DEPEND 2016

The Ninth International Conference on Dependability

ISBN: 978-1-61208-492-3

July 24 - 28, 2016

Nice, France

DEPEND 2016 Editors

Elena Troubitsyna, Abo Akademi University, Finland

Pascal Lorenz, University of Haute Alsace, France

DEPEND 2016

Forward

The Ninth International Conference on Dependability (DEPEND 2016), held between July 24-28, 2016 in Nice, France, provided a forum for detailed exchange of ideas, techniques, and experiences with the goal of understanding the academia and the industry trends related to the new challenges in dependability on critical and complex information systems.

With large scale and complex systems, their parts expose different static and dynamic features that interact with each others; some systems are more stabile than others, some are more scalable, while others exhibit accurate feedback loops, or are more reliable or fault-tolerant.

Inter-system dependability and intra-system feature dependability require more attention from both theoretical and practical aspects, such as a more formal specification of operational and non-operational requirements, specification of synchronization mechanisms, or dependency exception handing.

We take here the opportunity to warmly thank all the members of the DEPEND 2016 technical program committee, as well as the reviewers. We also kindly thank all the authors that dedicated much of their time and effort to contribute to DEPEND 2016.

We also gratefully thank the members of the DEPEND 2016 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope DEPEND 2016 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in the area of dependability. We also hope that Nice, France provided a pleasant environment during the conference and everyone saved some time enjoy the beautiful French Riviera.

DEPEND 2016 Advisory Committee

Sergio Pozo Hidalgo, University of Seville, Spain
Manuel Gil Perez, University of Murcia, Spain
Vincenzo De Florio, MOSAIC/Universiteit Antwerpen & MOSAIC/iMinds, Belgium

DEPEND 2016 Industry Liaison Chairs

Piyi Yang, Wonders Information Co., Ltd., China
Timothy Tsai, Hitachi Global Storage Technologies, USA

DEPEND 2016 Research/Industry Chair

Michiaki Tatsubori, IBM Research Tokyo, Japan

DEPEND 2016 Special Area Chairs

Cross-layers dependability
Szu-Chi Wang, National Ilan University, Taiwan
Big Data and dependability
Cesario Di Sarno, University of Naples Parthenope, Italy
Empirical assessments
Marcello Cinque, University of Naples Federico II, Italy
Security and Trust
Syed Naqvi, Birmingham City University, United Kingdom

DEPEND 2016

Committee

DEPEND Advisory Committee

Sergio Pozo Hidalgo, University of Seville, Spain
Manuel Gil Perez, University of Murcia, Spain
Vincenzo De Florio, MOSAIC/Universiteit Antwerpen & MOSAIC/iMinds, Belgium

DEPEND 2016 Industry Liaison Chairs

Piyi Yang, Wonders Information Co., Ltd., China
Timothy Tsai, Hitachi Global Storage Technologies, USA

DEPEND 2016 Research/Industry Chair

Michiaki Tatsubori, IBM Research Tokyo, Japan

DEPEND 2016 Special Area Chairs

Cross-layers dependability
Szu-Chi Wang, National Ilan University, Taiwan

Big Data and dependability
Cesario Di Sarno, University of Naples Parthenope, Italy

Empirical assessments
Marcello Cinque, University of Naples Federico II, Italy

Security and Trust
Syed Naqvi, Birmingham City University, United Kingdom

DEPEND 2016 Technical Program Committee

Habtamu Abie, Norwegian Computing Centre, Norway
Don Adjeroh, West Virginia University, USA
Muhammad Afzaal, Lahore Leads University, Pakistan
Joxe Inaxio Aizpurua Unanue, University of Strathclyde, UK
Eduardo Alchieri, University of Brasília, Brazil
Murali Annavaram, University of Southern California, USA
Luciana Arantes, Université Pierre et Marie Curie (Paris 6), France
Afonso Araújo Neto, University of Coimbra, Portugal
José Enrique Armendáriz-Iñigo, Universidad Pública de Navarra, Spain
Radu F. Babiceanu, Embry-Riddle Aeronautical University, USA

Neminath Hubballi, Infosys Lab Bangalore, India
Bukhary Ikhwan Ismail, MIMOS Berhad, Malaysia
Ravishankar K. Iyer, University of Illinois at Urbana-Champaign, U.S.A.
Arshad Jhumka, University of Warwick - Coventry, UK
Shouling Ji, Georgia Institute of Technology, USA
Zhanpeng Jin, State University of New York at Binghamton, U.S.A.
Yoshiaki Kakuda, Hiroshima City University, Japan
Zbigniew Kalbarczyk, University of Illinois at Urbana-Champaign, U.S.A.
Hui Kang, Stony Brook University, USA
Aleksandra Karimaa, Turku University/TUCS and Teleste Corporation, Finland
Sokratis K. Katsikas, Center for Cyber and Information Security - Gjøvik University College, Norway
Dong-Seong Kim, University of Canterbury, New Zealand
Ezzat Kirmani, St. Cloud State University, USA
Seah Boon Keong, MIMOS Berhad, Malaysia
Kenji Kono, Keio University, Japan
Israel Koren, University of Massachusetts - Amherst, USA
Mani Krishna, University of Massachusetts - Amherst, USA
Mikel Larrea, University of the Basque Country UPV/EHU, Spain
Inhwan Lee, Hanyang University - Seoul, Korea
Matthew Leeke, University of Warwick, UK
Jane W. S. Liu, Academia Sinica, Taiwan
Yun Liu, Boeing Company, USA
Xuanwen Luo, Sandvik Mining, USA
Miroslaw Malek, Humboldt-Universitaet zu Berlin, Germany
Amel Mammar, Mines Telecom/ Telecom SudParis, France
Antonio Mana Gomez, University of Malaga, Spain
Mohammad Mannan, Concordia University, Canada
Gregorio Martinez, University of Murcia, Spain
Célia Martinie, Université Toulouse 3, France
Rivalino Matias Jr., Federal University of Uberlandia, Brazil
Yutaka Matsuno, Nagoya University, Japan
Manuel Mazzara, Innopolis University, Russia
Per Håkon Meland, SINTEF ICT, Norway
Carlos Julian Menezes Araujo, Federal University of Pernambuco, Brazil
Hugo Miranda, University of Lisbon, Portugal
Shivakant Mishra, University of Colorado at Boulder, USA
Costas Mourlas, National and Kapodistrian University of Athens, Greece
Francesc D. Muñoz-Escoí, Universitat Politècnica de València, Spain
Jogesh K. Muppala, The Hong Kong University of Science and Technology, Hong Kong
Jun Na, Northeastern University, China
Syed Naqvi, Birmingham City University, United Kingdom
Sarmistha Neogy, Jadavpur University, India
Mats Neovius, Åbo Akademi University - Turku, Finland
Dimitris Nikolos, University of Patras, Greece
Satoru Ohta, Toyama Prefectural University, Japan
Hong Ong, MIMOS Bhd, Malaysia
Frank Ortmeier, Otto-von-Guericke-Universität Magdeburg, Germany
Roberto Palmieri, Virginia Tech, USA

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

Integrating Security Considerations Into a Safety Case Construction

Elena Troubitsyna
Åbo Akademi University
Tuomionkirkontori 3, 20500 Turku, Finland
e-mail: Elena.Troubitsyna@abo.fi

***Abstract*— Wide-spread reliance on networking in modern safety-critical control systems makes security increasingly interwoven with safety. Hence, we need novel methodologies integrating security consideration into the process of system development and safety case construction. Safety case is a structured argument justifying system safety. In this paper, we propose an approach that relies on the systems-theoretic analysis to construct security-aware safety cases. We define a number of generic patterns facilitating definition of security-aware safety cases. Our approach allows the developers to analyse the mutual interdependencies between safety and security in the design of networked control systems. It provides the engineers with a systematic top-down method for deriving constraints that should be imposed on the system and software behavior to guarantee safety in the presence of accidental and malicious faults.**

Keywords-*safety case; systems-theoretical approach; controlling software; security; integrated analysis*

I. INTRODUCTION

Traditionally safety-critical systems have been considered as closed systems that should ensure safety despite (accidental) components faults [1]. However, increasing openness and reliance on networking has introduced security attacks, i.e., malicious faults, as an important factor to be analyzed in the process of system development and verification [2].

Since safety and security are often considered as separate fields, there is a lack of integrated approaches that support the holistic analysis of software-intensive systems that can guarantee safety in presence of both malicious and accidental faults [1]. However, recent research experiments have demonstrated, e.g., that cars security vulnerabilities allow to remotely override safety functions and take control over break and steering [3]. Therefore, there is a clear need for the approaches that provide the developers with an integrated view on system safety and security.

In this paper, we propose an approach to integrating the security consideration into the process of safety case construction for networked safety-critical control systems.

Safety case is a structured argument about system safety [4]-[8]. Often, it is defined using Goal Structuring Notation [9]. While constructing a safety case, we explicitly define the links between top-level goal of achieving system safety and the satisfaction of constraints that should be imposed on the system design to achieve it.

To derive safety and security constraints required for achieving safety, we propose to employ the systems-theoretic analysis [10]. Systems theory considers the problem of ensuring safety as a control problem and as such, provides us with a more inclusive model of accident causality. Therefore, the systems-theoretic perspective supports an integrated consideration of safety and security constraints that are essential in designing networked control systems.

In this paper, we demonstrate how an application of the systems-theoretic analysis allows us to define the main classes of causes that might lead to unsafe behavior and define the corresponding safety goals. By top-down decomposition of such goals, we define safety and security constraints that should be imposed on the system design to guarantee safety. We define the patterns of safety case fragments that allow us to justify safety in presence of both accidental and malicious faults.

We believe that an application of the proposed approach enables holistic analysis of safety and security interdependencies and facilitates construction of safe networked control systems.

The paper is structured as follows: in Section II, we introduce the notion of the safety case and the Goal Structuring Notation. In Section III, we describe the principles of systems-theoretical analysis. In Section IV, we present our approach to constructing security-aware safety cases using systems theory. In Section V, we overview the related work. Finally, in Section VI, we discuss the proposed approach.

II. SAFETY CASES

A safety case is "a structured argument, supported by a body of evidence that provides a convincing and valid case that a system is safe for a given application in a given operating environment" [4] [5].

The construction, review and acceptance of safety cases are the important steps in safety assurance process of safety-critical systems. Several standards, e.g., ISO 26262 [6] for the automotive domain, EN 50128 [7] for the railway domain, and the UK Defense Standard 00-56 [8], prescribe production and evaluation of safety (or more generally assurance) cases for certification of such systems.

 ISBN: 978-1-61208-492-3

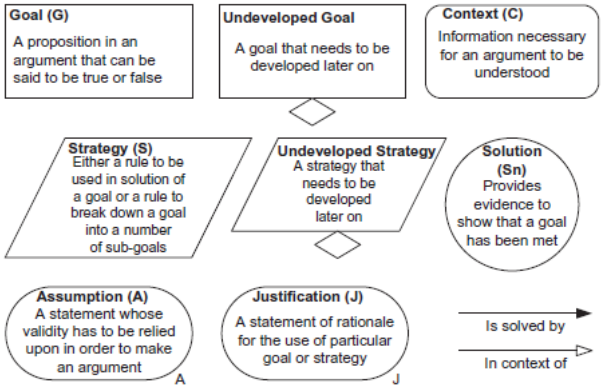

Figure 1. Basic elements of GSN.

A safety case can be defined textually or graphically. Currently, Goal Structuring Notation (GSN) – a graphical notation proposed by Kelly [9] – is getting increasingly popular for describing safety case. GSN aims at a graphical representation of safety case elements as well as the relationships that exist between these elements. The main building blocks of GSN are shown in Figure 1.

Essentially, a safety case constructed using GSN consists of goals, strategies and solutions. Here goals are propositions in an argument that can be said to be true or false (e.g., claims of requirements to be met by a system). Solutions contain the information extracted from analysis, testing or simulation of a system (i.e., evidence) to show that the goals have been met. Finally, strategies are reasoning steps describing how goals are decomposed and addressed by sub-goals. Thus, a safety case constructed in GSN presents a decomposition of the given safety case goals into the sub-goals until they can be supported by the direct evidence (a solution). It also explicitly defines the argument strategies, relied assumptions, the context in which goals are declared, as well as justification for the use of a particular goal or strategy.

The elements of a safety case can be in two types of relationships: ‘‘Is solved by’’ and ‘‘In context of’’. The former is used between goals, strategies and solutions, while the latter links a goal to a context, a goal to an assumption, a goal to a justification, a strategy to a context, a strategy to an assumption, a strategy to a justification.

A typical high-level structure of the safety case is shown in Figure 2. The high-level goal **G1** contains the proposition that the system is safe. The strategy **S1** is to decompose top-level goal into lower level subgoals **G2-G$_{N+1}$** aiming at demonstrating that each individual hazard has been mitigated. The safety case is valid under the assumption **C1** that all hazards have been identified.

Usually, to achieve completeness of hazard identification the developers rely on safety analysis, e.g., fault trees, Failure Modes and Effect Analysis, etc. However, Leveson [10] points out that such techniques rely on linear causality models and lack the power to exhaustively analyse

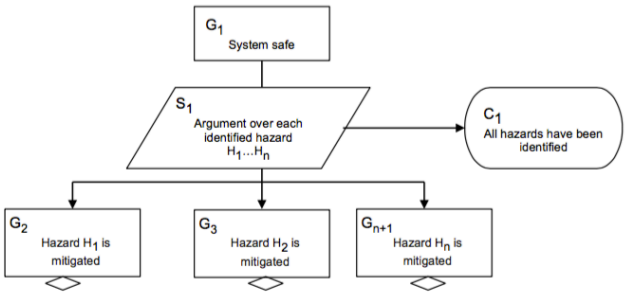

Figure 2. High-level safety case.

hazardous behaviour in complex software-intensive systems. She argues that we need to rely on systems-theoretic approaches to guarantee safety of such complex systems.

In complex software-intensive systems, hazards might be caused not only by accidental (i.e., non-malicious) component failures but also security failures caused by attacks on system network infrastructure, design errors, unforeseen component interactions, etc. Therefore, we need the integrated systems-theoretic approaches that allow us to identify the strategy for protecting the services and functions that are essential for ensuring system safety in presence of disruptions of various natures.

Next, we present a systems-theoretic approach to integrated reasoning about safety of complex networked systems that are subjects of accidental and malicious faults.

III. SYSTEMS-THEORETIC APPROACH

Systems theory establishes foundations for engineering complex systems [11]. It provides a more inclusive model of accident causality called STAMP – System-Theoretic Accident Model and Processes [10]. STAMP envisions losses as resulting from interactions among humans, physical system components and the environment that lead to the violation of safety constraints. The main difference between STAMP and the traditional approaches to safety is that it shifts the focus from preventing failures to enforcing safety constraints on system behavior.

To illustrate the main principles of a systems-theoretic approach, let us consider let us consider a generic control system. A control system is a reactive system with two main entities: an environment and a controller. The environment behaviour evolves according to the involved physical processes and the control signals provided by the controller. The controller monitors the behaviour of the plant and adjusts it to provide intended functionality and maintain safety. The control systems are usually cyclic, i.e., at periodic intervals they get input from sensors, process it and output the new values to the actuators. The general structure of a control system is shown in Figure 3.

The controller is a hierarchical control structure that constraints the system behavior. Each layer of it enforces the required constraints on the behavior of the components at

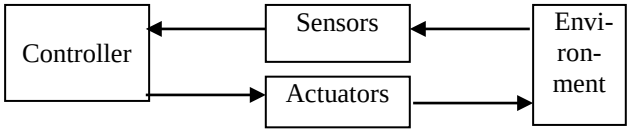

Figure 3. A general structure of a control system.

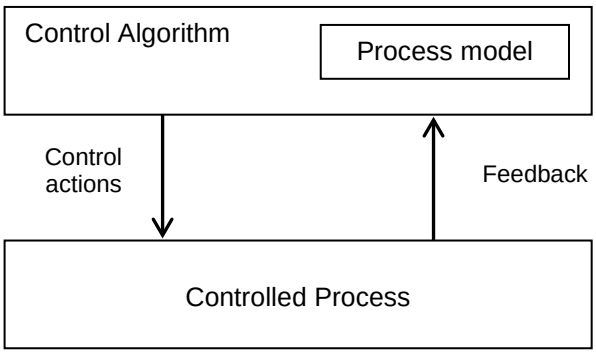

Figure 4. Systems-theoretic view on control system.

the next lower level. The control loops operate between the layers. To achieve safety, we should guarantee that, under the hostile environmental conditions and in presence of accidental and malicious faults, the control actions prevent hazard occurrence.

In systems and control theory, the controller contains a model of the process that it controls. Such a model serves as a basis for defining the necessary control actions, as shown in Figure 4. Hazards often occur as a result of inconsistencies between the controller's model of the controlled process and the actual process state.

By applying systems-theoretic analysis and analyzing the general structure of a control system, we can observe that safety can be violated due to three types of causes

- Controller cannot built a correct model of the process because the measurements provided by the sensors are invalid
- Controller has the correct model of the process but there is a logical error in implementing correct control actions
- The actuator fails to correctly implement the control actions.

This observation allows us to refactor our generic safety case pattern presented in Figure 1 as shown in Figure 5. It reflects the systems-theoretic approach to ensuring safety and establishes a systematic way for construct the safety case by further decomposition of subgoals **G2 –G4.**

In the next section, we propose a systematic approach to analyzing how the accidental and malicious faults introduce inconsistencies into the controller's model of the process, distort the logic of the controller or prevent correct implementation of the controller actions.

IV. SECURITY-INFORMED SAFETY CASES

Let us again consider the generic control cycle presented in Figure 3. For many control systems, safety can be formulated as the following proposition:

The value of critical parameter p always remains within safe boundaries.

To achieve this safety goal, we need to systematically analyse the causes that can introduce hazardous deviations in the controller's model of the process controlling *p* and the actual state of *p*.

To build the corresponding model of the process, the controller relies on the measurement of *p* provided by the corresponding sensor. Therefore, the first condition for ensuring accuracy of the controller's model is validity of sensor's reading.

The sensor's readings can be distorted due to accidental faults of the sensor or security attacks. If the sensor fails and the controller does not detect it, then it starts to rely on wrong data. Hence, to guarantee safety, we should ensure that the sensor health is monitored and upon detection of failure the controller starts to rely on alternative reliable sources of measurement of *p*.

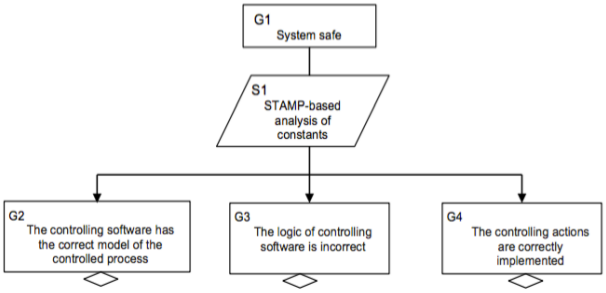

Figure 5. Systems-theoretic approach to safety case.

Typically, safety-critical control systems contain some form of redundancy. For instance, there might be hot or cold spare sensors. In the first case, the controller simply switches to obtaining readings from the spare sensor without any disruption in measurement provisioning. In the second case, a certain time interval is required to activate the cold spare. The system design should ensure that the time period required for the reconfiguration is sufficiently short, i.e., it would not introduce dangerous deviations in the controller's model of the process while the measurements are not available.

The controller might also obtain the invalid measurements of *p* due to security attacks. In the context of our control loop, it is relevant to consider the following security failures:

- spoofing the identity of sensor and
- tampering sensor data by attacking the communication channel between the sensor and the controller.

By spoofing the sensor identity the attacker can supply the controller with the deliberately wrong measurements of *p*. They can "trick" the controller into thinking that the value

 ISBN: 978-1-61208-492-3

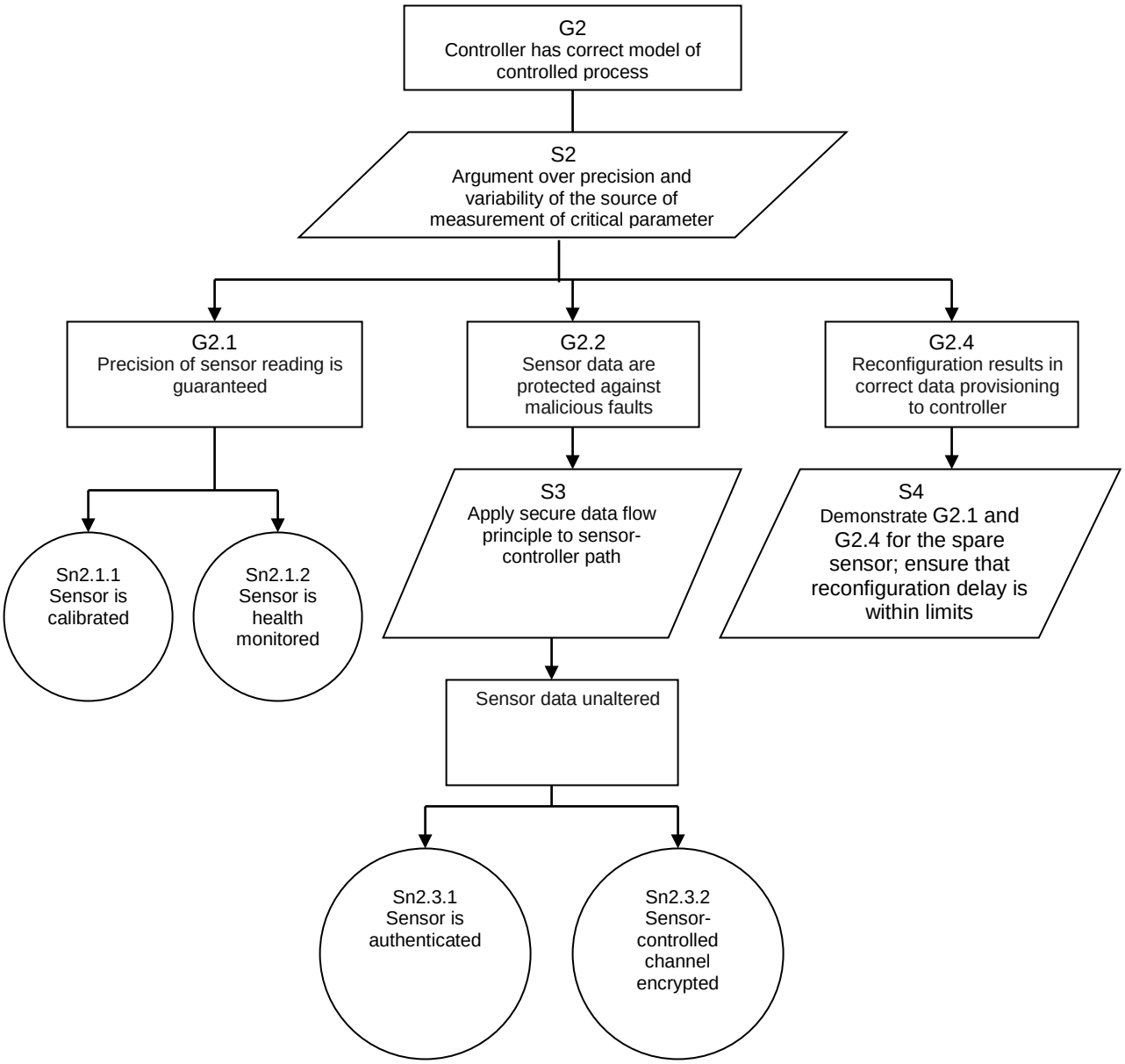

Figure 6. Pattern for **G2** decomposition.

of the controlled parameter is well within safety limits. This is a dangerous deviation of the controller's model of the process. Hence, it is very likely to result in hazard occurrence due to controller inability to issue the correct control commands required to maintain safe value of *p*. Tampering with sensor data has the same effect.

The analysis above demonstrates the direct impact of security on safety. The systems-theoretic approach allows us to identify the strategy for protecting the systems. Namely, we should guarantee that the source of measurement of *p* is authenticated and the communication link between the sensor and the controller is encrypted, i.e. does not allow for unauthorized data alternations. In the similar way, to ensure that the sensor failures are reliably detected, we need to guarantee that the health monitoring data is not tampered with.

We can also demonstrate that safety-security interdependencies are sometimes conflicting and require trade-offs.

To ensure security, the design of software-intensive systems typically follows multi-level secure systems principle introduced by La Padula and Bell [12]. Often the designers consider two security levels: high, meaning highly sensitive or highly trusted, and low, meaning less sensitive or less trusted. When the trusted components of the system interact with the untrusted parts, one has to ensure that there is no indirect leakage of sensitive information from the trusted to untrusted part. Usually it is defined as no "down-flow" policy. Such a security requirement is commonly

 ISBN: 978-1-61208-492-3

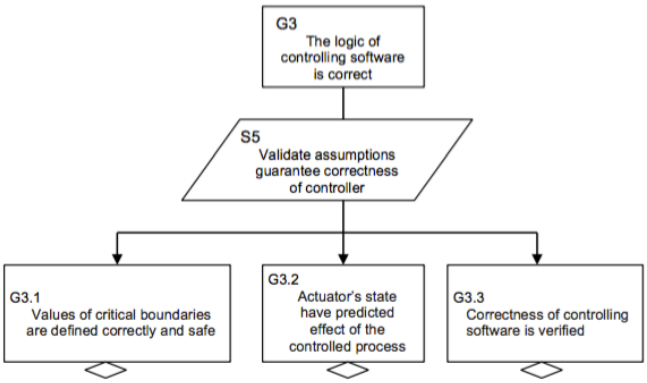

Figure 7. Pattern for **G3** decomposition.

called *secure information flow*.

Let us consider how this principle is implemented in our case. As a result of the analysis above, we have derived the requirements that ensure no down flow policy for the sensor providing the measurements of *p*. Now, let us consider the case when the sensor providing the measurements of *p* has failed and the controller has to use the alternative sources of measuring *p*.

For the spare sensor, the system might not have the authentication and encryption procedure implemented and hence switching to the use of the alternative sensor would break the secure information flow policy. By preventing the use of measurements provided by the spare sensor, we leave the controller without the feedback required to build the adequate model of the controlled process. On the other hand, the use of unauthenticated sensor and unencrypted channel introduces security vulnerability. It is clear that we should resolve this conflict. For instance, we might run spare sensor authentication upon reconfiguration and require to use the encryption once the spare sensor becomes the primary source of measurements of *p*.

The system-theoretic analysis allows us to construct the corresponding part of the safety case, as shown in Figure 6.

Now, let us discuss the constraints that should be imposed on the system to ensure that the goal **G3** is achieved. Essentially, we have to verify that the controller actions maintain the safety invariant *"p is within safety boundaries"*. To verify this we have to introduce a number of assumptions.

The first class of assumptions explicitly states the impact of the actuator state on the value of the controlled parameter. Let us explain it by an example. Assume that the controlled parameter is a temperature *t*. The actual temperature should be kept within safety boundaries *t_min_crit* and *t_max_crit*.

The temperature is controlled by switching on and off the heater. The assumptions that we make is that when the heater is switched on the temperature is increasing. Correspondingly, when the heater is switched off the temperature is decreasing.

Another class of assumptions that we need to introduce deals with the inertia of the controlled physical process and relies on the cyclic behavior of the system. Since the controller receives the measurements of the controlled parameter once per control cycle, it needs to issue the control actions changing the state of the actuator before the critical boundaries are reached.

To demonstrate that the actual value of the parameter always remains within safety limits, we need to constrain Δ - the maximum possible imprecision of the parameter in the process model as well as Δ_{max_cycle} – the maximum possible change of the parameter per cycle.

The state of the actuator should be changed to the one that leads to the increase of the parameter at *P_min*, which is greater than *p_min_crit* at least for the sum of Δ and Δ_{max_cycle}. The similar condition is imposed on P_{max}. Under these assumptions, we can verify that the controlling software maintains safety invariant, i.e., we can argue for achieving the goal **G3**. The corresponding fragment of the safety argument is shown in Figure 7.

Next, we investigate the constraints that should be imposed on the system to justify achieving goal **G4**. It is obvious that if the actuator fails and its failure remains undetected then it directly leads to failure to implement the commands of the controller in the correct way. Therefore, we should guarantee that the failures of the actuator are reliably detected and the system is put in a safe non-operational state upon it.

Now, let us consider the security-related constraints that should be satisfied to guarantee achieving goal **G4**. Even though the controller could have issued the correct control commands, due to spoofing controller identity or tampering commands the actuator might receive the incorrect settings that might breach safety. Therefore, we have to enforce secure data flow policy on the communication between the controller and the actuator as well. The corresponding fragment of the safety case is shown in Figure 8.

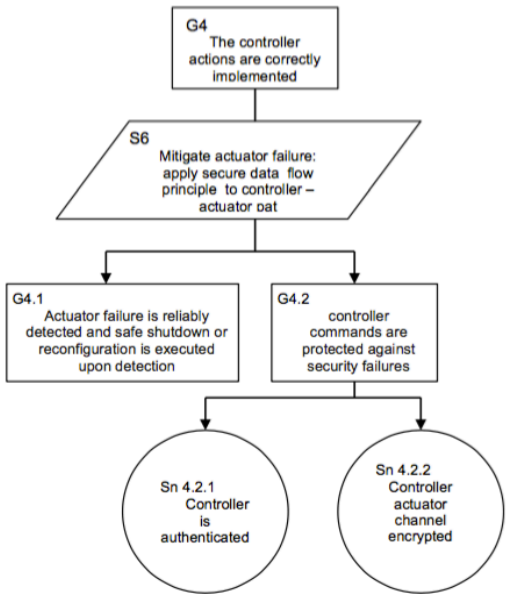

Figure 8. Pattern for **G4** decomposition.

We can summarize the proposed methodology as follows:

1. Apply systems-theoretic approach to analyse how the controller builds the process model.
2. Define the top-level safety goal and identify critical parameters that should be monitored and controlled. Define safety conditions over these parameters.
3. Create an architectural model of the system and identify
    a. the components involved into providing input to the controller allowing it to build the process model (sensors)
    b. the components responsible for implementing controller actions (actuators)
4. For the identified components analyse the impact of failures and define the mitigation actions required to achieve safety goals. Construct the corresponding fragments of the safety case
5. Analyse data flow and define security constraints guaranteeing secure data flow policy for monitoring the critical parameters and implementing controller actions. Construct the corresponding fragments of the safety case
6. Derive the constraints required to verify correctness of the controller logic. Construct the corresponding fragment of the safety case.

V. RELATED WORK

Currently, the problem of integrated analysis of safety and security is receiving significant research attention. Schmittner at al. propose an approach that adapts Failure Mode and Effect and Criticality Analysis to address safety failures [13]. The work demonstrates how to take into account the motives of the intruder as well as costs and complexity of exploiting vulnerabilities. The approach proposed by Schmittner et al. can be used as an input for the safety case construction technique presented in this paper.

The approach relying of the integration of safety consideration into fault tree analysis has been proposed by Steiner and Liggesmeyer. The approach provides the engineering with a structured way to discover and analyse security vulnerabilities that have safety implications. This work complements the systems-theoretical approach to construction of the safety cases proposed in this paper.

Formal approaches proposed to study security and safety interactions typically focus on finding conflicts between safety and security requirements [15]. The majority of the approaches demonstrate how access control rules contradict safety requirements. In our approach, we do not contrapose safety and security but rather derive the security and safety constraints in top-down manner based on the safety cases. The advantage of our approach lies in its ability to capture the dynamic nature of safety and security, e.g., resulting from the reconfiguration required to achieve fault tolerance.

VI. CONCLUSIONS

In this paper, we have proposed a systematic approach to construction of security-aware safety cases. In our approach, derivation of safety and security constraints proceed hand-in-hand with safety case construction. The use of systems-theoretic reasoning allows us to derive the constraints required for providing arguments for safety case in a disciplined top-down way. Such an approach supports an integrated reasoning about safety and security that facilitates analysis of requirements interdependencies and explicit identification of trade-offs required to achieve safety in presence of both malicious and accidental failures.

In our future work, we are planning to validate the proposed approach in a number of industrial case studies as well as provide an automated tool support linking systems-theoretic analysis and safety case construction.

REFERENCES

[1] R. Bloomfield, K. Netkachova, and R. Stroud, "Security-Informed Safety: If It's Not Secure, It's Not Safe". Workshop on Software Engineering for Resilient Systems, LNCS 8166, Springer, Sept. 2013, pp. 17-32. DOI 10.1007/978-3-642-40894-6_2.

[2] W. Young and N.G. Leveson, "An integrated approach to safety and security based on systems theory". Communication of ACM 57(2), 31-35, 2014, DOI 10.1145/2556938.

[3] Online http://www.bbc.com/news/technology-35841571. Accesed 02.04.2016.

[4] T. Kelly and J. McDermid, "Safety case construction and reuse using patterns". 16th International Conference on Computer Safety, Reliability and Security (SAFECOMP'97), Springer-Verlag, Sept. 1997, pp. 55–96, doi: 10.1007/978-1-4471-0997-6_5.

[5] P. Bishop and R. Bloomfield, "A methodology for safety case development", in: Safety-Critical Systems Symposium, Springer-Verlag, Feb.1998, pp.10-16, , doi: 10.1008/243-1-3382-06577-5_6.

[6] International Organization for Standardization, ISO 26262 Road Vehicles Functional Safety, 2011.

[7] European Committee for Electrotechnical Standardization (CENELEC), EN 50128 Railway Applications – Communication, Signalling and Processing Systems – Software for Railway Control and Protection Systems, 2011.

[8] Defence Standard 00-56, UK Ministry of Defence. 00-56 Safety Management Requirements for Defence Systems, 2007.

[9] T.P. Kelly, "Arguing safety -- a systematic approach to managing safety cases", PhD Thesis York University, UK, 1998.

[10] N. Leveson, "Engineering a safer world: Systems thinking applied to safety". In MIT Press. 2011.

[11] P.Checkland, Systems Thinking, Systems Practice. Wiley, 1981.

[12] D.E.Bell and L.J LaPadula.. "Secure Computer Systems: Mathematical Foundations", Journal of computer Security, 1996, 4:239-263.

[13] C. Schmittner, T. Gruber, P.P. Puschner, and E. Schoitsch, "Security application of failure mode and effect analysis (FMEA)". In: SAFECOMP 2014. LNCS 8666, Springer, Sept. 2014, pp. 310-325.

[14] M. Steiner and P. Liggesmeyer, "Combination of Safety and Security Analysis - Finding Security Problems That Threaten The Safety of a System". Workshop on Dependable, Embedded and Cyber-physical Systems, Toulouse, France, 2013, pp.7-14.

[15] P. Bieber and J. Brunel, "From Safety Models to Security Models: Preliminary Lessons Learnt", Workshops at SAFECOMP 2014, pp.269—281, DOI 10.1007/978-3-319-10557-4_

 ISBN: 978-1-61208-492-3

Static Worst-Case Execution Time Analysis Tool Development for Embedded Systems Software

Thomas Jerabek, Martin Horauer

University of Applied Sciences Technikum Wien
Höchstädtplatz 6, A-1200 Vienna, Austria
Email: {thomas.jerabek, martin.horauer}@technikum-wien.at

***Abstract*—Analyzing the *worst-case execution time* of embedded systems software is useful for assessing parameters like schedulability, performance (especially with regard to deadlines), etc. A commonly accepted approach to obtain these values is by way of static analysis that uses the software along with a model of the target processors architecture. This paper describes the required steps to construct a tool to assess the worst-case execution time of a given application with the help of an open-source framework. The ensuing evaluation provides a comparison of the results with other approaches. In addition, this paper can be used as guide to implement an instruction set architecture of a target processor in order to enable various static analyses with the aim to estimate the worst-case execution time.**

Keywords*—*architecture description language; instruction set simulator; worst-case execution time analysis.

I. INTRODUCTION

Embedded systems nowadays are ubiquitous in our daily life. One kind of embedded systems are real-time systems where the correct operation of the system depends on the logical correctness of the computations and upon the time at which the result is produced. Hence, knowledge about execution times – and here in particular the worst-case execution time (WCET) – is of relevance, e.g., to assess whether deadlines imposed by application requirements will be met, or to assess the schedulability of an implementation.

The WCET defines the longest time it takes to execute a program on a specific target processor. There are different ways to determine this value.

(1) One method uses static code analysis [1], [2] by way of a model of the processor's architecture. In fact, various analyses are in use therefore, e.g.:

- Control-flow analysis
- Value analysis
- Cache analysis
- Pipeline analysis
- Path analysis
- WCET estimation

Each of these analyses must be implemented and adapted for every new target architecture.

(2) Another WCET analysis method is measurement-based where the execution time of an application, function or task is recorded during runtime. To that end, the source code must be instrumented to provide suitable triggers for the measurement and appropriate stimuli are required to stress worst-case behavior. In practice, the latter requires elaborated test-setups [3], [4].

(3) The third approach combines both the static analysis method to evaluate input data and the measurement-based method to estimate the WCET [3].

Independent of the chosen approach, it is essential that the real WCET is never longer than the evaluated value and the result is as close as possible to the reality. These two aspects describe a safe and tight WCET evaluation.

Dependable systems have the ability to avoid service failures, which are unacceptable in terms of frequency and severity. Many aspects need to be considered in order to ensure such a behavior; however, this paper focuses on a specific detail: predictable execution times of dependable software (e.g., a hard real-time system) via WCET analysis. Such an analysis is an inherent part of the safety process during the design and development of automotive and avionics systems to avoid timing issues [5], [6]. For example, unmanned aerial vehicle (UAV) software contains various tasks (e.g., engine control or position sensing) where the knowledge about their WCET is mandatory for safe operation.

The contribution of this paper presents a generic approach of how to enable WCET analysis for a modern processor architecture following approach (1) using static analysis of executable binaries. Besides the architecture module, two analysis tools were implemented using the OTAWA framework's API. The result is a useful guide to implement WCET analysis for a certain processor architecture. It can be utilized for WCET-aware development to prevent systematic failures in order to increase the reliability of a dependable system.

The structure of the paper is as follows. First, we detail typical design patterns and their effect on WCET using some examples. Next, we present related work in Section III, followed by a description of the implementation in Section IV. Section V provides some benchmarks, and finally, a use case is presented in Section VI before we conclude the paper in Section VII.

II. WCET ESTIMATION OF EMBEDDED SYSTEMS SOFTWARE

Software for embedded systems typically follow either a bare-bone approach or employ some kind of (real-time) operating system. Bare-bone applications in turn either follow a super-loop architecture or a fore-/background approach [7].

 ISBN: 978-1-61208-492-3

When using a preemptive operating system task and/or thread models are in place. Below, we will use these patterns and show how WCET can be estimated by way of examples. This not only gives an insight on the analysis itself, but also encourages the technical background for the further chapters.

In general, static WCET evaluation is based on one of the three present calculation techniques: (1) path-based [8], (2) tree-based [9], and (3) implicit path enumeration technique (IPET) [10]. Since these approaches are usually applied on instructions or basic blocks, they need to be abstracted in order to use them for a higher program representation. The aim of this chapter is to describe the WCET of a program with a formalism using its most basic elements. This high-level formalism is inspired by the path-based approach where the WCET will be determined by first calculating times for different paths within a program and then looking for the path with the longest execution time. It consists of a header T_H and a content/path T_P that is multiplied with its loop-bound LB, as shown in (1). Depending on the programs structure, this basic equation needs to be adapted. All subsequent T variables are already defining the maximum execution time of the corresponding program part as processor clock cycles. This means, that the result is independent of the processors frequency; however, one can convert it into a time via dividing it by the processors clock rate.

$$WCET = T_H + T_P * LB \tag{1}$$

Bare-bone program structures consist of an initialization part T_{IN} and an endless loop T_L. The execution time of the latter equals a cycle time, which is especially relevant for super-loop architectures because they are not using interrupts at all and only detect events via requests (polling). It can be used as maximum response time for a certain event and can be evaluated as shown in (2).

$$WCET_P = T_P = T_L \tag{2}$$

The other kind of bare-bone applications is using an interrupt driven fore-/background architecture. Here, interrupts and the execution of their associated service routines (ISR) need to be considered for WCET evaluation. With its WCET and the execution rate, one can calculate an expected rate (periodicity) relative to the program under analysis. Depending on the referenced program section T_{PS}, the interrupt rate, as estimated by (4), contains only the loop section or the entire application (see (3)). For the latter, interrupts need to be enabled before the program part under analysis. The sum of all interrupt service routines T_{SI} is estimated by adding up the expected timing of every interrupt as shown in (5). The equation assumes that all involved interrupts are activated permanently.

$$T_{PS} = T_L \veebar (T_{IN} + T_L) \tag{3}$$

$$R_I = \frac{T_{PS}}{expected\ ISR\ Rate} \tag{4}$$

$$T_{SI} = \sum_{i \in SI} [T_{Ii} * R_{Ii}] \tag{5}$$

The WCET of one loop cycle of interrupt driven applications takes the interrupts into account and can be estimated by (6). The overall WCET is the result for termination after a certain number of cycles specified by the loop-bound (see (7)).

$$WCET_P = T_P = T_L + T_{SI} \tag{6}$$

$$WCET = T_{IN} + T_P * LB \tag{7}$$

An example is given for a fore-/background structure with 3 interrupt service routines. The WCET of one loop cycle should be estimated. Worst-case time behaviors of each individual part were already evaluated as listed below.

$$T_{IN} = 600\ clock\ cycles$$
$$T_P = 9000\ clock\ cycles$$
$$T_{I1} = 250\ clock\ cycles$$
$$T_{I2} = 890\ clock\ cycles$$
$$T_{I3} = 60\ clock\ cycles$$

We assume three ISRs with the following shortest possible periodicity:

$$ISR1:\ executed\ every\ 5000\ clock\ cycles$$
$$ISR2:\ executed\ every\ 20000\ clock\ cycles$$
$$ISR3:\ executed\ every\ 1400\ clock\ cycles$$

Thus we get:

$$R_{I1} = \frac{9000}{5000} = 1.8 \approx 2$$
$$R_{I2} = \frac{9000}{20000} = 0.45 \approx 1$$
$$R_{I3} = \frac{9000}{1400} = 6.43 \approx 7$$
$$T_{SI} = 250 * 2 + 890 * 1 + 60 * 7 = 1810\ clock\ cycles$$
$$WCET_P = 9000 + 1810 = 10810\ clock\ cycles$$

The result shows that an increasing number of interrupts significantly affects the WCET.

RTOS: There are major differences between bare-bone and real-time operating system (RTOS) structures, such as the administrative overhead and the interruption of execution by a higher priority task in real-time operating systems.

When assuming a priority based scheduler, it is a challenge to evaluate a task's WCET because each task, except the highest priority one, can be interrupted by a higher priority task. As a result, evaluation needs to be done by a top-down approach starting with the highest priority one. Equation (8) describes the maximum interruption time of a task by summing up both, all Tasks with a higher priority T_{HPT} and their administrative overhead T_{AO} (e.g., context switch, scheduling). The subsequent calculation is identical to bare-bone programs, as shown in (9) and (10).

$$T_{INT} = \sum_{i=1}^{n} T_{AOi} + \sum_{i=1}^{n} T_{HPTi} \tag{8}$$

$$WCET_P = T_P = T_L + T_{INT} \tag{9}$$

$$WCET = T_{IN} + T_P * LB \tag{10}$$

For tasks using an endless-loop pattern, the worst-case cycle time $WCET_P$ is typically the most relevant. The overall WCET is the result for termination after a specified number of cycles (loop-bound). For tasks using a run-to-completion

pattern there are typically no cycles and, therefore, the loop-bound equals 1 for the overall WCET.

An example calculation follows for a run-to-completion task, which is implemented into a real-time operating system where 2 tasks with a higher priority are existing. The task's entire WCET should be determined with the assumption that the administrative overhead is task independent. Worst-case time behaviors of each individual part were already evaluated as listed below.

$$T_{IN} = 200\ clock\ cycles$$
$$T_L = 3900\ clock\ cycles$$
$$T_{HPT1} = 2320\ clock\ cycles$$
$$T_{HPT2} = 1100\ clock\ cycles$$
$$T_{AO} = 590\ clock\ cycles$$

This results in:

$$T_{INT} = 590 + 590 + 2320 + 1100 = 4600\ clock\ cycles$$
$$T_P = 3900 + 4600 = 8500\ clock\ cycles$$
$$WCET = 200 + 8500 = 8700\ clock\ cycles$$

The outcome of 8700 cycles shows that the RTOS as well as tasks with a higher priority have a major impact on the tasks WCET.

III. Related Work

In the following, we provide a short comparison of (1) existing WCET analysis tools (cf. Table I) and further on (2) describe related architecture description languages (ADLs).

TABLE I. WCET ANALYSIS TOOL COMPARISON. (*ECLIPSE PLUGIN, **PARTLY, ***NOT REQUIRED)

Name	aiT	Bound-T	RapiTime	SWEET	OTAWA
open source	X	✓	X	✓	✓
static analysis approach	✓	✓	X	✓	✓
measurement-based approach	X	X	✓	X	X
annotations	✓	✓	✓	✓	✓
GUI	✓	X	✓	X	✓*
specify architectures	X	X	✓	X	✓
specify μC characteristics	✓**	✓**	X***	X	✓
binary file input	✓	✓	✓	✓	✓
ISO 26262	✓	X	✓	X	X
DO-178B	✓	X	✓	X	X

The aiT WCET Analyzer [1] from AbsInt features an easy to use GUI with a straightforward configuration. It is able to compute tight bounds of a programs WCET using static analysis. One can choose between different integer linear programming (ILP) solvers (e.g., CPLEX), analysis options and output reports (e.g., HTML and XML). User defined annotations can be provided using the AIS/AIS2 language in order to define loop bounds and other program information. It fulfills the ISO 26262 as well as the DO-178B level A qualifications.

Bound-T [11] (originally developed by Tidorum Ltd. and now released as open-source) is a command-line tool that uses static analysis for the WCET estimation. Annotations can be provided by the user, although, loop bounds can be derived automatically. Unfortunately, cached memory or the parallelism of functional units cannot be specified making it hardly unsuitable for many modern processors.

RapiTime [12] uses a measurement-based approach for the WCET evaluation. It derives a structural model of the program and instruments the source code during the build process of a program. Afterwards, it performs all given tests and extracts the timing data via execution traces. To finish up, a prediction of the worst-case path and WCET is carried out by combining the obtained timing information and the structural model of the code. It fulfills both the DO-178B/C and ISO 26262 qualification. Since RapiTime does not rely on a processor model, it is usable for a lot of targets as long as they support a mechanism to extract execution traces.

The SWEdish Execution Time analysis tool (SWEET) [13] is a research prototype that offers best-case execution time (BCET), WCET and flow analysis. It uses a program representation called ALF (Artist Flow Analysis Language) in which either a binary or source code has to be converted for further processing. It implements a flow analysis of a given program to detect infeasible paths and loop bounds. The latter can be exported to aiT or RapiTime flow facts format enabling further analyses. A low-level analysis tool called low-sweet allows evaluating the WCET by its own.

The Open Tool for Adaptive WCET Analysis (OTAWA) [14] is a static analysis framework that allows modifying, extending, or implementing analysis tools by using the OTAWA API. One of the existing tools, called OWCET, evaluates the WCET of a given program by providing the executable binary file, program flow information (flow facts) as well as a processor description (OTAWA script). With this information, it automatically links the corresponding architecture loader module and performs the analyses.

Besides RapiTime with its measurement-based approach, all of the mentioned tools use binary based static program analysis, which requires knowledge of the processor architecture in order to perform a timing analysis for WCET estimation. In fact, implementing support for a new target system requires the implementation/adaption of elaborate analyses to this new architecture.

At the core therefore, are usually architecture description languages (ADLs) that describe the processor model. They have a wide range of application; a majority is the generation of target specific tools (e.g., compiler or simulator). They are also used for the development and rapid prototyping of application-specific instruction-set processors (ASIPs).

There are different ADLs available to create an instruction set simulator (e.g., EXPRESSION [18], LISA [19] and nML [20]), however each of them describe the instruction set architecture of a specific processor family without detailed information of the microcontroller. This makes it possible to use the simulator for every microcontroller with the chosen processor architecture. For timing analyses, there are some details like cache or pipeline behavior missing at this point; therefore, this information is provided via additional description by the developer or end user. Hardware ADLs are likewise known as processor or machine description language and are not only used for simulators but also for processor development [21, p. 2].

These architecture description languages are classified in

three content-based (structural, behavioral and mixed) as well as four objective-based categories (compilation, simulation, synthesis and validation). This classification allows developers to choose an ADL by either content (e.g., instruction-set description) or purpose (e.g., generation of an instruction-set simulator). Due to the fact that not all ADLs can describe the instruction-set behavior with a detailed timing model, only a few are usable for WCET analysis.

Modeling a processor based on the ARMv5 architecture using the ArchC ADL is described in [15]. They divided the implementation into three phases, starting with the choice of a suitable ADL, followed by constructing of a toolchain (e.g., compiler and simulator) for the architecture; and phase three, analyzing the executed instructions by the simulator using program patterns. Their aim was to evaluate often used instruction patterns that can be merged to a new complex instruction in order to increase the performance.

A retargetable software timing analyzer for WCET estimation using the EXPRESSION ADL for processor models is described in the work of [16]. For their case study, a MIPS processor including its instruction set architecture was modeled and evaluated.

An approach for generating instruction set simulators from an enhanced nML architecture description is presented in [17]. The tool generates a simulator usable for static analysis. For demonstration purposes, their tool was applied to an ARMv5 architecture implementation.

For the approach described here, we choose the Sim-nML/nMP formalism in order to create an ARMv7E-M instruction set simulator. Sim-nML is an extension of the nML formalism with the purpose to perform efficient simulations; and the Macro Preprocessor (nMP) extends the Sim-nML syntax (e.g., recursive macros or macro calls within macros) in order to simplify the implementation. OTAWA is the chosen platform for the WCET evaluation because it is expendable, allowing to integrate new architectures and analysis tools. The decision to use a static code analysis approach was based on the requirements of the R&D project, which is intended to extend the analysis environment with other static analyses (e.g., stack usage, control-flow graph) at a later stage. A more detailed description of this choice is given in Section IV.

IV. Implementation

The OTAWA framework features an architecture abstraction layer, enabling to use the analysis framework independently of the actual target platform. This layer applies an architecture plug-in as an interface to the corresponding architecture loader, which contains all relevant details for the instruction set simulation.

Figure 1 shows the structure of the OTAWA framework and highlights the mentioned concept of hardware abstraction by binding modules for the TriCore, PowerPC or ARMv5 architecture towards this layer. The OTAWA core links all modules including a set of analyses (e.g., data flow analysis), graph generation (e.g., control-flow graph), providing an abstract representation of the program as well as accessing an external ILP solver (e.g., lp_solve). From a programmer's perspective, the architecture loader together with a processor description in the form of an OTAWA script are necessary to support WCET analyses for a certain processor. These two

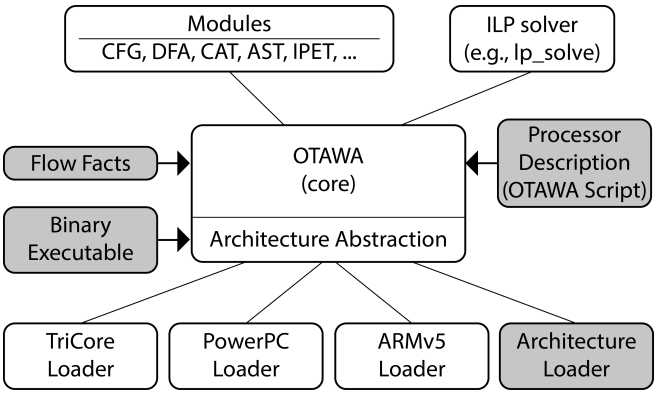

Figure 1. OTAWA Structure

components are highlighted on the right side in Figure 1. One can use the OTAWA framework to perform WCET analyses by providing the binary executable and the flow facts of a program, as shown on the figure's left side. An overview of the necessary implementation steps is given below:

1) Description of the instruction set architecture as loader module.
2) Implementation of an interface for the architecture abstraction layer in order to link the loader module to the OTAWA framework.
3) Creation of a script for the target processor.
4) Adaption and extension of the analysis tool if additional features are desired.
5) Verification of the implementation.

The next sub-sections describe our implementation of the mentioned steps in a generic manner. Starting with step one and two in Section IV-A, followed by the processor script in Section IV-B as well as two tools (MKFFX and OSWA) in Section IV-C and Section IV-D that are using the OTAWA framework for evaluating a programs WCET, flow facts and basic block statistics. Details of our concrete implementation are given in Section IV-E, and finally, the verification is explained in Section IV-F.

A. Architecture Loader

This module is the core part of the work presented in this paper and includes a description of the processor architecture in form of the instruction set. Most of the code was written in the Sim-nML/nMP language including information of the syntax, the binary representation and the semantics of each instruction from the architecture. The syntax is important for the disassembler output and control-flow graph because it is the representation of the assembler syntax. The image is used for linking the bits of a decoded instruction to the corresponding parameter (e.g., register or immediate value). Within the action part, the parameters from the image are used for describing the instruction's function. This means that calculations (e.g., shift or add), writing and reading registers, as well as updating flags is part of the action.

Beside the instructions themselves, there are registers, conditions, modes and exceptions within the Sim-nML/nMP part. Additionally, macros were defined to decrease the implementation effort and at the same time increase the readability. The rest of the implementation, containing auxiliary functions and algorithms, was written in the C-language. Afterwards, the

Generator of Libraries for Instruction Set Simulators (GLISS) was used to generate a C-library out of both implementation parts. This library along with the ARM module from OTAWA serve as input for the generation of the instruction set simulator, a so-called "architecture loader". In a final step, it was necessary to define the instructions kind (e.g., ALU or branch), target, semantic and used registers within the OTAWA ARM module in order to interpret each instruction correctly. All relevant architecture information for the work can be found in the corresponding architecture reference manual.

B. Processor Script

This section describes the implementation of processor characteristics as OTAWA script, so that applications targeting a specific microcontroller can be analyzed with the OTAWA framework. The script is written in XML format and consists of several files, each for one component of the processor. The separation is described by the following listing:

- Main: This file is usually named after the microcontroller and includes information about the used architecture. It links all parts of the platform description (e.g., memory) and allows to configure items to fine-tune the analysis. In addition, necessary analysis steps can be included, which are accessible through the OTAWA API (e.g., BB_TIME_FEATURE ensures that the execution time computation of each basic block has been performed).
- Memory: The processor's different memory banks with their properties are described in this file. A typical description of a memory bank includes a name, the start address and its size, followed by the type (e.g., FLASH or SRAM). Read/write latencies can be defined in order to set a number of cycles for accessing or writing the memory. This is especially relevant for external memories with high access times. Finally, one can specify if a memory is writable or cachable.
- Pipeline: This file describes the processor's pipeline as big picture because its complexity is in many cases not describable. Each stage is described by an ID, a name, a width defining the number of parallel processed instructions, a latency for multi-cycle operations and a type. The type is typical *fetch* for the very first stage (e.g., instruction fetch from memory) and *commit* for the very last stage to declare the exit. In between, there are either lazy stages (e.g., decode) waiting for a defined time as well as execution stages. It is possible to define functional units for the execution stages (e.g., arithmetic logic or floating-point unit), allowing to link certain types of instructions to them.
- Cache: The processor's caches are described in this part. It is possible to state data, instruction or unified caches, whereby the elements are all the same. Each configuration consists of a replacement policy (e.g., LRU or FIFO), the size of a cache block, the number of blocks in each set and the number of sets in the cache. Additionally, different levels of cache can be defined.

It is essential to create a script for any used microcontroller as already small differences can result in a WCET deviation.

C. Flow Facts Evaluation Tool

Since the exact control flow of a program depends on input data, it is impossible to make an estimation without program execution. So-called flow facts, include program flow information like maximum loop iteration counts (loop bounds) or recursion depths and are provided by the user. These details improve the precision of the analysis result; often they are necessary to evaluate a program's WCET at all. In cases where no explicit limitations (e.g., loop bound depends on input parameter) are given, either the user defines high but safe bounds (e.g., maximum value of the parameter's data type), which makes the WCET result inaccurate, or the estimation is infeasible. Defining the flow facts by hand is exhausting and imply a risk of incompleteness. Therefore, some analysis tools can automatically detect flow facts and save them into a respective file. The task of filling in missing information (e.g., boundary for a found loop) remains to be done prior to the evaluation of the WCET.

The introduced tool, called mkffx, generates flow facts in XML format by combining various input methods. First, it reads possibly existing flow facts from a given file and saves them in an internal representation, followed by analyzing the binary file to detect and record loops and other control information. Next, it will invoke the oRange tool [22], which analyzes loop bounds and extracts flow facts from the source code. This is an optional feature, since the source code is not available in every use case. Afterwards, the mkffx tool merges all results and outputs the flow facts. In this way, it reduces the necessary effort of describing them by hand because the combination of several inputs increases the rate of automatically detected loop bounds. Our mkffx tool extends the features of the mkff tool, which already comes with the OTAWA framework.

D. WCET Analysis Tool

Evaluating a programs WCET takes several analysis steps which can vary depending on the processors architecture. Figure 2 shows a typical scenario of a WCET estimation using OTAWA.

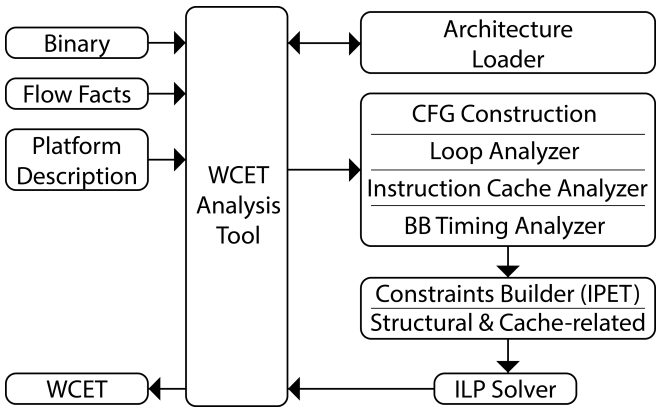

Figure 2. OTAWA Scenario

The very first step is to load the program under analysis in the form of the binary executable as well as its flow facts information and the corresponding processor script (platform description). Next, the program from the binary file is transformed into an internal representation using the architecture

loader and the CFG builder constructs its control-flow graph. Afterwards, analyses are applied starting with the loop analyzer which uses the loop boundaries from the provided flow facts. The platform description is taken into account to analyze the instruction caches behavior. With this information, the execution time of each basic block can be calculated. Sets of graph flow constraints (e.g., program flow and basic-block execution time bounds) are built for the implicit path enumeration technique (IPET) based calculation approach. These constraints are transformed into an integer linear programming problem with a goal function (WCET) and then solved using an external solver (e.g., lp_solve). In the end, the analysis tool can output the resulting WCET. The OTAWA framework includes all of the mentioned modules as shown in Figure 1.

The OTAWA Stack and Worst-case execution time Analysis (OSWA) tool combines several features into one application. Beside the two main functions derived from its name: stack usage evaluation and WCET analysis, it can generate a control-flow graph with various output kinds and creates a basic block timing statistic. The latter allows identifying the most time consuming basic blocks within a given function or code snippet. An additional feature is the calculation of a ratio between the time spent inside and outside the function, which can be used to find out how much time is spent in sub-functions. For this paper, the WCET analysis feature is the most important one. OSWA performs the analysis of a specified function from a given binary file by involving flow facts and a processor description (OTAWA script). Our OSWA tool extends the features of the owcet tool, which already comes with the OTAWA framework.

E. Specific Implementation for the Use Case

The goal of our work is to enable WCET analysis for software targeting ARM Cortex-M4 processors. For this reason, the implementation in Section IV-A was accomplished for the ARMv7E-M architecture. It features the Thumb-2 technology with both, 16 and 32 bit operations. This architecture loader is based on an existing ARMv5 loader because its 16 bit Thumb instructions are mostly equivalent with the ARMv7 technology.

The Infineon XMC4500-F100K1024 microcontroller features an ARM Cortex-M4 processor core and was chosen for further evaluation (see Section VI). Its characteristics were described in the form of an OTAWA script as presented in Section IV-B.

F. Verification

As described in Section IV-A, the architecture implementation is split in a Sim-nML/nMP description and code written in the C language. As a result of the build process, a C-library that contains both parts is generated. This entire implementation as well as Sim-nML/nMP parts were verified using simulation, code reviews, and disassembler output comparison. The verification of the C parts is completed with the following methods: model checking, static code analysis, and test drivers based on boundary value analysis and equivalence class partitioning. In addition, a plausibility check of the implementation was performed by comparing WCET results of selected test cases with measurements and results from another tool, as shown in the subsequent section.

V. Benchmarks

A comparison of WCET results was made between (1) OTAWA with the implemented ARMv7E-M architecture loader as well as the OSWA tool, (2) the Advanced Analyzer for ARM (A^3) version 14.04 from AbsInt GmbH, and (3) a measurement-based approach. Although A^3 only supports the ARM Cortex-M3 and not the ARM Cortex-M4 processor family, a comparison is possible because both are using the ARMv7 architecture with the Thumb2 instruction set. This behavior is valid as long as no ARM Cortex-M4 specific instructions (e.g., DSP extension) are used, otherwise, the executable would be different from the ARM Cortex-M3 version and not compatible with A^3. The measurement-based WCET analysis uses a manually written test driver with input parameters that cause a worst-case scenario. One can identify the worst-case behavior by hand for the chosen benchmarks because of their rather simple program flow; however, it would be a challenge to cause the worst-case behavior for more complex applications. By toggling an I/O pin before and after a certain program code, one can record its execution time using an oscilloscope or logic analyzer. The Infineon XMC4500 is the microcontroller of choice for the measurements, which operates at a frequency of 120 MHz. For the purpose of comparing the measured WCET with the analysis tools, all results are converted into a time unit (based on the processors clock rate) and are recorded in μs rather than in cycles. Table II shows the WCET evaluation results of 4 test cases. The first test case (For-If-Add) is a very basic example, only containing a loop with an if-else construct and some additions. The functions Factorial and Fibonacci are clearly assigned to a known algorithm by their names; however, their results are only valid for the given input scenario (e.g., Fibonacci number and factorial of 50). The fourth and last test case contains a preliminary implementation of the resolution advisory (RA) component from the Traffic Alert and Collision Avoidance System (TCAS). Its purpose is to issue climb or decent directives in case of conflicting aircrafts [23].

Table II. WCET results comparison for the test cases.

Test Case	OTAWA	AbsInt A^3	Measurement
For-If-Add	18.16 μs	18.07 μs	17.98 μs
Factorial	23.33 μs	23.72 μs	22.82 μs
Fibonacci	14.35 μs	8.09 μs	8.07 μs
TCAS	8.71 μs	7.82 μs	6.49 μs

Overall, the results show that the measurement-based approach leads in every case to a lower WCET. This circumstance is very important because otherwise the WCET analysis tools would evaluate a wrong or underestimated result which cannot be used for safety-critical real-time applications or generally for verifying timing constraints as it can lead to software misbehavior that might have a catastrophic impact. In all four test cases, the WCET evaluations by OTAWA and A^3 deliver a safe upper bound, meaning it is above the real value and therefore trustworthy. Additionally, the gap between the real WCET and the estimated ones are especially in the first two test cases minor. Since the goal is to get as close as possible to the reality, this tight output is desirable. The For-If-Add test case shows, that A^3 delivers a value 0.09 μs above the measured one but also 0.09 μs tighter than OTAWA. At the second function, factorial, OTAWA estimates a 0.39 μs

 ISBN: 978-1-61208-492-3

tighter result than A^3 and 0.51 μs above the measured WCET. Next, the Fibonacci algorithm shows that OTAWA calculates a weaker WCET bound, whereas A^3 estimates a extremely tight value. Finally, the TCAS test case challenges the analysis tools, as there are many sub-routine calls and thereby, initiating several pipeline refills with a variable duration depending on things like the width of the target instruction. In general, the different results between both tools can be caused by deviating analysis techniques or the usage of other integer linear problem solver.

In summary, it can be stated that the OTAWA Framework with both, the ARMv7E-M loader and OSWA can compete with a commercial tool by delivering safe and mostly tight results.

VI. Use Case

The use case shows a software part of an unmanned aerial vehicle (UAV), more specific, a quad-copter. Since UAV software contains plenty of software components, it is important to ensure that all of them have enough resources to do their tasks in order to satisfy any deadline, hence, to guarantee safe operation. This section discusses one functionality exemplary, though the entire UAV software needs to be analyzed. The goal is that the quad-copter can remain static in the air at given height between 20 and 150 centimeters. This task consists of a measurement unit to detect the current height and pass on the distance information to the engine task which uses control algorithms for adapting the current height to a given value. Stabilization is performed by the engine control using a triple-axis gyroscope.

This use case describes timing analysis of the distance measuring and evaluation software. An infrared proximity sensor is used to measure the distance between the quad-copter and ground. It delivers an analog output with a nonlinear distance measuring characteristics. The sensors characteristic is approximated and expressed as mathematical equation. As a result, it is necessary to use an analog-digital converter to read in the latest sensor value and calculate the current distance according to its characteristic.

Listing 3 shows the source code of the sensor read function and is described in this paragraph. One can pass the number of measurements as argument to the function. If the parameter is zero or one, only a single value will be measured (see *GetADCValues* function). Otherwise, the given size equals the number of measurements from which the mean value will be generated (see *mean* function). In both cases, the distance is calculated using the *evaluateDistance* function that implements the sensor characteristics as formula. The functions *GetADCValues* and *mean* each contain a loop, which bound depends on the given size. In Addition, the analog-digital conversion executed within the *GetADCValues* function takes several cycles, depending on the ADC configuration (e.g., conversion width of 12 bits or the divider factor for the analog internal clock). For the WCET analysis, this delay is considered and implemented as busy-waiting loop.

Although this software component is a relatively small one, its importance is beyond debate because it delivers information about the current height and wrong or no up-to-date data could lead to an accident. Therefore, it is necessary to estimate the timing behavior of this software component by evaluating its worst-case execution time.

```
uint16_t readSensor(uint16_t size)
{
    uint16_t distance;

    if(size < 2)
    {
        uint16_t adc_value;
        GetADCValues(&adc_value, 1);
        distance = evaluateDistance(adc_value);
    }
    else
    {
        uint16_t adc_values[size];
        uint16_t temp;
        GetADCValues(adc_values, size);
        temp = mean(adc_values, size);
        distance = evaluateDistance(temp);
    }
    return distance;
}
```

Figure 3. Distance measuring source code

The process of analyzing its worst-case timing behavior start with the binary file of the program, shown in Listing 3, by identifying and evaluating all loop bounds using the mkffx tool. In the use case, a mean value from 32 analog-digital conversions is used for the height estimation. Therefore, the loop bounds within the *GetADCValues* and *mean* function need to be 32. After the generated flow facts are checked, the OSWA tool can be executed to estimate the functions WCET.

Table III. WCET results of the use case.

Function	OTAWA
readSensor	68.58 μs
GetADCValues	62.22 μs
mean	5.20 μs
evaluateDistance	0.425 μs

Table III shows the worst-case execution time evaluation results of the functions from the use case. All results are estimated for the Infineon XMC4500 microcontroller operating with a clock rate of 120 MHz. The distance calculation, including all sub-routine calls, takes 68.58 μs in the worst-case. A measurement was performed where an execution time of 67.37 μs was recorded, giving the information that the WCET analysis is safe and tight. The results of the sub-routines are revealing where most of the time is spent. In this case, recording the ADC values takes the majority and evaluating the distance the least of the time. This ratio depends on the number of analog-digital conversions taken into account for one distance calculation. Finally, the main statement of the results is emphasizing the necessary time budget of 68.58 μs for the entire task.

VII. Conclusion

This paper elaborates on the implementation of a processor model for worst-case execution time analysis. The presented approach integrates with the open-source framework OTAWA and, hence, can serve as guide for similar efforts.

It starts with the architecture implementation, which is split into a Sim-nML/nMP model and an OTAWA script, resulting in a behavioral architecture description with timing information of operations in order to generate a cycle-accurate instruction set simulator. In particular, we choose the ARMv7E-M architecture that is used by ARM Cortex-M4 devices.

 ISBN: 978-1-61208-492-3

The resulting toolset was evaluated by way of a benchmark in order to underline its save and tight WCET calculation.

Our lesson learned is that the seamless and correct description of a processor model is exhaustive; further, the implementation quality depends on the correctness of the architectures datasheet. We experienced that the OTAWA framework is capable of much more than WCET analysis, because the existing analyses can be adapted to fulfill own requirements or purposes.

In summary, the presented approach enables static binary analysis of a program targeting an implemented architecture. This allows to evaluate information of the application like the WCET, which can be used for creating a statement regarding possible violations of deadlines or task scheduling in real-time systems.

ACKNOWLEDGMENT

This work has been conducted in the context of the public funded R&D project Software Analysis Toolbox managed by the Vienna City Council MA23.

REFERENCES

[1] C. Ferdinand, "Worst case execution time prediction by static program analysis," In 18th International Parallel and Distributed Processing Symposium, IPDPS'04, 2004, pp. 125a.

[2] H. Cass, H. Ozaktas, and C. Rochange, "A Framework to Quantify the Overestimations of Static WCET Analysis," In 15th International Workshop on Worst-Case Execution Time Analysis, WCET'15, 2015, pp. 1–10.

[3] R. Kirner, P. Puschner, and I. Wenzel, "Measurement-based worst-case execution time analysis," In IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, SEUS'05, 2005, pp. 7–10.

[4] F. Guet, L. Santinelli, and J. Morio, "On the Reliability of the Probabilistic Worst-Case Execution Time Estimates," In Proceedings of the Embedded Real-time Software and Systems, ERTS'16, 2016, pp. 758–767.

[5] X. Jean, S. Girbal, A Roger, T. Megel, and V. Brindejonc, "Safety considerations for WCET evaluation methods in avionic equipment," In IEEE/AIAA 34th Digital Avionics Systems Conference, DASC'15, 2015, pp. 7A4–1 to 7A4–15.

[6] M. Paolieri and R. Mariani, "Towards functional-safe timing-dependable real-time architectures," In IEEE 17th International On-Line Testing Symposium, IOLTS'11, 2011, pp. 31–36.

[7] S. Fischmeister and I. Lee, "Temporal Control in Real-Time Systems: Languages and Systems," In Handbook of Real-Time and Embedded Systems, 2007, pp. 10–1 to 10–18. CRC Press.

[8] F. Stappert, A. Ermedahl, and J. Engblom, "Efficient longest executable path search for programs with complex flows and pipeline effects," In Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES '01, 2001, pp. 132–140.

[9] A. Colin and G. Bernat, "Scope-tree: a program representation for symbolic worst-case execution time analysis," In 14th Euromicro Conference on Real-Time Systems, ECRTS '02, 2002, pp. 50–59.

[10] Y. T. S. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration," In Design Automation, DAC '95, 1995, pp. 456–461.

[11] N. Holsti and S. Saarinen, "Status of the Bound-T WCET tool," In 2nd Int. Workshop on Worst-Case Execution Time Analysis, WCET02, 2002, pp. 36–41.

[12] G. Bernat et al., "Identifying Opportunities for Worst-case Execution Time Reduction in an Avionics System," Ada User Journal, Volume 28, Number 3, 2007, pp. 189–194.

[13] J. Gustafsson, A. Ermedahl, C. Sandberg, and B. Lisper, "Automatic Derivation of Loop Bounds and Infeasible Paths for WCET Analysis Using Abstract Execution," In 27th IEEE International Real-Time Systems Symposium, RTSS'06, 2006, pp. 57–66.

[14] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat, "Otawa: An open toolbox for adaptive wcet analysis," In Software Technologies for Embedded and Ubiquitous Systems, SEUS'10, 2010, pp. 35–46.

[15] H. Arora, A. Gupta, R. Singhai, and D. Purwar, "Design space exploration of risc architectures using retargetability," In VLSI Systems, Architecture, Technology and Applications, VLSI-SATA'15, 2015, pp. 1–3.

[16] X. Li, A. Roychoudhury, T. Mitra, P. Mishra, and X. Cheng, "A retargetable software timing analyzer using architecture description language," In Design Automation Conference, ASP-DAC'07, 2007, pp. 396–401.

[17] T. Ratsiambahotra, H. Cassé, and P. Sainrat, "A versatile generator of instruction set simulators and disassemblers," In Symposium on Performance Evaluation of Computer & Telecommunication Systems, SPECTS'09, 2009, pp. 65–72.

[18] P. Grun et al., "Expression: An ADL for System Level Design Exploration," Technical Report TR 98-29, University of California, Irvine, USA, 1998.

[19] V. Zivojnovic, S. Pees, and H. Meyr, "Lisa-machine description language and generic machine model for hw/sw co-design," In VLSI Signal Processing, IX, 1996, pp. 127–136.

[20] A. Fauth, J. Van Praet, and M. Freericks, "Describing instruction set processors using nml," In European Design and Test Conference, ED TC'95, 1995, pp. 503–507.

[21] P. Mishra and N. Dutt, "Processor Description Languages: Applications and Methodologies," vol. 1, chapter 1, Morgan Kaufmann, 2008, ISBN: 978-0-12-374287-2.

[22] M. Michiel, A. Bonenfant, C. Ballabriga, and H. Cassé, "Partial Flow Analysis with oRange," In International Symposium on Leveraging Applications, ISoLA'10, Part II, Springer, 2010, pp. 479–482.

[23] U.S. Department of Transportation, Federal Aviation Administration, "Introduction to TCAS II Version 7.1," 2011.

 ISBN: 978-1-61208-492-3

Expurgated Codes for Detecting Jamming in Multi-level Memories

Yaara Neumeier
Faculty of Engineering
Bar-Ilan University
Email: yaara.neumeier@biu.ac.il

Osnat Keren
Faculty of Engineering
Bar-Ilan University
Email: osnat.keren@biu.ac.il

***Abstract*—Robust q-ary codes can efficiently detect jamming in multilevel memories when q is a power of two. When q is not a power of two, a binary information word has to be converted and encoded into a q-ary codeword. This conversion expurgates the code; some of the q-ary codewords are never used. Unless properly designed, expurgation can significantly degrade the efficiency of the code in terms of its error detection capability. This work presents a q-ary robust Quadratic-Sum code for arbitrary q's and analyzes the error masking probability of the expurgated code when applied to multilevel memories. It is shown that by wisely designing the converter, this degradation can be minimized, and in some cases, the expurgated code's efficiency can be superior to the one of the original code. This work suggests how to construct a converter to optimize code properties.**

Index Terms*—Robust codes; Multi-level Memories; Jamming attacks; Hardware security.

I. Introduction

Memory arrays are prone to jamming attacks [1], where an adversary injects faults into the memory to alter a stored value. The injected fault manifests itself as an additive error of an arbitrary multiplicity; i.e., any number of bits may be flipped or distorted. [2]. Fault injection can be executed, for example, using variations on voltage, temperature, white light, laser, ion beams, etc. An attacker can inject faults into the memory to change its content and then acquire information about the system by analyzing its behavior [2].

Several countermeasures to jamming attacks on memories have been proposed [2][3]. For example, one approach to protect memories is to implement intrusion detection mechanisms based on active protection using tamper-proof box and sensors to make the device physically inaccessible. Since different sensors are used against different injection methods, this method becomes expensive and inappropriate for simple, small devices. Moreover, it is powerless against new types of attacks that were not considered by the designers. Furthermore, internal information about the design may help the attacker bypass this protection. An alternative approach is to detect the manifestation of the fault as an error using error detecting codes.

Classic coding theory addresses the problem of the *reliability* of information transmitted over a noisy channel or stored in storage media. In classic coding theory, the errors are assumed to be random with a relatively small probability. Consequently, a reliability oriented code should protect the system from a small number of random errors (small multiplicity). Many known codes designed for reliability (such as the parity bit code, Hamming code, BCH codes, etc) are linear [4], however; in linear codes, all the errors that are codewords are never detected. As a result, reliability oriented codes cannot be used to provide *security* against an attacker that can inject any error.

Jamming can be detected by nonlinear robust codes capable of detecting any non-zero error. The efficiency of these codes is measured in terms of their error masking probability $\overline{Q}_{\mathcal{M}} = \max_{\mathbf{e}\neq 0} Q(\mathbf{e})$ where $Q(\mathbf{e})$ is the probability that an error $\mathbf{e}$ is masked by codewords in $\mathcal{C}$. This probability depends on the probability mass function of the codewords; that is,

$$Q(\mathbf{e}) = \sum_{\mathbf{c},\mathbf{c}+\mathbf{e}\in\mathcal{C}} p(\mathbf{c}),$$

where $p(\mathbf{c})$ is the probability that $\mathbf{c} \in \mathcal{C}$ is used.

The Quadratic-Sum (QS) code [5] is a nonlinear q-ary high-rate robust code of length n and dimension k defined over a finite field, i.e., for a q that is a power of a prime. When all the codewords are equally likely to occur, the code is an optimum code, and its error masking probability equals $\overline{Q}_{\mathcal{C}} = q^{-(n-k)}$ [5]. If these conditions are not fulfilled, the performance of the code may significantly degrade [6].

The encoding complexity of a binary QS code is relatively low with respect to other robust codes (e.g., the codes in [7][8] which involve computations over finite fields of high order); its k information symbols are treated as $2s$ symbols from $\mathbb{F}_2^r$ and its single redundant symbol x_{2s+1} is the sum $\sum_{i=1}^{s} x_{2i-1}x_{2i}$ over $\mathbb{F}_2^r$. This simple structure makes the code an attractive countermeasure to jamming in binary and q-ary multilevel memories, where q is a power of two.

However, in some cases, the code's alphabet size is not a power of two. Note that the number of levels in a multilevel memory, l, may be a power of two. Nevertheless, the code's alphabet size q may be smaller than l. For example, in Write-Once-Memory codes and rank-modulation codes the alphabet size is smaller than the number of levels to enable several write cycles to the same address before block-erasure. As far as we know, all known robust codes ([5][7][8][9]) are defined over a finite field, i.e., where q is a power of a prime, and

 ISBN: 978-1-61208-492-3

cannot be used in the case where the number of states is not a power of a prime.

Another problem that arises when the code's alphabet size is not a power of two is that each binary information word has to be converted to a q-ary word by a dedicated conversion circuit [10]. A conversion circuit maps a binary vector of length k_2 to a q-ary vector of length k_q. A conversion circuit is constructed from sub-blocks, denoted DCC_i. The input of each sub-block is a binary w_2-bit vector, and its output is a q-ary vector of length w_q. The values w_2 and w_q are chosen such that $w_q = \lceil w_2 \log_q 2 \rceil$. A schematic illustration of a multilevel memory with a conversion circuit is shown in Figure 1. Since $q^{w_q} < 2^{w_2}$, some of the codewords of the (original) q-ary code $\mathcal{C}$ are never used. When these unused words are chosen arbitrarily, the error masking probability of the expurgated code, denoted by $\mathcal{M}$, can become higher than the error masking probability of the original code.

Robust codes over finite fields for a non-uniform distribution of codewords were discussed in [6][11][12]. In [6], the authors showed that when *most of the codewords* appear with low probability, which is the case for some Final State Machines (FSMs), it is possible to avoid the worst case scenario by pre-mapping the information word before the encoding. In [12], a general approach for mapping the most probable codewords to a predefined set was suggested. In [11] the authors dealt with the non-uniform characteristics of FSMs using randomized masking. Another way to cope with a non-uniform distribution of codewords is by embedding randomness [13][14]; these codes are also defined over finite fields. However, since the random symbols are an integral part of the codeword, their rate is lower than the rate of (deterministic-encoding) robust codes such as the QS and the Punctured-Cubic/Square in [7][8]. These solutions are appropriate for applications where a small portion of the states appear with high probability; they are less suitable for applications such as multilevel memories where some words never occur and other words appear with uniform probability.

This paper expands the QS construction to codes over integer rings, proves its robustness, and examines the security related implications of applying expurgated codes on q-ary memory systems with data conversion circuits in cases where q is not a power of a prime. It is shown that by choosing $\mathcal{M}$ properly, the practical error masking probability $\overline{Q}_{\mathcal{M}}$ may be even better than $\overline{Q}_{\mathcal{C}}$. The main ideas and results presented in this paper are the following:

- A QS-based code $\mathcal{C}$ is robust over rings.
- The maximal error masking probability $\overline{Q}_{\mathcal{M}}$ of code $\mathcal{M}$ is bounded by
$$\frac{(2|\mathcal{M}| - |\mathcal{C}|)}{p_1|\mathcal{M}|} \leq \overline{Q}_{\mathcal{M}} \leq \frac{|\mathcal{C}|}{p_1|\mathcal{M}|} < 2\overline{Q}_{\mathcal{C}}, \quad (1)$$
where p_1 is the smallest divisor of q. Since $p_1 \geq 2$, an expurgated code is robust; it can detect any nonzero error with a probability greater than zero.
- If $p_1 = 2$ there exists an expurgated code $\mathcal{M}$, which provides a smaller error masking probability, i.e., $\overline{Q}_{\mathcal{M}} < \overline{Q}_{\mathcal{C}}$. If $p_1 \neq 2$ and
$$(p_1 - 1)k_q + 1 \leq |\mathcal{C}| - |\mathcal{M}| \quad (2)$$
there exists an $\mathcal{M}$ with $\overline{Q}_{\mathcal{M}} < \frac{|\mathcal{C}|}{p_1|\mathcal{M}|}$.
- A code construction for $\mathcal{M}$ which minimizes $\overline{Q}_{\mathcal{M}}$ in cases where $p_1 = 2$ for a given set of parameters, and has
$$\frac{(2|\mathcal{M}| - |\mathcal{C}|)}{p_1|\mathcal{M}|} \leq \overline{Q}_{\mathcal{M}} \leq \overline{Q}_{\mathcal{C}},$$
is presented.

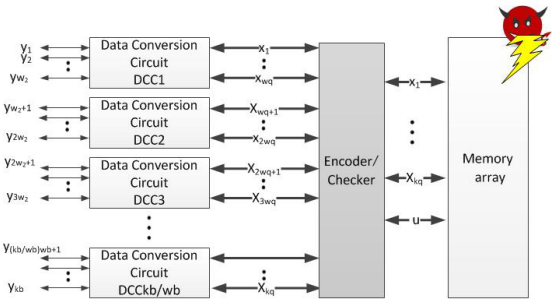

Figure 1. Multilevel memory system with data conversion circuit, protected by encoder and a checker.

The rest of this paper is organized as follows. Section II defines and analyzes the Quadratic-Sum code for a general q. Section III presents the expurgated code and the security problem that arises when applying the codes to q-ary memories where q is not a power of two. Then, lower and upper bounds on the error masking probability are presented. Section IV suggests how to choose $\mathcal{M}$ in cases where $p_1 = 2$ to minimize its error masking probability and Section V concludes the paper.

II. The Extended QS code

Notations: Regular lowercase letters are used to represent scalars. Boldface lowercase letters are used to denote row vectors, e.g., $\mathbf{x} = (x_1, \ldots, x_n)$ is a vector of length n, where $w_H(\mathbf{x})$ denotes the Hamming weight of $\mathbf{x}$. Double stroke capital letters are used to denote algebraic structures, e.g., $\mathbb{F}_q$ is a finite field with q elements. Regular uppercase letters are used to represent sets, e.g., S, where $|S|$ is the number of elements in S. Calligraphic capital letters are used to denote codebooks, e.g., $\mathcal{C}$.

Consider a multilevel memory whose levels are mapped into symbols in an alphabet of size q. In this paper, we refer to such a memory as a q-ary memory. The set of q symbols with addition and multiplication form an algebraic structure. If $q = p^t$ and p is prime, the algebraic structure is a finite field $\mathbb{F}_q$; otherwise, it is a ring $\mathbb{R}_q$ in which operations are computed modulo q. To simplify the text, when it is clear from the context, we denote the algebraic structure by $\mathbb{Z}_q$, and denote addition and subtraction by the symbols $\oplus$ and $\ominus$, respectively.

 ISBN: 978-1-61208-492-3

Known robust codes are defined over a finite field, i.e., the size of the alphabet, q, is a power of a prime. If the number of different states (voltage levels) that a memory cell can have in each write cycle is not a power of a prime, a robust code over a ring is required. Note that the computation of all these known robust codes over finite fields involves multiplication. However, in the case of a ring, there are elements in the ring with no multiplicative inverse, which may affect the analysis and the resulting error masking probability. In this section we introduce an extension of the QS code. The resulting code is a robust code over a ring. The maximal error masking probability of the extended code is different (higher) than the maximal error masking probability of the original QS code over a finite field.

The QS code is defined in [5] for the case where q is a power of a prime (PoP). The number of redundancy symbols in [5] is $r \leq k$. The code can be extended for q's which are not necessarily PoPs as follows:

Construction 1. *Let $q = \prod_{i=1}^{v} {p_i}^{t_i}$ where $p_i < p_{i+1}$. Let $k = 2sr$, where $r = 1$ if q is not a PoP. Let $\mathbf{x} = (x_1, x_2, \ldots, x_{2s})$ where $x_i \in \mathbb{Z}_q^r$ for $1 \leq i \leq 2s$. The code QS code is*

$$\mathcal{C} = \{(\mathbf{x}, \mathbf{u}) : \mathbf{x} \in \mathbb{Z}_q^{2s}, \mathbf{u} = \sum_{i=1}^{s} x_{2i-1}x_{2i} \in \mathbb{Z}_q^r\}.$$

Note that when q is not a PoP, we take $r = 1$ since a larger r cannot improve the code's efficiency. To simplify the notation, from here on, unless otherwise stated, q is not a PoP. The case where q is a PoP can be viewed as subcase of the general case with $p_1 = q$.

Let $\mathbf{e} = (\mathbf{e}_x, e_u)$ be an error vector, where $\mathbf{e}_x = (e_{x_1}, \ldots e_{x_{2s}}) \in \mathbb{R}_q^{2s}$ and $e_u \in \mathbb{R}_q$. The error is masked by a codeword $\mathbf{c}$ if $(\mathbf{x} \oplus \mathbf{e}_x, u \oplus e_u) \in \mathcal{C}$. In other words, the error masking equation of the code is

$$\sum_{i=1}^{s}(x_{2i-1} \oplus e_{x_{2i-1}})(x_{2i} \oplus e_{x_{2i}}) = \sum_{i=1}^{s} x_{2i-1}x_{2i} \oplus e_u. \quad (3)$$

Equivalently,

$$\mathbf{a}\mathbf{x}^T = b \quad (4)$$

where $\mathbf{a} \in \mathbb{R}_q^{2s}$ and $b \in \mathbb{R}_q$ are

$$a_i = \begin{cases} e_{x_{i+1}} & \text{if } i \text{ is odd} \\ e_{x_{i-1}} & \text{if } i \text{ is even} \end{cases}, \text{ and } b = e_u \ominus \sum_{i=1}^{s} e_{x_{2i-1}}e_{x_{2i}}.$$

Let $B(\mathbf{a})$ be the set

$$B(\mathbf{a}) = \{b | \exists \mathbf{x} : \mathbf{a}\mathbf{x}^T = b\}.$$

Clearly, $B(\mathbf{a}) = B(\mathbf{e}_x)$. To analyze which elements are in $B(\mathbf{a})$, it is convenient to use the greatest common divisor (gcd) over a set of nonzero integers; define $g(\mathbf{a}) \in \mathbb{R}_q$ as

$$g(\mathbf{a}) = \gcd(\{a_i | a_i \neq 0\} \cup \{q\}).$$

The set $B(\mathbf{a})$ contains all the multiples of $g(\mathbf{a})$ modulo q. Therefore $|B(\mathbf{a})| = \frac{q}{g(\mathbf{a})}$. In addition, for all $1 \leq i \leq 2s$, a_i is also in $B(\mathbf{a})$. For example, if $q = 6$, $r = 1$, $k = 2$ and $\mathbf{a} = (0, 4)$, then $g(\mathbf{a}) = 2$, $B(\mathbf{a}) = \{0, 2, 4\}$ and $|B(\mathbf{a})| = 3$.

Property 1. *Let $\mathbf{a} \in \mathbb{R}_q^{2s} \setminus \{\mathbf{0}\}$. Then, (4) has $q^k|B(\mathbf{a})|^{-1}$ solutions if $b \in B(\mathbf{a})$ and 0 solutions otherwise.*

For uniformly distributed codewords, the number of solutions of (4) for a given $\mathbf{e}$ defines the error masking probability;

Theorem 1. *Let q not be a PoP. Let $\mathcal{C}$ be a QS code where the codewords in $\mathcal{C}$ are uniformly distributed. The error masking probability of $\mathcal{C}$ for any nonzero error $\mathbf{e}$ is $Q(\mathbf{e}) = \frac{1}{|B(\mathbf{a})|}$ if $b \in B(\mathbf{a})$, and $Q(\mathbf{e}) = 0$ otherwise. In particular, the maximal error masking probability of the QS code is $\overline{Q}_{\mathcal{C}} = 1/p_1$.*

The set of all $\mathbf{e}_x$'s can be divided into subsets according to their error masking probability. Let $\overline{E}_x$ be the set of $\mathbf{e}_x$'s that have the maximal error masking probability. Any $\mathbf{e}_x \in \overline{E}_x$ can be written as $\mathbf{e}_x = \frac{q}{p_1}\tilde{\mathbf{e}}_x$ where $\tilde{\mathbf{e}}_x \in \mathbb{Z}_{p_1}^k$. Since $B(\mathbf{e}_x) = B(\mathbf{e'}_x)$ for all $\mathbf{e}_x, \mathbf{e'}_x \in \overline{E}_x$, there are $({p_1}^k - 1)p_1$ distinct error vectors $\mathbf{e}$ that maximize the error masking probability.

Example 1. *Consider the case where $q = 6$, $k = 2$, and $r = 1$. In this case, the set $\overline{E}_x = \{03, 30, 33\}$ and $B(\overline{E}_x) = \{0, 3\}$. Each one of the errors $030, 033, 300, 303, 330, 333$ has an error masking probability $Q(e) = 0.5 = \overline{Q}$.*

III. THE EXPURGATED CODE

Consider a k_2-bit binary word to be stored in a q-ary memory array where q is not a power of two. For converting the binary word into a q-ary word of length k_q, the k_2 bits are divided into blocks of w_2 bits which are then mapped into blocks of w_q q-ary symbols; whereas,

$$\lfloor \frac{k_2}{w_2} \rfloor w_q + \lceil (k_2 \mod w_2) \log_q 2 \rceil \leq k_q \leq \lceil \frac{k_2}{w_2} \rceil w_q.$$

For simplicity, we assume that $\frac{k_2}{w_2}$ is an integer (however, our results equally apply to non integers). Since q is not a power of two, some of the q-ary vectors are never used; denote by D_w the set of the combinations over in $\mathbb{F}_q^{w_q}$ that are never used,

$$|D_w| = q^{w_q} - 2^{w_2} < 2^{w_2}.$$

Denote by $D \subseteq \mathbb{Z}_q^{k_q}$ the set of q-ary vectors of length k_q that never occur at the output of the converter, and by $M = \mathbb{Z}_q^{k_q} \setminus D$ the set of vectors that can appear at the output of the converter. Each vector in D corresponds to a codeword in $\mathcal{C}$ that is never used; denote by $\mathcal{D} \in \mathcal{C}$ the set of unused codewords, and by $\mathcal{M} = \mathcal{C} \setminus \mathcal{D}$ the expurgated code,

$$\begin{aligned} |\mathcal{M}| &= |M| = 2^{k_2} > |\mathcal{C}|/2, \\ |\mathcal{D}| &= |D| = (q^{k_q} - 2^{k_2}) < |\mathcal{C}|/2. \end{aligned} \quad (5)$$

It is assumed that the codewords in $\mathcal{M}$ are uniformly distributed.

Clearly the error masking probability of $\mathcal{M}$ may be different from the error masking probability of $\mathcal{C}$. Denote by $R(\mathbf{e})$ the

 ISBN: 978-1-61208-492-3

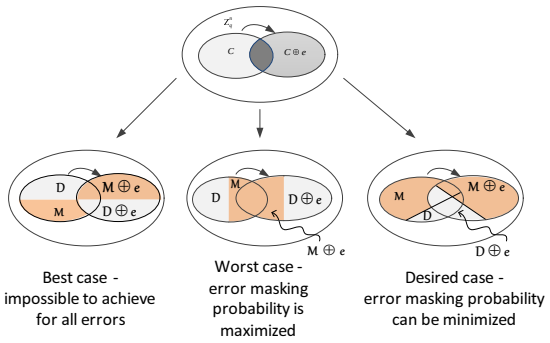

Figure 2. Three types of errors in expurgated codes.

number of codewords that mask the error vector $\mathbf{e}$,

$$R_{\mathcal{C}}(\mathbf{e}) = |\{\mathbf{c}|\mathbf{c} \in \mathcal{C} \text{ and } \mathbf{c} \oplus \mathbf{e} \in \mathcal{C}\}|.$$

For uniformly distributed codewords we have, $Q(\mathbf{e}) = R_{\mathcal{C}}(\mathbf{e})/|\mathcal{C}|$. Denote by $\Lambda_{\mathcal{C}_1,\mathcal{C}_2}(\mathbf{e})$ the cross-correlation from a code $\mathcal{C}_1$ to a code $\mathcal{C}_2$; i.e,

$$\Lambda_{\mathcal{C}_1,\mathcal{C}_2}(\mathbf{e}) = |\{\mathbf{c}|\mathbf{c} \in \mathcal{C}_1 \text{ and } \mathbf{c} \oplus \mathbf{e} \in \mathcal{C}_2\}|.$$

Since $\mathcal{C} = \mathcal{M} \cup \mathcal{D}$ and $\mathcal{M} \cap \mathcal{D} = \emptyset$ the autocorrelation of the code $\mathcal{C}$ can be rewritten as

$$R_{\mathcal{C}}(\mathbf{e}) = R_{\mathcal{M}}(\mathbf{e}) + \Lambda_{\mathcal{M},\mathcal{D}}(\mathbf{e}) + \Lambda_{\mathcal{D},\mathcal{M}}(\mathbf{e}) + R_{\mathcal{D}}(\mathbf{e}). \quad (6)$$

Figure 2 illustrates the contribution of each component in (6) to $R_{\mathcal{C}}(\mathbf{e})$ for three types of errors. The codewords of expurgated code $\mathcal{M}$ and its shifted set $(\mathbf{e}+\mathcal{M})$ are shown in red, and the codewords that correspond to $\mathcal{D}$ appear in light gray. $R_{\mathcal{M}}(\mathbf{e})$ is the number of codewords in the intersection of the two red areas. The best case is shown on the left hand side of the figure. Since $R_{\mathcal{M}}(\mathbf{e})$ is the autocorrelation of code $\mathcal{M}$, the best case is where $R_{\mathcal{M}}(\mathbf{e}) = 0$, however, it is impossible to achieve this for all errors, since for each two codewords $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{M}$ there is an error vector $\mathbf{e}$ such that $\mathbf{c}_1 \oplus \mathbf{e} = \mathbf{c}_2$. The worst case is where $R_{\mathcal{M}}(\mathbf{e})$ is maximized. The desired case is where the maximal value of $R_{\mathcal{M}}(\mathbf{e})$ is minimized.

A standard checker of a separable code uses the k_q information symbols that are read from the memory to compute the expected redundant symbols. If the computed value matches the value of the r_q symbols stored in memory, the checker declares that no error has occurred, otherwise, it raises a flag. Such a checker masks an error $\mathbf{e}$ with a probability

$$\frac{R_{\mathcal{M}}(\mathbf{e}) + \Lambda_{\mathcal{M},\mathcal{D}}(\mathbf{e})}{|\mathcal{M}|}. \quad (7)$$

which may be higher than $\overline{Q}_{\mathcal{M}}$. This problem can be avoided if the checker also verifies that the received codeword belongs to M (i.e., it verifies that the information vector is a legal output of the converter). Now the error masking probability is reduced to the true error masking probability of $\mathcal{M}$, i,e,

$$Q_{\mathcal{M}}(\mathbf{e}) = \frac{R_{\mathcal{M}}(\mathbf{e})}{|\mathcal{M}|}.$$

In what follows we assume that the latter checker is used.

A. An upper bound on the error masking probability

Theorem 2. *The expurgated code $\mathcal{M}$ is robust. Its error masking probability is upper bounded by*

$$\overline{Q}_{\mathcal{M}} \leq \frac{|\mathcal{C}|}{p_1|\mathcal{M}|} < \frac{2}{p_1} = 2\overline{Q}_{\mathcal{C}}. \quad (8)$$

The error masking probability of the expurgated code depends on the choice of the set $\mathcal{M}$; in particular, $Q_{\mathcal{M}}(\mathbf{e})$ may be larger, smaller, or identical to the error masking probability of the original code. The following example demonstrates how sensitive the error masking probability is to the choice of $\mathcal{M}$.

Example 2. *Consider the case where $k_2 = w_2 = 6$ bits of information are converted to $k_q = w_q = 2$ symbols over alphabet $q = 10$, and are protected by a single redundant symbol. In this case, $|\mathcal{C}| = 10^2$, $|\mathcal{M}| = 2^6$ and $|\mathcal{D}| = |\mathcal{C}| - |\mathcal{M}| = 36$. The maximal error masking probability of the original QS code $\mathcal{C}$ is $\overline{Q}_{\mathcal{C}} = 0.5$.*

Let $D = \{10-19, 30-39, 50, 51-59, 70-75\}$. Consider the error vector $\mathbf{e} = 050$; the corresponding parameters are $\mathbf{a} = 50$ and $b = 0$. Note that for each $x \in D$, $a\mathbf{x}^T = 5x_1 \oplus 0x_2 \neq 0$; namely, all the vectors in D are not in $X_{\mathcal{C}}(050)$. Therefore, $R_{\mathcal{M}}(050) = R_{\mathcal{C}}(050) = 50$, and the error masking probability is $Q_{\mathcal{M}}(050) = \overline{Q}_{\mathcal{M}} \sim 0.78$. In the following section we introduce a method to choose a D which provides a $\overline{Q}_{\mathcal{M}}$ of 0.3125; this D consists of the following vectors:

$$\{02-09, 12-19, 20, 21, 24, 25, 30, 31, 34, 35, 40, 41, 50, 51, 60, 61, 70, 71, 80, 81, 90, 91\}.$$

B. A lower bound on the error masking probability

Denote by $X_{\mathcal{C}}(\mathbf{e})$ the set of the information words that mask an error $\mathbf{e}$,

$$X_{\mathcal{C}}(\mathbf{e}) = \{\mathbf{x}|(\mathbf{x}, \mathbf{u}(\mathbf{x})) \in \mathcal{C} \text{ and } (\mathbf{x}, \mathbf{u}(\mathbf{x})) \oplus \mathbf{e} \in \mathcal{C}\}.$$

Note that $|X_{\mathcal{C}}(\mathbf{e})| = R_{\mathcal{C}}(\mathbf{e})$.

The choice of D_w determines D, and hence $\mathcal{D}$. Denote by $\triangle(\mathbf{e})$ the difference between the number of codewords that mask $\mathbf{e}$ in $\mathcal{C}$ and the number of codewords that mask it in $\mathcal{M}$,

$$\triangle(\mathbf{e}) = R_{\mathcal{C}}(\mathbf{e}) - R_{\mathcal{M}}(\mathbf{e}).$$

If $\triangle(\mathbf{e})$ equals zero, $R_{\mathcal{C}}(\mathbf{e}) = R_{\mathcal{M}}(\mathbf{e})$ and $Q_{\mathcal{M}}(\mathbf{e})$ is maximized. If $\triangle(\mathbf{e}) > 0$ then $R_{\mathcal{M}}(\mathbf{e}) < R_{\mathcal{C}}(\mathbf{e})$ and $Q_{\mathcal{M}}(\mathbf{e})$ is smaller than its upper bound. From (6) it follows that

$$\triangle(\mathbf{e}) = \Lambda_{\mathcal{M},\mathcal{D}}(\mathbf{e}) + \Lambda_{\mathcal{D},\mathcal{M}}(\mathbf{e}) + R_{\mathcal{D}}(\mathbf{e}).$$

The sum $\Lambda_{\mathcal{D},\mathcal{M}}(\mathbf{e}) + R_{\mathcal{D}}(\mathbf{e})$ is the number of codewords that mask $\mathbf{e}$ in $\mathcal{C}$ and are in $\mathcal{D}$ and therefore are not in $\mathcal{M}$. In fact, it equals the size of the intersection between the set of codewords that mask $\mathbf{e}$ and the set of deleted codewords, that is,

$$\Lambda_{\mathcal{D},\mathcal{M}}(\mathbf{e}) + R_{\mathcal{D}}(\mathbf{e}) = |D \cap X_{\mathcal{C}}(\mathbf{e})|.$$

 ISBN: 978-1-61208-492-3

Similarly, denote by $(D - \mathbf{e}_x) = \{\mathbf{x} \ominus \mathbf{e}_x | \mathbf{x} \in D\}$, then

$$\Lambda_{\mathcal{M},\mathcal{D}}(\mathbf{e}) + R_{\mathcal{D}}(\mathbf{e}) = |(D - \mathbf{e}_x) \cap X_{\mathcal{C}}(\mathbf{e})|.$$

Therefore,

$$\triangle(\mathbf{e}) \leq |D \cap X_{\mathcal{C}}(\mathbf{e})| + |(D - \mathbf{e}_x) \cap X_{\mathcal{C}}(\mathbf{e})|. \quad (9)$$

Thus,

$$\triangle(\mathbf{e}) \leq 2|D|.$$

The rationale behind the choice of M (and hence, the choice of D), is to decrease $\overline{Q}_{\mathcal{M}}$ by decreasing the error masking probability of the errors that maximize it in $\mathcal{C}$; these errors form the set $\overline{E}_x$. In other words, denote by

$$\underline{\triangle} = \min_{\mathbf{e} \in \overline{E}_x, b \in B(\mathbf{e}_x)} \triangle(\mathbf{e}),$$

the minimal difference of the error masking probabilities over all the errors that maximize $Q_{\mathcal{C}}(\mathbf{e})$. The goal is to maximize $\underline{\triangle}$ so as to minimize $\overline{Q}_{\mathcal{M}}$.

Theorem 3. *The error masking probability of $\mathcal{M}$ is*

$$\overline{Q}_{\mathcal{M}} \geq \frac{(|\mathcal{M}| - |\mathcal{D}|)}{p_1 |\mathcal{M}|}. \quad (10)$$

Proof. Let $\mathbf{e}_x \in \overline{E}_x$. For all $b_i \neq b_j \in B(\mathbf{e}_x)$, we have,

$$\{\mathbf{x} | \mathbf{a}\mathbf{x}^T = b_i\} \cap \{\mathbf{x} | \mathbf{a}\mathbf{x}^T = b_j\} = \emptyset.$$

Recall that the size of $B(\mathbf{e}_x)$ is p_1. Therefore, for each $\mathbf{e}_x \in \overline{E}_x$ there are p_1 distinct non-empty and disjoint sets $X_{\mathcal{C}}(\mathbf{e})$.

Consider the intersection of an arbitrary set $S \subseteq \mathbb{Z}_{q^r}^{k_q}$ with all the sets $X_{\mathcal{C}}(\mathbf{e})$ where $\mathbf{e}_x \in \overline{E}_x$. The minimal size of the intersection is smaller or equal to the average; that is,

$$\min_{\mathbf{e}, \mathbf{e}_x \in \overline{E}_x \text{ and } b \in B(\mathbf{e}_x)} |S \cap X_{\mathcal{C}}(\mathbf{e})| \leq \frac{|S|}{p_1}.$$

By applying this upper bound to the sets D and $(D - \mathbf{e}_x)$, we get,

$$\underline{\triangle} \leq \min_{\mathbf{e} \in \overline{E}_x} |D \cap X_{\mathcal{C}}(\mathbf{e})| + |(D - \mathbf{e}_x) \cap X_{\mathcal{C}}(\mathbf{e})| \leq \frac{2|D|}{p_1}.$$

Therefore, the minimal difference of the errors in $\overline{E}_x$ is upper bounded by $\underline{\triangle} \leq 2|D|/p_1$ for any D, and

$$\overline{Q}_{\mathcal{M}} = \frac{\max_{\mathbf{e} \neq 0} R_{\mathcal{C}}(\mathbf{e}) - \underline{\Delta}}{|\mathcal{M}|} \geq \frac{(|\mathcal{M}| - |\mathcal{D}|)}{p_1 |\mathcal{M}|}.$$

□

C. The impact of the size of $\mathcal{M}$ on its error masking probability

Before we address the question of how to choose D_w (and hence, D), we need to relate to cases where the choice of D has no impact. In such cases, regardless of the choice of D, the error masking probability coincides with the worst case given in Th. 2; that is, $\underline{\triangle} = 0$.

Theorem 4. *If $p_1 = 2$, it is always possible to choose D such that $\underline{\triangle} > 0$. If $p_1 \neq 2$ and*

$$|D| \geq (p_1 - 1)k_q + 1,$$

it is possible to choose D such that $\underline{\triangle} > 0$; and similarly, if

$$|D_w| \geq (p_1 - 1)w_q + 1$$

then it is possible to choose D_w such that $\underline{\Delta} > 0$.

Proof ommitted.

IV. Converter structure

Usually, a converter is built from identical sub-blocks. Hence, it is sufficient to determine the set D_w of unused vectors for a single sub-block. In this section, it is assumed that $p_1 = 2$; however, with a small modification the results can be applied to other cases. Recall that we assume that $\frac{k_2}{w_2}$ is an integer, and that $k_q = 2s$. The case where $\frac{k_2}{w_2}$ is not an integer can be viewed as a subcase of this case. The output of the converter is a q-ary vector of length w_q, $\mathbf{x} = (x_1, \ldots x_{w_q})$.

We define a Hamming ball of dimension w_q and radius p_1 as the set

$$H^{w_q, p_1} = \Big\{ \mathbf{z}_j = \sum_{i=1}^{w_q} h_{j_i} \mathbf{v}_i \mid h_{j_i} \in \{0, \ldots, p_1 - 1\} \Big\},$$

where $\mathbf{v}_i = 0^{i-1} 1 0^{w_q - i - 1}$ is a unite vector of Hamming weight one.

The size of a Hamming ball is $p_1^{w_q}$. If $p_1 = 2$ then $|H^{w_q, p_1}|$ divides $|D_w|$. Hence, D_w can be a union of shifted disjoint Hamming balls.

The following construction is designed to maximize $\underline{\Delta}$, hence to minimize the error masking probability.

We start by defining a set of offset vectors Γ^{w_q},

$$\Gamma^{w_q} = \{\theta \in \mathbb{Z}_q^{w_q} \mid \theta = \sum_{i=1}^{k_q} \mathbf{v}_i p_1 t_i, \quad t_i \in \{0, \ldots, \frac{q}{p_1} - 1\}\}.$$

Notice that the symbols of θ are multiples of p_1. Therefore, for any two vectors $\theta_1 \neq \theta_2$, the intersection $(\theta_1 \oplus H^{w_q, p_1}) \cap (\theta_2 \oplus H^{w_q, p_1})$ is empty.

Construction 2 (Disjoint Hamming Balls)**.** *Define the set D_w as*

$$D_w = \bigcup_{\theta_i \in \Theta} \theta_i \oplus H^{w_q, p_1}. \quad (11)$$

where $\Theta \subseteq \Gamma^{w_q}$ is an arbitrary subset of offsets vectors, $|\Theta| = |D_w| / {p_1}^{w_q}$.

Recall that each information word is a concatenation of k_q/w_q vectors of length w_q. If one of these vectors is in D_w, the resulting information word is in D; if none of them is in D_w, the resulting information word is in M. In other words, the set of unused information words is

$$D = \bigcup_{\psi \in \Psi} \psi \oplus H^{k_q, p_1},$$

where a vector ψ is in Ψ if at least one of its k_q/w_q portions is a vector in Θ and the others are in Γ^{w_q}.

Theorem 5. *If D is chosen according to Const. 2 and $p_1 = 2$ then*

$$\frac{(|\mathcal{M}| - |\mathcal{D}|)}{p_1|M|} \leq Q_{\mathcal{M}}(\mathbf{e}) \leq \frac{1}{p_1} = \overline{Q}_{\mathcal{C}}.$$

Proof ommitted.

Is it possible to reach the lower bound on $\overline{Q}_{\mathcal{M}}$? Recall the proof of Theorem 3; in the proof, an upper bound on $\underline{\Delta}$ was obtained by adding the sizes of two sets. If these two sets are disjoint, an equality holds. In other words, it is possible to reach the lower bound on $\overline{Q}_{\mathcal{M}}$ if for all $\mathbf{x} \in D$, $\mathbf{x} \notin D - \mathbf{e}_x$. The following example shows that in some cases this situation cannot be avoided.

Example 3. *Consider the case where $q = 6, k_q = w_q = 3$ and $k_2 = w_2 = 7$. In this case D is a union of 11 shifted Hamming balls $|D| = 11 \cdot |H^{3,2}| = 88$, here $|\Psi| = 11$. In fact, there are $(q/2)^3 = 27$ possible vectors out of which Ψ is chosen. Therefore, there must be at least two linearly dependent vectors in Ψ. Without loss of generality, assume that $\psi_1 = 002$ and $\psi_2 = 004$ are in Ψ. In this case, for $\mathbf{e}_x = 003 \in \overline{E}_x$ and $\mathbf{x} = 005$ we have,*

$$\begin{aligned} \mathbf{x} &= (\psi_2 \oplus 001) \in D \\ \mathbf{x} &= (\psi_1 \oplus 000) \ominus 003 \in D - \mathbf{e}_x \end{aligned}$$

That is, $\mathbf{x}$ is both in D and $D - \mathbf{e}_x$.

V. Conclusion

This work analyzed the efficiency of robust codes when used to protect multilevel memories. When the code's alphabet size, q, is not a power of two, the binary information must be converted into a q-ary word. It was shown that this conversion can significantly degrade the error masking probability of the codes. However, by wisely designing the converter, the degradation of the code properties can be minimized. Bounds on the practical error masking probability were given. A construction for the converter in cases where the QS code is applied to the multilevel memory was provided. It was shown that this construction indeed reduces the error masking probability of the resulting code.

Acknowledgment

This research was supported by the ISRAEL SCIENCE FOUNDATION (grant No. 1200/12).

References

[1] S. Skorobogatov and R. Anderson, "Optical fault induction attacks," in *Cryptographic Hardware and Embedded Systems-CHES 2002*. Springer, 2003, pp. 2–12.

[2] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012.

[3] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, 2006.

[4] R. E. Blahut, "Theory and practice of error control codes," *Reading*, 1985.

[5] M. Karpovsky, K. Kulikowski, and Z. Wang, "Robust error detection in communication and computational channels," *Spectral Methods and Multirate Signal Processing. SMMSP'2007. International Workshop on.*

[6] I. Shumsky, O. Keren, and M. Karpovsky, "Robustness of security-oriented binary codes under non-uniform distribution of codewords," in *DEPEND 2013, The Sixth International Conference on Dependability*, 2013, pp. 25–30.

[7] Y. Neumeier and O. Keren, "Robust generalized punctured cubic codes," *IEEE Transactions on Information Theory*, vol. 60, pp. 2813–2822, 2014.

[8] N. Admaty, S. Litsyn, and O. Keren, "Puncturing, expurgating and expanding the q-ary bch based robust codes," in *Electrical Electronics Engineers in Israel (IEEEI), 2012 IEEE 27th Convention of*, Nov 2012, pp. 1–5.

[9] M. Karpovsky and A. Taubin, "New class of nonlinear systematic error detecting codes," *Information Theory, IEEE Transactions on*, vol. 50, no. 8, pp. 1818–1819, 2004.

[10] M. Bauer, "Data path for multi-level cell memory, methods for storing and methods for utilizing a memory array," Mar. 24 2011, uS Patent App. 12/956,977. [Online]. Available: http://www.google.com/patents/US20110069548 [accessed: 2016-5-22]

[11] K. D. Akdemir, G. Hammouri, and B. Sunar, "Non-linear error detection for finite state machines," in *Information Security Applications*. Springer, 2009, pp. 226–238.

[12] A. Levina and S. Taranov, "Spline-wavelet robust code under non-uniform codeword distribution," in *Computer, Communication, Control and Information Technology (C3IT), 2015 Third International Conference on*. IEEE, 2015, pp. 1–5.

[13] R. Cramer, Y. Dodis, S. Fehr, C. Padró, and D. Wichs, "Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors," in *Advances in Cryptology–EUROCRYPT 2008*. Springer, 2008, pp. 471–488.

[14] Z. Wang and M. Karpovsky, "Robust fsms for cryptographic devices resilient to strong fault injection attacks," in *On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International*. IEEE, 2010, pp. 240–245.

 ISBN: 978-1-61208-492-3

Safe Transitions of Responsibility in Highly Automated Driving

Rolf Johansson
SP, Technical Research Institute of Sweden
Borås, Sweden
e-mail: rolf.johansson@sp.se

Jonas Nilsson
Volvo Car Corporation
Göteborg, Sweden
e-mail: jonas.nilsson@volvocars.com

Martin Kaalhus
Semcon Sweden
Göteborg, Sweden
e-mail: martin.kaalhus@semcon.com

***Abstract*—This paper presents a method for achieving functional safety for an automated vehicle system with respect to safe transitions between a manual and an automated driver, where any single mistake of the human driver is tolerated. Safety analysis and assessment of an implementation example show how to allocate safety requirements on Human-Machine Interface (HMI) components to handle the risks of unfair transitions and mode confusion. Results from this example show that it is sufficient to allocate safety requirements on the sensor of, and the lock of, a single lever to ensure safe transitions. No safety requirements are needed on visual feedback to the driver, e.g., displays.**

Keywords-functional safety; highly automated driving; safety assessment.

I. INTRODUCTION

Presently, the most critical factor for road vehicle safety is the behavior of the driver. There are different estimates, but a common understanding is that humans directly cause significantly more than 90% of serious accidents. More advanced functionality and intelligence implemented in the vehicle means that more of the responsibility to drive safely may be shifted from the skill of the driver to the capability of the functionality implemented in the vehicle.

The potential safety benefit of increased vehicle automation is undoubtedly huge but it is important that the extra risks coming from potential failures of automation are limited to a minimum. In the discipline of functional safety, there are methods to assess risks of malfunctioning E/E implemented functionality, and to reduce these sufficiently. For road vehicles, ISO 26262, [1], is the functional safety standard.

This paper focuses on systems where the vehicle in some specific situations takes full responsibility for the driving task, i.e., level 3 (L3) automation according to the scale defined by the National Highway Traffic Safety Administration (NHTSA). Regarding the responsibility of the manual driver (MD), the precise L3 definition says: "The driver is expected to be available for occasional control, but with sufficiently comfortable transition time", and furthermore: "the driver is not expected to constantly monitor the roadway", [2].

In order to prove that a L3 vehicle is functionally safe, there are two general strategies how to consider the interaction between the manual driver and the vehicle, what in the ISO 26262 terminology is denoted controllability.

In the less conservative strategy, controllability is investigated in detail for all possible scenarios where the manual driver is "expected to be available for occasional control". In the traditional research field of human factors, this is a research question that is currently very much investigated [3], [4], [5]. A rather recent overview of what controllability assumptions that are reasonable on NHTSA L2 and L3 is found in [6].

In the more conservative strategy, there are no assumptions that the manual driver can take back control within a bounded time. One could say that we do not require the driver to be comfortable with any short transition time. This strategy is the one chosen by Volvo Cars in the DriveMe project [7], where the vehicle takes full responsibility to safely handle any critical situation during automated driving. We can call this an autopilot with full responsibility for safety, as it does not need to rely on any responsiveness from the manual driver to stay safe.

An unsolved question so far, is if the more conservative assumption about the human capability still can enable the design of a functionally safe car. The introduction of an autopilot with full responsibility leads to two new challenges within functional safety. We need to ensure safety when the autopilot is in charge, but also ensure safe transitions between the manual driver and the automated driver (AD). This paper investigates the latter.

The contribution of this paper is a method for achieving functional safety for an automated vehicle system with respect to safe transitions between a manual driver and an autopilot with full responsibility for safety. We assume that both the human driver and the autopilot are capable of safe driving, as well as judging its own ability to drive safely. Thus, neither the driver nor the autopilot are required to take control and thus the vehicle will be in a safe state if either the driver or the autopilot accept to take control of the vehicle.

There are simulator studies suggesting that human drivers may change their driving behaviour when taking back control from an autopilot, [8]. This is not considered in this paper as we focus on functional safety rather than design of the HMI or autopilot driving behaviour.

This paper is organized as follows. Section II describes the new hazards related to the driving mode transitions introduced by NHTSA L3. In Section III, we discuss how to

 ISBN: 978-1-61208-492-3

define a safe transition and the acceptable level of fault tolerance. Section IV elaborates on possible implementations using a system example and corresponding functional safety analysis and assessment. Finally, Section V presents concluding remarks.

II. WHAT CAN CAUSE THE ROAD VEHICLE TO BE UNSAFE

One interpretation of a hazard analysis & risk assessment (HA&RA) today according to ISO26262 is that the vehicle itself is considered safe, if it only puts the driver in situations that are possible to manage safely. The driver is ultimately responsible for safe driving, and the malfunctions of the vehicle should be restricted in such a way that the driver can keep the vehicle in a safe state. The explicit method for determining the requested Automotive Safety Integrity Level (ASIL), restricting a certain hypothetical vehicle failure, is to measure three factors: exposure, severity and controllability. The two first factors are the traditional ones that are part of the definition of risk, i.e., a combination of probability and severity. The third factor is the one that takes into account that the driver may sometimes have a possibility to keep the vehicle safe, even though the ordinary (safety-related) functionality is failing.

When we shift from a situation where a manual driver has the ultimate responsibility, to highly automated driving where the manual driver and an autopilot are alternating, this will have an impact on the HA&RA. So, what will become different when going from NHTSA L2 to L3? This new challenge has partly been addressed in [9].

We require the same from an autopilot as from a manual driver. This means focusing on a safe style of driving, making the driver capable to handle also unexpected events. When programming an autopilot, this is what we cover on the tactical level [10], [11]. The autopilot should always choose to perform the maneuvers in such a way that reasonable, but still unexpected, situations could be handled safely. For example, the decision whether or not to initiate an overtaking maneuver is on the tactical level. An optimistic decision to overtake may cause the vehicle in a situation where avoiding one accident may cause another. The solution to this dilemma is of course to initiate an overtaking maneuver only when the entire operation is foreseen to be possible to fulfil in a safe manner.

Note the contrast to Advanced Driver Assistance Systems (ADAS), where the vehicle takes over only on the operational time scale, and then assumes the manual driver to continue according to the (maybe revised) tactical plan. The ADAS functionality today, does not take the ultimate responsibility to drive the vehicle safely. Firstly, it only operates on the operational time scale. Secondly, it only assists the manual driver. When the responsibility is transferred from the manual driver to the autopilot, there is no longer an assistance relation. The transfer means that from then on, the automated driver is fully responsible for driving the vehicle safely.

Given that the autopilot can drive safely once in command, the HA&RA must also cover the transitions between the driver and the autopilot. In NHTSA L3, these transitions introduce two new types of hazards, namely unfair transitions and mode confusion. These are described in detail in the following sections.

A. *Unfair transitions*

As we noted in the above section, it may be complicated for the driver to make a proper override of a failing tactical decision of the automated driver. This is because drivers may find different tactical solutions to a certain driving situation, and each of these may be correct. It may be hard for a driver to distinguish a faulty tactical decision from a one that is just different from his or her own favorite pattern. Even more, it may be very hard to continue to fulfill a tactical plan of another driver if the responsibility is transferred in the middle of the intended sequence. This difficulty is both for a manual driver to continue a plan of the automated driver, and for the automated driver to continue what has been initiated by the manual driver.

If the manual driver realizes that the automated driver has handed over responsibility, without the manual driver agreeing to this, this is a new risk to consider when entering NHTSA L3. We can say that the manual driver is put in a situation of *unfair transition*. For a driver with the same understanding of the planned tactics, the situation may be easy to handle, but an unfair transition may put the driver in a situation where driving safely will be very difficult.

The problem of unfair transitions may appear in both directions. It is reasonable to assume that the automated driver can drive safely as long as it can choose its own tactics. This is a far easier task than being able to understand and solve arbitrary situations.

To summarize, if the responsibility is transferred from one driver to the other, this must include a confirmation from the receiving driver. Otherwise, the transition may be regarded as unfair, and it is a non-negligible risk that the second driver is incapable of handling the situation, on both operational and tactical time scales.

B. *Mode confusion*

In order to make the entire trip from start to stop safe, it is critical that the two drivers always agree on which of them that currently is in charge. If they misunderstand each other, there is a risk that either there are two drivers trying to control the vehicle, or there is no one taking care of the ride. Both these potential *mode confusions* need to be addressed.

If we allow both the manual driver and the automated driver to override each other, there is an obvious risk that the resulting non-harmonized commanding of the vehicle may result in dangerous situations. This is especially probable because the two drivers most likely make different tactical decisions now and then, and as consequence regard the operative command of the other as faulty. For safe driving in NHTSA L3, it is important to reduce the risk of this reciprocal *override*.

It is perhaps even more obvious that it will become dangerous if neither the manual driver nor the automated driver regard herself as the ultimately responsible. Such reciprocal *underride* is therefore obviously important to reduce properly when performing the risk assessment for driving on NHTSA L3.

 ISBN: 978-1-61208-492-3

C. State-of-the-art comparison with other industries

This section describes technology, systems and concepts from other industries where similar problems arise caused by mode confusion and unsafe transitions. The focus has been on nuclear, rail, avionics and space since these industries deal with complex systems, is in a regulated environment and all demand active users for proper operations. Experiences from other industries give valuable insight into how to design interfaces and processes that ensure safe transitions in the context of autonomous driving. However, these inspiration sources material and solutions need to be adapted to fit into the automotive context in order to be a viable tool.

As the existing autonomous systems within the automotive industry are still in their infant stages and the majority of them still are semi-autonomous (i.e. NHTSA L1-L2) at time of writing, these systems are excluded. The interested reader may study results from several research efforts on this topic; PReVENT, HAVE IT, ADAPTIVE and INTERACTIVE to mention a few.

When reviewing earlier experiences from nuclear, avionics, rail and space industries we make one important observation. Within these industries, the technical solutions are operated by educated users, certified to use the specific equipment, often in controlled environments and in cooperation with colleagues supporting them.

In avionics, there is a system called Auto Ground Collision Avoidance System (AGCAS) that monitors the pilot's response in certain situations and if the pilot does not respond to an alarm, the system takes over and performs the necessary manoeuvre. After avoiding the threat, control is returned to the pilot. Inagaki describes this as situation-adaptive autonomy where authority over a system is transferred between human and machine agents, [12]. However, the main point of reference within avionics is that an educated pilot is responsible for operation of the airplane at all times, differing from the automotive situation.

Two major players in the avionics industry, Boeing and Airbus, apply different philosophies regarding automation. Boeing implements a strict assisting role for technology and automation, where the pilot always acts as the final authority. Airbus rather sees automation as a way of enhancing flight performance by assisting the responsible pilot. This subtle difference in philosophy causes different problems, where the Boeing strategy allows the pilot to perform errors that may cause accidents and the Airbus strategy may interfere and prevent the pilot from performing necessary maneuverers needed for safety in extreme situations [13] [14].

Within nuclear there are numerous processes to monitor. This is handled with different interfaces displaying process information. In Sweden there are different systems ensuring correct decisions regarding the operation of the nuclear plant. There are regulations stating that the plants are to be designed in such a way that operators always have a 30 minute window to perform an action. In other words, the plant is fully autonomous for 30 minutes at a time. There are also mechanisms that require several users to acknowledge an action independently in order to perform it, which can be compared to the needed protocol in automotive. However, the time constants at play are significantly different when compared to the automotive setting.

In the nuclear industry, one main control board always represents the true state of the processes. It is assured to a higher safety integrity and acts as the primary source of information should different sources provide inconsistent information. The inconsistent information does pose less of a problem since nuclear operators are well educated with the system and knows what information to depend on. However, translating this into the automotive setting is problematic, where most of the information sources primary purpose is to enhance and ease the experience rather than to provide safety-assured information on the system state.

Within the space industry, interfaces and systems are often complex and users need to understand how to use many different systems at the same time for proper operation. Because of this, a lot of effort is put into mental models of the systems; the users do not need to understand how and why the systems work only have a basic understanding of what situations the system can be used in and what the outcome would be. This could be translated into the automotive setting where the situation is similar, although on a smaller scale and users need to quickly gain a basic understanding about the autonomous systems capabilities and limitations.

Studies from the rail industry have analysed operator workload and the possibilities of it causing human errors. Two main ways of managing human performance have been formulated, through either technology or human resource management. Assessment of individual possibilities to manage the required workload has been performed through psychometric testing, as well as limiting workdays and issuing regular breaks [15].

As the automotive setting makes it difficult to limit usage periods, the technology and interfaces must be designed to ensure safe usage under these circumstances. Adaptive interface features linked to specific task requirements with consistency in interface design across different modes of system operation is recommended in order for the users to effectively apply mental models [16].

III. WHAT MAKES A TRANSITION SAFE

In the previous section, we have listed new categories of risks to handle related to the dual driving modes when going up in automation degree to NHTSA L3. In the following sections we summarize our current understanding on how to handle these.

A. General Strategy

A main strategy to eliminate *unfair transitions* is to introduce a fair procedure for handover. This means that the current responsible driver (manual or automated) stays responsible until there is an agreement for a handover. If we can find out how to design safe handover of responsibility, this will then solve the problem of unfair transitions. For a handover to be regarded as safe, we need to address both what is reasonable to assume of a driver, and what safety

requirements we need to allocate on the elements implementing the vehicle part of the handover protocol.

The problem of *Mode confusion* can be solved by combining safe handover mechanisms with requirements on each of the two candidate drivers to remember who is currently in charge. When the automated driver is responsible, the manual driver should then try to avoid interfering with the AD. This can be solved by not allowing the MD to have any impact on the vehicle, if not first going through a handover procedure. If we want, we can transfer part of that manual responsibility to the vehicle by putting safety requirements on ignoring any try from the manual driver to control the motion of the vehicle. Furthermore, we require of the manual driver to stay responsible once becoming in charge. In a similar way, we put safety requirements on the vehicle to remember who the agreed responsible driver is.

B. Fault Tolerance

As stated in the previous section, we require from a safe transition that the two candidate responsible drivers (MD and AD) regard the transitions fair and have a common understanding who has received the responsibility. This implies that both drivers need to explicitly confirm that a transition is possible and fair to perform. Furthermore, it implies that both drivers really are aware of what has been agreed.

Already today, we have a substantial amount of serious traffic accidents caused by driver lapses. There is no reason why not to regard the manual driver of a highly automated vehicle as prone to mistakes in any HMI, including the one for transition of responsibility. Because of this, we need to have a procedure where the manual driver has to perform several and coordinated actions, in order to allow a transition. Every single action can be assumed as performed by mistake, but the more of coordinated multiple actions that are required the less probable it is that the driver is not aware of what she or he is doing.

For the vehicle, we assume that safety requirements are allocated to all elements critical for achieving a transition in such a way that it can be considered as fair and consistently understood by both drivers. We make a conservative assumption that the ASIL attribute to use is the one that is representing the highest ASIL among the possible induced hazards. In practice, this means that we need ASIL D on guaranteeing freedom from mode confusion and from unfair transitions. Of course, redundancy patterns may be applied allowing the ASIL D to be decomposed onto different elements of the implementation.

A way to argue that a transition is safe is to check what happens if there is either a manual mistake or an E/E failure, or combination of these. This must be checked for any state in the transition protocol. For any hazardous consequence, it must be shown that the corresponding E/E failure is prevented with an appropriate safety requirement. If a manual failure may lead to a hazardous consequence even in a fault free case, the protocol implementation is obviously not robust enough.

IV. General Implementation Suggestion

In order to make a transition tolerant to any single manual mistake, there are a few different general ways to design the protocol. The redundant action from the manual driver can in general be either in time or in space, or a combination of these. By time redundancy, we mean here to request a sequence of actions where the second must follow in a certain time interval after the first one. Space redundancy is on the other hand when the manual driver is requested to apply several actions simultaneously. In both cases, the idea is that it can be argued that the set of actions is extremely unlikely to be performed by mistake.

A. Example HMI Protocol and Implementatios

As an example in this paper, we chose to describe a protocol based on manual time redundancy. This means that we always require two actions from the driver for any transition from MD to AD or from AD to MD. Furthermore, we say that the second action of the manual driver defines the transition, which means that there is no requirement on the manual driver to observe the resulting outcome correctly, more than knowing what she is doing herself. As long as the second action is fulfilled, the transition is deemed to have occurred.

In Figure 1, a general protocol is illustrated, where two coordinated actions are required from the manual driver. When implementing this it is important not to allow the driver to perform the second action, without having acknowledged the first one.

In this example, we chose the first action to be a press of a button and the second to be a change of lever position. This lever has exactly two possible positions: AD and MD. The vehicle is always started in MD, and the driver may change the mode after reaching the proper state in the transition protocol. We consider the lever to be locked at any other time. Furthermore, if the lever is not moved fast enough after getting acknowledge by the autopilot, it will be locked again requiring the protocol to start over again in order to perform a transition.

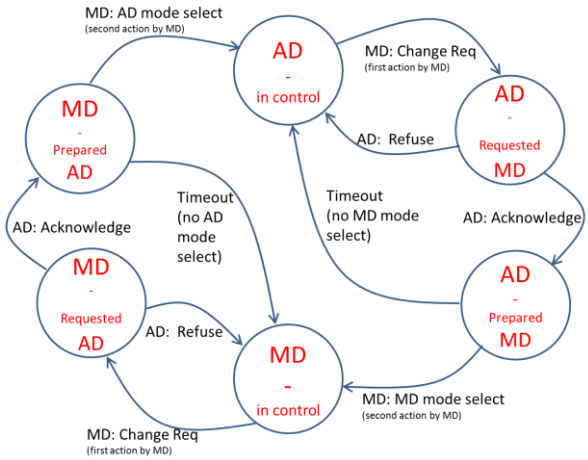

Figure 1. Example of a simple transition protocol.

This protocol is based on the assumption that it is always safe to keep the mode if nothing else is agreed. The current driver should always be able to continue to take care of the vehicle in a safe manner.

We can extend the protocol to cover the cases where the AD can suggest a transition, either by declaring that the AD is ready to take over from the MD, or by telling the MD that the AD performance is limited. Such a protocol is depicted in Figure 2.

To implement this protocol we suggest the following HMI components:

- Telltale light showing the AD view of preferred mode
- Pushbutton to for the MD to ask for mode change (first action)
- Telltale light showing whether the AD is prepared for a change as requested by the MD
- Lever for the MD to select mode (second action)

Any failure mode of these four HMI components then needs to be included in the safety analysis, and this in combination by any single mistake by the manual driver.

To summarize, a fault-free uninterrupted transition from the MD to the AD in this example follow the steps:

- The MD drives the vehicle (MD mode)
- The AD declares it is ready to take over by changing the preference telltale to AD available
- The MD asks to take over by pressing the pushbutton
- The AD acknowledges that it is prepared by indicating the readiness telltale and unlocking the lever
- The MD changes the lever to AD position
- The AD locks the lever, and continues to drive in AD mode

The transition from AD to MD is performed in a similar way, i.e., the MD may either independently, or suggested by the AD, start by asking for a mode change. The AD then acknowledges by indicating on the readiness telltale and unlocking the lever. Finally, the MD changes the lever to the MD position and starts to drive manually.

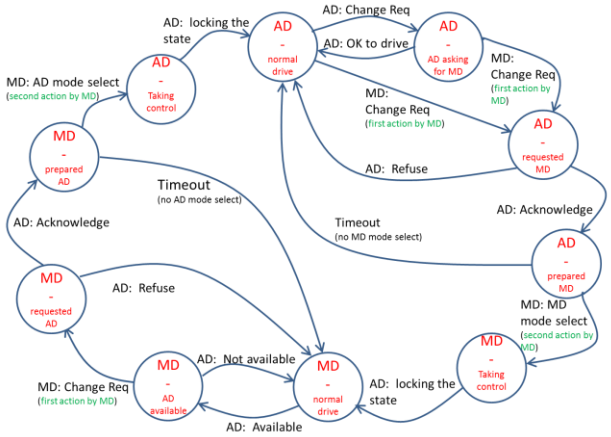

Figure 2. Example of an elaborated transition protocol.

B. *Safety Analysis*

In the following section, the above protocol and implementation is analyzed with respect to its sensitivity to any human mistake, vehicle component failure, or a combination of these. Hence, we walk through the detailed state diagram and investigate the possible failure consequences at any state.

When doing the safety analysis, we document the result in Table 1. The columns are:

- Protocol state
- HMI failure to investigate
- Possible driver mistake
- Consequence in words
- Consequence in terms of safe/unsafe

Each row in this table marked as unsafe in the last column needs to be protected by a corresponding safety requirement allocated to restrict this HMI failure. If all occurrences of an unsafe consequence are protected by appropriate safety requirements, the protocol implementation is deemed safe. In order for the safety argumentation to be valid, it is important that the table is shown to be complete. This includes an argumentation that all possible human mistakes are considered.

C. *Safety Assessments*

As concluded from the safety analysis in Table 1, there are three ways for the example protocol to fail in an unsafe way, caused by either of a manual mistake, a vehicle component failure, or a combination of these. The three failures that we need to avoid to maintain safety are:

- The AD cannot correctly sense the mode lever position, which may cause mode confusion.
- The AD cannot guarantee lock of the mode lever according to the protocol. This in combination with the MD moving the mode lever to AD mode, without noticing it, may cause mode confusion (or unfair transition if discovered by the MD).
- The AD cannot guarantee locking of the mode lever according to the protocol. This in combination with the MD changing lever position from MD to AD, without getting acknowledgment of a prepared AD, may cause unfair transition.

As we assume that the MD may make any single failure at any time, the way to argue for avoiding the above failures is to put the entire responsibility on the vehicle. This implies that we put two safety requirements on the HMI.

- ASIL D on restricting faulty lever sensor, i.e., the lever sensor needs to be always correct.
- ASIL D on restricting lever lock faulty unlocked (faulty locked consider as safe).

If we can guarantee that the HMI is implemented according to these two safety requirements we can claim that we make a safe transition even in the presence of an arbitrary single manual mistake. This handles both the mode confusion and the unfair transition aspects of a safe transition.

 ISBN: 978-1-61208-492-3

TABLE I. SAFETY ANALYSIS OF TRANSITION PROTOCOL

Protocol state	HMI failure	Driver mistake	Consequence	Safe/ Unsafe
MD - normal drive	Fault in lever lock	No	MD driver not trying to touch lever. Stay in MD.	Safe
MD - normal drive	Fault in lever lock	Driver changes lever position without asking for change first.	Unfair transition.	Unsafe
MD - normal drive	Fault in preference telltale	Any mistake or correct behaviour	MD cannot change locked lever. Stay in MD- normal drive.	Safe
MD - AD available	Fault in lever lock	No	MD driver not trying to touch lever. Stay in MD.	Safe
MD - AD available	Fault in lever lock	Driver changes lever position without asking for change first.	Unfair transition.	Unsafe
MD - AD available	Fault in preference telltale	No	Stay in MD	Safe
MD - AD available	Fault preference telltale	Driver ignores lack of availability	Transition sequence fulfilled. Change to AD.	Safe
MD - requested AD	Fault in pushbutton	Any mistake or correct behavior	No Acknowledge by AD. Lever still locked. Stay in MD.	Safe
MD - prepared AD	Fault in prepared telltale	Driver correct: Driver stops transition sequence	Time-out in protocol. Stay in MD.	Safe
MD - prepared AD	Fault in prepared telltale	Driver incorrect: Driver ignores lack of ack.	Transition sequence fulfilled. Change to AD	Safe
MD - prepared AD	Fault in lever lock	Driver correct: Driver tries but cannot fulfil transition sequence.	Time-out in protocol. Stay in MD.	Safe
MD - prepared AD	Fault in lever lock	Driver incorrect:Driver doesn't continue transition sequence.	Time-out in protocol. Stay in MD.	Safe
AD – taking control	Fault in lever sensor	Any mistake or correct behavior	Mode confusion	Unsafe
AD – normal drive	Fault in lever lock	No	MD driver not trying to touch lever. Stay in MD.	Safe
AD – normal drive	Fault in lever lock	Driver changes lever position to MD without asking for change first, and without noticing what is happening.	Mode confusion. (Unfair transition, if realized later).	Unsafe
AD – normal drive	Fault in preference telltale	No	MD acts as in normal AD mode. Stay in AD or ask for transition.	Safe
AD – normal drive	Fault in preference telltale	Driver tries to changes lever position but it is locked in AD position.	Stay in AD.	Safe
AD – asking for MD	Fault in lever lock	No	MD not touching lever without asking for change first. Stay in AD.	Safe
AD – asking for MD	Fault in lever lock	Driver changes lever position by mistake without noticing it in the first place, and without asking for change first.	Mode confusion (Unfair transition, if realized later).	Unsafe
AD – asking for MD	Fault in preference telltale	Any mistake or correct behavior	MD can request MD mode or stay in AD mode.	Safe
AD – requested MD	Fault in pushbutton	Any mistake or correct behavior	No Acknowledge by AD. Lever still locked. Stay in AD.	Safe
AD – prepared MD	Fault in prepared telltale	No	Driver stops transition sequence. Time-out in protocol. Stay in AD.	Safe
AD – prepared MD	Fault in prepared telltale	Driver ignores lack of ack.	Transition sequence fulfilled. Change to MD	Safe
MD – taking control	Fault in lever lock	Any mistake or correct behavior	Driver tries but cannot fulfil transition sequence. Time-out in protocol. Stay in AD.	Safe
MD – taking control	Fault in lever sensor	Any mistake or correct behavior	Mode confusion	Unsafe

If ASIL D sensors and/or ASIL D locks are considered either unavailable or very expensive, we may consider redundancy implementation techniques. Instead of one sensor always telling the correct lever position with ASIL D attribute, we may consider three (sic!) sensors each with ASIL B. If at least two of the three are correct, we can stay safe. This means that we need to restrict that two of the three are failing. This shall be guaranteed with a total ASIL D, which we distribute as ASIL B on each sensor. Similarly, using ASIL A sensors would require seven times redundancy. If four out of seven are working we consider it as safe. This means that we need to restrict that four of the sensors are failing. This shall be guaranteed with a total ASIL D, which we distribute as ASIL A on each sensor.

Instead of one lever lock always guaranteeing that the lever is never faulty unlocked, we may consider two locks each with ASIL B. We consider faulty locked as a safe state. If at least one of two locks can guarantee freedom from faulty unlocked, we can stay safe. This means that we need to restrict that both of the two locks are faulty unlocked. This shall be guaranteed with a total ASIL D, which we distribute as ASIL B on each lock. Similarly, using locks guaranteeing absence of faulty unlocked with ASIL A would require quadruple redundancy. If only one of the locks is avoiding faulty unlocked, we consider it as safe. This means that we need to restrict that all the four locks are faulty unlocked. This shall be guaranteed with a total ASIL D, which we distribute as ASIL A on each lock.

 ISBN: 978-1-61208-492-3

V. CONCLUSION

When introducing an autopilot which in some driving situations takes full responsibility to drive the vehicle, it becomes crucial to ensure safe transitions between the manual and the automated driver. The existence of dual driving modes brings two new sources of risk, namely unfair transitions and mode confusion.

We propose to define a safe transition as a transition where either a manual mistake or an E/E failure, or combination of these, leads to an unfair transition or mode confusion. Furthermore, we demonstrate on a system example how to allocate safety requirements on system elements to ensure safe transitions.

Results from this example show that it is sufficient to allocate safety requirements on the sensor and lock of a single lever to ensure safe transitions. No safety requirements are needed on visual feedback to the driver, e.g., displays. We remark that the example implementation by no means is a unique solution to the safe transitions problem.

ACKNOWLEDGMENT

This research has been supported by the Swedish government agency for innovation systems (VINNOVA) in the FUSE project (ref 2013-02650).

REFERENCES

[1] ISO, "International Standard 26262 Road vehicles -- Functional safety", November 2011.

[2] National Highway Traffic Safety Administration, "Preliminary Statement of Policy Concerning Automated Vehicles", http://www.eenews.net/assets/2016/01/14/document_pm_01.pdf, retrieved: June 2016.

[3] C. Gold, D. Damböck, K. Bengler, and L. Lorenz, "Partially Automated Driving as a Fallback Level of High Automation," 6. Tagung Fahrerassistenzsysteme. Der Weg zum Autom. Fahren., 2013.

[4] M. H. Martens and A. P. Van Den Beukel, "The road to automated driving: Dual mode and human factors considerations," IEEE Conf. Intell. Transp. Syst. Proceedings (ITSC) , 2013, pp. 2262–2267.

[5] F. Naujoks, C. Mai, and A. Neukum, "The effect of urgency of take-over requests during highly automated driving under distraction conditions," Adv. Hum. Asp. Transp. Part I, vol. 7, July 2014, p. 431.

[6] National Highway Traffic Safety Administration, "Human Factors Evaluation of Level 2 And Level 3 Automated Driving Concepts Past Research, State of Automation Technology, and Emerging System Concepts", http://www.nhtsa.gov/DOT/NHTSA/NVS/Crash%20Avoidance/Technical%20Publications/2014/812043_HF-EvaluationLevel2andLevel3AutomatedDrivingConceptsV2.pdf, retrieved: June 2016.

[7] Volvo Cars, "THE SELF-DRIVING CAR IN ACTION – DRIVE ME", http://www.volvocars.com/intl/about/our-innovation-brands/intellisafe/intellisafe-autopilot/drive-me, retrieved: June 2016.

[8] S. Brandenburg and E. Skottke, "Switching from manual to automated driving and reverse: Are drivers behaving more risky after highly automated driving?," IEEE 17th Int. Conf. Intell. Transp. Syst. (ITSC), 2014, pp. 2978–2983.

[9] R. Johansson, C Bergenhem, and H. Sivencrona, "Challenges of Functional Safety in ADAS and Autonomous Functions", SAE World Congress, Detroit, April 2014.

[10] R. Sukthankar, "Situation Awareness for Tactical Driving", Ph.D. thesis, Robotics Institute, Carnegie Mellon University, USA, January 1997.

[11] T. X. P. Diem and M. Pasquier, "From Operational to Tactical Driving: A Hybrid Learning Approach for Autonomous Vehicles", 2008 10th Intl. Conf. on control, Automation, Robotics and Vision, Hanoi, Vietnam, December 2008.

[12] T. Inagaki, "Design of human–machine interactions in light of domain-dependence of human-centered automation", Cognition, Technology & Work, Volume 8, Issue 3, 2006, pp 161-167.

[13] A. Marinik, R. Bishop, V. Fitchett, J. F. Morgan, T. E. Trimble, M. Blanco. "Human factors evaluation of level 2 and level 3 automated driving concepts: Concepts of operation." (Report No. DOT HS 812 044). Washington, DC: National Highway Traffic Safety Administration., July 2014.

[14] H. Orlady, R. Barnes, "A Methodology for Evaluating the Operational Suitability of Air Transport Flight Deck System Enhancements", SAE Technical Paper # 975642, 1997.

[15] J. Cunningham "Break the monotony." Professional Engineering, 20(20), 33-33. 2007.

[16] D.B. Kaber, L. J. Prinzel, "Adaptive and adaptable automation design: A critical review of the literature and recommendations for future research." (NASA/TM-2006-214504), September 2006.

 ISBN: 978-1-61208-492-3