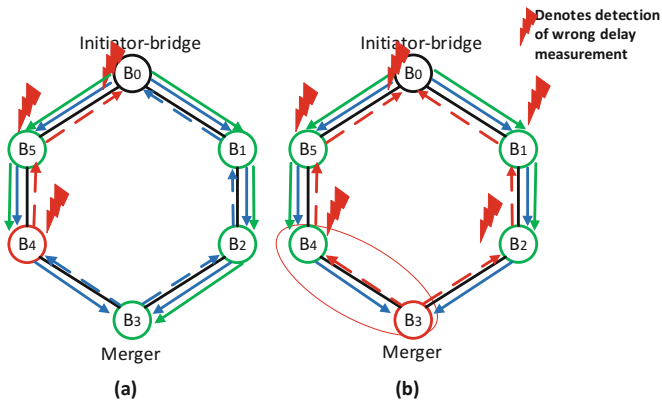


Fig. 5. (a) Bridge B₄ is faulty and commits wrong delay measurement of the answer-message (red-dashed arrows) (b) The merger B₃ is faulty and commits wrong delay measurement of the answer-message (red-dashed arrows) in both half-rings. (Color figure online)

replacement-message this can be detected by its neighbor-bridge or bridges and the message is discarded.

4 Analysis of the Protocol

4.1 Quantification of Temporal Behavior

Figure 6 shows the behavior of two external clocks. Their clock times deviate at most by β before and at most by α after adjustment of their clocks. The complete

synchronization interval $T_{nextSync}$ from one adjustment to the next is composed of the duration $T_{protocol}$ for the execution of the synchronization protocol and the temporal separation T_{sep} for controlling the frequency of clock synchronization, to achieve a desired compromise between the maximum clock deviation β and the overhead of the periodic synchronization protocol. Moreover, bridges must wait $T_{waitA} = \beta$ before clock adjustment to guarantee that all bridges have finished their synchronization protocol. After adjustment the waiting time $T_{waitP} = \beta$ guarantees that no bridge starts the next protocol execution before all bridges have adjusted their clocks. When subdividing $T_{protocol}$ into T_{FP} and T_{SP} for the sub-protocols FP and SP, we get

$$T_{nextSync} = T_{waitP} + T_{sep} + T_{protocol} + T_{waitA} = T_{sep} + T_{FP} + T_{SP} + 2\beta \quad (6)$$

The durations of the sub-protocols FP and SP depend on the number of hops, h_{FP} and h_{SP} , respectively, the maximum delay T_{forw} for forwarding a message by a fault-free bridge, the inaccuracy τ of measuring the actual forwarding delay, and the maximum clock drift ρ which may temporally extend the protocol execution by a factor of $(1 + \rho)$. The inaccuracy of delay compensation is 2τ because the wrong delay measurement can be both positive or negative. This leads to:

$$T_{FP} = (h_{FP} \times (T_{forw} + \beta) + 2\tau) \cdot (1 + \rho) = (n \cdot (T_{forw} + \beta) + 2\tau) \cdot (1 + \rho) \quad (7)$$

$$T_{SP} = (h_{SP} \times (T_{forw} + \beta) + 2\tau) \cdot (1 + \rho) = ((n - 1) \cdot (T_{forw} + \beta) + 2\tau) \cdot (1 + \rho) \quad (8)$$

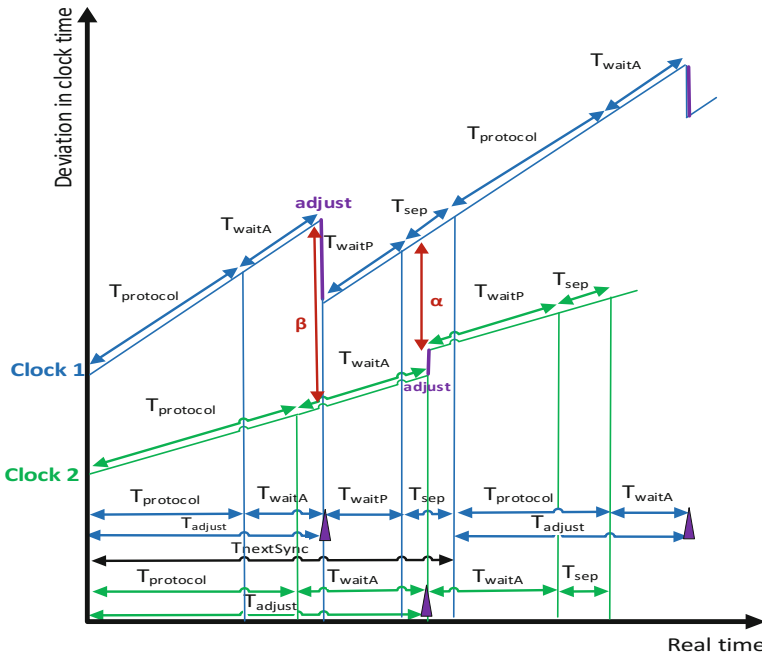


Fig. 6. Main parameters of the analysis

where n is the number of bridges in the ring. When FP starts, the timeout T_{adjust} for clock adjustment is set to:

$$T_{\text{adjust}} = T_{\text{FP}} + T_{\text{SP}} + T_{\text{waitA}} = ((2n - 1) \cdot (T_{\text{forw}} + \beta) + 4\tau) \cdot (1 + \rho) + \beta \quad (9)$$

In SFC a timeout $T_{\text{timeMsg}}(i)$ is used by each bridge I for checking the timeliness of an incoming time-message. Its value depends on the number $h_{\text{initiator},i}$ of hops from the initiator to bridge i . Accordingly the timeout $T_{\text{answerMsg}}(i)$ for an answer message depends on the number $h_{\text{merger},i}$ of hops from the merger to bridge i :

$$T_{\text{timeMsg}}(i) = (h_{\text{initiator},i} \cdot (T_{\text{forw}} + \beta) + 2\tau) \cdot (1 + \rho) \quad (10)$$

$$T_{\text{answerMsg}}(i) = (h_{\text{merger},i} \cdot (T_{\text{forw}} + \beta) + 2\tau) \cdot (1 + \rho) \quad (11)$$

After successful adjustment of the clocks to a difference of at most α the clocks drift apart and reach a difference of up to β at the end of a synchronization interval. The effect is caused by the oscillator drift ρ that can be both positive or negative, thus

$$\beta = \alpha + 2\rho T_{\text{nextSync}} \quad (12)$$

Clock adjustment corrects the clocks according to FTMA which guarantees a convergence of $1/2$ plus the difference e in time by which two fault-free bridges observe other clocks (6) where $e = n \cdot (1 + 2\rho) \cdot 2\tau$. Consequently

$$\alpha = 1/2 \cdot \beta + n \times (1 + 2\rho) \cdot 2\tau \quad (13)$$

By substituting α from (13) and T_{nextSync} from (6) with T_{FP} and T_{SP} from (7) and (8) in Eq. (12) we resolve the maximum deviation β between fault-free clocks, which is the main achievement of clock synchronization:

$$\beta = \frac{n\tau \cdot (1 + 2\rho) + \rho T_{\text{sep}} + ((2n - 1) \cdot T_{\text{forw}} + 4\tau) \cdot \rho \cdot (1 + \rho)}{1/4 - (2n - 1) \cdot \rho \cdot (1 + \rho) - 2\rho} \quad (14)$$

As can be easily seen, this expression is linear in all four, the maximum message forwarding delay T_{forw} , the temporal separation T_{sep} , the inaccuracy of local delay measurement τ and the number n of bridges. The linear dependency of the inaccuracy from the distance between the nodes is a general property of synchronization protocols.

4.2 Overhead

The number of messages transmitted in the absence of faults is $n/2$ time-messages and $n/2$ answer messages in each half-ring leading to a total of $N_{\text{message-fault-free}} = 2 \cdot 2 \cdot n/2 = 2n$ messages in FP. In the case of error detection the initiator launches SP and sends a replacement-message which passes up to $n - 1$ bridges. Therefore, the worst-case number of messages is $N_{\text{message-max}} = N_{\text{message-fault-free}} + n - 1 = 3n - 1$.

This formula shows that overhead of the protocol used in a ring-network is linear in number of bridges, which is more than the overhead in a tree-network, but compared to other protocols used in redundant topologies with requirement of more disjoint paths between bridges, the overhead is only linear rather than square.

4.3 Simulation

SFC has been implemented on a microcontroller network for validation and demonstration purposes.

Furthermore the protocol has been simulated to evaluate the quality of the synchronization. The maximum clock deviation β among the bridges and the average message number $N_{message-average}$ have been quantified for different failure types (and combinations thereof) in different scenarios with variation of the following parameters:

- ρ the clock drift, randomly chosen from $[10^{-5}, 10^{-3}]$. This corresponds to good and medium quartz oscillators.
- Δ the inaccuracy of the local delay measurements, randomly chosen from the range $[-\tau, +\tau]$, where $\tau = 0.5$ (time unit).
- θ the forwarding delay, randomly chosen from $[0, T_{forw}]$, where $T_{forw} = 1$ (time unit).

The simulation has been conducted with 1.000.000 samples for each parameter combination for $n = 6$ bridges, where all forwarders are sources ($n = s$). Six bridges may correspond to small and medium size networks. Larger networks are likely to be built by a collection of connected rings. The simulations have shown that the worst-case occurs rarely, even in the presence of a fault. This holds for all investigated malfunctions of the faulty bridge, see Table 1.

Table 1. Max deviation and Overhead.

	Fault-free	Fail-omission	Delay	Delay and fail-omission	Corruption	Corruption and delay
β	0,4079	0,4259	4,0271	2,6654	2,4	2,56
$N_{message-average}$	12	13,250085	14,20654	13,882075	13,99931	14,338153

5 Conclusion

The new fault-tolerant clock synchronization protocol SFC is appropriate for ring networks where the number of disjoint paths between bridges is kept low, which leads to a low-cost solution in contrast to fault-tolerant protocols using either a fully-meshed network or a special redundant topology. Furthermore the protocol guarantees the tolerance of all type of failures of one bridge. If self-compensating errors can be excluded, the precision of the synchronization compares to those of other protocols when an equal number of hops between the bridges is presupposed. Otherwise the precision depends also on the forwarding speed along the ring.

Currently, extensions of SFC are studied, in particular with respect to large topologies. Networks of connected rings, for example, can be synchronized by executing SFC in just one ring and adding a protocol similar to SFC in every other ring to distribute the synchronized time network-wide.

References

1. Mills, D.L.: Internet time synchronization: the network time protocol. *IEEE Trans. Commun.* **39**, 1482–1493 (1992)
2. Eidson, J.C., Fischer, M., White, J.: IEEE-1588 standard for a precision clock synchronization protocol for network measurement and control systems. In: *Proceedings of the 34th Annual Precise Time and Time Interval Meeting*, pp. 243–254. IEEE (2002)
3. Ferrari, P., Flammini, A., Marioli, D., Taroni, A.: IEEE 1588-based synchronization system for a displacement sensor network. *IEEE Trans. Instrum. Meas.* **57**(2), 254–260 (2008)
4. IEEE Instrumentation and Measurement Society. IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems and Control Systems
5. Gaderer, G., Loschmidt, P., Sauter, T.: Improving fault tolerance in high-precision clock synchronization. *IEEE Trans. Ind. Inform.* **6**(2), 206–215 (2010)
6. Gaderer, G., Rinaldi, S., Kero, N.: Master failures in the precision time protocol. In: *Proceedings of IEEE ISPCS*, pp. 59–64, 22–26 September 2008
7. Welch, J.L., Lynch, N.: A new fault-tolerant algorithm for clock synchronization. *Inf. Comput.* **77**(1), 1–36 (1988)
8. The Institute of Electrical and Electronic Engineers. ANSI/IEEE Std 801.2D, Media Access Control (MAC) Bridges (2004)
9. Kirrmann, H., Kleineberg, O.: Seamless and low-cost redundancy for substation automation systems (high availability seamless redundancy HSR). In: *Proceedings of the IEEE Power and Energy Society General Meeting*, San Diego, USA, pp. 1–7, 24–29 July 2011
10. IEC 62439-3(Ed. 2). Industrial communication networks: High availability automation networks Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR), Rev. Ed.2 (2012)
11. Hall, B., Driscoll, K., Paulitsch, M., Dajani-Brown, S.: Ringing out fault tolerance. a new ring network for superior low-cost dependability. In: *2005 International Conference on Dependable Systems and Networks (DSN 2005)*, pp. 298–307, June 2005
12. Ruiz-Martínez, A., Sánchez-Martínez, D., Martínez-Montesinos, M., Gómez-Skarmeta, A. F.: A survey of electronic signature solutions in mobile devices. *J. Theor. Appl. Electron. Commer. Res.* **2**(3), 94–109 (2007)
13. Echte, K., Kimmekamp, T.: Fault-tolerant and fail-safe control systems using remote redundancy. In: *21th International Conference on Architecture of Computing Systems ARCS 2009, Delft, ARCS 2009 Workshop Proceedings*, pp. 101–106 (2009)
14. Echte, K., Moztarzadeh, Z.: Fault-tolerant clock synchronization in ring-networks. In: *Proceedings of the Symposium on Applied Computing 2017*, pp. 465–468 (2017)



MORE: Model-based Redundancy for Simulink

Kai Ding^(✉), Andrey Morozov, and Klaus Janschek

Institute of Automation, Technische Universität Dresden, Dresden, Germany
{kai.ding, andrey.morozov, klaus.janschek}@tu-dresden.de

Abstract. Fault tolerance plays a significant role in the safety-critical system design that enables a system to continue performing its intended functions in presence of faults. Redundancy is the key underlying method to achieve fault tolerance. Hardware redundancy and software redundancy are well-known redundancy techniques. In case of model-based development, redundancy mechanisms can be applied directly at the model level, e.g. to a Simulink model. This paper introduces a new, model-based redundancy technique to tolerate hardware faults, called *model-based redundancy* (MORE). Applications of fault-tolerant design patterns, such as *comparison*, *voting*, and *sparing*, to Simulink models are introduced. A Simulink PID controller model is demonstrated as a case study to show the effectiveness and feasibility of the introduced approach. The paper also addresses the mutual optimization of reliability properties and system performance. We apply the MORE separately to the P , I , D terms and analyze system performance and achieved reliability properties, evaluated using a stochastic dual-graph error propagation model.

Keywords: Fault tolerance · Redundancy · Model-based design
Dependability · Reliability · Design patterns · Stochastic method
Soft errors · Silent data corruption · Simulink

1 Introduction

Model-based development is an efficient, reliable, and cost-effective paradigm for the design and implementation of complex embedded systems, e.g. in safety-related applications. The commonly used modeling methods include UML/SysML, AADL, MATLAB[®] and Simulink[®]. MATLAB/Simulink is one of the most well-known model-based system development paradigm, which is very popular among control engineers. Model-based design with MATLAB/Simulink helps to improve the product quality and reduce the development time. Simulink Coder[™] provides automated generation of executable code and deployment on a target hardware platform that help to avoid manual coding errors.

Embedded systems are prone to hardware faults, such as bit-flips. The likelihood of the occurrence of these faults is increasing due to the continuously

decreasing feature sizes of integrated circuits. A Single event upset (SEU), as a result of interfering electromagnetic fields or radiation, is a state change caused by a single ionizing particle. An SEU can cause a bit-flip, which is an unintentional change of a bit state from 0 to 1, or vice versa in memory of microelectronic devices [5, 21]. Bit-flips may occur in safety-critical systems, e.g. in space [27] or automotive [6] applications. Electronic devices might exhibit abnormal behavior due to the occurrence of bit-flips. During the flight of the spacecraft Cassini, the NASA reports a rate of 280 soft errors per day [26]. Bit-flips can be manifested in a microprocessor as e.g. timing errors, control-flow errors and data errors [3, 24]. Data errors occur when a bit-flip alters the content of a memory cell or a CPU register and are more common than timing or control-flow errors [3].

The concept of *model-based redundancy* is inspired by the *model-implemented fault injection* [3, 25]. Well-known approaches are hardware-implemented fault injection and software-implemented fault injection. For safety analyses during model-based development, fault injection mechanisms can be applied directly to models. Tools, like MODIFI (MOdel-Implemented Fault Injection) [25] and ErrorSim [19], can inject faults in behavior models developed using MATLAB Simulink in order to evaluate model robustness against several types of hardware faults. The injection of bit-flips into Simulink models for robustness assessment is addressed in [23, 24].

The remainder of the paper is organized as follows. Section 2 briefly discusses the related work in the domains of fault tolerance and redundancy, soft errors protections at various levels of abstraction. Section 3 is dedicated to the description of the *model-based redundancy* using the applications of the design patterns, e.g. *voting pattern*, *comparison pattern*, to an illustrative Simulink PID controller model. The analytical evaluation method is introduced in Sect. 4. Finally, the conclusions are given in Sect. 5.

2 State of the Art

2.1 Fault Tolerance and Redundancy

Traditionally, fault-tolerant designs are classified into hardware, software, time, and information redundancy. Hardware redundancy is used to tolerate hardware systematic and/or random faults, whereas software redundancy is used to deal with software systematic faults, e.g. bugs, or hardware faults. Hardware redundancy techniques are classified into three types [4]: passive can mask faults, while active is used for faults detection and recovery, hybrid redundancy is a combination of active and passive redundancy. Software redundancy techniques can be divided into two groups [10]: single version techniques achieve fault tolerance of a software component by the mechanisms for fault detection, containment, and recovery, while multi-version techniques employ design diversity to tolerate software design faults.

We have proposed a new classification of implementation-independent fault-tolerant designs that organizes existing fault tolerance techniques into a structured pattern system [1]. Most of the well-known fault-tolerant designs follow

these patterns. *Comparison*, *voting*, and *sparing* are three basic design patterns. *Comparison pattern* is used to detect faults, *voting pattern* is applied to mask faults, and *sparing pattern* can detect a fault with a built-in error detection unit and switch to a spare component. These three basic design patterns can be combined in different ways to improve the dependability even further. Four combined design patterns are also introduced in [1].

The difference between the software redundancy and the proposed *model-based redundancy* is that software redundancy techniques are mainly used to deal with software systematic faults (design faults), whereas *model-based redundancy* is applied in the design phase to tolerate soft errors, caused by hardware faults.

2.2 Soft Errors Protection at Hardware, Software, and Model Levels

A soft error is a type of error where a signal or datum is wrong that can be caused by hardware faults, e.g. bit-flips, and can be remedied by a reboot of the system. Various approaches have been proposed to protect the system from soft errors in memory and processor using both hardware and software techniques. Soft error detection and recovery using special hardware to replicate an application execution and detect errors in the output are addressed in [14, 16]. Reliable systems typically exploit hardware fault-tolerant techniques, such as triple modular redundancy, to mask soft errors. Several approaches have been developed to deal with unexpected bit-flips, including RAM parity checking, and error correcting codes (ECC). However, hardware redundancy increases design complexity and cost. It is demonstrated that detection of most transient faults can be accomplished without the need for specialized hardware [17]. Many authors have proposed compiler-level instruction replication and result checking for software-based soft error tolerance, such as error detection by duplicating instructions (EDDI) [15] or by AN Encoding [18, 20], software implemented fault tolerance (SWIFT) [17], and replication of data inside AVX registers with ELZAR [8]. Soft errors protection at software is lower-cost and flexible, however, the methods are dependent on a hardware platform.

2.3 Contributions of the Paper

Redundancy is traditionally developed and applied at hardware or software levels. For fault-tolerant designs during model-based development, which are increasingly being used during system development, redundancy mechanisms can be applied directly at the model level, e.g. to a Simulink model, to tolerate soft errors. Thus, we define this approach as *model-based redundancy* that is achieved in the Simulink model by providing two or more copies of a block. MORE can achieve soft errors tolerance because program instructions are replicated and values are compared automatically, instead of manually replication of instructions. MORE brings also the inevitable drawback of the system performance

degradation in terms of additional used resources, e.g. memory, computation time. However, the application of MORE has several attractive features:

1. First and foremost, the technique is independent of any development tool-chain (code generators, compilers) and target hardware platform.
2. Second, redundancy can be applied in the design phase and will be included in the generated software, compiled and deployed into the target platform automatically.
3. Third, for model-based design engineers, it is more transparent to protect vulnerable parts directly with fault tolerance mechanisms at the model level.

These are the main driving forces behind the concept of the *model-based redundancy*. We apply the MORE to Simulink, since it is widely used and provides automated code generation for the proof of concept. The MORE approach can be extended equally to other model types or tools, such as TASTE (ESA), Embedded Software Designer (Siemens), or SCADE (Esterel Technologies).

Another contribution is the optimization method of reliability properties and system performance based on stochastic error propagation analysis [11, 12]. We apply the MORE as a case study to the P , I , D terms of the PID controller, respectively, and compare achieved reliability properties and system performance, in order to show the most suitable design solution to apply the MORE approach.

3 Model-Based Redundancy

3.1 Application of the *Voting Pattern* in Simulink

In the *voting pattern* [1], the components are replicated to perform the same operation in parallel. The N produced results are compared by a majority-voting system (a voter) in order to determine the correct result. N is selected to be odd and in the basic case, N equals three. The system functions correctly as long as the majority of components are fault-free. This pattern masks $(N - 1)/2$ faults.

For illustrative purposes, we use a Simulink PID controller model as a demonstrative example. In Simulink, the *Gain*, *Discrete-Time Integrator*, and *Discrete Derivative block* can be used to model the P , I , D terms of a PID controller, respectively. The particular implementation of the *voting pattern* to the P term of a PID controller is shown in Fig. 1, where the *Gain block* is triplicated, one *Relational Operator block* and one *Switch block* are used. The *Relational Operator block* applies the selected relational operator, here “==”, to the inputs. If two inputs agree, it generates a Boolean signal TRUE, i.e. 1. For a control input of 1, the *Switch block* passes through the first input, otherwise the third input. In the *voting pattern*, a fault in one *Gain block* can be masked. Note that the value e in the PID control system is assumed to be always correct, even in case of an erroneous y .

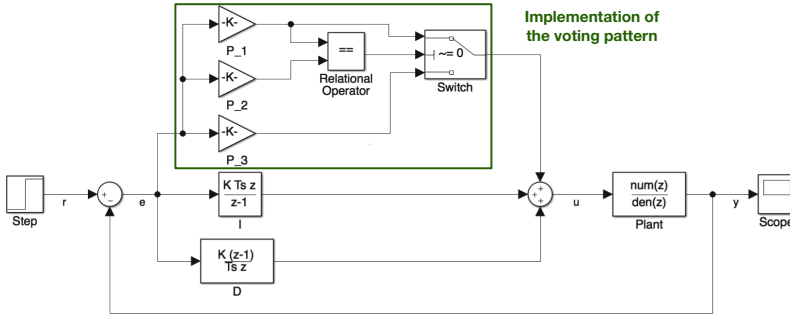


Fig. 1. Application of the *voting pattern* to the *P* term of a PID controller.

3.2 Application of the *Comparison Pattern* in Simulink

Figure 2 shows the application of the *comparison pattern* to the *P* term of a Simulink PID controller, where two components (*Gain blocks*) perform the same computation in parallel and their results are compared by a comparator (*Relational Operator block*) in order to detect an error. If an error is detected, an error signal is generated either to shut down the entire system or to switch the system into a fail-safe state. If the two inputs data are not equal, the *Relational Operator block* outputs a Boolean signal TRUE, then the *Stop Simulation block* stops the simulation. In [28], the authors introduced a robust duplex integrator with a recovery buffer which can detect and recover (roll-back to the previous integrator state) from errors caused by bit-flips.

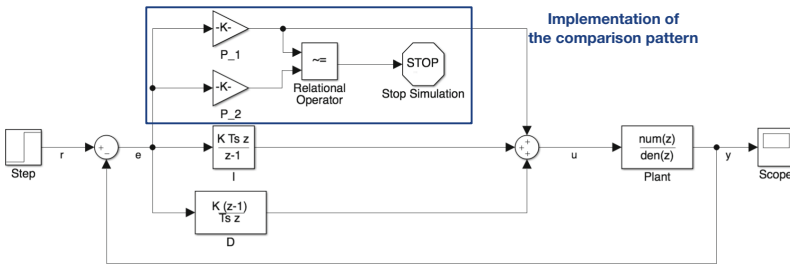


Fig. 2. Application of the *comparison pattern* to the *P* term of a PID controller.

3.3 Application of the *Sparing Pattern* in Simulink

In the *sparing pattern*, only one component is operational and the remaining $N - 1$ components serve as spares. If an error is detected by a built-in error detection unit in an active component, a spare component takes over. This enables a system to continue its correct operation. An error detection technique is required in this pattern. An error detection unit identifies faults, e.g. by range check or

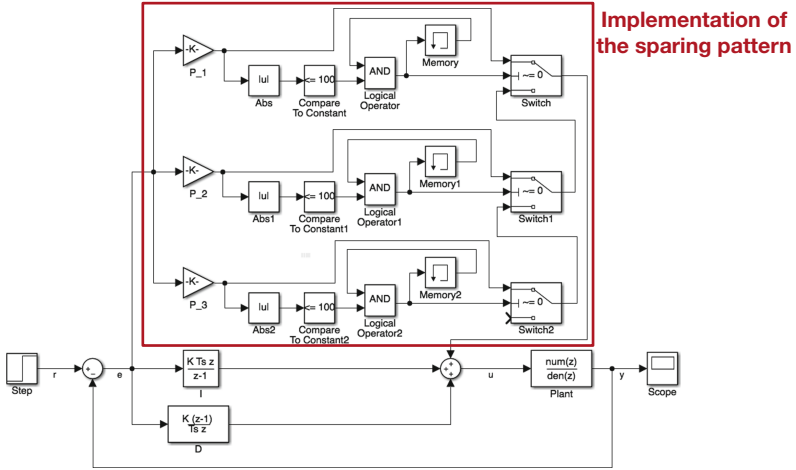


Fig. 3. Application of the *sparing pattern* to the *P* term of a PID controller.

state check. If an error is detected in the active component, the switch will turn the operation to a spare component. Ideally, this pattern can tolerate $N - 1$ faults. However, not all the faults can be detected by the error detection unit. Koren [7] has defined the coverage factor c as the probability of successful detection that the faulty active component will be correctly diagnosed, identified and disconnected. Figure 3 shows the application of the *sparing pattern* to the *P* term of a Simulink PID controller using range check. Other detection mechanisms, e.g. state or derivative checks, can also be implemented. The *Compare To Constant* block checks the range of the output of *P* term. The *Memory* block applies one integration step delay and its output is the previous input value. The initial condition is specified as 1. If the output of the *P* term is bigger than the user-defined maximum value, then the *Switch* block receives a FALSE control signal and turns the operation to a spare one. The faulty component will never be switched back. The application of the *sparing pattern* to the *P* term of a PID controller helps to tolerate soft errors that occur in the gain values.

3.4 Application of the *Comparison Then Sparing Pattern* in Simulink

Figure 4 shows the application of the *comparison then sparing pattern* to the *P* term of a Simulink PID controller. All the components are grouped in pairs. The first pair is on-line and its outputs are compared to detect a mismatch. If the results disagree, another pair takes over. The system is operational until the last pair fails.

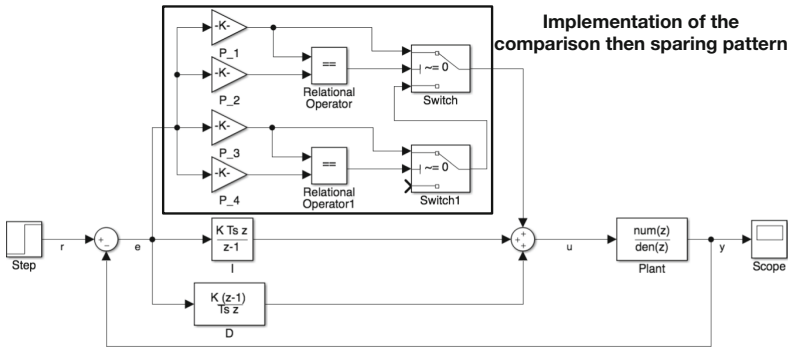


Fig. 4. Application of the *comparison then sparing* pattern to the *P* term of a PID controller.

4 Evaluation

In order to show the effectiveness and feasibility of the *model-based redundancy* approach for Simulink models, we analytically evaluate the reliability properties of the application of the *voting pattern* to *P*, *I*, *D* terms of a Simulink PID controller, respectively (see Fig. 1). The analytical method numerically evaluates the system reliability using underlying discrete-time Markov chain models that are parametrized with data errors probabilities. Therefore, the method can support, in particular, realistic low probabilities of the error occurrence, in contrast to the experimental approaches, e.g. by MODIFI or ErrorSim, that require a huge number of simulations in order to obtain confident results. The PID controller is tuned for a plant model with the following parameters: proportional gain $K_p = 49.63$, integral gain $K_i = 118.9$, derivative gain $K_d = 5.178$, sample time $T_s = 0.05$ s, and end time $T_{end} = 5$ s. The output u of the PID controller is considered to be critical since it is a control variable that is sent to the plant. Figure 5 shows an overview of the analytical evaluation process. The redundancy mechanism is added at the model level, then Simulink Coder generates the C code automatically. The next step is the compilation of the generated high-level code into the low-level assembly code. Then, the assembly code is transformed into the dual-graph error propagation model (DEPM) [2, 11, 12] for the error propagation analysis. After that, probabilistic modeling of data errors is performed in the DEPM. Finally, the reliability properties, computed from the DEPM applying stochastic error propagation analysis, and the system performance, evaluated from the assembly code, are analyzed in terms of a reliability-performance Pareto frontier.

Generated C Code: Redundancy at Simulink model level will be automatically generated inside the executable source code, then compiled and deployed on a target hardware platform. In Fig. 6a, seven user-specified variables of the model are declared and initialized. The variable type *real.T* is used as the generalization

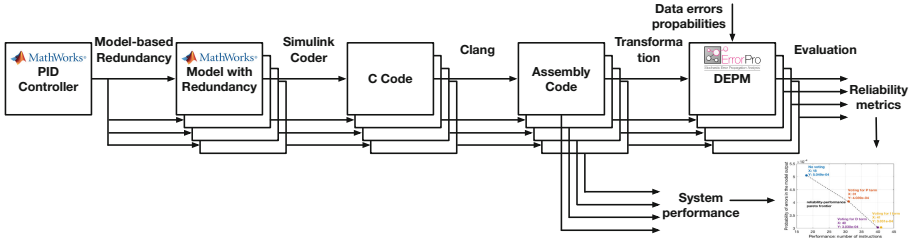


Fig. 5. Overview of the analytical evaluation process.

of the *float* and *double* data types. The *initialize* function in Fig. 6b is invoked only once at the start of the execution to initialize the state variables.

```

/* Block parameters (auto storage) */
real_T P_1_Gain = 49.63; /*by 'P_1'*/
real_T P_2_Gain = 49.63; /*by 'P_2'*/
real_T P_3_Gain = 49.63; /*by 'P_3'*/
real_T I_gainval = 5.945; /*by 'I'*/
real_T TSamp_WtEt = 103.56; /*by 'TSamp'*/
real_T LIC = 0.0; /*by 'I'*/
real_T UD_IC = 0.0; /*by 'UD'*/

/* Model initialize function */
void initialize(void)
{
    /* InitializeConditions for Integrator: 'I' */
    I_DSTATE = I_IC;
    /* InitializeConditions for UnitDelay: 'UD' */
    UD_DSTATE = UD_IC;
}
    
```

(a) The model parameters. (b) The initialize function.

Fig. 6. The generated model parameters and the initialize function.

Figure 7a shows the main functional fragment of the automatically generated C code, the *step* function, which is invoked in each iteration to calculate the output *u* of the PID controller and update the state variables. The *Gain block* is triplicated, and *P_1.Gain*, *P_2.Gain*, and *P_3.Gain* are the specified gain values that are stored in different addresses (memory). In our experiments, we explicitly check that the compiler applies no optimization and stores the values in three different addresses. The multiplication of *P_1.Gain* and *e* is compared to the multiplication of *P_2.Gain* and *e*. If they disagree, *P_3.Gain* continues with the rest of the execution. MORE efficiently manages redundancy by reclaiming values and inserts comparison function at specific points at the code-level during automated code generation. During execution, values are effectively computed and compared to determine the correct value.

Compiled Assembly Code: The *step* function in Fig. 7a is compiled into the assembly code with Clang (Apple LLVM version 8.1.0 (clang-802.0.42)). The resulting assembly code has 31 instructions, presented in AT&T syntax in Fig. 7b. In general, each instruction consists of an operation and two operands. The first operand is the source operand and the second operand is the destination operand. Register names are prefixed by %. Instruction 1 loads the value of *P_1.Gain*. The multiplication of *P_1.Gain* and *e* is stored in $-8(\%rbp)$. Notice that Instruction 2 loads the value of *P_2.Gain*. The multiplication of *P_2.Gain*

and e is stored into a different register $\%xmm0$. Instruction **3** compares the value stored in $-8(\%rbp)$ with the value stored in $\%xmm0$. If the results disagree, Instruction **4** loads the value of P_3_Gain from a different memory and continue with the rest execution.

```

1: movsd   _P_1_Gain(%rip), %xmm0
   mulsd   _e(%rip), %xmm0
   movsd   %xmm0, -8(%rbp)
2: movsd   _P_2_Gain(%rip), %xmm0
   mulsd   _e(%rip), %xmm0
3: ucomisd -8(%rbp), %xmm0
   jne LBB1_1
   jp LBB1_1
   jmp LBB1_2
LBB1_1:
4: movsd   _P_3_Gain(%rip), %xmm0
   mulsd   _e(%rip), %xmm0
   movsd   %xmm0, -8(%rbp)
LBB1_2:
   movsd   _I_gainval(%rip), %xmm0
   mulsd   _e(%rip), %xmm0
   addsd   _I_DSTATE(%rip), %xmm0
   movsd   %xmm0, -24(%rbp)
   movsd   _e(%rip), %xmm0
   mulsd   _TSamp_WtEt(%rip), %xmm0
   movsd   %xmm0, -16(%rbp)
   movsd   -8(%rbp), %xmm0
   addsd   -24(%rbp), %xmm0
   movsd   -16(%rbp), %xmm1
   subsd   _UD_DSTATE(%rip), %xmm1
   addsd   %xmm1, %xmm0
   movsd   %xmm0, _u(%rip)
   movsd   -24(%rbp), %xmm0
   movsd   %xmm0, _I_DSTATE(%rip)
   movsd   -16(%rbp), %xmm0
   movsd   %xmm0, _UD_DSTATE(%rip)
}

```

(a) The generated model step function.

(b) The compiled assembly code.

Fig. 7. The generated model step function and the compiled assembly code.

Generated DEPM: Recently, we have introduced a stochastic dual-graph error propagation model (DEPM) for the error propagation analysis of safety-critical systems [2, 11, 12]. This model is a formal mathematical abstraction that captures control and data flow aspects of a system and allows the computation of several reliability metrics, such as (i) *mean number of errors* (N_{err}) and (ii) *probability of errors* (P_{err}), in critical system outputs for specified faults probabilities, using underlying discrete-time Markov chain models. The former metric, N_{err} , is the average number of occurred errors during the given system execution time in a data storage and the latter metric, P_{err} , is the occurrence probability of an error during the given system execution time in a data storage. Discrete-time Markov chain models are computed using an interface with the PRISM model checker [9].

The probabilistic modeling of data errors caused by bit-flips in RAM is conducted at the assembly code level. Thus, the compiled assembly code in Fig. 7b is transformed into a DEPM, shown in Fig. 8, based on the following mapping rules: (i) The operations, such as *movsd*, *mulsd*, *addsd*, are mapped into DEPM elements, e.g. *mov1* or *mul1*. Each element may receive input data and provide output data. (ii) The operands, such as *P_1_Gain*, *xmm0*, are mapped into DEPM data storages that can be read or written by an Element. (iii) The execution sequences of instructions are mapped into the DEPM control-flow arcs (black lines in Fig. 8), weighted with transition probabilities. (iv) The relations between the operation and its operands are mapped into the DEPM data-flow arcs (blue lines in Fig. 8) that describe data transfer between the elements.

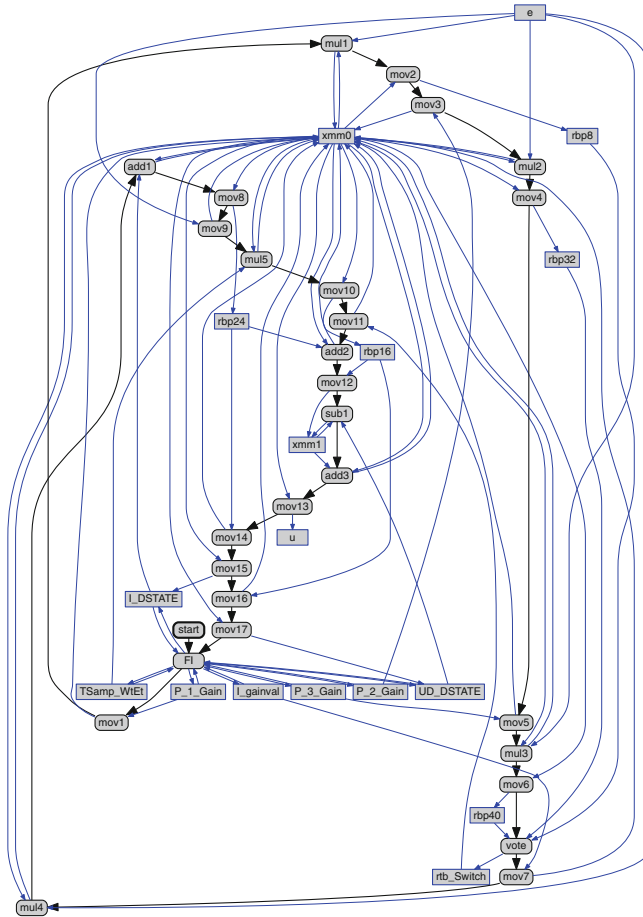


Fig. 8. The DEPM that is generated from the compiled assembly code (Fig. 7b). (Color figure online)

The data errors occurrence probabilities (fault activation), as well as the error propagation through the DEPM elements, are modeled using probabilistic conditions of the DEPM elements. It is assumed that the data errors probability in each variable during the sample time, i.e. each iteration, is independent and defined as $p_{RAM} = 1.0e-6$. The assumed fixed data errors probabilities are selected for the simplicity and transparency of the method explanation. The discussion on varying probabilities and real-world applications will follow later.

Instead of fault injection simulation, we focus on the probabilistic modeling of data errors caused by bit-flips in RAM and the error propagation analysis. An additional DEPM element *FI* is generated (see Fig. 8), and connected with all seven user-specified variables of the *step function*. The element *FI* is executed at the beginning of each control loop iteration to model data errors occurrence with probability p_{RAM} . Each data storage in the generated DEPM has two states: correct or erroneous. The conditions of the element *FI* are specified as an independent combination of each variable. For instance, the conditions for the variable *P_1_Gain* in *FI* are specified, as follows: (i) if *P_1_Gain* is correct: a) then *P_1_Gain* is correct with probability $1-p_{RAM}$, b) or *P_1_Gain* is erroneous with probability p_{RAM} , (ii) if *P_1_Gain* is erroneous: a) then *P_1_Gain* is erroneous with probability 1. The computed values, $P_1_Gain \times e$, $P_2_Gain \times e$, and $P_3_Gain \times e$, are stored in different registers *rbp8*, *rbp32*, and *rbp40* (see Fig. 8) that are compared in order to determine the correct value. The element *vote* has 2^3 conditions. For instance, one condition of the element *vote* is specified: (i) if *rbp8* is correct and *rbp32* is correct and *rbp40* is erroneous: (a) then *rtb_Switch* is correct with probability 1. In order to evaluate the reliability metrics, (i) N_{err} and (ii) P_{err} in the output *u*, discrete-time Markov chain models that describe instructions execution and error propagation processes are automatically generated from the DEPM and computed using the PRISM model checker [9].

Numerical Results: The application of the *model-based redundancy* can improve the system tolerance to soft errors, however, it also entails the generation of extra source code, resulting in a considerable computational overhead and, as a consequence, leads to the performance degradations. In our previous work [13], we proposed the approach to minimize the negative performance impact while maintaining the required system reliability level. Figure 9 shows the relationship between the system performance (*number of instructions, used as representative metrics*) and the evaluated reliability properties for the Simulink PID controller model with applications of the *voting pattern* to *P*, *I*, *D* terms with the assumed $p_{RAM} = 1.0e-6$. In Fig. 9a, for the achieved N_{err} , the application of the *voting pattern* to the *D* term of a PID controller is not practical, since compared to the *P* term, the performance changes from 31 to 40, but N_{err} stays almost the same. In Fig. 9b, for the evaluated P_{err} , the applications of the *voting pattern* to the *D* term and to the *I* term show almost the same reliability property and system performance, although voting for *I* term is above the Pareto frontier.

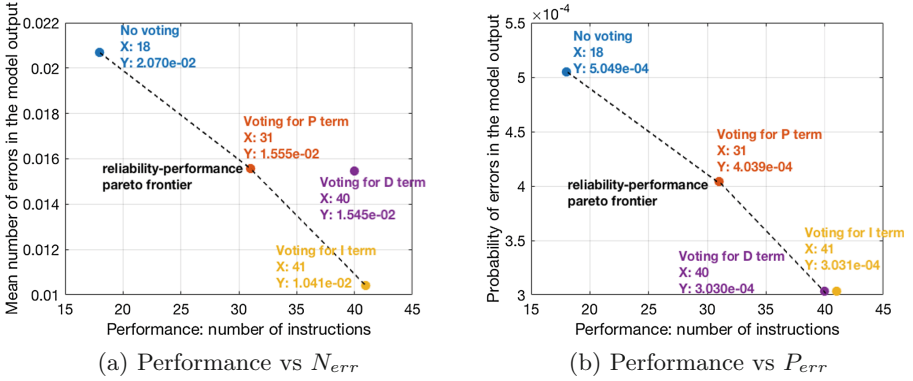


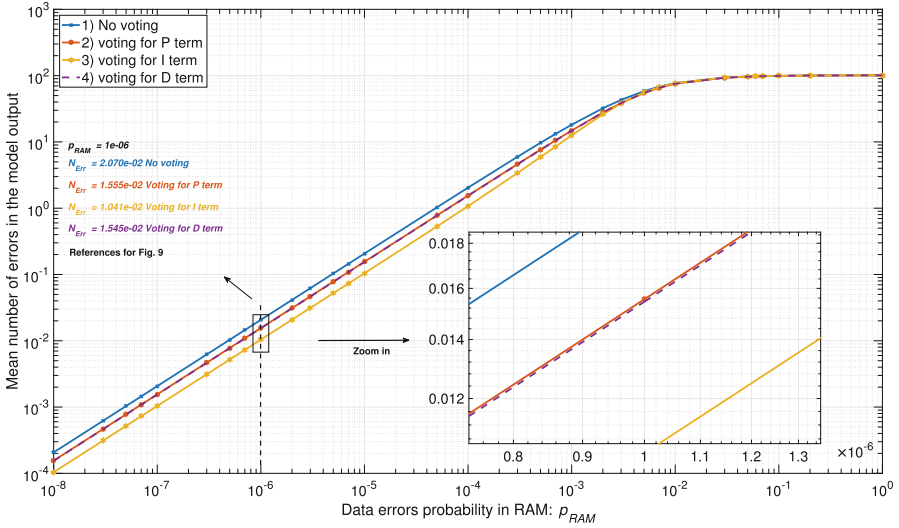
Fig. 9. The relationship between the system performance and the achieved reliability properties for $p_{RAM} = 1.0e-6$.

In reality, the data errors probabilities are strongly dependent on many environmental and technology parameters [22]. Therefore, we evaluate the reliability properties of the Simulink models with a broad range of data errors probabilities in order to give more practical values, shown in Fig. 10. The plots are created using a base 10 logarithmic scale for the x-axis (data errors probabilities from 10^{-8} to 1) and for the y-axis (reliability properties).

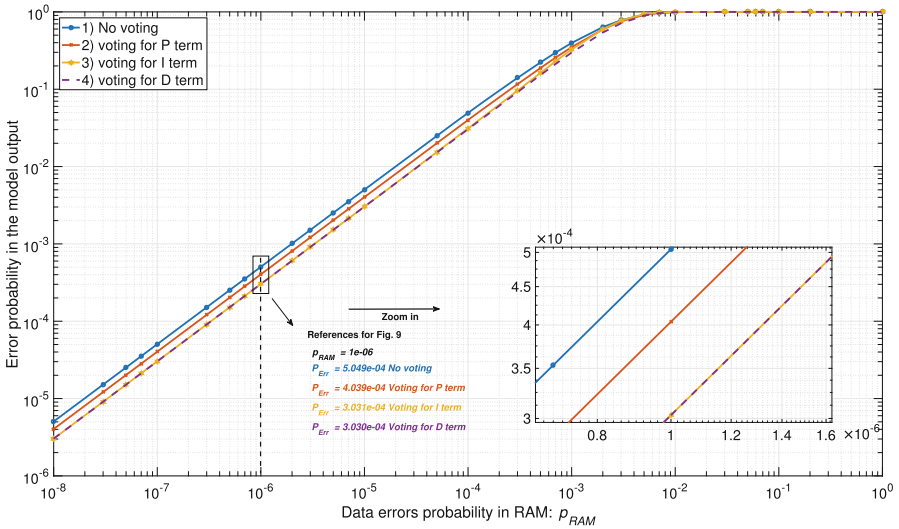
In Fig. 10a, the evaluated N_{err} reaches almost its maximum 101 when the data errors probability in RAM is about 10^{-2} for all plots, since we model altogether 101 iterations from 0s to 5s with $T_s = 0.05$ s, and in the worst case, an error propagates to the critical output in each iteration. For this case study, the reliability analysis has shown that the application of the *voting pattern* to the *I* term of a PID controller (yellow plot) is the most tolerant to data errors caused by bit-flips. The application of the *voting pattern* to the *P* term (red plot) and to the *D* term (dashed purple plot) show almost the same reliability. The evaluated N_{err} with *voting pattern* for the *I* term of a PID controller model is half of that of the PID controller model without any protection.

In Fig. 10b, the application of the *voting pattern* to the *I* term (yellow plot) and to the *D* term (dashed purple plot) almost coincide, showing the lowest P_{err} . Followed by is the application of the *voting pattern* to the *P* term (red plot) of a PID controller. The evaluated P_{err} for the PID controller model with *voting pattern* to the *I* or *D* term is about 60% of that for the PID controller model without any *voting pattern*.

From the system performance comparison in Fig. 9 and the evaluated reliability property in Fig. 10, we can conclude that the application of the *voting pattern* to the *I* term of a PID controller is the most suitable design solution. The analytical method, introduced in this paper, can also be applied to the other dependability design patterns [1], such as *comparison* or *sparing* patterns.



(a) The evaluated *mean number of errors* (N_{err}) in the output of the PID controller with the modeling of data errors with varying data errors probabilities p_{RAM} .



(b) The evaluated *probability of errors* (P_{err}) in the output of the PID controller with the modeling of data errors with varying data errors probabilities p_{RAM} .

Fig. 10. The evaluated reliability properties. (Color figure online)

5 Conclusion

In this paper, we have proposed MORE, a *model-based redundancy* approach to achieve fault tolerance. MORE can be applied in the model-based design phase,

e.g. to a Simulink model, to tolerate soft errors caused by hardware-related faults. Redundancy at the Simulink model level is included inside the automatically generated C code, then compiled and deployed on a target hardware platform. Values and instructions are replicated automatically, then a comparison function is executed to detect soft errors or determine the correct values. Thus, *model-based redundancy* is independent of any development tool-chain or target hardware platform. We have shown the application of fault-tolerant design patterns, such as *comparison*, *voting* patterns, to an illustrative Simulink PID controller model. Assembly code, compiled from the automatically generated C code, was transformed into the DEPM for the stochastic evaluation of reliability properties. Finally, we have addressed the reliability-performance optimization problem building a Pareto frontier for different protection strategies. Future work includes the comparison of the proposed *model-based redundancy* technique with other existing techniques, e.g. SWIFT.

Acknowledgements. This work is supported by the German Research Foundation (DFG) under project No. JA 1559/5-1.

References


1. Ding, K., Morozov, A., Janschek, K.: Classification of hierarchical fault-tolerant design patterns. In: 2017 IEEE 15th International Dependable, Autonomic and Secure Computing, 15th International Conference on Pervasive Intelligence and Computing, 3rd International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), pp. 612–619. IEEE (2017)
2. Ding, K., Mutzke, T., Morozov, A., Janschek, K.: Automatic transformation of UML system models for model-based error propagation analysis of mechatronic systems. IFAC-PapersOnLine **49**(21), 439–446 (2016)
3. Eriksson, H.: D 5.1 - simulating hardware-related faults at model level. Technical report (2011)
4. Johnson, B.W.: Design and Analysis of Fault Tolerant Digital Systems. Addison-Wesley Longman Publishing Co. Inc., Boston (1988)
5. Karnik, T., Hazucha, P.: Characterization of soft errors caused by single event upsets in cmos processes. IEEE Trans. Dependable Secure Comput. **1**(2), 128–143 (2004)
6. Koopman, P.: A case study of Toyota unintended acceleration and software safety. Presentation, September 2014
7. Koren, I., Krishna, C.M.: Fault-Tolerant Systems. Morgan Kaufmann, Burlington (2010)
8. Kuvaiskii, D., Oleksenko, O., Bhatotia, P., Felber, P., Fetzer, C.: Elzar: triple modular redundancy using Intel AVX. In: Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2016) (2016)
9. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
10. Lyu, M.R., et al.: Handbook of Software Reliability Engineering (1996)

11. Morozov, A.: Dual-graph Model for Error Propagation Analysis of Mechatronic Systems. Jörg Vogt Verlag, Dresden (2012)
12. Morozov, A., Janschek, K.: Probabilistic error propagation model for mechatronic systems. *Mechatronics* **24**(8), 1189–1202 (2014). <https://doi.org/10.1016/j.mechatronics.2014.09.005>
13. Morozov, A., Janschek, K.: Flight control software failure mitigation: design optimization for software-implemented fault detectors. *IFAC-PapersOnLine* **49**(17), 248–253 (2016)
14. Mukherjee, S.S., Kontz, M., Reinhardt, S.K.: Detailed design and evaluation of redundant multi-threading alternatives. In: *Proceedings of 29th Annual International Symposium on Computer Architecture*, pp. 99–110. IEEE (2002)
15. Oh, N., Shirvani, P.P., McCluskey, E.J.: Error detection by duplicated instructions in super-scalar processors. *IEEE Trans. Reliab.* **51**(1), 63–75 (2002)
16. Reinhardt, S.K., Mukherjee, S.S.: Transient fault detection via simultaneous multithreading, vol. 28. ACM (2000)
17. Reis, G.A., Chang, J., Vachharajani, N., Rangan, R., August, D.I.: Swift: software implemented fault tolerance. In: *Proceedings of the International Symposium on Code Generation and Optimization*, pp. 243–254. IEEE Computer Society (2005)
18. Rink, N.A., Castrillon, J.: Trading fault tolerance for performance in an encoding. In: *Proceedings of the Computing Frontiers Conference*, pp. 183–190. ACM (2017)
19. Saraoğlu, M., Morozov, A., Söylemez, M.T., Janschek, K.: ErrorSim: a tool for error propagation analysis of simulink models. In: Tonetta, S., Schoitsch, E., Bitsch, F. (eds.) *SAFECOMP 2017*. LNCS, vol. 10488, pp. 245–254. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66266-4_16
20. Schiffel, U., Schmitt, A., Süßkraut, M., Fetzer, C.: ANB- and ANBDMem-encoding: detecting hardware errors in software. In: Schoitsch, E. (ed.) *SAFECOMP 2010*. LNCS, vol. 6351, pp. 169–182. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15651-9_13
21. Schroeder, B., Pinheiro, E., Weber, W.D.: Dram errors in the wild: a large-scale field study. In: *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, pp. 193–204. ACM (2009)
22. *Semiconductors Technology: Soft errors in electronic memory* (2012)
23. Svenningsson, R.: *Model-Implemented Fault Injection for Robustness Assessment*. KTH, Stockholm (2011)
24. Svenningsson, R., Eriksson, H., Vinter, J., Törngren, M.: Model-implemented fault injection for hardware fault simulation. In: *2010 Workshop on Model-Driven Engineering, Verification, and Validation (MoDeVVA)*, pp. 31–36. IEEE (2010)
25. Svenningsson, R., Vinter, J., Eriksson, H., Törngren, M.: MODIFI: a Model-implemented fault injection tool. In: Schoitsch, E. (ed.) *SAFECOMP 2010*. LNCS, vol. 6351, pp. 210–222. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15651-9_16
26. Swift, G.M., Guertin, S.M.: In-flight observations of multiple-bit upset in DRAMs. *IEEE Trans. Nucl. Sci.* **47**(6), 2386–2391 (2000)
27. Verzola, I., Lagny, A.E., Biswas, J.: A predictive approach to failure estimation and identification for space systems operations. In: *SpaceOps 2014 Conference*, p. 1722 (2014)
28. 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: 22/06/2003-25/06/2003*, pp. 415–424. IEEE Communications Society (2003)

Safety and Security Risk



Diversity in Open Source Intrusion Detection Systems

Hafizul Asad and Ilir Gashi^(✉) 

Centre for Software Reliability, City, University of London, London, UK
{hafiz.ul-asad.1, ilir.gashi.1}@city.ac.uk

Abstract. We present an analysis of the diversity that exists in the rules and blacklisted IP addresses of the Snort and Suricata Intrusion Detection Systems (IDSs). We analysed the evolution of the rulesets and blacklisted IP addresses of these two IDSs over a 5-month period between May and October 2017. We used three different off-the-shelf default configurations of the Snort IDS and the Emerging Threats (ET) configuration of the Suricata IDS. Analysing the differences in these systems allows us to get insights on where the diversity in the behaviour of these systems comes from and how does it evolve over time. This gives insight to Security architects on how they can combine and layer these systems in a defence-in-depth deployment. To the best of our knowledge a similar experiment has not been performed before. We will also show results on the observed diversity in behaviour of these systems, when they analysed the network data of the DMZ network of City, University of London.

Keywords: Security assessment · Security tools · Intrusion detection systems
Design diversity

1 Introduction

An important part of design for security is defence-in-depth, consisting of “layers” of defence that reduce the probability of successful attacks. Guidance documents now advocate defence in depth as an obvious need¹ but their qualitative guidance ignores the decision problems. Crucially, these questions concern diversity: defences should be diverse in their weaknesses. Any attack that happens to defeat one defence should with high probability be stopped or detected by another one. Ultimately, diversity and defence in depth are two facets of the same defensive design approach. The important questions are not about defence in depth being “a good idea”, but about whether a set of specific defences would improve security more than another set; and about – if possible – quantifying the security gains.

Network Intrusion Detection Systems (IDSs) are some of the most widely used security defence tools. Some of these IDSs are available open-source, and the most widely used open-source IDSs are Snort and Suricata. Both of these tools are signature-based and rely on rules to identify malicious activity. The rules identify malicious activity based on content, protocols, ports etc., as well as on the origin of the

¹ www.nsa.gov/ia/_files/support/defenseindepth.pdf.

activity/traffic - in this latter case, the suspicious IP addresses are “blacklisted” and traffic originating from these IPs are alerted. Depending on the configuration of the IDS the traffic can be alerted but allowed, or alerted and dropped – the latter happens when the IDS is running in Intrusion Prevention System (IPS) mode.

In this paper, we present an analysis of the diversity that exists between the Snort and Suricata rules and blacklisted IP addresses. We analysed the evolution of the rulesets and blacklisted IP addresses of these two IDSs over a 5-month period between May and October 2017. We used three different off-the-shelf default configurations of the Snort IDS and the Emerging Threats configuration of the Suricata IDS. Analysing the differences in these systems and how they evolve over time, allows us to get insights on where the diversity in the behaviour of these systems comes from. To the best of our knowledge a similar experiment has not been performed before. We will also show results on the observed diversity in behaviour of these systems, when they analysed the network data of the DMZ network of City, University of London.

The rest of the paper is organised as follows: Sect. 2 describes the experimental architecture. The next three sections present results of diversity analysis of the following aspects of Snort and Suricata: blacklists (Sect. 3); rulesets (Sect. 4); behaviour on real network traffic (Sect. 5). Section 6 presents a discussion of the results and limitations. Section 7 presents related work and finally Sect. 8 presents conclusions and further work.

2 Description of the Experiment and the Architecture

We ran an experiment for 5 months from 20th May 2017 to 31st October 2017. During these dates we did the following. We downloaded and saved snapshots of the **black-listed** IP addresses of Snort and Suricata as they were on each day of the experiment. To retrieve the rules we used the pulledpork tool². For Snort, the blacklisted files³ were downloaded every 15 min for the duration of the experiment. We therefore have a total of 15,812 blacklist files for Snort. Note that the total duration of the experiment is 165 days for which there should have been 15,840 files, but in some cases there were no updates for blacklisted IPs in every 15-min slot of our collection period. The black-listed IP addresses for Suricata are located inside the rules files⁴, so we extracted the blacklisted IP addresses from these rule files. We also ran pulledpork every 15 min for Suricata, but contrary to Snort, the rate of updates of the Suricata ET blacklists appear to be on daily basis rather than every 15 min.

We downloaded and saved the **rules** of Snort for three different default rule configurations available from the Snort webpages (Community rules, Registered rules, and Subscribed rules). The difference between these rules are explained in the Snort website⁵. In summary, the website states the following for these different rules: the

² <https://github.com/shirkdog/pulledpork>.

³ <http://labs.snort.org/feeds/ip-filter.blf>.

⁴ <https://rules.emergingthreats.net/open/suricata/emerging.rules.tar.gz>.

⁵ <https://snort.org/documents/registered-vs-subscriber>.

Subscribed (paid) rules are the ones that are available to users in real-time as they are released; the Registered rules are available to registered users 30 days after the Subscribed users; the Community rules are a small subset of the subscribed/registered rule sets and are freely available to all users. For Suricata we used the Emerging Threats (ET) ruleset. We ran the pulledpork to update the rules every 15 min, but we observed that rules were updated on average every 24 h. Similar to blacklisted files, we saved snapshots of these rules files on each day of the experiment.

The University's IT team saved copies of the network traffic (in packet capture (pcap) format) for retrospective analysis of attacks and incidents. We replayed the pcap traffic collected over a one week From 2 May to 8 May 2017, to the three different versions of Snort outlined above and to Suricata ET.

The data collection and analysis infrastructure runs on a virtualized environment based on VMware VSphere data center. This data collection setup has five data hosts each having, 150 TB storage capacity, 200 GB RAM, and 32×2.3 GHz of CPU processing speed. At the start of the experiment, we installed the latest versions of these IDSs on the FreeBSD operating system: Snort 2.9.9.0 and Suricata 3.2.1.

3 Diversity in the IP Blacklists of Snort and Suricata

3.1 Analysis of Each Individual IDS

In this section, we present the analysis of our research on how blacklisted IP addresses evolve over time in Snort and Suricata. As we mentioned previously, we obtained these blacklisted IPs from May 20 2017 to October 31 2017, at a sampling rate of every 15 min. We kept the same sampling frequency for Suricata to make the analysis as comparable as possible, though we observed that the rate of change of the blacklisted files was, in some cases, less frequent than every fifteen minutes for Snort and further less frequent for Suricata (which tended to be every 24 h). Figure 1 shows the evolution of the blacklisted IP addresses as obtained from Snort (left plot) and Suricata (right plot). The y-axis shows the total count of the blacklisted IPs and x-axis shows the data collection points. From Fig. 1 we observe a large fluctuation in the number of blacklisted IP addresses over time. For example around 21 June 2017 a large number of IP addresses were removed from the blacklists. However, afterwards, the number of blacklisted IP addresses increased again. In Suricata we also saw a large drop in the number of blacklisted IP addresses around this time, but the total number of blacklisted IPs did not increase again as it did for Snort.

We note that some IP addresses remained blacklisted for the entire duration of our experiment (or change their states only once, e.g., they are removed from the blacklists), whereas we observed other IP addresses that changed state twice or more (e.g. blacklisted, removed, blacklisted etc.) We therefore divide the IP addresses into those that remained “**continuously**” blacklisted IP addresses (or change their states only once) and “**discrete**” blacklisted IP addresses (those that changed state more than once). General statistics are given in Table 1: the second column shows counts of the total number of files containing blacklisted IP addresses for the whole experiment period; the third column shows the total number of distinct IP addresses; the fourth and fifth columns show the counts of the “continuous” and “discrete” IP addresses.

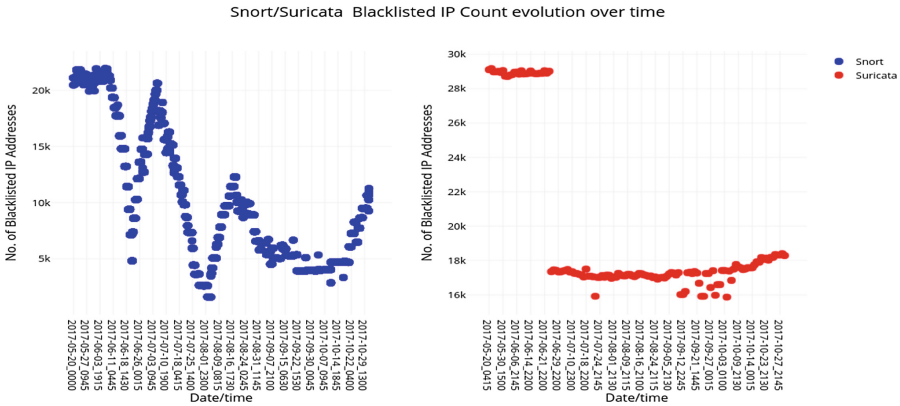


Fig. 1. Count of Blacklisted IPs in Snort and Suricata in our collection period

Table 1. General statistics of Blacklisted IP addresses

Blacklisted IP Source	Count of Files	Count of IP Addresses	Count of IPs that do not change state (“continuous”)	Count of IPs that change state (“discrete”)
Snort	15,812	46,701	5,383	41,318
Suricata	129	135,791	28,883	106,908

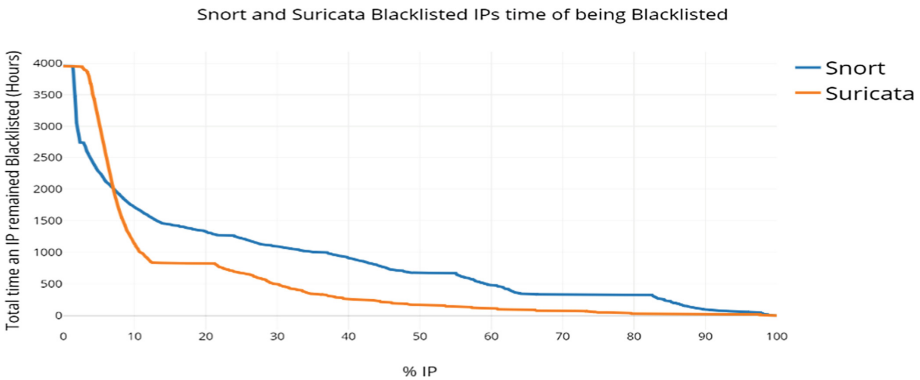


Fig. 2. Total time (hours) an IP remained Blacklisted

Figure 2 gives the distribution of total time blacklisted for all the IP addresses over the entire period of our experiment. We plotted the proportion of IP addresses (x-axis) against the total time a particular IP remained blacklisted (y-axis). We observe that IP addresses stayed blacklisted longer in Snort than in Suricata.

3.2 Diversity Analysis of the Blacklisted IP Addresses

We then analysed the similarity and diversity in the Snort and Suricata blacklisted IP addresses. We compared blacklisted IP addresses from the Snort and Suricata sources at exact time/date points (to the nearest second). In total, out of 15,812 Snort files, and 129 of Suricata, 128 files had a common date/time overlap. The analysis on this section is based on this overlap. Figure 3 shows the date/time slots for which the analysis was carried out (in the x-axis) and the counts of different categories of blacklisted IP addresses (y-axis). We have three main categories of interest: IP addresses which were blacklisted in Snort only (depicted as “_snort” in the graph), IP addresses which were blacklisted in Suricata only (“_suricata”), and IP addresses which were blacklisted in both Snort and Suricata (“_snort_suricata”). We observe that the overlap between the two blacklisted IP addresses sets is relatively small and the total number of IPs that appear in blacklists of both Snort and Suricata is relatively constant for the duration of our experiment.

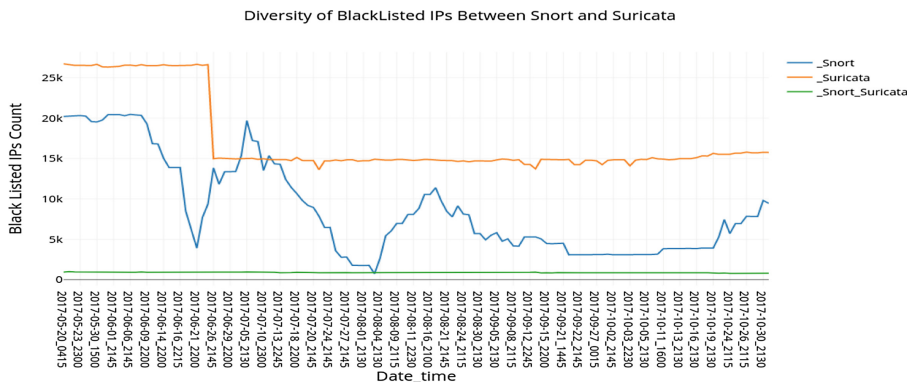


Fig. 3. Diversity in Blacklisted IPs as collected from Snort and Suricata sources

Table 2 shows the general statistics for all IPs and the data points in the dataset of 128 files of blacklisted IP addresses in Snort and Suricata. We have a total of 177,504 distinct IP addresses observed in either Snort or Suricata in these 128 files. Of these, 3,991 have been observed in both Snort and Suricata. We can think of each data point in our dataset consisting of an IP/date pair, and for each of these data points the value is either “observed in Snort-only” (abbreviated 01), “observed in Suricata only” (abbreviated 10), or “observed in both Snort and Suricata” (abbreviated 11). The statistics for these data points are given in the last three rows of Table 2. Table 3 then shows a more detailed breakdown for each of the 177,504 IP addresses. The first two columns show the totals count of IP addresses which in the observation period were observed in Snort only, Suricata only, or both in Snort and Suricata at the same time (these are depicted as “single state” IP addresses). The third and fourth columns show the total number of IP addresses in which we observed multiple states over the experiment period. For example, the first row shows that there are 79 IP addresses that were observed in both

Table 2. Statistics of the datapoints observed in Snort and Suricata overlapping periods

Total number of IPs in the 128 files of Snort	46,187	
Total number of IPs in the 128 files of Suricata	135,308	
Total number of IPs observed in either Snort or Suricata	177,504	
Total number of IPs observed in both Snort and Suricata	3,991	
Total number of data points (IP/date pairs) observed in Snort and Suricata overlapping periods.	Snort only (01)	1,129,180
	Suricata only (10)	2,219,330
	Snort and Suricata (11)	113,152

Table 3. Statistics of blacklisted IPs observed in Snort and Suricata overlapping periods

Single states	Count of IPs	Multiple states	Count of IPs	Observed in:	Count of IPs
Snort only (01)	42,196	(01,10) only	79	Snort (01)	35
				Suricata (10)	44
Suricata only (10)	131,317	(01,11) only	2,834	Snort (01)	1,257
				Both (11)	1,577
Both Snort and Suricata only (11)	588	(10,11) only	250	Snort (01)	84
				Both (11)	166
		(01,10,11) only	240	Snort (01)	102
				Both (11)	82
				Both (11)	56

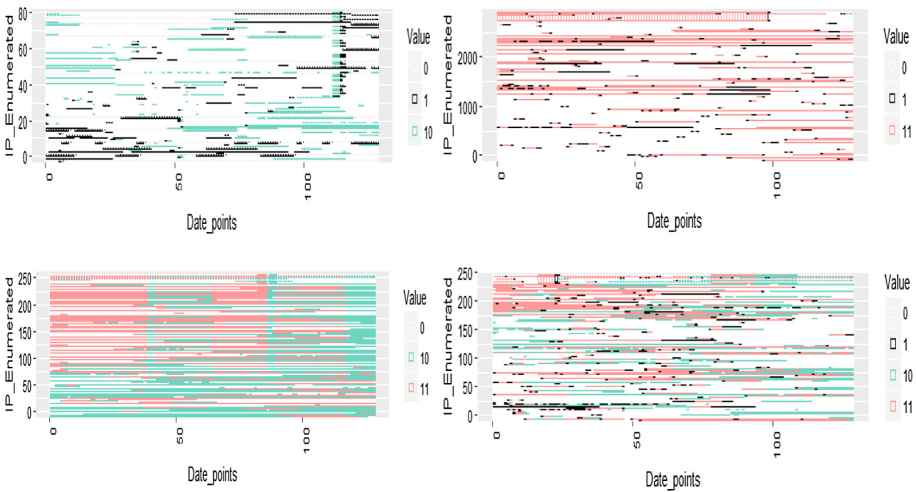


Fig. 4. Distribution of blacklisted IPs on which we observed multiple states: top-left (01, 10); top-right (01, 11); bottom-left (10, 11); bottom-right (01, 10, 11). **Note:** 0 (white) = no data. (Color figure online)

Snort and Suricata blacklisted files, but never at the same time. Columns 5 and 6 then show a further breakdown of these IP addresses depending on where they were observed first: for these 79 IP addresses, 35 were observed in Snort blacklists first, and 44 in Suricata.

Figure 4 shows the IP addresses for which we observed multiple states (i.e. those of columns three and four from Table 3). The x-axis shows the number of date/time points and the y-axis shows the enumeration of those blacklisted IP addresses. We kept the same ordering of the plots as the corresponding rows in column 3 of Table 3: the top-left plot shows the 79 IP addresses that were either observed in both Snort and Suricata but not at the same time, the top right the 2,834 observed in Snort only, or both Snort and Suricata at the same time etc.) The bottom-left plot shows an interesting behavior for IPs that are either in Suricata or in both. From time to time it appears many IP addresses are being removed from Snort, before being reinstated again (we can see blocks of red (Snort and Suricata) becoming green (Suricata only), and then red again).

4 Diversity in Rules Used by Snort and Suricata

4.1 Overall Analysis

In this section, we present results of the quantitative analysis of the diversity in Snort and Suricata rulesets. For this analysis, we collected rulesets of Snort and Suricata from 20 May 2017 to 31 October 2017. We considered the following Snort rulesets available from the Snort website: Community, Registered and Subscribed. For Suricata we used the Emerging Threats (ET) rulesets. Similar to blacklisted IP addresses, our sampling rate was every 15 min. However, the rate at which the rules were updated was much lower compared with blacklisted IP addresses: mainly every 24 h, but sometimes with lags of 5 days with no updates. Snort Community rules are an exception where we noticed an update of 4 rules multiple times a day. We present the analysis from comparing the rulesets across all versions once every 24 h.

Table 4 shows the details of the data that we used for this analysis. The total number of rules for Suricata is double that for Snort Registered and Snort Subscribed (which are very similar), while the total number of rules in Snort Community is much smaller. Additionally, we looked at how the rules change. We noticed that for some rules the SID (Signature ID) remains the same, but the version number of that rule may change: columns four and five of Table 3 give these counts. More than 80% of the Snort Registered and Subscribed rulesets, and 97% of Suricata ET rulesets reported version changes during the experiment.

Table 4. General statistics of different rule sets

Rule Set	Number of Files	Number of Rules	Rules with no version changes during the experiment	Rules with versions changes during the experiment
Snort Reg	52	10,675	2,259	8,416
Snort Sub	51	10,736	2,399	8,337
Snort Com	166	903	472	431
SuricataET	106	19,584	523	19,061

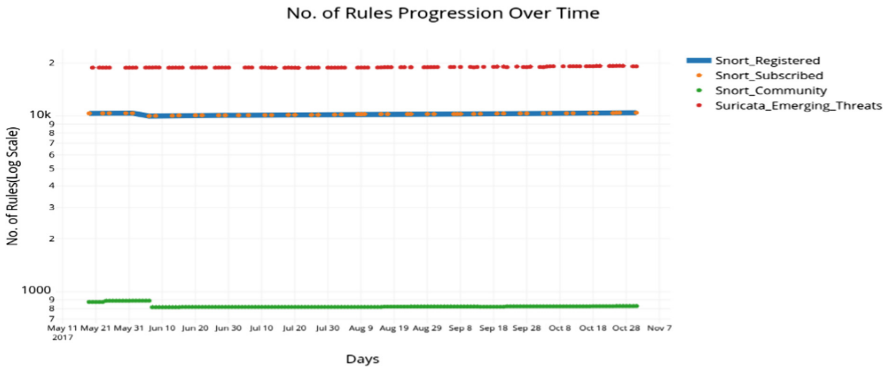


Fig. 5. Snort and Suricata rule counts over the duration of the experiment

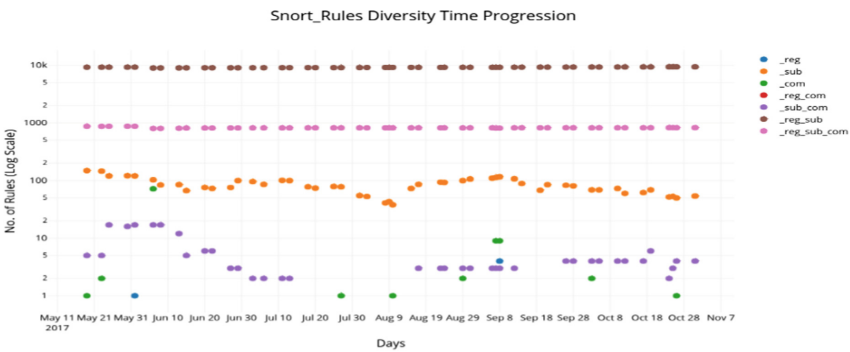


Fig. 6. Time progression of diversity in Snort rules (Color figure online)

Figure 5 shows the count of rules of each of these rulesets as they evolve over the duration of the experiment. We notice that the total number of rules in each set remains relatively constant for the duration of the experiment.

4.2 Snort Diversity Analysis

Next, we look at a comparison of the rulesets of Snort. The SID along with the version number is a unique identifier for each rule, and they are used consistently across the different rulesets (i.e. the same SID and same version number in Registered and Subscribed means that the rule is also the same). Figure 6 shows the diversity in time among the Snort rulesets. The y-axis shows, in a log scale, the counts of rules in different categories for each day of the experiment (x-axis). “_reg” is the count of rules which are only in the Snort Registered set, “_reg_com” shows only those rules that in the Registered and Community rulesets etc. We notice that the majority of the rules are those that exist in both Registered and Subscribed rulesets (brown dots), followed by those that are common amongst all three rulesets (pink dots), and those that exist in the Subscribed ruleset only (orange dots).

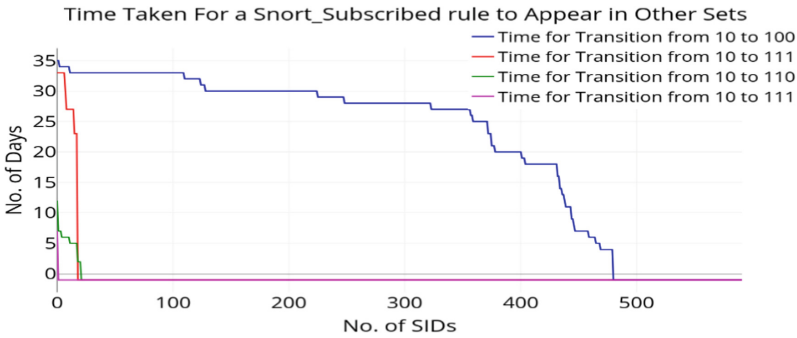


Fig. 7. The time lag for Subscribed rules to appear in the other Snort rulesets

4.3 Diversity Analysis of Snort and Suricata Rules

Suricata ET rules use different SIDs to Snort, so the comparison of Snort and Suricata rules was done using the “content” field in the rules. This field contains the “signature” of the malicious payload of a packet that is inspected by the IDS. Hence, the ‘content’ field represents the important signatures information for a malicious traffic that these IDSs are intended to detect/capture. Not all Snort and Suricata rules have the ‘content’ field so the analysis in this section is based on only those rules that have it (73.4% of the rules of Snort Registered and Subscribed have this field, 77.8% of Suricata ET and 97.7% of Snort Community rules have the “content” field).

Figure 8 shows the diversity of Snort and Suricata rulesets based on the content field. Here, the x-axis shows the days and the y-axis the number of SIDs with content fields, in log scale. The shortcut notation is the same as previous (e.g., “_ET” represents the SIDs observed only in the Suricata ET ruleset etc.) The largest overlap between Suricata and Snort is in the rules that exist in ET, Registered and Subscribed rulesets (the magenta dotted line that hovers around the 100 mark in the y-axis).

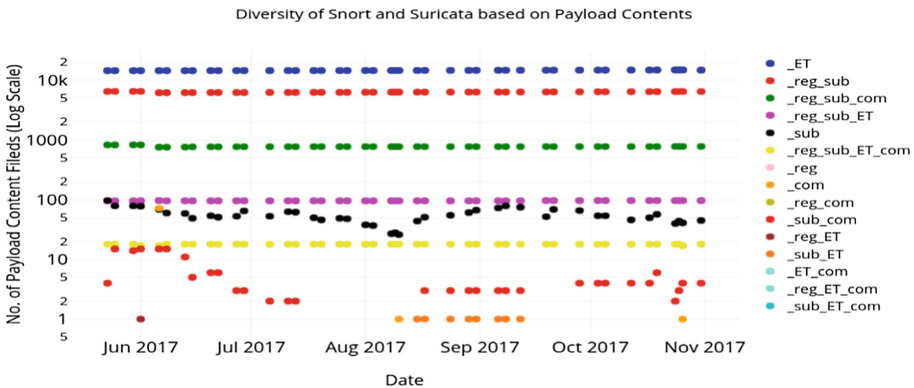


Fig. 8. Diversity in time of the Snort and Suricata rulesets

5 Diversity in the Behavior of Snort and Suricata

So far, we looked at the diversity that exists in the internals of these products and the way in which they evolve. In this section, we will analyse how this diversity in design manifests itself in the alerting behavior of these products when analyzing network traffic. We analysed 7 days of pcap data from 2 May to 8 May 2017. The data was captured in the DMZ network of the City, University of London. In those 7 days, we had 326 GB, 330 GB, 280 GB, 252 GB, 186 GB, 204 GB and 316 GB of network data respectively. The breakdown of the traffic based on different types of protocols is listed in Fig. 9.

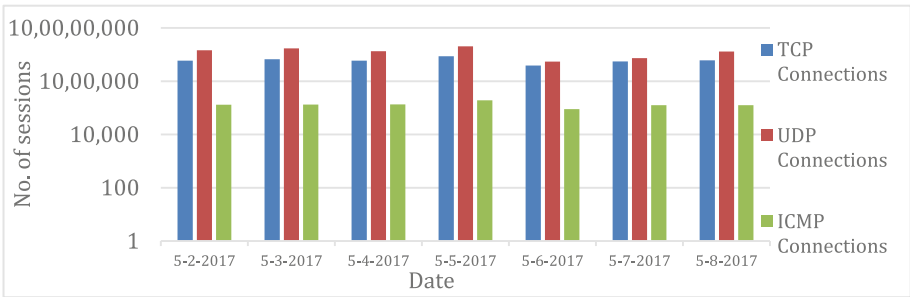


Fig. 9. Count of sessions per protocol for each day of experiment

We analysed this data using Snort and Suricata with the rulesets discussed so far (we used one snapshot of the ruleset for the analysis). Figure 10 presents the results. We used the same notations as in Sects. 3 and 4 (e.g. “_et” means alerted by Suricata ET only). We notice that Snort Registered and Subscribed rules generated alerts of an order of magnitude more than Suricata ET. As observed in the ruleset and black-listed IP addresses analysis from Sects. 3 and 4, there is little overlap in the alerts of Suricata ET and Snort, which means these systems exhibit very diverse alerting behavior when analysing this traffic.

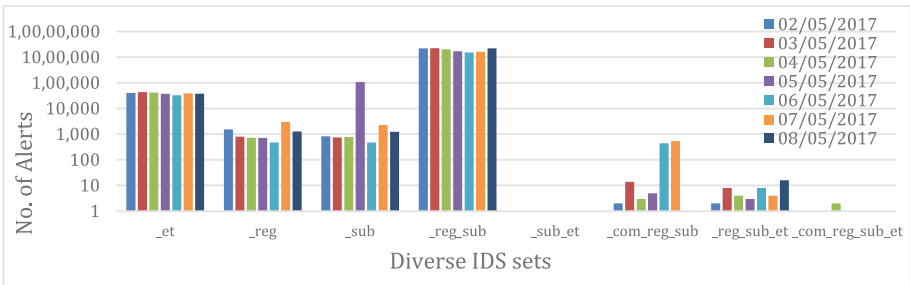


Fig. 10. Number of alerts generated by different combinations of rulesets of Snort and Suricata

6 Discussion and Limitations

The results are intriguing, and they show that there is a large amount of diversity in the rules and blacklists of Snort and Suricata. Whether this diversity is helpful or harmful for a given deployment depends on the context. The rules and blacklists alert for potentially harmful behavior that has been observed somewhere in the world by users of these products. In a different deployment, the alerts from some of these rules may not cause harm. For example, a service or port for which a rule alerts may not exist in that environment. Hence even if the alerts are for malicious traffic it is likely that this attack will not cause any harm in the systems of that deployment. The dataset we used in Sect. 5, real traffic that the University's IT team gave us access to, is unfortunately not labelled, so we cannot do a conventional analysis of sensitivity and specificity of these IDSs and their diverse combinations. We did share the findings with the University's IT team and they found the results interesting. Currently they use a smaller subset of Suricata ruleset for analysis. Interestingly, they mentioned that even if the alerts are for services that they do not run (hence would be harmless in their environment) they would like to know about them as it gives them insight on security exposure for services that users may request in the future, and also because they can use the alerts to check if they are precursors for attacks on other services that they value.

How can individual user organizations decide whether diversity is a suitable option for them, with their specific requirements and usage profiles? The cost is reasonably easy to assess: costs of the software products, the required middleware (if any), added complexity of management, hardware costs, run-time costs and possibly more complex diagnosis and more laborious alert sifting. The gains in improved security (from protection to attacks and exploits) are difficult to predict except empirically. This uncertainty will be compounded, for many user organizations, by the lack of trustworthy estimates of their baseline security. We note that, for some users, the evidence we have presented would already indicate that diversity to be a reasonable and relatively cheap precautionary choice, even without predictions of its effects. These are users who have serious concerns about security (e.g., high costs for interruptions of service or undetected exploits), and sufficient extra personnel to deal with a larger number of alerts.

7 Related Work

The security community is well aware of diversity as potentially valuable [1–3]. Discussion papers argue the general desirability of diversity among network elements, like communication media, network protocols, operating systems etc. Research projects studied distributed systems using diverse off-the-shelf products for intrusion tolerance (e.g. the U.S. projects Cactus [4], HACQIT [5] and SITAR⁶; the EU MAFTIA project⁷), but only sparse research exists on how to choose diverse defenses (some examples in [3, 6–8]).

⁶ <http://people.ee.duke.edu/~kst/sitar.html>.

⁷ <http://research.cs.ncl.ac.uk/cabernet/www.laas.research.ec.org/maftia/>.

A very extensive survey on evaluation of intrusion detection systems is presented in [9]. This survey analyses and systematizes a vast number of research works on the field. The main features analyzed in the survey are the workloads used to test the IDSs, the metrics utilised for the evaluation of the collected experimental data, and the used measurement methodology. The survey demonstrates that IDS evaluation is a key research topic and that one of the main benefits that IDSs evaluation can bring are related with guidelines on how to improve IDS technologies.

8 Conclusions

In this paper, we presented an analysis of the diversity that exists between the Snort and Suricata rules and blacklisted IP addresses. We analysed the evolution of the rulesets and blacklisted IP addresses of these two IDSs over a 5-month period between May and October 2017. We used three different off-the-shelf default configurations of the Snort IDS and the Emerging Threats configuration of the Suricata IDS. We performed the analysis to provide insight to Security architects on how they can combine and layer these systems in a defence-in-depth deployment. We also showed results on the observed diversity in behaviour of these systems, when they analysed the network data of the DMZ network of City, University of London.

The main conclusions from our analysis are:

- There is a significant amount of diversity in the blacklists of Snort and Suricata, and this is maintained throughout our observation period. The amount of overlap between these IPs is relatively small. Depending on the adjudication mechanism that a system architect wishes to deploy, having access to a larger pool of blacklisted IP addresses may be beneficial to increase protection against a larger pool of malicious sources. However, if a user observes a large number of false positives from these blacklists at a given period of time, then diversity can be help to keep the false positive rate low (for example by only raising alarms only if an IP appears in multiple blacklist) until the vendors “clean up” the blacklists;
- We observe a significant amount of diversity in the rules of Snort and Suricata. When analyzing the rules based on the “content” field, only 1% of the rules of Snort and Suricata return a match. This indicates that these systems would alert on potentially very diverse traffic. This is indeed confirmed from a small experiment that we ran with real traffic from City, University of London. There was very little overlap in the alerting behavior of these products.

We have underscored that these results are only *prima facie* evidence for the usefulness of diversity. What is important is to assess these products in real deployment on their capability to improve the security of a given system. The results presented here will, we hope, provide the security architects with the evidence on the diversity that exists in the design of these products and whether this diversity remains as these products evolve.

As further work, we plan to investigate the diversity with IDSs and other defence-in-depth tools in real deployments, with labelled datasets, to assess the benefits as well as potential harm that diversity may bring due to the interplay between the risks from

false negatives and false positives. Currently we are investigating the adjudication mechanisms that can help balance the risks associated with these failures.

Acknowledgment. This work was supported by the UK EPSRC project D3S and in part by the EU H2020 framework DiSIEM project.

References

1. Elia, I.A., Fonseca, J., Vieira, M.: Comparing SQL injection detection tools using attack injection: an experimental study. In: 2010 IEEE 21st International Symposium on Software Reliability Engineering (2010)
2. Littlewood, B., Strigini, L.: Redundancy and diversity in security. In: Samarati, P., Ryan, P., Gollmann, D., Molva, R. (eds.) ESORICS 2004. LNCS, vol. 3193, pp. 423–438. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30108-0_26
3. Garcia, M., et al.: Analysis of operating system diversity for intrusion tolerance. *Softw.: Pract. Exp.* **44**(6), 735–770 (2014)
4. Hiltunen, M.A., et al.: Survivability through customization and adaptability: the Cactus approach. In: Proceedings of DARPA Information Survivability Conference and Exposition, DISCEX 2000 (2000)
5. Reynolds, J., et al.: The design and implementation of an intrusion tolerant system. In: International Conference on Dependable Systems and Networks, DSN 2002, Washington, D.C., USA (2002)
6. Sanders, W.H., et al.: Probabilistic validation of intrusion tolerance. In: International Conference on Dependable Systems and Networks, Fast Abstracts Supplement, DSN 2002, Bethesda, Maryland (2002)
7. Gupta, V., Lam, V., Ramasamy, H.V., Sanders, W.H., Singh, S.: Dependability and performance evaluation of intrusion-tolerant server architectures. In: de Lemos, R., Weber, T. S., Camargo, J.B. (eds.) LADC 2003. LNCS, vol. 2847, pp. 81–101. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45214-0_9
8. Bishop, et al.: Diversity for security: a study with off-the-shelf antivirus engines. In: 22nd IEEE International Symposium on Software Reliability Engineering (ISSRE 2011) (2011)
9. Milenkoski, A., et al.: Evaluating computer intrusion detection systems: a survey of common practices. *ACM Comput. Surv.* **48**(1), 12:1–12:41 (2015)



Inter-device Sensor-Fusion for Action Authorization on Industrial Mobile Robots

Sarah Haas¹(✉), Andrea Höller¹, Thomas Ulz², and Christian Steger²

¹ Development Center Graz, Infineon Technologies Austria AG, Graz, Austria
{sarah.haas, andrea.hoeller}@infineon.com

² Institute for Technical Informatics, Graz University of Technology, Graz, Austria
{thomas.ulz, steger}@tugraz.at

Abstract. Usage of mobile robots in industry increased significantly in recent years. However, mobile robots introduce additional safety issues for human workforce and pose a higher risk of failures in production due to possible abnormal robot behavior. Such abnormal behavior could, among other things, be caused by security weaknesses that entail attacks. These problems lead to a need for action authorization mechanisms to protect humans and mitigate possible costly failures. In this paper, we propose an authorization mechanism for critical actuator actions on industrial mobile robots. The mechanism relies on security principles that prevent adversaries from unauthorized action execution. To the best knowledge of the authors, no similar concept for secured action authorization for industrial mobile robots is currently known in research. Our evaluation shows more than 80% of additional safety hazard causes introduced by the lack of security can be mitigated with the proposed authorization mechanism.

1 Introduction

The increased automation in production facilities entails a rapid rise of mobile robots in industrial applications. The number of mobile robots in industrial automation will be even higher in future *Smart Factories* [27] and Industry4.0 environments. Mobile robots in future production facilities will typically interact with machines, humans, or other robots to fulfill any given task such as fetching material or delivering material. A research testbed for smart factory related technologies is the RoboCup Logistics League [21]. A mobile robot in this testbed basically consists of actuators, sensors, a central computing unit, and a communication unit. The actuators are used to interact with the physical world. The sensors are used to gain information about the environment. The central computing unit processes incoming and outgoing data, coordinates the components, and instructs them. The communication unit acts as router that connects all components on a mobile robot, and provides technologies to communicate to other devices via, e.g. Industrial WiFi.

The advantage of using a router in a robotic system also entails the advantage of modularity and easy connection of components. However, a disadvantage is that it would be easy for an adversary to access sensors and actuators by simply sending commands via the router, or even hijack a robot. It is generally easier to attack devices that communicate wirelessly. Especially Industry4.0 environments will deal with larger attack surface due to the increased connectivity, usage of wireless communication, increased data exchange, and many more [11]. Besides that, the safety of human workforce cannot be ensured anymore if an adversary is able to hack a robot. To support the safety of human workforce, security needs to be introduced [4, 8, 15]. Since the interaction between mobile robots and machines is a typical use case, and will occur in close proximity to human workforce, secured authorization of actuator actions needs to be performed. Actuator actions might include manipulations of production material with a robotic arm, or movement of production material between machine and mobile robot. If such actions can be initiated or manipulated by an adversary, human workforce will be exposed to serious safety hazards.

To prevent such malicious manipulations, we propose a secured authorization mechanism for critical actions on industrial mobile robots when interacting with other entities. The authorization approach uses available sensors to gain information about the current environmental states such as the distance between a mobile robot and a machine. This information is combined with security mechanisms to prevent adversaries from injecting or manipulating false action commands, and also prevent the mobile robot from executing harmful actions caused by software bugs or other errors.

To the best knowledge of the authors, no concept similar to the proposed one is currently known in research. The related work shows several applications for multi-sensor fusion and authorization concepts. However, none have combined these concepts before. Furthermore, none of the concepts take security into account even though the safety could easily be compromised by security weaknesses in many of these scenarios.

To summarize, the contributions of this paper are:

- The first action authorization approach for industrial mobile robots that relies on inter-device sensor-fusion and cryptographic principles to support the safety of human workforce.
- A combined safety and security analysis showing that more than 80% of the additional safety hazard causes can be mitigated by introducing the proposed authorization approach.

2 Related Work and Background

Secure Element

A secure element (SE) stores confidential data such as key material and is able to perform cryptographic operations such as signature computation or hash computation. General purpose microcontrollers or CPUs are typically prone to side channel attacks that spy on calculation times depending on the input or try to

physically manipulate the CPU to compromise calculations that might reveal confidential data. SEs, in contrast, are tamper-resistant which means that they are built to withstand such attacks and are, therefore, used for security critical applications such as bank cards or trusted computing in, e.g. Trusted Platform Modules [1].

One-Time Passwords

One-Time Passwords (OTPs) are password schemes where a password becomes invalid after its first use and were introduced by Lamport [14]. Lamport intended to overcome issues with plain text passwords such as interception of plain text passwords by adversaries and later, as a countermeasure against replay attacks. Such attacks capture a user's login credentials and use them to access a system [10]. OTPs use non-invertible cryptographic hash functions such as SHA-256 to create passwords. It is necessary that the same non-invertible cryptographic hash function is available on the client and the host for OTP generation and verification.

In 2005, M'Raihi et al. [19] proposed OTPs based *Hashed Message Authentication Codes* (HMAC) called *HOTP*. The authors introduced a counter that is combined with the secret key and forwarded to the hash function. The counter is synchronized with a trusted entity such as a server to enable the verification of the OTP. The counter is incremented by a specific amount known by the client and server every time an OTP is generated or validated. The counter enables individual passwords for unchanged data, and the secret key enables the authenticity of the client and server.

The tickets used for the authorization approach are based on HOTPs that provide integrity and authenticity. HOTPs use a secret key and moving factors such as a counter value to prevent the possibility of replay attacks. The secret key is used to protect the hashes from brute-force attacks on the counter value, and enables authenticity. The counter is necessary since the data exchanged between robot and machine might be identical. The same data would always result in the same valid hash value. If an adversary would capture a valid ticket sent by the robot, he could send it to the machine over and over again, and the machine would authorize the actions. To generate different tickets for identical data, the counter values are used to generate passwords that are only valid once. One might also use signatures instead of HOTPs, however, the problem of the exact same signature for equal input data would remain the same, and the operation would also require some kind of moving value for individual signature values. Furthermore, asymmetric cryptographic calculations such as signatures are much slower than symmetric cryptographic calculations such as HMACs.

Multi-sensor Fusion

Multi-sensor fusion is used to combine the data provided by several different sensors or other data sources to improve accuracies and robustness [9]. The concept of multi-sensor fusion has been used for years in a wide range of areas including artificial intelligence, medical diagnostics, environmental monitoring, robotics, and much more. Especially mobile robots strongly rely on multi-sensor

fusion since they deal with localization problems, odometry inaccuracies, and other many other problems [13, 22].

Kam et al. [12] reviewed existing sensor fusion techniques for robot navigation back in 1997, especially addressing self-localization in maps constructed by the robot. Recent research by Lynen et al. [16] also addressed multi-sensor fusion for navigation and self-localization purposes in a framework that is able to process any absolute, relative, or delayed data from an almost unlimited amount of sensors. Even though, sensor fusion is widely used in robotic applications, it was, as far as we know, never used for authorization mechanisms before.

Authorization for Mobile Robots

Authorization was defined as granting privileges to processes or users by Fraser in 1997 [6] and is used widely in any operating system, company network or production system. Current research focuses on topics such as authorization and access rights in cloud environments [25] or Internet of Things (IoT) systems [5]. In the mobile robotics domain, authorization is not a key topic. A very simple authorization mechanism was shown by Gonçalves et al. [7]. The authors proposed a realistic sensor and actuator model for wheeled mobile robot simulations that included a boolean register whose value was checked before executing an action on a robotic arm. As far as we know, the only approach that includes authorization related to robotics and other mobile devices was proposed by Popovici et al. in 2003 [23]. The authors proposed a middleware platform for mobile devices that uses an authorization mechanism to prevent unauthorized entities from executing actions on, e.g., a robotic arm. This middleware checks an entities' rights to execute actions in a physical system but does not include any current environmental information.

To the best knowledge of the authors, none of the existing approaches use sensor-fusion or security measures in their authorization approaches, or would even combine these topics.

3 Proposed Authorization Mechanism

The interaction between robots and machines is a typical scenario that will occur in smart factories. If interactions between these entities is unauthorized, serious safety hazards for humans can occur, and production material could be damaged. Therefore, the approach proposed in this paper assures that only authorized actions are executed by actuators to support safety. The proposed mechanism uses the sensor data of both, robot and machine, to make sure that the robot is authorized to, for example, drop off production material on a machine. Using the combined sensor data instead of just the sensor data from the robot can prevent critical actions from being performed in case of errors on the robot or malicious manipulations, to protect human workforce and production material. The sensor data of the robot and a machine are combined and checked. If for example, the laser scanner values of the robot lie within a certain range while approaching a machine, the machine is notified. The machine would then check the light barrier on its input. If the light barrier is interrupted, the machine notifies the

robot. The robot could then generate an authorization ticket and send it to the actuator. The actuator executes the command if the authorization ticket is valid. Each authorization ticket expires after it was used to overcome issues with replay attacks. The sensors in this paper are assumed to be trustworthy since this topic would exceed the scope of this paper, and other researchers already focus on sensor trustworthiness [18,24,26]. The required components to successfully execute an action are the central computing units (CCU) of both robot and machine, the robot's SE, the actuator's microcontroller, the actuator's SE, and the machine's SE. To perform action authorization, the following preconditions need to be fulfilled. (1) Robot, actuators and machine, are equipped with a SEs to store key material and securely compute and verify HOTPs. (2) Robot's and actuator's SEs share a secret key K_A and a counter cnt_A . (3) Machine's and robot's SE share a secret key K_M and a counter cnt_M . (4) Robot's SE and the sensors share secret keys K_S and counters cnt_S . (5) All secret keys and counter values are already stored in the corresponding SEs or sensors.

3.1 Authorization Approach

The authorization approach is divided into 15 steps that can be seen in Table 1. The following section describes the authorization process in detail. In the text, the numbers in brackets refer to the line numbers in Table 1. Each instruction colored in red means that the calculation is done in an SE. The function $HOTP_G$ refers to the generation of an HOTP on a SE and also increments the counter values cnt_M , cnt_R or cnt_S . The $HOTP_G$ function requires a secret key, a counter and some data as inputs. The function $HOTP_V$ refers to a validation of an HOTP in an SE and also increments the counter values cnt_M , cnt_R or cnt_S . The validation of an HOTP can be done if the secret key, counter and data, as well as the HOTP to validate against, are provided as input. The secret key, counter and data must match the values used to generate the initial HOTP. Otherwise, the HOTP will not be valid. The *inRange* function checks whether the sensor data fulfills pre-defined conditions on the robot. The *fuse* function fuses the sensor readings of the robot and machine, and checks if the pre-defined conditions for the robot's and machine's sensor readings are fulfilled. The function *ReqSenData* requests the sensor data from one or more sensors.

The authorization is initiated by the robot's CCU. The robot's CCU requests data from any sensor, e.g. the laser scanner to compute the distance from the next obstacle or the distance to equipment on the production floor. The sensor generates an HOTP over the data using a counter value and secret key, and provides the HOTP and data sd_R to the robot's CCU (1). The sensor's HOTP is validated by the robot's SE (2). The robot's CCU checks if the sensor data satisfies certain pre-defined conditions (3). A possible condition would be the distance measured by a laser scanner. If the distance is within a specific range, the condition is satisfied. If the sensor data's HOTP was valid, and the sensor data was in a certain range, the command cmd and sensor data sd_R are sent to the robot's SE. The received data is combined with the secret key K_M and counter cnt_M , and the request ticket $hotp$ is generated (4). The command cmd , sensor data sd_R and hash $hotp$ are sent to the machine (5). The machine's CCU

Table 1. Sequence diagram of a complete authorization process.

Machine	Robot	Actuator
1 :	$sd_R \leftarrow ReqSenData()$	
2 :	$v \leftarrow HOTP_V(sd_R)$ $R \leftarrow K_M, cnt_M, cmd, sd_R$ $in \leftarrow inRange(sd_R)$	
3 :	if v AND in	
4 :	$hotp \leftarrow HOTP_G(R)$	
5 :	$Send\ hotp, cmd, sd_R$ \longleftarrow	
6 :	$sd_M \leftarrow ReqSenData()$	
7 :	$v_1 \leftarrow HOTP_V(hotp)$	
8 :	$v_2 \leftarrow HOTP_V(sd_M)$ $M \leftarrow K_M, cnt_M, hotp$ $in \leftarrow fuse(sd_M, sd_R)$	
9 :	if v_1 AND v_2 AND in	
10 :	$auth \leftarrow HOTP_G(M)$	
11 :	$Send\ auth$ \longrightarrow	
12 :	if $HOTP_V(auth)$ $R \leftarrow K_A, cnt_A, cmd$	
13 :	$act \leftarrow HOTP_G(R)$	
14 :	$Send\ act, cmd$ \longrightarrow	
15 :		if $HOTP_V(act)$ $execute(cmd)$

requests the sensor data sd_M (6), and instructs the SE to validate both, the request ticket $hotp$ and the sensor data's hash (7, 8). The sensor data of both robot and machine are passed to the $fuse$ function to perform the sensor fusion, and necessary checks on the sensor readings. The $fuse$ function simply returns *true* if the data is valid or *false* if the was invalid (8). If both hashes $hotp$ and sd_M are valid, and the requested sensor data fulfilled the preconditions (9), the response ticket $auth$ is generated by the SE (10). The response ticket is sent to the robot (11) and the robot's SE validates the received response ticket (12). If the ticket was valid, the robot's SE generates an authorization ticket act for the actuator (13). The authorization ticket act and command cmd are sent to the actuator (14) and if the actuator's SE confirms the validity of the received authorization ticket act , the command cmd is executed (15).

4 Implementation Remarks

This section discusses a proof-of-concept implementation of our proposed authorization approach, and includes explanations regarding the sensors and SEs.

4.1 Proof-of-Concept Implementation

As a proof-of-concept, our proposed authorization approach was implemented using several Raspberry Pi 3 and SEs by Infineon Technologies. Figure 1 shows the setup of the implementation. The sensors are simple simulations but calculate HOTPs for their measurements. The Raspberry representing the machine and the Raspberry representing the CCU are both equipped with SEs, and communicate via a router with WiFi. The Raspberry representing the actuator is also equipped with an SE, and is connected via a router with Ethernet to the Raspberry representing the CCU.

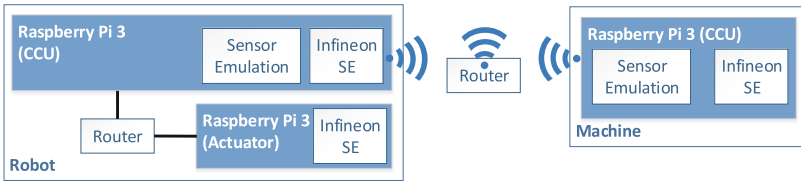


Fig. 1. Setup for the proof-of-concept with several Raspberry Pi 3 and SEs.

The proof-of-concept shows that the introduction of SEs in this scenario causes a significant increase of the overall runtime of one authorization. The mechanisms that provide tamper-resistance to such devices cause the overhead. However, in industrial use cases, the protection of key material and other confidential information, and the introduction of security is of utmost importance to support safety of human workforce and prevent damage on production material. Since mobile robots might perform real-time operations, the increased latency caused by the introduced security measures need to be taken into account when defining any real-time condition. However, since our proposed authorization mechanism is deterministic, the additional overhead can be calculated in advance. The overhead highly depends on the used hard- and software which makes it very hard to give a generally accepted assumption about the overhead. The overhead analysis would exceed the scope of this paper since the focus of this paper is to support the safety of human workforce by introducing security measures. The overhead analysis as well as possible optimizations to reduce the overhead are postponed to future work. One possibility for latency reduction would be to already start the authorization process while the robot approaches the machine, meaning that the steps taken on the robot before communicating to the machine can be done in parallel to the actual task of the robot. Another

possibility would be the direct forwarding of the authorization ticket from the machine to the actuator. Furthermore, the machine could receive sensor readings periodically, meaning it does not need to actively request them. These optimizations would already reduce the latency significantly.

4.2 Application of SEs in the Authorization Approach

The SEs in this scenario are, in principle, optional since each HOTP computation could also be done by a general purpose CPUs in plain software. However, if an attacker is able to have physical access to a general purpose CPU, he could perform side-channel attacks that reveal the secret key used for HOTP generation. The SEs are used to increase security by protecting the key material, counter and also the password generation process from such side-channel attacks. Physical attacks tend to reveal confidential information by analyzing the devices interface using, e.g. power or time analysis attacks, or try to physically attack the device by bombarding them with a laser to generate an error. Remote attacks tend to reveal data stored in software such as files or folders, or try to hijack a device. SEs cannot protect a device from remote attacks but can protect the secret keys used to perform cryptographic operations since the secret key cannot be read from the device. The question whether an SE is necessary highly depends on the actual use case and desired security level. Furthermore, SEs introduce a trade-off between latency and security level since SEs tend to be much slower than general purpose CPUs due to the implemented protection mechanisms against side-channel attacks.

5 Combined Safety and Security Analysis

To highlight the safety enhancing security features of our proposed approach, a combined safety and security analysis is conducted. The analysis uses the basic idea of a combined safety and security as suggested by Macher et al. [17] for the automotive domain. However, the functional safety analysis is adopted to match the ANSI/RIA R15.06 norm [2] for industrial robot safety. For the security analysis, a threat analysis [20] including countermeasures enabled by the proposed authorization approach is executed. The threats are then categorized similar to the risk level determination in the ANSI/RIA R15.06. The combination of the safety and security analysis show that additional causes for the existing safety threats arise from a lack of security. However, the analysis also shows that the proposed authorization mechanism reduces the additional safety hazard causes significantly. Before the analysis is performed, assumptions on the attacker and attack possibilities are made. (1) Taking over a robot is assumed to be possible since the equipped wireless communication technology opens a wider attack surface, and the attacker can directly attack the robot without the need to infiltrate the company network. (2) Taking over a machine remotely is assumed to be not attractive for an attacker since machines are connected by wire to the factory network, meaning that an attacker would have to gain access to the network,

and then would need to make it to the machine. It is assumed to be too much of an effort since an attacker would attack a more attractive target such as a central server rather than one specific machine. (3) An attacker trying to locally extract the secret keys from machine or robot is assumed to be possible since an attacker would then be able to read and send commands to devices and into the network using valid secret keys. Extracting the keys would not require to take over a machine or robot since it can be done with different side-channel attacks.

5.1 Methodology

This section shows how identified threats are categorized similar to the ANSI/RIA R15.06 risk assessment [2] to show the severity of the identified threats.

Table 2. Classification of required resources *RR* to execute a threat.

Level	Resource	Example
RR0	No tool	Manipulation of sensors with Mirror, Laser Pointer, Tape, etc.
RR1	Simple tool	Laptop, Smart Phone for network access, etc.
RR2	Standard tool	Network Sniffer, Oscilloscope for message capturing, power or time analysis attacks, etc.
RR3	Advanced tool	MITM tools, tools for targeting attacks for manipulation of messages, advanced physical attacks, etc.

Table 3. Classification of required know-how *RK* to execute a threat.

Level	Know-How	Example
RK0	Basic	Functionality of sensors, accessing of networks, use of physical interfaces, etc.
RK1	Advanced	Basic physical/remote Attack know-how, network know-how, e.g. protocols, power analysis etc.
RK2	Expert or Insider	Advanced remote/physical attack Know-how, e.g. targeting attacks, MITM, etc.

Table 4. Classification of required accessibility *RA* to execute a threat.

Level	Access	Example
RA0	Remote	Other country, outside facility, etc.
RA1	Local	Physical presence at attacked target

The threat level determination based on the risk assessment determination described in the ANSI/RIA 15.06 [2] utilizes the categories required resource *RR*, required know-how *RK* and required accessibility *RA*. The *RR* gives examples of the required tools to successfully deploy the security threat (see Table 2).

The *RK* defines the necessary know-how an attacker has to have to successfully execute the attack (see Table 3). The *RA* defines if an attack can be launched remotely, or if an attacker needs to be physically present to execute an attack (see Table 4). These categories are used to identify the severity of a threat and determine its threat level as shown in Table 5. The higher the threat level, the more severe a threat is when exploited in a system.

Table 5. Threat level determination matrix depending on the required resource, know-how and access based on the ANSI/RIA R15.06 risk level determination matrix.

Required Resource	Required Know-How	Required Accessibility	Threat Level				
			negligible=0	low=1	medium=2	high=3	very high=4
RR0	RK0	RA2	0				
RR0	RK1	RA2		1			
RR0	RK2	RA2			2		
RR1	RK0	RA1/RA2		1			
RR1	RK1	RA1/RA2			2		
RR1	RK2	RA1/RA2			2		
RR2	RK0	RA1/RA2		1			
RR2	RK1	RA1			2		
RR2	RK1	RA2				3	
RR2	RK2	RA1				3	
RR2	RK2	RA2					4
RR3	RK0	RA1/RA2			2		
RR3	RK1	RA1				3	
RR3	RK1	RA2					4
RR3	RK2	RA1/RA2					4

5.2 Threat Analysis and Threat Level Determination

To apply the defined methodology on our proposed authorization approach, we perform a threat analysis according to Myagmar et al. [20], and determine the threat level of each threat. Table 7 lists the identified threats *T*, countermeasures *C* and remaining residual risks *R*. Table 6 shows the determination of the threat level for each identified threat.

5.3 Results

For a mobile robot with a robotic arm, the safety analysis identified a total of 27 hazards with 38 safety hazard causes. The safety hazards are inspired by Bartos [3] and the ANSI/RIA 15.06 norm [2]. The safety hazards for the machine are not listed separately since they are a subset of the safety hazards identified for the mobile robot. Since security threats can also cause safety hazards, each threat was applied to the safety hazard scenarios to check whether the security threat could cause a safety hazard.

Table 6. Threat level determination for the identified security threats.

Threat	Required Resource	Required Know-How	Required Accessibility	Threat Level
T1	RR2	RK1	RA1	2
T2	RR2	RK1	RA2	3
T3	RR1	RK1	RA1	2
T4	RR3	RK2	RA1	4
T5	RR1	RK1	RA1	2
T6	RR0	RK1	RA2	1
T7	RR1	RK0	RA1	1
T8	RR2	RK1	RA1	2
T9	RR3	RK2	RA1	4
T10	RR3	RK1	RA1	3
T11	RR1	RK0	RA1	1

For the proposed authorization approach, the combined analysis shows that 10 of the 27 identified hazards could also be caused by the identified security threats from Table 7. The left hand side of Table 8 shows the safety analysis according to the ANSI/RIA R15.06 risk level assessment for all safety hazards that can also be caused by security threats. The right hand side of Table 8 shows the corresponding security threats that can cause the safety hazard. To identify which safety hazard can be caused by malicious actions of an attacker, each security threat listed in Table 7 was applied to each safety hazard. As the analysis shows, 10 safety hazards can also be caused when security threats are exploited. As an example, safety hazard #13 where the laser scanner is blinded, can intentionally be caused by the two security threats *T7* where an attacker would physically manipulate the sensor using e.g. tape, and *T11* where the attacker would perform a DoS attack to prevent the laser scanner from sensor readings by flooding it with messages. The other listed security threats cannot cause this safety hazard.

The combined safety and security analysis shows that for 10 safety hazards a total of 43 additional causes due to the lack of security can be identified for an insecure authorization scenario. The bar chart in Fig. 2 shows the number of total additional safety causes and the number of mitigated causes when applying the proposed secured action authorization for each risk level. The bar on the left hand side in blue shows the number of total additional safety hazard causes for each risk level, and the bar on the right hand side in green shows the mitigated causes.

The threat analysis lists countermeasures enabled by our proposed security-enhanced action authorization approach. The introduced security measures reduce the number of additional safety hazard causes from 43 to 7 for the proposed authorization approach. 36 additional safety hazard causes can be mitigated by our proposed authorization mechanism according to the countermeasures identified in Table 7. The 7 remaining additional causes are all related to

Table 7. Threat analysis of the proposed authorization mechanism. The most important threats, possible countermeasures and remaining residual risks are listed.

Threat	Countermeasure or Residual Risk
(T1) Backdoors in SE, either intentional or unintentional, on robot or machine. Weak implemented or wrong cryptography in SE.	(C1) SEs are certified for a specific security level to prevent the existence of backdoors, and check that strong and correct cryptographic algorithms are used.
(T2) Physical attack (e.g., side-channel attacks) to reveal the secret key on a robot or machine.	(C2) SE provides tamper resistance; therefore, the shared secret is protected from being extracted by an attacker.
(T3) Injection of false sensor data on the machine or robot.	(C3) The computed sensor data hashes prevent manipulations during transfer.
(T4) Manipulation of request ticket, response or authorization ticket during transmission.	(C4) The integrity provided by the generated HOTPs would reveal the manipulations and prevent the action execution.
(T5) Injection of false commands to execute an action.	(C5) The approach prevents execution of commands with missing or invalid tickets since the ticket cannot be validated.
(T6) Physical manipulation of a robot's or a machine's sensor to generate false sensor values.	(C6) The sensors would generate a matching hash for the sensor values. However, the action would not be authorization since the non-manipulated sensor's values would not fulfill the given condition.
(T7) Denial-of-Service (DoS) attack on communication interfaces or tickets.	(R7) DoS attacks would prevent a system from executing tasks since the system is overloaded with request or data. These attacks cannot be mitigated by any security mechanism as they don't try to manipulate a service but shut it down.
(T8) Replay attack on request, response or authorization ticket.	(C8) The counter prevents old tickets from being valid since the HOTP expires after the first use.
(T9) Relay attack on wireless communication interface.	(C9) Relay attacks are related to man-in-the-middle attacks. The adversary tries to act as if he was a valid device. However, the attacker is not in possession of the secret key and cannot generate a valid ticket.
(T10) Manipulation of sensor data on the machine or robot during transmission.	(C10) The sensors use HMACs for their values to provide integrity and prevent manipulations during transmit.
(T11) DoS attack on sensors.	(R11) DoS attacks on sensors would prevent the robot or machine from acquiring sensor measurements. These attacks cannot be mitigated by any security mechanism.

DoS attacks caused by the threats $T7$ and $T11$. These attacks would shut down the authorization process since the communication interface or CPU cannot execute tasks anymore due to a huge amount of incoming data. Both causing threats $T7$ and $T11$ were categorized with a low threat level, meaning that the damage

Table 8. Risk level determination of safety hazards with additional causes due to security threats. Threats marked red remain as additional hazards after application of the security measures.

#	Safety Hazard	Injury Severity	Exposure	Prob. of Avoidance	Risk Level	Security Threats causing the Hazard
1	Robot tips over	S3	E2	A1	high	T1-6, T8-T10
2	Person struck by robot or arm	S3	E1	A1	high	T5, T11
3	Robot strikes object	S3	E2	A2	high	T1-6, T8-T10
7	Failure of obstacle avoidance system	S2	E2	A1	medium	T3, T6, T10, T11
13	Laser scanner blinded	S2	E2	A1	medium	T6, T11
14	False Sensing	S2	E2	A1	medium	T3, T6, T10, T11
19	CPU overheat and shut down	S2	E2	A1	medium	T7
22	Tactile system failure	S3	E2	A3	medium	T3, T6, T10, T11
23	Bumper collision avoidance failure	S3	E1	A1	very high	T3, T6, T10, T11
26	Laser energy hazard to person’s eyes	S1	E1	A2	low	T1, T2, T5, T8

to a system when the threat is executed is low since shutting down the authorization process means that no action execution is performed on the robot or machine.

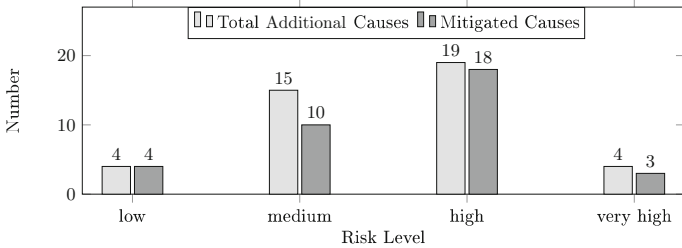


Fig. 2. Numbers of total additional safety hazard causes introduced by a lack of security, and number of additional safety hazard causes mitigated by the proposed authorization approach for each risk level.

6 Conclusion

In this paper, a sensor-fusion based authorization mechanism for industrial mobile robots and equipment on a production floor is shown. The mechanism fuses the sensor data of multiple devices to validate the physical presence of

the robot at the corresponding machine, and that only authorized actions are performed to protect human workforce and in further consequence production material from damage or harm. The mechanism is supported by SEs to increase the security and protect the key material from being revealed by an adversary.

The combined safety and security analysis shows that a significant number of additional safety hazard causes is introduced by a lack of security. The analysis shows that around 83% of the additional safety hazard causes due to a lack of security can be mitigated with the proposed secured authorization mechanism.

Acknowledgment. This work has been performed in the project Power Semiconductor and Electronics Manufacturing 4.0 - (Semi40), under grant agreement No 962466. The project is cofunded by grants from Austria, Germany, Italy, France, Portugal and - Electronic Component Systems for European Leadership Joint Undertaking (ECSEL JU).

References

1. ISO/IEC 11889-1 Trusted platform module library - Part 1: Architecture, August 2015
2. Robotic Industries Association: ANSI/RIA R15.06-2012 American National Standard for Industrial Robots and Robot Systems - Safety Requirements. Technical report (2013)
3. Bartos, R.J.: System safety analysis of an autonomous mobile robot. Technical report, Fernald Environmental Restoration Management Corp., Cincinnati, OH (United States). Fernald Environmental Management Project (1994)
4. Bloomfield, R., Netkachova, K., Stroud, R.: Security-informed safety: if it's not secure, it's not safe. In: Gorbenko, A., Romanovsky, A., Kharchenko, V. (eds.) SERENE 2013. LNCS, vol. 8166, pp. 17–32. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40894-6_2
5. Cirani, S., Picone, M., Gonizzi, P., Veltri, L., Ferrari, G.: IoT-OAS: an OAuth-based authorization service architecture for secure services in IoT scenarios. *IEEE Sens. J.* **15**(2), 1224–1234 (2015)
6. Fraser, B.Y.: Site Security Handbook (1997). RFC2196
7. Gonçalves, J., Lima, J., Oliveira, H., Costa, P.: Sensor and actuator modeling of a realistic wheeled mobile robot simulator. In: *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2008*, pp. 980–985. IEEE (2008)
8. Grieco, L.A., et al.: IoT-aided robotics applications: technological implications, target domains and open issues. *Comput. Commun.* **54**, 32–47 (2014)
9. Hall, D.L., Llinas, J.: An introduction to multisensor data fusion. *Proc. IEEE* **85**(1), 6–23 (1997)
10. Haller, N.: The S/KEY One-Time Password System (1995). RFC 1760
11. He, H., et al.: The security challenges in the IoT enabled cyber-physical systems and opportunities for evolutionary computing other computational intelligence. In: *2016 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1015–1021, July 2016
12. Kam, M., Zhu, X., Kalata, P.: Sensor fusion for mobile robot navigation. *Proc. IEEE* **85**(1), 108–119 (1997)

13. Kim, J.H., Keller, B., Lattimer, B.Y.: Sensor fusion based seek-and-find fire algorithm for intelligent firefighting robot. In: 2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pp. 1482–1486, July 2013
14. Lamport, L.: Password authentication with insecure communication. *Commun. ACM* **24**(11), 770–772 (1981)
15. Line, M.B., Nordland, O., Røstad, L., Tøndel, I.A.: Safety vs security? In: Proceedings of 8th International Conference on Probabilistic Safety Assessment and Management (PSAM 2006), New Orleans, USA (2006)
16. Lynen, S., Achtelik, M.W., Weiss, S., Chli, M., Siegwart, R.: A robust and modular multi-sensor fusion approach applied to MAV navigation. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3923–3929, November 2013
17. Macher, G., Höller, A., Sporer, H., Armengaud, E., Kreiner, C.: A combined safety-hazards and security-threat analysis method for automotive systems. In: Koornneef, F., van Gulijk, C. (eds.) SAFECOMP 2015. LNCS, vol. 9338, pp. 237–250. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24249-1_21
18. Mirzamohammadi, S., Chen, J.A., Sani, A.A., Mehrotra, S., Tsudik, G.: Ditio: trustworthy auditing of sensor activities in mobile & IoT devices. In: Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems, p. 14. ACM (2017)
19. M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D., Ranen, O.: HOTP: an HMAC-based one-time password algorithm. Technical report (2005)
20. Myagmar, S., Lee, A.J., Yurcik, W.: Threat modeling as a basis for security requirements. In: Symposium on Requirements Engineering for Information Security (SREIS), vol. 2005, pp. 1–8. Citeseer (2005)
21. Niemueller, T., Ewert, D., Reuter, S., Ferrein, A., Jeschke, S., Lakemeyer, G.: RoboCup logistics league sponsored by festo: a competitive factory automation testbed. In: Jeschke, S., Isenhardt, I., Hees, F., Henning, K. (eds.) Automation, Communication and Cybernetics in Science and Engineering 2015/2016, pp. 605–618. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42620-4_45
22. Pfitzner, C., et al.: 3D Multi-sensor data fusion for object localization in industrial applications. In: 41st International Symposium on Robotics, ISR/Robotik 2014, pp. 1–6, June 2014
23. Popovici, A., Frei, A., Alonso, G.: A proactive middleware platform for mobile computing. In: Endler, M., Schmidt, D. (eds.) Middleware 2003. LNCS, vol. 2672, pp. 455–473. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44892-6_23
24. Rezvani, M., Ignjatovic, A., Bertino, E., Jha, S.: Secure data aggregation technique for wireless sensor networks in the presence of collusion attacks. *IEEE Trans. Dependable Secure Comput.* **12**(1), 98–110 (2015)
25. Sun, W., Yu, S., Lou, W., Hou, Y.T., Li, H.: Protecting your right: attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. In: 2014 Proceedings IEEE, INFOCOM, pp. 226–234. IEEE (2014)
26. Yi, X., Bouguettaya, A., Georgakopoulos, D., Song, A., Willemson, J.: Privacy protection for wireless medical sensor data. *IEEE Trans. Dependable Secure Comput.* **13**(3), 369–380 (2016)
27. Zuehlke, D.: SmartFactory-towards a factory-of-things. *Ann. Rev. Control* **34**(1), 129–138 (2010)



Towards a Common Ontology of Safety Risk Concepts for Railway Vehicles and Signaling

Bernhard Hulin¹(✉), Hermann Kaindl², Roland Beckert², Thomas Rathfux², and Roman Popp²

¹ NTC-Systems GmbH, Gilching, Germany
bernhard.hulin@ntc-systems.com

² Institute of Computer Technology, TU Wien, Vienna, Austria
{hermann.kaindl,roland.beckert,thomas.rathfux,roman.popp}@tuwien.ac.at

Abstract. In the railway domain, different methods are applied for estimating *safety targets* (like SIL) in the subdomains of railway rolling stock (e.g., SIRF) and railway control, command and signaling (e.g., BP-Risk), respectively, which are referred to as railway vehicles and railway signaling for the rest of this paper. Such methods are also based on different terminology underlying different concepts used, e.g., as parameters. Even worse, similar terms often mean different concepts. This may lead to different risk estimates for these subdomains of the railway domain.

Our approach for addressing these problems has been to create a common *safety ontology* covering the important concepts of both subdomains. Hence, we analyzed the methods SIRF and BP-Risk with regard to the terms and parameters used. Based on this analysis and a previous safety ontology for railway vehicles, we created a new common ontology for railway vehicles and signaling. It is also consistent with the related terminology of EN 50126 (for railway systems) and ISO 26262 (for automobiles). Such an ontology should facilitate the reuse of hazard and risk analyses from one subdomain to the other, and it should have important application areas such as estimating safety targets consistently.

1 Introduction

In safety-critical domains like railway, estimating safety targets is of great importance. From the safety-criticality of hazards and from the safety-criticality/relevance of functions that lead to or facilitate the hazard, a safety target can be allocated to the function. This safety target can be expressed, e.g., in TFFR (tolerable function failure rate), THR (tolerable hazard rate) or SIL (safety integrity level), and is a risk acceptance criterion for the explicit risk estimation.

Bernhard Hulin did major part of this work when he was with the Transportation division at Assystem, Munich.

For the calculation of such a safety target, TR 50126-2 [2] suggests the *Risk Graph* method. SIRF (Sicherheitsrichtlinie Fahrzeug, see [8]) is a German tailoring of EN 50126 for safety assessment of functions of railway vehicles. It was first released in 2011 by the German national safety authority and has been applied successfully in many projects in Germany and Austria for main-line railway vehicles as well as for railway vehicles for urban transport (e.g., metro lines and people movers).

For risk assessment of railway signaling systems, BP-Risk (Best Practice Risk, see [7]) is another tailoring of EN 50126 [1]. Although it is not standardized, it is widely used around the world, e.g., in Germany, Austria, Switzerland (see [5]) and Korea (see [13]).

As it stands, these methods focus on their specific subdomains, and for some functions they cannot reasonably be applied in the other subdomain. For instance, the parameters for risk assessment of the railway vehicle subdomain defined by SIRF only vaguely comply with the specifics of railway signaling, e.g., for some functions of railway level crossing SIRF assigns a SIL that is too low for the design (i.e., the safety target is too weak). Vice versa, the risk assessment method for railway signaling (BP-Risk) is not detailed enough for the application to some functions of railway vehicles and, hence, would lead to erroneous results. For example, calculating the safety targets for air conditioning functions in railway vehicles would lead for high speed lines to SIL 4 with BP-Risk and to a more realistic SIL 1 with SIRF. This is not surprising since BP-Risk considers the speed of the train for the calculation of safety targets, but the speed does not matter for air conditioning functions.

For some other functions like *hot axle box detection*¹ both methods, SIRF and BP-Risk can reasonably be applied. However, both methods result in different safety targets for the same function in the same operational situation and the same accident scenario. This usually leads to heated discussions between experts of both railway subdomains.

Such contradicting safety targets are quite usual at intersections between these two subsystems. These intersections occur at railway functions

- that are facilitated by using components of both railway subdomains, or
- that are integrated on a component that fulfils functions of the other subdomain (e.g., electro-pneumatic modules of the braking system), or
- that can be executed in both railway subdomains independently (e.g., wayside and on-board hot axle box detection), or
- that are currently operated by the components of the railway vehicle including the driver and shall be operated in the future by railway signaling components (e.g., functions for automated train operation (ATO)), or
- whose potential failures can be caused by components of railway vehicles or signaling components.

¹ Hot axle box detection can be monitored with both wayside and on-board devices. It depends on the definition to which subdomain these devices are assigned to.

To simplify the discussions on such interfaces by giving a common understanding of the safety risk concepts of both subdomains, we created a new *ontology* that integrates the relevant concepts of SIRF and BP-Risk. It facilitates a better understanding of the underlying concepts (and relations), in spite of the different terminology used in the different methods/standards, and this is supposed to lead to improved risk assessment.

For addressing similar compatibility problems with risk-related terminology between railway vehicles and automobiles, we previously created a small common ontology of safety concepts (see [12]). The objective of this former common ontology has been to make safety analyses transferable between both domains such that the safety artefacts of the development life-cycle can be reused more easily in the other domain. Since this previously developed ontology is just for vehicles (railway vehicles and automobiles) it does not cover important concepts for risk assessment of railway signaling.

The remainder of this paper is organized in the following manner. First, we briefly review related work. Then we present an analysis of the terminology used in the investigated methods for railway risk assessment. After that, we present our common ontology and the rationale for some ontological decisions we made in the course of its creation. We also describe use cases for illustrating the potential usefulness of this new ontology for avoiding errors. Finally, we conclude and sketch future work.

2 Related Work

The objectives of the European Project DESTINATION RAIL² include achieving a safer rail infrastructure. To this end, it compared different risk assessment methods including BP-Risk but not, for instance, SIRF. Hence, its results (so far) did not help us to resolve the problem addressed in this paper.

SafetyMet presented by Vara et al. [18] deals with transferring safety compliance information from one domain to another. This approach, developed in the OPENCROSS Project (see <http://www.opencross-project.eu/>), tries to facilitate that using model transformation. A common metamodel is defined and transformation rules are used to convert compliance information from one model to another.

Gallina and Szatmári [9] addressed the problem of inconsistencies between different safety *processes* and is complementary to our work on resolving inconsistencies between safety *concepts*. The effort of the Object Management Group (OMG) to create a Dependability Assurance Methodology (DAF) [10] as an umbrella methodology for all dependability attributes such as safety, security, integrity, etc. is also complementary. This effort includes the Dependability Conceptual Model (DCMs) to unify the terminology and the vocabulary from different dependability standards. However, this metamodel does not go into details of safety concepts and their relationships as our ontology-based approach.

² See <http://www.destinationrail.eu>.

A few ontologies for safety and risk concepts exist already, for example, the safety ontologies of Haavik [11]. They are more dedicated to system development than to risk estimation, however.

Lawrynowicz and Ławniczak [16] presented a core ontology for the occupational safety and health domain. Most of the concepts defined there have the same or similar names as in our ontology. However, some of the concepts seem to be defined quite differently. For example, Hazardous Event is defined there as an event with at least one participating worker exposed to an Occupational Hazard. This definition is quite different from the one in the current version of IEC 61508-4 [3].

Luo et al. [17] defined an ontology for certification of components for different domains. Among others, the railway and the automotive domains are taken into account. Luo et al. modeled all the assessment parameters for the allocation of ASIL mentioned in ISO 26262 as properties of a concept that they named “hazardevent”, which is, in our opinion, Hazardous Event.

Kostov et al. [15] classified the concept Safety Event into subclasses Deviation, Collision, Warning System Alarm, Regulation Violation Event, Loss of Function, and Near Collision. Loss of Function corresponds to the definition of Failure in ISO26262-1: “termination of the ability of an element, to perform a function as required”. Our ontology is closer to related standards by including the concepts Accident and Hazardous Event instead, and by keeping the terminology of the standards.

The closest work to ours is the one by Zhou et al. [19], who presented an ontology of basic safety risk concepts for the railway domain based on the standard EN 50126-1 [1]. That is why some of the core concepts including their intermediate relations could be used for our ontology. The scope of our ontology is, however, different. We focus on the concepts (such as “possibility of avoidance”) that are necessary for the determination of a SIL. Most of the concepts for that purpose are not mentioned in [19], however.

In our own previous work (see [14]), we also created a small ontology of concepts such as Risk, Harm, Hazard, etc., which we consider as the core concepts. As a result, this *core ontology* of safety risk concepts reconciled the scientific literature with standards. Since it matches the terminology of the related standards, it may serve as a reference model. In fact, we already used it ourselves for systematically studying where human error may compromise safety. While some of these core concepts are also included in our new ontology presented in this paper, this new ontology relates them to several important safety risk concepts for railway vehicles and signaling as newly conceptualized here.

3 An Analysis of Railway Risk Assessment Terminology

Before defining a conceptual model for an ontology, we had to analyze the terminology behind the concepts of railway risk assessment. Let us sketch here our related analyses of SIRF and BP-Risk terminology as well as other important railway safety terminology of EN 50126.

3.1 Terminology of SIRF

For the determination of safety targets, SIRF provides its own method. (Since SIRF is a German directive without an official and agreed English translation, we translated the definitions of terms for safety target determination on our own.) It defines five parameters:

- S_A – number of affected persons (SIRF: “Anzahl der betroffenen Personen”)
- S_V – degree of injury (SIRF: “Verletzungsgrad”)
- W – probability that a function failure results in the expected severity of harm (SIRF: “Eintrittswahrscheinlichkeit”)
- E – mean duration of exposure to a hazard (SIRF: “Expositionszeit”)
- V – possibility of avoidance of the severity of a harm by the person at risk, after the occurrence of the primary hazard (SIRF: “Vermeidung”)

Parameters S_A and S_V shall be estimated for a realistic worst-case scenario. Their combination is an estimate of the severity of harm.

Parameter W of SIRF is alternatively often referred to as inevitability of the transition from a function failure to the related severity of harm. More specifically, the transition is between one special function failure mode and a set of events that belong to a special class of accident. Among others, a class of accidents can be defined according to the energy type involved (e.g., kinetic, electric, thermic), according to the direction of energy flow (e.g., side-swipe collision, rear-front collision), according to the type of harm (e.g., accident with physical injury, accident with material damage), or according to severity of harm (e.g., catastrophic accident, severe accident, accident without more than 1 fatality and at most 9 fatalities). In SIRF, parameter W accidents are classified according to their severity. Thus, parameter W is the probability p_3 as illustrated in Fig. 1.

In addition, SIRF uses the terms function (SIRF: “Funktion”), failure (SIRF: “Versagen”), accident (SIRF: “Unfall”), hazard (SIRF: “Gefährdung”), harm (SIRF: “Schaden”), severity of harm (SIRF: “Schadensausmaß”) and operational situation (SIRF: “Betriebsbedingungen”), but it does not use the term hazardous event.

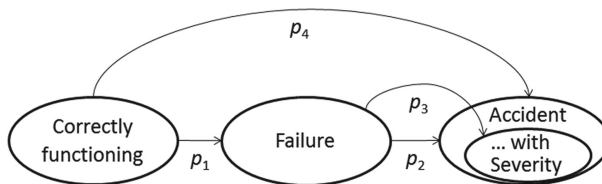


Fig. 1. Types of transitions and their probabilities

3.2 Terminology of BP-Risk

For the determination of the safety target, BP-Risk uses five parameters (see [5], the latest version):

- B – operating density (BP-Risk: “Betriebsdichte”)
- M – human prevention (BP-Risk: “Menschliche Gefahrenabwehr”)
- T – mass of train (BP-Risk: “Masse des Zuges”)
- V – velocity of train (BP-Risk: “Geschwindigkeit des Zuges”)
- A – number of affected persons (BP-Risk: “Anzahl betroffener Personen”)

Unfortunately, some of the parameters are defined slightly differently in different publications. Parameter M was alternatively defined as “human mitigation”, “human prevention of accidents and reduction of the resulting severity of harm”, “human averting of danger”, “corrective action” (see [13]), and “potential mitigation of an accident or undesired event”. Even if the intentions of the different definitions of parameter M were the same, the *concepts* behind the different definitions of this parameter M are different.

From a railway signaling point of view, the different concepts behind parameter M can be assumed to be the same, since only accidents that release kinetic energy (e.g., collision, derailment) are considered. Thus, for railway signaling, a prevention of harm is often just possible with a prevention of an accident. Note, that this holds just for railway signaling but not for the railway vehicle domain.

It seems as though the authors of BP-Risk intended to relate the concept “human prevention” to harm. This is especially the case in operational situations where an accident is inevitable but there may be different types of harm (e.g., material damage or physical injuries).

Only those possible actions shall be considered for parameter M that are not defined as being part of the railway system. That is, a human prevention of harm can be done by a road user, by a passenger, by the police, by workers, by train drivers, etc.

Whereas parameter B is widely defined as operating density, in some publications it is defined as “probability of confrontation” (e.g., for the application for hot box detection, see [6]), which is from an ontological point of view a different concept. For our analysis, we take parameter B as “operating density”, since this is the definition in the latest version of BP-Risk.

In addition, BP-Risk uses the terms function (BP-Risk: “Funktion”), failure (BP-Risk: “Versagen”), accident (BP-Risk: “Unfall”), hazard (BP-Risk: “Gefährdung”), function failure (BP-Risk: “Funktionsversagen”) harm (BP-Risk: “Schaden”), severity of harm (BP-Risk: “Schadensausmaß”) and operational situation (BP-Risk: “betriebliche Randbedingung”), but it does not use the term hazardous event. The severity in BP-Risk refers to both material damage and human harm, while severity in SIRF just refers to human harm.

3.3 Other Important Railway Safety Terminology of EN 50126

The term “hazardous event” is used in the railway standard EN 50126 [1] as well as in the technical report TR 50126-2 [2]. We considered (and still consider) this term as being important, since it is used in many safety analyses in several domains, e.g., the railway domain and the automotive domain. That is why we included it into our previous ontology, see [12].

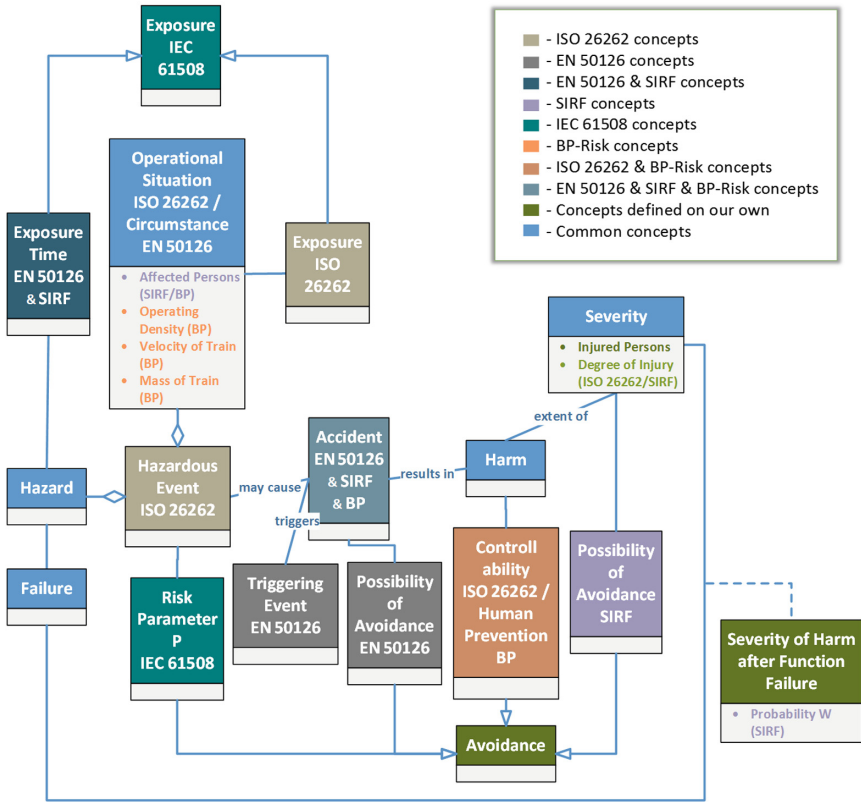


Fig. 2. Common ontology of safety risk concepts for railway vehicles and signaling

As we pointed out already in [12], the definitions of “hazardous event” are different in TR 50126-2 [2], ISO 26262 [4] and IEC 61508 [3]. The most reasonable definition in our opinion is the one of the ISO 26262 standard: “combination of a hazard and an operational situation”.

In addition, the technical report TR 50126-2 defines the term “circumstance”. The concept behind is in our view the same as the concept behind the term operational situation of SIRF or BP-Risk.

Moreover, TR 50126-2 uses the term “possibility of avoiding the accident”, which we include in our analyses since it is slightly different from both SIRF and BP-Risk terminology.

The term “probability of occurrence of an accident” of TR 50126-2 is illustrated in Fig. 1 as p_4 . Note, that this is different from parameter W of SIRF.

4 Our Common Ontology

Figure 2 shows our new common ontology. This diagram uses notation of the Unified Modeling Language (UML), see <http://www.omg.org/spec/UML/> for

the current version. Each arrow head points from a *class* representing a specific concept to a more general one, which represents a *generalization*. The other lines between classes show *association* relations, i.e., we use them for representing relations between concepts. A specific association called *aggregation* is indicated by a diamond at the end of the line with the composite class, which we use to model a kind of composition of concepts. While we model most of the concepts as classes, some are represented here as *attributes* of classes. This indicates that these concepts serve as properties of other concepts.

Our new common ontology of risk concepts for railway vehicles and signaling is based on our previous safety ontology of railway vehicles and automobiles (see [12]). In order to keep this paper self-contained, we first summarize its essence here. In more detail, we elaborate on specific concepts relevant for either railway vehicles or signaling, or both.

4.1 Essence of Previous Ontology

Already our previous safety ontology of railway vehicles and automobiles represents, of course, a few concepts commonly used in all the standards discussed in this paper: Hazard, Harm and Severity. They seem to be core concepts both in the automotive and the railway domains, and they are shared with our new ontology presented in this paper. Strictly speaking, there is also a merged concept in common, Operational Situation ISO 26262/Circumstance EN 50126, since we consider these concepts of the two standards very similar.

There is a generalization according to the generic standard IEC 61508. The similar concepts Exposure ISO 26262 and Exposure Time EN 50126 & SIRF have a pendant in the IEC 61508 risk parameter *F*. Conceptually, we consider the latter as a generalization of the former concepts. Hence, this concept is included here under the name Exposure IEC 61508, with generalization relations in the model.

Also the similar concepts Controllability ISO 26262, Possibility of Avoidance EN 50126 and Possibility of Avoidance SIRF have a pendant in the IEC 61508 risk parameter *P*. However, the latter is *not* a generalization of the former concepts, since it refers to yet another concept, Hazardous Event ISO 26262. Ontologically, these four concepts deserve a common generalization. Hence, for integrating all these avoidance concepts, we created one with the name Avoidance (also Controllability ISO 26262 is actually an avoidance concept).

For more details on our previous ontology, see [12].

4.2 Avoidance and Its Specializations

For railway vehicles, the concept Possibility of Avoidance (of SIRF) relates to Severity, as given in our previous ontology already. In SIRF, the following example is given (translation from German by these authors): “possibility of fleeing of a passenger from a burning part of the train”. Hence, the Possibility of Avoidance (of SIRF) does not directly relate to Harm, but indirectly through Severity.

In contrast, as discussed above for railway signaling, Human Prevention of Harm influences the whole train and, thus, all Affected Persons' health. According to our analysis, this BP-Risk concept is reminiscent of the concept Controllability of the automotive domain [4]. Hence, we introduced the concept Human Prevention BP into our new ontology by merging it with the previous concept Controllability into one concept, which directly relates to Harm.

4.3 Severity of Harm After Function Failure

While our previous ontology already dealt with SIRF to a certain extent, it did not include its parameter W, which reflects the probability of a function failure resulting in the expected severity of harm. Introducing it into our new ontology was deemed necessary, but this turned out to be difficult. While it obviously relates to both Failure (of a function) and Severity (which relates to Harm), being (ontologically) a *probability* does not allow its introduction directly "between" the classes representing these concepts simply as yet another class. In fact, as shown in Fig. 1, it is the probability of a transition denoted as p_3 .

In contrast, ontologies like ours represent static concepts and their relations. Hence, we defined an association relation between Failure and Severity, but the ontology had to reflect a particular property of this relation as well. Fortunately, UML facilitates that through an *association class*, which is both an association and a class. As being a class as well allows defining a property as its attribute. Using this modeling construct, we introduced the concept Severity of Harm after Function Failure (not explicitly defined in SIRF), which is also an association relation between Failure and Severity. For the class representing this concept, we defined the attribute "Probability W (SIRF)".

4.4 Injured Persons

Parameter S_A of SIRF, the number of affected persons, is in our new ontology a property of Operational Situation/Circumstance. However, this parameter is used for the estimation of Severity of Harm in case of an Accident. Hence, the relationship of this parameter to the concepts involved is not precisely defined in SIRF.

Of course, only those persons can be involved in an Accident that are affected by the preceding Hazardous Event, and only those persons' health can be affected by an Accident that are directly involved in this Accident. Thus, for functional safety in railway engineering, Eq. 1 holds, where p_c is the number of persons affected by a Hazardous Event in a certain Circumstance, p_a the number of persons affected by an Accident, and p_i the number of injured persons:

$$p_c \geq p_a \geq p_i \quad (1)$$

In order to address this SIRF problem, we introduced Injured Persons as a property of the concept Severity. These are all the persons whose health is causally affected by an Accident. For worst-case estimation, the number of Injured Persons is the number of Affected Persons (by a Hazardous Event/Circumstance), assuming that all these persons will actually be injured.

4.5 Implicit Relation Between Energy and Severity

Usually, the amount of energy involved in a (railway) Accident has a strong influence on the Severity of Harm. Hence, we intensively discussed the relations of the BP-Risk parameters velocity and mass of a train with other concepts, since they can be used for the estimation of the kinetic energy of a train.

As a result of these discussions, we made an ontological decision to not link these two BP-Risk parameters directly with Severity. Instead, we introduced them as properties of Operational Situation/Circumstance.

Assuming that train passengers are not directly protected by any system (e.g., a seat belt or an airbag) in the train, it may be possible to associate the velocity with the SIRF parameter S_V (the degree of injury). The degree of injury of passengers in a train is usually independent of the mass of the train.

4.6 Operating Density

As indicated above, we considered parameter B of BP-Risk as the concept Operating Density. According to the user manual of BP-Risk³, it could be defined in terms of trains per hour. This does not take into account, e.g., situations like boarding, of course. Assuming that each train has at least one person on board, we can estimate the probability of such persons' exposure with Operating Density for collision and derailment accidents. Hence, we decided to integrate Operating Density as a property of Operational Situation.

4.7 Summary

We showed that some of the concepts behind the parameters for determining the safety target used in SIRF and BP-Risk are the same (e.g., Affected Persons), some can be associated with each other (e.g., Degree of Injury with Velocity of Train and Mass of Train), and some have a common generalization (e.g., Avoidance is a generalization of Possibility of Avoidance and Human Prevention). Considering the specific circumstances of railway vehicles and signaling, both SIRF and BP-Risk are reasonable and usable methods for their respective subdomains. Their essential concepts are integrated in our common ontology.

5 Use Cases for the New Ontology

In our own experience, we observed cases of misinterpretation of BP-Risk parameters by safety experts used to SIRF. We conjecture that this may happen analogously the other way round as well. The reason behind is transfer of the meaning based on similar wording in natural language, which conceals the different concepts behind. Let us illustrate such cases by assuming that a BP-Risk expert has to use SIRF for the first time, in two use cases with two different parameters each. We explain how our new ontology may help here to avoid errors through its representation of concepts and their relations.

³ See http://archiv.ivt.ethz.ch/oev/risk_safety_rail/Benutzerhandbuch.pdf.

5.1 Railway Level Crossing

In our first use case, a misinterpretation of the meaning of parameter V of SIRF (possibility of avoidance) by transferring the meaning of the parameter M of BP-Risk (human prevention) would lead to an error.

The context of this use case is a railway level crossing with full barriers and a level-crossing obstacle detection device (implemented with radar) that monitors the danger zone for metal road cars that are larger than $1\text{ m} \cdot 0.5\text{ m} \cdot 0.5\text{ m}$. Assume that a detection of persons is impossible for this device. When the level crossing shall be activated, approaching road users are signalled not to enter the level crossing (e.g., by blinking red lights). Once the danger zone is free of cars, the barriers will be activated and close the level crossing. After that, the train receives the permission to pass the level crossing via the train control system.

A failure in the obstacle detection device could lead to closing the barriers and passing permission to trains with (e.g., broken down or crashed) cars at the level crossing, even though correctly behaving car drivers had have entered the level crossing before the road side signals were activated. For this example, we assume a train speed of 160 km/h , at which this level crossing is visible for the train driver only after it is too late to brake for standstill before the train reaches the level crossing.

According to BP-Risk, in a collision between a train and a road participant, a few persons are affected ($A = 2$). Since the train cannot be stopped in time, the only possibility of “human prevention” is by road participants. They could try to move the car outside the danger zone (e.g., manually or by restarting the car) or they could leave the car and the danger area by foot. Therefore, the possibility of human prevention of harm resulting from a collision between a train and a car can be estimated as somewhere between “sometimes possible” ($M = 3$ for entering the level crossing in case of a traffic jam) and “nearly impossible” ($M = 5$ in case of a heavy crash between two cars). Hence, for this level crossing we estimate the possibility of human prevention with “seldom possible” ($M = 4$).

For somebody who is used to BP-Risk but yet unfamiliar with SIRF, the SIRF-parameter V looks very similar to the BP-Risk parameter M . Hence, he may estimate parameter V as “not possible” ($V = 1$). However, this estimate is not correct, and the error cannot be easily understood from the SIRF documentation without studying it in-depth.

In contrast, from our ontology one can easily see that the concept Possibility of Avoidance behind parameter V is related to Severity (of harm) but not directly to the concept Harm itself. Even if harm cannot be prevented, its severity can be reduced by the occupants of the car by leaving it. According to SIRF, the assumed worst-case severity (i.e., “several fatalities”) can be avoided ($V = 1.3$) by the car occupants. This avoidance can be considered as possible since the barriers of the level crossing are closing, which is visible to the occupants of the car or at least the car driver, who can alarm the other occupants.

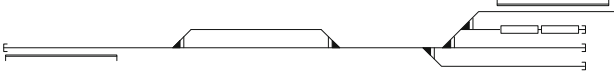


Fig. 3. Railway network for the overspeed example

5.2 Train Overspeed

In our second use case, a misinterpretation of the meaning of parameter E of SIRF (mean duration of exposure) by transferring the meaning of the parameter B of BP-Risk (operating density) would lead to an error. The context of this use case is a network used only for cargo transport, there are no level crossings, and a maximum of two locomotives are operating in this network (see Fig. 3). The maximum possible speed of the locomotives is 60 km/h. The track is designed and constructed in such a way that derailment is not a matter of this maximum speed. However, overspeed of locomotives (or whole trains) may lead to collisions with buffer stops or the other train.

According to BP-Risk, the “operational density” can be estimated as “low” ($B = 1$), assuming that there is only light cargo traffic on the tracks of this network, i.e., it happens just once a day that two cargo trains pass each other.

For somebody who is used to BP-Risk but yet unfamiliar with SIRF, the SIRF-parameter E looks very similar to the BP-Risk parameter B . Hence, he may estimate the exposure time of the train driver using parameter E as “low” ($E = 1$), since the driver is exposed just for a short time to the operational situation of buffer stops or another train passing. In fact, the outcome of the hazard “unrecognized overspeed” is highly dependent on the operational situation in which this overspeed occurs and, hence, “unrecognized overspeed” is limited to collisions with buffer stops or other railway vehicles in the given network. However, estimating $E = 1$ is not correct, and this cannot be easily understood from the SIRF documentation without studying it in-depth.

In contrast, from our ontology one can see that parameter B of BP-Risk is represented as an attribute (Operating Density) of (and hence, related to) the Concept Operational Situation, while the concept Exposure Time behind parameter E of SIRF is related to the concept Hazard. This means that the exposure time shall not be estimated with regard to the hazardous event or operational situation but to the hazard, as defined by SIRF. Hence, the exposure time of the train driver must be estimated with “long” ($E = 1.3$) according to SIRF, because the train driver is exposed to the hazard “unrecognized overspeed” for the whole period of being in his locomotive.

6 Conclusion and Future Work

In this paper, we present a common ontology of safety risk concepts of two widely used methods for safety target determination in the railway domain. SIRF is suitable for railway vehicles only, and BP-Risk for railway signaling only. Our

common ontology is especially useful for safety assessments of systems at the border between these subdomains.

In particular, our two use cases indicate the usefulness of our new ontology for avoiding errors. This may even work more generally, whenever the representation of concepts and their relations helps to clarify their essence. In these use cases, already the graphical representation of the ontology is supposed to be useful. For future tool support, we envisage that a formal representation in an ontology language may be useful as well.

Of course, also this common conceptual model is not anywhere near a fully-fledged common ontology for the railway domain. For instance, the term “hazardous zone” mentioned in the context of railway level crossings should be investigated for inclusion as a concept. Also the term “hazardous situation” should be analyzed, as defined in IEC 61508-4, and mentioned in Section 1.120 of ISO 26262-1 and in Section C.4 of TR 50126-2 [2] without definition. There are obviously also common parts not included, such as the core concept Risk. Hence, our new ontology presented in this paper should be merged with another ontology of core concepts that includes the Risk concept, see [14]. Still, we think the common conceptual model resulting from this paper may serve as a basis on the way towards a common safety ontology, e.g., for railway and automotive systems.

As indicated, future work will have to extend and to evolve the ontology, and it should include an appropriate *upper ontology*. Also case studies in the potential application areas will be important for the sake of validation of our proposed approach.

Acknowledgment. The RiskOpt project (No. 845610) is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program “ICT of the Future” between October 2014 and September 2018. More information can be found at <https://iktderzukunft.at/en/>.

References

1. EN 50126-1: Railway applications - The specification and demonstration of reliability, availability, maintainability and safety (RAMS). Part 1: Basic requirements and generic process, September 1999
2. CLC/TR 50126-2: Railway applications - The specification and demonstration of reliability, availability, maintainability and safety (RAMS). Part 2: Guide to the application of EN 50126-1 for safety, February 2007
3. IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, May 2010
4. ISO 26262: Road vehicles - Functional safety, November 2011
5. Bepperling, S.L., Fermaud, C.: Risikoanalyse für den Stellwerkersatz der Hafenbahn Schweiz AG. SIGNAL + DRAHT Ausgabe 07+08/2015, 18–21 (7+8 2015)
6. Bepperling, S.L., Schäbel, A.: Estimation of safety requirements for wayside hot box detection systems. In: Schnieder, E., Tarnai, G. (eds.) FORMS/FORMAT 2010, pp. 135–143. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-14261-1_14

7. Braband, J.: Definition and analysis of a new risk priority number concept. In: Spitzer, C., Schmocker, U., Dang, V.N. (eds.) *Probabilistic Safety Assessment and Management*, pp. 2006–2011. Springer, London (2004). https://doi.org/10.1007/978-0-85729-410-4_322
8. Eisenbahn Bundesamt Deutschland: Sicherheitsrichtlinie Fahrzeug (SIRF), June 2012. <http://www.eba.bund.de/>
9. Gallina, B., Szatmári, Z.: Ontology-based identification of commonalities and variabilities among safety processes. In: Abrahamsson, P., Corral, L., Oivo, M., Russo, B. (eds.) *PROFES 2015*. LNCS, vol. 9459, pp. 182–189. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26844-6_13
10. Object Management Group: *Dependability Assurance Framework For Safety-Sensitive Consumer Devices* (2016)
11. Haavik, T.K.: On the ontology of safety. *Saf. Sci.* **67**(Suppl. C), 37–43 (2014). <https://doi.org/10.1016/j.ssci.2013.09.004>. The Foundations of Safety Science
12. Hulin, B., Kaindl, H., Rathfux, T., Popp, R., Arnautovic, E., Beckert, R.: Towards a common safety ontology for automobiles and railway vehicles. In: *European Dependable Computing Conference* (2016). <https://doi.org/10.1109/EDCC.2016.15>
13. Jo, H., Hwang, J.G., Kim, Y.K.: Risk assessment method for guaranteeing safety in the train control system. In: *URBAN TRANSPORT*, pp. 567–576, August 2007
14. Kaindl, H., Rathfux, T., Hulin, B., Beckert, R., Arnautovic, E., Popp, R.: A core ontology of safety risk concepts. In: Bogdan, C., et al. (eds.) *HCSE/HESSD 2016*. LNCS, vol. 9856, pp. 165–180. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44902-9_11
15. Kostov, B., Ahmad, J., Křemen, P.: Towards ontology-based safety information management in the aviation industry. In: Ciuciu, I., et al. (eds.) *OTM 2016*. LNCS, vol. 10034, pp. 242–251. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55961-2_25
16. Lawrynówicz, A., Ławniczak, I.: Towards a core ontology of occupational safety and health. In: Tamma, V., Dragoni, M., Gonçalves, R., Lawrynówicz, A. (eds.) *OWLED 2015*. LNCS, vol. 9557, pp. 134–142. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33245-1_14
17. Luo, Y., van den Brand, M., Engelen, L., Klabbers, M.: From conceptual models to safety assurance. In: Yu, E., Dobbie, G., Jarke, M., Purao, S. (eds.) *ER 2014*. LNCS, vol. 8824, pp. 195–208. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12206-9_16
18. de la Vara, J.L., Panesar-Walawege, R.K.: SafetyMet: a metamodel for safety standards. In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P. (eds.) *MODELS 2013*. LNCS, vol. 8107, pp. 69–86. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41533-3_5
19. Zhou, J., Hänninen, K., Lundqvist, K., Provenzano, L.: An ontological interpretation of the hazard concept for safety-critical systems. In: *The 27th European Safety and Reliability Conference*, June 2017. <http://www.es.mdh.se/publications/4707->

Author Index

- Adler, Rasmus 59, 73
Amarnath, Rakshith 73
Artho, Cyrille 123
Asad, Hafizul 267
Ascheid, Gerd 205
- Beckert, Roland 297
Bejga, Mahlet 188
Böde, Eckard 139
Boydens, Jeroen 220
Breu, Ruth 157
Brunner, Michael 157
Büker, Matthias 139
Burton, Simon 45
- Carlan, Carmen 157
Choi, Eun-Hye 123
Chowdhury, Thomas 172
- da Silva, Leandro Avila 73
Ding, Kai 250
Dropmann, Christoph 73
- Eberle, Ulrich 139
- Feth, Patrik 59
Forsberg, Håkan 27
Fränzle, Martin 139
Frigerio, Alessandro 12
Fukuda, Takeshi 59
- Gargantini, Angelo 123
Gashi, Ilir 267
Gauerhof, Lydia 45
Gerwinn, Sebastian 139
Goossens, Kees 12
Guntoro, Andre 205
- Haas, Sarah 282
Hallez, Hans 220
Höllner, Andrea 282
Huber, Michael 157
Hulin, Bernhard 297
- Iliasov, Alexei 91
Ishigooka, Tasuku 59
- Janschek, Klaus 250
Jung, Matthias 73
- Kaindl, Hermann 297
Kang, Eunsuk 172
Kim, BaekGyu 172
Kitamura, Takashi 123
Koopman, Philip 3
Kramer, Birte 139
- Laibinis, Linas 91
Lawford, Mark 172
Lesiuta, Eric 172
Lin, Chung-Wei 172
- Maissonneuve, Quentin 123
Morozov, Andrey 250
Moztarzadeh, Zoha 235
Munk, Peter 45, 73
- Otsuka, Satoshi 59
- Penneman, Niels 220
Popp, Roman 297
- Rathfux, Thomas 297
Rikley, Kerianne 172
Romanovsky, Alexander 91
- Sauerwein, Clemens 157
Schneider, Daniel 59
Schorn, Christoph 205
Schweizer, Markus 73
Schwierz, Andreas 27
Shiraishi, Shinichi 172
Steger, Christian 282
- Taylor, Dominic 91
Thaden, Eike 73
Trapp, Mario 73
Troubitsyna, Elena 107

Uecker, Denis [59](#), [73](#)
Ulz, Thomas [282](#)

Vankeirsbilck, Jens [220](#)
Vermeulen, Bart [12](#)
Vistbakka, Inna [107](#)

Wassyng, Alan [172](#)
Watson, Venesa [188](#)

Yoshimura, Kentaro [59](#)