Dipl.-Ing. Roland Mader Bakk.techn.

Computer-Aided Model-Based Safety Engineering of Automotive Systems

Dissertation

vorgelegt an der
Technischen Universität Graz

zur Erlangung des akademischen Grades
Doktor der Technischen Wissenschaften
(Dr. techn.)

durchgeführt am Institut für Technische Informatik
Technische Universität Graz
Betreuer: Em.Univ.-Prof. Dipl.-Ing. Dr.techn. Reinhold Weiß

Graz, im November 2012

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ....

(Unterschrift)

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....

date (signature)

Kurzfassung

Die automotive Industrie erfährt einen Trend hin zur Elektrifizierung von Antriebssträngen. Das führt zu neuen Antriebsstrangtopologien wie zum Beispiel Antriebssträngen für Hybridfahrzeuge. Diese Antriebsstränge enthalten zusätzliche Komponenten wie elektrische Maschinen oder Hochvoltbatterien. Diese Komponenten sowie deren Interaktion werden von eingebetteten Systemen gesteuert. Daraus resultierend, können Fehler oder Ausfälle dieser eingebetteten Systeme zu schweren Gefährdungen führen. Deshalb sind die eingebetteten Systeme von Hybridfahrzeugen sicherheitskritisch. Darüber hinaus tragen die zusätzlichen Komponenten von Hybridfahrzeugen dazu bei, dass sich die ohnehin steigende Komplexität von automotiven, eingebetteten Systemen weiter erhöht. Die Entwicklung zunehmend komplexer, sicherheitskritischer eingebetteter Systeme erfordert geeignete Safety Engineering Workflows, die sich an den funktionalen Sicherheitsstandard ISO 26262 für die automotive Domäne anlehnen, sowie Werkzeugunterstützung, die es ermöglicht, mühsame und fehleranfällige Aktivitäten zu automatisieren. Unter den typischerweise werkzeugunterstützten Aktivitäten für die Entwicklung sicherheitsgerichteter, automotiver eingebetteter Systeme finden sich (a) Frühzeitige Gefährdungsanalyse, (b) Fehlerbaumanalyse, FMEA (Failure Modes and Effects Analysis) und Allokieren von Sicherheitsparametern basierend auf Systemmodellen sowie (c) Generierung von Quellcode und Modellen. Jedoch ist die Werkzeugunterstützung in Bezug auf (a), (b) und (c) verbesserungsfähig.

Der Beitrag dieser Dissertation ist es, den Stand der Technik in Bezug auf (a), (b) und (c) zu verbessern und beschreibt einen Ansatz zum Model-Based Safety Engineering of Automotive Systems. Insbesondere umfasst dieser Ansatz Werkzeugunterstützung für (1) die Erstellung sicherheitsrelevanter Modelle, (2) die Generierung von Fehlerbäumen, FMEA Tabellen und ASIL (Automotive Safety Integrity Level) Allokierungen sowie für (3) die Konfigurierung und Codegenerierung. Das Werkzeug OASIS (Aut**O**motive **A**nalysis and **S**afety Eng**I**neering In**S**trument) stellt eine Implementierung von (1), (2) und (3) dar. Dieses Werkzeug ist Teil einer Werkzeugkette, die einen automotiven Safety Engineering Workflow unterstützt. Diese Werkzeugkette wurde verwendet, um den präsentierten Ansatz zum Computer-Aided Model-Based Safety Engineering of Automotive Systems unter der Verwendung der Fallstudie einer Hybridfahrzeugentwicklung experimentell zu evaluieren. Metriken zeigen, dass Computer-Aided Model-Based Safety Engineering of Automotive Systems zu reduziertem Zeitaufwand sowie zu gesteigerter Produktqualität führt, was auf die Vereinfachung sowie die Automatisierung von Arbeitsschritten zurückzuführen ist. Aus diesem Grund kann Computer-Aided Model-Based Safety Engineering of Automotive Systems die Industrie dabei unterstützen, sichere und leistbare Produkte zu entwickeln.

Abstract

The shift of the automotive industry towards powertrain electrification leads to new powertrain topologies such as HEV (Hybrid Electric Vehicle) powertrains. HEV powertrains contain additional components such as electric machines or high voltage batteries. These components and their interaction are controlled by embedded systems. Consequently, faults and failures of these embedded systems can lead to severe hazards. Thus, the embedded systems of HEVs are safety-critical. Furthermore, the additional components of HEV powertrains pursue the trend of increasingly complex automotive embedded systems. The development of increasingly complex safety-critical automotive embedded systems requires appropriate safety engineering workflows aligned with the functional safety standard ISO 26262 for the automotive domain as well as tool support for automating tedious and error-prone activities. Among the typically tool-supported activities required for automotive safety-critical embedded systems development are (a) Early Hazard Analysis, (b) FTA (Fault Tree Analysis), FMEA (Failure Modes and Effects Analysis) and Allocation of Safety Parameters based on System Models as well as (c) Generation of Source Code and Models. However, tool support for (a), (b) and (c) is improvable with respect to several aspects.

This thesis contributes by advancing the state of the art with respect to (a), (b) and (c) and presents an approach to Computer-Aided Model-Based Safety Engineering of Automotive Systems. In particular, this approach comprises support for (1) Safety-Relevant Model Creation, (2) Generation of Fault Trees, FMEA Tables and ASIL (Automotive Safety Integrity Level) Allocations as well as (3) Configuration and Code Generation. The tool OASIS (Aut**O**motive **A**nalysis and **S**afety Eng**I**neering In**S**trument) constitutes an implementation of (1), (2) and (3). This tool is part of a tool chain supporting an automotive safety engineering workflow. The tool chain was used to apply the automotive safety engineering workflow and to experimentally evaluate the presented approach to Computer-Aided Model-Based Safety Engineering of Automotive Systems using the case study of an HEV development. Metrics show that Computer-Aided Model-Based Safety Engineering of Automotive Systems leads to reduced expenditure of time and improved product quality due the simplification and automation of workflow steps. Thus, Computer-Aided Model-Based Safety Engineering of Automotive Systems can potentially help the industry to create safe products at affordable prices.

Acknowledgements

The contributions of this thesis were elaborated in course of the MEPAS project that was financially supported by the 'COMET K2 Forschungsförderungs-Programm' of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economics and Labour (BMWA), Österreichische Forschungsförderungsgesellschaft mbH (FFG), Das Land Steiermark and Steirische Wirtschaftsförderung (SFG). Working on the MEPAS project and completing this thesis involved the contribution of many people. However, they are too many to list them all. It is, however, my wish to express my gratitude to a few of them in the following.

Foremost, I want to thank Prof. Reinhold Weiß for offering me the opportunity of being part of his institute, tackling research problems in an industrial context and facing the challenge of writing a thesis. Furthermore, I would like to thank him for advising me how to compose and complete this thesis.

I also would like to thank the MEPAS project leaders of the participating organizations, Christian Steger and Christian Kreiner (Institute for Technical Informatics), Eric Armengaud and Peter Reichenpfader (Virtual Vehicle Competence Center) as well as Gerhard Grießnig (AVL List GmbH). Whereas they provided a good working environment to execute this project, they left me a lot of open space to be creative and innovative. Furthermore, I wish to thank all colleagues at the Institute for Technical Informatics, the Virtual Vehicle Competence Center and AVL List GmbH for a good and fruitful collaboration during more than three and a half years.

In particular, I would like to thank my colleagues Gerhard Grießnig, Eric Armengaud and Quentin Bourrouilh at AVL List GmbH for excellent discussions, providing qualified comments, supporting the experimental evaluation of research results using an industrial case study and providing an atmosphere of friendship. Thanks to their help, it was possible to tackle some of the problems of the automotive industry.

I would like to express my greatest gratitude to my parents, Franz Mader and Roswitha Mader, for their life-long support and their understanding during a challenging, distressful and exhausting time of elaborating research results, writing publications and completing this thesis. Finally, I also want to thank my grandmother Gisela Griesmayr for encouraging me to complete this thesis.

This thesis is dedicated to my grandmother Gisela Griesmayr, who died on the 21st of November 2011.

Bruck an der Mur/Austria
November, 2012 Roland Mader

Extended Abstract

The shift of the automotive industry towards powertrain electrification leads to new powertrain topologies such as HEV (Hybrid Electric Vehicle) powertrains. HEVs are an attempt to combine the advantages of classical ICE (Internal Combustion Engine) cars and EVs (Electric Vehicles). Thus, besides the primary unidirectional energy converter (ICE) and the primary power source (petroleum fuel tank), an HEV contains an additional secondary bidirectional energy converter (electric machine) and an additional secondary power source (e.g. high voltage battery). The secondary bidirectional energy converter can be used as an electric motor to support the ICE by providing supplementary or substitutive torque. Furthermore, the secondary bidirectional energy converter can be used as a generator for regaining kinetic energy during braking or motoring to charge the high voltage battery. An HEV supports several operating modes that bring additional flexibly and can be used to optimize overall performance, efficiency and emissions with proper configuration and control.

The additional electric machine and the high voltage battery as well as other powertrain components are controlled by an embedded system consisting of dozens of control units connected by automotive bus systems. Faults and failures of this embedded system can lead to severe hazards such as fire/explosion caused by overcharging of the high voltage battery or unintended vehicle movement caused by the unintended provision of electric machine torque. Thus, the embedded system of an HEV is safety-critical. The additional electric machine and the high voltage battery of an HEV also further pursue the increasing complexity of automotive embedded systems in terms of number of control units and number of object code instructions. Meanwhile, automotive embedded systems contain about 70 control units and more than 10^8 object code instructions.

The development of increasingly complex safety-critical automotive embedded systems requires appropriate safety engineering workflows aligned with the functional safety standard ISO 26262 for the automotive domain in order to avoid the introduction of systematic faults during development. Furthermore, tool support for automating tedious and error-prone activities is required. Among the typically tool-supported activities required for automotive safety-critical embedded systems development are (a) Early Hazard Analysis, (b) FTA (Fault Tree Analysis), FMEA (Failure Modes and Effects Analysis) and Allocation of Safety Parameters based on System Models as well as (c) Generation of Source Code and Models.

(a) Early Hazard Analysis denotes hazard analysis techniques that are applied earliest in the development process in order to systematically identify, assess and classify potential hazards. In the automotive domain co-called ASILs (Automotive Safety Integrity Levels) are determined depending on the classification of hazards. Based on the results of Early Hazard Analysis, (b) FTA, FMEA and Allocation of Safety Parameters based on System

Models are required in later development steps. FTA and FMEA are analysis techniques that allow assessing potential faults and failures of the embedded system, their propagation and their effects on the vehicle. In line with the results of analyses, an allocation of ASILs to the components of the safety-critical embedded system is necessary in order to determine the rigor of the development process as well as necessary runtime fault detection capabilities of the embedded system's components. Under consideration of (b), embedded software is developed. This is often supported by (c) Generation of Source Code and Models. This comprises source codes determining the behavior of vehicle functions such as recuperation and braking as well as runtime tests generated in order to detect random hardware faults during system operation.

However, tool support for (a), (b) and (c) is improvable. There are approaches to (a) foreseeing the use of structured descriptions that can be subject to automatic checking in order to identify imperfections. However, present approaches do not support automatic corrections. The state of the art with respect to (b) foresees the automatic generation of fault trees and FMEA tables from an underlying model. However, the elaboration of this underlying model is not supported. The state of the art with respect to (b) also foresees the allocation of safety parameters using proprietary algorithms or constraint solvers. However, an approach to the allocation of safety parameters using constraint solvers that considers the specifics of the automotive domain does not yet exist. There are several approaches to (c) foreseeing the generation of models from more abstract models, the generation of source codes from models or the generation of SBSTs (Software-Based Self Tests). Similarly, approaches supporting the generation of models from a more abstract model and the computer-aided configuration and generation of SBST functionality from the same model do not yet exist.

To advance the state of the art with respect to (a), (b) and (c), this thesis presents an approach to Computer-Aided Model-Based Safety Engineering of Automotive Systems. The term Computer-Aided Model-Based Safety Engineering of Automotive Systems denotes a form of model-based safety engineering emphasizing and advocating the automation of analysis and synthesis based on computerized models by means of tool support. Computer-Aided Model-Based Safety Engineering of Automotive Systems comprises the major contributions of this thesis.

Namely the major contributions of this thesis are (1) Safety-Relevant Model Creation, (2) Generation of Fault Trees, FMEA (Failure Modes and Effects Analysis) Tables and ASIL (Automotive Safety Integrity Level) Allocations as well as (3) Configuration and Code Generation. (1) denotes a technique supporting the automatic checking and automatic correction of models and thus improving (a) and (b). (2) comprises tool support for the generation of fault trees and FMEA tables from models. Furthermore, (2) comprises a tool-supported method for the allocation of safety parameters to the components of an automotive system architecture based on a constraint solver and respects the specifics of the automotive domain. (3) denotes an approach that allows generating models from a more abstract model. Based on the same more abstract model, the computer-aided configuration and generation of so-called safety drivers covering SBST functionality is supported.

The tool OASIS (Aut**O**motive **A**nalysis and **S**afety Eng**I**neering In**S**trument) was implemented as a plugin for the tool Papyrus for UML and constitutes an implementation of the major contributions of this thesis. OASIS is a part of a tool chain containing the tools

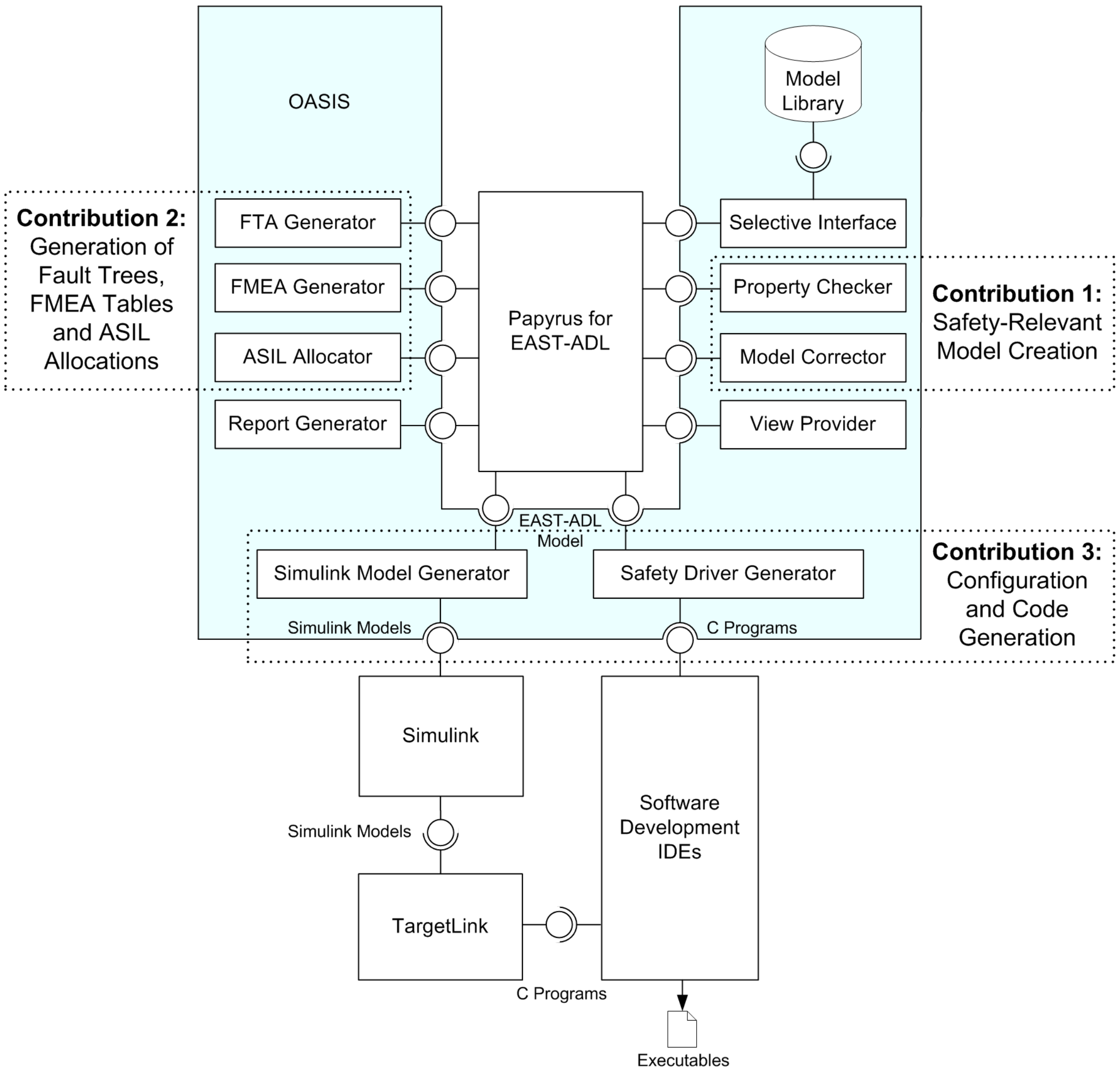


Figure 1: The tool OASIS is part of a tool chain required for the application of the safety engineering workflow. OASIS supports the application of this workflow. Its support for (a) Safety-Relevant Model Creation, its support for (b) Generation of Fault Trees, FMEA Tables and ASIL Allocations and its support for (c) Configuration and Code Generation are the major contributions of this thesis.

Papyrus for UML, OASIS, Matlab/Simulink, TargetLink as well as software development IDEs (Integrating Development Environments). This tool chain supports an automotive safety engineering workflow that is aligned with ISO 26262's Concept Phase, System Level Development Phase and Software Level Development Phase. OASIS and its support for Computer-Aided Model-Based Safety Engineering of Automotive Systems are able to support the application of this safety engineering workflow by simplifying and automating workflow steps. Apart from (1), (2) and (3), OASIS also provides support for Document Generation, Reuse of Modeling Elements and the Provision of Views as these features are of utmost importance for the industrial application of Computer-Aided Model-Based Safety Engineering of Automotive Systems. Figure 1 illustrates the tool chain including OASIS. Furthermore, OASIS' modules constituting the implementation of the major contributions are illustrated.

The safety engineering workflow was experimentally applied using the tool chain including OASIS for the case study of HEV development. During the workflow application, a model was created representing a part of an HEV powertrain consisting of 3 sensors, 3 actuators and 4 control units. This model consists of 957 modeling elements, although only a part of an HEV powertrain was modeled. OASIS was used to automatically derive different entities from the model. Among the derived entities are 160 automatically identified imperfections, 6 fault trees consisting of 245 nodes as well as a safety driver for a multi-core microcontroller consisting of 4224 lines of code. The numbers illustrate the value of OASIS. Although the presented approach does not replace the intellectual process of safety engineering, its capabilities for simplifying and automating workflow steps increase product quality (by identifying and correcting imperfections and by automating error-prone and tedious activities) and reduce required expenditure of time (by rendering manual creations of entities redundant). Thus, Computer-Aided Model-Based Safety Engineering of Automotive Systems can potentially help the industry to create safe products at affordable prices.

Contents

List of Figures

List of Tables

List of Abbreviations

AADL	Architecture Analysis & Design Language
ADL	Architecture Description Language
ARP	Aerospace Recommended Practice
ASIL	Automotive Safety Integrity Level
AVL	Anstalt für Verbrennungskraftmaschinen
BMU	Battery Management Unit
BMW	Bayerische Motoren Werke
CAAM	Combined Architecture Algorithm Model
CAN	Controller Area Network
CCU	Clutch Control Unit
cFMEA	Component Failure Modes and Effects Analysis
COAL	Component-Oriented Architecture Language
CPU	Central Processing Unit
DAL	Design Assurance Level
E/E system	Electrical and/or Electronic system
EAST-ADL	Electronics Architecture and Software Technology Architecture Description Language
EMS	Engine Management System
EV	Electric Vehicle
FHA	Functional Hazard Assessment
FMEA	Failure Modes and Effects Analysis
FMECA	Failure Modes, Effects and Criticality Analysis
FTA	Fault Tree Analysis
HAZOP	Hazard and Operational Studies
HCU	Hybrid Control Unit
HEV	Hybrid Electric Vehicle

I/O	Input/Output
ICE	Internal Combustion Engine
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IP	Intellectual Property
ISA	Instruction Set Architecture
ISO	International Organization for Standardization
MBD	Model-Based Development
MCU	Motor Control Unit
MEPAS	Methods and Processes for Automotive Embedded Software Development, Verification and Validation
OASIS	Automotive Analysis and Safety Engineering Instrument
OCL	Object Constraint Language
PHA	Preliminary Hazard Analysis
RAM	Random Access Memory
RCM	Reliability Configuration Model
RIDL	Reliability Imbedded Design Language
ROM	Read Only Memory
RTL	Register Transfer Level
RTOS	Real Time Operating System
RTSC	Real Time State Chart
SAA	Safety-Aware Architecture
SBST	Software-Based Self Test
SCADE	Safety Critical Application Development Environment
SFTM	Static Fault Tree Model
SIL	Safety Integrity Level
SOC	State of Charge
SoS	System of Systems
STPA	Systems Theoretic Process Analysis
SysML	Systems Modeling Language
TCU	Transmission Control Unit

UML	Unified Modeling Language
USART	Universal Synchronous Asynchronous Receiver Transmitter

Glossary

Application Software
Embedded software determining the behavior of the vehicle functions such as Recuperative Braking or Boost.

Basic Software
Embedded software responsible for tasks like hardware abstraction, initialization, communication, detection of random hardware faults and error-handling.

Computer-Aided Model-based Safety Engineering
A form of model-based safety engineering emphasizing and advocating the automation of analysis and synthesis based on computerized models by means of tool support.

Diagrammatic Language [1]
A language whose expressions include syntactic elements such as boxes, ovals, lines, curves or arrows.

Domain-Specific Language
A programming language or specification language dedicated to a particular problem domain.

E/E (Electrical and/or Electronic) system [2]
System consisting of electrical and (programmable) electronic elements.

Failure [2]
Termination of the ability of an element to perform a function as required.

Fault [2]
Abnormal condition that can cause an element or item to fail.

FMEA (Failure Modes and Effects Analysis) [3]
Inductive analysis technique starting with individual component failures and determining the effects on the overall system.

FTA (Fault Tree Analysis) [3]
Deductive analysis technique starting with identified hazards and tracking them back to possible causative faults.

Functional Safety [2]
: Absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems.

Harm [2]
: Physical injury or damage to the health of persons.

Hazard [2]
: Potential source of harm caused by malfunctioning behavior of the item.

Meta Model
: Defines frames and rules for modeling a predefined class of problems.

Minimum Cut Set [3]
: A set of basic events of a fault tree that cannot be reduced in number and leads to the top event of the fault tree.

Model-based Safety Engineering
: Computerized models are used to support communication, documentation, analysis and synthesis as part of the safety engineering.

PHA (Preliminary Hazard Analysis) [3]
: An analysis technique that is qualitatively applied early in the development process by a team of people with a wide variety of expert knowledge and skills.

Risk [2]
: Combination of the probability of occurrence of harm and the severity of that harm.

Safety [2]
: Absence of unreasonable risk.

Safety Driver
: Part of the basic software required for initialization, runtime fault detection and error-handling of microcontrollers.

Safety Parameter
: A parameter assigned to functions or components of a system under development in order to determine development process rigor or runtime fault detection capabilities.

Security
: The degree of protection against danger, damage, loss, and crime (must take into account the actions of people attempting to cause destruction).

Tool Support
: A software tool or a set of software tools that can be used to effectively apply a workflow.

Unreasonable Risk [2]

Risk judged to be unacceptable in a certain context according to valid societal moral concepts.

Workflow

Description of a systematic sequence of working steps to solve a class of problems.

Chapter 1

Introduction

1.1 Motivation

The automotive industry has been experiencing a shift towards powertrain electrification. The automotive industry pursues this shift to achieve better fuel economy, to reduce emissions and to achieve better drivability. This has led to new powertrain topologies comprising additional components like electric machines or high voltage batteries. Some of these new powertrain topologies are for HEVs and can be classified into series hybrid, parallel hybrid, series-parallel hybrid and complex hybrid depending on their connections between the components that define the energy flow routes and control ports [4].

HEVs are an attempt to combine the advantages of (a) classical vehicles that solely contain an ICE and a petroleum fuel tank for propulsion and power supply and (b) EVs that solely contain an electric machine and a high voltage battery for propulsion and power supply. Whereas the advantages of (a) are good performance and long operating range, its disadvantages are poor fuel economy and environmental pollution. Whereas the advantages of (b) are high energy efficiency and zero environmental pollution, its disadvantages are performance and poor operation range per battery charge.

To overcome the disadvantages of (a) and (b), HEVs contain two power sources (a primary, unidirectional power source and a secondary, bidirectional power source) as well as two energy converters (an unidirectional energy converter and a bidirectional energy converter). Whereas a petroleum fuel tank and an ICE are used as unidirectional power source and power converter, a high voltage battery and an electric machine are usually used as bidirectional power source and power converter. The high voltage battery and the electric machine are useful (1) for regaining (kinetic) energy by using the electric machine as a generator and by charging the high voltage battery. Furthermore, the electric machine is useful (2) for supporting the ICE by providing additive or substitutive torque to the powertrain by using the electric machine as an electric motor. (1) and (2) are two of several operating modes of HEVs that provide additional flexibility and can be used to optimize overall performance, efficiency and emissions with proper configuration and control.

An example for an HEV is AVL's Turbohybrid [5]. This prototype is based on a commercial BMW 320i. It contains a 1.6l turbocharged engine and a 20 kW synchronous electric machine. It was possible to achieve ten percent additional fuel efficiency with respect to comparable BMW cars. The Turbohybrid prototype is illustrated in Figure 1.1.

Figure 1.1: AVL's Turbohybrid prototype is an example for an HEV based on a BMW vehicle. It contains a turbocharged engine and a synchronous electric machine. Figure used with permission of AVL List GmbH.

1.1.1 Safety-Criticality of Embedded Systems in HEVs

The power sources and power converters as well as other vehicle components of HEVs are controlled by embedded systems. The embedded system of an HEV powertrain consists of dozens of control units connected by multiple bus systems.

To illustrate the role of the embedded system in an HEV powertrain, a part of a parallel HEV powertrain including a part of its embedded system is schematically illustrated in Figure 1.2. This HEV powertrain is a pretransmission single-shaft torque combination powertrain [4]. In this configuration, the torques of the engine and the electric machine are modified by the same transmission. An engine, a clutch, an electric machine, a transmission, a final (differential) gear and the wheels of the vehicle are mechanically coupled. Furthermore, the electric machine is electrically coupled to the high voltage battery through an inverter.

The individual powertrain components such as engine, clutch, transmission, inverter and high voltage battery are controlled by control units such as EMS, CCU, TCU, MCU and BMU. These control units communicate via a CAN or FlexRay bus. The HCU is a control unit that coordinates the other control units. For example, the HCU computes the division of the demanded driver torque into the demanded engine torque and the demanded electric machine torque depending on the states of the vehicle such as SOC of the high voltage battery. The demanded torques of the engine and the electric machine are continuously transmitted via a bus system to the EMS and to the MCU that control the engine and the electric machine respectively.

The introduction of HEVs has the following impacts on automotive embedded systems. First, a failure of the embedded system of an HEV can potentially lead to hazards such as explosion of the high voltage battery (caused by overloading of the high voltage battery)

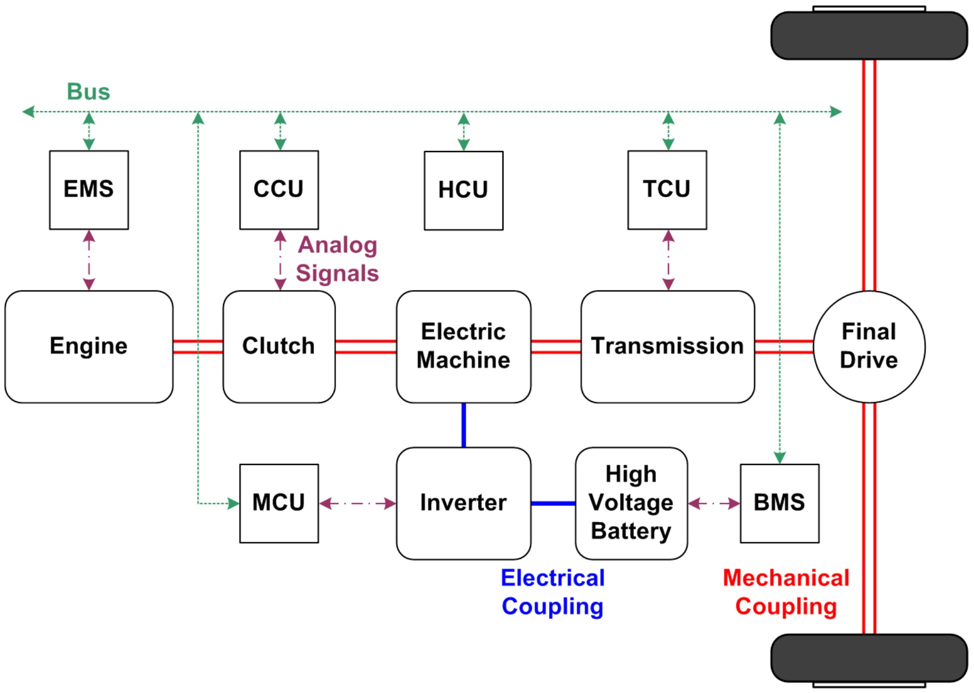


Figure 1.2: Components of an HEV powertrain such as the inverter or the high voltage battery are controlled by a safety-critical embedded system.

and unintended vehicle movement (caused by the unintended provision of electric machine torque). Such potential hazards can harm people, pollute the environment or damage property. Thus, the embedded system of an HEV is *safety-critical* [6].

Second, the additional components of an HEV such as electric machine or high voltage battery require additional control units. This further pursues the ongoing trend of *increasingly complex* automotive embedded systems in terms of number of control units and number of object code instructions. For example, the number of control units in vehicles has increased from about 10 by the end 1980s to about 70 by the end of the 2000s. Also the number of object code instructions of automotive embedded software has increased from more than 10^7 by the end of the 1980s to more than 10^8 by the end of the 2000s [7].

1.1.2 Development of Safety-Critical Embedded Systems in HEVs

Due to their safety-criticality, the embedded system of an HEV is developed according to safety standards such as the automotive functional safety standard ISO 26262 [2]. This standard requires a rigorous development process in order to avoid the introduction of systematic faults during development. For example, this development process comprises additional activities such as analyses like FTA or FMEA that usually cannot be found in standard development processes. Furthermore, the standard ISO 26262 requires embedded systems to be able to detect random hardware faults during runtime in order to achieve and maintain a safe state, if necessary. Runtime fault detection is usually achieved using hardware redundancy, information redundancy or time redundancy.

The development of increasingly complex safety-critical embedded systems requires structured workflows and adequate tool support in order to cope with the complexity. Whereas structured workflows allow systematic and comprehensible development, adequate tool support allows supporting, partially automating or fully automating tedious, time-consuming and error-prone tasks such as the generation of source code. Together, structured workflows and adequate tool support are considered to increase product quality and to reduce the required expenditure of time.

Among the tool-supported activities for safety-critical embedded system development are (a) Early Hazard Analysis, (b) FTA, FMEA and Allocation of Safety Parameters based on System Models as well as (c) Generation of Source Code and Models. In the following, the state of the art with respect to (a), (b) and (c) as well as potential improvements are summarized.

1.1.2.1 Early Hazard Analysis

Development of safety-critical embedded systems requires the early identification, assessment and classification of potential hazards (this process is often denoted as PHA) based on a functional description of the analysis subject and the according derivation of safety requirements to mitigate and control the hazards.

Presently, there are many systematic approaches to early hazard analysis that incorporate the use of models in order to support the process of hazard analysis. Other approaches foresee the use of diagrammatic languages to annotate the models required for hazard analysis. Finally, few approaches foresee the provision of sophisticated tool support allowing automatic checking of structured descriptions (e.g. models) to support the safety engineer. Nevertheless, present tool support does not sustain automatic corrections and is, thus, improvable.

1.1.2.2 FTA, FMEA and Allocation of Safety Parameters based on System Models

Safety-critical embedded system development requires the annotation of a system model describing the architecture and the behavior of the system under development. Architecture and behavior need to be analyzed and verified using analysis techniques such as FTA and FMEA. Presently, there are many approaches supporting FTA and FMEA by automatically generating fault trees and FMEA tables from the system model that is complemented with information about the propagation of the system's faults, failures as well as their propagation. However, the approaches do not support the creation of the underlying model and are, thus, improvable.

In line with the results of analyses, safety parameters like ASILs must be allocated to the components or functions of the system under development. Once allocated, the safety parameters define the rigor of the development process as well as required runtime fault detection capabilities. Tool support exists aiming at automating this allocation taking minimum cut sets as input. Nevertheless, present approaches do not consider the requirements of the automotive domain or they lead to many different allocations that must be investigated manually by the safety engineer. Thus, there is space for improvements.

1.1.2.3 Generation of Source Code and Models

The development of safety-critical embedded systems is often supported by tools allowing to generate source codes or models required for development. First, there are approaches allowing to generate artifacts such as source codes (e.g. application software expressed in C) or models (e.g. Simulink models) from models at higher levels of abstraction. However, these approaches do not support the generation of SBSTs that can be used for runtime detection of random hardware faults in microcontrollers from models at higher levels of abstraction. Thus, these approaches are improvable.

Second, there are approaches foreseeing the generation of SBSTs that can be used for the detection of random hardware faults in a microcontroller's programmable resources like CPU, RAM, ROM or periphery during runtime. Nevertheless, these approaches do neither generate the SBSTs from more abstract models nor do they support the computer-aided configuration of these SBSTs based on the same model. Thus, present approaches are improvable with respect to these points.

1.2 Computer-Aided Model-Based Safety Engineering of Automotive Systems

1.2.1 The MEPAS Project

This thesis is one of the outcomes of the MEPAS project. The involved project partners were AVL List GmbH, the Institute for Technical Informatics at Graz University of Technology and the Virtual Vehicle Competence Center. The goal of the project was to design, implement and evaluate a seamless and systematic development environment for automotive embedded systems.

Among the challenges were (1) the definition of a seamless and systematic development process, (2) the optimization of the interactions between the different methods and tools in order to obtain a seamless development environment, (3) the support of continuous validation during the system development process and (4) the evaluation using an industrial application.

1.2.2 The OASIS Tool

To advance the state of the art in the field of safety-critical embedded systems development with respect to the identified potentials for improvement (see Section 1.1.2) and to support achieving the goals of the MEPAS project (see Section 1.2.1), a tool prototype named OASIS was designed and implemented. OASIS stands for Aut**O**motive Analysis and Safety Eng**I**neering In**S**trument.

This tool supports the application of a safety engineering workflow aligned with the automotive safety standard ISO 26262. OASIS provides features for Computer-Aided Model-Based Safety Engineering of Automotive Systems. These features allow creating consistent and complete work products and to simplify and automate workflow steps. More precisely, it provides support for (a) model creation and reuse, (b) analysis and documentation and (c) configuration and code generation.

OASIS represents an implementation of the major contributions of this thesis and was used for their experimental evaluation using the case study of HEV development. These contributions are listed in the following.

- Support for Safety-Relevant Model Creation
- Support for Generation of Fault Trees, FMEA Tables and ASIL Allocations
- Support for Configuration and Code Generation

Furthermore, OASIS supports features that are of utmost importance for industry and pave the way for the industrial application of the approach, but are not considered to belong to the major contributions of this thesis. These features are listed thereafter.

- Support for Document Generation
- Support for View Provision
- Support for Reuse of Modeling Elements

1.2.3 Organization of this Thesis

The thesis at hand is organized as follows. Chapter 2 reviews related work, identifies potentials for improvements and defines this thesis' major contributions. Chapter 3 describes the safety engineering workflow aligned with ISO 26262, the tool chain supporting the safety engineering workflow as well as the tool OASIS that is part of the tool chain and constitutes an implementation of the major contributions of this thesis. Chapter 4 describes the experimental evaluation of OASIS with respect to the major contributions of the thesis using the case study of HEV development. Chapter 5 concludes the thesis and describes future work. Finally, Chapter 6 contains the publications that contain descriptions of the major contributions of the thesis.

Chapter 2

Related Work

2.1 Early Hazard Analysis

Development of safety-critical embedded systems requires the identification, assessment and classification of potential hazards based on a functional description of the analysis subject (this activity is often denoted as PHA) and the according derivation of safety requirements to mitigate and control the hazards. This can be done early in the development process, even if detailed and quantitative information about the vehicle under development is available insufficiently.

Section 2.1.1 surveys approaches to early hazard analysis foreseeing the use of models. Section 2.1.2 surveys approaches to early hazard analysis foreseeing the use of models that are annotated using diagrammatic languages. Finally, Section 2.1.3 lists approaches foreseeing the use of sophisticated tool support that allows automatic checking.

2.1.1 Systematic Approaches Incorporating Models

The works described in [8, 9, 10, 11, 12, 13, 14] focus on defining systematic approaches that support the intellectual process of identifying and classifying hazards and defining means to mitigate or control them. All of them consider models to be a valuable aid for the application of hazard analysis techniques. They are described thereafter.

In [8, 9], a technique called Actuator Based Hazard Analysis is proposed that can be carried out early in the development process, when only little information concerning the system implementation is available. The approach is based on the assumption that only the actuators of the system can affect their environment. The method defines three fault classes (commission, omission and stuck). Each system effect that describes an undesired enactment of an actuator is defined by a fault class, an analyzed actuator and a user intent. The method defines four severity classes (Catastrophic, Critical, Marginal and Negligible). All severity classes are applied to each actuator, and the distribution between the severity classes is determined. Based on distribution and weighting, a criticality level can be determined that serves as input to the solvability analysis and the design selection that allows choosing the design concept most likely to handle the identified hazards.

Another approach to PHA for automotive systems similar to the one required by ISO 26262 is described in [10]. The approach starts with hazard identification based on a system model. A PASSPORT diagram with supplementary descriptions is used as a system model. A further step of the approach is hazard classification according to severity, controllability and exposure.

In [11], an ISO 26262-compatible approach to PHA is presented. The approach incorporates an architectural model and starts with (1) scope definition. In this phase, safety-critical functions of a vehicle are illustrated in a block diagram including control units, gateways, sensors, actuators and communication systems. The next step is (2) the definition of a role model. A control unit can contribute to multiple functions. In the context of different functions, the control unit may have different roles (e.g. actuation, calculation, monitoring). The next step is (3) the creation of a tabular architectural model. This starts with the mapping of functions onto architectural elements. Subsequently, severity, exposure and controllability are evaluated and an ASIL is determined for each function. Then, roles (depending on functions) are assigned to each architectural element. Finally, each architectural element has roles with corresponding ASILs.

The work proposed in [12] describes an approach to hazard analysis of safety-critical software-intensive systems called STPA for early application in the development process. The approach starts with the identification of hazards and related requirements or constraints. Subsequently, inadequate control actions, control flaws and inadequate control executions that lead to inadequate control actions are identified. This is an input to a design process aiming at creating new constraints, refining existing constraints, creating a new design or modifying the existing design until all hazards are eliminated, mitigated or controlled. This process is iterative. The approach relies on a model describing the control flow of the system under analysis and causes of accidents. The applicability of the approach is illustrated using a spaceflight application.

In [13], an approach to hazard analysis of SoS is described that is intended to be applied early in the development process. This hazard analysis technique is focused on the interfaces between the particular systems. The approach is based on a model of the SoS as well as guidewords. Input/Output Analysis as well as Network Analysis are carried out in the course of the hazard analysis. The probability of the occurrence of accidents is assessed. A validation framework is established incorporating the definition of goals. Based on these goals, metrics (e.g. percentage software safety requirements traceable to hazards) are defined indicating the quality of the conducted hazard analysis. Some of the defined metrics depend on knowledge gained from previous analyses.

A methodology for safety-critical systems development is proposed in [14]. Amongst other activities, this methodology requires the identification of those functions that are safety-critical. Thereafter, hazards are identified, risks are assessed and risk mitigation means are defined and associated early in the development process. The work proposes metrics based on the identified hazards. An example is the metric *percentage software hazards* defined as number of software safety hazards divided by the number of system safety hazards. The approach is evaluated using a railway application.

2.1.2 Incorporation of Diagrammatic Languages

In contrast to aforementioned related works, [15, 16, 17] explicitly refer to the use of diagrammatic languages such as UML, SysML or EAST-ADL to support identification, assessment and classification of hazards of a system and the derivation of safety requirements. The remainder of this section describes these related works.

An approach combining hazard analysis and the use of a diagrammatic language to create models is described in [15]. A subset of UML (component and deployment diagrams) is used to support hazard analysis at an early design stage. Boolean logic is used to formally model hazards and failure propagation. Starting with a component model of the system to be analyzed, (1) fault trees for all system hazards are derived. Subsequently, (2) the propagation of component failures is analyzed for each component. Then, (3) related behavior of deployment nodes and hardware devices has to be derived. Finally, (4) boolean equations can be used to apply analysis techniques. The approach allows identifying the most serious hazards and failures and to determine components requiring a more detailed safety analysis and assumed restrictions to fault propagation. This facilitates the systematic derivation of safety requirements.

The work described in [16] aims at solving the problems posed by the derivation of safety requirements and by conducting hazard analysis. The first step is the identification and description of functions associated with the level under study. UML use cases and scenarios are used for function description. The second step is the failure identification. In this step, a technique is applied that is inspired by techniques such as FHA that is typically applied early in the development process and makes use of guidewords. In the third step, based on the analysis, new safety-related functional requirements are identified. The approach was evaluated using an avionics use case.

An approach to PHA using EAST-ADL is proposed in [17]. A workflow is presented that starts with the description of the functions (e.g. Cruise Control) of the vehicle, their operation needs and other stakeholder requirements. Thereafter, a feature tree model is used to structure the vehicle functions. After the allocation of requirements to the features, the vehicle is well determined in terms of its requirements, functions and modes. This is the input to the identification and classification of hazards based on the functions and their related requirements. Finally, safety goals are derived constituting top-level safety requirements.

2.1.3 Provision of Sophisticated Tool Support

In contrast to aforementioned related works, approaches that allow conducting early hazard analysis in the context of more sophisticated tool support are defined in [18, 19, 20, 21]. These approaches are described in the following.

The authors of [18] describe a prototype tool named HazLog that aims at supporting hazard management in accordance with the safety standard Def(Aust) 5679. Hazard management refers to documenting and tracking hazards and their associated resolutions across the entire life cycle of a safety-critical system. Hazard management in accordance with Def(Aust) 5679 requires the application of PHA that is applied early in the development process. HazLog is built on the requirements management tool DOORS and makes use of DOORS' capability of storing and linking information. HazLog foresees the structured description of hazards in accordance with a conceptual model similar to a meta model. Data integrity checks are supported by HazLog allowing to check the consistency of the structured description.

The work proposed in [17] in combination with the tools [19] and [20] allows the definition of properties using OCL. The combined approach allows the definition and checking of properties on demand.

Tools that aim at supporting the safety standard ISO 26262 are reviewed in [21]. Among the reviewed tools is a tool named Medini Analyze following an MBD approach. It supports the definition of vehicle functions and the application of hazard analysis early in the development process. The tool allows defining constraints using the OCL language that can automatically be validated on demand. Besides a predefined set of checking rules, users can define their own rules.

2.2 FTA, FMEA and Allocation of Safety Parameters based on System Models

In course of the development of a safety-critical embedded system, a system model needs to be annotated describing the architecture of the system under development, its interacting environment and its behavior. The system architecture and the system behavior need to be analyzed and verified using analysis techniques such as FTA and FMEA. Whereas FTA is a deductive analysis technique, FMEA is an inductive analysis technique. The techniques complement each other. Their qualitative application is especially useful in early development phases, when less quantitative information about the vehicle, its embedded system, its sensors and actuators is available. Section 2.2.1 surveys approaches that attempt to support the application of FTA and FMEA by the generation of fault trees and FMEA tables from system models.

Various safety standards attempt to quantify functional safety by defining discrete safety parameters such as SIL (IEC 61508), ASIL (ISO 26262) or DAL (ARP4754a). In line with the results of analyses, these parameters must be allocated to the functions or components of the safety-critical embedded system under development. Once allocated, they define the rigor of the required development process or determine the necessary runtime fault detection capabilities of the safety-critical embedded system. The allocation of these safety parameters to the functions or the components of a safety-critical embedded system is challenging. Thus, tool-supported approaches exist aiming at automating this allocation taking minimum cut sets as input. Section 2.2.2 presents two methods for the allocation of safety parameters to functions or components of a system architecture.

2.2.1 Generation of Fault Trees and FMEA Tables from System Models

The approaches explained in [22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35] use models describing structure and/or behavior of a system (typically of a computer-based system that shall be dependable, reliable or safe). These models are complemented with quantitative or qualitative information concerning the behavior of the system in presence of faults (typically about faults and failures and their propagation). These underlying models are used by all approaches as input to fault tree generation and/or FMEA table generation, supporting the application of FTA and/or FMEA. The approaches are described thereafter.

In [22], a graphical design language named RIDL for the modeling of digital systems is presented. This language allows embedding redundancy and failure information within block schematics. An algorithm is presented that allows generating fault trees from RIDL models. This capability allows to do design and analysis in parallel.

An approach that combines system architecture modeling and FTA is described in [23]. The approach allows continuous assessment of an evolving system design. A system model is input to HAZOP. Each component of the system model is analyzed, and component failure modes are determined. The HAZOP results in a model defining failure modes that can be observed at the component outputs as results of internal component malfunctions as well as deviating component inputs. In [24], an extension of [23] is presented that allows FMEA table generation. In [25], the extended approach is integrated with an EAST-ADL modeling tool using a model transformation technique. This allows generation of fault trees and FMEA tables from EAST-ADL models.

In [26], an approach to fault tree generation from system models is presented. The approach foresees performing reliability analysis in parallel with system design and using UML to model fault-tolerant, software-intensive systems. A modeling methodology is presented requiring the use of stereotypes to express concepts such as hardware, redundancy, spares, dependencies and reconfiguration. A three-pass algorithm is presented that allows generating fault tree code.

The authors of [27] integrate architectural modeling languages with safety analysis languages to improve consistency. When a safety-critical software architecture is developed, an initial architecture is proposed. This architecture is annotated and enriched with safety-relevant information. Safety analysis of the architecture is carried out. Results influence the software architecture. This design and analysis process is cyclic. A meta model for component-based SAAs is available allowing to complement architectural descriptions with safety-relevant information such as safety objectives and mitigation means. Meta models for FTA and FMECA are proposed. A tool implementation is presented allowing the generation of FTA models and FMECA models from an SAA model.

A methodology combining safety analyses and a component-oriented, model-based software engineering approach is described in [28]. The authors aim at supporting safety analyses in the earlier stages of development. A hierarchical model for component-based software engineering is available. The model allows defining a failure specification and a failure realization as well as a functional specification and a functional realization for each software component. Fault trees can be generated from the component model.

In [29] tool support for automated FMEA generation is presented. Input to the presented method is a component model of a system including so-called safety interfaces that can automatically be generated. Safety interfaces can be seen as formal descriptions of the components in terms of failures affecting the components. From the safety interface descriptions, cFMEAs can be created for each component. Subsequently, the cFMEAs are input to the generation of a system-level FMEA.

In [30], an approach to fault tree generation is described requiring the creation of a model of the system under investigation. This model describes system structure, system behavior as well as the flows of information and energy through the system. Moreover, top events are defined for system parameters such as component inputs or component outputs. This model is input to a trace-back algorithm generating a fault tree.

The authors of [31] present a novel methodology for the construction of fault trees from system Simulink models. The methodology foresees the manual creation of a Simulink model and the according complementation of the model with other information required for fault tree generation. This model is input to fault tree generation.

The authors of [32] describe a tool set named COMPASS making use of a formal semantics for AADL. The approach foresees the creation of a hierarchical system model describing the behavior of the system under normal conditions. This model is complemented with an error model expressing how the system can fail. The presented tool set includes the capability of generating fault trees and FMEA tables from the system model and the error model. In [33], the COMPASS tool set was evaluated using the case study of a satellite platform under development. Among other activities, a fault tree consisting of 66 nodes and an FMEA table were automatically generated in course of this case study.

In [34], an approach to fault tree generation from UML models is presented. The approach aims at supporting reliability analyses in early design stages, when the overall system architecture is still subject to refinement. They separate application-independent information from application-dependent information to sustain reuse and to avoid remodeling. To achieve this, separate UML-profiles for architectural models and application models are used. These models are input to fault tree generation.

In [35], an approach to the automatic generation of static fault trees from system models that are specified with SysML is described. The authors use internal block diagrams and sequence diagrams to describe a system. These diagrams are an input the automatic generation of an RCM. Then, an SFTM is developed to generate static fault trees from the RCM specifications. The approach is experimentally evaluated using the case study of a fault-tolerant parallel processor.

2.2.2 Allocation of Safety Parameters

The works presented in [36, 37] address the problem of allocating safety parameters like ASILs or DALs to the functions or the components of a system architecture under consideration of safety and costs. In the following, these approaches are described.

In [36], an approach to the automatic allocation of SILs to subsystems and components of complex hierarchical networked architectures is presented. The approach can be used in the context of development using EAST-ADL. The approach supports ASIL decomposition [2]. Thus, if a component contributes to a failure only in conjunction with other components, it may receive a lower ASIL than a component directly causing the failure. ASILs of functions as well as minimum cut sets extracted from fault trees are an input to an algorithm computing possible allocations of ASILs to the components of a system architecture. The most economic potential allocations are presented to the user.

In [37], a method and an according tool implementation for the allocation of DALs to software and hardware functions of an aircraft according to ARP4754a (Guidelines for Development of Civil Aircraft and Systems) are presented. The allocated DALs determine the applicable portfolio of techniques for the avoidance of systematic faults during the development process according to DO178B. Thus, the allocated DALs affect the development costs. Input to the presented method are identified and assessed system-level hazards, minimum cut sets that can lead to the system-level hazards as well as additional preferences of the safety engineer. Constraints are derived according to a formalization of the DAL allocation rules and impose two constraint satisfaction problems. These problems are input to a constraint solver that is part of the presented tool implementation. The tool implementation can be used to propose economic allocations or to verify existing allocations.

2.3 Generation of Source Code and Models

The development of safety-critical embedded systems requires the creation of different kinds of models and source codes in course of the development process. This is often supported by tools that automate the generation of source codes or models. First, there are approaches foreseeing the automatic generation of artifacts like source codes or models from other, more abstract models. For example, there are approaches foreseeing the automatic generation of Lustre models or Simulink models from more abstract models like UML models. Section 2.3.1 surveys approaches that attempt to support safety-critical embedded systems development by automatically generating models or source code from more abstract models.

Second, other approaches foresee the generation of non-intrusive SBSTs that can be used for the runtime detection of random hardware faults in a microcontroller's programmable resources like CPU, RAM, ROM or periphery by executing test patterns defined using the processor's instruction set [38]. In the field of safety-critical systems, this is necessary to achieve and maintain a safe state, if necessary. SBSTs can be divided into structural and functional SBSTs. Structural SBSTs proved to be more effective. However, their applicability is limited because they require detailed structural information about the microcontroller like gate-level net lists. In contrast, more abstract information about the processor such as the ISA is sufficient to apply functional SBSTs. Thus, functional SBSTs are more suited to be applied for industrial microcontrollers because vendors usually do not release detailed structural information such as gate-level net lists to protect their IP. Section 2.3.2 surveys approaches that aim at generating functional SBSTs for detecting random hardware faults in microcontrollers.

2.3.1 Generation of Source Code and Models From Other Models

The works in [39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50] describe approaches foreseeing the automatic generation of source codes from models or the automatic generation of models from more abstract models to support safety-critical embedded systems development. The remainder of this section is devoted to the description of these approaches.

In [39], a tool suite named SCADE is described. This tool provides a modeling environment that can be used to annotate models using a graphical language based on the language Lustre. These models are input to SCADE's certified code generator that can generate source code for safety-critical systems. The author advocates the combination of SCADE with UML-based languages like SysML and states that SCADE's graphical language and SysML complement each other well.

In [40], a layered approach to support the development of safety-critical, distributed control systems using model and code generation is presented. The approach comprises (1) a high-level modeling and simulation layer, (2) a middle-level programming and validation layer and (3) a low-level execution layer. Simulink is used for (1), SCADE/Lustre is used for (2), and (3) follows the principles of time-triggered architecture. The authors present algorithms and tool support to traverse from one layer to the next. The extension of the approach supporting a safe subset of Stateflow is described in [41].

In [42], an approach is outlined that foresees the translation of STATEMATE models to SCADE/Lustre in order to allow generating source code for safety-critical embedded systems using SCADE's certified code generator. STATEMATE allows defining behavior in terms of activity charts and state charts. A translation strategy is presented. Furthermore, a concept for the verification of the translation is presented based on testing.

An approach to model-driven development of safety-critical real-time systems is presented in [43]. In this approach, the structure of a safety-critical system is annotated using UML component diagrams. The behavior of the same system is annotated using RTSCs that are enhanced UML state machines. A process is outlined that involves the use of a model checker to verify an annotated system. Furthermore, the process foresees the use of a code generator in order to generate source code for the safety-critical real-time system from the UML/RTSC model.

In [44], an approach to the generation of code from models in order to support safety-critical embedded system development is presented. The approach is user-extensible and is based on the use of generic templates that cover recurring aspects of safety-critical embedded software development. The code generator can adapt these templates depending on the needs of the application annotated by an application engineer in the form of a model. Furthermore, the application developer needs to develop application code. Finally, the adapted templates together with the application code form the generated code.

In [45], an embedded software design flow is described foreseeing the use of UML for the modeling of the whole system. A UML model is input to a model transformation resulting in an executable and synthesizable Simulink CAAM. The Simulink CAAM is then input to a code generator to generate source code. The approach supports the automatic allocation of processors, the automatic mapping of threads onto processors and the automatic insertion of Simulink modeling elements. Two case studies are presented to show the applicability of the approach.

In [46], a tool-suite named OCARINA is presented supporting the development of safety-critical distributed applications using AADL. The tool-suite allows the syntactic and semantic analysis of AADL models. It supports the generation of ADA code from the AADL models. Therefore, an ADA subset is used guaranteeing schedulability and safety. Furthermore, OCARINA is open for tool integration and supports a middleware targeting high-integrity systems.

In [47], a design approach is described relying on code generation techniques in order to implement complex, adaptive and critical systems. The approach foresees the representation of the system's dynamic behavior using COAL. COAL allows (a) enumerating the system's operational modes, (b) representing mode switches as communicating mode automata and (c) specifying valid and invalid architecture characteristics for each mode. Resulting COAL models can be used by MyCCM-High Integrity (a component-based framework dedicated to critical and adaptive systems) to generate AADL models that can be further processed by OCARINA in order to generate source code.

The work presented in [48] is focused on component-based development of embedded systems. A software development process is supported by a tool chain called Save-IDE. The software development process is a topdown approach with an emphasis on reusability. It consists of phases for design, analysis and realization. The tool chain allows component-based design using the component model SafeCCM. For analysis, a Timed Automata Editor, a simulator and a model-checker are available. Moreover, the Save-IDE allows synthesizing from SafeCCM towards an abstraction layer that is capable of abstracting different operating systems and hardware platforms.

The authors of [49] present a software component model named ProCom that aims at safety-critical real-time embedded systems. This software component model relies on asynchronous, formal semantics. A code generation strategy is presented that allows generating source code from models annotated using ProCom. Furthermore, an approach to formalize the generated source code is presented. This allows using model checking to prove that the generated code preserves the asynchronous semantics of ProCom.

In [50], an approach integrating diagnosis functionality such as SBST into a holistic design space exploration of automotive E/E-architectures at system level is presented. The aim of design space exploration is to support the system designer in finding optimal system designs. A design flow is proposed that foresees the creation of a model describing (a) an application representing an algorithmic problem description and (b) an architecture representing available hardware components. In course of design space exploration mappings of (a) onto (b) are automatically generated using system synthesis that makes use of constraint solvers. These mappings are optimized with respect to multiple and potentially contradicting design objectives (e.g. test quality and energy consumption) while meeting all given design constraints. The approach was experimentally evaluated using the case study of an automotive subnet that requires that application of SBSTs.

2.3.2 Generation of Functional SBSTs for Microcontrollers

The works [51, 52, 53, 54, 55] present approaches to the automatic generation of functional SBSTs for microcontrollers in order to support the detection of random hardware faults during runtime. The approaches do not require low level descriptions such as gate level net lists and are, thus, likely to be applicable for industrial microcontrollers.

In [51], an approach to functional SBST without assuming an a priori fault model is described. The instruction set of the programmer's manual is the main input to the methodology. The instruction set with additional information is transformed to machine-readable form. Subsequently, this description is input to a code generator that can generate an instruction sequence enumerating all combinations of operations and systematically selected operands. Also randomization is supported. In case of closed-loop testing, the instruction sequences are able to perform SBST of the processor. In that case, signatures are created from the test responses that can be compared to stored, golden signatures that were computed offline.

The work described in [52] proposes a methodology for functional SBST of the peripherals of systems-on-chip. The approach does not rely on a fault model and exclusively uses the instruction set of the included processor for testing. The approach uses a signature compression method to compress test results in order to decrease memory consumption. Each peripheral communicates with the processor via the system bus. Using this bus, the processor can issue write commands and read the status of the peripheral. Therefore, the processor has dedicated instructions to write or read the ports of the peripheral. The approach was evaluated using a USART peripheral.

The authors of [53] describe another approach to functional SBST of microprocessors. For testing, the cache of the processor is used. Before testing, a functional test is loaded into the processor cache, while the processor is in test mode. Then the processor enters the normal mode, and the test is executed. Finally, the processor enters test mode again, and the results are unloaded. Test programs are so-called kernels that are able to generate pseudo-random instructions sequences as well as operands during runtime of the processor. The test results are observable in the form of memory dumps at the end of the test. Then, test results can be verified by a comparison with golden responses.

In [54], an approach to functional SBST of microprocessors is proposed. The method uses an instruction library that describes the assembly language syntax of the microprocessor to be tested by listing each instruction with the correct operands. Additionally, a simulator is used that can evaluate the effectiveness of test programs with respect to a coverage metric based on an RTL-level description of the processor. This metric is RTL-instantiated statement coverage. A genetic algorithm uses the library and the simulator to iteratively create new test programs and, finally, selects the best test program with respect to the coverage metric and the test program length. The authors evaluated their approach using a Leon2 microprocessor including a pipeline to show that the proposed approach can achieve 100% RTL-instantiated statement coverage.

In [55], an approach to cache-based functional SBST is presented. A testing approach named Load&Go foresees the insertion of code and data into the internal cache of the processor and directing the execution flow on a particular code stream. Test results are emitted as signatures via a serial port of the processor. This approach requires (a) test programs and (b) knowledge about the cache contents required for the execution of each test program. For (a), automatically generated, pseudo-random instruction sequences are used and for (b) a test conversion strategy is defined that incorporates the use of an RTL simulation. This RTL simulation is used to find out which cache contents need to be loaded for which test. Thus, an executable RTL specification of the processor is required.

2.4 Potentials for Improvement

Based on the related work that is summarized in the Sections 2.1, 2.2 and 2.3, this section identifies potentials for improvement with respect to reviewed related work. In the following, the potentials for improvement are described.

2.4.1 Potential for Improvement 1

Related work concerning Early Hazard Analysis is presented in Section 2.1. Whereas the works described in Section 2.1.1 are focused on describing systematic approaches foreseeing the use of models, the works reviewed in Section 2.1.2 go one step further and define systematic approaches foreseeing the use of diagrammatic languages such as UML, SysML or EAST-ADL to annotate models using tool support. The works reviewed in Section 2.1.3 present more sophisticated tool support that allows checking structured descriptions (e.g. models) of hazards and related artifacts for consistency and completeness to sustain the safety engineer in identifying imperfections. Nevertheless, a *potential improvement* is to support not only the automatic identification of imperfections but also the automatic identification and application of corrective measures.

2.4.2 Potential for Improvement 2

Section 2.2.1 reviews approaches supporting the application of FTA and FMEA using fault tree generation and/or FMEA table generation. Input to the generation of fault trees and

FMEA tables are models describing structure and/or behavior of a system (typically of a computer-based system that shall be dependable, reliable or safe). The models are complemented with quantitative or qualitative descriptions of the behavior of the system in presence of faults and failures (typically about faults and failures and their propagation). However, a *potential improvement* is to support the application of FTA and FMEA not only by generating fault trees and/or FMEA tables from a system model, but also to aid the creation of the system model by the automatic identification of imperfections and the automatic application of corrective measures.

2.4.3 Potential for Improvement 3

Section 2.2.2 reviews works aiming at providing tool support for the allocation of safety parameters such as SILs, ASILs or DALs to the functions or components of a system under development. One work is based on the use of minimum cut sets and makes use of constraint solvers, but focuses on the aerospace domain and proposes a method for the allocation of DALs to functions of an aircraft. Thus, this approach is not applicable for the automotive domain that is constrained by different regulations. Another approach considers the regulations applying to the automotive domain. However, this approach is based on a proprietary algorithm requiring minimum cut sets and potentially leading to many different allocations that must be manually investigated by the safety engineer. Consequently, a *potential improvement* is a tool-supported method considering the regulations of the automotive domain while making use of constraint solvers in order to lead to a single, optimal solution.

2.4.4 Potential for Improvement 4

Section 2.3 reviews works that aim at supporting safety-critical embedded systems development by generating source codes and models. The works reviewed in Section 2.3.1 present approaches foreseeing the generation of source codes from models or the generation of models from more abstract models. Although some of these works consider the necessity for SBSTs, none consider generating source codes for SBSTs from models. The works reviewed in Section 2.3.2 support the generation of functional SBSTs to support runtime detection of random hardware faults in microcontrollers, but they do not consider the computer-aided configuration and generation of SBSTs based on a model. Thus, a *potential improvement* is to support safety-critical embedded systems development by generating models from a more abstract model as well as to support the configuration of SBSTs and the according generation of source codes for SBSTs from the same model.

2.5 Contribution and Significance

This section describes how this thesis contributes unveiling the potentials for improvement described in Section 2.4. The major contributions of this thesis were implemented and form the tool OASIS (see Section 3.2.2). OASIS is part of a tool chain (see Section 3.2) supporting an automotive safety engineering workflow (see Section 3.1) aligned with ISO 26262. Furthermore, OASIS' features support Computer-Aided Model-Based Safety Engineering of Automotive Systems comprising the following.

2.5.1 Contribution 1: Safety-Relevant Model Creation

Safety-Relevant Model Creation unveils Potential Improvement 1 and Potential Improvement 2. In particular, this contribution constitutes sophisticated tool support for the elaboration of a model describing a system and complementing artifacts such as hazards, faults and failures. This is achieved by *automatic property checking and automatic model correction*. First, this further improves the application of early hazard analysis. Second, this improves the application of FTA and FMEA in combination with support for fault tree generation and FMEA table generation (see also Contribution 2). Contribution 1 is described in Section 3.3.

2.5.2 Contribution 2: Generation of Fault Trees, FMEA Tables and ASIL Allocations

Generation of Fault Trees, FMEA Tables and ASIL Allocations unveils Potential Improvement 2 and Potential Improvement 3. More precisely, this contribution supports the application of FTA and FMEA by *fault tree generation and FMEA table generation* combined with automatic property checking and automatic model correction (see also Contribution 1). Furthermore, minimum cut sets can automatically be extracted from the fault trees and can be input to a tool-supported method for the *allocation of ASILs* to the components of an automotive system architecture. This method is based on the interpretation of ASIL allocation as an optimization problem and foresees the use of a constraint solver to solve this problem leading to a single, optimal ASIL allocation. Contribution 2 is described in Section 3.4.

2.5.3 Contribution 3: Configuration and Code Generation

Configuration and Code Generation unveils Potential Improvement 4. This contribution constitutes an approach that foresees supporting safety-critical embedded systems development by providing support for the generation of models and source code. In particular, this approach foresees the *generation of models from an abstract model* in order to support application software development. Moreover, this approach foresees the *computer-aided configuration of safety drivers covering the functionality of SBSTs and the according generation of source code* based on the same abstract model to support basic software development. Contribution 3 is described in Section 3.5.

Chapter 3

Computer-Aided Model-Based Safety Engineering of Automotive Systems

3.1 Safety Engineering Workflow

This section describes a safety engineering workflow (see Figure 3.1) aligned with the safety standard ISO 26262. This workflow can be subdivided into multiple phases and allows iterations. It covers activities required by ISO 26262's Concept Phase, its System Level Development Phase and its Software Level Development Phase (other activities and phases required by ISO 26262 but not covered by the workflow are outside the scope of this thesis). The remainder of this section describes the phases of the safety engineering workflow.

3.1.1 Concept Phase

The Concept Phase of the safety engineering workflow foresees the functional description of the vehicle under development and the application of PHA. PHA is an analysis technique that is qualitatively applied early in the development process. The application of PHA aims at the identification, classification and assessment of potential hazards of a newly developed vehicle. In addition, the derivation of top-level safety requirements and functional safety requirements is required.

3.1.1.1 Definition of the Analysis Subject

Information about the vehicle under development is collected and modeled. Functions of the vehicle (e.g. motoring or recuperative braking) are defined. Requirements on the functions are determined and allocated (e.g. conditions for activation or deactivation). In addition, relevant modes (e.g. drive, creep or acceleration) are identified for each function and associated with the requirements.

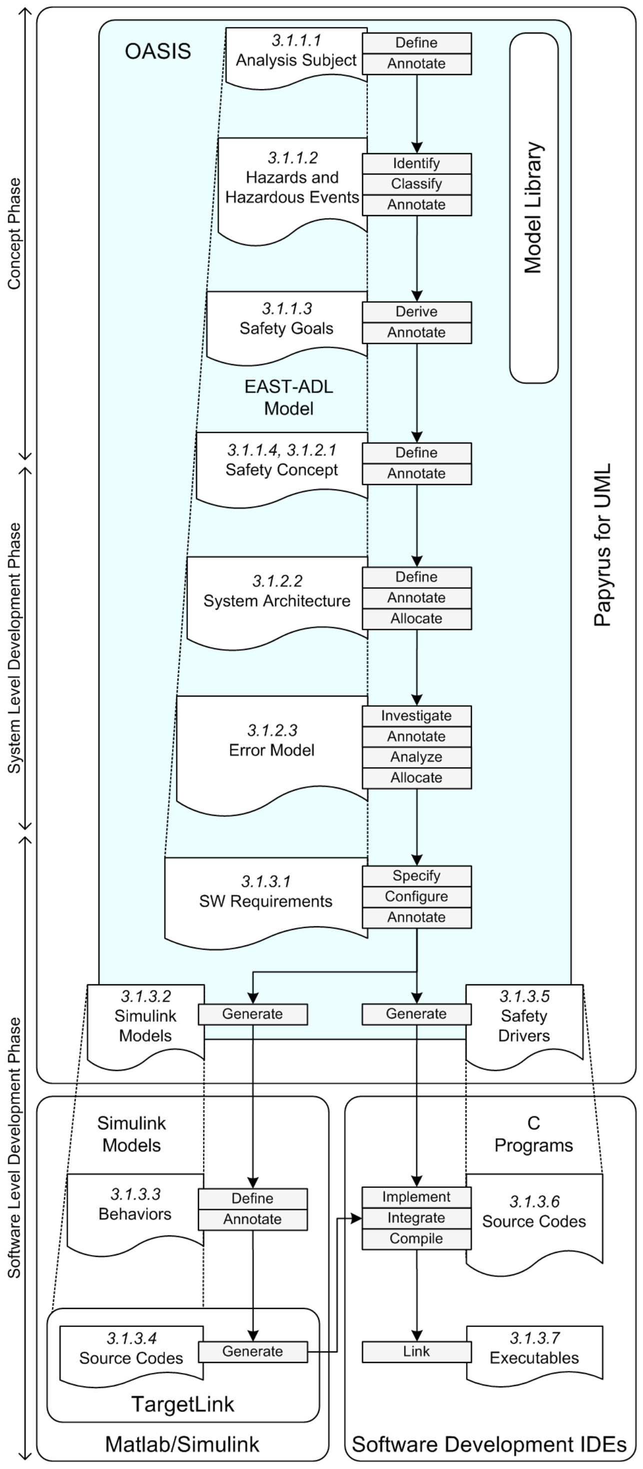


Figure 3.1: The safety engineering workflow is aligned with ISO 26262 and supported by a tool chain. The tools support the languages required for the annotation of the work products of the workflow.

3.1.1.2 Identification of Hazards and Hazardous Events

Based on the definition of the analysis subject, possible malfunctions are identified. Hazards are derived for each malfunction (e.g. unintended acceleration of the vehicle). Thereafter, operational situations such as traffic situations (e.g. oncoming traffic on a highway in a curve) and maintenance situations (e.g. vehicle at lifting ramp) are defined. Moreover, use cases describing the behavior (e.g. overtaking or changing oil) of the related actors (e.g. driver or mechanic) are described. Hazardous events are determined for relevant combinations of hazards, use cases and operational situations. A library of relevant use cases and operational situations from earlier workflow applications is used as input. If necessary, this library is updated for future workflow applications. Relevant modes are identified for each hazardous event. The criticality of each hazardous event is assessed in terms of controllability, severity and exposure, and an ASIL is determined.

3.1.1.3 Derivation of Safety Goals

For each hazardous event classified as ASIL A, ASIL B, ASIL C or ASIL D, a safety goal is derived and associated. Furthermore, a safe state is defined (e.g. switch open) for each safety goal. Alternatively, a safe mode (e.g. limp home mode) is determined. The determined safety goals are top-level safety requirements.

3.1.1.4 Definition of Functional Safety Concept

The functional safety concept is derived from the safety goals. The functional safety concept consists of functional safety requirements on the automotive embedded system, connected sensors and controlled actuators. Traces are created between safety goals and derived functional safety requirements as well as among functional safety requirements, if necessary.

3.1.2 System Level Development Phase

The System Level Development Phase focuses on the definition of the system architecture, the allocation of requirements and the application of FTA and FMEA. The analysis techniques complement each other. Their qualitative application is especially useful in early development phases when less quantitative information about the vehicle, its embedded system, its sensors and actuators is available. In line with results of analyses such as PHA and FTA, an allocation of ASILs to the components of the system architecture must be carried out. Once allocated, they determine applicable requirements of ISO 26262 and the necessary safety measures of the system components to avoid unreasonable residual risk.

3.1.2.1 Definition of Technical Safety Concept

The technical safety concept is derived from the functional safety concept. The technical safety concept consists of technical safety requirements. Traces are created between functional safety requirements and derived technical safety requirements as well as among technical safety requirements, if necessary.

3.1.2.2 Definition of System Architecture

The system architecture is defined in terms of the embedded system, connected sensors and controlled actuators. Moreover, the components of the environment interacting with the sensors and actuators are modeled. Thereafter, functional and technical safety requirements and functions are allocated to the components of the system architecture.

3.1.2.3 Investigation and Annotation of Faults and Failures

Information flows and energy flows through the system architecture and its environment are investigated. Possible faults and failures are estimated, and their propagation is analyzed and annotated. The estimated faults can either be random hardware faults or potential process faults introduced during the development process. Moreover, it is investigated and annotated how the failures lead to the malfunctions of the system. The system architecture is analyzed and verified using qualitative FTA and FMEA. Finally, ASILs are allocated to the components of the system architecture.

3.1.3 Software Level Development Phase

The Software Level Development Phase is focused on the derivation of software requirements and the according implementation, generation and integration of software. Developed software must be divided into application software and basic software designed and implemented in different manners. Whereas application software determines the behavior of the vehicle functions, basic software fulfills tasks like hardware abstraction, initialization, communication, detection of random hardware faults and error-handling. The part of the basic software for a component of the system architecture dedicated to safe initialization, runtime fault detection and error-handling is referred to as safety driver. Thereafter, the subphases of the Concept Phase, the System Level Development Phase and the Software Level Development Phase are described.

3.1.3.1 Specification of Embedded Software

It is defined which components of the system architecture are required to execute software. Requirements (including safety requirements) on the application software and the basic software are defined and allocated to these components. Safety drivers are configured for the components of the system architecture that are required to execute basic software. Traceability links are created if a software requirement is derived from any other requirements (e.g. from the functional or technical safety concept). A report is generated that lists software requirements, describes their allocation and the work products of the preceding workflow phases.

3.1.3.2 Generation of Simulink Models

Simulink models are generated to serve as basis for application software development. They reflect the structure of the components of the system architecture required to execute application software.

3.1.3.3 Definition of Behaviors

Application software is designed and implemented based on the generated Simulink models for the components of the system architecture. Allocated software requirements are considered. The enhanced and refined Simulink models determine the behavior of the components required to execute application software.

3.1.3.4 Generation of Source Codes

Program code is generated from the enhanced and refined Simulink models for the components of the system architecture required to execute the application software.

3.1.3.5 Generation of Safety Drivers

Based on their configurations, safety drivers are generated for the components of the system architecture required to execute basic software. These safety drivers determine the random hardware fault detection capabilities and error-handling capabilities of the components of the system architecture that are required to execute basic software.

3.1.3.6 Implementation and Integration of Source Codes

Program code is manually implemented. Afterwards, program code for application software and code for basic software like RTOS, legacy code, I/O-related code or generated safety drivers is integrated. This is done by considering allocated basic software requirements for the components of the system architecture required to execute basic software.

3.1.3.7 Compilation and Linking of Executable

The created software is compiled and linked to form executables for the components of the system architecture required to execute software.

3.2 Supporting Tool Chain

The safety engineering workflow described in Section 3.1 is supported by a tool chain. The tools of the tool chain support different languages required for the annotation of the work products of the workflow. Figure 3.1 maps the phases of the safety engineering workflow on the tools and languages necessary for the workflow application. The remainder of this section describes the tools, languages and the mapping.

3.2.1 Papyrus for UML

Papyrus for UML [19] is an Eclipse-based open-source tool that allows UML modeling as well as the definition of UML profiles. An open-source plugin is available that allows creating EAST-ADL models. EAST-ADL is a domain-specific language and adapted to the needs of the automotive domain. It is diagrammatic such as UML. Its abstract syntax is defined by its meta model, and its semantic domain and semantic mapping are defined using natural language. EAST-ADL allows (a) describing a system architecture from different viewpoints and on different levels of abstraction, (b) to express work products

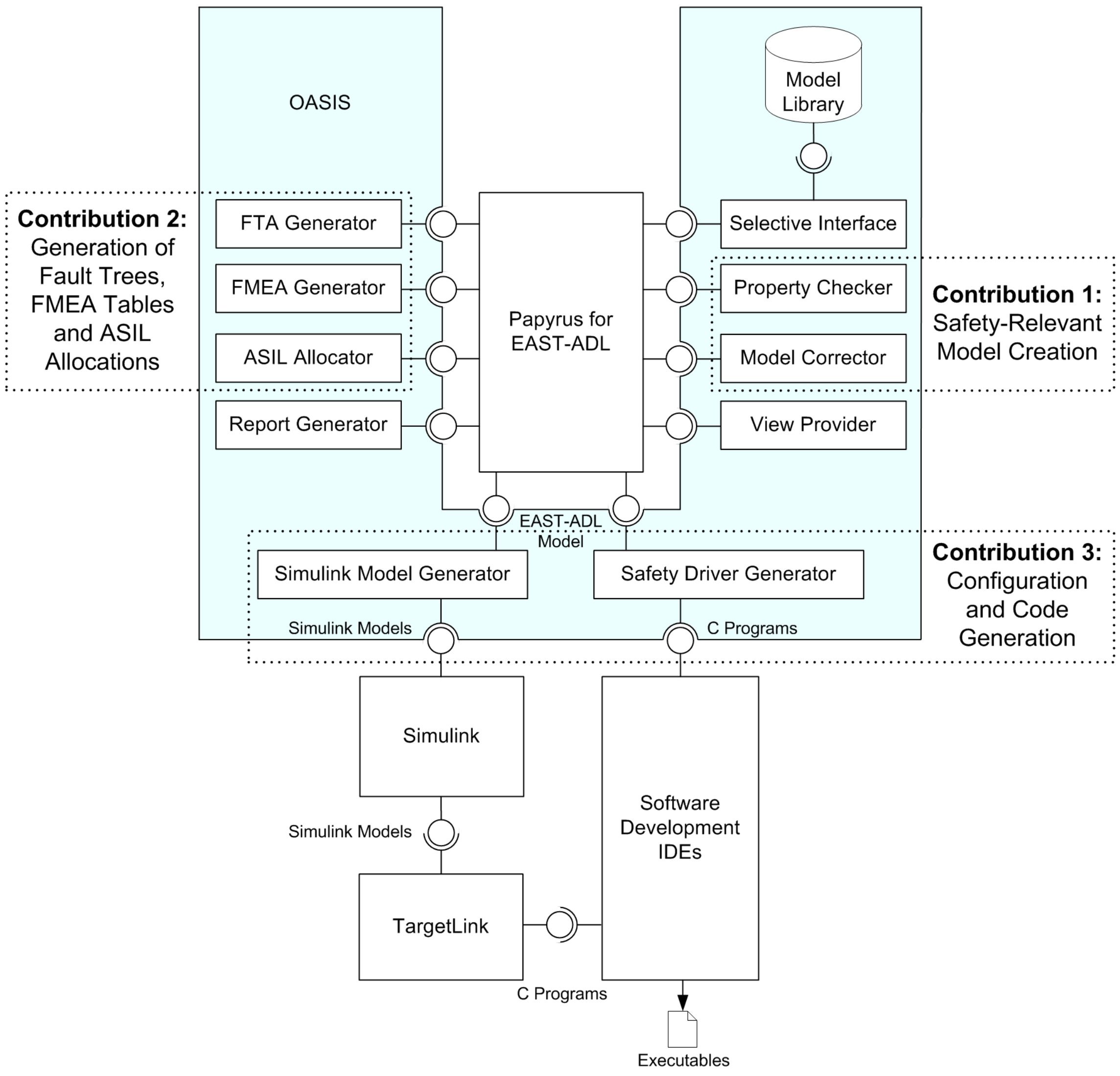


Figure 3.2: The tool OASIS is part of a tool chain required for the application of the safety engineering workflow. OASIS supports the application of the safety engineering workflow. Its support for (a) Safety-Relevant Model Creation, its support for (b) Generation of Fault Trees, FMEA Tables and ASIL Allocations and its support for (c) Configuration and Code Generation are the major contributions of this thesis.

and artifacts required by ISO 26262, (c) to describe relations and dependencies and (d) to structure the resulting information. Papyrus for UML is used to annotate a central EAST-ADL model in the course of the safety engineering workflow phases defined in the Sections 3.1.1, 3.1.2 and 3.1.3.1. This central EAST-ADL model reflects all the work products created in the course of the aforementioned workflow phases in an organized and structured manner.

3.2.2 OASIS

OASIS is a plugin for the tool Papyrus for UML and supports *Computer-Aided Model-Based Safety Engineering of Automotive Systems*. Its modules and its relations to other tools of the tool chain are illustrated in Figure 3.2. OASIS exploits the central EAST-ADL model (contains the work products in an organized and structured manner) and the EAST-ADL meta model (defines rules for the structure of the information) in order to support the application of the safety engineering workflow. The support is described thereafter.

- *Support for Model Creation and Reuse* is provided by the modules *Selective Interface*, *Property Checker* and *Model Corrector*. They support creation and maintenance of a complete and consistent EAST-ADL model and are especially useful during the workflow phases described in the Sections 3.1.1, 3.1.2 and 3.1.3.1. This is achieved by providing (a) systematic reuse of modeling elements, (b) automatic checking of the model and (c) automatic suggestion and application of corrective measures. OASIS' support for Model Creation and Reuse comprises support for *Safety-Relevant Model Creation*, which is a major contribution (see Section 2.5.1) of this thesis and is described in Section 3.3.

- *Support for Analysis and Documentation* is provided by OASIS' modules *View Provider*, *FTA Generator*, *FMEA Generator*, *ASIL Allocator* and *Report Generator*. Taking the EAST-ADL model as input, they support continuous analyses, communication and reviews and are especially useful during the workflow phases described in the Sections 3.1.1, 3.1.2 and 3.1.3.1. This is achieved by providing (a) automatic extraction of views, (b) automatic generation of fault trees, (c) automatic generation of FMEA tables, (d) automatic and optimal ASIL allocation and (e) automatic report generation. OASIS' support for Analysis and Documentation comprises support for *Generation of Fault Trees, FMEA Tables and ASIL Allocations*, which is a major contribution (see Section 2.5.2) of this thesis and is described in Section 3.4.

- *Support for Configuration and Code Generation* is provided by OASIS' modules *Safety Driver Generator* and *Simulink Model Generator*. They ease design and implementation of software in consistency with the EAST-ADL model and are especially useful during the workflow phases described in the Sections 3.1.3.2 and 3.1.3.5. This is achieved by providing (a) computer-aided configuration and automatic generation of safety drivers and (b) automatic generation of Simulink models. OASIS' support for *Configuration and Code Generation* is a major contribution (see Section 2.5.3) of this thesis and is described in Section 3.5.

3.2.3 Simulink

Simulink [56] is tightly integrated with Matlab and is a development environment for multidomain simulation and model-based design of dynamic systems. A graphical modeling language is available containing sinks, sources, linear blocks, nonlinear blocks and connectors. Add-ons for Simulink can be built that allow extending Simulink to other model domains by providing blocksets. Simulink is used for behavioral modeling in the course of the workflow phase defined in Section 3.1.3.3. The resulting models determine the behavior of the system components required to execute application software.

3.2.4 TargetLink

TargetLink [57] is an add-on to Simulink supporting a specialized blockset for code generation. TargetLink allows the generation of C-code from models created using this blockset. C is a general-purpose programming language frequently used for embedded software development. TargetLink allows designing control algorithms and generating production code for every microcontroller supported by a compiler and a linker. TargetLink is used in the workflow phase defined in Section 3.1.3.4 to generate C-code.

3.2.5 Software Development IDEs

Software development IDEs support developing C-programs, compilation, linking and debugging. Examples of embedded software development IDEs are Code Composer Studio [58], Code Warrior [59] or μVision [60]. Typically, IDEs are provided by the vendors of microcontrollers (e.g. Code Composer Studio or Code Warrior) or provided by third parties (e.g. μVision). Software development IDEs are used in the workflow phases described in the Sections 3.1.3.6 and 3.1.3.7.

3.3 Safety-Relevant Model Creation

The application of the Concept Phase (see Section 3.1.1), the System Level Development Phase (see Section 3.1.2) and the early Software Level Development Phase (see Section 3.1.3) of the safety engineering workflow requires (a) the capability to describe a system and related artifacts using the domain-specific language EAST-ADL, (b) the capability to assess the quality of the resulting model with respect to consistency and completeness and (c) the capability to consistently maintain the model depending on discussions, reviews, analyses and changes. These activities are typically performed by the safety engineer. OASIS provides Support for Property Checking (see Section 3.3.1) and Support for Model Correction (see Section 3.3.2) to aid these activities.

3.3.1 Support for Property Checking

Contemporary vehicles, their embedded systems, sensors and actuators are complex. This complexity results in a large set of information that needs to be managed during the application of the safety engineering workflow. The application of the safety engineering workflow and the complete and consistent projection of its results on the EAST-ADL model are challenging, cumbersome and error-prone.

OASIS' *Property Checker* allows automatically checking for properties that indicate the quality of the resulting model. This allows the continuous assessment of the model with respect to completeness and unambiguity. Furthermore, it sustains discussing and reviewing. The Property Checker is exhaustively described in Section V of Publication 1, Section 4 of Publication 2 and Section 5.2 of Publication 5, which can be found in Section 6 of this thesis.

3.3.2 Support for Model Correction

If violated, properties unveil an erroneous application of the safety engineering workflow, the EAST-ADL model needs to be corrected accordingly. To ease the correction of errors, it is proposed to support the identification and application of proper correction measures.

OASIS' Model Corrector can take corrective actions if the model is incomplete or ambiguous. Thus, the creation and the maintenance of a consistent and complete model is supported. The Model Corrector is exhaustively described in Section VI of Publication 1, Section 5.1 of Publication 2 and Section 5.3 of Publication 5, which can be found in Section 6 of this thesis.

3.4 Generation of Fault Trees, FMEA Tables and ASIL Allocations

The application of the System Level Development Phase (see Section 3.1.2) of the safety engineering workflow requires (a) continuous analyses as well as (b) communication of safety engineers, domain experts and other stakeholders based on the contents of the model. Languages like EAST-ADL are complex, and the evolving model is difficult to assess. Thus, it is important to support activities like analyzing, discussing and reviewing. These activities can be aided by OASIS' Support for Fault Tree Generation (see Section 3.4.1), Support for FMEA Table Generation (see Section 3.4.2) and Support for Automatic and Optimal Allocation of ASILs (see Section 3.4.3).

3.4.1 Support for Fault Tree Generation

The qualitative application of FTA is required to analyze and verify the system architecture. For the application of qualitative FTA, fault trees are required. A manual construction of fault trees that are consistent with the EAST-ADL model (describes PHA results and system architecture) is cumbersome and error-prone. Furthermore, extracting the minimum cut sets from the fault trees is challenging. Thus, it is proposed to automate the construction of fault trees and the extraction of minimum cut sets.

OASIS' *FTA Generator* allows the automated generation of fault trees from the model and the according extraction of minimum cut sets. This supports the application of FTA and the assessment of the system architecture's robustness against faults. Furthermore, it sustains discussing and reviewing. The FTA Generator is exhaustively described in Section 5.2 of Publication 2 and Section 6.2 of Publication 5, which can be found in Section 6 of this thesis.

3.4.2 Support for FMEA Table Generation

The safety engineering workflow foresees the qualitative application of FMEA to analyze and verify the system architecture. For the application of qualitative FMEA, an FMEA table is required. A manual construction of an FMEA table that is consistent with the EAST-ADL model (contains PHA results and a description of the system architecture) is cumbersome and error-prone. Thus, it is proposed to automate FMEA table construction.

OASIS' *FMEA Generator* allows the consistent generation of FMEA tables based on information contained in the model. This supports the application of FMEA. Moreover, a generated FMEA table is valuable for discussions and reviews. The FMEA Generator is exhaustively described in Section 5.2 of Publication 2 and Section 6.3 of Publication 5, which can be found in Section 6 of this thesis.

3.4.3 Support for Automatic and Optimal Allocation of ASILs

The safety engineering workflow requires the allocation of ASILs to the components of the system architecture. The allocated ASILs determine applicable requirements of ISO 26262 and the necessary safety measures to avoid unreasonable residual risk. The requirements of ISO 26262 affect the rigor of the development process of the system components (e.g. the portfolio of applicable verification techniques). The necessary safety measures determine the runtime fault detection capabilities of the system components. Thus, the allocated ASILs strongly affect system safety. Furthermore, the allocated ASILs influence development costs and costs per piece.

An allocation of ASILs to the components of the system architecture shall, therefore, (1) assure that the required level of functional safety is achieved and (2) permit an economic solution with respect to development costs and cost per piece. Manual elaboration of an ASIL allocation that fulfills these requirements is complex, cumbersome and should be tool-supported.

OASIS' *ASIL Allocator* allows inspecting the constraints on the allocation of ASILs to the components of the system architecture based on PHA and FTA results. Furthermore, it allows the automatic allocation of ASILs to the components of the system architecture. This makes a manual elaboration superfluous. The ASIL Allocator is exhaustively described in Section 5 of Publication 3 and Section 6.4 of Publication 5, which can be found in Section 6 of this thesis.

3.5 Configuration and Code Generation

The application of the later Software Level Development Phase (see Section 3.1.3) of the safety engineering workflow requires the design and the implementation of software for the components of the system architecture. These are challenging tasks, and it is difficult to design and implement software that is consistent with the EAST-ADL model. These activities can be aided by OASIS' Support for Generation of Simulink Models (see Section 3.5.1) and Support for Configuration and Generation of Safety Drivers (see Section 3.5.2).

3.5.1 Support for Generation of Simulink Models

OASIS' *Simulink Model Generator* allows the automated generation of Simulink models from the EAST-ADL model. This ensures consistency between EAST-ADL model and Simulink models and is the basis for the subsequent description of behaviors (implementation of the functions). The Simulink Model Generator is exhaustively described in Section IV of Publication 4 and Section 7.1 of Publication 5, which can be found in Section 6 of this thesis.

3.5.2 Support for Configuration and Generation of Safety Drivers

OASIS' *Safety Driver Generator* allows the computer-aided configuration and generation of safety drivers for safe initialization, runtime-testing and error-handling of microcontrollers. This module extracts information from the EAST-ADL model, guides the safety engineer, updates the model and generates program code. The Safety Driver Generator including its support for the industrial lock-step multi-core microcontroller TMS570LS20216 [58] is exhaustively described in Section V of Publication 4 and Section 7.2 of Publication 5, which can be found in Section 6 of this thesis.

Chapter 4

Experimental Evaluation

4.1 Case Study of HEV Development

The tool OASIS including its support for (a) Safety-Relevant Model Creation, (b) Generation of Fault Trees, FMEA Tables and ASIL Allocations as well as (c) Configuration and Code Generation was designed and implemented as a plugin for the tool Papyrus as described in the Sections 3.3, 3.4 and 3.5. A tool chain was set up as described in Section 3.2 (Code Composer Studio was chosen as software development IDE).

The presented approach was experimentally evaluated using the case study of HEV [4] development. One of the main characteristics of HEVs is the addition of an electric machine supporting the classic combustion engine providing supplementary or substitutive torque. If such a vehicle uses its electric machine as an electric motor to support the combustion engine, it discharges the battery. If the electric machine is used as a generator to regain energy while the vehicle decelerates (recuperation), it recharges the battery and/or supplies the auxiliaries with electrical energy.

The application of the safety engineering workflow for the case study of HEV development is described in Section VII of Publication 1, Section 6 of Publication 2, Section 6 of Publication 3, Section VI of Publication 4 and Section 8.1 of Publication 5. Furthermore, it is described how OASIS' modules that realize (a) Safety-Relevant Model Creation, (b) Generation of Fault Trees, FMEA Tables and ASIL Allocations as well as (c) Configuration and Code Generation supported the workflow application and the creation of an HCU demonstrator based on the multi-core microcontroller TMS570LS20216 [58].

4.2 Complexity of Resulting Model and Derived Entities

While the safety engineering workflow was applied for the case study of HEV development, an EAST-ADL model was created and different entities were derived from the model. This section presents metrics together with values characterizing the complexity of the resulting model and the derived entities. These metrics and values fall into different categories and are presented in Table 4.1.

Category (1) describes the complexity of the resulting EAST-ADL model. Whereas the workflow was only applied for a part of the HEV powertrain consisting of 3 sensors, 3 actuators and 4 control units, the resulting EAST-ADL contained 957 modeling elements.

The large number of modeling elements is not ascribable to EAST-ADL that allowed appropriately structuring the information and to efficiently express the work products. On the contrary, the large number of modeling elements is ascribable to the complexity of an HEV powertrain. This complexity is existent and challenging, regardless whether an EAST-ADL model is used or not.

The rigor of the information structure of the EAST-ADL model was exploited by OASIS to support the creation of consistent and complete work products and to simplify and automate workflow steps. The categories (2), (3) and (4) describe the complexity of the entities automatically derived to support the workflow application thanks to OASIS' support for (a) Safety-Relevant Model Creation, (b) Generation of Fault Trees, FMEA Tables and ASIL Allocations as well as (c) Configuration and Code Generation (the major contributions of this thesis). The presented numbers underpin the value of OASIS. For example, a manual identification of 152 improper modeling elements or a manual elaboration of fault trees with 245 nodes would have been time-consuming and error-prone tasks. Thus, (a), (b) and (c) lead to decreased expenditure of time and improved product quality due to automation and can potentially help the industry to create safe products at affordable prices. Additional metrics that illustrate OASIS' value with respect to other capabilities can be found in Section 8.2 of Publication 5.

(1) EAST-ADL Model Metrics	Value
# of Modeled Sensors	3
# of Modeled Actuators	3
# of Modeled Control Units	4
# of EAST-ADL Modeling Elements	957
(2) Metrics for Safety-Relevant Model Creation	Value
# of Property Violations	160
# of Violating Modeling Elements	152
(3) Metrics for Generation of Fault Trees, FMEA Tables and ASIL Allocations	Value
# of Generated Fault Trees	6
# of Generated Fault Tree Nodes	245
# of Extracted Minimum Cut Sets	45
# of Generated FMEA Table Lines	192
# of Derived Constraints for ASIL Allocation	43
# of Automatically Allocated ASILs	10
(4) Metrics for Configuration and Code Generation	Value
# of Generated Simulink Model Ports	5
# of Requirements from Safety Driver Configuration	44
# of Lines of Code of Generated Safety Driver	4224

Table 4.1: This table presents metrics and values characterizing the complexity of the HEV use case and automatically derived entities with respect to the major contributions of this thesis.

4.3 Assessment of Benefits

Based on the case study of HEV development, this section systematically evaluates the major contributions of this thesis with respect to benefits.

First, evaluation criteria were defined to allow a systematic evaluation. The criteria *Model Creation* and *Model Maintenance* concern the ability to efficiently elaborate an EAST-ADL model by creating and manipulating modeling elements. The criteria *Model Consistency* and *Model Completeness* are related to the capability of assessing the model with respect to particular characteristics indicating quality. The criteria *Model Examination*, *Application of Analyses*, *Application of Reviews* and *Communication* concern the ability of validating the information contained in the model. Finally, the criterion *Software Development* describes the capability of bridging the gap between system and software development activities. The evaluation criteria are listed in the columns of Table 4.2.

Second, the achieved benefits of the modules Property Checker, Model Corrector (realize support for Safety-Relevant Model Creation), FTA Generator, FMEA Generator, ASIL Allocator (realize support for Generation of Fault Trees, FMEA Tables and ASIL Allocations), Simulink Model Generator and Safety Driver Generator (realize support for Configuration and Code Generation) were assessed. These modules realize the major contributions of this thesis and are listed in the rows of Table 4.2. The achieved benefits were classified as *high (3)*, *medium (2)*, *low (1)* or *not applicable (-)*. Each cell of Table 4.2 contains the classification of a module with respect to an evaluation criterion.

	Model Creation	Model Maintenance	Model Consistency	Model Completeness	Model Examination	Application of Analyses	Application of Reviews	Communication	Software Development	
(a) Safety-Relevant Model Creation										
Property Checker	3	3	3	2	2	2	-	-	1	16
Model Corrector	3	3	3	1	2	2	-	-	1	15
(b) Generation of Fault Trees, FMEA Tables and ASIL Allocations										
FTA Generator	-	-	2	1	3	3	2	2	-	13
FMEA Generator	-	-	2	1	3	3	2	2	-	13
ASIL Allocator	1	1	2	1	3	3	2	2	-	15
(c) Configuration and Code Generation										
Simulink Model Generator	-	-	3	1	-	-	-	-	3	7
Safety Driver Generator	1	1	3	1	-	-	2	-	3	11
	8	8	18	8	10	10	8	6	8	

Table 4.2: Depending on evaluation criteria, the achieved benefits of OASIS' modules realizing the major contributions of this thesis were assessed and classified as *high (3)*, *medium (2)*, *low (1)* or *not applicable (-)* based on experience from the HEV use case.

The best results were achieved with respect to the evaluation criteria (1) *Model Consistency*, (2) *Application and Analyses* and (3) *Model Examination.* Considering the complexity of an HEV powertrain, this is justified by the fact that (1) the elaboration of a consistent model, (2) the manual creation of means supporting analyses and (3) the examination of particular details are time-consuming and error-prone tasks.

According to the results, the most valuable OASIS modules realizing the major contributions of this thesis were (a) the Property Checker, (b) the Model Corrector and (c) the ASIL Allocator. Considering the complexity of an HEV powertrain, this is justified by the fact that (a) and (b) provide support for a lot of phases of the presented workflow and (c) replaces the manual application of a highly safety-relevant development step with an automatic and optimal application of the same working step.

An enhanced assessment of benefits comprising an additional evaluation criterion as well as other modules of OASIS can be found in Section 8.3 of Publication 5.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The automotive industry experiences a trend towards powertrain electrification to reduce emissions, improve drivability and improve fuel economy. This trend leads to new powertrain components such as electric machines or high voltage batteries. An attempt to combine the advantages of traditional ICE cars and EVs are HEVs.

The sensors and actuators of an HEV are controlled by an embedded system consisting of dozens of control units that are connected via automotive bus systems. Faults and failures of this embedded system can lead to severe hazards such as fire or explosion of the high voltage battery or unintended vehicle movement. Thus, the embedded system of an HEV is safety-critical. Furthermore, the additional components of an HEV pursue the complexity increase of automotive embedded systems.

To appropriately develop safety-critical embedded systems for HEVs, the functional safety standard ISO 26262 for the automotive domain needs to be considered. This standard requires a rigorous development process to avoid the introduction of systematic faults during development. Furthermore, the standard requires that safety-critical embedded systems are capable of detecting random hardware faults during runtime to be able to achieve and maintain a safe state, if necessary.

Adequate measures to cope with the increasing complexity of safety-critical embedded systems are the definition of structured workflows to allow systematic and comprehensible development as well as the provision of adequate tool support in order to automate tedious and error-prone work steps. Both measures are considered to improve product quality and reduce the required expenditure of time.

Among the typically tool-supported activities for safety-critical embedded systems development are (a) Early Hazard Analysis, (b) FTA, FMEA and Allocation of Safety Parameters based on System Models as well as (c) Generation of Source Code and Models. Based on reviewed literature, potentials for improvements with respect to (a), (b) and (c) were identified. This thesis unveils these potentials and provides the following main contributions:

- Support for Safety-Relevant Model Creation
- Support for Generation of Fault Trees, FMEA Tables and ASIL Allocations

- Support for Configuration and Code Generation

The tool OASIS supports Computer-Aided Model-Based Safety Engineering of Automotive Systems and constitutes an implementation of these contributions. OASIS is part of a tool chain supporting an automotive safety engineering workflow. Furthermore, OASIS has features (Document Generation, View Provision, Reuse of Modeling Elements) that do not belong to the main contributions of this thesis but are of utmost importance for the industrial application of the approach.

The case study of HEV development was used to experimentally evaluate OASIS with respect to (1) Support for Safety-Relevant Model Creation, (2) Support for Generation of Fault Trees, FMEA Tables and ASIL Allocations as well as (3) Support for Configuration and Code Generation. Although the presented approach did not replace the intellectual process of safety engineering, it turned out that (1), (2) and (3) lead to decreased expenditure of time and improved product quality due the simplification and automation of workflow steps. This can potentially help the industry to create safe products at affordable prices.

5.2 Future Work

This thesis contributes to the state of the art by bringing improvements with respect to (a) Early Hazard Analysis, (b) FTA, FMEA and Allocation of Safety Parameters based on System Models as well as (c) Generation of Source Code and Models. This is achieved by providing (1) Support for Safety-Relevant Model Creation, (2) Support for Generation of Fault Trees, FMEA Tables and ASIL Allocations as well as (3) Support for Configuration and Code Generation. Nevertheless, there is space for future work and further improvements. Future work with respect to the major contributions is described thereafter.

5.2.1 Safety-Relevant Model Creation

OASIS provides Support for Safety-Relevant Model Creation. First, OASIS allows automatically checking an evolving EAST-ADL model for properties indicating consistency and completeness of the model. A potential improvement with respect to property checking is a multi-level mechanism checking for two different kinds of properties. The violation of the first category of properties could result in warnings indicating potential or easy imperfections. The violation of the second category of properties could result in error messages indicating severe imperfections.

Second, OASIS also supports the automatic proposition and application of corrective measures. Presently, the suggestion of corrective measures solely depends on the meta model of the model that is subject to correction. This could be further improved by incorporating ontologies that describe domain-specific knowledge. The suggested corrections could depend on the meta model and on the ontologies resulting in more precise suggestions of corrective measures.

5.2.2 Generation of Fault Trees, FMEA Tables and ASIL Allocations

OASIS provides Support for Generation of Fault Trees, FMEA Tables and ASIL Allocations. First, it allows generating fault tress and an FMEA table from an EAST-ADL model. This supports the verification of the system architecture and the definition of proper functional and technical safety concepts. The present modeling approach could be improved by foreseeing the creation of traces from the faults and failures of the error model to the functional and technical safety requirements. This mapping could define which requirements aim at detecting or handling of which faults or failures. Furthermore, the generated fault trees could contain information about these safety requirements. Moreover, the generated FMEA tables could contain an additional column listing safety requirements that aim at detection or handling causative faults.

Second, OASIS supports automatic and optimal allocation of ASILs. This approach to ASIL allocation takes minimum cut sets extracted from qualitative fault trees as input. This method could be enhanced to consider also the probability of occurrence of faults and failures in order to achieve a more precise ASIL allocation. However, this would require the use of quantitative fault trees as input.

5.2.3 Configuration and Code Generation

OASIS provides Support for Configuration and Code Generation. First, it allows automatically generating Simulink models from a more abstract EAST-ADL model in order to support application software development. EAST-ADL is an ADL and has, thus, limited capabilities for behavioral modeling. Consequently, the generation of Simulink models is presently limited to structural elements such as containers and ports. This could be further improved by supplementing the EAST-ADL meta model with behavioral language constructs for modeling of activities or state-based behavior. Models based on such an enhanced EAST-ADL meta model could be used to generate Simulink models containing also behavioral language constructs like state machines annotated in Simulink/Stateflow.

Second, the configuration and generation of safety drivers is supported by OASIS to sustain basic software development. This could be enhanced in order to configure and generate also other parts of the basic software such as communication drivers and RTOS to provide a holistic approach to code generation.

Chapter 6

Publications

This chapter contains five publications by the author of this thesis presenting the approach described in Chapter 3 and the results illustrated in Chapter 4 in greater detail. Whereas the Publications 1, 2, 3 and 4 are focused on particular contributions of this thesis, Publication 5 is an invited paper under review for possible post-conference publication on a special issue of the Journal of Reliability Engineering & System Safety and provides an integrated and holistic view on the contributions of this thesis and additional functionalities demanded by industry. In the following, the five publications are listed.

Publication 1: *A Computer-Aided Approach to Preliminary Hazard Analysis for Automotive Embedded Systems*, 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems (ECBS), Las Vegas, USA, April 27–29, 2011

Publication 2: *A Computer-Aided Approach to PHA, FTA and FMEA for Automotive Embedded Systems*, 30th International Conference on Computer Safety, Reliability and Security (SafeComp), Naples, Italy, September 19–21, 2011

Publication 3: *Automatic and Optimal Allocation of Safety Integrity Levels*, 58th Reliability and Maintainability Symposium (RAMS), Reno, USA, January 28–30, 2012

Publication 4: *A Bridge from System Development to Software Development for Automotive Embedded Systems*, 38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Çeşme, Turkey, September 5–8, 2012

Publication 5: *OASIS: An Automotive Analysis and Safety Engineering Instrument*, Post-Conference Publication on a Special Issue of the Journal of Reliability Engineering & System Safety (Invited Paper under Review), June 10, 2012

Figure 6.1 illustrates the mapping of the five publications onto the tool chain (see Section 3.2) including OASIS (see Section 3.2.2) and the automotive safety engineering workflow (see Section 3.1). Whereas OASIS supports *Safety-Relevant Model Creation*, *Generation of Fault Trees, FMEA Tables and ASIL Allocations* and *Configuration and Code Generation*, the automotive safety engineering workflow comprises a *Concept Phase*, a *System Level Development Phase* and a *Software Level Development Phase*.

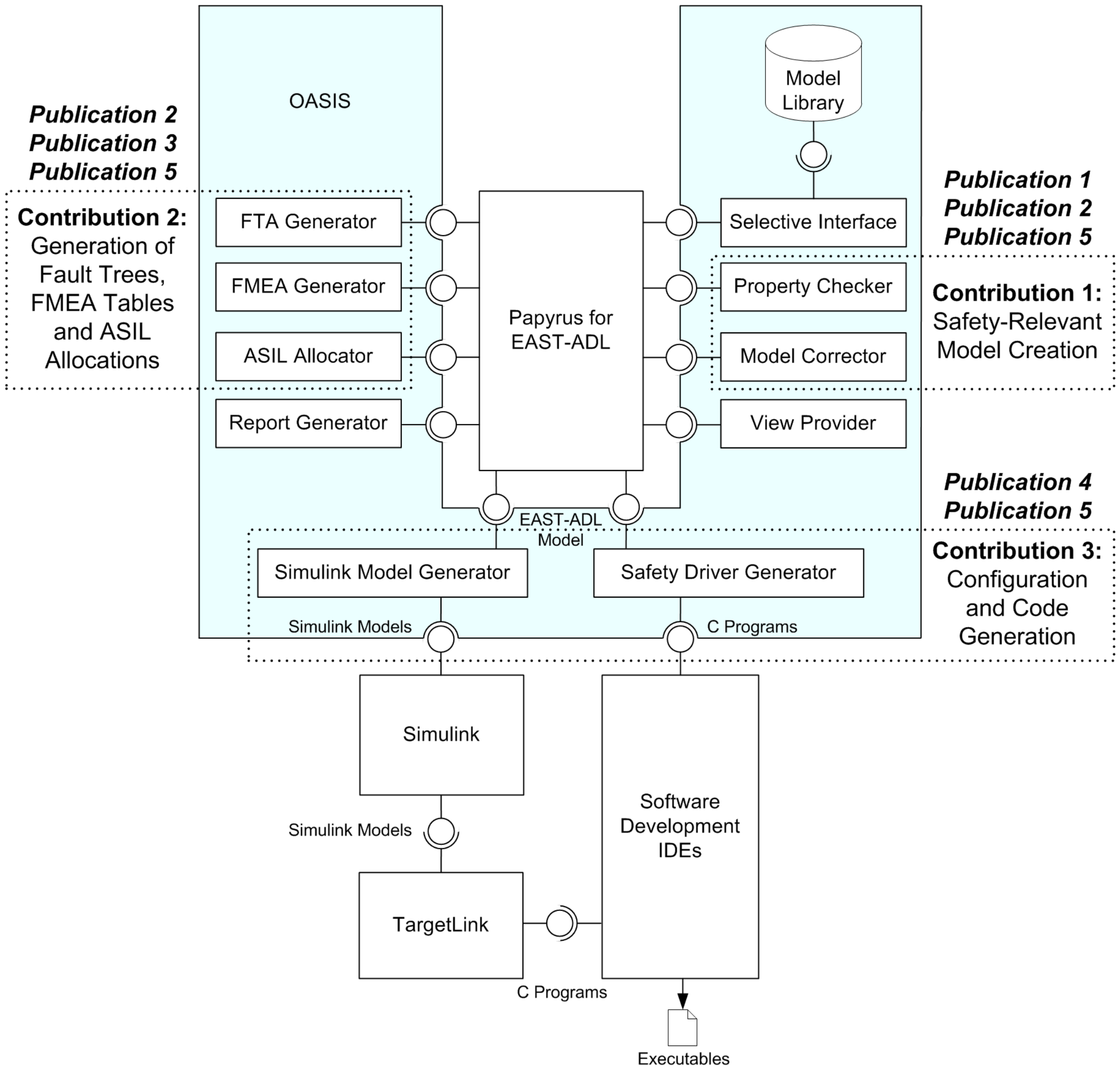


Figure 6.1: The mapping of the five publications onto the tool chain including OASIS that provides support for (a) Safety-Relevant Model Creation, (b) Generation of Fault Trees, FMEA Tables and ASIL Allocations as well as (c) Configuration and Code Generation.

Publication 1 is focused on the early subphases of the *Concept Phase* of the automotive safety engineering workflow and describes how OASIS' support for *Safety-Relevant Model Creation* (see Section 3.3) eases the application of these subphases. More precisely, Publication 1 describes how PHA can be carried out using the diagrammatic language EAST-ADL, while being supported by automatic checking of the evolving model and the automatic suggestion and application of corrective measures.

Publication 2 is about the *Concept Phase* and the *System Level Development Phase.* OASIS' *Support for Safety-Relevant Model Creation* (see Section 3.3), OASIS' *Support for Fault Tree Generation* (see Section 3.4.1) and OASIS' *Support for FMEA Table Generation* (see Section 3.4.2) are described. More precisely, based on Publication 1, Publication 2 shows, how the diagrammatic language EAST-ADL can be used to derive functional and technical safety requirements, to define an automotive system architecture and to define an error model. Furthermore it is described how the automatic checking of the evolving model and the automatic suggestion and application of corrective measures can support the Concept Phase and the System Level Development Phase. Finally, Publication 2 shows how fault trees can automatically be generated from the EAST-ADL model, how minimum cut sets can be automatically extracted from the fault trees and how an FMEA table can be generated from the same EAST-ADL model to support qualitative FTA and FMEA, while being consistent with PHA results.

Publication 3 is focused on the final subphase of the *System Level Development Phase* and describes how OASIS can ease the application of this subphase using its *Support for Automatic and Optimal Allocation of ASILs* (see Section 3.4.3). More precisely, based on Publication 3, it describes how the problem of finding an optimal (with respect to an objective function) allocation of ASILs to the components of an automotive system architecture can be mapped onto an optimization problem that is defined in terms of constraints and can be solved by constraint solvers. The approach foresees the automatic derivation of constraints from qualitative FTA results and preferences of the safety engineer ensuring the consistency of the ASIL allocation to PHA and FTA whereas meeting the preferences. The solution can be automatically projected on the EAST-ADL model.

Publication 4 is about the effective transition from the *System Level Development Phase* to the *Software Level Development Phase* due to OASIS's support for *Configuration and Code Generation* (see Section 3.5). This support allows generating Simulink models from the EAST-ADL model. Furthermore, the configuration and generation of safety drivers for microcontrollers from the same EAST-ADL model is supported. These capabilities ease the development of application software and safety-relevant basic software. The approach to configure and generate safety drivers is illustrated using the example of an industrial multi-core microcontroller for safety-critical applications.

Publication 5 provides a holistic and integrated view on the contributions described in the Publications 1, 2, 3 and 4. It presents some of OASIS' additional functionalities that were not subject of the other four publications, but are required by industry. These additional functionalities are not considered to be major contributions of this thesis and include support for the selective reuse of modeling elements, document generation and the provision of views on the work products of the automotive safety engineering workflow. In addition, Publication 5 presents the experimental evaluation of the presented approach using the case study of HEV development. Finally, evaluation criteria and metrics are presented indicating the complexity of the case study and OASIS' value.

2011 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems

A Computer-Aided Approach to Preliminary Hazard Analysis for Automotive Embedded Systems

Roland Mader[1,2], Gerhard Grießnig[1,2], Andrea Leitner[2], Christian Kreiner[2], Quentin Bourrouilh[1], Eric Armengaud[3], Christian Steger[2], Reinhold Weiß[2]
[1]*AVL List GmbH, Austria*
[2]*Institute for Technical Informatics, Graz University of Technology, Austria*
[3]*Virtual Vehicle Competence Center, Austria*

***Abstract*—Powertrain electrification of automobiles leads to a higher number of sensors, actuators and control functions, which in turn increases the complexity of automotive embedded systems. The safety-criticality of the system requires the application of Preliminary Hazard Analysis early in the development process. This is a necessary first step for the development of an automotive embedded system that is acceptably safe. Goal of this activity is the identification and classification of hazards and the definition of top level safety requirements that are the basis for designing a safety-critical embedded system that is able to control or mitigate the identified hazards. A computer-aided framework to support Preliminary Hazard Analysis for automotive embedded systems is presented in this work. The contribution consists of (1) an enhancement for Preliminary Hazard Analysis to the domain-specific language EAST-ADL, as well as (2) the identification of properties that indicate the correct application of Preliminary Hazard Analysis using the language. These properties and an analysis model reflecting the results of the Preliminary Hazard Analysis are used for the automated detection of an erroneously applied Preliminary Hazard Analysis (property checker) and the automated suggestion and application of corrective measures (model corrector). The applicability of the approach is evaluated by the case study of hybrid electric vehicle development.**

***Keywords*-functional safety; ISO 26262; preliminary hazard analysis; safety goal; automotive embedded system**

I. INTRODUCTION

Nowadays cars contain embedded systems that incorporate up to 70 microcontrollers. These microcontrollers communicate via bus systems, gather sensor data or command actuators of the vehicle. At the same time the shift of the automotive industry towards powertrain electrification introduces new automotive sensors, actuators and functions. Electronic Control Units (ECUs) are responsible for managing the components (e.g. battery, motor) that can be found in HEVs (Hybrid Electric Vehicles) and components (e.g. engine, transmission) that can also be found in traditional vehicles. Therefore the safe operation of the vehicle depends on the correct operation of the embedded system.

Safety-critical automotive embedded systems are developed according to rigorous development processes. A necessary first step in such a development process is the application of *Preliminary Hazard Analysis* [1] (PHA) by a team of people with a wide variety of knowledge and skills. This analysis technique is applied earliest in the development process before neither concrete design solutions are elaborated nor enough numerical values are defined to allow the application of other techniques such as simulation or quantitative analyses. The purpose of PHA is the identification, classification and assessment of potential *hazards*[1] of a newly developed vehicle that are caused by potential failures of its embedded system. The early knowledge about the existence of hazards allows the definition of *safety goals* [2] (top-level safety requirements to the embedded system), even if detailed and quantitative information about the vehicle under development is insufficiently available. Considering the safety goals, a safety-critical embedded system design (typically including runtime fault detection capabilities) is developed that is able to best achieve the defined safety goals and to control or mitigate the identified hazards. Although the application of PHA alone cannot ensure safety of a vehicle, it is a necessary first step in order to eliminate or control hazards through adequate design, implementation, integration, verification and validation. Moreover the results of PHA serve as a baseline for later analyses. The early application of PHA is required by the safety standard ISO 26262 [3] for automotive E/E system development. This safety standard refers to this activity as *Hazard Analysis and Risk Assessment*.

Parallel to that, the field of model-based development (MBD) is rapidly evolving in order to manage product complexity, process complexity and organization complexity. Major application fields covered are communicating ideas and design, documenting and managing design information, automated model analysis and automated synthesis [4]. Potential benefits of these techniques are reduced time-to-market, reduction of costs and improved quality. Consequently, different approaches to early hazard analysis incorporate the use of *diagrammatic languages* [5] such as UML (Unified Modeling Language) in order to facilitate clear communication and documentation. However, existing approaches do not fully exploit the potential of diagrammatic languages for automated model analysis and automated syn-

[1]A hazard is defined as, "A state or set of conditions of a system (or an object) that, together with other conditions in the environment of the system (or object), will lead inevitably to an accident (loss event). [1]"

978-0-7695-4379-6/11 $26.00 © 2011 IEEE
DOI 10.1109/ECBS.2011.43

thesis. This hinders the early identification of imperfections that potentially affect the successful mitigation and control of hazards. Manual reviews and modifications of the analysis results are usually complicated and subject to mistake, and should be computer-aided.

The contribution of this work is a computer-aided approach to PHA in the development process of an automotive embedded system that is based on the domain-specific language EAST-ADL (Electronics Architecture and Software Technology-Architecture Description Language) [6]. Based on a detailed description of the work flow for PHA and defining safety goals, we propose (1) a language enhancement to EAST-ADL that allows to describe malfunctions and *operational situations* [2] in a more systematic way. Further, (2) we identify properties that indicate the correct application of PHA in the development process of a safety-critical automotive embedded system. We propose a tool implementation that can check these properties based on an analysis model reflecting the results of the PHA. This allows to automatically discover omissions that potentially affect safety of the vehicle under development. In the case a property is violated, the safety engineer automatically receives suggested solutions. Subsequently the most proper proposed solution can be accepted or it can be decided to solve the problem manually. If a suggested solution is accepted, the analysis model is automatically corrected.

This work is organized as follows: In Section II, related work is discussed. Section III describes the computer-aided approach to PHA including the work flow. In Section IV, the analysis model that is created in the course of PHA is described. In Section V and Section VI, our approaches to computer-aided checking and correction of the analysis model are described. Section VII outlines the experimental application of the approach using hybrid drive vehicle development. Finally Section VIII concludes this work.

II. Related Work

In the following, approaches to hazard analysis that are intended to be applied early in the development process are reviewed. The contributions [7], [8], [9], [10], [11], [12], [13] focus on defining systematic approaches that support the intellectual process of identifying and classifying hazards and defining means to mitigate or control them. All of them consider models to be a valuable aid for the application of hazard analysis techniques. None of these approaches refers to the use of diagrammatic languages to create models. In contrast we use a domain-specific, diagrammatic language for modeling to support coping with complexity, communicating ideas and facilitating automated model analysis and automated synthesis.

In [7] a technique called Actuator Based Hazard Analysis is proposed that can be carried out early in the development process, when only little information concerning the system implementation is available. The approach is based on the assumption that only the actuators of the system can affect their environment. The method defines three fault classes (commission, omission and stuck). Each system effect that describes an undesired enactment of an actuator is defined by a fault class, an analyzed actuator and an user intent. The method defines four severity classes (Catastrophic, Critical, Marginal and Negligible). All severity classes are applied to each actuator and the distribution between the severity classes is determined. Based on distribution and weighting, a criticality level can be determined that is the major input to the solvability analysis and design selection that allows to choose the design concept that is most likely to handle the identified hazards.

Another approach to preliminary hazard analysis for automotive systems that is similar to the one required by ISO 26262 [3] is described in [9]. The approach starts with hazard identification based on a system model. A PASSPORT diagram with supplementary descriptions is used as a system model. A further step of the approach is hazard classification according to severity, controllability and exposure.

In [10] an ISO 26262-compatible approach to preliminary hazard analysis is presented. The approach incorporates an architectural model and starts with (1) scope definition. In this phase, safety-critical functions of a vehicle are illustrated in a block diagram including control units, gateways, sensors, actuators and communication systems. The next step is (2) the definition of a role model. A control unit can contribute to multiple functions. In the context of different functions, the control unit may have different roles (e.g. actuation, calculation, monitoring). The next step is (3) the creation of a tabular architectural model. This starts with the mapping of functions to architectural elements. Subsequently, severity, exposure and controllability are evaluated and a resulting ASIL (Automotive Safety Integrity Level) is determined for each function. Then, roles (depending on functions) are assigned to each architectural element. Finally each architectural element has roles with corresponding automotive safety integrity levels.

The work proposed in [11] describes an approach to hazard analysis of safety-critical software-intensive systems called STPA (Systems Theoretic Process Analysis) for early application in the development process. The approach starts with the identification of hazards and related requirements or constraints. Subsequently inadequate control actions, control flaws, and inadequate control executions that lead to inadequate control actions are identified. This is input to a design process that aims on creating new constraints, refining existing constraints, creating a new design or modifying the existing design until all hazards are eliminated, mitigated or controlled. This process is iterative. The approach relies on a model that describes the control flow of the system under analysis and causes of accidents. The applicability of the approach is illustrated using a spaceflight application.

In [12] an approach to hazard analysis of SoS (System of Systems) is described that is intended to be applied early in the development process. This hazard analysis technique is focused on the interfaces between the particular systems. The approach is based on a model of the SoS as well as guidewords. In the course of the hazard analysis they carry out Input/Output Analysis as well as Network Analysis. The probability of the occurrence of accidents is assessed. A validation framework is established that incorporates the definition of goals. Based on these goals, metrics (e.g. percentage software safety requirements traceable to hazards) are defined that indicate the quality of the conducted hazard analysis. Some of the defined metrics depend on knowledge gained from previous analyses.

A new methodology to safety-critical system development is proposed in [13]. Amongst other activities, this methodology requires to identify those functions that are safety-critical. Thereafter hazards are identified, risks are assessed and risk mitigation means are defined and associated early in the development process. The work proposes metrics based on the identified hazards. An example is the metric *percentage software hazards* that is defined as number of software safety hazards divided by the number of system safety hazards. The approach is evaluated using a railway application.

In contrast to aforementioned related works, [14], [15], [16] explicitly refer to the use of diagrammatic languages to create models. In contrast to our approach, none of these approaches uses the annotated model to check for properties that indicate the correct application of the analysis technique.

An approach that combines hazard analysis and the use of a diagrammatic language to create models is described in [14]. The authors consider the use of UML models to be appropriate in order to handle the increasing complexity of safety-critical software systems. They use a subset of UML (component and deployment diagrams) to support hazard analysis at an early design stage. Boolean logic is used to formally model hazards and failure propagation. Starting with a component model of the system to be analyzed, (1) fault trees for all system hazards are derived. Subsequently (2) the propagation of component failures is analyzed for each component. Then (3) related behavior of deployment nodes and hardware devices has to be derived. Finally (4) boolean equations can be used to apply analysis techniques. The approach allows to identify the most serious hazards and failures and to determine components that require a more detailed safety analysis and assumed restrictions to fault propagation. This facilitates the systematic derivation of safety requirements.

The work described in [15] aims on improving the problems posed by the derivation of safety requirements and by conducting hazard analysis. The first step is the identification and description of functions associated with the level under study. Use cases and scenarios are used for function description. The second step is the failure identification. In this step a technique is applied that is inspired by techniques such as FHA (Functional Hazard Assessment) that is typically applied early in the development process and makes use of guidewords. In the third step, based on the analysis new safety related functional requirements are identified. The approach was evaluated using an use case from the avionics domain.

An approach to preliminary hazard analysis using EAST-ADL is proposed in [16]. The proposed workflow starts with the description of the functions (e.g. Cruise Control) of the vehicle, their operation needs and other stakeholder requirements. Thereafter a feature tree model is used to structure the vehicle functions. After the allocation of requirements to the features, the vehicle is well determined in terms of its requirements, functions and modes. This is the input to the identification and classification of hazards based on the functions and their related requirements. Thereafter safety goals are derived that constitute top level safety requirements.

In contrast to aforementioned related works, approaches that allow to conduct hazard analysis early in the development process in the context of more sophisticated MBD tool support are defined in [17], [18], [19]. These approaches provide a framework to support modeling using a domain-specific, diagrammatic language as well as the definition and automated checking of properties, but none of them defines properties that support the conduction of hazard analysis. In contrast our approach supports computer-aided checking, based on well defined properties that are presented in this work and allows the automated correction of the analysis model.

The work proposed in [16] in combination with the tools [17] and [18] allows the definition of properties using OCL (Object Constraint Language). The combined approach allows the definition and checking of properties on demand.

Tools that aim to support the safety standard ISO 26262 are reviewed in [19]. Among the reviewed tools is a tool named Medini Analyze following an MBD approach. It supports the definition of vehicle functions and the application of hazard analysis early in the development process. The tool allows to define constraints using the OCL language that can be automatically validated on demand. Besides a predefined set of checking rules, users can define their own rules.

III. Computer-Aided Preliminary Hazard Analysis

The application of PHA early in the development process of an automotive embedded system when less detailed and quantitative information about the vehicle under development is available is a cornerstone in the development process of an embedded system that is acceptably safe. The identified and classified hazards and the derived safety goals determine

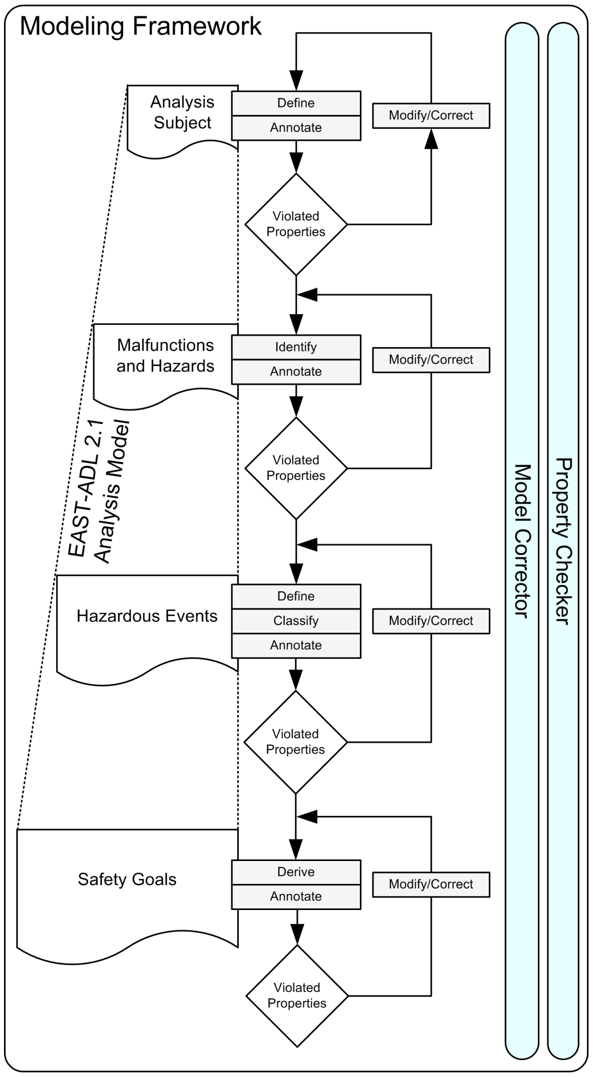


Figure 1. Computer-Aided Approach to Preliminary Hazard Analysis.

the design of the safety-critical embedded system including its fault detection capabilities.

The proposed methodology for PHA is based on the work described in [16] and is depicted in Figure 1. In this approach an analysis model is annotated, systematically enhanced and refined using the domain-specific language EAST-ADL until the results of the PHA are satisfactory. This analysis model describes the vehicle under development, the results of the PHA as well as the derived safety goals. We use EAST-ADL including an enhancement (see also Section IV-A and Section IV-B). The proposed methodology is explained in the following:

1) **Definition of the Analysis Subject:** First information concerning the vehicle under development is collected and modeled. Functions of the vehicle (e.g. motoring or recuperative braking) are described. In addition, requirements are associated with these functions (e.g. requirements for activation or deactivation). In addition modes (e.g. drive, creep or acceleration) are associated with each function and with relevant requirements.
2) **Identification of Malfunctions and Hazards:** Based on the definition of the analysis subject, possible malfunctions of the vehicle are identified. This is carried out using guidewords (see Section IV-A). Each function is analyzed with regard to each guideword. When a guideword applies to a function, the violated requirements are identified. Thereafter the malfunction (e.g. *unintended* positive torque) is described and associated with the analysis subject and the violated requirements. Hazards (e.g. unintended acceleration of the vehicle) are derived for each malfunction and associated.
3) **Definition and Classification of Hazardous Events:** The next step is to identify typical traffic situations (oncoming traffic on a highway in a curve), maintenance situations (e.g. vehicle at lifting ramp) (see Section IV-B) as well as other operational situations. Moreover use cases that describe the behavior (e.g. overtaking or changing oil) of the related actors (e.g. driver or mechanic) are described. Thereafter *hazardous events* [2] are determined as relevant combinations of hazards, use cases and operational situations. Subsequently relevant modes are identified for each hazardous event and associated with it. The criticality of each hazardous event is assessed in terms of its *controllability*, *severity* and *exposure* [2] and an *ASIL* [3] (Automotive Safety Integrity Level) is assigned. This is underpinned by the definition of classification assumptions.
4) **Derivation of Safety Goals:** For each hazardous that has a sufficiently high ASIL, a safety goal is defined and associated. A safe state is defined (e.g. switch open) for each safety goal. Alternatively a safe mode (e.g. limp home mode) is determined for each hazardous event. These safety goals are the top level safety requirements to the safety-critical automotive embedded system.

After the completion of these working steps lower-level safety requirements can be derived from the safety goals and an embedded system architecture including fault detection capabilities can be defined. Based on these requirements, software and hardware of the safety-critical embedded system are implemented, integrated, verified and validated. However, these steps of the development process of a safety-critical automotive embedded system are beyond the scope of this paper.

The application of PHA for contemporary vehicles is challenging. The analysis subject is complex and potentially contains many sensors, actuators and functions, whose combination can lead to a multitude of hazards caused by failures of the embedded system. This complexity results in a large set of information to manage during PHA, and the analysis model grows in size. The exhaustive application of PHA and the projection of its results onto an analysis model that is complete, consistent and allows traceability is cumbersome and error prone.

Challenges are (1) the thorough understanding of the system functionalities and environment, (2) the correct application of a method for the systematic identification and classification of the related hazards including the derivation of safety goals, and (3) the correct modeling using the domain-specific language.

Therefore we propose to aid the process of applying PHA and creating the analysis model by automated property checking (Property Checker) as well as automated correction (Model Corrector) in order to detect an erroneously applied PHA and to enable corrections (see Section V and Section VI). Although not able to conduct PHA automatically, this approach strongly supports the process of PHA in providing a guidance for its application and the reflection of its results on the analysis model. This guidance allows identifying omissions, inconsistencies and missing traceability links during PHA and supports the creation of a consistent and complete set of safety goals.

IV. Analysis Model

In the course of the PHA, results are annotated using the language EAST-ADL (see Figure 1). This language is domain-specific and tailored to the needs of automotive embedded systems development. It is diagrammatic such as UML and consist of syntactic elements such as boxes, ovals, lines or arrows. Its *abstract syntax* is defined by its meta model and its *semantic domain* and *semantic mapping* [5] are defined using natural language. The EAST-ADL specification can be found in [6].

EAST-ADL allows to describe concepts relevant to the application of PHA in accordance with the safety standard ISO 26262. It allows the definition of a vehicle in terms of its functions (meta class VehicleFeature), modes (meta class Mode) and requirements (meta class Requirement). By using these modeling concepts, the analysis subject can be described. Furthermore meta classes are available that allow to describe malfunctions (meta class FeatureFlaw) and resulting hazards (Hazard). Moreover operational situations (meta class OperationalSituation) can be defined. The behavior of the relevant actors in the context of the operational situation can be described with use cases. A combination of hazard, mode, operational situation and use case defines a hazardous event (meta class HazardousEvent). Moreover top level safety requirements (meta class SafetyGoal) can be defined for hazardous events. Associations between the aforementioned concepts can be created to precisely define their relations and assure proper traceability.

We propose an enhancement to EAST-ADL that allows to define malfunctions and operational situations in more detail. The enhancement is defined in a similar manner as EAST-ADL in the following. The relation of the enhancement's meta model to EAST-ADL is depicted by Figure 2.

A. *Language enhancement: Malfunctions*

The use of guidewords has been proved to be beneficial and is also applied by approaches such as [7], [12], [15]. In our approach, the aim of the use of guidewords is to systematically identify potential malfunctions of the vehicle.

EAST-ADL does not include predefined guidewords to classify malfunctions (meta class FeatureFlaw). Therefore EAST-ADL's meta class FeatureFlaw has been extended to support the use of guidewords. Using our enhancement, malfunctions (meta class GuidedFeatureFlaw) can be characterized by the following guidewords.

- **No:** The vehicle omits to carry out a function although it is demanded.
- **Unintended:** A function is carried out without demand.
- **Reverse:** The vehicle carries out a function but fails in applying it in the demanded direction.
- **More:** The vehicle carries out an intended function but exceeds the demanded degree of intensity.
- **Less:** The vehicle carries out an intended function but falls below the demanded degree of intensity.
- **Other:** If none of the guidewords above are able to characterize the malfunction.

B. *Language enhancement: Operational Situations*

An operational situation is characterized by conditions external to the vehicle. If a hazard inevitably leads to an accident depends on the operational situation in context as well as on the behavior of the involved actors. As an example, if the vehicle brakes without demand of the driver (hazard) while the vehicle waits in front of a crosswalk and a pedestrian crosses, the result is harmless. In contrast if the same hazard occurs while the vehicle moves at high speed on a curvy road, the hazardous event can lead to a lethal accident. From this example it becomes obvious that an acceptably precise definition of operational situations is a cornerstone for the estimation of severity, exposure and controllability of the corresponding hazardous events since they determine the ASILs of the derived safety goals. The necessity of precisely describing operational situations in the context of hazards is also emphasized by other approaches [11], [15].

EAST-ADL rudimentarily supports the definition of generic operational situations. Frequently occurring operational situations are traffic situations (the vehicle is driven) and maintenance situations (the vehicle is being repaired or serviced). Therefore we propose an enhancement to EAST-ADL to facilitate a more accurate definition of traffic situations and maintenance situations. The meta classes TrafficSituation and MaintenanceSituation are derived from EAST-ADL's OperationalSituation. Each of these meta classes

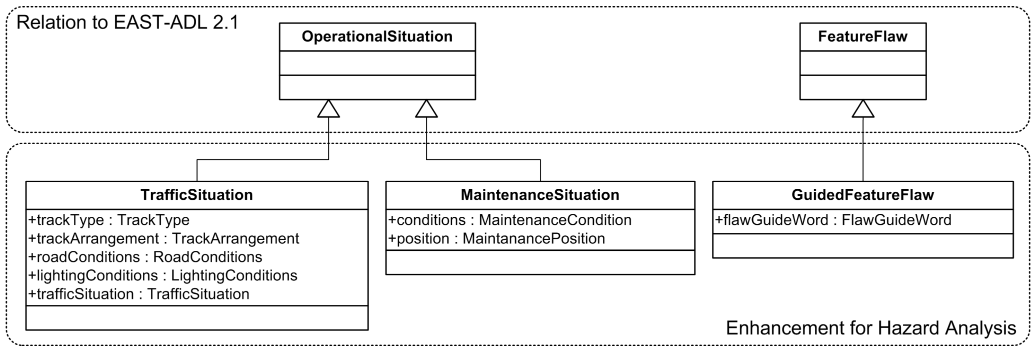

Relation to EAST-ADL 2.1
OperationalSituation
FeatureFlaw
TrafficSituation
+trackType : TrackType
+trackArrangement : TrackArrangement
+roadConditions : RoadConditions
+lightingConditions : LightingConditions
+trafficSituation : TrafficSituation
MaintenanceSituation
+conditions : MaintenanceCondition
+position : MaintanancePosition
GuidedFeatureFlaw
+flawGuideWord : FlawGuideWord
Enhancement for Hazard Analysis

Figure 2. Proposed enhancement to EAST-ADL for Preliminary Hazard Analysis (referenced enumerations are textually described).

can be described using attributes. For each attribute an enumeration is defined. Traffic situations can be coarsely characterized by following attributes:

- **Type of track:** The kind of the underlying track the vehicle is operating on such as highway, road, city street, mountainous track, crossings, drive-up, descent or parking area.
- **Track arrangement:** The geometry of the track the vehicle is operating on such as curve or straight.
- **Road conditions:** The road conditions the moving vehicle is exposed to because of the weather such as icy, normal, snowy or wet.
- **Lighting conditions:** The conditions that depend on the time of the day and the weather such as good visibility, medium visibility or poor visibility.
- **Traffic situation:** The conditions caused by other road users such as stop and go, crashing vehicles, colliding vehicles, pedestrian crossing, vehicle crossing, vehicle overtaking, adjacent vehicle, congestion, platoon traffic, oncoming traffic, no other road users, game crossing or street workers present.

Maintenance situations (MaintainanceSituation) can be coarsely defined by following attributes.

- **Maintenance position:** The position of the vehicle during maintenance such as lifting ramp or ground.
- **Maintenance conditions:** The conditions the maintenance operation is exposed to such as no other people or other people around.

Besides the defined value range, each enumeration contains the value *undefined*. This allows to express that a certain attribute is irrelevant in context of a hazardous event. This helps to collapse the number of traffic situations and maintenance situations to be dealt with during PHA.

V. Automated Property Checking

The analysis model that is annotated using EAST-ADL including the proposed enhancement reflects the results of the PHA. We propose properties that *indicate* the correct application of PHA, if they are fulfilled by the analysis model. If these properties are violated, the correct application of PHA is *not assured*. Note that the fulfillment of all the properties does not guarantee the exhaustiveness of the PHA, since the correct and complete understanding of the system by the team of people that applies PHA cannot be verified with this approach. This approach can show the erroneous application of PHA, but it cannot prove the absence of errors.

The defined properties are automatically checked using the evolving analysis model (see Figure 3). A property checker uses the definition of properties to continuously check the analysis model while PHA is carried out. It automatically identifies modeling elements that violate properties and presents a list of violating modeling elements and the properties affected. This information allows to early identify errors in the PHA before they can affect the definition of an adequate set of top level safety requirements to the safety-critical embedded system. Since automated checking is performed concurrently with the application of PHA and the creation of the analysis model, the property checker is also a valuable guide while carrying out PHA.

Input to the definition of these properties was (1) the domain-specific language EAST-ADL including the enhancements defined above and (2) the automotive safety standard ISO 26262 [3]. This standard defines requirements to the application of PHA for vehicles. Moreover the standard defines how the criticality of hazardous events shall be classified in terms of severity, exposure and controllability and defines how the required safety integrity level shall be derived. Moreover it is defined how safety goals shall be derived from defined hazardous events.

The properties are listed in Table I. Column *Meta Class* denotes the meta class of the enhanced EAST-ADL language (see Section IV) that can violate the corresponding property. Column *Property Definition* defines the properties for the corresponding meta classes in natural language.

Assume M is an analysis model, M_{MM} is the meta model of the enhanced EAST-ADL language and P is the set of properties as defined in Table I. Assume e is a modeling element of the analysis model, t is a type defined by the meta model of the analysis model and p is a property (Expression 1).

Property ID	Meta Class	Property Definition
0	Item	A complementary description has been defined
0a	Item	At least one VehicleFeature has been defined
1	Item	At least one Hazard has been identified
2	Item	At least one FeatureFlaw has been identified
0b	VehicleFeature	A complementary description has been defined
0c	VehicleFeature	Associated with at least one Item
23	VehicleFeature	A Requirement is satisfied
21	Requirement	An ID is defined
22	Requirement	A requirements text is defined
24	Requirement	Is satisfied
26	Mode	A condition has been defined
3	FeatureFlaw	At least one Hazard is identified
4	FeatureFlaw	Associated with at least one Item
5	FeatureFlaw	A complementary description has been defined
6	Hazard	At least one FeatureFlaw is associated
7	Hazard	At least one Item is associated
8	Hazard	At least one HazardousEvent has been identified
9	Hazard	A complementary description has been defined
10	HazardousEvent	At least one Hazard is associated
11	HazardousEvent	At least one UseCase is associated
12a	HazardousEvent	At least one SafetyGoal is associated if ASIL greater than QM
13	HazardousEvent	Associated with at least one OperationalSituation
14	HazardousEvent	ASIL has been correctly derived from Controllability, Severity and Exposure
17	HazardousEvent	Classification assumptions have been defined
25	HazardousEvent	Associated with at least one Mode
15	SafetyGoal	A HazardousEvent is associated
16	SafetyGoal	A safe state is defined
18	SafetyGoal	The ASIL has been correctly derived from associated HazardousEvents
20	OperationalSituation	A complementary description has been defined
27	All	A name must be assigned

Table I
PROPERTIES OF THE ANALYSIS MODEL THAT ARE AUTOMATICALLY CHECKED WHILE PRELIMINARY HAZARD ANALYSIS IS APPLIED.

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

Moreover, $I(e,t)$ pertains, if e is of type t, $D(t,p)$ pertains, if p is defined for type t and $H(e,p)$ pertains if property p holds for modeling element e. If a model M *indicates* the correct application of PHA, Expression 2 is valid. In this case, no modeling element violates a property.

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

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

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

VI. COMPUTER-AIDED MODEL CORRECTION

If violated properties indicate an erroneous application of PHA, corrective measures need to be carried out. To ease the correction of errors, we propose to support the identification of a proper correction measure. This can be achieved by consulting the model corrector that suggests possible solutions depending on the violated property and the current analysis model. Input to the definition of suggestions are again ISO 26262 and the enhanced language EAST-ADL. Depending on concerned model elements and violated properties, the model corrector identifies possible solutions (see Figure 3). If the user decides to accept a solution, the model is automatically modified accordingly.

Suggestions depending on the violated property are listed in Table II. Column *Meta Class* denotes the meta class of the enhanced EAST-ADL language that can be subject to the suggestion of an automated correction. Column *Suggestion* defines the possible suggestions for the corresponding meta classes in natural language.

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

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

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

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

If the user accepts suggestion s_1, the analysis model M is

Property ID	Meta Class	Suggested Solution
0	Item	Creation and association of Comment
0a	Item	Creation and association of VehicleFeature
0a	Item	Associate one of the VehicleFeatures without Item
1	Item	Creation and association of Hazard
1	Item	Associate one of the Hazards without Item
2	Item	Creation and association of FeatureFlaw
2	Item	Associate one of the FeatureFlaws without Item
0b	VehicleFeature	Creation and association of Comment
0c	VehicleFeature	Creation and association of Item
0c	VehicleFeature	Associate one of the Items
23	VehicleFeature	Creation and association of Requirement
23	VehicleFeature	Associate one of the unsatisfied Requirements
24	Requirement	Creation and association of VehicleFeature
24	Requirement	Associate one of the VehicleFeatures without Requirement
3	FeatureFlaw	Creation and association of Hazard
3	FeatureFlaw	Associate one of the Hazards without FeatureFlaw
3	FeatureFlaw	Associate one of the Hazards with FeatureFlaw
4	FeatureFlaw	Creation and association of Item
4	FeatureFlaw	Associate one of the Items
5	FeatureFlaw	Creation and association of Comment
6	Hazard	Creation and association of FeatureFlaw
6	Hazard	Associate one of the FeatureFlaws without Hazard
6	Hazard	Associate one of the FeatureFlaws with Hazard
7	Hazard	Creation and association of Item
7	Hazard	Associate one of the Items
8	Hazard	Creation and association of HazardousEvent
8	Hazard	Associate one of the HazardousEvents without Hazard
9	Hazard	Creation and association of Comment
10	HazardousEvent	Creation and association of Hazard
10	HazardousEvent	Associate one of the Hazards without HazardousEvent
10	HazardousEvent	Associate one of the Hazards with HazardousEvent
11	HazardousEvent	Creation and association of UseCase
11	HazardousEvent	Associate one of the UseCases
12a	HazardousEvent	Creation and association of SafetyGoal
12a	HazardousEvent	Associate one of the SafetyGoals without HazardousEvent
13	HazardousEvent	Creation and association of OperationalSituation
13	HazardousEvent	Associate one of the OperationalSituations
14	HazardousEvent	Correction of the ASIL according to the requirements of ISO 26262
25	HazardousEvent	Creation and association of Mode
26	HazardousEvent	Associate one of the Modes without HazardousEvent
15	SafetyGoal	Creation and association of HazardousEvent
15	SafetyGoal	Associate one of the HazardousEvents with ASIL larger than QM and without SafetyGoal
18	SafetyGoal	Modification of the ASIL according to the ASIL of the corresponding HazardousEvent
20	OperationalSituation	Creation and association of Comment

Table II
POSSIBLE SOLUTIONS TO PROBLEMS THAT ARE AUTOMATICALLY SUGGESTED DEPENDING ON THE ANALYSIS MODEL.

automatically corrected and transformed to analysis model M' by function γ depending on M, e_1, t_1, p_1 and s_1 (Expression 6).

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

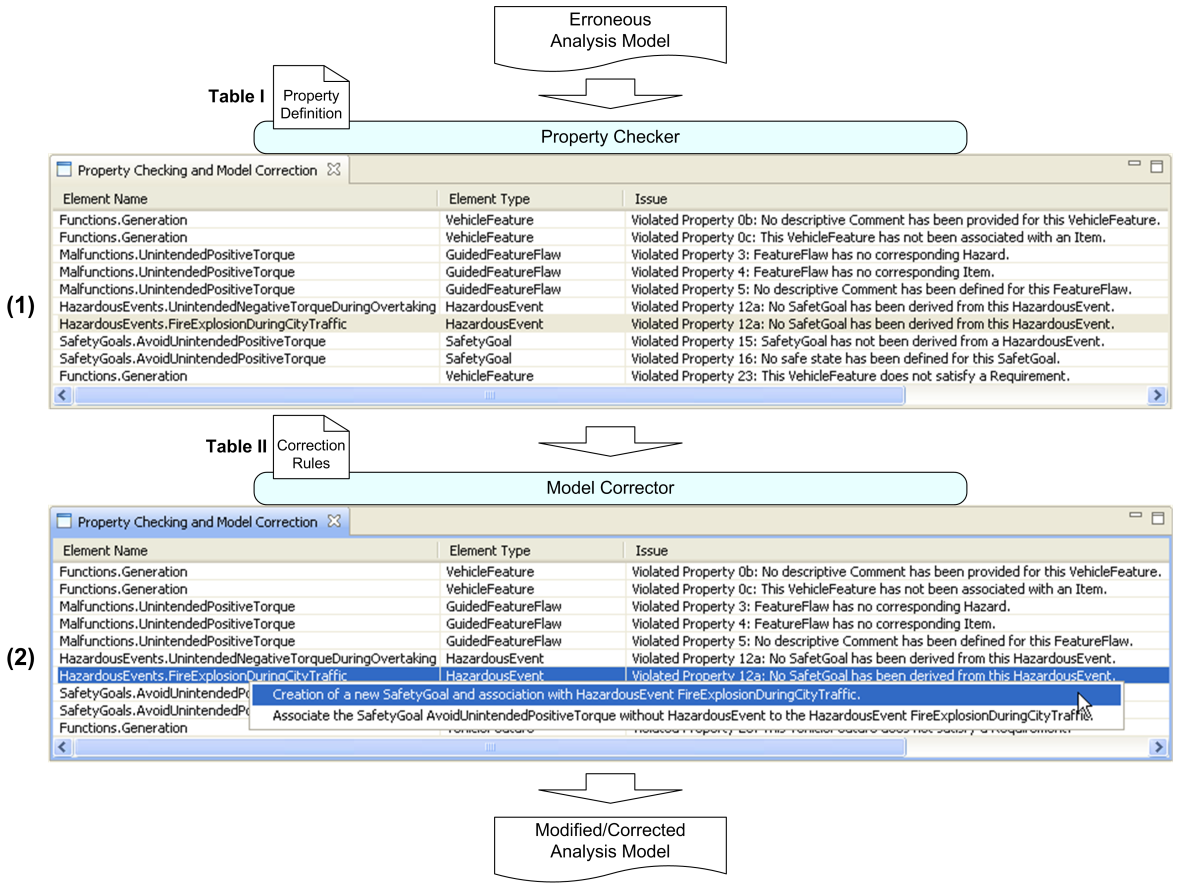
After the modification (postcondition) e_1 is an element of M', e_1 is still of type t_1 and does not violate p_1 any more (Expression 7).

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

VII. Experimental Evaluation

An Eclipse-based open source tool named Papyrus [17] is available that facilitates UML-modeling as well as the definition of UML profiles. An open source plugin is available for Papyrus [18] that allows the creation of EAST-ADL models. This plugin was enhanced to support the creation of analysis models such as described in Section IV. Another plugin for the Papyrus tool was developed that facilitates property checking as well as model correction such as proposed in Section V and Section VI.

Thereafter the proposed approach to PHA of automotive embedded systems was experimentally evaluated by the case study of HEV [20] development. One of the main characteristics of this type of vehicle is the addition of an electric motor that supports the classic combustion engine providing supplementary or substitutive positive torque. If such a vehicle uses its E-motor to support the combustion engine, it



Property Checking and Model Correction

Element Name	Element Type	Issue
Functions.Generation	VehicleFeature	Violated Property 0b: No descriptive Comment has been provided for this VehicleFeature.
Functions.Generation	VehicleFeature	Violated Property 0c: This VehicleFeature has not been associated with an Item.
Malfunctions.UnintendedPositiveTorque	GuidedFeatureFlaw	Violated Property 3: FeatureFlaw has no corresponding Hazard.
Malfunctions.UnintendedPositiveTorque	GuidedFeatureFlaw	Violated Property 4: FeatureFlaw has no corresponding Item.
Malfunctions.UnintendedPositiveTorque	GuidedFeatureFlaw	Violated Property 5: No descriptive Comment has been defined for this FeatureFlaw.
HazardousEvents.UnintendedNegativeTorqueDuringOvertaking	HazardousEvent	Violated Property 12a: No SafetGoal has been derived from this HazardousEvent.
HazardousEvents.FireExplosionDuringCityTraffic	HazardousEvent	Violated Property 12a: No SafetGoal has been derived from this HazardousEvent.
SafetyGoals.AvoidUnintendedPositiveTorque	SafetyGoal	Violated Property 15: SafetyGoal has not been derived from a HazardousEvent.
SafetyGoals.AvoidUnintendedPositiveTorque	SafetyGoal	Violated Property 16: No safe state has been defined for this SafetGoal.
Functions.Generation	VehicleFeature	Violated Property 23: This VehicleFeature does not satisfy a Requirement.

Property Checking and Model Correction

Element Name	Element Type	Issue
Functions.Generation	VehicleFeature	Violated Property 0b: No descriptive Comment has been provided for this VehicleFeature.
Functions.Generation	VehicleFeature	Violated Property 0c: This VehicleFeature has not been associated with an Item.
Malfunctions.UnintendedPositiveTorque	GuidedFeatureFlaw	Violated Property 3: FeatureFlaw has no corresponding Hazard.
Malfunctions.UnintendedPositiveTorque	GuidedFeatureFlaw	Violated Property 4: FeatureFlaw has no corresponding Item.
Malfunctions.UnintendedPositiveTorque	GuidedFeatureFlaw	Violated Property 5: No descriptive Comment has been defined for this FeatureFlaw.
HazardousEvents.UnintendedNegativeTorqueDuringOvertaking	HazardousEvent	Violated Property 12a: No SafetGoal has been derived from this HazardousEvent.
HazardousEvents.FireExplosionDuringCityTraffic	HazardousEvent	Violated Property 12a: No SafetGoal has been derived from this HazardousEvent.
SafetyGoals.AvoidUnintendedPo		
SafetyGoals.AvoidUnintendedPo		
Functions.Generation		

Creation of a new SafetyGoal and association with HazardousEvent FireExplosionDuringCityTraffic.
Associate the SafetyGoal AvoidUnintendedPositiveTorque without HazardousEvent to the HazardousEvent FireExplosionDuringCityTraffic.

Figure 3. Violated properties are automatically identified and possible solutions are suggested on demand.

discharges the battery. If the E-motor is used as a generator to regain energy while the vehicle decelerates (recuperation), it recharges the battery and/or supplies electrical energy to the auxiliaries. These operations are controlled by an embedded system. This embedded system is clearly safety-critical since a failure of this system can cause malfunctions such as overcharging of the battery that might lead to hazards such as fire and/or explosion.

For the experimental application of the approach, PHA was carried out and an analysis model was created in course of PHA. One of the modeling elements in this analysis model is a hazardous event named *FireExplosionDuringCityTraffic*. The origin of this hazardous event is the hazard *FireExplosion* that is caused by overcharging of the battery because of unintended negative torque provided by the electric motor to the powertrain due to a failure of the safety-critical embedded system. The unintended negative torque causes the E-motor acting as a generator that finally unintendedly overcharges the battery of the vehicle.

No safety goal (top level safety requirement) has been derived from this hazardous event although it is safety-critical. This omission leads to the fact that no top level safety requirement has been derived from the hazardous event *FireExplosionDuringCityTraffic* that demands the mitigation of the hazard *FireExplosion*. Thus PHA was applied erroneously what is reflected by the analysis model. Figure 3 illustrates the property checker (1) that has detected the erroneous application based on the analysis model and reports that modeling element *FireExplosionDuringCityTraffic* violates property 12a.

Due to the erroneous application of PHA, corrective measures must be carried out and the analysis model needs to be modified. As illustrated in Figure 3, on demand the model corrector (2) proposes the creation of a new safety goal or the association with a safety goal that is already part of the analysis model. In this case the proper solution is the creation of a new safety goal and the association with the hazardous event *FireExplosionDuringCityTraffic*. Once selected, the analysis model is automatically modified and a new traceable safety goal is created. Subsequently the newly created modeling element can be refined using textual descriptions.

VIII. Conclusion

This work presents a novel approach to Preliminary Hazard Analysis (PHA) for automotive embedded systems.

The proposed framework comprises (1) an enhancement to the domain-specific language EAST-ADL, as well as (2) the identification of properties that indicate the correct application of PHA. These properties can be automatically checked based on the analysis model that reflects the results of the PHA. If properties are violated, the approach supports the automated identification of possible solutions and the automated correction of the analysis model. This guided and computer-aided approach strongly supports the application of PHA and the creation of an analysis model that properly reflects the results of PHA. The approach has been evaluated using the case study of hybrid electric vehicle development. While the use of the property checker and the model corrector did not replace the intellectual process of carrying out PHA exhaustively by a team of people with a wide variety of knowledge and skills, the automated identification of an erroneously applied PHA and the guided correction during the analysis process proved to be highly valuable to improve the quality of the analysis.

ACKNOWLEDGMENT

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

REFERENCES

[1] Nancy G. Leveson, *Safeware: system safety and computers*. Addison-Wesley Publishing Company, 1995.

[2] International Organization for Standardization, “ISO/DIS 26262-1 Road vehicles - Functional safety - Part 1: Vocabulary,” 2009.

[3] ——, “ISO/DIS 26262-3 Road vehicles - Functional safety - Part 3: Concept phase,” 2009.

[4] M. Törngren, D. Chen, D. Malvius, and J. Axelsson, “Model-Based Development of Automotive Embedded Systems,” in *Automotive Embedded Systems Handbook*. CRC Press, 2008, ch. 10.

[5] D. Harel and B. Rumpe, “Meaningful Modeling: What’s the Semantics of "Semantics"?” *IEEE Transactions on Computers*, vol. 37, pp. 64–72, Oct. 2004.

[6] ATESST2 Project Consortium, “EAST-ADL Domain Model Specification,” 2010, version 2.1, Release Candidate 3.

[7] P. Johannessen, F. Törner, and J. Torin, “Actuator Based Hazard Analysis for Safety Critical Systems,” in *Proc. of the 23th International Conference on Computer Safety, Reliability and Security*, Sep. 2004, pp. 130–141.

[8] F. Törner, P. Johannessen, and P. Öhman, “Assessment of Hazard Identification Methods for the Automotive Domain,” in *Proc. of the 25th International Conference on Computer Safety, Reliability and Security*, Sep. 2006, pp. 247–260.

[9] P. Jesty, D. Ward, and R. Rivett, “Hazard Analysis for Programmable Automotive Systems,” in *Proc. of the 2nd IET International Conference on System Safety 2007*, Dec. 2007, pp. 106–111.

[10] H. Schubotz, “Hazard Analysis and Risk Assessment for Complex EE-Architectures,” in *Proc. of the SAE World Congress & Exhibition*, no. 2010-01-0029, Apr. 2010.

[11] M. Stringfellow, N. Leveson, and B. Owens, “Safety-Driven Design for Software-Intensive Aerospace and Automotive Systems,” *Proceedings of the IEEE*, vol. 98, pp. 515–525, 2010.

[12] J. Michael, M.-T. Shing, K. Cruickshank, and P. Redmond, “Hazard Analysis and Validation Metrics Framework for System of Systems Software Safety,” *IEEE Systems Journal*, vol. 4, pp. 186–197, 2010.

[13] S. Kumar, P. Ramaiah, and V. Khanaa, “A Methodology for Building Safer Software based Critical Computing Systems,” in *Proc. of the 2nd IEEE International Conference on Advance Computing (IACC’2010)*, Feb. 2010, pp. 422–429.

[14] H. Giese, M. Tichy, and D. Schilling, “Compositional Hazard Analysis of UML Component and Deployment Models,” in *Proc. of the 23th International Conference on Computer Safety, Reliability and Security*, Sep. 2004, pp. 166–179.

[15] K. Allenby and T. Kelly, “Deriving Safety Requirements Using Scenarios,” in *Proc. of the 5th IEEE International Symposium on Requirements Engineering*, Aug. 2001, pp. 228–235.

[16] A. Sandberg, D.-J. Chen, H. Lönn, R. Johansson, L. Feng, M. Törngren, S. Torchiaro, R. T. Kolagari, and A. Abele, “Model-Based Safety Engineering of Interdependent Functions in Automotive Vehicles Using EAST-ADL2,” in *Proc. of the 29th International Conference on Computer Safety, Reliability and Security*, Sep. 2010, pp. 332–346.

[17] A. Lanusse, Y. Tanguy, H. Espinoza, C. Mraidha, S. Gerard, P. Tessier, R. Schnekenburger, H. Dubois, and F. Terrier, “Papyrus UML: an open source toolset for MDA.” in *Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009)*, Jun. 2009, pp. 1–4.

[18] ATESST2 Project Consortium, “Refined EAST-ADL2 tool support,” Tech. Rep., 2010, Deliverable D3.2.

[19] D. Makartetskiy, D. Pozza, and R. Sisto, “An Overview of Software-based Support Tools for ISO 26262,” in *Proc. of the 3rd International Workshop on Innovation in Information Technologies: Theory and Practice*, Sep. 2010, pp. 1–6.

[20] M. Ehsani, Y. Gao, S. Gay, and A. Emadi, “Hybrid Electric Vehicles,” in *Modern Electric, Hybrid Electric, and Fuel Cell Vehicles Fundamentals, Theory, and Design*. CRC Press, 2005, ch. 5.

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

Roland Mader[1,2], Eric Armengaud[1,3], Andrea Leitner[2], Christian Kreiner[2], Quentin Bourrouilh[1], Gerhard Grießnig[1,2], Christian Steger[2], and Reinhold Weiß[2]

[1] AVL List GmbH
{roland.mader,quentin.bourrouilh,gerhard.griessnig}@avl.com
[2] Institute for Technical Informatics, Graz University of Technology
{andrea.leitner,christian.kreiner,steger,rweiss}@tugraz.at
[3] Virtual Vehicle Competence Center (ViF)
eric.armengaud@v2c2.at

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

1 Introduction

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

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

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

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

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

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

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

2 Related Work

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

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

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

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

The authors of [11] integrate architectural modeling languages with safety analysis languages to improve consistency. When a safety-critical software architecture is developed an initial architecture is proposed. This architecture is annotated and enriched with safety-relevant information. Safety analysis of the architecture is carried out. Results influence the software architecture. This design and analysis process is cyclic. A meta model for component-based, safety-aware architectures (SAA) is available allowing to complement architectural descriptions with safety-relevant information such as safety objectives and mitigation means. Meta models for FTA and FMECA (Failure Modes, Effects and Criticality Analysis) are proposed. A tool implementation is presented that allows the generation of FTA models and FMECA models from a SAA model.

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

3 Safety Engineering Workflow

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

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

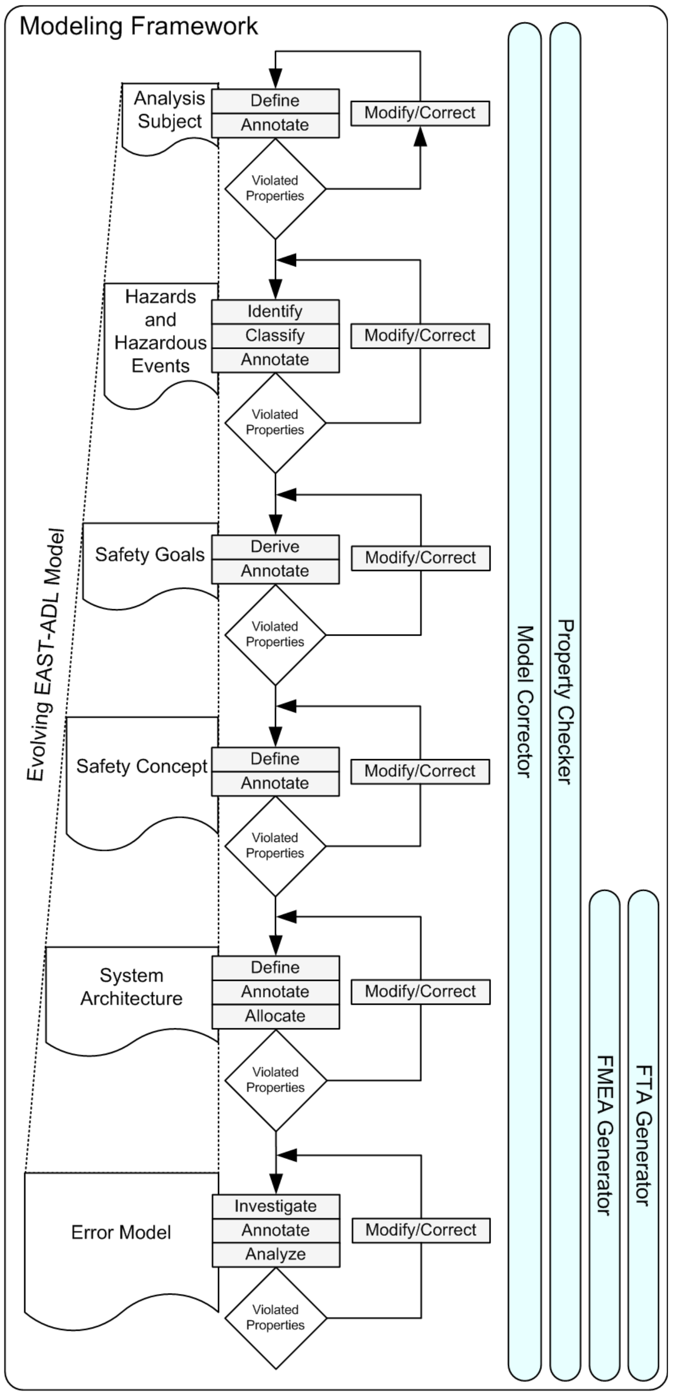


Fig. 1. Computer-Aided Safety Engineering Workflow

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

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

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

Due to the complexity of contemporary vehicles, the application of the workflow is cumbersome and error-prone. Therefore we propose to aid the safety engineering workflow defined above by automated property checking (property checker), automated model correction (model corrector), automated fault tree synthesis and automated FMEA table synthesis (see Section 4 and Section 5). This allows to early identify erroneously applied working steps and enables the automated suggestion and application of corrective measures. Moreover it is not necessary to construct fault trees and FMEA tables manually. Instead, they are consistently generated from the EAST-ADL model. While property checker and model corrector aid the entire workflow, FTA generator and FMEA generator are especially useful for the verification of the system architecture defining an embedded system together with connected sensors and controlled actuators. The automated analysis and synthesis capabilities of the modeling framework provide guidance and strongly support the application of the workflow and the creation of a complete and consistent set of work products.

4 Computer-Aided Checking

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

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

$$e \epsilon M, t \epsilon M_{MM}, p \epsilon P \tag{1}$$

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

$$\neg \exists e \neg \exists t \neg \exists p (I(e,t) \wedge D(t,p) \wedge \neg H(e,p)) \tag{2}$$

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

$$\exists e \exists t \exists p (I(e,t) \wedge D(t,p) \wedge \neg H(e,p)) \tag{3}$$

5 Automated Synthesis

5.1 Model Correction

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

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

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

Table 2. Possible solutions to problems that are automatically suggested

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

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

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

$$e_1 \epsilon M, t_1 \epsilon M_{MM}, p_1 \epsilon P, s_1 \epsilon S \tag{4}$$

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

$$I(e_1, t_1) \wedge D(t_1, p_1) \wedge \neg H(e_1, p_1) \tag{5}$$

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

$$\gamma(M, e_1, t_1, p_1, s_1) \rightarrow M' \tag{6}$$

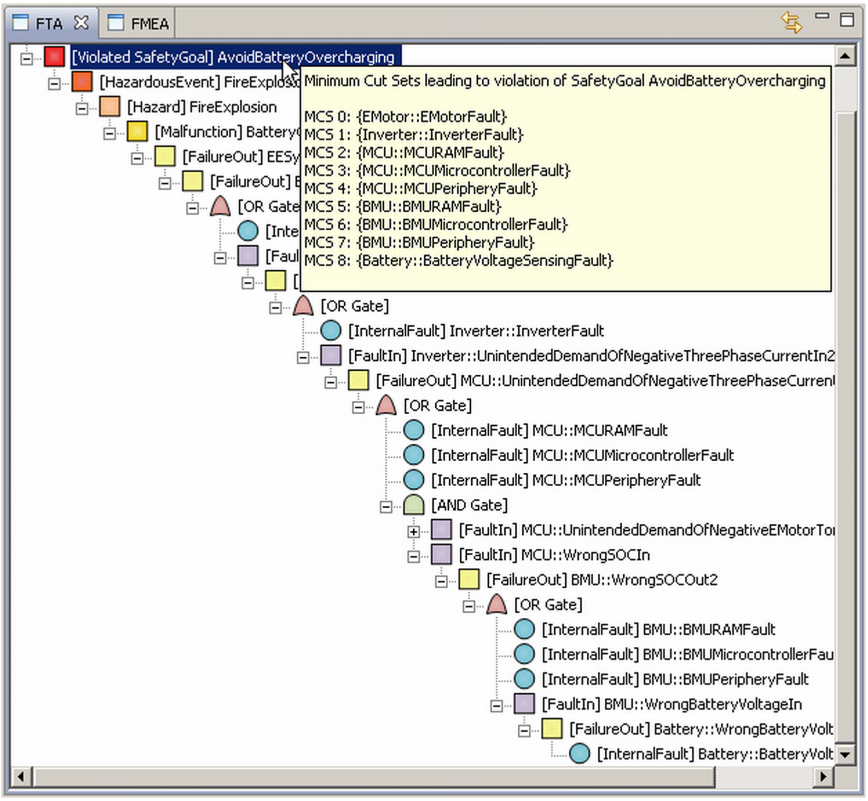
After the modification (postcondition) e_1 is an element of M', e_1 is still of type t_1 and does not violate p_1 any more (Expression 7).

$$e_1 \epsilon M', I(e_1, t_1) \wedge D(t_1, p_1) \wedge H(e_1, p_1) \tag{7}$$

5.2 Fault Tree and FMEA Table Synthesis

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

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



```
[Violated SafetyGoal] AvoidBatteryOvercharging
  [HazardousEvent] FireExplos
    [Hazard] FireExplosion
      [Malfunction] Battery
        [FailureOut] EESy
          [FailureOut] E
            [OR Gate
              [Inte
              [Faul
                [
                  [OR Gate]
                    [InternalFault] Inverter::InverterFault
                    [FaultIn] Inverter::UnintendedDemandOfNegativeThreePhaseCurrentIn2
                      [FailureOut] MCU::UnintendedDemandOfNegativeThreePhaseCurren
                        [OR Gate]
                          [InternalFault] MCU::MCURAMFault
                          [InternalFault] MCU::MCUMicrocontrollerFault
                          [InternalFault] MCU::MCUPeripheryFault
                          [AND Gate]
                            [FaultIn] MCU::UnintendedDemandOfNegativeEMotorTo
                            [FaultIn] MCU::WrongSOCIn
                              [FailureOut] BMU::WrongSOCOut2
                                [OR Gate]
                                  [InternalFault] BMU::BMURAMFault
                                  [InternalFault] BMU::BMUMicrocontrollerFau
                                  [InternalFault] BMU::BMUPeripheryFault
                                  [FaultIn] BMU::WrongBatteryVoltageIn
                                    [FailureOut] Battery::WrongBatteryVolt
                                      [InternalFault] Battery::BatteryVolt
```

```
Minimum Cut Sets leading to violation of SafetyGoal AvoidBatteryOvercharging

MCS 0: {EMotor::EMotorFault}
MCS 1: {Inverter::InverterFault}
MCS 2: {MCU::MCURAMFault}
MCS 3: {MCU::MCUMicrocontrollerFault}
MCS 4: {MCU::MCUPeripheryFault}
MCS 5: {BMU::BMURAMFault}
MCS 6: {BMU::BMUMicrocontrollerFault}
MCS 7: {BMU::BMUPeripheryFault}
MCS 8: {Battery::BatteryVoltageSensingFault}
```

Fig. 2. Fault trees can be synthesized from the EAST-ADL model

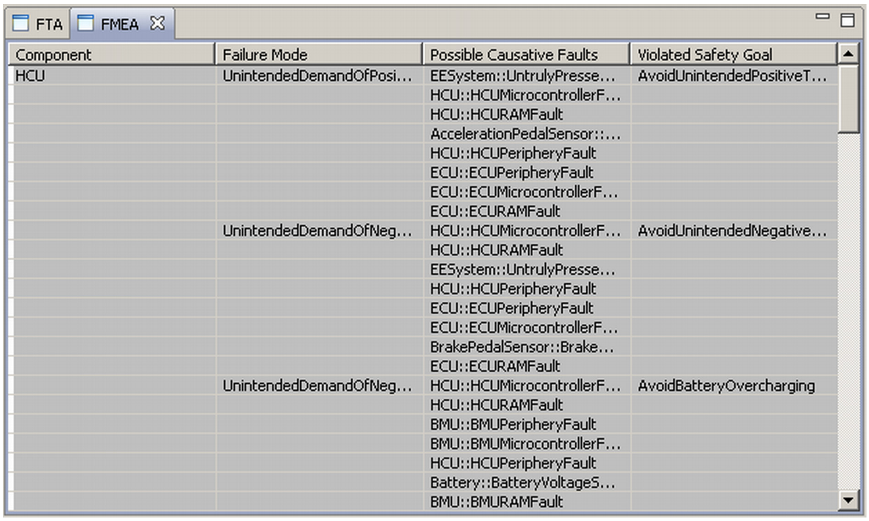
fault tree per safety goal is generated) and adds the causative malfunctions, hazards and hazardous events that were identified during PHA as offsprings. Component faults, component failures and gates that were identified during the definition of the error model are used as offsprings of the malfunctions. Thus the generated fault trees are consistent to the earlier elaborated PHA results.

It is possible to automatically extract the minimum cut sets from each fault tree. A *minimum cut set* [12] is a set of basic events leading to the top event (violation of the safety goal) that cannot be reduced in number. For every violated safety goal the minimum cut sets can be displayed on demand (see Figure 2).

Assume M is an EAST-ADL model and S is the subset of M that contains hazards, hazardous events, safety goals, system architecture and the error model (see Expression 8).

$$S \subseteq M \tag{8}$$

Given that Expression 2 holds for all $e \epsilon S$, FTA generator $\rho(S)$ can generate graphical fault trees Υ that allow examining how component faults and failures can contribute to the violation of safety goals (see Expression 9).



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

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

$$\rho(S) \rightarrow \Upsilon \tag{9}$$

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

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

$$\alpha(S) \rightarrow \Xi \tag{10}$$

6 Experimental Evaluation

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

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

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

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

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

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

7 Conclusion

This work presents a modeling framework with analysis and synthesis capabilities. This modeling framework supports a safety engineering workflow. In the course of the workflow a model is annotated using the domain-specific language EAST-ADL. This model integrates the work products of the workflow phases. The modeling framework contains a property checker that allows to unveil the incorrect application of the workflow and a model corrector that suggests and automatically performs corrections of the evolving model. Moreover fault trees and a FMEA table can be automatically synthesized allowing the application of qualitative FTA and FMEA. This tightly integrated approach ensures consistency of PHA results, fault trees and FMEA table. The approach was evaluated using the case study of a hybrid electric vehicle development. While the analysis and synthesis capabilities of the modeling framework did not replace the intellectual process of applying the workflow, they strongly supported its application.

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

References

1. ATESST2 Project Consortium: EAST-ADL Domain Model Specification, version 2.1, Release Candidate 3 (2010)
2. Biehl, M., DeJui, C., Törngren, M.: Integrating Safety Analysis into the Model-based Development Toolchain of Automotive Embedded Systems. In: Proc. of the Conference on Languages, Compilers and Tools for Embedded Systems, pp. 125–131 (2010)
3. Domis, D., Trapp, M.: Integrating Safety Analyses and Component-Based Design. In: Proc. of the 27th International Conference on Computer Safety, Reliability and Security, pp. 58–71 (September 2008)
4. Elmqvist, J., Nadjm-Tehrani, S.: Tool Support for Incremental Failure Mode and Effects Analysis of Component-Based Systems. In: Proc. of the Design, Automation and Test in Europe Conference and Exhibition (DATE 2008), pp. 921–927 (April 2008)
5. Harel, D., Rumpe, B.: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Transactions on Computers 37, 64–72 (2004)
6. International Electrotechnical Commission: IEC 61025 - Ed. 2.0 Fault tree analysis (FTA) (2006)
7. International Organization for Standardization: ISO/DIS 26262-3 Road vehicles - Functional safety - Part 3: Concept phase (2009)
8. Lanusse, A., Tanguy, Y., Espinoza, H., Mraidha, C., Gerard, S., Tessier, P., Schnekenburger, R., Dubois, H., Terrier, F.: Papyrus UML: an open source toolset for MDA. In: Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009), pp. 1–4 (June 2009)
9. Mader, R., Grießnig, G., Leitner, A., Kreiner, C., Bourrouilh, Q., Armengaud, E., Steger, C., Weiß, R.: A Computer-Aided Approach to Preliminary Hazard Analysis for Automotive Embedded Systems. In: Proc. of the IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS), pp. 169–178 (2011)

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

Automatic and Optimal Allocation of Safety Integrity Levels

Roland Mader, AVL List GmbH

Eric Armengaud, AVL List GmbH

Andrea Leitner, Graz University of Technology

Christian Steger, Graz University of Technology

Key Words: Automotive Safety Integrity Level, Functional Safety, ISO 26262, Integer Linear Programming

SUMMARY & CONCLUSIONS

Powertrain electrification of vehicles leads to a higher number of sensors, actuators and control functions resulting in increasing complexity. Due to the safety-criticality of the functionalities, safety standards must be considered during system development. The safety standard ISO 26262 defines discrete ASILs (Automotive Safety Integrity Levels) that must be identified and allocated to the components of the system under development. Once allocated, they determine the applicable requirements of ISO 26262 and the necessary safety measures to accordingly minimize residual risk. Furthermore, the allocated ASILs directly influence the development efforts and the costs per piece of the system components. Manual elaboration of an ASIL allocation that is economic and assures functional safety is complex and cumbersome. This work presents a method that allows the automatic allocation of ASILs to the system components. In our approach ASIL allocation is interpreted as an ILP (Integer Linear Programming) problem. This allows obtaining an ASIL allocation that is optimal with respect to an objective function that is subject to constraints. These constraints are derived from the results of PHA (Preliminary Hazard Analysis), FTA (Fault Tree Analysis) and preferences of the safety engineer. The approach is evaluated by the case study of hybrid electric vehicle development.

1 INTRODUCTION

Powertrain electrification of vehicles transforms powertrains into complex, mechatronic systems. Due to the safety-criticality of interconnected ECUs (electronic control units), connected sensors and controlled actuators they are developed according to safety standards such as the automotive functional safety standard ISO 26262 [1].

This standard attempts to quantify functional safety by defining discrete ASILs (Automotive Safety Integrity Levels). They are QM (not safety-critical), ASIL A (least stringent), ASIL B, ASIL C, and ASIL D (most stringent). The ASILs allocated to the components of the system under development determine applicable requirements of ISO 26262 and the necessary safety measures to avoid unreasonable residual risk. The requirements of ISO 26262 affect the stringency of the development process of the system components (e.g. the portfolio of applicable verification techniques). The applicable safety measures determine the runtime fault detection capabilities of the system components. Thus the allocated ASILs heavily influence safety of the system in the context of its environment (another system that interacts with the sensors and actuators). Furthermore the allocated ASILs influence development costs and costs per piece.

An allocation of ASILs to the system under development shall therefore (1) assure that the required level of functional safety is achieved and (2) permit an economic solution with respect to development costs and cost per piece. Manual elaboration of an ASIL allocation that fulfills the requirements is complex, cumbersome and should be automated.

In this work a modeling framework is presented that supports an automotive safety engineering workflow that includes ASIL allocation. The work products of this workflow are annotated using the domain-specific language EAST-ADL (Electronics Architecture and Software Technology-Architecture Description Language) [2]. The contribution of this work is an automatic method for the allocation of ASILs to the system under development. Inputs to this method are results of PHA (Preliminary Hazard Analysis), FTA (Fault Tree Analysis) [3] and additional preferences of the safety engineer concerning the ASIL allocation. We show that the allocation of ASILs to the components of the system under development can be interpreted as an ILP (Integer Linear Programming) [11] problem. If solved, the solution is optimal with respect to an objective function and is consistent to the results of PHA, FTA and defined preferences. Furthermore the solution can be automatically projected on the evolving EAST-ADL model making a manual model modification superfluous. The applicability of the approach is evaluated by the case study of HEV (hybrid electric vehicle) development.

The remainder of the work is organized as follows. Section 2 reviews related work. Section 3 presents the automotive safety engineering workflow. Section 4 describes the modeling framework that allows applying the workflow including automatic and optimal ASIL allocation. Section 5 describes the approach to ASIL allocation. Finally Section 6 describes the experimental evaluation of the approach.

2 RELATED WORK

In [4] a workflow for embedded systems development is presented that aims on integration of functional modeling and safety analyses. They present a V-model that incorporates the application of PHA, FTA and FMEA (Failure Modes and Effects Analysis). Furthermore the workflow includes the determination of ASILs using a qualitative risk graph method.

In [5] an approach is presented that allows to automatically prove the correctness and completeness of fault trees based on a formal model. This is achieved using a model checking technique. The fault trees are quantitative and their top events are the PFDs (probabilities for failure on demand) of SIFs (safety instrumented functions). These quantities are used to derive the safety integrity levels for the SIFs of a SIS (safety instrumented system) according to IEC 61508.

In [6] a fuzzy probabilistic technique is presented that aims on compliance with IEC 61508. The method allows evaluating the SIL (Safety Integrity Level) of a SIS under development. In contrast to other quantitative techniques for SIL determination that highly depend on the credibility of the used data, the approach is tolerant towards uncertainties about the component failure probabilities. Failure rates and fuzzy probabilities are used to evaluate the fuzzy SIS PFD and the SIL of the SIS.

The works presented in [4], [5] and [6] are focused on the qualitative or quantitative determination of safety integrity levels. In contrast this work is focused on the allocation of ASILs to the components of a system under development.

In [7] an approach to the automatic allocation of SILs to subsystems and components of complex hierarchical networked architectures is presented. The approach can be used in the context of development using EAST-ADL. The approach supports ASIL decomposition [1]. Thus, if a component contributes to a failure only in conjunction with other components it may receive a lower SIL than a component that directly causes the failure. Minimum cut sets that were extracted during FTA are input to an algorithm that computes possible allocations of SILs to the components. The most promising potential allocations are presented to the user.

The approach presented in [7] potentially leads to lots of possible ASIL allocations that must be investigated by the safety engineer to select the preferred allocation. In contrast, our approach leads to a single allocation that is optimal with respect to an objective function and considers preferences of the safety engineer that were defined in advance.

3 SAFETY ENGINEERING WORKFLOW

This section describes an ISO 26262-compatible, automotive safety engineering workflow that is based on the work described in [8], [9] and [10]. This workflow is subdivided into multiple phases and allows iterations, if necessary. An EAST-ADL model is annotated, systematically enhanced and refined in course of this workflow. EAST-ADL is a domain-specific language for the automotive domain. It is UML-based (Unified Modeling Language) and diagrammatic. Its abstract syntax is defined by a meta model and its semantic domain and semantic mapping are defined using natural language. Control units, sensors, actuators, related concepts (e.g. requirements, features, hazards, faults and failures), their relations and their dependencies can be described using EAST-ADL. Thereafter the workflow phases are described.

3.1 Definition of the Analysis Subject

First information about the vehicle under development is collected and modeled. Functions of the vehicle (e.g. motoring or recuperative braking) are defined. Requirements to these functions are determined and allocated (e.g. conditions for activation or deactivation). In addition relevant modes (e.g. drive, creep or acceleration) are identified for each function and associated with the requirements.

3.2 Identification of Hazards and Hazardous Events

Based on the definition of the analysis subject, PHA (for more details see also [9]) is carried out. Possible malfunctions are identified. Hazards are derived for each malfunction (e.g. unintended acceleration of the vehicle). Thereafter operational situations such as traffic situations (e.g. oncoming traffic on a highway in a curve) and maintenance situations (e.g. vehicle at lifting ramp) are defined. Moreover use cases describing the behavior (e.g. overtaking or changing oil) of the related actors (e.g. driver or mechanic) are described. Hazardous events are determined for relevant combinations of hazards, use cases and operational situations. Moreover relevant modes are identified for each hazardous event. The criticality of each hazardous event is assessed in terms of its controllability, severity and exposure and an ASIL is determined.

3.3 Derivation of Safety Goals

For each hazardous event that has an ASIL assigned (ASIL A, ASIL B, ASIL C or ASIL D), a safety goal is derived and associated. Furthermore, a safe state is defined (e.g. switch open) for each safety goal. Alternatively a safe mode (e.g. limp home mode) is determined. The determined safety goals are top-level safety requirements.

3.4 Definition of Safety Concept

The safety concept is derived from the safety goals. This safety concept consists of functional and technical safety requirements to the automotive embedded system, connected sensors and controlled actuators. Traces are created between safety goals, functional safety requirements and technical safety requirements.

3.5 Definition of System Architecture

The system architecture is defined in terms of the embedded system, connected sensors and controlled actuators. Moreover the parts of the environment are modeled that interact with the sensors and actuators. Thereafter the functional and technical safety requirements are allocated to the components of the system architecture. Furthermore functions are allocated to the system components.

3.6 Investigation and Annotation of Faults and Failures

Information flows and energy flows through the embedded system, connected sensors, controlled actuators and their environment are investigated. Possible faults and failures are estimated and their propagation is analyzed and annotated. Moreover it is investigated and annotated how the failures lead to the malfunctions that were identified during PHA. Thereafter FTA and FMEA [3] are applied. Finally, ASILs are allocated to the components of the system (this paper is focused on this workflow activity). The allocated ASILs determine applicable requirements of ISO 26262 (e.g. process stringency) as well as the necessary safety measures (e.g. runtime fault detection capabilities). Furthermore, they determine the efforts for achieving functional safety with respect to the development efforts as well as the costs per piece of the system components.

4 MODELING FRAMEWORK

An EAST-ADL model is elaborated in course of the safety engineering workflow that contains its work products. This model is created using a framework that supports EAST-ADL modeling.

Furthermore, the modeling framework supports the automatic generation of fault trees from a subset of the EAST-ADL model. The violated safety goals that were identified during PHA are used as top events of these fault trees. Their descendants are causative hazardous events, hazards, malfunctions, faults and failures. Minimum cut sets can be automatically extracted from the fault trees. A minimum cut set [3] is a set of basic events leading to the top event (violation of safety goal) that cannot be reduced in number. The automated generation of fault trees supports the application of FTA and the verification of the system architecture. Furthermore the automatically extracted minimum cut sets are input to the presented approach to ASIL allocation. For more details on fault tree generation, illustrations of the fault trees and a description of other capabilities of the modeling framework refer to [9] and [10].

The modeling framework is equipped with a constraint solver. This constraint solver can automatically find an allocation of ASILs to the components of the system. This allocation depends on the safety goals defined during PHA, the minimum cut sets extracted during FTA as well as particular preferences of the safety engineer. The allocation is optimal with respect to an objective function and ensures consistency to PHA results and FTA results. Preferences can be defined using a dedicated graphical user interface that is illustrated in Figure 1. Once an optimal solution is found, the solution can be automatically projected on the EAST-ADL model, making a manual model modification superfluous. The approach to ASIL allocation is thereafter described.

5 ASIL ALLOCATION

5.1 Integer Linear Programing Problems

Linear programming denotes an approach to modeling and solution of linear mathematical models and more specifically those models that seek to optimize a linear measure of performance [11]. A single-objective linear programming model can be stated mathematically. Find the variables $x_1, x_2, \dots, x_n \geq 0$ so as to optimize (either maximize or minimize) the objective function that is subject to specified constraints. Expression 1 defines the objective function z.

$$z(x_1, x_2, \dots., x_n) = \sum_{i=0}^{n} x_i \cdot c_i \qquad (1)$$

In addition the m specified constraints are defined by Expression 2 for the n variables.

$$\begin{gathered} x_1 \cdot c_{11} + \cdots + x_n \cdot c_{1n} \{\leq, =, \geq\} b_1 \\ x_1 \cdot c_{21} + \cdots + x_n \cdot c_{2n} \{\leq, =, \geq\} b_2 \\ \vdots \\ x_1 \cdot c_{m1} + \cdots + x_n \cdot c_{mn} \{\leq, =, \geq\} b_m \end{gathered} \qquad (2)$$

An integer linear programming problem is a linear program in which some or all of the variables $x_1, x_2, \dots, x_n$ are restricted to integer values.

5.2 ASIL Arithmetic

ISO 26262 defines a method called ASIL decomposition. This method allows reducing the ASILs allocated to sufficiently independent components that can only jointly cause the violation of a safety goal. If a component can solely cause the violation of a safety goal its ASIL cannot be reduced.

Rules for the reduction of the ASILs are defined. To formalize these rules, QM, ASIL A, ASIL B, ASIL C and ASIL D can be interpreted as the integer numbers 0, 1, 2, 3 and 4.

Assume SG_j is a safety goal. Furthermore assume $asil(SG_j)$ denotes the ASIL of a safety goal interpreted as an integer number (see Expression 3).

$$asil(SG_j) \epsilon \{1,2,3,4\} \qquad (3)$$

Assume C_k denotes a component of the embedded system, a connected sensor or controlled actuator. Furthermore assume $asil(C_k)$ denotes the ASIL that is allocated to the component C_k interpreted as an integer number (see Expression 4).

$$asil(C_k) \epsilon \{0,1,2,3,4\} \qquad (4)$$

According to ISO 26262, if ASIL decomposition is applied for l components that can only jointly cause the violation of the safety goal SG_j, an inequality shall be fulfilled (see Expression 5).

$$\sum_{k=0}^{l} asil(C_k) \geq asil(SG_j) \qquad (5)$$

This implies that ASILs, interpreted as integer numbers, can be added resulting in a new ASIL that is limited to 4.

5.3 ASIL Allocation as Integer Linear Programing Problem

The problem of allocating ASILs to the components of the system under development can be interpreted as an integer linear programing problem. This approach is based on the assumption that (a) ASILs can be interpreted as integer numbers and (b) ASILs can be input to additions that result in a new ASIL (see also Section 5.2).

Generally, the greater an allocated ASIL, the greater the efforts for the development process and the higher the costs per piece of a component. It can therefore be assumed that a low sum of allocated ASILs leads to an economic ASIL allocation for the system under development. An objective function F_{min} can be defined that shall be minimized in order to find an economic ASIL allocation to n components (see Expression 6).

$$F_{min}(C_1, C_2, \dots, C_n) = \sum_{i=0}^{n} asil(C_i) \qquad (6)$$

Safety goals can be modeled using EAST-ADL and ASILs can be assigned. Assume that SGS is the set of safety goals that was defined in course of the safety engineering workflow (see Expression 7).

$$SG_k \in SGS \qquad (7)$$

EAST-ADL allows modeling of components (e.g. sensor, actuator, control unit) of the system under development and the system environment. Furthermore faults, failures and their propagation can be modeled. Fault and failures are unambiguously relatable to the components of the system under development or the system environment. Assume f_i is a fault or failure of the system under development or the system environment.

Assume f_i is part of minimum cut set MCS_j that was automatically extracted during FTA and can lead to the violation of safety goal SG_k (see Expression 8).

$$f_i \in MCS_j \qquad (8)$$

Based on the set of safety goals SGS and the minimum cut sets that contribute to the violation of the safety goals, l constraints can be defined for m safety goals and n components (see Expression 9). These constraints assure that the allocation of ASILs is consistent to the safety goals and the minimum cut sets and assure the achievement of functional safety according to ISO 26262.

A constraint is defined for each minimum cut set that can lead to the violation of a safety goal in SGS and exclusively consists of faults and failures that result from a component C_k of the system under development. Every constraint contains a coefficient c_{xy} per component. If a failure f_i of a component C_k is element of a minimum cut set MCS_j, the coefficient c_{xy} is set to 1 in the corresponding constraint. Else the coefficient is set to 0. Thus, if components can only jointly cause the violation of a safety goal, their coefficients are 1 while the remaining coefficients are 0 in the corresponding constraint. If a component can solely cause the violation of a safety goal, its coefficient is the only coefficient that is 1 in the corresponding constraint.

$$\begin{gathered} asil(C_1) \cdot c_{11} + \dots + asil(C_n) \cdot c_{1n} \geq asil(SG_1) \\ \vdots \\ asil(C_1) \cdot c_{p1} + \dots + asil(C_n) \cdot c_{pn} \geq asil(SG_1) \\ asil(C_1) \cdot c_{q1} + \dots + asil(C_n) \cdot c_{qn} \geq asil(SG_2) \\ \vdots \\ asil(C_1) \cdot c_{l1} + \dots + asil(C_n) \cdot c_{ln} \geq asil(SG_m) \end{gathered} \qquad (9)$$

Usually a safety engineer has certain preferences concerning the allocation of safety integrity levels to the components of the system under development. A reason may be the intended reuse of a component C_k developed in an earlier project according to a particular ASIL. Another reason may be the selected limitation of the ASIL allocated to a component in order to limit costs per piece or development costs. Assume $pasil(C_k)$ is the preferred ASIL for a component C_k (see Expression 10).

$$pasil(C_k) \epsilon \{0,1,2,3,4\} \qquad (10)$$

To ensure the consideration of such preferences, an additional constraint on the ASIL of each of the n system components can be defined by the safety engineer (see Expression 11). If a preference for a component C_k is defined, the corresponding coefficient d_{xy} of the component is the only one that is 1.

$$\begin{gathered} asil(C_1) \cdot d_{11} + \dots + asil(C_n) \cdot d_{1n} \leq pasil(C_1) \\ asil(C_1) \cdot d_{21} + \dots + asil(C_n) \cdot d_{2n} \leq pasil(C_2) \\ asil(C_1) \cdot d_{31} + \dots + asil(C_n) \cdot d_{3n} \leq pasil(C_3) \\ \vdots \\ asil(C_1) \cdot d_{n1} + \dots + asil(C_n) \cdot d_{nn} \leq pasil(C_n) \end{gathered} \qquad (11)$$

The constraint solver of the modeling framework can be used to solve the ILP problem. This constraint solver attempts to determine the parameters $asil(C_i)$ that lead to a minimum of the function F_{min} under consideration of the constraints on the parameters $asil(C_i)$ imposed by Expression 9 and Expression 11. If the constraint solver finds a solution, it is (a) optimal with respect to the objective function F_{min}, (b) consistent to results of PHA and FTA and (c) considers the preferences of the safety engineer.

A safety engineer can inspect the solution to the problem. If the solution is satisfactory, the safety engineer can initiate the allocation of the ASILs to the components. In this case the EAST-ADL model is automatically modified according to the accepted solution. If the result is not satisfactory, the safety engineer can decide to change the preferences concerning the ASIL allocation and run the constraint solver again.

If the imposed ILP problem cannot be solved, the safety engineer needs to revise the preferences concerning the ASIL allocation and needs to rerun the constraint solver.

6 EXPERIMENTAL EVALUATION

A plugin for the open source tool Papyrus [12] was created that supports the generation of fault trees as well as the automatic, optimal allocation of safety integrity levels such as described in Section 4 and Section 5. This plugin embeds a constraint solver.

The presented approach was experimentally evaluated using the case study of HEV development. This type of vehicle contains an additional electric motor that supplements the internal combustion engine providing substitutive or additive torque. The safety engineering workflow including ASIL allocation was carried out for a part of the HEV powertrain according to Section 3.

Figure 1 illustrates the GUI of the plugin that was used to carry out ASIL allocation. The grey lines of the table list the safety goals that were defined during PHA and the white lines list the HEV components (column Origin). The ASILs of the safety goals and the preferred component ASILs defined by the safety engineer are listed (column Required ASIL). Furthermore the faults and failures of each component leading to safety goal violations are listed (column Fault/Failure). Finally the constraints on the component ASILs (see also Expressions 9 and 11) that were derived from PHA results, FTA results and preferences of the safety engineer are displayed (column Constraints).

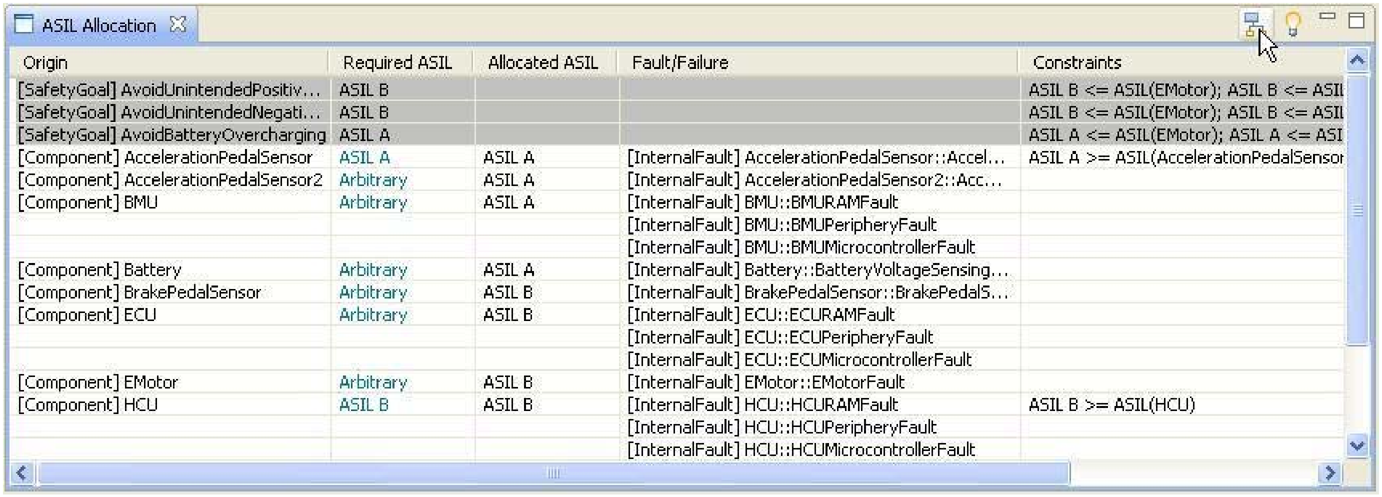
The HEV contains two redundant acceleration pedal sensors. While one sensor measures the pedal position, the other one measures the pedal pressure. Due to the redundancy, the fault or failure of one acceleration pedal can only cause the violation of a safety goal in conjunction with the fault or failure of the other one. This is the prerequisite for ASIL decomposition of these two components. It was decided to use a proven ASIL A pedal position sensor that was used in earlier projects. Therefore it was decided to set the preferred ASIL of the sensor to ASIL A.

The HCU (hybrid control unit) manages the interaction of HEV components such as battery, E-motor, inverter as well as other control units. Therefore the development efforts for a HCU are typically high. Thus it was decided to set the preferred ASIL of the HCU to ASIL A in order to keep development costs low.

After defining preferred ASILs, the constraint solver was activated by clicking on the bulb icon at the upper right (see Figure 1). The constraint solver could not find a solution for the defined constraints. This indicated that the preferred ASILs of the HCU or the pedal sensor were in conflict with the results of PHA and FTA. This made it necessary to modify the preferences.

The preferred ASIL of the HCU was set to ASIL B. Thereafter the constraint solver was activated again. The constraint solver was able to find a solution for the modified constraint set (see column Allocated ASIL in Figure 1). This solution is optimal with respect to the defined objective function (see also Expression 6) and consistent to the defined safety goals, the minimum cut sets extracted during FTA and the defined preferences for the ASIL allocation. Furthermore the constraint solver identified the possibility for ASIL decomposition of the two redundant acceleration pedal sensors under consideration of the defined constraints and decomposed the ASILs of the acceleration pedal sensors to ASIL A+ASIL A.

Thereafter the results were stored by clicking the rectangle icon on the upper right (see Figure 1). This resulted in the automated manipulation of the EAST-ADL model and the allocation of ASILs to the components of the system under development.



Origin	Required ASIL	Allocated ASIL	Fault/Failure	Constraints
[SafetyGoal] AvoidUnintendedPositiv...	ASIL B			ASIL B <= ASIL(EMotor); ASIL B <= ASIL
[SafetyGoal] AvoidUnintendedNegati...	ASIL B			ASIL B <= ASIL(EMotor); ASIL B <= ASIL
[SafetyGoal] AvoidBatteryOvercharging	ASIL A			ASIL A <= ASIL(EMotor); ASIL A <= ASI
[Component] AccelerationPedalSensor	ASIL A	ASIL A	[InternalFault] AccelerationPedalSensor::Accel...	ASIL A >= ASIL(AccelerationPedalSensor
[Component] AccelerationPedalSensor2	Arbitrary	ASIL A	[InternalFault] AccelerationPedalSensor2::Acc...	
[Component] BMU	Arbitrary	ASIL A	[InternalFault] BMU::BMURAMFault	
			[InternalFault] BMU::BMUPeripheryFault	
			[InternalFault] BMU::BMUMicrocontrollerFault	
[Component] Battery	Arbitrary	ASIL A	[InternalFault] Battery::BatteryVoltageSensing...	
[Component] BrakePedalSensor	Arbitrary	ASIL B	[InternalFault] BrakePedalSensor::BrakePedalS...	
[Component] ECU	Arbitrary	ASIL B	[InternalFault] ECU::ECURAMFault	
			[InternalFault] ECU::ECUPeripheryFault	
			[InternalFault] ECU::ECUMicrocontrollerFault	
[Component] EMotor	Arbitrary	ASIL B	[InternalFault] EMotor::EMotorFault	
[Component] HCU	ASIL B	ASIL B	[InternalFault] HCU::HCURAMFault	ASIL B >= ASIL(HCU)
			[InternalFault] HCU::HCUPeripheryFault	
			[InternalFault] HCU::HCUMicrocontrollerFault	

Figure 1: The elaborated ASIL allocation is optimal with respect to a predefined objective function.

ACKNOWLEDGEMENTS

The MEPAS project team wishes to thank the "COMET K2 Forschungsförderungs-Programm" of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economics and Labour (BMWA), Österreichische Forschungsförderungsgesellschaft mbH (FFG), Das Land Steiermark and Steirische Wirtschaftsförderung (SFG) for their financial support.

REFERENCES

1. International Organization for Standardization, “ISO 26262 - Road vehicles - Functional Safety”, 2010.
2. ATESST2 Project Consortium, “EAST-ADL Domain Model Specification”, V2.1, Release Candidate 3, 2010.
3. N. G. Leveson, “Safeware: system safety and computers”, Addison Wesley, 1995.
4. H. Zhang, W. Li and J. Qin, “Model-based Functional Safety Analysis Method for Automotive Embedded System Applications”, Proc. International Conference on Intelligent Control and Information Processing, (Aug.) 2010, pp 761-765.
5. Y. Lee, J. Kim, J. Kim and I. Moon, “A Verification of Fault Tree for Safety Integrity Level Evaluation”, Proc. ICROS-SICE International Joint Conference, (Aug.) 2009, pp 5548-5551.
6. M. Sallak, C. Simon and J.-F. Aubry, “A Fuzzy Probabilistic Approach for Determining Safety Integrity Level”, IEEE Transactions on Fuzzy Systems, vol. 16, (Feb.) 2008, pp 239-248.
7. Y. Papadopoulos, M. Walker, M.-O. Reiser, M. Weber, D. Chen, M. Törngren, D. Servat, A. Abele, F. Stappert, H. Lönn, L. Berntsson, R. Johansson, F. Tagliabo, S. Torchiaro and A. Sandberg, “Automatic Allocation of Safety Integrity Levels”, Proc. 1st Workshop on Critical Automotive applications: Robustness & Safety, (Apr.)

2010, pp 7-10.
8. A. Sandberg, D. Chen, H. Lönn, R. Johansson, L. Feng, M. Törngren, S. Torchiaro, R. T. Kolagari and A. Abele, “Model-Based Safety Engineering of Interdependent Functions in Automotive Vehicles Using EAST-ADL2”, Proc. 29th International Conference on Computer Safety, Reliability and Security, (Sep.) 2010, pp 332-346.
9. R. Mader, G. Grießnig, A. Leitner, C. Kreiner, Q. Bourrouilh, E. Armengaud, C. Steger and R. Weiß, “A Computer-Aided Approach to Preliminary Hazard Analysis for Automotive Embedded Systems”, Proc. IEEE International Conference and Workshops on Engineering of Computer Based Systems, (Apr.) 2011, pp 169-178.
10. R. Mader, E. Armengaud, A. Leitner, C. Kreiner, Q. Bourrouilh, G. Grießnig, C. Steger and R. Weiß, “Computer-Aided PHA, FTA and FMEA for Automotive Embedded Systems”, Proc. 30th International Conference on Computer Safety, Reliability and Security, (Sep.) 2011, pp 113-127.
11. J. P. Ignazio and T. M. Cavalier, “Linear Programming”, Prentice Hall, 1994.
12. A. Lanusse, Y. Tanguy, H. Espinoza, C. Mraidha, S. Gerard, P. Tessier, R. Schnekenburger, H. Dubois and F. Terrier, “Papyrus UML: an open source toolset for MDA.”, Proc. Fifth European Conference on Model-Driven Architecture Foundations and Applications, (Jun.) 2009, pp 1-4.

BIOGRAPHIES

Roland Mader
AVL List GmbH, Graz University of Technology
Graz, Austria

e-mail: roland.mader@avl.com

Roland Mader received the Dipl.-Ing. degree (M.Sc.) with honors in Telematics from Graz University of Technology, Austria. Since 2009 he is employee of AVL List GmbH and a Ph.D. student at Graz University of Technology. His research interests include safety-critical embedded systems as well as methods for their development.

Eric Armengaud
AVL List GmbH, Virtual Vehicle Competence Center
Graz, Austria

e-mail: eric.armengaud@avl.com

Eric Armengaud received the Diplome d'Ingenieur (M.Sc.) in Electrical Engineering in 2002 from the ESIEE Paris. In 2008 he received the Dr. techn. degree (Ph.D.) from Vienna University of Technology. He is employee of AVL List GmbH and the Virtual Vehicle Competence Center. His research interests include distributed real-time systems, testing as well as safety engineering.

Andrea Leitner
Graz University of Technology
Graz, Austria

e-mail: andrea.leitner@tugraz.at

Andrea Leitner received the Dipl.-Ing. degree (M.Sc.) from Graz University of Technology, Austria. Since 2010 she is a Ph.D. student at Graz University of Technology. Her research interests include product line engineering, variant management and knowledge-oriented software engineering.

Christian Steger
Graz University of Technology
Graz, Austria

e-mail: steger@tugraz.at

Christian Steger received the Dipl.-Ing. degree (M.Sc.) in 1990 and the Dr. techn. degree (Ph.D.) in electrical engineering in 1995 from Graz University of Technology. Since 1992 he is Assistant Professor at the Institute for Technical Informatics at Graz University of Technology. His research interests include embedded systems, hardware/software co-design and hardware/software co-verification.

2012 38th Euromicro Conference on Software Engineering and Advanced Applications

A Bridge from System to Software Development for Safety-Critical Automotive Embedded Systems

Roland Mader[1,2], Gerhard Grießnig[1], Eric Armengaud[1,3], Andrea Leitner[2],
Christian Kreiner[2], Quentin Bourrouilh[1], Christian Steger[2], Reinhold Weiß[2]
[1]AVL List GmbH, Austria
[2]Institute for Technical Informatics, Graz University of Technology, Austria
[3]Virtual Vehicle Competence Center, Austria

***Abstract*—In this paper, we present a tool enhancement that allows an effective transition from the system level development phase to the software level development phase of a tool-supported safety engineering workflow aligned with the automotive functional safety standard ISO 26262. The tool enhancement has capabilities for model generation and code generation. Whereas the generation of Simulink models supports the development of application software, the configuration and generation of safety drivers supports the development of the basic software required for initialization, runtime fault detection and error handling. We describe the safety engineering workflow and its supporting tool chain including the tool enhancement. Moreover we demonstrate that the enhancement supports the transition from the system level development phase to the software level development phase using the case study of a hybrid electric vehicle development.**

***Index Terms*—functional safety; automotive embedded system; ISO 26262; EAST-ADL; multi-core microcontroller**

I. Introduction

Powertrain electrification of vehicles transforms powertrains into complex, mechatronic systems. Due to the safety-criticality of the embedded system, sensors and actuators, they are developed according to safety standards like the automotive functional safety standard ISO 26262 [1].

The standard requires the application of a system level development phase. In this phase safety requirements are defined. Furthermore, the system architecture (architecture of embedded system, connected sensors and controlled actuators) is designed and analyzed using FTA (fault tree analysis) and FMEA (failure modes and effects analysis). In addition, safety requirements and ASILs (Automotive Safety Integrity Levels) are allocated to the components of the system architecture. ISO 26262 also requires a software level development phase based on the results of the system level development phase. In this phase software requirements are defined and software units are designed, implemented and integrated under consideration of allocated ASILs.

Tool support for the application of the required phases exists. The EAST-ADL [2] (Electronics Architecture and Software Technology-Architecture Description Language) tool Papyrus for UML [3] (Unified Modeling Language) is useful during the entire system level development phase and the early part of the software level development phase. During the main part of the software level development phase Simulink [4], TargetLink [5] and software development IDEs (integrated development environments) like Code Composer Studio [6] are especially useful.

Whereas EAST-ADL is a domain-specific language for the automotive domain and allows to model a system architecture and related artifacts from different viewpoints and on different levels of abstraction, Papyrus for UML is an open source tool that allows EAST-ADL modeling. Simulink is a software tool that is tightly integrated with Matlab and provides a graphical modeling language allowing behavioral modeling using sinks, sources, linear blocks, nonlinear blocks and connectors. TargetLink is a code generator that allows generating C-programs from Simulink models. Software development IDEs support developing C-programs, compilation, linking and debugging.

However, due to insufficient support for model generation and code generation, an effective transition from the system level development phase to the software level development phase is difficult. To overcome these problems, sophisticated tool support with capabilities for model generation and code generation is required.

The contribution of this work is an enhancement of the EAST-ADL tool Papyrus for UML. The enhancement consists of (a) a Simulink Model Generator and (b) a Safety Driver Generator. Taking an EAST-ADL model as input, the Simulink Model Generator allows the generation of Simulink models required for the development of application software. Taking the same EAST-ADL model as input, the Safety Driver Generator allows the configuration and generation of safety drivers forming a part of the basic software required for initialization, runtime fault detection and error handling of microcontrollers under consideration of ASILs. Both, (a) and (b) support the transition from the system level development phase to the software level development phase of a tool-supported automotive safety engineering workflow aligned with the safety standard ISO 26262.

This work is organized as follows. Section II summarizes related work. Section III describes the automotive safety engineering workflow and its supporting tool chain including the proposed tool enhancement. Section IV describes the Simulink model generation capabilities of the tool enhancement. The capabilities of the proposed enhancement for configuration and generation of safety drivers is described in Section V. Section VI presents the experimental evaluation and Section VII concludes this work.

II. Related Work

An approach to safety engineering using EAST-ADL is presented in [7]. This work is focused on a model-based approach to PHA (Preliminary Hazard Analysis) required by ISO 26262's concept phase. In contrast, our work is focused on providing a more effective transition from the system level development phase to the software level development phase of an automotive safety engineering workflow using the generation of Simulink models and the configuration and generation of safety drivers.

The works in [8], [9] are concerned with component-based software development for safety-critical embedded systems. Whereas [8] is focused on integrating safety analyses with component-based software development, [9] is focused on simulation and model-checking based on a component model. Both do not address the problem of supporting safety-critical software development for initialization, runtime-testing and error-handling of microcontrollers.

The works in [10], [11], [12], [13] describe approaches to embedded software development that include Simulink. They do not focus on safety-critical embedded software development. Thus, they do not address the problem of supporting the development of safety-critical software that allows initialization, runtime-testing and error-handling of microcontrollers.

A survey of techniques for SBST (Software-based Self Testing) can be found in [14]. SBST can be used for runtime-testing of microcontrollers in the field to detect faults in on-chip programmable resources like CPU, RAM, ROM or periphery. This is also one of the purposes of safety drivers. However, the works surveyed in [14] do not present tool support that allows the supported configuration and according generation of this functionality.

In contrast to the works presented in [8], [9], [10], [11], [12], [13], [14], the presented tool enhancement allows to configure and generate safety drivers covering safe initialization, runtime fault detection and error handling of microcontrollers.

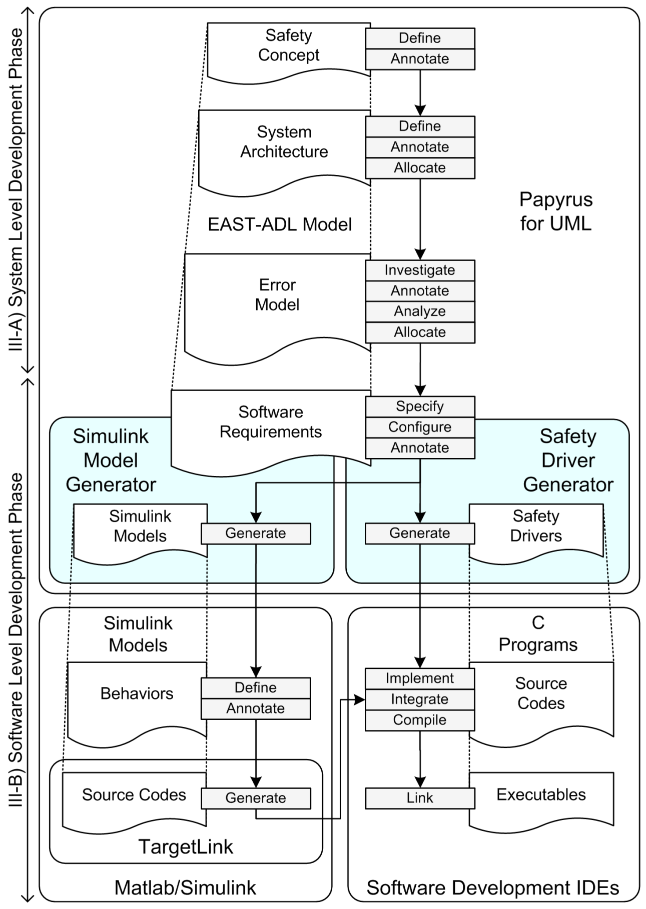


Fig. 1. Safety-Engineering Workflow and Supporting Tools

III. Safety Engineering Workflow

This section describes an automotive safety engineering workflow that is aligned with the safety standard ISO 26262, supported by a tool chain and requires the use of different languages. This workflow can be subdivided into multiple phases and allows iterations. It covers activities required by ISO 26262's system level development phase and its software level development phase (other activities and phases required by ISO 26262 but not covered by the workflow are outside the scope of this paper). The remainder of this section describes the phases of the safety engineering workflow. Figure 1 illustrates the workflow phases, their supporting tools and the languages required for the workflow application.

A. System Level Development Phase

The *Safety Concept* is defined in terms of safety requirements on the automotive embedded system, connected sensors and controlled actuators. Traces are created between related requirements. The *System Architecture* is defined in terms of the embedded system, connected sensors and controlled actuators. Moreover the components of the environment are modeled that interact with the sensors and actuators. Thereafter safety requirements and functions are allocated.

Information flows and energy flows through the system architecture and their environment are investigated. Possible faults and failures are estimated and their propagation is analyzed and annotated in the form of an *Error Model*. Moreover it is investigated and annotated how the failures lead to the malfunctions of the system. Thereafter FTA and FMEA are applied (see also [15]). Finally, ASILs are allocated to the components of the system architecture (see also [16]).

B. Software Level Development Phase

It is defined which components of the system architecture are required to execute software. *Software Requirements* (including software safety requirements) on the application software and the basic software are defined and allocated to these components. Safety drivers are configured under consideration of ASILs, the application and the safety concept for the components of the system architecture required to carry out basic software. Traces are created if a software requirement is derived from any other requirements.

Simulink Models are generated to serve as basis for application software development. They reflect the structure of the components of the system architecture that are required to execute application software. Based on the generated models application software is designed and implemented.

Allocated software requirements are considered. The enhanced and refined Simulink models determine the *Behaviors* of the components that are required to execute application software. Moreover, *Source Codes* are generated from the enhanced and refined Simulink models.

Depending on their configurations, *Safety Drivers* are generated for the components of the system architecture that are required to carry out basic software. These safety drivers determine the random hardware fault detection capabilities and error-handling capabilities of their respective components.

Source Codes are manually implemented. Afterwards, source codes for application software and source codes for basic software like RTOS (real time operating system), legacy code, I/O-related code or generated safety drivers are integrated. This is done by considering allocated basic software requirements for the components of the system architecture required to execute basic software. Finally the integrated software is compiled and linked to form *Executables*.

IV. Simulink Model Generator

Typically, some of the components of the system architecture are required to carry out application software using microcontrollers. The EAST-ADL model describes these components together with their interfaces defined in terms of ports. The safety engineering workflow requires the creation of Simulink models for design and implementation of application software. Such a model shall contain a top-level block with an interface defined in terms of ports as well.

The Simulink Model Generator is capable of generating Simulink models from the EAST-ADL model. These models reflect the structure of the components of the system architecture required to carry out application software in terms of top-level block and interface. The generated Simulink models are refined and enhanced using Simulink's graphical modeling language in the course of the safety engineering workflow. Once enhanced and refined, C-code can be generated from these models. In the following, our approach to Simulink model generation is described.

In the EAST-ADL model, each component of the system architecture is represented by a modeling element C_k. Each component C_k is typed by a type T_i. If a type T_i is required to carry out application software according to the EAST-ADL model, it is a *candidate* for Simulink model generation. Whereas each component C_k has a single type T_i, a type T_i can type multiple components of the system architecture. Thus, if multiple components are typed by the same type T_i, application software needs once to be developed per type T_i but not per component C_k.

The Simulink Model Generator can investigate the EAST-ADL model and identify candidates for Simulink model generation. The safety engineer can choose candidates for Simulink model generation. Assume that M is an EAST-ADL model containing at least one application software component typed by T_i. The Simulink Model Generator $\eta(M, T_i)$ can generate a Simulink model S_{T_i} that is consistent with T_i in terms of (a) top-level block and (b) interface (Expression 1).

$$\eta(M, T_i) \rightarrow S_{T_i} \qquad (1)$$

V. Safety Driver Generator

Typically, some of the components of the system architecture are required to carry out basic software using microcontrollers. This basic software is partly dedicated to (1) safe initialization, (2) runtime-testing and (3) error-handling. Whereas (1) and (2) are required to detect random hardware faults during system operation, (1) and (3) are required to achieve and maintain a safe state upon detecting random hardware faults. The presented tool enhancement contains a Safety Driver Generator that is capable of configuring and generating so-called safety drivers. The term *safety driver* refers to the part of the basic software for a component of the system architecture that is dedicated to (a) safe initialization, (b) runtime detection of random hardware faults and (c) error-handling of its supported microcontroller.

A. Safety Driver Configuration

The Safety Driver Generator allows to configure safety drivers. Furthermore, to aid safety driver configuration, the Safety Driver Generator can suggest safety driver configurations based on the ASILs allocated to the components of the system architecture. In the following, our approach to configure and generate is described.

In the EAST-ADL model, each component of the system architecture is represented by a modeling element C_k. Each component C_k is typed by a type T_i. If a type T_i is required to carry out basic software according to the EAST-ADL model, it is a *candidate* for safety driver generation. Whereas each component C_k has a single type T_i, a type T_i can type multiple components of the system architecture. Thus, if multiple components are typed by the same type T_i, basic software needs once to be developed per type T_i but not per component C_k.

Assume $asil(C_k)$ is the ASIL allocated to a component C_k of the system architecture using our approach to automatic and optimal ASIL allocation [16]. Moreover, assume that $rasil(T_i)$ is the maximum ASIL of all components C_k typed by T_i. We refer to $rasil(T_i)$ as the *relevant ASIL* of T_i (Expression 2).

$$rasil(T_i) = \max_{k=0}^{n} asil(C_k) \qquad (2)$$

The Safety Driver Generator can investigate the EAST-ADL model and identify candidates for safety driver configuration and generation. The Safety Driver Generator presents these candidates together with their relevant ASILs to the safety engineer. The safety engineer can use the Safety Driver Generator to select (a) candidates, (b) the target microcontrollers for the candidates and (c) the software development IDEs (because a safety driver may contain IDE-specific language constructs) depending on the selected target microcontrollers.

In addition, (d) it is possible to configure microcontroller-dependent options for the safety driver generation. These options depend on the microcontroller's specifics and features (e.g. CPU instruction set, memory sizes, control registers or hardware-based self-test capabilities). These options can significantly differ depending on the type of microcontroller.

The Safety Driver Generator can automatically propose configurations of these options depending on the relevant ASILs of the candidates. ASIL-dependent, proposed configurations are predefined based on expert knowledge for all ASILs.

An ASIL-dependent, proposed configuration guides and supports the safety engineer in defining an adapted configuration meeting the needs of the application and the safety concept elaborated in the course of the safety engineering workflow. Although the Safety Driver Generator provides guidance for selecting options, a safety engineer still needs to be aware of the specifics of the selected microcontroller to appropriately adapt a proposed configuration to the needs of the application and the safety concept.

Assume K_{T_i} is a configuration of a safety driver for the microcontroller of a particular candidate T_i. The Safety Driver Generator $\phi(rasil(T_i))$ can propose a configuration K_{T_i} of the safety driver based on the relevant ASIL of T_i (Expression 3).

$$\phi(rasil(T_i)) \rightarrow K_{T_i} \tag{3}$$

A safety engineer can adapt the proposed configuration based on the needs of the application and the safety concept. For example, if the application requires a particular microcontroller module (e.g. a communication peripheral) and the safety concept requires according runtime-testing, the safety engineer can adapt the proposed configuration to enable safe initialization and appropriate runtime-testing of this module. In contrast, if a particular module is not required, the safety engineer can omit the options for the module's safe initialization and runtime-testing to safe computational resources.

Once the safety driver configurations for the candidates are satisfactory, the Safety Driver Generator can automatically project the configurations on the EAST-ADL model, rendering a manual modification redundant. The configurations are projected in (1) a machine-readable representation and in (2) a human-readable representation on the EAST-ADL model. The machine-readable representation is required for subsequent safety driver generation and mainly consists of machine-readable name/value pairs. The human-readable representation describes the configuration of the safety drivers using natural language as software safety requirements. Traces from the safety concept to the generated software safety requirements need to be manually created in accordance with the safety engineering workflow.

Assume TS is the set of all safety driver configurations defined for the candidates (Expression 4).

$$K_{T_i} \epsilon TS \tag{4}$$

Furthermore, assume M is an EAST-ADL model. The Safety Driver Generator $\phi(M, TS)$ can project the configurations of the safety drivers for the candidates onto the EAST-ADL model. This results in an EAST-ADL model M' (Expression 5).

$$\phi(M, TS) \rightarrow M' \tag{5}$$

B. Safety Driver Generation

After the configuration of safety drivers for the candidates and the projection of the configurations onto the EAST-ADL model, the safety engineer can use the Safety Driver Generator to generate the safety drivers from the EAST-ADL model. Assume M' is an EAST-ADL model containing the safety driver configurations TS defined for the candidates. Assume K_{T_i} is a configuration in TS. The Safety Driver Generator $\phi(M', K_{T_i})$ can generate a safety driver D_{T_i} for each configuration (Expression 6).

$$\phi(M', K_{T_i}) \rightarrow D_{T_i} \tag{6}$$

Each generated safety driver contains a header file describing the safety-driver's interface in terms of functions for (1) initialization, (2) runtime-testing and (3) error-handling. This interface is automatically documented during the generation process and describes the safety driver configuration.

A generated safety driver impacts runtime fault detection and error-handling capabilities of a component's microcontroller. This has a direct impact on the safe operation of the components of the system architecture. Thus, to make the presented approach practically applicable, (a) the Safety Driver Generator must either be certified, or (b) a generated safety driver must be verified.

C. Support for an Industrial Multi-Core Microcontroller

To illustrate the concept of configuration and generation of safety drivers, the Safety Driver Generator's support for an industrial lock-step multi-core microcontroller is summarized in this section. This microcontroller is the TMS570LS20216 [6] belonging to the TMS570 family that is based on the lock-step architecture [17]. The TMS570LS20216 offers numerous safety-related features and modules.

To adapt the TMS570LS20216 to the needs of the application and the safety concept and to manage the built-in safety-related features, a multitude of control and status registers is available that can be used by the programmer. The multitude of safety features and configuration possibilities motivates the presented approach that foresees the configuration and generation of safety drivers using the Safety Driver Generator.

Support for configuration and generation of safety drivers was designed with respect to a safety manual [18] for the TMS570LS20216 provided by the vendor. Consequently, generated safety drivers can cover microcontroller modules such as the CCM-R4F (CPU Compare Module for Cortex-RF4) module, the ESM (Error Signaling Module) or the VIM (Vector Interrupt Module) that are commonly used by all typical applications according to the safety manual as well as modules supporting the automotive communication protocol CAN (Controller Area Network).

Depending on the configuration, a generated safety driver for the industrial multi-core processor provides an automatically documented interface describing its configuration. The interface consists of up to eight C-functions that can be used by the application. One of these functions is dedicated to safe initialization, two functions are dedicated to start-up testing, one function is dedicated to periodic online testing and four functions are dedicated to error handling.

VI. Experimental Evaluation

The tool enhancement was implemented as a plugin for Papyrus for UML such as described in the Sections IV and V.

Metric	Value
# of Generated Simulink Model Ports	5
# of Requirements from Safety Driver Configuration	44
# of Lines of Code of Generated Safety Driver	4224

TABLE I
METRICS INDICATING THE VALUE OF THE TOOL ENHANCEMENT.

A tool chain was set up such as described in Section III. Code Composer Studio was chosen as software development IDE. Thereafter the presented approach was experimentally evaluated using the case study of an HEV (hybrid electric vehicle) development.

This type of vehicle contains an electric motor that supplements the internal combustion engine providing substitutive or additive torque. The electric motor is controlled by the automotive embedded system. One of the components of the embedded system is an HCU (hybrid control unit) that manages the interaction of components of the HEV such as battery or E-motor as well as other control units.

The safety engineering workflow (see Section III) was carried out for a part of an HEV powertrain and an EAST-ADL model was created accordingly. It was decided to configure a safety driver for the *HCUType* that defines the behavior of the HCU. A TMS570LS20216 was selected as target microcontroller. On demand the Safety Driver Generator proposed a configuration of the HCU's safety driver depending on the HCU's relevant ASIL (ASIL B). This configuration was manually refined to fit the needs of the *HCUType*. For example, the options for FPU backup before CPU testing (the HCU application requires the FPU), safe CAN initialization (the HCU application requires CAN) and parity startup testing of particular message boxes of the CAN Module 1 (required by the HCU application) were selected. After the configuration, a Simulink model and a Safety Driver were generated for the *HCUType*. The metrics presented in Table I illustrate the value of the tool enhancement. For example, the manual design and implementation of basic software with 4224 lines of codes would have been a time-consuming and error-prone task.

Finally, an HCU demonstrator was created using the generated artifacts. This demonstrator is capable of continuously computing the required E-motor torque of the HCU depending on the SOC (state of charge) of the HEV battery and the pedal positions, while carrying out runtime tests and handling errors.

VII. CONCLUSION

The paper at hand presents a novel tool enhancement. Taking an EAST-ADL as input, this enhancement is capable of generating Simulink models to support application software development as well as configuring and generating safety drivers for initialization, runtime testing and error handling of microcontrollers to support basic software development. Based on the use of metrics, the case study of a hybrid electric vehicle development was used to demonstrate the effectiveness of the approach. The results show that the tool enhancement sustains an effective transition from the system level development phase to the software level development phase of a tool-supported automotive safety engineering workflow aligned with ISO 26262.

ACKNOWLEDGMENT

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

REFERENCES

[1] International Organization for Standardization, "ISO 26262 Road vehicles - Functional safety," 2011.
[2] ATESST2 Project Consortium, "EAST-ADL Domain Model Specification," 2010, version 2.1, Release Candidate 3.
[3] A. Lanusse, Y. Tanguy, H. Espinoza, C. Mraidha, S. Gerard, P. Tessier, R. Schnekenburger, H. Dubois, and F. Terrier, "Papyrus UML: an open source toolset for MDA." in *Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009)*, Jun. 2009, pp. 1–4.
[4] The MathWorks, Inc. (2012, Feb.) Simulink - Simulation und Model-Based Design. [Online]. Available: www.mathworks.de/products/simulink
[5] dSPACE GmbH. (2012, Feb.) TargetLink. [Online]. Available: www.dspace.com/en/inc/home/products/sw/pcgs/targetli.cfm
[6] Texas Instruments Inc. (2012, Feb.) Texas Instruments Website. [Online]. Available: www.ti.com
[7] A. Sandberg, D. Chen, H. Lönn, R. Johansson, L. Feng, M. Törngren, S. Torchiaro, R. T. Kolagari, and A. Abele, "Model-Based Safety Engineering of Interdependent Functions in Automotive Vehicles Using EAST-ADL2," in *Proc. of the 29th International Conference on Computer Safety, Reliability and Security*, Sep. 2010, pp. 332–346.
[8] D. Domis and M. Trapp, "Integrating Safety Analyses and Component-Based Design," in *Proc. of the 27th International Conference on Computer Safety, Reliability and Security*, Sep. 2008, pp. 58–71.
[9] S. Sentilles, A. Pettersson, and I. Crnkovic, "Safe-IDE - A Tool for Design, Analysis and Implementation of Component-Based Embedded Systems," in *Proc. of the 31st IEEE International Conference on Software Engineering (ICSE)*, May 2009, pp. 607–610.
[10] R. Bartosinski, Z. Hanzálek, P. Stružka, and L. Waszniowski, "Integrated Environment for Embedded Control Systems Design," in *Proc. of the IEEE International Symposium on Parallel and Distributed Processing Symposium (IPDPS)*, Jun. 2007, pp. 1–8.
[11] C.-J. Sjöstedt, J. Shi, M. Törngren, D. Servat, D. Chen, V. Ahlsten, and H. Lönn, "Mapping Simulink to UML in the design of embedded systems: Investigating scenarios and transformations," in *Proc. of the OMER 4 Workshop*, Sep. 2008.
[12] T. Farkas, E. Meiseki, C. Neumann, K. Okano, A. Hinnerichs, and S. Kamiya, "Integration of UML with Simulink into embedded software engineering," in *Proc. of the ICROS-SICE International Joint Conference*, Aug. 2009, pp. 474–479.
[13] M. Biehl, C.-J. Sjöstedt, and M. Törngren, "A Modular Tool Integration Approach - Experiences from two Case Studies," in *Proc. of the 3rd Workshop on Model-Driven Tool & Process Integration (MDTPI)*, Jun. 2010, pp. 19–30.
[14] M. Psarakis, D. Gizopoulos, E. Sanchez, and M. Reorda, "Microprocessor Software-Based Self-Testing," *IEEE Design & Test of Computers*, vol. 27, pp. 4–19, 2010.
[15] R. Mader, E. Armengaud, A. Leitner, C. Kreiner, Q. Bourrouilh, G. Grießnig, C. Steger, and R. Weiß, "Computer-Aided PHA, FTA and FMEA for Automotive Embedded Systems," in *Proc. of the International Conference on Computer Safety, Reliability and Security (SafeComp)*, 2011, pp. 113–127.
[16] R. Mader, E. Armengaud, A. Leitner, and C. Steger, "Automatic and Optimal Allocation of Safety Integrity Levels," in *Proc. of the Reliability and Maintainability Symposium (RAMS)*, 2012, pp. 258–263.
[17] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, M. Peri, and S. Pezzini, "Fault-tolerant Platforms for Automotive Safety-Critical Applications," in *Proc. of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES'03)*. ACM, Nov. 2003, pp. 170–177.
[18] Texas Instruments, "Safety Manual - TMS570LS20216S Device," 2010.

OASIS: An Automotive Analysis and Safety Engineering Instrument

Roland Mader[a,b], Eric Armengaud[a,c], Gerhard Grießnig[a], Christian Kreiner[b], Christian Steger[b], Reinhold Weiß[b]

[a]*AVL List GmbH*
[b]*Graz University of Technology*
[c]*Virtual Vehicle Competence Center*

Abstract

In this paper, we present a novel software tool named OASIS (Aut**O**motive **A**nalysis and **S**afety Eng**I**neering In**S**trument). This tool supports the application of a safety engineering workflow aligned with the automotive safety standard ISO 26262. OASIS provides features for safety engineering that allow to create consistent and complete work products and to simplify and automate workflow steps. More precisely, it provides support for (a) model creation and reuse, (b) analysis and documentation and (c) configuration and code generation. We present the safety engineering workflow, OASIS' features and architecture. Moreover, we systematically demonstrate that OASIS is able to strongly support the application of the safety engineering workflow using the case study of hybrid electric vehicle development.

Keywords: ASIL allocation, FMEA, FTA, functional safety, ISO 26262, multi-core microcontroller, PHA, safety driver, system architecture

1. Introduction

Powertrain electrification of vehicles transforms powertrains into complex, mechatronic systems. Due to the safety-criticality of the embedded system, sensors and actuators, they are developed according to safety standards like the automotive functional safety standard ISO 26262 [1].

The standard requires the application of a concept phase. This phase aims at the functional description of a newly developed vehicle, the early identification, assessment and classification of hazards (preliminary hazard analysis) and the according derivation of safety requirements (top-level and

Post-Conference Publication on a Special Issue of the Journal of Reliability Engineering & System Safety (Invited Paper under Review)

functional). The standard also requires a system level development phase depending on the results of the concept phase. In this phase, the system architecture (architecture of embedded system, connected sensors and controlled actuators) is designed and analyzed (using fault tree analysis and failure modes and effects analysis). Furthermore, technical safety requirements are defined. In addition, safety requirements and ASILs (Automotive Safety Integrity Levels) are allocated to the components of the system architecture. ISO 26262 also requires a software level development phase depending on the results of the system level development phase. In this phase software requirements are defined and embedded software units are designed, implemented and integrated.

Tool support for the application of the required phases exists. However, due to lacking support for safety engineering, (1) the concept phase and the system level development phase are difficult to apply. Another difficulty is (2) the effective transition from the system level development phase to the software level development phase. Both hinder the efficient creation of a complete and consistent set of work products. To overcome these problems, sophisticated tool support is required that provides features for safety engineering allowing to create consistent and complete work products and to simplify and automate workflow steps.

The contribution of this paper is a novel tool named OASIS (Aut**O**motive **A**nalysis and **S**afety Eng**I**neering In**S**trument). OASIS supports a safety engineering workflow that is aligned with ISO 26262's concept phase, system level development phase and software level development phase. OASIS' features allow (a) model creation and reuse (e.g. by checking an evolving model for properties), (b) analysis and documentation (e.g. by generating fault trees and extracting minimum cut sets) and (c) configuration and code generation (e.g. by allowing configuration and generation of safety-critical embedded software). These features allow to create consistent and complete work products and to simplify and automate workflow steps.

The paper is organized as follows. Section 2 describes the safety engineering workflow. Section 3 summarizes the state of the art and shows how OASIS is different from existing approaches. Section 4 describes the tool chain that is used to apply the safety engineering workflow. Section 5 describes OASIS' support for model creation and reuse. Section 6 describes OASIS' support for analysis and documentation. Section 7 describes OASIS' support for configuration and code generation. Section 8 describes the experimental evaluation of our approach and Section 9 concludes the paper.

2. Safety Engineering Workflow

This section describes a safety engineering workflow (see Figure 1) aligned with ISO 26262. This workflow can be subdivided into multiple phases and allows iterations. It covers activities required by ISO 26262's concept phase, its system level development phase and its software level development phase (other activities and phases required by ISO 26262 but not covered by the workflow are outside the scope of this paper). The remainder of this section describes the phases of the safety engineering workflow.

2.1. Concept Phase

The concept phase of the safety engineering workflow foresees the functional description of the vehicle under development and the application of PHA [2] (preliminary hazard analysis). PHA is an analysis technique that is qualitatively applied early in the development process by a team of people with a wide variety of expert knowledge and skills. The application of PHA aims at the identification, classification and assessment of potential hazards of a newly developed vehicle. In addition, the derivation of top-level safety requirements and functional safety requirements is required.

2.1.1. Definition of the Analysis Subject

Information about the vehicle under development is collected and modeled. Functions of the vehicle (e.g. motoring or recuperative braking) are defined. Requirements on the functions are determined and allocated (e.g. conditions for activation or deactivation). In addition, relevant modes (e.g. drive, creep or acceleration) are identified for each function and associated with the requirements.

2.1.2. Identification of Hazards and Hazardous Events

Based on the definition of the analysis subject, possible malfunctions are identified. Hazards are derived for each malfunction (e.g. unintended acceleration of the vehicle). Thereafter, operational situations such as traffic situations (e.g. oncoming traffic on a highway in a curve) and maintenance situations (e.g. vehicle at lifting ramp) are defined. Moreover, use cases describing the behavior (e.g. overtaking or changing oil) of the related actors (e.g. driver or mechanic) are described. Hazardous events are determined for relevant combinations of hazards, use cases and operational situations. A library of relevant use cases and operational situations from earlier workflow applications is used as input. If necessary, this library is updated for

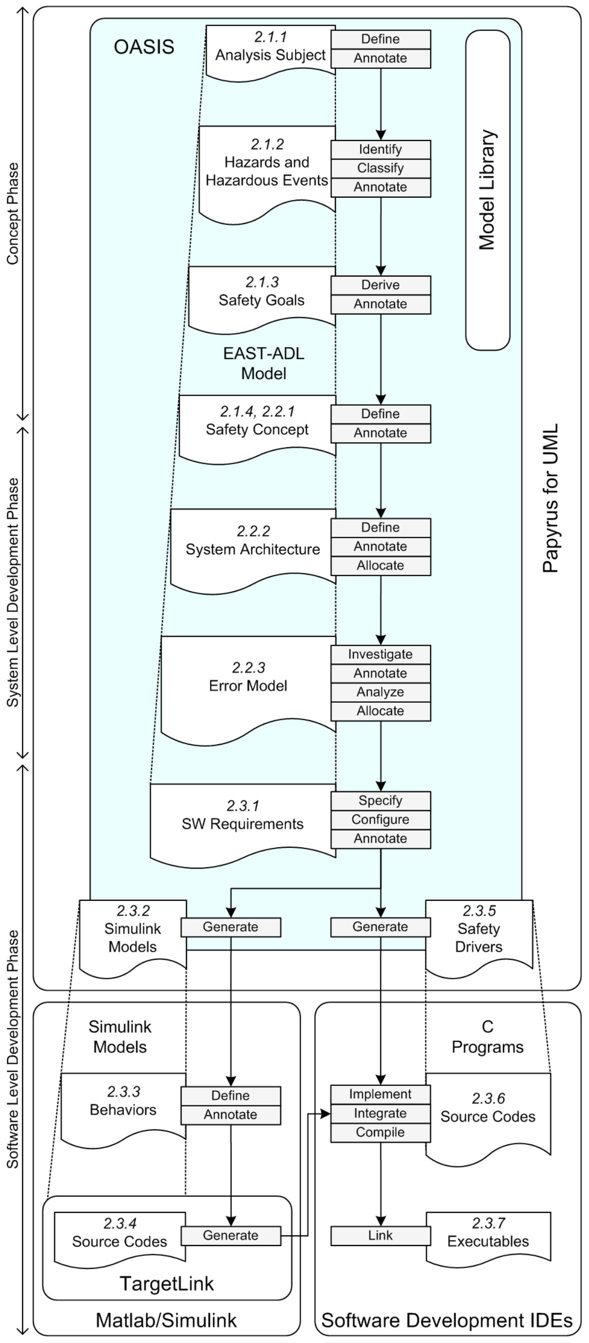


Figure 1: The safety engineering workflow is aligned with life cycle phases of ISO 26262 and supported by a tool chain. The tools support the languages required for the annotation of the work products of the workflow.

future workflow applications. Relevant modes are identified for each hazardous event. The criticality of each hazardous event is assessed in terms of controllability, severity and exposure, and an ASIL is determined.

2.1.3. Derivation of Safety Goals

For each hazardous event classified as ASIL A, ASIL B, ASIL C or ASIL D, a safety goal is derived and associated. Furthermore, a safe state is defined (e.g. switch open) for each safety goal. Alternatively, a safe mode (e.g. limp home mode) is determined. The determined safety goals are top-level safety requirements.

2.1.4. Definition of Functional Safety Concept

The functional safety concept is derived from the safety goals. The functional safety concept consists of functional safety requirements on the automotive embedded system, connected sensors and controlled actuators. Traces are created between safety goals and derived functional safety requirements as well as among functional safety requirements, if necessary.

2.2. System Level Development Phase

The system level development phase focuses on the definition of the system architecture, the allocation of requirements and the application of FTA [2] (fault tree analysis) and FMEA [2] (failure modes and effects analysis). Whereas FTA is a deductive analysis technique, FMEA is an inductive analysis technique. The techniques complement each other. Their qualitative application is especially useful in early development phases when less quantitative information about the vehicle, its embedded system, its sensors and actuators is available. In line with results of analyses such as PHA and FTA, an allocation of ASILs to the components of the system architecture must be carried out. Once allocated, they determine applicable requirements of ISO 26262 and the necessary safety measures of the system components to avoid unreasonable residual risk.

2.2.1. Definition of Technical Safety Concept

The technical safety concept is derived from the functional safety concept. The technical safety concept consists of technical safety requirements. Traces are created between functional safety requirements and derived technical safety requirements as well as among technical safety requirements, if necessary.

2.2.2. Definition of System Architecture

The system architecture is defined in terms of the embedded system, connected sensors and controlled actuators. Moreover, the components of the environment interacting with the sensors and actuators are modeled. Thereafter, functional and technical safety requirements and functions are allocated to the components of the system architecture.

2.2.3. Investigation and Annotation of Faults and Failures

Information flows and energy flows through the system architecture and its environment are investigated. Possible faults and failures are estimated, and their propagation is analyzed and annotated. The estimated faults can either be random hardware faults or potential process faults that are introduced during the development process. Moreover, it is investigated and annotated how the failures lead to the malfunctions of the system. The system architecture is analyzed and verified using qualitative FTA and FMEA. Finally, ASILs are allocated to the components of the system architecture.

2.3. Software Level Development Phase

The software level development phase is focused on the derivation of software requirements and the according implementation, generation and integration of software. Developed software must be divided into application software and basic software designed and implemented in different manners. Whereas application software determines the behavior of the vehicle functions, basic software fulfills tasks like hardware abstraction, initialization, communication, detection of random hardware faults and error-handling. We refer to the part of the basic software for a component of the system architecture that is dedicated to safe initialization, runtime fault detection and error-handling as safety driver. Thereafter, the sub phases of the concept phase, the system level development phase and the software level development phase are described.

2.3.1. Specification of Embedded Software

It is defined which components of the system architecture are required to execute software. Requirements (including safety requirements) on the application software and the basic software are defined and allocated to these components. Safety drivers are configured for the components of the system architecture that are required to execute basic software. Traceability links are created if a software requirement is derived from any other requirements (e.g.

from the functional or technical safety concept). A report is generated that lists software requirements, describes their allocation and the work products of the preceding workflow phases.

2.3.2. Generation of Simulink Models

Simulink models are generated to serve as basis for application software development. They reflect the structure of the components of the system architecture that are required to execute application software.

2.3.3. Definition of Behaviors

Application software is designed and implemented based on the generated Simulink models for the components of the system architecture. Allocated software requirements are considered. The enhanced and refined Simulink models determine the behavior of the components that are required to execute application software.

2.3.4. Generation of Source Codes

Program code is generated from the enhanced and refined Simulink models for the components of the system architecture that are required to execute the application software.

2.3.5. Generation of Safety Drivers

Based on their configurations, safety drivers are generated for the components of the system architecture that are required to execute basic software. These safety drivers determine the random hardware fault detection capabilities and error-handling capabilities of the components of the system architecture that are required to execute basic software.

2.3.6. Implementation and Integration of Source Codes

Program code is manually implemented. Afterwards, program code for application software and code for basic software like RTOS (real time operating system), legacy code, I/O-related code or generated safety drivers is integrated. This is done by considering allocated basic software requirements for the components of the system architecture that are required to execute basic software.

2.3.7. Compilation and Linking of Executable

The created software is compiled and linked to form executables for the components of the system architecture that are required to execute software.

3. State of the Art

This section summarizes the state of the art with respect to techniques the are relevant to the application of the safety engineering workflow. We compare these techniques to OASIS' support for the workflow application and show how OASIS contributes to the field.

3.1. Concept Phase

The concept phase requires identifying, assessing and classifying hazards based on a functional vehicle description and the according derivation of safety requirements to mitigate and control the hazards.

The works in [3, 4, 5, 6] focus on defining systematic approaches supporting the process of identifying and classifying hazards and defining means to mitigate or control them. Although they refer to the use of models they do not explicitly foresee the use of diagrammatic languages.

In contrast, [7, 8, 9] present approaches requiring the use of diagrammatic languages in the context of tool support. However, these approaches do not allow the automatic identification of imperfections.

Approaches that go one step further and foresee the use of more sophisticated tool support are described in [10, 11]. These approaches make use of diagrammatic languages and, furthermore, support the automatic identification of imperfections. However, they do not allow the automatic correction of the model. In contrast, OASIS supports the automatic checking of properties and the automatic proposition and application of corrective measures based on using the diagrammatic language EAST-ADL [12] (Electronics Architecture and Software Technology-Architecture Description Language).

3.2. System Level Development Phase

The definition of a system architecture, the application of FTA and FMEA as well as the allocation of ASILs are important aspects of the system level development phase.

The approaches explained in [13, 14, 15, 16, 17, 18, 19] use models that describe the structure of a system. These models are complemented with safety-relevant information (typically about faults and failures and their propagation). The underlying models are used by all approaches as input to fault tree generation and/or for FMEA table generation, supporting the application of FTA and/or FMEA. However, these approaches do not support the elaboration of the underlying model. In contrast, OASIS supports fault tree generation and FMEA table generation and, furthermore, the elaboration of the underlying model using property checking and model correction.

The works presented in [20, 21] address the problem of allocation of safety-relevant levels to the components of a system architecture under consideration of safety and costs. The approach presented in [21] focuses on the aerospace domain and proposes a method for the allocation of DALs (Design Assurance Levels) to the components of an avionic system architecture. The approach presented in [20] focuses on the automotive domain, but potentially leads to lots of possible ASIL (Automotive Safety Integrity Level) allocations that must be manually investigated by the safety engineer. In contrast, OASIS supports an approach to ASIL allocation leading to a single allocation that is optimal with respect to an objective function and considers preferences of the safety engineer that were defined in advance.

3.3. Software Level Development Phase

An important aspect of the software level development phase is the design, implementation and integration of embedded software. The works in [16, 22] are concerned with component-based software development for safety-critical embedded systems. Whereas [16] is focused on integrating safety analyses with component-based software development, [22] is focused on simulation and model-checking based on a component model. Both do not address the problem of supporting safety-critical software development for initialization, runtime-testing and error-handling of microcontrollers. In contrast, OASIS allows to configure and generate safety drivers covering these functionalities.

The works in [23, 24, 25, 26] describe approaches to embedded software development that include Simulink. They do not focus on safety-critical embedded software development. Thus, they do not address the problem of supporting the development of safety-critical software that allows initialization, runtime-testing and error-handling of microcontrollers. In contrast, OASIS supports the development of safety-critical software by configuring and generating safety drivers.

A survey of techniques for SBST (Software-based Self Testing) can be found in [27]. SBST can be used for runtime-testing of microcontrollers in the field to detect faults in on-chip programmable resources like CPU, RAM, ROM or periphery. This is also one of the purposes of safety drivers. However, the works surveyed in [27] do not present tool support that allows the supported configuration and according generation of this functionality. In contrast OASIS supports the configuration and generation of safety drivers.

4. Tool Chain

The safety engineering workflow that is described in Section 2 is supported by a tool chain. The tools of the tool chain support different languages required for the annotation of the work products of the workflow. Figure 1 maps the phases of the safety engineering workflow on the tools and languages necessary for the workflow application. The remainder of this section describes the tools, languages and the mapping.

4.1. Papyrus for UML

Papyrus for UML [10] is an Eclipse-based open-source tool that allows UML (Unified Modeling Language) modeling as well as the definition of UML profiles. An open-source plugin is available that allows to create EAST-ADL models. EAST-ADL is a domain-specific language and adapted to the needs of the automotive domain. It is *diagrammatic* [28] such as UML. It consists of syntactic elements such as boxes, ovals, lines or arrows. Its *abstract syntax* is defined by its meta model and its *semantic domain* and *semantic mapping* are defined using natural language [28]. EAST-ADL allows (a) to describe a system architecture from different viewpoints and on different levels of abstraction, (b) to express work products and artifacts required by ISO 26262, (c) to describe relations and dependencies and (d) to structure the resulting information. Papyrus for UML is used to annotate a central EAST-ADL model in the course of the safety engineering workflow phases that are defined in the Sections 2.1, 2.2 and 2.3.1. This central EAST-ADL model reflects all the work products created in the course of the aforementioned workflow phases in an organized and structured manner.

4.2. OASIS

OASIS is a plugin for the tool Papyrus for UML. Its relations to other tools of the tool chain are illustrated in Figure 2. OASIS exploits the central EAST-ADL model (contains the work products in an organized and structured manner) and the EAST-ADL meta model (defines rules for the structure of the information) in order to support the application of the safety engineering workflow. The support is thereafter described.

- *Support for Model Creation and Reuse* is provided by the modules described in Section 5. They support creation and maintenance of a complete and consistent EAST-ADL model and are especially useful

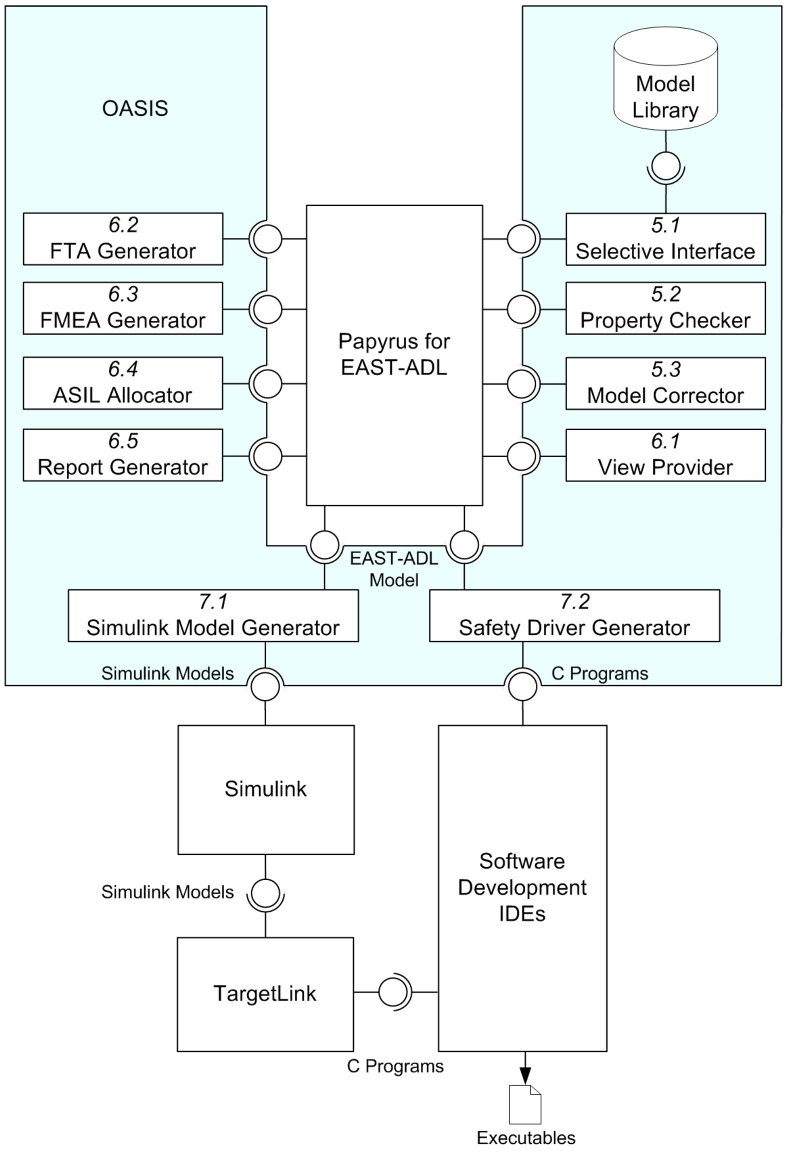


Figure 2: The tool OASIS is part of a tool chain that is required for the application of the safety engineering workflow. OASIS supports the application of the safety engineering workflow by providing (5.x) support for model creation and reuse, (6.x) support for analysis and documentation and (7.x) support for configuration and code generation.

during the workflow phases described in the Sections 2.1, 2.2 and 2.3.1. This is achieved by providing (a) systematic reuse of modeling elements, (b) automatic checking of the model and (c) automatic suggestion and application of corrective measures.

- *Support for Analysis and Documentation* is provided by the modules described in Section 6. Taking the EAST-ADL model as input, they support continuous analyses, communication and reviews and are especially useful during the workflow phases described in the Sections 2.1, 2.2 and 2.3.1. This is achieved by providing (a) automatic extraction of views, (b) automatic generation of fault trees, (c) automatic generation of FMEA tables, (d) automatic and optimal ASIL allocation and (e) automatic report generation.

- *Support for Configuration and Code Generation* is provided by the modules that are described in Section 7. They support design and implementation of software in consistency with the EAST-ADL model and are especially useful during the workflow phases described in the Sections 2.3.2 and 2.3.5. This is achieved by providing (a) computer-aided configuration and automatic generation of safety drivers and (b) automatic generation of Simulink models.

4.3. Simulink

Simulink [29] is tightly integrated with Matlab and is a development environment for multidomain simulation and model-based design of dynamic systems. A graphical modeling language is available containing sinks, sources, linear blocks, nonlinear blocks and connectors. Add-ons for Simulink can be built that allow to extend Simulink to other model domains by providing blocksets. Simulink is used for behavioral modeling in the course of the workflow phase defined in Section 2.3.3. The resulting models determine the behavior of the system components that are required to execute application software.

4.4. TargetLink

TargetLink [30] is an add-on to Simulink supporting a specialized blockset for code generation. TargetLink allows the generation of C-code from models created using this blockset. C is a general-purpose programming language that is frequently used for embedded software development. TargetLink allows to design control algorithms and to generate production code for every microcontroller that is supported by a compiler and a linker. TargetLink is used in the workflow phase defined in Section 2.3.4 to generate C-code.

4.5. Software Development IDEs

Software development IDEs (integrated development environments) support developing C-programs, compilation, linking and debugging. Examples of embedded software development IDEs are Code Composer Studio [31], Code Warrior [32] or μVision [33]. Typically, IDEs are provided by the vendors of microcontrollers (e.g. Code Composer Studio or Code Warrior) or provided by third parties (e.g. μVision). Software development IDEs are used in the workflow phases described in the Sections 2.3.6 and 2.3.7.

5. Support for Model Creation and Reuse

The application of the safety engineering workflow requires (a) the capability to describe a system and related artifacts using the domain-specific language EAST-ADL, (b) the capability to assess the quality of the resulting model with respect to consistency and completeness and (c) the capability to consistently maintain the model depending on discussions, reviews, analyses and changes. These activities are typically performed by the safety engineer. OASIS provides the following modules to support the safety engineer:

- The *Selective Interface* (see Section 5.1) to a Model Library allows to systematically reuse particular modeling elements (e.g. use cases). This supports the model creation and improves the model quality thanks to the reuse of mature and complete information.
- The *Property Checker* (see Section 5.2) can automatically check the model for properties that indicate the quality of the resulting model. This allows the continuous assessment of the model with respect to completeness and unambiguity.
- The *Model Corrector* (see Section 5.3) can take corrective actions if the model is incomplete or ambiguous. Thus, the creation and the maintenance of a consistent and complete model is supported.

5.1. Selective Interface

The safety engineering workflow foresees the application of PHA in the concept phase. PHA requires the identification, classification and assessment of potential hazards of the vehicle under development in the context of potential operational situations and potential use cases. In the presented approach, the results of PHA are annotated using EAST-ADL. The language allows to express concepts that are relevant to PHA such as operational situations, use cases, malfunctions, hazards, hazardous events and safety goals.

There are numerous operational situations that are likely to occur in everyday life. Passenger vehicles are frequently exposed to these situations regardless of their vehicle features, type or technology (e.g. oncoming traffic on a highway in a curve). Also driver behavior that is described in terms of use cases is frequently recurring (e.g. braking, coasting, accelerating or gear switching). Thus, operational situations and use cases are well suited to be reused from workflow application to workflow application.

In contrast, malfunctions, hazards, hazardous events and safety goals are specific to the vehicle under development (e.g. the hazard of unintended movement of a vehicle is typically differently classified for HEVs and traditional vehicles). Thus, their reuse from workflow application to workflow application can mislead the safety engineer. Consequently, there is a risk that potential malfunctions and hazards of a vehicle are inappropriately identified, classified and assessed.

OASIS contains a Selective Interface to a Model Library (see Figure 2). This Selective Interface allows to export operational situations and use cases (including actors, extensions points and respective relations) to an external Model Library. The Selective Interface omits the export of other modeling elements such as malfunctions, hazards, hazardous events and safety goals. Once exported, operational situations and use cases are stored in the Model Library. Stored operational situations and use cases can be imported from the Model Library and used as starting point for new PHA applications. The reuse of operational situations and use cases supports the creation of the model and allows the safety engineer to carry out PHA more systematically and efficiently.

Assume that M is the EAST-ADL model that is elaborated in course of PHA. Furthermore, assume that e is an OperationalSituation that is contained in M (precondition) and shall be stored for later reuse (Expression 1).

$$e \epsilon M \tag{1}$$

OperationalSituation e can be exported to L using the Selective Interface χ. χ updates the Model Library L to Model Library L' accordingly (Expression 2).

$$\chi(L, e) \rightarrow L' \tag{2}$$

After the export process (postcondition), the updated Model Library L' contains a copy e' of e (Expression 3).

$$e \epsilon M \wedge e' \epsilon L' \tag{3}$$

Assume that S is a subset of M that exclusively consists of UseCases, Actors, ExtensionPoints, Include relations and Extend relations that shall be stored to be reused for future PHA applications (precondition) (Expression 4).

$$S \subseteq M \tag{4}$$

Assume that L is the Model Library. S can be exported to L using the Selective Interface χ. χ updates the Model Library L to Model Library L' accordingly (Expression 5).

$$\chi(L, S) \rightarrow L' \tag{5}$$

Thereafter (postcondition), a copy S' of S is contained in L' (Expression 6).

$$S \subseteq M \wedge S' \subseteq L' \tag{6}$$

Whenever a new EAST-ADL model is required for another application of the safety engineering workflow, the Selective Interface χ can be used to import the content of the Model Library L' to create a new model M' that comprises modeling elements that can be reused in the context of PHA (Expression 7).

$$\chi(L') \rightarrow M' \tag{7}$$

ID	Meta Class	Property Definition
12a	HazardousEvent	At least one SafetyGoal is associated if ASIL greater than QM

Table 1: The Property Checker supports 71 properties. An example property for the meta class HazardousEvent is presented.

5.2. Property Checker

Contemporary vehicles, their embedded systems, sensors and actuators are complex. This complexity results in a large set of information that needs to be managed during the application of the safety engineering workflow. The workflow application and the complete and consistent projection of its results on the EAST-ADL model are challenging, cumbersome and error-prone.

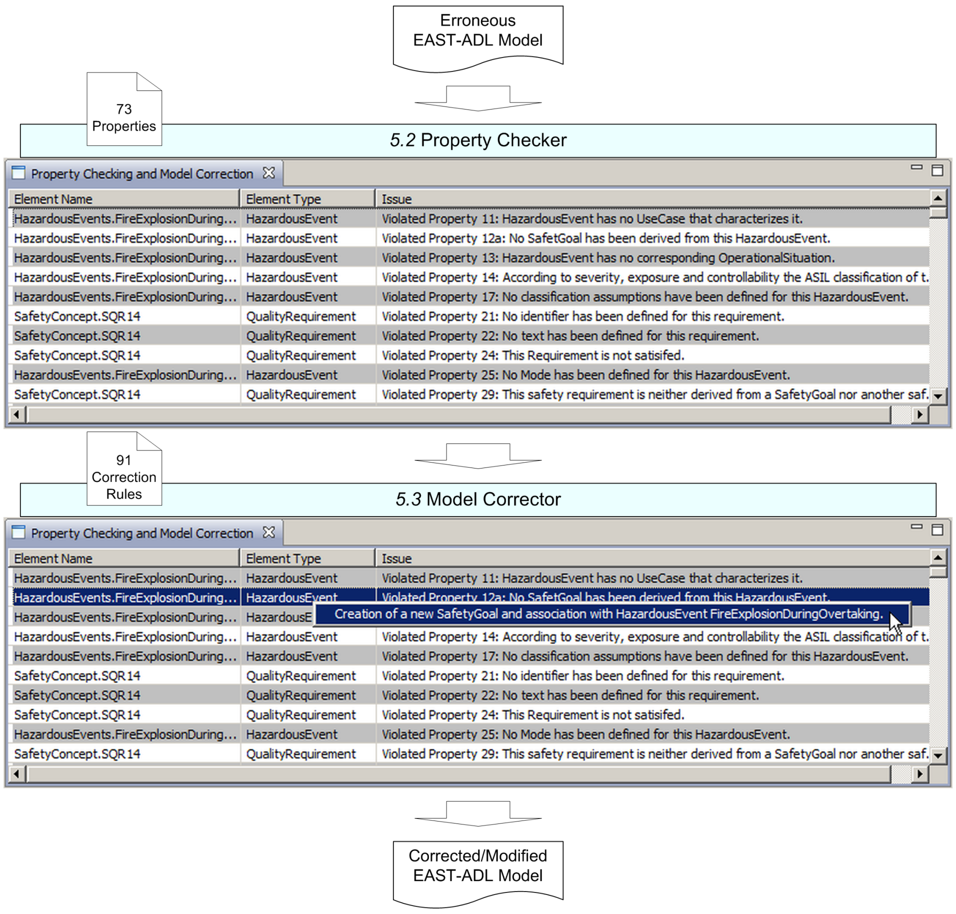
To address this problem, we propose to automatically check the evolving EAST-ADL model for properties during the application of the workflow. The fulfillment of such properties shall *indicate* the correct application of the safety engineering workflow. The violation of such properties shall unveil the erroneous application of the safety engineering workflow.

OASIS contains a Property Checker (see Figure 2) that supports 71 properties based on the properties defined in [34, 35]. Input to their definition were (1) the specification of the domain-specific language EAST-ADL and (2) the safety standard ISO 26262 [1]. An example for a property is presented in Table 1. Column *Meta Class* denotes the meta class of the EAST-ADL language that can violate the corresponding property. Column *Property Definition* defines the property for the corresponding meta class in natural language.

The Property Checker continuously checks the evolving model. It automatically presents violating modeling elements to the safety engineer (see Figure 3). This information allows to early identify inadequately performed workflow steps before their results can affect subsequent workflow steps. The Property Checker does not only allow the identification of errors, it is also a valuable guide during the application of the safety engineering workflow.

Assume M is an EAST-ADL model, M_{MM} is the EAST-ADL meta model, and P is the set of properties an EAST-ADL model is expected to hold. Assume e is a modeling element of the EAST-ADL model, t is a type defined by the EAST-ADL meta model, and p is a property (Expression 8).

$$e \epsilon M, t \epsilon M_{MM}, p \epsilon P \tag{8}$$



Element Name	Element Type	Issue
HazardousEvents.FireExplosionDuring...	HazardousEvent	Violated Property 11: HazardousEvent has no UseCase that characterizes it.
HazardousEvents.FireExplosionDuring...	HazardousEvent	Violated Property 12a: No SafetGoal has been derived from this HazardousEvent.
HazardousEvents.FireExplosionDuring...	HazardousEvent	Violated Property 13: HazardousEvent has no corresponding OperationalSituation.
HazardousEvents.FireExplosionDuring...	HazardousEvent	Violated Property 14: According to severity, exposure and controllability the ASIL classification of t.
HazardousEvents.FireExplosionDuring...	HazardousEvent	Violated Property 17: No classification assumptions have been defined for this HazardousEvent.
SafetyConcept.SQR14	QualityRequirement	Violated Property 21: No identifier has been defined for this requirement.
SafetyConcept.SQR14	QualityRequirement	Violated Property 22: No text has been defined for this requirement.
SafetyConcept.SQR14	QualityRequirement	Violated Property 24: This Requirement is not satisifed.
HazardousEvents.FireExplosionDuring...	HazardousEvent	Violated Property 25: No Mode has been defined for this HazardousEvent.
SafetyConcept.SQR14	QualityRequirement	Violated Property 29: This safety requirement is neither derived from a SafetyGoal nor another saf.

Element Name	Element Type	Issue
HazardousEvents.FireExplosionDuring...	HazardousEvent	Violated Property 11: HazardousEvent has no UseCase that characterizes it.
HazardousEvents.FireExplosionDuring...	HazardousEvent	Violated Property 12a: No SafetGoal has been derived from this HazardousEvent.
HazardousEvents.FireExplosionDuring...	HazardousE	
HazardousEvents.FireExplosionDuring...	HazardousEvent	Violated Property 14: According to severity, exposure and controllability the ASIL classification of t.
HazardousEvents.FireExplosionDuring...	HazardousEvent	Violated Property 17: No classification assumptions have been defined for this HazardousEvent.
SafetyConcept.SQR14	QualityRequirement	Violated Property 21: No identifier has been defined for this requirement.
SafetyConcept.SQR14	QualityRequirement	Violated Property 22: No text has been defined for this requirement.
SafetyConcept.SQR14	QualityRequirement	Violated Property 24: This Requirement is not satisifed.
HazardousEvents.FireExplosionDuring...	HazardousEvent	Violated Property 25: No Mode has been defined for this HazardousEvent.
SafetyConcept.SQR14	QualityRequirement	Violated Property 29: This safety requirement is neither derived from a SafetyGoal nor another saf.

Creation of a new SafetyGoal and association with HazardousEvent FireExplosionDuringOvertaking.

Figure 3: Violated properties are automatically identified and possible solutions to problems are suggested on demand.

Moreover, $I(e,t)$ pertains if e is of type t. $D(t,p)$ pertains if p is defined for t and $H(e,p)$ pertains if p holds for e. If M *indicates* the correct application of the workflow, Expression 9 is valid. In this case, no modeling elements violate properties.

$$\neg\exists e \neg\exists t \neg\exists p (I(e,t) \wedge D(t,p) \wedge \neg H(e,p)) \tag{9}$$

If a model M *shows the erroneous application* of the workflow, Expression 10 is valid. In this case, at least one modeling element violates a property.

$$\exists e \exists t \exists p (I(e,t) \wedge D(t,p) \wedge \neg H(e,p)) \tag{10}$$

ID	Meta Class	Suggested Solution
12a	HazardousEvent	Creation and association of SafetyGoal
12a	HazardousEvent	Associate one of the SafetyGoals without HazardousEvent

Table 2: The Model Corrector supports 91 correction rules. Two example correction rules for the meta class HazardousEvent are presented.

5.3. Model Corrector

If violated properties unveil an erroneous application of the safety engineering workflow, the EAST-ADL model needs to be corrected accordingly. To ease the correction of errors, we propose to support the identification and application of proper correction measures.

OASIS contains a Model Corrector (see Figure 2) that supports 91 correction rules based on the correction rules described in [34, 35]. Input to their definition were again ISO 26262 and EAST-ADL. Two example correction rules are presented in Table 2. Column *Meta Class* denotes the meta class of the enhanced EAST-ADL language that can be subject to the suggestion of an automated correction. Column *Suggested Solution* defines the possible suggestion for the corresponding meta class in natural language.

The Model Corrector can automatically suggest possible solutions for problems unveiled by the Property Checker. On demand, it identifies and proposes possible solutions depending on the affected modeling element, based on the current EAST-ADL model and the violated property (see Figure 3). If the safety engineer decides to accept a solution, the model is automatically modified accordingly. Manual modifications are superfluous.

Assume M is an EAST-ADL model, M_{MM} is the meta model of the EAST-ADL language, P is the set of properties an EAST-ADL model is expected to hold, and S is the set of suggestions supported by the Model Corrector. Assume e_1 is a modeling element of the EAST-ADL model, t_1 is a type defined by the meta model, p_1 is a property, and s_1 is a suggestion (Expression 11).

$$e_1 \epsilon M, t_1 \epsilon M_{MM}, p_1 \epsilon P, s_1 \epsilon S \tag{11}$$

Assume that before an automated model correction is carried out (precondition), e_1 is of type t_1 and violates p_1 that is defined for type t_1 (Expression 12).

$$I(e_1, t_1) \wedge D(t_1, p_1) \wedge \neg H(e_1, p_1) \tag{12}$$

If the safety engineer accepts suggestion s_1, the EAST-ADL model M is automatically corrected and transformed to EAST-ADL model M' by the Model Corrector $\gamma(M, e_1, t_1, p_1, s_1)$ depending on M, e_1, t_1, p_1 and s_1 (Expression 13).

$$\gamma(M, e_1, t_1, p_1, s_1) \rightarrow M' \tag{13}$$

After the modification (postcondition), e_1 is an element of M', still e_1 is of type t_1 and does not violate p_1 any more (Expression 14).

$$e_1 \epsilon M', I(e_1, t_1) \wedge D(t_1, p_1) \wedge H(e_1, p_1) \tag{14}$$

6. Support for Analysis and Documentation

The application of the safety engineering workflow requires (a) continuous analyses as well as (b) communication of safety engineers, domain experts and other stakeholders based on the contents of the model. Languages like EAST-ADL are complex, and the evolving model is difficult to assess. Thus, it is important to automatically highlight specific aspects of the model, and to generate documents to support activities like analyzing, discussing and reviewing. To support these activities, OASIS provides the following modules:

- The *View Provider* (see Section 6.1) allows the automated extraction of model information (e.g. PHA results) that is relevant to particular aspects or phases of the safety engineering workflow. This enables focused discussions or reviews and improves communication efficiency.
- The *FTA Generator* (see Section 6.2) allows the automated generation of fault trees from the model and the according extraction of minimum cut sets. This supports the application of FTA and the assessment of the system architecture's robustness against faults. Furthermore, it sustains discussing and reviewing.
- The *FMEA Generator* (see Section 6.3) allows the consistent generation of FMEA tables based on information contained in the model. This supports the application of FMEA. Moreover a generated FMEA table is valuable for discussions and reviews.

- The *ASIL Allocator* (see Section 6.4) allows inspecting the constraints on the allocation of ASILs to the components of the system architecture based on PHA and FTA results. Furthermore, it allows the automatic allocation of ASILs to the components of the system architecture. This makes a manual elaboration superfluous.

- The *Report Generator* (see Section 6.5) permits the automated generation of text-based reports from the model. The resulting documentation of the work products of the safety engineering workflow allows reviews and discussions.

6.1. View Provider

In the course of the safety engineering workflow, the EAST-ADL model grows large in size and reflects an increasing number of work products. Because of the complexity, it is increasingly difficult to examine particular work products or particular aspects of the model. Thus, we propose supporting the safety engineer by providing tabular views on work products and particular aspects of the EAST-ADL model.

OASIS contains a View Provider (see Figure 2). This View Provider can automatically extract information from the model that is especially relevant to particular aspects or workflow phases and presents it in tables. These tables supplement the GUIs that are provided by other OASIS modules like the Property Checker, the Model Corrector, the FTA Generator, the FMEA Generator, the ASIL Allocator or the Safety Driver Generator. Tables are available to examine (a) the definition of the analysis subject, (b) the PHA, (c) the safety concept, (d) the system connectors, (e) the system components, (f) the software requirements and (k) the violated properties.

Assume M is an EAST-ADL model. The View Provider $\beta(M)$ can create tabular views V from the EAST-ADL model (Expression 15).

$$\beta(M) \rightarrow V \tag{15}$$

6.2. FTA Generator

The safety engineering workflow foresees the qualitative application of FTA to analyze and verify the system architecture. For the application of qualitative FTA, fault trees are required. A manual construction of fault trees that are consistent with the EAST-ADL model (describes PHA results and system architecture) is cumbersome and error-prone. Furthermore extracting the minimum cut sets from the fault trees is challenging. Thus, we propose automating the construction of fault trees and the extraction of minimum cut sets.

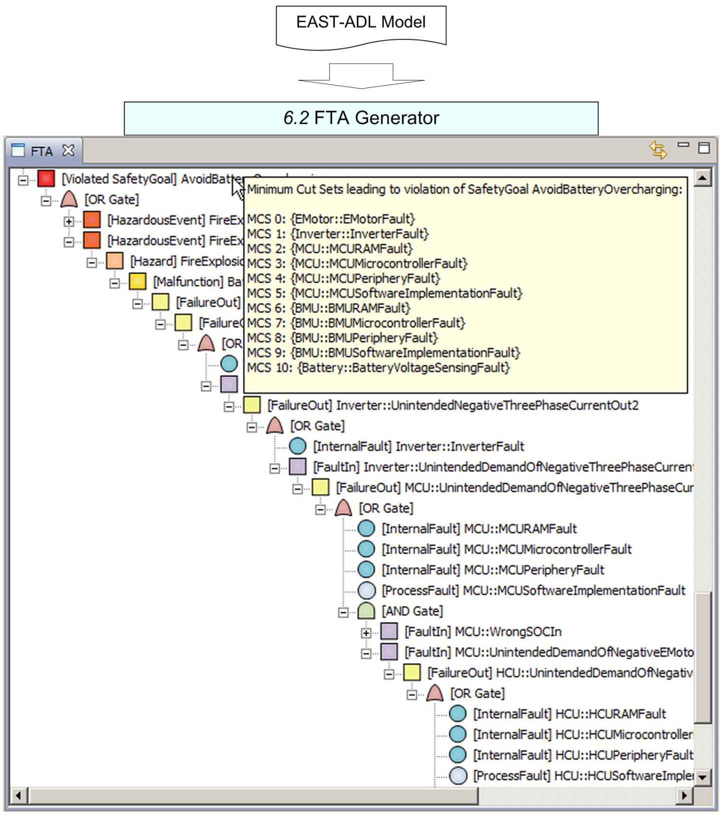
OASIS contains a FTA Generator (see Figure 2). Taking the EAST-ADL model as input, the FTA Generator can generate fault trees (see Figure 4) and extract their minimum cut sets. The generated fault trees are consistent with the results of PHA and describe how each safety goal can be violated by the faults and failures of the components of the system architecture. Faults are either random hardware faults or potential process faults that can occur in the course of the development process.

The FTA Generator uses the safety goals that were identified during PHA as top events of the generated fault trees (one fault tree per safety goal is generated). It adds the malfunctions, hazards and hazardous events that were identified during PHA and can lead to the violation of the safety goals as ancestors. Component faults, component failures and gates that were identified during the definition of the error model are used as ancestors of the malfunctions.

The minimum cut sets can be automatically extracted from each fault tree. A *minimum cut set* [2] is a set of basic events (faults and failures of components) leading to the top event (violation of the safety goal) that cannot be reduced in number. For every violated safety goal, the minimum cut sets can be displayed on demand.

The safety engineer can examine the minimum cut sets either as tool tip (see Figure 4) in the context of the fault tree or as MCS (minimum cut set) fault tree (see Figure 5). An MCS fault tree is a tree representation of the minimum cut sets together with their top event. Each MCS fault tree contains a violated safety goal as top event. The safety goals' ancestors are causative faults and failures of the components of the system architecture as well as logic symbols. The logic symbols show how the top event can be caused depending on faults and failures. To switch between fault trees and MCS fault trees, the safety engineer needs to click the arrow icon at the upper right of the FTA Generator GUI (see Figures 4 and 5).

The FTA Generator considers the recommendations of IEC 61025 [36]. Thus, the shapes of the symbols of the fault trees and the MCS fault trees are adapted to the shapes of the recommended symbols. Basic events (faults, failures) are represented by circles, complex events (faults, failures, malfunctions, hazards, hazardous events, violated safety goals) are represented by rectangles, and gates (and, or) are represented by logic symbols.



```
[Violated SafetyGoal] AvoidBatte
  [OR Gate]
    [HazardousEvent] FireEx
    [HazardousEvent] FireEx
      [Hazard] FireExplosic
        [Malfunction] Ba
          [FailureOut]
            [Failure
              [OR
                [FailureOut] Inverter::UnintendedNegativeThreePhaseCurrentOut2
                  [OR Gate]
                    [InternalFault] Inverter::InverterFault
                    [FaultIn] Inverter::UnintendedDemandOfNegativeThreePhaseCurren
                      [FailureOut] MCU::UnintendedDemandOfNegativeThreePhaseCur
                        [OR Gate]
                          [InternalFault] MCU::MCURAMFault
                          [InternalFault] MCU::MCUMicrocontrollerFault
                          [InternalFault] MCU::MCUPeripheryFault
                          [ProcessFault] MCU::MCUSoftwareImplementationFault
                          [AND Gate]
                            [FaultIn] MCU::WrongSOCIn
                            [FaultIn] MCU::UnintendedDemandOfNegativeEMoto
                              [FailureOut] HCU::UnintendedDemandOfNegativ
                                [OR Gate]
                                  [InternalFault] HCU::HCURAMFault
                                  [InternalFault] HCU::HCUMicrocontroller
                                  [InternalFault] HCU::HCUPeripheryFault
                                  [ProcessFault] HCU::HCUSoftwareImple

Minimum Cut Sets leading to violation of SafetyGoal AvoidBatteryOvercharging:

MCS 0: {EMotor::EMotorFault}
MCS 1: {Inverter::InverterFault}
MCS 2: {MCU::MCURAMFault}
MCS 3: {MCU::MCUMicrocontrollerFault}
MCS 4: {MCU::MCUPeripheryFault}
MCS 5: {MCU::MCUSoftwareImplementationFault}
MCS 6: {BMU::BMURAMFault}
MCS 7: {BMU::BMUMicrocontrollerFault}
MCS 8: {BMU::BMUPeripheryFault}
MCS 9: {BMU::BMUSoftwareImplementationFault}
MCS 10: {Battery::BatteryVoltageSensingFault}
```

Figure 4: Fault trees can be generated from the EAST-ADL model to support the application of qualitative FTA.

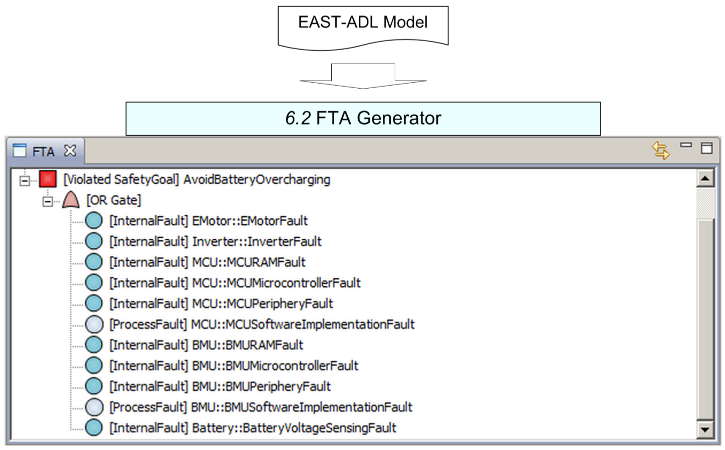
Assume M is an EAST-ADL model, and S is the subset of M that describes hazards, hazardous events, safety goals, system architecture and the error model (Expression 16).

$$S \subseteq M \tag{16}$$

The FTA Generator $\rho(S)$ can generate fault trees Υ that allow to examine how component faults and failures can contribute to the violation of safety goals (Expression 17).

$$\rho(S) \rightarrow \Upsilon \tag{17}$$

Assume that MCS_j is a minimum cut set of a generated fault tree. Furthermore, assume that SS_{MCS_j} is the set of all minimum cut sets that can



[Violated SafetyGoal] AvoidBatteryOvercharging
[OR Gate]
[InternalFault] EMotor::EMotorFault
[InternalFault] Inverter::InverterFault
[InternalFault] MCU::MCURAMFault
[InternalFault] MCU::MCUMicrocontrollerFault
[InternalFault] MCU::MCUPeripheryFault
[ProcessFault] MCU::MCUSoftwareImplementationFault
[InternalFault] BMU::BMURAMFault
[InternalFault] BMU::BMUMicrocontrollerFault
[InternalFault] BMU::BMUPeripheryFault
[ProcessFault] BMU::BMUSoftwareImplementationFault
[InternalFault] Battery::BatteryVoltageSensingFault

Figure 5: An MCS (minimum cut set) fault tree is a tree representation of the minimum cut sets that lead to the violation of a safety goal. MCS fault trees can be generated from the EAST-ADL model to support the application of qualitative FTA.

lead to the violation of the safety goals (Expression 18).

$$MCS_j \epsilon SS_{MCS_j} \tag{18}$$

The FTA Generator $\rho(\Upsilon)$ can automatically extract SS_{MCS_j} from the generated fault trees Υ (Expression 19).

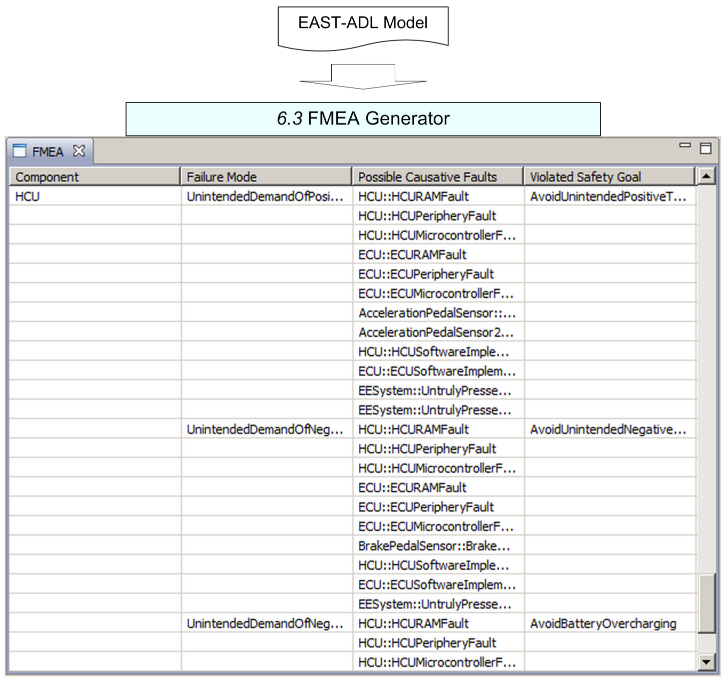
$$\rho(\Upsilon) \rightarrow SS_{MCS_j} \tag{19}$$

The FTA Generator $\rho(S)$ can generate MCS fault trees Υ' that allow examining the minimum cut sets that are included in SS_{MCS_j} as trees (Expression 20).

$$\rho(SS_{MCS_j}) \rightarrow \Upsilon' \tag{20}$$

6.3. FMEA Generator

The safety engineering workflow foresees the qualitative application of FMEA to analyze and verify the system architecture. For the application of qualitative FMEA, an FMEA table is required. A manual construction of an FMEA table that is consistent with the EAST-ADL model (contains PHA results and a description of the system architecture) is cumbersome and error-prone. Thus, we propose automating FMEA table construction.



Component	Failure Mode	Possible Causative Faults	Violated Safety Goal
HCU	UnintendedDemandOfPosi...	HCU::HCURAMFault	AvoidUnintendedPositiveT...
		HCU::HCUPeripheryFault	
		HCU::HCUMicrocontrollerF...	
		ECU::ECURAMFault	
		ECU::ECUPeripheryFault	
		ECU::ECUMicrocontrollerF...	
		AccelerationPedalSensor::...	
		AccelerationPedalSensor2...	
		HCU::HCUSoftwareImple...	
		ECU::ECUSoftwareImplem...	
		EESystem::UntrulyPresse...	
		EESystem::UntrulyPresse...	
	UnintendedDemandOfNeg...	HCU::HCURAMFault	AvoidUnintendedNegative...
		HCU::HCUPeripheryFault	
		HCU::HCUMicrocontrollerF...	
		ECU::ECURAMFault	
		ECU::ECUPeripheryFault	
		ECU::ECUMicrocontrollerF...	
		BrakePedalSensor::Brake...	
		HCU::HCUSoftwareImple...	
		ECU::ECUSoftwareImplem...	
		EESystem::UntrulyPresse...	
	UnintendedDemandOfNeg...	HCU::HCURAMFault	AvoidBatteryOvercharging
		HCU::HCUPeripheryFault	
		HCU::HCUMicrocontrollerF...	

Figure 6: An FMEA table can be generated from the EAST-ADL model to support the application of qualitative FMEA.

OASIS contains an FMEA Generator (see Figure 2). The FMEA Generator can generate an FMEA table from the EAST-ADL model (see Figure 6). The FMEA table shows the relations between the failure modes of the components of the system architecture, causative faults and effects in terms of violated safety goals. The generated FMEA table is consistent with the generated fault trees.

A generated FMEA table contains four columns denoting the names of the components (*Component*), the component failure modes leading to the violation of safety goals (*Failure Mode*), faults that potentially cause the component failure modes (*Possible Causative Faults*) and the violated safety goals (*Violated Safety Goal*).

Assume M is an EAST-ADL model and S is the subset of M that describes hazards, hazardous events, safety goals, system architecture and the error model (Expression 21).

$$S \subseteq M \tag{21}$$

The FMEA Generator $\alpha(S)$ can generate an FMEA table Ξ that allows to examine how component faults and failures can contribute to the violation of safety goals (Expression 22).

$$\alpha(S) \rightarrow \Xi \tag{22}$$

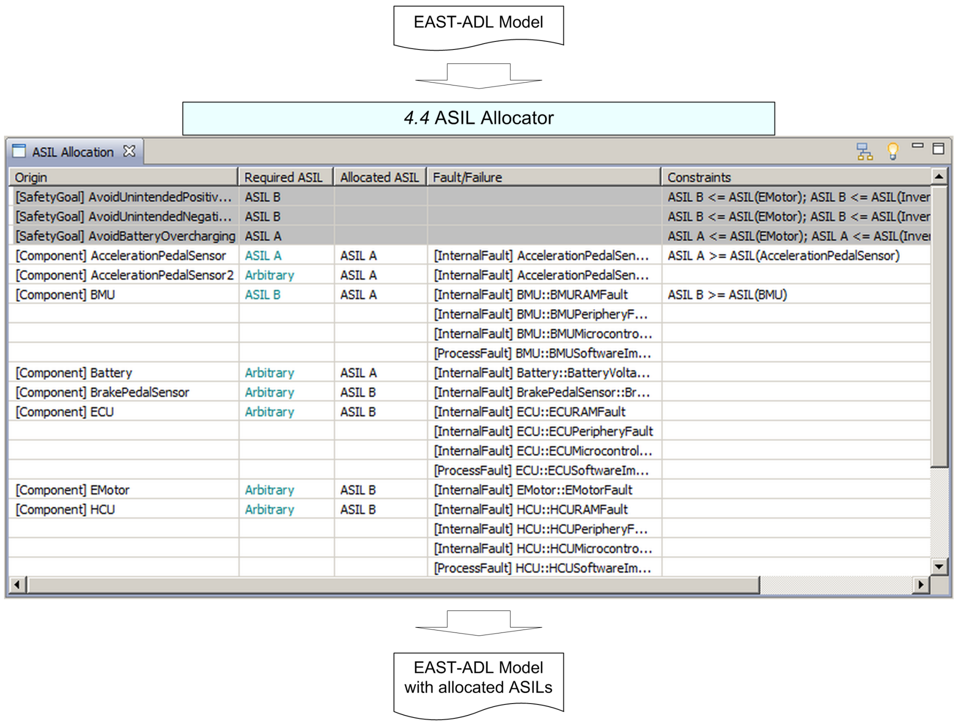
6.4. ASIL Allocator

The safety engineering workflow requires the allocation of ASILs to the components of the system architecture. The allocated ASILs determine applicable requirements of ISO 26262 and the necessary safety measures to avoid unreasonable residual risk. The requirements of ISO 26262 affect the rigor of the development process of the system components (e.g. the portfolio of applicable verification techniques). The necessary safety measures determine the runtime fault detection capabilities of the system components. Thus, the allocated ASILs strongly affect system safety. Furthermore, the allocated ASILs influence development costs and costs per piece.

An allocation of ASILs to the components of the system architecture shall, therefore (1) assure that the required level of functional safety is achieved and (2) permit an economic solution with respect to development costs and cost per piece. Manual elaboration of an ASIL allocation that fulfills the requirements is complex, cumbersome and should be automated.

OASIS contains an ASIL Allocator (see Figure 2) that uses a constraint solver. This constraint solver can automatically find an allocation of ASILs to the components of the system architecture. First, this allocation depends on constraints that are derived from (a) the safety goals defined during PHA and (b) the generated fault trees and their automatically extracted minimum cut sets (see Section 6.2). Second, this allocation depends on constraints that are derived from particular preferences of the safety engineer. The resulting allocation is optimal with respect to an objective function and consistent with PHA results, generated fault trees, their automatically extracted minimum cut sets and preferences of the safety engineer.

The GUI of the ASIL Allocator is illustrated in Figure 7. The gray lines of the table list the safety goals that were defined during PHA, and the white lines list the components of the system architecture (column Origin). The ASILs of the safety goals and the preferred component ASILs that were defined by the safety engineer are listed (column Required ASIL). Furthermore, the faults and failures of each component that lead to safety goal violations are listed (column Fault/Failure). Finally, the constraints on the component



Origin	Required ASIL	Allocated ASIL	Fault/Failure	Constraints
[SafetyGoal] AvoidUnintendedPositiv...	ASIL B			ASIL B <= ASIL(EMotor); ASIL B <= ASIL(Inver
[SafetyGoal] AvoidUnintendedNegati...	ASIL B			ASIL B <= ASIL(EMotor); ASIL B <= ASIL(Inver
[SafetyGoal] AvoidBatteryOvercharging	ASIL A			ASIL A <= ASIL(EMotor); ASIL A <= ASIL(Inve
[Component] AccelerationPedalSensor	ASIL A	ASIL A	[InternalFault] AccelerationPedalSen...	ASIL A >= ASIL(AccelerationPedalSensor)
[Component] AccelerationPedalSensor2	Arbitrary	ASIL A	[InternalFault] AccelerationPedalSen...	
[Component] BMU	ASIL B	ASIL A	[InternalFault] BMU::BMURAMFault	ASIL B >= ASIL(BMU)
			[InternalFault] BMU::BMUPeripheryF...	
			[InternalFault] BMU::BMUMicrocontro...	
			[ProcessFault] BMU::BMUSoftwareIm...	
[Component] Battery	Arbitrary	ASIL A	[InternalFault] Battery::BatteryVolta...	
[Component] BrakePedalSensor	Arbitrary	ASIL B	[InternalFault] BrakePedalSensor::Br...	
[Component] ECU	Arbitrary	ASIL B	[InternalFault] ECU::ECURAMFault	
			[InternalFault] ECU::ECUPeripheryFault	
			[InternalFault] ECU::ECUMicrocontrol...	
			[ProcessFault] ECU::ECUSoftwareIm...	
[Component] EMotor	Arbitrary	ASIL B	[InternalFault] EMotor::EMotorFault	
[Component] HCU	Arbitrary	ASIL B	[InternalFault] HCU::HCURAMFault	
			[InternalFault] HCU::HCUPeripheryF...	
			[InternalFault] HCU::HCUMicrocontro...	
			[ProcessFault] HCU::HCUSoftwareIm...	

Figure 7: ASILs (Automotive Safety Integrity Levels) can be automatically and optimally allocated to the components of the system architecture.

ASILs that were derived from PHA results, generated fault trees, extracted minimum cut sets and the preferences of the safety engineer are displayed (column Constraints).

Once an optimal solution is found, the solution can be automatically projected on the EAST-ADL model, rendering a manual model modification redundant. The approach to ASIL allocation is described subsequently.

6.4.1. Integer Linear Programing Problems

Linear programming denotes an approach to modeling and solving linear mathematical models, and more specifically those models that seek to optimize a linear measure of performance [37]. A single-objective linear programming model can be stated mathematically. Find the variables $x_1, x_2, ..., x_n \geq 0$ so as to optimize (either maximize or minimize) the objective function that is subject to specified constraints. Expression 23 defines the objective function z.

$$z(x_1, x_2, ..., x_n) = \sum_{i=0}^{n} x_i * c_i \tag{23}$$

In addition, the m specified constraints are defined by Expression 24 for the n variables.

$$\begin{gathered} x_1 * c_{11} + ... + x_n * c_{1n}\{\leq, =, \geq\}b_1 \\ x_1 * c_{21} + ... + x_n * c_{2n}\{\leq, =, \geq\}b_2 \\ \vdots \\ x_1 * c_{m1} + ... + x_n * c_{mn}\{\leq, =, \geq\}b_m \end{gathered} \tag{24}$$

An integer linear programming problem is a linear program in which some or all of the variables $x_1, x_2, ..., x_n$ are restricted to integer values.

6.4.2. ASIL-Arithmetic

ISO 26262 defines a method called ASIL decomposition. This method allows reducing the ASILs allocated to sufficiently independent components that can only jointly cause the violation of a safety goal. If a component can solely cause the violation of a safety goal, its ASIL cannot be reduced.

Rules for the reduction of the ASILs are defined by ISO 26262. To formalize these rules, QM, ASIL A, ASIL B, ASIL C and ASIL D can be interpreted as the integer numbers 0, 1, 2, 3 and 4.

Assume SG_j is a safety goal. In addition, assume $asil(SG_j)$ denotes the ASIL of a safety goal interpreted as an integer number (Expression 25).

$$asil(SG_j)\epsilon\{1, 2, 3, 4\} \tag{25}$$

Assume C_k denotes a component of the system architecture. Moreover, assume $asil(C_k)$ denotes the ASIL that is allocated to the component C_k interpreted as an integer number (Expression 26).

$$asil(C_k)\epsilon\{0, 1, 2, 3, 4\} \tag{26}$$

According to ISO 26262, if an ASIL decomposition is applied for l components that can only jointly cause the violation of the safety goal SG_j, an inequality shall be fulfilled (Expression 27).

$$\sum_{k=0}^{l} asil(C_k) \geq asil(SG_j) \tag{27}$$

This implies that ASILs, interpreted as integer numbers, can be added resulting in a new ASIL that is limited to 4.

6.4.3. ASIL-Allocation as Integer Linear Programing Problem

The problem of allocating ASILs to the components of the system under development can be interpreted as an integer linear programing problem. This approach is based on the assumption that (a) ASILs can be interpreted as integer numbers and (b) ASILs also can be an input to additions that result in a new ASIL (see also Section 6.4.2).

Commonly, the greater an allocated ASIL, the greater the efforts for the development process and the higher the costs per piece of a component. It can, therefore, be assumed that a low sum of allocated ASILs leads to an economic ASIL allocation for the system under development. An objective function F_{min} can be defined that shall be minimized in order to find an economic ASIL allocation to n components (Expression 28).

$$F_{min}(C_1, C_2, ..., C_n) = \sum_{i=0}^{n} asil(C_i) \tag{28}$$

Assume that SGS is the set of safety goals that was defined in the course of the safety engineering workflow (Expression 29).

$$SG_k \epsilon SGS \tag{29}$$

Assume f_i is a fault or failure of a component of the system architecture or the system environment. Additionally, assume f_i is part of the minimum cut set MCS_j that was automatically extracted from the generated fault trees (see Section 6.2) and can lead to the violation of safety goal SG_k (Expression 30).

$$f_i \epsilon MCS_j \tag{30}$$

Based on the set of safety goals SGS and the minimum cut sets that contribute to the violation of the safety goals, l constraints can be defined for m safety goals and n components (Expression 31). These constraints assure that the allocation of ASILs is consistent with the safety goals and the minimum cut sets and assure the achievement of functional safety according to ISO 26262.

$$\begin{array}{c} asil(C_1) * c_{11} + ... + asil(C_n) * c_{1n} \geq asil(SG_1) \\ \vdots \\ asil(C_1) * c_{p1} + ... + asil(C_n) * c_{pn} \geq asil(SG_1) \\ asil(C_1) * c_{q1} + ... + asil(C_n) * c_{qn} \geq asil(SG_2) \\ \vdots \\ asil(C_1) * c_{l1} + ... + asil(C_n) * c_{ln} \geq asil(SG_m) \end{array} \tag{31}$$

A constraint is defined for each minimum cut set that can lead to the violation of a safety goal in SGS and exclusively consists of faults and failures that result from a component C_k of the system under development. Every constraint contains a coefficient c_{xy} per component. If a failure f_i of a component C_k is element of a minimum cut set MCS_j, the coefficient c_{xy} is set to 1 in the corresponding constraint. Else, the coefficient is set to 0. Thus, if components can only jointly cause the violation of a safety goal, their coefficients are 1 while the remaining coefficients are 0 in the corresponding constraint. If a component can solely cause the violation of a safety goal, its coefficient is the only coefficient that is 1 in the corresponding constraint.

Usually, a safety engineer has certain preferences concerning the allocation of safety integrity levels to the components of the system under development. A reason for that may be the intended reuse of a component C_k developed in an earlier project according to a particular ASIL. Another reason may be the selected limitation of the ASIL allocated to a component C_k in order to limit costs per piece or development costs. Assume $pasil(C_k)$ is the preferred ASIL for a component C_k (Expression 32).

$$pasil(C_k) \epsilon \{0, 1, 2, 3, 4\} \tag{32}$$

To ensure the consideration of such preferences, an additional constraint on the ASIL of each of the n system components can be defined by the safety engineer (Expression 33). If a preference for a component C_k is defined, the corresponding coefficient d_{xy} of the component is the only one that is 1.

$$\begin{array}{c} asil(C_1) * d_{11} + ... + asil(C_n) * d_{1n} \leq pasil(C_1) \\ asil(C_1) * d_{21} + ... + asil(C_n) * d_{2n} \leq pasil(C_2) \\ asil(C_1) * d_{31} + ... + asil(C_n) * d_{3n} \leq pasil(C_3) \\ \vdots \\ asil(C_1) * d_{n1} + ... + asil(C_n) * d_{nn} \leq asil(C_n) \end{array} \tag{33}$$

The constraint solver of the ASIL Allocator can be used to solve the ILP problem. Therefore, the safety engineer needs to click the bulb icon at the upper right of the ASIL Allocator GUI (see Figure 7). Then, the constraint solver attempts to determine the parameters $asil(C_i)$ leading to a minimum of the function F_{min} under consideration of the constraints on the parameters $asil(C_i)$ imposed by Expression 31 and Expression 33. If the constraint solver finds a solution, the solution is (a) optimal with respect to the objective function F_{min}, (b) consistent with results of PHA, generated fault trees and their minimum cut sets and (c) considers the preferences of the safety engineer.

A safety engineer can inspect the solution to the problem. If the solution is satisfactory, the safety engineer can initiate the allocation of the ASILs to the components. In this case, the EAST-ADL model is automatically modified according to the accepted solution. Therefore, the safety engineer needs to click the rectangle icon at the upper right of the ASIL Allocator GUI (see Figure 7).

If the result is not satisfactory or the imposed ILP problem cannot be solved, the safety engineer needs to revise the preferences concerning the ASIL allocation and needs to rerun the constraint solver.

6.5. Report Generator

The EAST-ADL model that is created in the course of the safety engineering workflow reflects the work products of the phases that are defined in the Sections 2.1, 2.2 and 2.3.1. EAST-ADL is a language that can be understood by professionals like safety engineers. In contrast, project stakeholders or reviewers typically await a document-based representation of work products. The manual writing of a report describing the work products of the safety engineering workflow and consistent with the EAST-ADL model is cumbersome and error-prone. Thus, automatic report generation is required to support communication and reviews.

OASIS contains a Report Generator (see Figure 2) that can generate a document-based report from the EAST-ADL model. This report describes the created work products as tables or trees and is consistent with the EAST-ADL model. The report can contain chapters for (a) the definition of the analysis subject, (b) the PHA, (c) the safety concept, (d) the system connectors, (e) the system components, (f) the software requirements and (k) the violated properties. The safety engineer can decide which chapters to include or omit.

Assume M is an EAST-ADL model. Furthermore, assume that Report Generator $\alpha(M)$ can generate a report R from the EAST-ADL model (Expression 34).

$$\alpha(M) \rightarrow R \tag{34}$$

7. Support for Configuration and Code Generation

The application of the safety engineering workflow requires the design and the implementation of software for the components of the system architecture. These are challenging tasks, and it is difficult to design and implement software that is consistent with the EAST-ADL model. To support these activities, OASIS provides the following modules:

- The *Simulink Model Generator* (see Section 7.1) allows the automated generation of Simulink models from the EAST-ADL model. This ensures consistency between EAST-ADL model and Simulink models and is the basis for the subsequent description of behaviors (implementation of the functions).
- The *Safety Driver Generator* (see Section 7.2) allows the computer-aided configuration and generation of safety drivers for safe initialization, runtime-testing and error-handling of microcontrollers. This module extracts information from the EAST-ADL model, guides the safety engineer, updates the model and generates program code.

7.1. Simulink Model Generator

Typically, some of the components of the system architecture are required to execute application software using microcontrollers. The EAST-ADL model describes these components together with their interfaces defined in terms of ports. The safety engineering workflow requires the creation of Simulink models for design and implementation of application software. Such a model shall contain a top-level block with an interface defined in terms of ports as well. These Simulink models with their top-level blocks and ports must be consistent with the EAST-ADL model. The manual annotation of these Simulink models is cumbersome and error-prone and should be computer-aided.

OASIS contains a Simulink Model Generator (see Figure 2) that is capable of generating Simulink models from the EAST-ADL model. These Simulink models reflect the structure of the components of the system architecture that are required to execute application software in terms of top-level block and interface. The generated Simulink models are refined and enhanced using Simulink's graphical modeling language in the course of the safety engineering workflow. Once enhanced and refined, C-code can be generated from these models. In the following, our approach to Simulink model generation is described.

In the EAST-ADL model, each component of the system architecture is represented by a modeling element C_k. Each component C_k is typed by a type T_i. If a type T_i is required to execute application software according to the EAST-ADL model, it is a *candidate* for Simulink model generation. Whereas each component C_k has a single type T_i, a type T_i can type multiple components of the system architecture. Thus, if multiple components are typed by the same type T_i, application software needs once to be developed per type T_i but not per component C_k.

The Simulink Model Generator can investigate the EAST-ADL model and identify candidates for Simulink model generation. The safety engineer can use the GUI illustrated in Figure 9 for Simulink model generation. Assume that M is an EAST-ADL model containing at least one application software component typed by T_i. The Simulink Model Generator $\eta(M, T_i)$ can generate a Simulink model S_{T_i} that is consistent with T_i in terms of (a) top-level block and (b) interface (Expression 35).

$$\eta(M, T_i) \rightarrow S_{T_i} \tag{35}$$

7.2. Safety Driver Generator

Typically, some of the components of the system architecture are required to execute basic software using microcontrollers. This basic software is partly dedicated to (1) safe initialization, (2) runtime-testing and (3) error-handling. Whereas (1) and (2) are required to detect random hardware faults during system operation, (1) and (3) are required to achieve and maintain a safe state upon detecting random hardware faults. The design and the implementation of these functionalities are difficult tasks. Thus, we propose tool support for their design and implementation.

OASIS contains a Safety Driver Generator (see Figure 2) that is capable of configuring and generating so-called safety drivers. The term safety driver refers to the part of the basic software for a component of the system architecture that is dedicated to (a) safe initialization, (b) runtime detection of random hardware faults and (c) error-handling of its supported microcontroller. To aid safety driver configuration, the Safety Driver Generator can suggest safety driver configurations based on the ASILs allocated to the components of the system architecture. There are GUIs for the configuration (see Figure 8) and the generation (see Figure 9) of safety drivers. In the following, our approach to configure and generate is described. To illustrate this approach, we describe OASIS' support for an industrial multi-core microcontroller.

7.2.1. Safety Driver Configuration

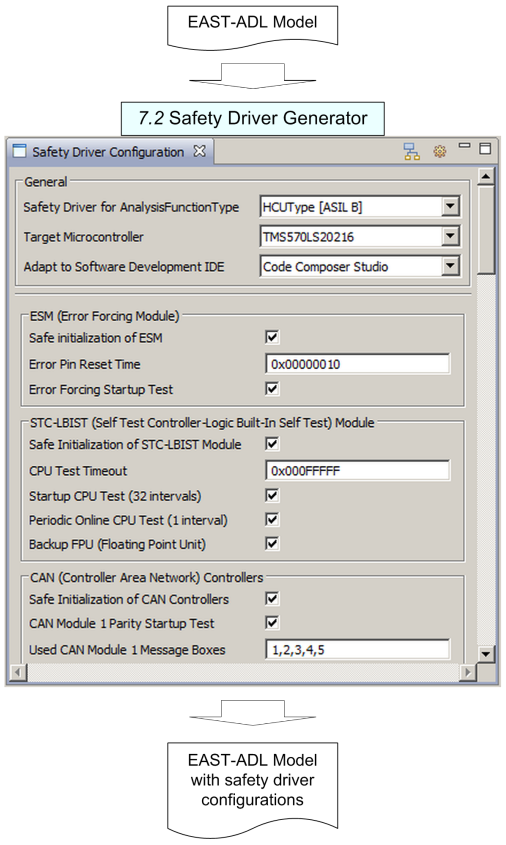
In the EAST-ADL model, each component of the system architecture is represented by a modeling element C_k. Each component C_k is typed by a type T_i. If a type T_i is required to execute basic software according to the EAST-ADL model, it is a *candidate* for safety driver generation. Whereas each component C_k has a single type T_i, a type T_i can type multiple components of the system architecture. Thus, if multiple components are typed by the same type T_i, basic software needs once to be developed per type T_i but not per component C_k.

Based on the ASIL Arithmetic defined in Section 6.4.2, assume that $rasil(T_i)$ is the maximum ASIL of all components C_k typed by T_i. We refer to $rasil(T_i)$ as the *relevant ASIL* of T_i (Expression 36).

$$rasil(T_i) = \max_{k=0}^{n} asil(C_k) \tag{36}$$

The Safety Driver Generator can investigate the EAST-ADL model and identify candidates for safety driver configuration and generation. The Safety Driver Generator provides a GUI that presents these candidates together with their relevant ASILs (see Figure 8). The GUI can be used to select the candidates for whom a safety driver has to be configured. In any case, safety driver configuration for a candidate is optional.

The safety engineer can also select a target microcontroller for each candidate. In this case, a microcontroller-dependent part of the GUI appears. This includes a list of software development IDEs. The items of this list depend on the Safety Driver Generator's support for the selected microcontroller. It is important to choose the right software development IDE because a generated safety driver may contain proprietary language constructs that can only be understood by particular IDEs.


EAST-ADL Model
7.2 Safety Driver Generator
Safety Driver Configuration
EAST-ADL Model with safety driver configurations

General
Safety Driver for AnalysisFunctionType: HCUType [ASIL B]
Target Microcontroller: TMS570LS20216
Adapt to Software Development IDE: Code Composer Studio

ESM (Error Forcing Module)
Safe initialization of ESM: ✓
Error Pin Reset Time: 0x00000010
Error Forcing Startup Test: ✓

STC-LBIST (Self Test Controller-Logic Built-In Self Test) Module
Safe Initialization of STC-LBIST Module: ✓
CPU Test Timeout: 0x000FFFFF
Startup CPU Test (32 intervals): ✓
Periodic Online CPU Test (1 interval): ✓
Backup FPU (Floating Point Unit): ✓

CAN (Controller Area Network) Controllers
Safe Initialization of CAN Controllers: ✓
CAN Module 1 Parity Startup Test: ✓
Used CAN Module 1 Message Boxes: 1,2,3,4,5

Figure 8: Safety drivers for initialization, random hardware fault detection and error-handling can be configured for the components of the system architecture.

Furthermore, the microcontroller-dependent part of the GUI includes options for the safety driver generation that depend on the specifics and features (e.g. CPU instruction set, memory sizes, control registers or hardware-based self-test capabilities) of the selected microcontroller. These options can significantly differ from microcontroller to microcontroller.

The Safety Driver Generator can automatically propose configurations of these options depending on the relevant ASILs of the candidates. ASIL-dependent, proposed configurations are predefined based on expert knowledge for potentially relevant ASILs. To obtain a proposed configuration, the safety engineer needs to click the bulb icon illustrated in Figure 8.

An ASIL-dependent, proposed configuration guides and supports the safety engineer in defining an adapted configuration meeting the needs of the application and the safety concept elaborated in the course of the safety engineering workflow. Although the Safety Driver Generator provides guidance for selecting options according to a particular ASIL, a safety engineer still needs to be aware of the specifics of the selected microcontroller to appropriately adapt a proposed configuration to the needs of the application and the safety concept.

Assume K_{T_i} is a configuration of a safety driver for the microcontroller of a particular candidate T_i. The Safety Driver Generator $\phi(rasil(T_i))$ can propose a configuration K_{T_i} of the safety driver based on the relevant ASIL of T_i (Expression 37).

$$\phi(rasil(T_i)) \to K_{T_i} \tag{37}$$

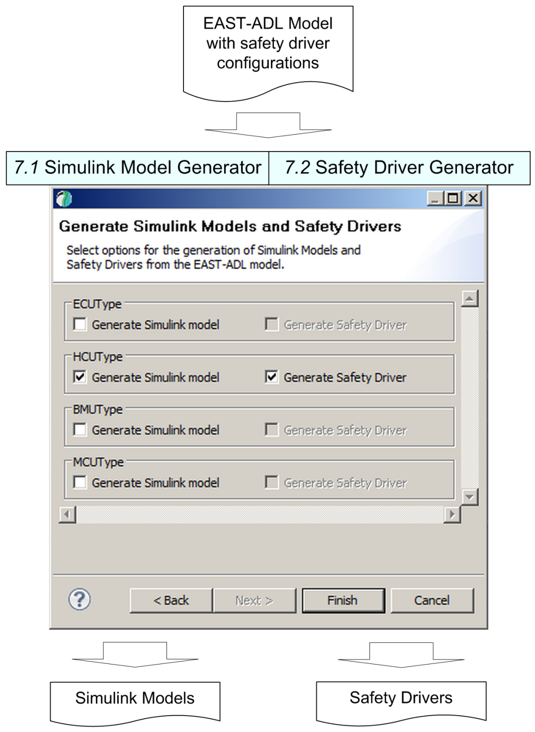
A safety engineer can adapt the proposed configuration based on the needs of the application and the safety concept. For example, if the application requires a particular microcontroller module (e.g. a communication peripheral) and the safety concept requires according runtime-testing, the safety engineer can adapt the proposed configuration to enable safe initialization and appropriate runtime-testing of this module. In contrast, if a particular module is not required, the safety engineer can turn off the options for the module's safe initialization and runtime-testing to safe computational resources.

Once the driver configurations for the candidates are satisfactory, the safety engineer can project these configurations on the EAST-ADL model by clicking the rectangle icon. In this case, the EAST-ADL model is automatically modified. The configurations are projected in (1) a machine-readable representation and in (2) a human-readable representation on the EAST-ADL model. The machine-readable representation is required for subsequent safety driver generation and mainly consists of machine-readable name/value pairs. The human-readable representation describes the configuration of the safety drivers using natural language as software safety requirements. Traces from the safety concept to the generated software safety requirements need to be manually created in accordance with the safety engineering workflow.

Assume TS is the set of all safety driver configurations defined for the candidates (Expression 38).

$$K_{T_i} \epsilon TS \tag{38}$$

Furthermore, assume M is an EAST-ADL model. The Safety Driver Generator $\phi(M, TS)$ can project the configurations of the safety drivers for


EAST-ADL Model with safety driver configurations
7.1 Simulink Model Generator
7.2 Safety Driver Generator
< Back
Next >
Finish
Cancel
Simulink Models
Safety Drivers

Generate Simulink Models and Safety Drivers

Select options for the generation of Simulink Models and Safety Drivers from the EAST-ADL model.

ECUType
- [ ] Generate Simulink model
- [ ] Generate Safety Driver

HCUType
- [x] Generate Simulink model
- [x] Generate Safety Driver

BMUType
- [ ] Generate Simulink model
- [ ] Generate Safety Driver

MCUType
- [ ] Generate Simulink model
- [ ] Generate Safety Driver

Figure 9: Safety drivers and Simulink models can be generated from the EAST-ADL model supporting the development of application software and basic software.

the candidates onto the EAST-ADL model. This results in an EAST-ADL model M' (Expression 39).

$$\phi(M, TS) \rightarrow M' \tag{39}$$

7.2.2. Safety Driver Generation

After the configuration of safety drivers for the candidates and the projection of the configurations onto the EAST-ADL model, the safety engineer can generate the safety drivers from the EAST-ADL model. The Safety Driver Generator provides a GUI for safety driver generation (see Figure 9). This GUI allows to select the candidates whose safety drivers were configured for safety driver generation.

Assume M' is an EAST-ADL model containing the safety driver configurations TS defined for the candidates. Assume K_{T_i} is a configuration in TS.

The Safety Driver Generator $\phi(M', K_{T_i})$ can generate a safety driver D_{T_i} for each configuration (Expression 40).

$$\phi(M', K_{T_i}) \rightarrow D_{T_i} \tag{40}$$

Each generated safety driver contains a header file describing the safety-driver's interface in terms of functions for (1) initialization, (2) runtime-testing and (3) error-handling. This interface is automatically documented during the generation process and describes the configuration of the generated safety driver.

A generated safety driver impacts runtime fault detection and error-handling capabilities of a microcontroller. This has a direct impact on the safe operation of the components of the system architecture. Thus, to make the presented approach practically applicable, (a) the Safety Driver Generator must either be certified, or (b) a generated safety driver must be verified.

7.2.3. Support for an Industrial Multi-Core Processor

To illustrate the concept of configuration and generation of safety drivers, OASIS' support for an industrial lock-step multi-core microcontroller is presented in this section. This microcontroller is the TMS570LS20216 [31] belonging to the TMS570 family. This family represents a new generation of industrial multi-core microcontrollers aiming at safety-critical applications. Other representatives of this new generation are the MPC564xL family [32] and the AUDO family [38]. The MPC564xL family and the TMS570 family are based on the lock-step architecture [39]. In contrast, the AUDO family follows an approach that foresees the implementation of an asymmetric controller strategy on a single chip [40].

The TMS570LS20216 offers built-in safety-related features. The two CPUs of the microcontroller can operate in lock-step mode. In this case, both Cortex-R4F CPUs execute the same program, and their results are continuously compared. Discrepancies indicate faults. Faults can be signaled internally or towards an external device. The TMS570LS20216 allows protecting flash and RAM using ECCs (error correction codes). It also provides hardware support for self tests of its CPUs, memories and peripherals.

To adapt the TMS570LS20216 to the needs of the application and the safety concept and to manage the built-in safety-related features, a multitude of control and status registers is available that can be used by the programmer. The multitude of safety features and configuration possibilities motivates the presented approach that foresees the configuration and generation of safety drivers using the Safety Driver Generator. This helps to cope with the complexity of the device and makes it superfluous to manually design and implement the functionality covered by a generated safety driver.

Support for configuration and generation of safety drivers was designed with respect to a safety manual for the TMS570LS20216 [41] provided by the vendor. Consequently, generated safety drivers can cover microcontroller modules commonly used by all typical applications according to the safety manual as well as modules supporting the automotive communication protocol CAN (Controller Area Network). Furthermore, support for the TMS570LS20216 is restricted to Code Composer Studio. Thus, generated safety drivers for the TMS570LS20216 can contain language constructs that can only be understood by the software development IDE.

The safety engineer can configure TMS570LS20216 safety drivers for the candidates by defining, selecting and deselecting options for each candidate. This can be done using the GUI illustrated in Figure 8.

Once the safety driver configurations for selected candidates are completed, the configurations can be projected on the EAST-ADL model. Taking this model as input, the Safety Driver Generator can generate safety drivers from the EAST-ADL model. Depending on the configuration, a generated safety driver for the industrial multi-core processor provides an automatically documented interface describing its configuration. The interface consists of up to eight C-functions and can be used by the application. One of these functions is dedicated to safe initialization, two functions are dedicated to start-up testing, one function is dedicated to periodic online testing and four functions are dedicated to error handling.

The remainder of this section describes the Safety Driver Generators's options for safety driver configuration and generation for the TMS570LS20216.

CAN. The CAN controllers can be initialized to activate parity checking for its message objects. Further options are a parity fault injection test and a loop back test for selectable CAN controllers and message boxes on startup.

CCM-R4F. The CCM-R4F (CPU Compare Module for Cortex-RF4) module can be initialized in lock-step mode. Furthermore, up to three startup tests checking the fault detection mechanisms of the CCM-R4F are selectable.

DMA. The DMA (Direct Memory Access) module can be initialized to enable parity checking of the control packets. Moreover, it can be chosen to test the parity checking schema of the control packets for selectable channels on startup. It can also be chosen to test the memory protection schema of the MPU (memory protection unit) for selectable regions and channels on startup.

ESM. The ESM (Error Signaling Module) can be initialized to signal detected faults through a pin of the microcontroller and to trigger high level interrupts on fault detection. The reset pin time is configurable. Additionally, a startup self test is selectable.

Flash. The Flash module can be safely initialized to enable flash error detection, flash error correction, single bit error interrupt creation and to disable flash writing during runtime. Furthermore, a corrupted value test and a golden value test are selectable to test the error detection and correction schema upon startup. The corrupted value, its address and the address of the golden value are configurable.

PBIST. The PBIST (Programmable Built-In Self Test) module can apply different test modes for the ROMs and RAMs upon startup. The RAMs and ROMs that shall be tested are selectable.

RTI. The RTI (Real-Time Interrupt) module provides timer functionality. Two startup tests are available that can check the behavior of the RTI module for selectable counters, counter compares and interrupts.

STC-LBIST. The STC-LBIST (Self Test Controller-Logic Built-In Self Test) module allows CPU testing during runtime. The timeout counter of the STC-LBIST can be initialized with a designated value. Furthermore, a startup test and a periodic online test of the CPUs of the microcontroller are selectable. A backup and restoration strategy for the FPU (floating point unit) status is a further option.

TCRAM Wrappers. The TCRAM (Tightly-Coupled RAM) Wrappers can be initialized to enable parity checking of the address bus and ECC protection of the RAM. Two startup self tests of the error detection schemes and a periodic online test for the SRAM are selectable.

VIM. The VIM (Vector Interrupt Module) can be initialized to enable parity checking and error-handling. Moreover, a startup fault injection test is selectable.

Watchdog. An interface towards an external windowed watchdog for program flow monitoring can be configured. The safety engineer can choose port, bit and determine the reset schema (active high/active low).

8. Experimental Evaluation

The tool OASIS was designed and implemented as a plugin for the tool Papyrus as described in the Sections 5, 6 and 7. A tool chain was set up like described in Section 4 (the tool Code Composer Studio was chosen as software development IDE). Section 8.1 presents the application of the safety engineering workflow for the case study of an HEV development. Section 8.2 presents metrics that describe the complexity of the resulting EAST-ADL model and derived entities. Section 8.3 systematically assesses the benefits of OASIS with respect to defined evaluation criteria.

8.1. Case Study of HEV Development

The presented approach was experimentally evaluated using the case study of HEV [42] development. One of the main characteristics of HEVs is the addition of an electric machine supporting the classic combustion engine providing supplementary or substitutive torque. If such a vehicle uses its electric machine as electric motor to support the combustion engine, it discharges the battery. If the electric machine is used as a generator to regain energy while the vehicle decelerates (recuperation), it recharges the battery and/or supplies the auxiliaries with electrical energy. The Sections 8.1.1, 8.1.2 and 8.1.3 describe the case study of HEV development for the Concept Phase, the System Level Development Phase and the Software Level Development Phase of the workflow.

8.1.1. Concept Phase

In the concept phase, PHA was carried out to identify, assess and classify hazards. Safety goals and functional safety requirements were derived. The results of PHA were annotated using EAST-ADL. During the analysis, the View Provider was used to examine the definition of the analysis subject, the PHA results, the safety goals and the functional safety requirements.

It turned out that the embedded system, sensors or actuators of the HEV are safety-critical, because their failures can cause malfunctions such as overcharging of the battery. This can lead to hazards like fire and/or explosion. Thus, a modeling element named *FireExplosion* was created. Furthermore, a hazardous event named *FireExplosionDuringOvertaking* was created containing the results of the assessment of the hazard *FireExplosion* in the context of an operational situation.

The safety engineering workflow was applied erroneously. No safety goal (top-level safety requirement) was derived from the hazardous event *FireExplosionDuringOvertaking* to demand the mitigation of the hazard *FireExplosion.* This was reflected by the EAST-ADL model. Figure 3 illustrates the GUI of the Property Checker that detected the erroneous application based on the EAST-ADL model and reported that modeling element *FireExplosionDuringOvertaking* violated property 12a.

Due to the erroneous application of PHA, it was necessary to apply corrective measures and to modify the EAST-ADL model. As illustrated in Figure 3, the Model Corrector proposed the creation of a new safety goal and the association with the hazardous event *FireExplosionDuringOvertaking* on demand. This is the appropriate measure to correct the erroneously applied workflow with respect to the violated property. After having selected this option, the EAST-ADL model was automatically modified and a new traceable safety goal was created. Subsequently, the newly created modeling element was named *AvoidBatteryOvercharging* and refined using textual descriptions.

Finally, the Selective Interface was used to export operational situations and use cases from the resulting model to the Model Library for future reuse.

8.1.2. System Level Development Phase

In the system level development phase, technical safety requirements and a part of the system architecture and its environment were defined. The part of the system architecture includes the HCU (Hybrid Control Unit). This control unit manages the interaction of HEV components such as battery, electric machine, inverter as well as other control units.

The propagation of faults and failures was estimated, and an error model was created accordingly. The failure *UnintendedNegativeTorque2* of the component *EMotor* was identified to be the cause for the malfunction *BatteryOvercharging* that was identified during PHA. This failure can occur due to a failure of the electric machine or faults propagating from a sensor and networked control units such as the HCU.

Property Checker and Model Corrector supported the workflow steps. Furthermore, the View Provider was used to examine the definition of the safety concept, the system connectors and the system components. Fault trees and an FMEA table were generated from the EAST-ADL model using the FTA Generator and the FMEA Generator. Thereafter, qualitative FTA and qualitative FMEA were applied to verify the system architecture.

Figure 4 illustrates a fault tree showing the relations between safety goal *AvoidBatteryOvercharging*, hazardous event *FireExplosionDuringOvertaking*, hazard *FireExplosion*, malfunction *BatteryOvercharging* as well as the causative faults and failures of the components of the system architecture. The extracted minimum cut sets that can cause the violation of the safety goal *AvoidBatteryOvercharging* are also illustrated. The same minimum cut sets are shown in Figure 5 as MCS Fault Tree.

Figure 6 shows a part of the generated FMEA table. It reveals that a failure mode of the HCU can lead to the violation of the safety goal *AvoidBatteryOvercharging*. It also presents causative faults for this failure mode.

The HEV contains two redundant and diverse acceleration pedal sensors. Due to the redundancy, the fault or failure of one pedal sensor can only cause the violation of a safety goal in conjunction with the fault or failure of the other one. This is the prerequisite for ASIL decomposition.

For economic reasons, it was decided to set the preferred ASILs of the HCU and one acceleration pedal sensor to ASIL A using the GUI of the ASIL Allocator (see Figure 7). The ASIL Allocator derived constraints on the ASIL allocation accordingly. The constraint solver was activated, but could not find a solution for the derived constraint set. This indicated that at least one of the preferred ASILs was in conflict with the constraints derived from the safety goals, the fault trees and the extracted minimum cut sets.

Thus, the preferred ASIL of the HCU was set to ASIL B, and the constraint solver was activated again. The constraint solver was able to find a solution for the modified constraint set (see column Allocated ASIL in Figure 7). This solution is optimal with respect to the defined objective function and consistent with the defined safety goals, the fault trees, their minimum cut sets and the defined preferences for the ASIL allocation. Furthermore, the constraint solver identified the possibility for ASIL decomposition and decomposed the ASILs of the redundant acceleration pedal sensors to ASIL A+ASIL A. Afterwards, the ASIL allocation was automatically projected on the EAST-ADL model.

8.1.3. Software Level Development Phase

In the software level development phase, the components of the system architecture that are required to execute application software and basic soft-

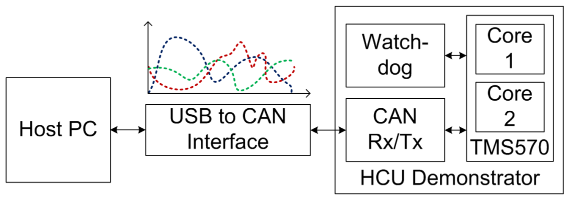


Figure 10: The safety engineering workflow was applied for the case study of hybrid electric vehicle development. Finally a control unit demonstrator was created that is based on a multi-core microcontroller and a residual bus simulation was set up.

ware were determined and software requirements were derived and allocated. One of these components is the HCU that is typed by the *HCUType*.

The Safety Driver Generator was used to configure a safety driver for the *HCUType* assuming a TMS570LS20216 as target microcontroller and Code Composer Studio as software development IDE. The Safety Driver Generator proposed an initial configuration of the safety driver according to the relevant ASIL of the *HCUType* (ASIL B). This configuration was manually refined to fit the needs of the *HCUType*. For example, the options for FPU backup (the HCU application requires the FPU), safe CAN initialization (the HCU application requires CAN) and parity startup testing of the message boxes 1, 2, 3, 4 and 5 of the CAN Module 1 (required by the HCU application) were selected. Thereafter, the configuration was automatically projected onto the EAST-ADL model resulting in (a) a group of name/value-pairs and (b) a list of software safety requirements describing the configuration in a human-readable manner. Traces from the newly created software safety requirements to the safety concept were created. A part of the configuration of the safety driver is illustrated in Figure 8.

Property Checker and Model Corrector supported the definition of software requirements and the creation of traces. Furthermore, the View Provider was used to examine the software requirements and their allocation. The Report Generator was employed to generate a report that described the software requirements, their allocation as well as all work products that were created in the preceding workflow phases.

After the configuration of the safety driver, the Safety Driver Generator and the Simulink Model Generator were used to configure and generate a safety driver and to generate a Simulink model. Whereas the safety driver was fitted to the TMS570LS20216 and the *HCUType*, the Simulink model was consistent with the *HCUType* in terms of top-level block and interface.

Thereafter, based on the generated Simulink model, the TargetLink blockset was used to define *HCUType* functionality according to the allocated application software requirements. The behavior determines the electric machine torque demanded by the HCU depending on the SOC (state of charge) of the HEV-battery, and the acceleration and brake pedal positions of the HEV. Subsequently, C-code was generated from the Simulink model using TargetLink. An embedded RTOS was instantiated, legacy code and I/O-related code were created. Then, the generated C-code (safety driver and application software) was integrated with the RTOS, the legacy code and the I/O-related code according to the allocated basic software requirements.

The program was compiled and linked and the resulting executable was deployed on a TMS570LS20216 microcontroller prototyping board. This prototyping board served as HCU demonstrator and is capable of safely initializing the TMS570LS20216, carrying out startup tests and periodic runtime tests, handling errors and executing the HCU application. A residual bus simulation was set up. Simulation software was executed on a PC to transmit stimuli to the HCU demonstrator via a USB-to-CAN adapter. The HCU demonstrator responded accordingly. Stimuli and responses were visualized using the simulation software. This setup is illustrated in Figure 10.

8.2. Complexity of Resulting Model and Derived Entities

While the safety engineering workflow was applied for the case study of hybrid electric vehicle development, an EAST-ADL model was created and different entities were derived from the model. This section presents metrics together with values characterizing the complexity of the resulting model and the derived entities. These metrics and values fall into different categories and are presented in Table 3.

Category (1) describes the complexity of the resulting EAST-ADL model. Whereas the workflow was only applied for a part of the hybrid electric vehicle powertrain consisting of 3 sensors, 3 actuators and 4 control units, the resulting EAST-ADL contained 957 modeling elements. The large number of modeling elements is not ascribable to EAST-ADL that allowed to appropriately structure the information and to efficiently express the work products. On the contrary, the large number of modeling elements is ascribable to the complexity of an HEV powertrain. This complexity is existent and challenging, regardless whether an EAST-ADL model is used or not.

The rigor of the information structure of the EAST-ADL model was exploited by OASIS to support the creation of consistent and complete work

(1) EAST-ADL Model Metrics	Value
# of Modeled Sensors	3
# of Modeled Actuators	3
# of Modeled Control Units	4
# of EAST-ADL Modeling Elements	957
(2) Model Creation and Reuse Metrics	**Value**
# of Property Violations	160
# of Violating Modeling Elements	152
# of Modeling Elements for Future Reuse	36
(3) Analysis and Documentation Metrics	**Value**
# of Generated Fault Trees	6
# of Generated Fault Tree Nodes	245
# of Extracted Minimum Cut Sets	45
# of Generated FMEA Table Lines	192
# of Derived Constraints for ASIL Allocation	43
# of Automatically Allocated ASILs	10
# of Provided View Table Lines	701
# of Words of Generated Report	7299
# of Tables of Generated Report	9
# of Trees of Generated Report	6
(4) Configuration and Code Generation Metrics	**Value**
# of Generated Simulink Model Ports	5
# of Requirements from Safety Driver Configuration	44
# of Lines of Code of Generated Safety Driver	4224

Table 3: This table presents metrics together with values characterizing the complexity of the EAST-ADL model that results from the HEV use case and the derived entities.

products and to simplify and automate workflow steps. The categories (2), (3) and (4) describe the complexity of the entities that were automatically derived by OASIS in order to support the workflow application. The presented numbers underpin the value of OASIS. For example, a manual identification of 152 improper modeling elements or a manual elaboration of fault trees with 245 nodes would have been time-consuming and error-prone tasks.

8.3. Assessment of Benefits

Based on the case study of HEV development, this section evaluates the benefits of OASIS' modules with respect to the application of the safety engineering workflow by a safety engineer.

	Selective Interface	Property Checker	Model Corrector	View Provider	FTA Generator	FMEA Generator	ASIL Allocator	Report Generator	Simulink Model Generator	Safety Driver Generator	
Model Creation	2	3	3	1	-	-	1	-	-	1	11
Model Reuse	3	-	-	-	-	-	-	-	-	-	3
Model Maintenance	-	3	3	-	-	-	1	-	-	1	8
Model Consistency	-	3	3	2	2	2	2	3	3	3	23
Model Completeness	1	2	1	-	1	1	1	1	1	1	10
Model Examination	-	2	2	3	3	3	3	1	-	-	17
Application of Analyses	2	2	2	3	3	3	3	2	-	-	20
Application of Reviews	-	-	-	2	2	2	2	3	-	2	13
Communication	-	-	-	2	2	2	2	3	-	1	12
Software Development	-	1	1	1	-	-	-	2	3	3	11
	8	15	15	14	13	13	15	15	7	12	

Table 4: Depending on evaluation criteria, the achieved benefits of OASIS' modules were assessed and classified as *high (3)*, *medium (2)*, *low (1)* or *not applicable (-)* based on experience from the HEV use case.

First, evaluation criteria were defined to allow a systematic evaluation. The criteria *Model Creation*, *Model Reuse* and *Model Maintenance* concern the ability to efficiently elaborate an EAST-ADL model by creating and manipulating modeling elements. The criteria *Model Consistency* and *Model Completeness* are related to the capability of assessing the model with respect to particular characteristics indicating quality. The criteria *Model Examination*, *Application of Analyses*, *Application of Reviews* and *Communication* concern the ability of validating the information contained in the model. Finally, the criterion *Software Development* describes the capability of bridging the gap between system and software development activities.

Second, the achieved benefits of OASIS' modules were assessed with respect to the evaluation criteria. The achieved benefits were classified as *high*

(3), *medium (2)*, *low (1)* or *not applicable (-)*. The results of the assessment are presented in Table 4.

The best results were achieved with respect to the evaluation criteria (1) *Model Consistency*, (2) *Application and Analyses* and (3) *Model Examination.* Considering the complexity of an HEV powertrain, this is justified by the fact that (1) the elaboration of a consistent model, (2) the manual creation of means that support analyses and (3) the examination of particular details are time-consuming and error-prone tasks.

According to the results, the most valuable OASIS modules were (a) the Property Checker, (b) the Model Corrector, (c) the Report Generator and (d) the ASIL Allocator. Considering the complexity of an HEV powertrain, this is justified by the fact that (a,b,c) provide support for a lot of phases of the presented workflow and (d) replaces the manual application of a highly safety-relevant development step with an automatic and optimal application of the same working step.

9. Conclusion

The paper presents a novel software tool named OASIS (Aut**O**motive **A**nalysis and **S**afety Eng**I**neering In**S**trument). OASIS supports the application of a safety engineering workflow aligned with the automotive safety standard ISO 26262. OASIS sustains the creation of consistent and complete of work products and simplifies and automates workflow steps. This is achieved by providing support for (a) model creation and reuse, (b) analysis and documentation and (c) configuration and code generation. Based on the use of metrics and evaluation criteria, the case study of an hybrid electric vehicle development was used to demonstrate the effectiveness of the approach. OASIS was able to cope with the complexity imposed by a hybrid electric vehicle powertrain and greatly eased the application of the workflow. It turned out that OASIS is especially useful for achieving model consistency, applying analyses and examining the model.

Acknowledgment

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

References

[1] International Organization for Standardization, ISO 26262 Road vehicles - Functional safety (2011).

[2] N. G. Leveson, Safeware: system safety and computers, Addison-Wesley Publishing Company, 1995.

[3] P. Johannessen, F. Törner, J. Torin, Actuator Based Hazard Analysis for Safety Critical Systems, in: Proc. of the 23th International Conference on Computer Safety, Reliability and Security, 2004, pp. 130–141.

[4] P. Jesty, D. Ward, R. Rivett, Hazard Analysis for Programmable Automotive Systems, in: Proc. of the 2nd IET International Conference on System Safety 2007, 2007, pp. 106–111.

[5] H. Schubotz, Hazard Analysis and Risk Assessment for Complex EE-Architectures, in: Proc. of the SAE World Congress & Exhibition, no. 2010-01-0029, 2010.

[6] M. Stringfellow, N. Leveson, B. Owens, Safety-Driven Design for Software-Intensive Aerospace and Automotive Systems, Proceedings of the IEEE 98 (2010) 515–525.

[7] H. Giese, M. Tichy, D. Schilling, Compositional Hazard Analysis of UML Component and Deployment Models, in: Proc. of the 23th International Conference on Computer Safety, Reliability and Security, 2004, pp. 166–179.

[8] K. Allenby, T. Kelly, Deriving Safety Requirements Using Scenarios, in: Proc. of the 5th IEEE International Symposium on Requirements Engineering, 2001, pp. 228–235.

[9] A. Sandberg, D. Chen, H. Lönn, R. Johansson, L. Feng, M. Törngren, S. Torchiaro, R. T. Kolagari, A. Abele, Model-Based Safety Engineering of Interdependent Functions in Automotive Vehicles Using EAST-ADL2, in: Proc. of the 29th International Conference on Computer Safety, Reliability and Security, 2010, pp. 332–346.

[10] A. Lanusse, Y. Tanguy, H. Espinoza, C. Mraidha, S. Gerard, P. Tessier, R. Schnekenburger, H. Dubois, F. Terrier, Papyrus UML: an open source toolset for MDA., in: Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009), 2009, pp. 1–4.

[11] D. Makartetskiy, D. Pozza, R. Sisto, An Overview of Software-based Support Tools for ISO 26262, in: Proc. of the 3rd International Workshop on Innovation in Information Technologies: Theory and Practice, 2010, pp. 1–6.

[12] ATESST2 Project Consortium, EAST-ADL Domain Model Specification, version 2.1, Release Candidate 3 (2010).

[13] Y. Papadopoulos, M. Maruhn, Model-Based Synthesis of Fault Trees from Matlab - Simulink models, in: Proc. of the International Conference on Dependable Systems and Networks (DSN 2001), 2001, pp. 77–82.

[14] Y. Papadopoulos, C. Grante, Evolving car designs using model-based automated safety analysis and optimisation techniques, The Journal of Systems and Software 76 (2004) 77–89.

[15] M. de Miguel, J. Briones, J. Silva, A. Alonso, Integration of safety analysis in model-driven software development, IET Software 2 (2008) 260–280.

[16] D. Domis, M. Trapp, Integrating Safety Analyses and Component-Based Design, in: Proc. of the 27th International Conference on Computer Safety, Reliability and Security, 2008, pp. 58–71.

[17] J. Elmqvist, S. Nadjm-Tehrani, Tool Support for Incremental Failure Mode and Effects Analysis of Component-Based Systems, in: Proc. of the Design, Automation and Test in Europe Conference and Exhibition (DATE'08), 2008, pp. 921–927.

[18] A. Majdara, T. Wakabayashi, A New Approach for Computer-Aided Fault Tree Generation, in: Proc. of the 3rd Annual IEEE Systems Conference, 2009, pp. 308–312.

[19] M. Biehl, C. DeJui, M. Törngren, Integrating Safety Analysis into the Model-based Development Toolchain of Automotive Embedded Systems, in: Proc. of the Conference on Languages, Compilers and Tools for Embedded Systems, 2010, pp. 125–131.

[20] Y. Papadopoulos, M. Walker, M.-O. Reiser, M. Weber, D. Chen, M. Törngren, D. Servat, A. Abele, F. Stappert, H. Lönn, L. Berntsson, R. Johansson, F. Tagliabo, S. Torchiaro, A. Sandberg, Automatic Allocation of Safety Integrity Levels, in: Proc. of the 1st Workshop on Critical Automotive applications (CARS'10), 2010, pp. 7–10.

[21] P. Bieber, R. Delmas, C. Seguin, DALculus - Theory and Tool for Development Assurance Level Allocation, in: Proc. of the International Conference on Computer Safety, Reliability and Security (SafeComp), 2011, pp. 43–56.

[22] S. Sentilles, A. Pettersson, I. Crnkovic, Safe-IDE - A Tool for Design, Analysis and Implementation of Component-Based Embedded Systems, in: Proc. of the 31st IEEE International Conference on Software Engineering (ICSE), 2009, pp. 607–610.

[23] R. Bartosinski, Z. Hanzálek, P. Stružka, L. Waszniowski, Integrated Environment for Embedded Control Systems Design, in: Proc. of the IEEE International Symposium on Parallel and Distributed Processing Symposium (IPDPS), 2007, pp. 1–8.

[24] C.-J. Sjöstedt, J. Shi, M. Törngren, D. Servat, D. Chen, V. Ahlsten, H. Lönn, Mapping Simulink to UML in the design of embedded systems: Investigating scenarios and transformations, in: Proc. of the OMER 4 Workshop, 2008.

[25] T. Farkas, E. Meiseki, C. Neumann, K. Okano, A. Hinnerichs, S. Kamiya, Integration of UML with Simulink into embedded software engineering, in: Proc. of the ICROS-SICE International Joint Conference, 2009, pp. 474–479.

[26] M. Biehl, C.-J. Sjöstedt, M. Törngren, A Modular Tool Integration Approach - Experiences from two Case Studies, in: Proc. of the 3rd Workshop on Model-Driven Tool & Process Integration (MDTPI), 2010, pp. 19–30.

[27] M. Psarakis, D. Gizopoulos, E. Sanchez, M. Reorda, Microprocessor Software-Based Self-Testing, IEEE Design & Test of Computers 27 (2010) 4–19.

[28] D. Harel, B. Rumpe, Meaningful Modeling: What's the Semantics of "Semantics"?, IEEE Transactions on Computers 37 (2004) 64–72.

[29] The MathWorks, Inc., Simulink - Simulation und Model-Based Design (Feb. 2012).
URL `www.mathworks.de/products/simulink`

[30] dSPACE GmbH, TargetLink (Feb. 2012).
URL `www.dspace.com/en/inc/home/products/sw/pcgs/targetli.cfm`

[31] Texas Instruments Inc., Texas Instruments Website (Feb. 2012).
URL `www.ti.com`

[32] Freescale Semiconductor Inc., Freescale Website (Feb. 2012).
URL `www.freescale.com`

[33] ARM Inc., Keil μVision (Feb. 2012).
URL `www.keil.com/uvision`

[34] R. Mader, G. Grießnig, A. Leitner, C. Kreiner, Q. Bourrouilh, E. Armengaud, C. Steger, R. Weiß, A Computer-Aided Approach to Preliminary Hazard Analysis for Automotive Embedded Systems, in: Proc. of the IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS), 2011, pp. 169–178.

[35] R. Mader, E. Armengaud, A. Leitner, C. Kreiner, Q. Bourrouilh, G. Grießnig, C. Steger, R. Weiß, Computer-Aided PHA, FTA and FMEA for Automotive Embedded Systems, in: Proc. of the International Conference on Computer Safety, Reliability and Security (SafeComp), 2011, pp. 113–127.

[36] International Electrotechnical Commission, IEC 61025 - Ed. 2.0 Fault tree analysis (FTA) (2006).

[37] J. P. Ignazio and T. M. Cavalier, Linear Programming, Prentice Hall, 1994.

[38] Infineon Technologies AG, Infineon Website (Feb. 2012).
URL `www.infineon.com`

[39] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, M. Peri, S. Pezzini, Fault-tolerant Platforms for Automotive Safety-Critical Applications, in: Proc. of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES'03), ACM, 2003, pp. 170–177.

[40] S. Brewerton, R. Schneider, D. Eberhard, Implementation of a Basic Single-Microcontroller Monitoring Concept for Safety Critical Systems on a Dual-Core Microcontroller, SAE SP (2121) (2007) 25–32.

[41] Texas Instruments, Safety Manual - TMS570LS20216S Device (2010).

[42] M. Ehsani, Y. Gao, S. Gay, A. Emadi, Hybrid Electric Vehicles, in: Modern Electric, Hybrid Electric, and Fuel Cell Vehicles Fundamentals, Theory, and Design, CRC Press, 2005, Ch. 5.

Bibliography

[1] D. Harel and B. Rumpe. Meaningful Modeling: What's the Semantics of "Semantics"? *IEEE Transactions on Computers*, 37:64–72, October 2004.

[2] International Organization for Standardization. ISO 26262 Road vehicles - Functional safety, 2011.

[3] N. G. Leveson. *Safeware: system safety and computers.* Addison-Wesley Publishing Company, 1995.

[4] M. Ehsani, Y. Gao, S.E. Gay, and A. Emadi. Hybrid Electric Vehicles. In *Modern Electric, Hybrid Electric, and Fuel Cell Vehicles Fundamentals, Theory, and Design*, chapter 5. CRC Press, 2005.

[5] AVL List GmbH. Turbohybrid Website, September 2012.

[6] D. Ward. System safety in electric and hybrid electric vehicles. In *Proc. of the Australian System Safety Conference (ASSC)*, pages 79–83, 2011.

[7] C. Ebert and C. Jones. Embedded Software: Facts, Figures, and Future. *IEEE Computer*, 42(4):42–52, 2009.

[8] P. Johannessen, F. Törner, and J. Torin. Actuator Based Hazard Analysis for Safety Critical Systems. In *Proc. of the 23th International Conference on Computer Safety, Reliability and Security*, pages 130–141, September 2004.

[9] F. Törner, P. Johannessen, and P. Öhman. Assessment of Hazard Identification Methods for the Automotive Domain. In *Proc. of the 25th International Conference on Computer Safety, Reliability and Security*, pages 247–260, September 2006.

[10] P.H. Jesty, D.D. Ward, and R.S. Rivett. Hazard Analysis for Programmable Automotive Systems. In *Proc. of the 2nd IET International Conference on System Safety 2007*, pages 106–111, December 2007.

[11] H. Schubotz. Hazard Analysis and Risk Assessment for Complex EE-Architectures. In *Proc. of the SAE World Congress & Exhibition*, number 2010-01-0029, April 2010.

[12] M.V. Stringfellow, N.G. Leveson, and B.D. Owens. Safety-Driven Design for Software-Intensive Aerospace and Automotive Systems. *Proceedings of the IEEE*, 98:515–525, 2010.

[13] J.B. Michael, Man-Tak Shing, K.J. Cruickshank, and P.J. Redmond. Hazard Analysis and Validation Metrics Framework for System of Systems Software Safety. *IEEE Systems Journal*, 4:186–197, 2010.

[14] S.P. Kumar, P.S. Ramaiah, and V. Khanaa. A Methodology for Building Safer Software based Critical Computing Systems. In *Proc. of the 2nd IEEE International Conference on Advance Computing (IACC'2010)*, pages 422–429, February 2010.

[15] H. Giese, M. Tichy, and D. Schilling. Compositional Hazard Analysis of UML Component and Deployment Models. In *Proc. of the 23th International Conference on Computer Safety, Reliability and Security*, pages 166–179, September 2004.

[16] K. Allenby and T. Kelly. Deriving Safety Requirements Using Scenarios. In *Proc. of the 5th IEEE International Symposium on Requirements Engineering*, pages 228–235, August 2001.

[17] A. Sandberg, D. Chen, H. Lönn, R. Johansson, L. Feng, M. Törngren, S. Torchiaro, R. Tavakoli Kolagari, and A. Abele. Model-Based Safety Engineering of Interdependent Functions in Automotive Vehicles Using EAST-ADL2. In *Proc. of the 29th International Conference on Computer Safety, Reliability and Security*, pages 332–346, September 2010.

[18] C. Hamoy, D. Hemer, and P. Lindsay. HazLog: tool support for hazard management. In *Proc. of the 9th Australian workshop on Safety critical systems and software (SCS)*, pages 77–87, 2004.

[19] A. Lanusse, Y. Tanguy, H. Espinoza, C. Mraidha, S. Gerard, P. Tessier, R. Schnekenburger, H. Dubois, and F. Terrier. Papyrus UML: an open source toolset for MDA. In *Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009)*, pages 1–4, June 2009.

[20] ATESST2 Project Consortium. Refined EAST-ADL2 tool support. Technical report, 2010. Deliverable D3.2.

[21] D. Makartetskiy, D. Pozza, and R. Sisto. An Overview of Software-based Support Tools for ISO 26262. In *Proc. of the 3rd International Workshop on Innovation in Information Technologies: Theory and Practice*, pages 1–6, September 2010.

[22] K.K. Vemuri, J.B. Dugan, and K.J. Sullivan. Automatic Synthesis of Fault Trees for Computer-Based Systems. *IEEE Transactions on Reliability*, 48:394–402, December 1999.

[23] Y. Papadopoulos and M. Maruhn. Model-Based Synthesis of Fault Trees from Matlab - Simulink models. In *Proc. of the International Conference on Dependable Systems and Networks (DSN 2001)*, pages 77–82, July 2001.

[24] Y. Papadopoulos and C. Grante. Evolving car designs using model-based automated safety analysis and optimisation techniques. *The Journal of Systems and Software*, 76:77–89, June 2004.

[25] M. Biehl, C. DeJui, and M. Törngren. Integrating Safety Analysis into the Model-based Development Toolchain of Automotive Embedded Systems. In *Proc. of the Conference on Languages, Compilers and Tools for Embedded Systems (LCTES)*, pages 125–131, 2010.

[26] G.J. Pai and J.B. Dugan. Automatic Synthesis of Dynamic Fault Trees from UML System Models. In *Proc. of the 13th International Symposium on Software Reliability Engineering (ISSRE)*, pages 243–254, 2002.

[27] M.A. de Miguel, J.F. Briones, J.P. Silva, and A. Alonso. Integration of safety analysis in model-driven software development. *IET Software*, 2:260–280, 2008.

[28] D. Domis and M. Trapp. Integrating Safety Analyses and Component-Based Design. In *Proc. of the 27th International Conference on Computer Safety, Reliability and Security*, pages 58–71, September 2008.

[29] J. Elmqvist and S. Nadjm-Tehrani. Tool Support for Incremental Failure Mode and Effects Analysis of Component-Based Systems. In *Proc. of the Design, Automation and Test in Europe Conference and Exhibition (DATE'08)*, pages 921–927, April 2008.

[30] A. Majdara and T. Wakabayashi. A New Approach for Computer-Aided Fault Tree Generation. In *Proc. of the 3rd Annual IEEE Systems Conference*, pages 308–312, 2009.

[31] G. Latif-Shabgahi and F. Tajarrod. A New Approach for the Construction of Fault Trees from System Simulink. In *Proc. of the International Conference on Availability. Reliability and Security (ARES)*, pages 712–717, 2009.

[32] M. Bozzano, A. Cimatti, J. Katoen, V. Nguyen, T. Noll, M. Roveri, and R. Wimmer. A Model Checker for AADL. In *Proc. of the 22nd International Conference on Computer Aided Verification (CAV)*, pages 562–565, 2010.

[33] M.-A. Esteve, J.-P. Katoen, V. Nguyen, B. Postma, and Y. Yushtein. Formal Correctness, Safety, Dependability, and Performance Analysis of a Satellite. In *Proc. of the 13th International Symposium on Software Reliability Engineering (ISSRE)*, pages 1022–1031, 2012.

[34] C. Lauer, R. German, and J. Pollmer. Fault Tree Synthesis from UML Models for Reliability Analysis at Early Design Stages. *ACM SIGSOFT Software Engineering Notes*, 36:1–8, January 2011.

[35] J. Xiang, K. Yanoo, Y. Maeno, and K. Tadano. Automatic Synthesis of Static Fault Trees from System Models. In *Proc. of the 5th International Conference on Secure Software Integration and reliability Improvement*, pages 127–136, 2004.

[36] Y. Papadopoulos, M. Walker, M.-O. Reiser, M. Weber, D. Chen, M. Törngren, D. Servat, A. Abele, F. Stappert, H. Lönn, L. Berntsson, R. Johansson, F. Tagliabo, S. Torchiaro, and A. Sandberg. Automatic Allocation of Safety Integrity Levels. In *Proc. of the 1st Workshop on Critical Automotive applications (CARS'10)*, pages 7–10, April 2010.

[37] P. Bieber, R. Delmas, and C. Seguin. DALculus - Theory and Tool for Development Assurance Level Allocation. In *Proc. of the International Conference on Computer Safety, Reliability and Security (SafeComp)*, pages 43–56, 2011.

[38] M. Psarakis, D. Gizopoulos, E. Sanchez, and M.S. Reorda. Microprocessor Software-Based Self-Testing. *IEEE Design & Test of Computers*, 27:4–19, 2010.

[39] J.U. Gärtner. Certified Software Factory: Open Software Toolsuites, Safe Methodologies and System Architectures. In *Proc. of the 11th Australian Workshop on Safety Related Programmable Systems (SCS)*, pages 19–22, 2006.

[40] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert. From Simulink to SCADE/Lustre to TTA: a Layered Approach for Distributed Applications. In *Proc. of the Conference on Languages, Compilers and Tools for Embedded Systems (LCTES)*, pages 153–162, 2003.

[41] N. Scaife, C. Sofronis, P. Caspi, S. Tripakis, and F. Maraninchi. Defining and Translating a ŞafeŠubset of Simulink/Stateflow into Lustre. In *Proc. of the International Conference on Embedded Software (EMSOFT)*, pages 259–268, 2004.

[42] R. Venky, S. Ulka, A. Kulkarni, and P. Bokil. STATEMATE to SCADE Model Translation. In *Proc. of the 1st India Software Engineering Conference (ISEC)*, pages 145–146, 2008.

[43] S. Burmester, H. Giese, M. Hirsch, D. Schilling, and M. Tichy. The Fujaba Real-Time Tool Suite: Model-Driven Development of Safety-Critical, Real-Time Systems. In *Proc. of the International Conference on Software Engineering (ICSE)*, pages 670–671, 2005.

[44] C. Buckl, A. Knoll, and G. Schrott. Model-Based Development of Fault-Tolerant Embedded Software. In *Proc. of the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, pages 103–110, 2006.

[45] L. Brisolara, M. Oliveira, R. Redin, L. Lamb, L. Carro, and F. Wagner. Using UML as Front-end for Heterogeneous Software Code Generation Strategies. In *Proc. of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 504–509, 2008.

[46] G. Lasnier, B. Zalila, L. Pautet, and J. Hugues. OCARINA: An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications. In *Proc. of the Ada-Europe International Conference on Reliable Software Technologies (Ada-Europe)*, pages 237–250, 2009.

[47] E. Borde, P.H. Feiler, Haïk, and L. Pautet. Model Driven Code Generation for Critical and Adaptative Embedded Systems. *ACM SIGBED Review - Special Issue on the 2nd International Workshop on Adaptive and Reconfigurable Embedded Systems*, 6(3), 2009.

[48] S. Sentilles, A. Pettersson, and I. Crnkovic. Safe-IDE - A Tool for Design, Analysis and Implementation of Component-Based Embedded Systems. In *Proc. of the 31st*

IEEE International Conference on Software Engineering (ICSE), pages 607–610, May 2009.

[49] E. Borde and J. Carlson. Towards Verified Synthesis of ProCom, a Component Model for Real-Time Embedded Systems. In *Proc. of the International ACM Sigsoft Symposium on Component Based Software Engineering (CBSE)*, pages 129–138, 2011.

[50] M. Eberl, M. Glaß, J. Teich, and U. Abelein. Considering Diagnosis Functionality during Automatic System-Level Design of Automotive Networks. In *Proc. of the Annual Design Automation Conference (DAC)*, pages 205–213, 2012.

[51] J. Shen and J.A. Abraham. Native Mode Functional Test Generation for Processors with Applications to Self Test and Design Validation. In *Proc. of the International Test Conference (ITC'98)*, pages 990–999, October 1998.

[52] K. Jayaraman, V.M. Vedula, and J.A. Abraham. Native Mode Functional Self-Test Generation for Systems-on-Chip. In *Proc. of the International Symposium on Quality Electronic Design (ISQED'02)*, pages 280–285, August 2002.

[53] P. Parvathala, K. Maneparambil, and W. Lindsay. FRITS - A Microprocessor Functional BIST Method. In *Proc. of the International Test Conference (ITC'02)*, pages 590–598, December 2002.

[54] F. Corno, E. Sanchez, M.S. Reorda, and G. Squillero. Automatic Test Program Generation: A Case Study. *IEEE Transactions on Design & Test of Computers*, 21:102–109, 2004.

[55] I. Bayraktaroglu, J. Hunt, and D. Watkins. Cache Resident Functional Microprocessor Testing: Avoiding High Speed IO Issues. In *Proc. of the IEEE International Test Conference (ITC)*, pages 1–7, 2006.

[56] The MathWorks, Inc. Simulink - Simulation und Model-Based Design, February 2012.

[57] dSPACE GmbH. TargetLink, February 2012.

[58] Texas Instruments Inc. Texas Instruments Website, February 2012.

[59] Freescale Semiconductor Inc. Freescale Website, February 2012.

[60] ARM Inc. Keil μVision, February 2012.