Dipl.-Ing. Ralph Peter Weissnegger, BSc

# Design and Verification Process for Safety-Critical Embedded Systems in the Automotive Domain

**DOCTORAL THESIS**

to achieve the university degree of

Doktor der technischen Wissenschaft

submitted to

**Graz University of Technology**

Supervisor

Univ.-Prof. Dipl.-Inform. Dr.sc.ETH Kay Römer

Institute of Technical Informatics

Advisor

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger

Graz, May 2017

## AFFIDAVIT

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 indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date

Signature

# Acknowledgments

This thesis has been carried out at the Institute of Technical Informatics at the Graz University of Technology, in cooperation with the industrial partner CISC Semiconductor GmbH. I would like to thank CISC, especially Dr. Markus Pistauer, for making my PhD study possible and for letting me participate in very interesting industrial projects. I would like to thank my supervisor Prof. Kay Uwe Römer for being open for discussions and for excellently supporting the official and organizational part of my PhD study. Furthermore, my thanks go to Dr. Damian Dalton who kindly agreed to serve as a second adviser for my thesis. In particular, I would like to express my gratitude to my mentor Ass.-Prof. Dr. Christian Steger. He not only managed the organizational framework, made great journeys to interesting conferences possible, gave me a lot of freedom regarding my research topics, and helped me to produce the scientific work, but he also supported me spiritually and morally. I also want to thank my students Markus Schuß and Martin Schachner for their great work and support during the project. Additionally, I would like to thank my colleagues who made my work days easier and more enjoyable.

Finally, I would like to thank my family for their lifelong support and their tolerance and encouragement during my whole studies. They always pointed me in the right direction, made life easier through helpful discussions and showed me what's really important in life. Another big shout-out goes out to my friends and former flatmates, which have always been there for me, for all the amazing (sport-) trips, journeys, come2gethers, ...
...resulting in unforgettable memories.

*Graz, May, 2017*
*Ralph Weissnegger*

# Abstract

The modern automotive market is heading towards fully automated self-driving cars. Following this evolution, the number of new assistance features for ensuring safe and reliable operations is rising. The sensing and controlling of these systems is the work of the highly distributed and connected electronic control units (ECU) and it is no surprise that up to 100 of these microcontrollers are integrated in the car of today. Moreover, these systems must be developed in compliance with various automotive standards to ensure dependability such as the ISO26262 (functional safety for road vehicles). Thus the design and verification of electric/electronic systems is becoming increasingly complex. Current tools and design flows hit the limits of complexity and are therefore not capable to efficiently address software and hardware design and optimization in a joint way. Furthermore, the technological, organizational and design gap in today's flows are not covered by current methods and tools. The increasing effort in system design verification through dynamically changing design makes the development very cost intensive and SoC integration unaffordable for many applications. This situation is even getting worse as the number of extra-functional properties, such as safety and security, and their importance, is increasing. To overcome these issues in the design and verification of safety-critical systems, we developed novel verification methodologies, embedded in a complete design process named SaVeSoC (Safety Aware Virtual Prototype Generation and Evaluation of a System on Chip). The SaVeSoC process defines a design methodology especially for safety-critical systems and is based on a standardized modeling language for real-time and embedded systems (UML/MARTE). A model-based approach eases the communication between different stakeholders, provides different views and serves as a central storage of information. By applying simulation-based verification including virtual prototyping, quantified reliability analysis and hardware evaluation on a seamless design process, we were able to speed-up the verification and to reduce the tools involved; resulting in completeness, correctness and consistency of the entire system. The whole design process, including the novel verification methodologies, has been implemented in the design and verification framework SHARC (Simulation and Verification of Hierarchical Embedded Microelectronic Systems) and applied to an industrial use-case of a battery management system (BMS). The complete approach has been evaluated by responsible design experts for safety-critical development and verification.

# Zusammenfassung

Der moderne Automobilmarkt bewegt sich hin in Richtung selbstfahrende vollautomatisierte Fahrzeuge. Aufgrund dieser Entwicklung steigt die Anzahl neuer Fahrerassistenzsysteme um ein sicheres und zuverlässiges Verhalten, zu garantieren. Da die Kundenanforderungen an Assistenzsystemen enorm steigen, ist es auch keine Überraschung, dass bis zu 100 elektronischer Steuergeräte (ECUs) in heutigen Autos integriert sind. Darüber hinaus müssen diese Systeme auch hinsichtlich der Anforderungen verschiedenster Automobilstandards wie dem Sicherheitsstandard für funktionelle Sicherheit bei Straßenfahrzeugen (ISO26262) erfüllt und entwickelt werden, um die Zuverlässigkeit zu gewährleisten. Aufgrund dieser Tatsachen wird das Entwickeln und Verifizieren von elektrischen/elektronischen Systemen immer komplexer. Aktuelle Werkzeuge und Entwicklungsprozesse stoßen an die Grenzen der Komplexität und sind daher nicht in der Lage, Software- und Hardware-Design und dessen Optimierung in einer gemeinsamen Weise effizient zu adressieren. Darüber hinaus sind die technologischen, organisatorischen und Entwicklungslücken in heutigen Prozessen durch aktuelle Methoden und Werkzeuge nicht abgedeckt. Eine Abnahme der Komplexität ist nicht in Sicht, da die Anzahl der außer-funktionalen Eigenschaften, wie Safety und Security, und deren Bedeutung, zunimmt. Um die Barrieren im Design von sicherheitskritischen Systemen und dessen Verifikation zu überwinden, entwickelten wir neuartige Verifikationsmethoden, die zudem in einem kompletten Designprozess namens SaVeSoC (Safety Aware Virtual Prototype Generation und Evaluation eines Systems on Chip) eingebettet sind. Der SaVeSoC-Prozess definiert eine Designmethodik speziell für sicherheitskritische Systeme und basiert auf standardisierten Modellierungssprachen für Echtzeit- und eingebettete Systeme (UML/MARTE). Durch das Anwenden simulationsbasierter Verifikation einschließlich virtuellem Prototyping, quantifizierter Zuverlässigkeitsanalyse und Hardware-Evaluierung in einem durchgehenden Designprozess, erreichten wir eine Beschleunigung der Verifikation und die Reduzierung der beteiligten Werkzeuge. Das Anwenden dieses Ansatzes führt zu Vollständigkeit, Korrektheit und Konsistenz des gesamten Systems. Der gesamte Designprozess, einschließlich der neuartigen Verifikationsmethoden, wurde im Design- und Verifikations-Framework SHARC (Simulation and Verification of Hierarchical Embedded Microelectronic Systems) implementiert und auf einen industriellen Anwendungsfall eines Batteriemanagementsystems (BMS) angewendet. Der komplette Ansatz wurde von verantwortlichen Designexperten für sicherheitskritische Entwicklung und Verifikation bewertet.

# Executive Summary

Over the past decades, we have been experiencing an increasing amount of electric/electronic (E/E) systems in the automotive domain. Today, we even speak about the electrification in the automotive field; thus we are heading towards fully electric cars which even speed up the growth in E/E systems. A modern car has up to 100 embedded Electronic Control Unit (ECU) [18] right now, and the number is even rising when we think about the emerging need for new assistance features in cars which help us to facilitate the vision towards the last stage of autonomous driving levels, stage 5. This stage refers to a fully autonomous system where the performance of a vehicle equals the experience of an adult human driver. This will be a major challenge for future vehicle development. The complex assistance features cannot properly work if these systems do not have a strong interaction (in-vehicle communication) and also inter-dependencies between them. Nowadays, we go even beyond the internal border of a car body for communication with a complex environment. For advanced driver features, modern cars have to communicate with each other (V2V - vehicle to vehicle), with the infrastructure (V2I - vehicle to infrastructure), but also with the cloud (vehicle2cloud) and other complex IoT devices (V2X). In the near future, this will enable a growth of new applications in the field of SmartCities and SmartMobility. This, in turn, raises also the amount of software in today's cars, which can lead up to a total of 150 million lines of code, which is vastly more than a Boeing 787 jet airliner [30]. Testing and verification of the software, hardware and complete systems in early stages is a bigger challenge in today's development. It raises the complexity level in the design, development and verification of complex systems and imposes an enormous effort for engineers in developing their applications. To fully test an autonomous car, it is predicted that a theoretical test track would need a length up to 88 times to the sun and back [66]. This utopistic distance could only be reached employing advanced simulation methodologies. Moreover, these systems must fulfill conformance to different automotive standards to guarantee dependability. In terms of safety, these systems must fulfill standards such as ISO26262 (functional safety for road vehicles), [42]. Since this standard is now treated as state of the art in court (2011), OEMs and their suppliers are required to develop and test their systems towards the recommended measures and methods. ISO 26262 is the functional safety standard for safety-critical systems in road vehicles. This standard addresses possible hazards caused by malfunctioning behavior of safety-related systems. It also provides an automotive safety lifecycle that covers safety aspects throughout the whole design and development process of modern products and systems. Depending on the Automotive Safety Integrity Level (ASIL) the system must be developed and tested

according to different recommended measures and methods (e.g. FMEA, FTA, simulation, virtual prototyping) and must achieve a high level of reliability (e.g. low failure rates in hardware evaluation). Current design tools and flows, especially regarding safety-critical systems, hit the limits of complexity and therefore are not capable to efficiently address software and hardware design in a joint way. Moreover, the technological, organizational and design gap between the different abstraction levels that exists in today's development processes are not covered by current methods and tools. Besides the complexity and heterogeneity, also different kinds of requirements and constraints associated with the design of embedded systems must be taken into account. Extra-functional properties such as timing performance, power consumption or reliability are only partially addressed by disjoint special flows. The number of tools involved to cover all these properties is rising, thus leading to a complex toolchain where important information is distributed over several disjoint tools and organizations. Information transfer between these tools leads to inconsistency, incorrectness and incompleteness of the entire design. The increasing effort in system design verification through dynamically changing design makes the development very cost intensive and SoC integration unaffordable for many applications. This situation is even getting worse as the number of extra-functional properties, such as safety and security, and their importance, is increasing. Furthermore, the expectations towards system reliability are also significantly increasing or even become stringent in applications such as automotive or transportation. A further demand for more features (X-by-wire) and improved functionality (ADAS, autonomous driving) increases the complexity of future systems dramatically. Without advanced methods and tools, the effort in tackling these issues leads to an increase in development costs and time, thus missing the time-to-market window.

To overcome these issues in the design and verification of safety-critical systems, we developed novel verification techniques, embedded in a complete design process named SaVeSoC (Safety Aware Virtual Prototype Generation and Evaluation of a System on Chip). The SaVeSoC process defines a design methodology especially for safety-critical systems. This methodology is based on a standardized modeling language for real-time and embedded systems and is derived from the Unified Modeling Language (UML) standard. The name of this modeling language is MARTE and relies on the Model Driven Architecture (MDA) design layers. Since these layers lacked a proper definition, our contribution was to introduce an additional layer where safety aspects in the design can be seamlessly defined. By using structural and behavioral diagrams in combination with MARTE, important properties, such as timing behavior of safe states, but also resource management for several applications running on one ECU can be described analyzed. Furthermore, we show how different reliability analysis such as FMEA or FTA can be applied to the model-based approach throughout the design process. For the specification of the detailed hardware, we used a standard, which is very well known in the industry, named IP-XACT.

By connecting the structural system level components to executable models, we are able to describe the behavior of the system in a more sophisticated way. This has been

done by defining a mapping between MARTE and SystemC. Since these two approaches share the same philosophy to start from a first system design down to a detailed hardware and software description, it was convenient to combine these two languages. The advantage of this approach is that no information must be transferred to other tools and the UML files retain as a single source of information. Furthermore, the configuration and parametrization of the simulation can be done using the defined MARTE properties.

To increase and go beyond the level of functional coverage, we developed a methodology to automatically derive test benches from (safety-) requirements and the functional specification. These test benches can be reused throughout the whole development process. The (safety-) requirements have been defined in an extension for the UML2 profile named SysML. This approach has the advantage of automatically testing the design under various environmental conditions and in different design configurations.

Since this approach leads to a significantly high amount of simulation tasks, we developed a new method and defined a layered architecture pattern for the distribution of independent simulation tasks to a cloud-based environment. This leads up to near linear speed-up in the coverage process. Results from the system verification can directly be used for the hardware (safety-) requirements and hardware platform design.

The detailed hardware specification, derived from our simulation experiments, can now be evaluated regarding single-point, latent and random hardware faults. This evaluation is mandatory in ISO26262 to verify the reliability of safety-critical hardware. By reusing safety properties and combining this approach with a standardized hardware description, we achieved to evaluate the hardware in earlier phases of development. Moreover, we could decrease the time for evaluating safety-critical hardware based on model-driven design.

Using the same modeling language for the hardware description, we achieved to generate a first (safety aware) virtual prototype by reusing hardware components from an open library. This approach enables to test critical software much earlier in the design process by generating different hardware platforms on the fly. Thus software can be tested under different hardware configurations. Furthermore, our approach allows for a seamless integration into the functional specification of the system design by automatically generating proper interfaces, in order to verify its functionality.

The whole design process, including the novel methodologies, has been implemented in the design and verification framework Simulation and Verification of Hierarchical Embedded Microelectronic Systems (SHARC). This framework is based on Eclipse and the UML editor Papyrus. By using our Eclipse plugins, every Papyrus editor is capable of verifying safety-critical embedded systems in line with the requirements of the functional safety standard ISO26262.

The complete approach has been evaluated by responsible design experts for safety-critical development and verification. The evaluation has been done regarding time (design effort), quality of the design, and verification coverage. Furthermore, we present the results of the OpenES design flow evaluation, to show the overall benefits of the newly developed approaches.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

**ADC** Analog Digital Converter.

**ASIL** Automotive Safety Integrity Level.

**AUTOSAR** Automotive Open Systems Architecture.

**BMS** Battery Management System.

**CAN** Controller Area Network.

**CDV** Coverage Driven Verification.

**CIM** Computational Independent Model.

**CPU** Central Processing Unit.

**CRV** Constraint Random Verification.

**DSL** Domain Specific Language.

**DUT** Design Under Test.

**ECU** Electronic Control Unit.

**ESL** Electronic System Level.

**FEV** Fully Electric Vehicle.

**FIFO** First In First Out.

**IC** Integrated Circuit.

**IP** Intellectual Property.

**ISS** Instruction Set Simulator.

**LSF** Linear Signal Flow.

**MDA** Model Driven Architecture.

**MDE** Model Driven Engineering.

**MPSoC** MultiProcessor System on Chip.

**OMG** Object Management Group.

**OpenES** Open ESL Technologies for Next Generation Embedded Systems.

**OVP** Open Virtual Platform.

**PCB** Printed Circuit Board.

**PIM** Platform Independent Model.

**preAA** preliminary Architectural Assumption.

**PSM** Platform Specific Model.

**QEMU** Quick Emulator.

**RTL** Register Transfer Level.

**SaaS** Software as a Service.

**SaVeSoC** Safety Aware Virtual Prototyp Generation and Evaluation of a System on Chip.

**SCL** System Component Library.

**SHARC** Simulation and Verification of Hierarchical Embedded Microelectronic Systems.

**SiP** System in Package.

**SoC** System-on-a-Chip.

**SR** Safety Requirement.

**SyAD** System Architecture Designer.

**TCL** Tool Command Language.

**TDF** Timed Data Format.

**TLM** Transaction Level Modeling.

| | |
|---|---|
| **UML** | Unified Modeling Language. |
| **UVM** | Universal Verification Methodology. |
| **VLNV** | Vendor Library Name Version. |
| **VP** | Virtual Prototype. |
| **VSL** | Value Specification Language. |

# 1 Introduction

In the world of today, the amount of embedded electric/electronic (E/E) systems in various domains is increasing to a very great extent. When we think about the complexity of the past few years, it is apparent that new applications have emerged in which systems are not only interacting with each other but also have impact on the physical world, the so-called cyber-physical systems. Depending on their application, they must fulfill different requirements ranging from timing constraints, performance behavior, low power consumption, thermal or even working capability under different environmental conditions. The point here is, we live in a world where cyber-physical systems are ubiquitous. They have an impact on our daily lives and the malfunction of these systems can lead to severe damage or injury to people. We must thus ensure the dependability of these systems.

This is even more obvious when we turn to the automotive domain. It can be observed that there is a shift towards fully E/E systems resulting from the trend towards electric vehicles. In fact, a car is now more or less a smartphone on wheels. The sensing and controlling is the work of the highly distributed electronic control units (ECU), and it is no surprise, that through all these new features in cars, more than 100 of these microcontrollers [18] are currently integrated in a modern car. This situation also has an impact on the amount of software in cars today, which can total 150 million lines of code [30]. The communication between these ECUs even extends beyond the border of a car body. For complex assistance features, cars have to communicate with each other (V2V ) and also the infrastructure must be involved in the computation (V2I). Although, the goal and motivation in the development of a car stayed the same. The aim is to develop better, more reliable and safe products which reduce the number of deadly accidents. But the industry is facing new problems through the emergence of many new (assistance-) features that are also influencing each other. In turn, this raises the complexity level in the design, development and verification of complex systems and imposes an enormous effort for engineers in developing their applications. In terms of safety, these systems must fulfill standards such as ISO26262 (functional safety for road vehicles), [42]. Since this standard is now treated as state of the art in court, OEMs and their suppliers are required to develop and test their systems towards the recommended measures and methods. It is no longer sufficient to test single hardware or software components, the functionality of the whole system must be verified.

With the aim to increase the level of abstraction and automation, the Model Driven Engineering (MDE) approach has found its way into the development of systems and software. MDE has the benefit of being more flexible towards platform and staff changes, of reducing development and maintenance costs, increasing quality and moreover faster development cycles. However, this approach still faces some major challenges when thinking about tooling support. Compared to the landscape of programming languages MDE tools are weak. Due to the fact that developers are more into coding than modeling the acceptance of MDE approaches is still low. Furthermore, using MDE in bigger projects can lead to badly formed models, when using a too general purpose language or having too many domain-specific languages for modeling. Some modeling languages only cover a part of the development cycle, which means that companies have to invest effort and costs to support the rest of the life-cycle [12].

When we discuss the design of a system, a single modeling language immediately springs to mind, the Unified Modeling Language (UML), [35]. Having its roots in the software domain, UML paved the way and established a model-based thinking in various engineering domains, far across the borders of conventional software design. Since UML comes with several extensions such as MARTE [63], SysML [62] or EAST-ADL [26], engineers from different domains can use the full potential of an object-oriented approach. MARTE was introduced to overcome the enormous complexity issues in the design of real-time and embedded systems. It provides capabilities to model hardware, software as well as system design and provides the representation of timing, resource and performance behavior. Furthermore, UML/MARTE is already used for several reliability analysis techniques. Moreover, many semiconductor companies and suppliers are relying on this modeling language and it is used by several European projects such as the Open ESL Technologies for Next Generation Embedded Systems (OpenES) [16]. OpenES is a European initiative to fill the gaps in today's design flows and to develop common solutions to stay competitive on the world market.

Since today's state of the art car of today exists not only in one single version but rather in several hundreds of variants all with different features, each of which must be exhaustively tested to fulfill the standards. Millions of test kilometers must be driven to ensure the reliability of a car and it is neither economic nor safe to test them in a real environment [56]. Simulation plays an ever increasing and important role in the verification of the modern car. The virtual environment can easily be modified and the car can be represented in its different variations resulting in an economic advantage. Simulations can be done in early development phases, where the detailed implementation of a function is still undecided and based on platform-specific models where the hardware and software are explicitly defined (virtual prototype). Applied verification methods and tests can be monitored, reproduced and rerun every time. Another advantage of simulation is that it cannot only be run day and night but also massively in parallel. A specification and simulation language, which shares the same philosophy as the UML/MARTE approach, is SystemC [3]. Like UML, it shares the Model Driven Architecture (MDA) approach, starting from a higher-level system design (computational independent), down to hardware and software design.

In this work we present a novel design and development flow for a safety aware virtual prototype. This design flow conforms to the ISO26262 standard and meets all its requirements to produce a reliable product in the end. The whole system, from a first functional specification down to hardware design, is specified by standardized modeling languages. Before the virtual prototype is generated from the hardware specification, several reliability analysis methods are applied and executed on this specification. This allows an early evaluation of the hardware design regarding safety before testing the prototype in simulation. Furthermore, we show the integration of the generated virtual prototype into the system design, to verify its functionality and interfaces. The whole methodology is available through the developed design and verification framework named SHARC (Simulation and Verification of Hierarchical Embedded Microelectronic Systems), [19].

## 1.1 Thesis Background

This thesis was carried out in cooperation with the industrial partner CISC Semiconductor GmbH. CISC is an international company within the field of automotive and RFID based in Klagenfurt, Graz (Austria) and Mountain View, California (US). In the automotive domain, their expertise lies in supporting the development process by enhanced verification methodologies and tools. This includes system integration of automotive systems through the support of simulation, virtual prototyping or HW/SW co-simulation. Being involved in joint R&D activities with all major European car manufactures and its suppliers in the area of microelectronic systems, CISC is driving new activities to handle the increased system complexity by their simulation-based system design methodology and wide spread system know-how. This expertise is also honored by various international standardization bodies, to which CISC is continuously contributing. Thanks to their organizational independence CISC can easily adapt to customer needs and offer project-specific tailoring and scaling of development processes and to adapt to project-dependent requirements. Furthermore, this thesis has profited from the influence of two major European projects, OpenES [16] and eRamp [31]. The TU Graz and CISC Semiconductor closely cooperated with well-known companies in the field of semiconductors. This includes partners from universities, research organizations (RTO) and industry such as ST, NXP, Infineon, CEA-List or Verimag.

## 1.2 Problem Statement

This thesis aims at improving the model-based development and verification of embedded automotive safety-critical systems from a first initial system design down to hardware and software implementation. To gain a good understanding of the problems and complications that can occur during the development process, a fundamental factor is to get a comprehensive overview of the related work in this domain.

To cope with the high complexity in the integration of advanced embedded systems, the use of advanced methods and design tools is more relevant than ever. Besides the complexity and heterogeneity, different kinds of constraints and requirements associated with the design of embedded systems must be taken into account. These systems combine ever increasing complex software with complex hardware components (Intellectual Property (IP), Integrated Circuit (IC), System-on-a-Chip (SoC), System in Package (SiP), Printed Circuit Board (PCB)) and their design is confronted with the following issues:

- Current tools and flows hit the limits of complexity and therefore are not capable to efficiently address software and hardware design and optimization in a joint way.
- Increasing design effort makes SoC integration not affordable for many applications.
- The technological, organizational and design gap between different abstraction levels in today's development processes is not covered by current methods and tools.
- Extra-functional properties like timing performance, power consumption or reliability are only partially addressed by disjoint special flows. Inconsistency, inefficient design iterations, incorrectness, incompleteness and finally sub-optimal results are the consequences.
- Increased effort in system design verification through dynamically changing design.
- The number of tools involved in today's design flow is rising, thus leading to a complex tool-chain where important information is distributed over several disjoint tools and organizations.
- Exchange of information between simulation-based verification and reliability analysis throughout the whole development process, which are both highly required by the ISO26262 standard.
- Manual steps in the design and verification process without tool support.

This situation is even getting worse as the number of extra-functional properties such as safety or security and their importance is increasing. Furthermore, the expectations on system reliability are also significantly increasing or even become stringent in automotive or transportation applications. A further demand for more features (X-by-wire) and improved functionality (ADAS, autonomous driving) increases the complexity of future systems dramatically. Without advanced methods and tools, the effort in tackling these issues leads to an increase in development costs and time, and thus to missing the time-to-market window.

In the aim to verify complex and heterogeneous systems, a simulation-based approach is an effective way to gain more insights on the behavior of the whole system This is also stated and highly recommended by the functional safety standard ISO26262 for higher safety levels. Since these systems can have several different configurations depending on their purpose and are affected by a fast changing environment, they must exhaustively be tested towards their specific requirements. For fast and safe ramp of these devices, full coverage of parameter variation in simulation is, therefore, a primary success factor. Moreover, these systems are composed of highly heterogeneous subsystems on different abstraction levels, e.g. digital, analog, mechanical or software, which has an impact on the complexity and demands to combine different simulators in one framework. Due to this, simulation of complex systems can lead to a greater challenge in terms of managing complexity, hardware resources, simulation time (days or even months) and the amount of data produced. Moreover, a seamless integration of a virtual prototype into the upper layer of the system level specification and thus closing the design and verification gap is still an open issue in today's development.

Another issue is the fragmentation of the information to be captured at several levels of abstraction. Today, top-level specification is usually described in natural languages, which can lead to ambiguities or diverging interpretations between several teams involved in the design process such as system, software, hardware or verification teams. This effect is reinforced through additional effort in the development of safety-critical systems. Since model-based approaches are not yet common in the automotive industry, new design methods and tools must be introduced and are essential to avoid redundancy of information and ensure consistency all over the design process.

Therefore, this thesis aims at solving the main research questions by targeting the following goals:

- A seamless design process which helps to close technological, organizational and design gaps across abstraction levels in today's development of embedded automotive systems regarding functional safety.
- Development and integration of novel, model-based reliability analysis techniques and simulation-based verification in the proposed design process.

Besides the stated primary goals of this thesis, the following secondary goals are targeted:

- Reduction of the number of tools involved in the design process, thus to provice high traceability across design, requirements, tests and results which lead to high consistency, correctness, and completeness of models.
- Reducing manual steps, by introducing novel automation techniques throughout the development process.

## 1.3 Contributions

The overall scientific contributions of this thesis are presented in Fig.1.1. It shows an industrial V-model with safety extensions as it is specified in the functional safety standard ISO26262, better known as safety lifecycle. The contributions are listed in a numerical format, regarding their chronological appearance in the development cycle. Throughout this thesis we will demonstrate our proposed solutions by an industrial use case of a Battery Management System (BMS) provided by CISC Semiconductor GmbH. The use case illustrates a simplified version of the real application; it does neither represent an exhaustive nor a commercially-sensitive project and shall only provide evidence of the proposed solutions. First (I), we propose a novel design method for safety-critical systems, based on standardized modeling languages. With the help of an additional layer in the MDA, a gap in the design could be closed, and a seamless flow could be achieved. This design flow includes structural but moreover behavioral models, which describe safety aspects within the focus on ISO26262. Furthermore, this thesis emphasizes different reliability analysis techniques, which can be applied on these models throughout the development process. In

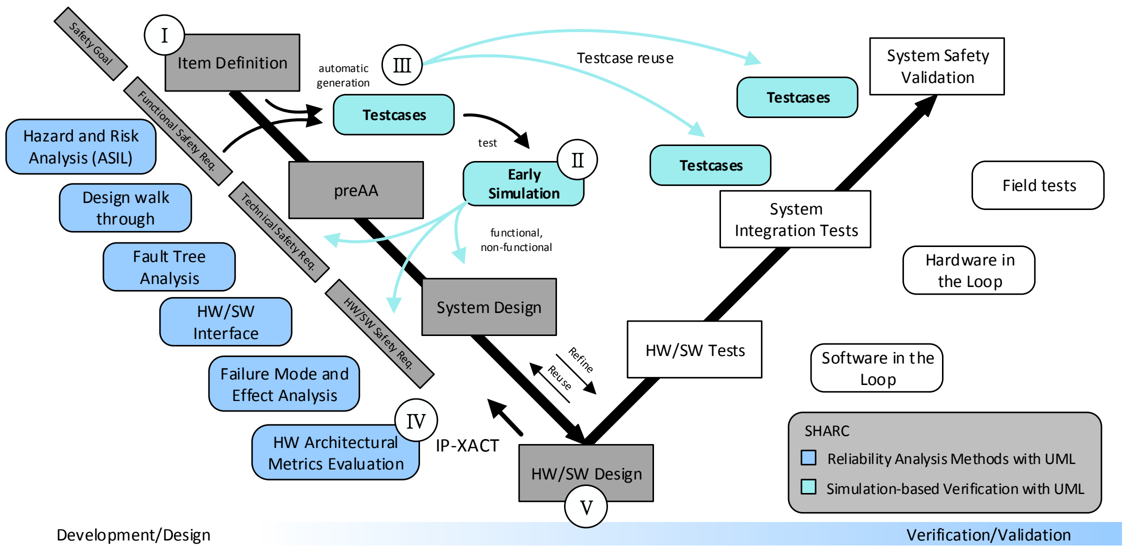


**Figure 1.1:** Overview of the contributions of this thesis that include techniques and tools that are applied during development to enhance the safety and reliability of the complete system.

(II), we describe how these structural models can be connected to executable models for behavioral simulation on system level and how to provide a seamless integration into the design flow. This includes the definition of interfaces for analog and digital simulation but also specification and configuration of the whole system. Contribution to (III) addresses the definition of safety requirements and constraints in a machine-readable and semi-formal structure. From this definition, testbenches for the verification of system safety requirements are automatically generated. This includes the verification of the system under different environment conditions and in different configurations to raise the level of functional coverage. To speed-up the coverage process, these independent simulation runs are distributed to a cloud-based environment. Results from the system verification can be directly used for the hardware platform design. The configuration and evaluation of the hardware platform regarding hardware faults is described in (IV). How this configuration leads to the automatic generation of a first virtual prototype will be explained in (V). Moreover, we show the seamless (back-)integration of the generated virtual prototype into the system design (functional specification) by using the novel methodologies implemented as a complete verification framework. This approach guarantees consistency, correctness and completeness within the overall system design and specified functionality.

## 1.4 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 3 discusses existing work in the area of design of safety-critical systems, design flows and virtual prototyping. Then, a novel design flow approach based on standardized modeling languages which includes design, evaluation, verification and integration of safety-critical systems is presented in Chapter 4. Chapter 5 describes how the methodologies have been implemented in the design and verification framework SHARC. Chapter 6 gives insights about the evaluation of the developed OpenES design flow and the specific SaVeSoC design flow. Finally, Chapter 7 concludes this thesis by summarizing the obtained results beyond the state of the art, and by providing hints on future research directions.

# 2 Background

This chapter provides definitions of key terms and a brief introduction to the background related to this thesis.

## 2.1 Functional Safety

ISO26262 is the functional safety standard for safety-critical systems in road vehicles. This standard addresses possible hazards caused by malfunctioning behavior of safety-related systems. It also provides an automotive safety life cycle that covers safety aspects throughout the whole design and development process of modern products and systems. In the first phase, the concept phase (Part 3), the to-be-developed item is defined. The item definition describes the boundaries and interfaces as well as assumptions about systems or arrays of systems, to which functional safety is applied. After performing a hazard and risk analysis on the item to identify and categorize the hazards, the safety goals are derived, and the corresponding Automotive Safety Integrity Levels (ASIL) are determined (2.1). The ASIL specify the item's necessary safety requirements to avoid unreasonable risk due to malfunction. The ASIL is determined by three impact factors: severity (S0-S3), probability (E0-E4) and controllability (C0-C3). This results in four ASIL levels (A to D), where ASIL A is the lowest and ASIL D the highest level. The class quality management (QM) denotes no safety requirements to comply with ISO 26262. Following the determination, the product is developed using recommended methods and measures according to its ASIL.

After determination, the product is developed with recommended methods and measures according to its ASIL. The derived functional safety concept contains safety measures, including the safety mechanisms to be implemented in the item's architectural elements and these are specified in the functional safety requirements. After defining the functional safety requirements, the technical requirements, including those for the hardware and software, are derived. This, in turn, results in a safety case, to show that the system is acceptably safe. The safety case is used to collect and present evidence and to support safety claims and arguments.

### 2.1.1 System Life Cycle

To more fully understand the safety life cycle and its concept we have to start with the definition of the global product/system life cycle. In integrated engineering, the produc-

**Table 2.1:** Risk graph according to ISO26262.

| Severity Class | Probability Class | Controllability Class | | |
|---|---|---|---|---|
| | | C1 | C2 | C3 |
| S1 | E1 | QM | QM | QM |
| | E2 | QM | QM | QM |
| | E3 | QM | QM | A |
| | E4 | QM | A | B |
| S2 | E1 | QM | QM | QM |
| | E2 | QM | QM | A |
| | E3 | QM | A | B |
| | E4 | A | B | C |
| S3 | E1 | QM | QM | A |
| | E2 | QM | A | B |
| | E3 | A | B | B |
| | E4 | B | C | D |

t/system life cycle comprises all phases that the product/system goes through from the idea to its end-of-life and revival. This includes the different phases in a closed loop from design, manufacturing, distribution, customization, the end-of-life and revival.

The safety life cycle is embedded in the product/system life cycle and is explicitly focusing on the safety aspects and issues regarding functional safety. Its emphasis is on the activities during the concept phase (Part 3), product development (Part 4-system, Part 5-hardware and Part 6-software), production, operation, service and decommissioning (Part 7). Part 1 and 2, as well as 8 to 10 are not part of the safety life cycle but supporting it for completeness.

The industrial V-model is a graphical representation of the system life cycle. It is an extension to the tradional waterfall model, where the process follows a sequential flow, while the V-model is a simultaneous process. It helps to ensure that the results to be provided are complete and have the desired quality. Instead of the waterfall model, results can be tested during development of the system. By applying standardized processes, effort concerning development, production, operation, and maintenance can be estimated and controlled over the entire project. Furthermore, it helps to gain a common understanding and improves the communication between all stakeholders. The V-model gives an overview of the main activities to be performed and results that have to be produced during development, from the decomposition of requirements (left side) to integration of parts and their validation (right side). However, validation can be applied on the left side as well. This brings us to the difference between validation (are we building the right thing?) and verification (are we building the thing right?). According to the PMBOK guide (Project Management Body of Knowledge) [1], these two terms are defined as:

- *"Validation. The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with verification."*

- *"Verification. The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with validation."*

### 2.1.2 Safety Life Cycle

The safety life cycle can also be represented as a V-model of the system life cycle with some minor adaption to the functional safety standard. The engineering processes have been organized in such a way that each process on the left side has a corresponding testing task on the right side. The overall V-model includes two smaller V-models, which handle the hardware and software development processes. The V-model follows a requirement-driven approach which means that design and tests are derived from requirements. Each system requirement must be traceable to one or more design elements, tests, and vice versa. In the ISO26262 they are separated in phases, where each activity in the standard has dependencies between either the requirements, the design or the test phase (Fig.2.1). These phases are arranged vertical, and each activity can be derived from the upper-level indifferent phases, e.g., the hardware safety requirements are derived from the technical safety requirements and also depend on the inputs from system design. We will discuss a more illustrative example in 4.2.

### 2.1.3 Stakeholder Analysis

The functional safety manager is responsible for coordinating and integrating all activities during the life cycle. Furthermore, he coordinates the different stakeholders, which take part in the whole life cycle. The definition of the stakeholder is "holding a stake," which means that everyone who has an interest in and can influence the process in a positive or negative way should be considered. A stakeholder analysis can give important inputs on who are the essential stakeholders regarding size (influence), distance (communication) and quality (relationship). Essential stakeholders in the context of the product life cycle are, e.g., clients, end users, customers, suppliers, design engineers (system, hardware, software), quality engineers, distributors, marketing, security engineers and safety engineers. In traditional product cycles, engineers are often decoupled from production or other departments, which can directly affect the development and design of a product/system. In modern, integrated design, all actors that are holding a stake in the product cycle should actively take part in the development process. This bi-directional relationship between designers and other stakeholders helps to achieve integrated design and to bring important properties such as safety or security into the product.

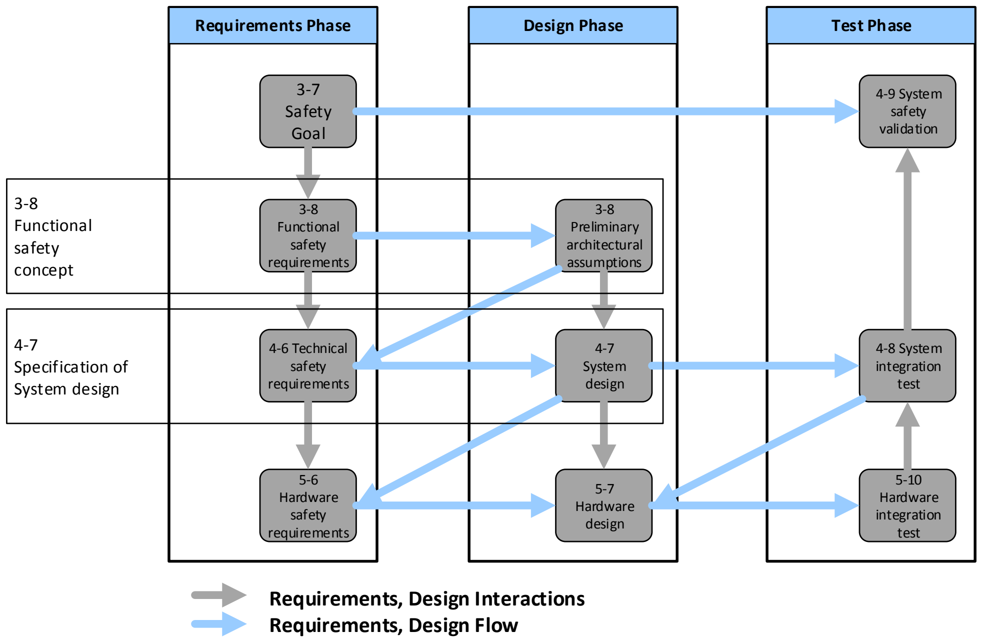

Requirements Phase
Design Phase
Test Phase
3-7 Safety Goal
4-9 System safety validation
3-8 Functional safety concept
3-8 Functional safety requirements
3-8 Preliminary architectural assumptions
4-7 Specification of System design
4-6 Technical safety requirements
4-7 System design
4-8 System integration test
5-6 Hardware safety requirements
5-7 Hardware design
5-10 Hardware integration test
Requirements, Design Interactions
Requirements, Design Flow

**Figure 2.1:** Relationship between the software, requirements and test phases in ISO26262.

This discussion brings us to another important topic in today's design teams, the differentiation between I- and T-shaped people. The I-shaped professionals are people who have a deep insight in one field of expertise (thinking narrow and tide). They are highly versed in a specific area and learn from drilling more deeply into a particular field. T-shaped people have broader skills and knowledge and learn by different perspectives from different specialties. The vertical bar of the T describes the expertise of professionals in a certain field. The horizontal bar depicts the ability to think and collaborate across various disciplines and systems. T-shaped people not only have in-depth knowledge, but they also share empathy and enthusiasm for other people's expertise. According to [85], they have the competencies and soft skills to cross boarders in terms of teamwork, communication, perspective, networks, critical thinking, global understanding or project management across one or many disciplines and systems. Although T-shaped people are important in today's development and design, this should not raise the impression that others are less important; both types are essential in modern organizations. It is the competence to communicate and collaborate across different areas of expertise and to share information competently with non-experts. From the academic perspective, Information and Computer Engineering (ICE) [34], former Telematics at the Technical University of Graz (TU Graz), can be seen as a place where T-shaped people are formed. Besides the education in computer science including different and diverse disciplines such as mathematics, electrical engineering, informatics or telecommunication, also extra-functional properties such as information security or safety are well embedded in the curriculum. Furthermore, the integration of soft skills in project management, team building, customer perspective or law, are the basis of a T-shaped professional. Several other shapes are mentioned nowadays in literature, such as hyphen-shaped, pi-shaped, H-shaped or V-shaped but are out of the scope of this thesis.

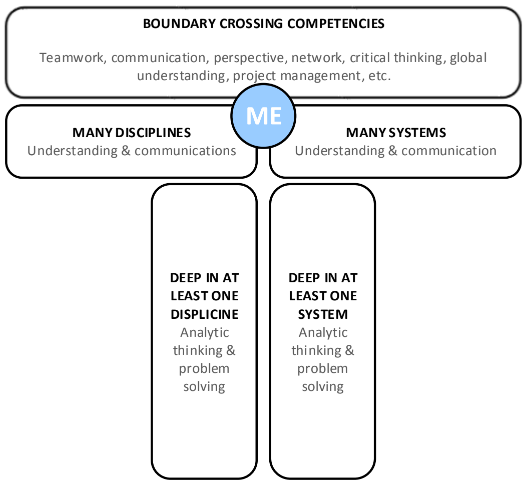

BOUNDARY CROSSING COMPETENCIES
Teamwork, communication, perspective, network, critical thinking, global understanding, project management, etc.
ME
MANY DISCIPLINES
Understanding & communications
MANY SYSTEMS
Understanding & communication
DEEP IN AT LEAST ONE DISPLICINE
Analytic thinking & problem solving
DEEP IN AT LEAST ONE SYSTEM
Analytic thinking & problem solving

**Figure 2.2:** The T-shaped professional.

If we project this view on functional safety, it is important that safety managers and engineers understand problems regarding hardware or software development and vice versa, engineers understand problems from the functional safety perspective. This forms, in the end, a Y-shaped professional, who combines two separate areas of expertise (which should naturally overlap) into one. In a modern, integrated design these people are key to integrate system/product properties such as functional safety.

A way to support this process and step towards achieving the goal of an integrated design is the model-based concept. It helps to communicate between several disciplines and to gather a common understanding of problems in the design. All in all, this is an important step towards a safe and reliable product, which also leads to faster time-to-market and reduction of development costs and product recalls.

## 2.2 Model-driven Engineering

In the aim to increase the level of abstraction and automation the MDE approach has found its way into the development of systems and software. One of the most popular MDE approaches is MDA, developed by the Object Management Group (OMG) standardization body. The MDA approach was driven by the diversity of systems, programming languages and frameworks in order to address interoperability and compatibility issues. As it is stated in the standard in the MDA Guide Version 1.0.1 from the OMG [57]: *"In the ideal sense, computing should be viewed by the users as "my" world, with no artificial barriers of the operating system, hardware architecture, network compatibility, or application incompatibility. Given the continued, and growing, diversity of systems, this will never be achieved by forcing all software development to be based on a single operating system, programming language, instruction set architecture, application server framework or any other choice. There are simply too many platforms in existence, and too many conflicting implementation requirements, to ever agree on a single choice in any of these fields."*

Conclusion: *"We must agree to coexist by translation, by agreeing on models and how to translate between them."*

Organization of specialized people in projects of a certain size requires a lot of effort. Therefore, it is becoming increasingly important that stakeholders from different domains, e.g., hardware, software, system design but also safety and security engineers can efficiently work together. Particularly in the verification of safety-critical systems, safety managers and specialists need an entire view of the system that includes all domains involved in the system design. Such a design, expressed in an architectural description, supports the understanding of the system's essence and key properties regarding its behavior, composition and evolution. This, in turn, can bring to light concerns regarding feasibility, maintainability, and utility of the system. To overcome the issues with different stakeholders involved in the design process, [45] defined that an architecture description can have one or more architecture views. A view helps to address the various concerns held by the system's stakeholders. According to [43]: *"A view is governed by its viewpoint: the viewpoint establishes the conventions for constructing, interpreting and analyzing the view to address concerns framed by that viewpoint. Viewpoint conventions can include languages, notations, model kinds, design rules, and/or modeling methods, analysis techniques and other operations on views. "*

One well-known viewpoint model is the "4+1" by [46]. This model allows to separately address the concerns of the various stakeholders of the architecture, as well as functional and extra-functional requirements. To capture large and challenging architectures, they propose a model consisting of logical view, process view, physical view and development flow. The logical view includes the used objects in the design. The process view captures the concurrency and synchronization aspects. The physical view describes the software to hardware mapping, whereas the development view depicts the static organization of the software in its development environment. These four views are organized around a fifth view, which represents the use cases or scenarios. The "4+1" view model described a rather generic approach, where other notations, tools or design methods can be used. For our purpose, we will focus on the unified modeling language standard (UML), which provides us with different diagrams to describe our system from different point of views.

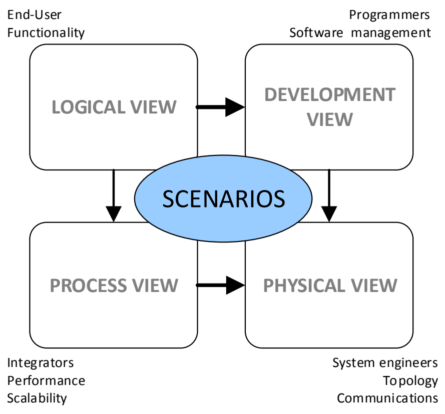


**Figure 2.3:** The 4+1 view model according to [46].

### 2.2.1 Design Languages

#### UML

UML is one modeling standard by the OMG [75]. It is a graphical representation for specification and documentation of software and other systems. With UML it is feasible to model application structure, behavior, architecture and also business processes and data structures. It delivers a complete view of the system, its individual components and the interactions between them. UML also declares how the system is expected to be used (special focus on system level use cases) providing different types of structural and behavioral diagrams. Another advantage is a common formalism to ease the exchange of information between stakeholders involved in the design and avoiding multiple captures of the same information. Although UML has advantages in capturing system level use cases and specifying system and software design, it lacks a proper definition for hardware design and non-functional properties such as timing, performance or thermal behavior. Extensions have been developed to overcome these issues.

#### SysML

SysML is an extension to UML2 and is a graphical modeling language for describing complex systems in system engineering. It supports the specification, analysis, verification and validation of a broad range of systems. It provides extensions to diagrams for describing behavioral and structural properties in UML2. Also, it has two additional diagrams (requirement, parametric) for requirements engineering and performance analysis. SysML is smaller and easier to learn than UML. One of the advantages of SysML is the mechanism to capture performance and quantitative information. It provides a good allocation of requirements to components, but it is inaccurate and provides no extra-functional properties.

### EAST-ADL

One modeling language that has established itself in the automotive domain is EAST-ADL. It allows capturing detailed automotive electric and electronic systems on five layers of abstraction, each with a clear separation of concerns: *Vehicle*, *Analysis*, *Design*, *Implementation* and *Operational Level*. Besides structural aspects, this modeling language allows the expression of behavior, requirements, verification and validation. The highest level is the *Vehicle Level* that describes electronic features to allow integration of product variability. The *Analysis Level* includes the Functional Analysis Architecture (FAA), which allows an abstract functional representation of the architecture (what the system shall do), in relation to the features from the *Vehicle Level*. The *Design Level* allows the decomposition of models in the FAA to Functional Design Architecture (FDA) models and Hardware Design Architecture (HDA) models. Within these models the functional representation of the architecture can be allocated onto the hardware platforms. The applications are represented by *DesignFunctionTypes* with annotated behavior and configurations. The hardware components are modeled by *Sensors*, *Nodes* (ECUs), *Actuators* and *HardwarePortConnector* (Buses) and more. They are interconnected by *IOHardwarePins* or *CommunicationHardwarePin* and wired by *HardwareConnectors*. The last two layers (Implementation, Operational) are the realization of the implementation in Automotive Open Systems Architecture (AUTOSAR). Therefore, the models on these levels are compliant with the AUTOSAR specification. The behavior of components on all these levels is not explicitly addressed in EAST-ADL. It can be either expressed by behavioral diagrams (state machines, activity diagram) or externally in tools like Matlab.

EAST-ADL as automotive modeling language also addresses parts of the functional safety standard ISO2626, which was one of the outcomes of international projects like ATESST[10] and MEANAD[53]. This enables the language for safety analysis like Fault Tree (FTA) or Failure Mode and Effect Analysis (FMEA), but also for defining safety requirements and achieving high traceability of models and behavioral diagrams. Furthermore, this language provides means to describe validation and verification activities by *VVCases*. Since EAST-ADL is included in an Eclipse UML2 Editor called Papyrus [28], complex systems can be designed without licensing costs.

### MARTE

MARTE is defined as a profile in UML2 and provides additional mechanisms for modeling real-time systems, which are missing in UML. Thanks to the UML extension mechanism, the software resource model (SRM) and hardware resource model (HRM) profiles extend UML2 with concepts for software and hardware. The purpose of the SRM package is to design software of real-time and embedded applications. It consists of the SW_ResourceCore, which provides the basic software resource concept. The HRM is an extension to UML and serves as a description of existing and for the conception of new hardware platforms. These descriptions can be made of different levels of granularity. The HRM is grouping most hardware concepts under a hierarchical taxonomy. It is composed of a logical view (HwLogical) that classifies hardware resources depending on their functional behavior and a physical view (HwPhysical) that focuses on the physical nature. The HWLogical model provides a classification for different hardware entities such as computing, storage or communication. This includes stereotypes like HwASIC, HwProcessor, HwBus, HwDevice or HwMemory. All the stereotypes defined in the HRM package are organized under a tree of inheritances (Fig. 2.4 from more generic stereotypes. This has the advantage that if no stereotype suits to model a used hardware component, a more generic stereotype may fit instead. As an example, a HwSensor inherits the properties from HwI/O and is furthermore a specialization of Hw_Device. In contrast, the HW-Physical package contains stereotypes as physical components. They describe their shape, size, and position within the platform, power consumption or other physical properties.

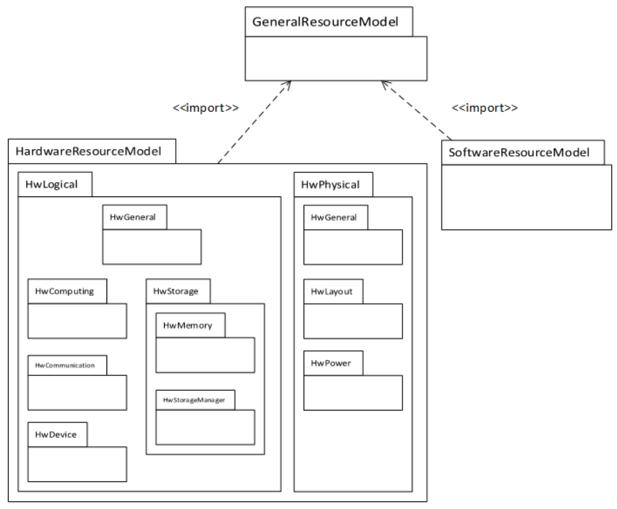


**Figure 2.4:** MARTE levels of granularity.

### Domain specific language

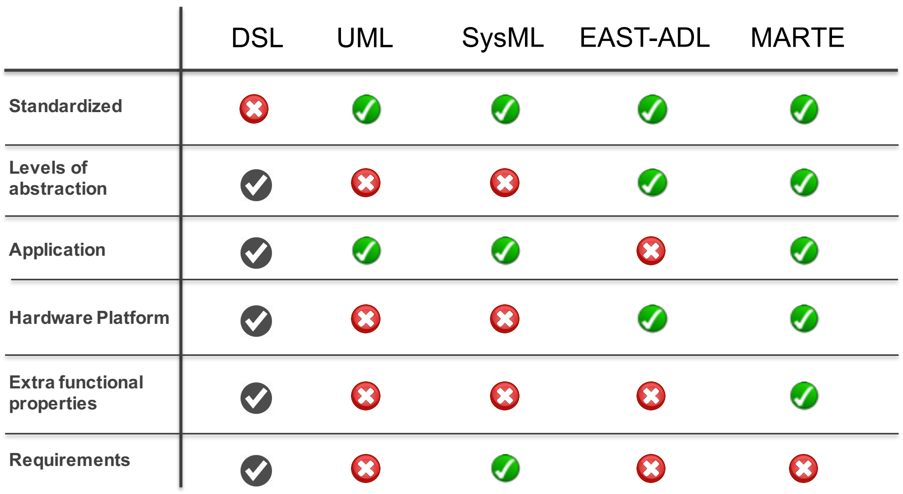
A Domain Specific Language (DSL), or application specific language is a formal language, which is used to describe a problem in a specific domain (e.g., avionics, financial services) and eases the interaction between human and computer. This language needs a specific design and implementation for a certain domain where it is supposed to be used. In the development of a DSL, the target is to achieve a high degree of specificity of a problem.

This concept shall ensure that everything in the target domain can be described, and everything surrounding this domain is negligible. Thus, a domain specific language can lead to an excess of highly specialized language concepts that is difficult to learn and cumbersome to use [77].

### Evaluation of Design Languages

To determine which design language is the most efficient to describe a complete safety-critical system in an automotive environment, we evaluated the before mentioned design languages regarding six important aspects (Figure 2.5):

- Standardization (interoperability with other tools)
- Level of abstraction (defined in the language)
- Application (software architecture)
- Hardware platform (architecture)
- Extra-functional properties (supports e.g., timing behavior)
- Requirements (supports definition of requirements)



| | DSL | UML | SysML | EAST-ADL | MARTE |
|---|---|---|---|---|---|
| Standardized | ✗ | ✓ | ✓ | ✓ | ✓ |
| Levels of abstraction | ✔ | ✗ | ✗ | ✓ | ✓ |
| Application | ✔ | ✓ | ✓ | ✗ | ✓ |
| Hardware Platform | ✔ | ✗ | ✗ | ✓ | ✓ |
| Extra functional properties | ✔ | ✗ | ✗ | ✗ | ✓ |
| Requirements | ✔ | ✗ | ✓ | ✗ | ✗ |

**Figure 2.5:** Comparison of common modeling languages.

As we can see in Figure 2.5, a DSL is too specific to be used in several tools and would imply an over-sized effort in the specification of the language, since all mentioned aspects have to be defined. Moreover, the costs for learning a new design language are too high. In contrast, a standardized language such as UML is widely known by the software community and can be applied in several domains. SysML, as system design language, also has extensions for the definition of requirements, but does not provide specific levels of abstraction for the development of hardware and software. On the other hand, EAST-ADL with its roots in the automotive domain already provides these levels to be used in an automotive development life cycle. Nevertheless, EAST-ADL only reaches its full potential in combination with the software development in AUTOSAR. Since MARTE provides several abstraction levels to be used in a system life cycle, models for application and hardware development, extra-functional properties for timing, resources, and performance, using this language is the appropriate choice for our design. Complementary with SysML for requirements definition we are able to provide an adequate basis for the design of safety-critical systems throughout the various development phases and to be used in several tools in this domain.

### 2.2.2 Virtual Prototyping

The ISO26262 recommends different verification methodologies used for the hardware platform. These include design walk through, FTA/ FMEA, hardware architectural metrics evaluation, but more importantly, hardware prototyping and simulation for higher ASIL levels. These simulations, including virtual prototyping, can then be used for further hardware verification methods such as fault injection test, which is currently the key for testing the reliability of the hardware. Several research institutes are now working on executing fault injection, also on higher abstraction level such as Transaction Level Modeling (TLM). This has the advantage that this method can be applied on faster simulation models without losing information from the more detailed models (Register Transfer Level (RTL)).

Virtual prototyping has the benefit that embedded software can be tested much earlier before a first real hardware prototype is available. Also, the hardware/software interface can be tested towards consistency. The drawback is that a complex Virtual Prototype (VP) is not developed overnight. It takes a lot of effort, experienced designers and engineers to build a so-called digital twin of the actual hardware. Our proposal is thus to reuse models for virtual hardware prototyping from open libraries such as Open Virtual Platform (OVP), [64]. OVP comes with a growing model library, which offers processor cores, memory, and various peripherals.

#### Functional Coverage

Functional coverage is defined as a metric, which is used to determine the completeness and verification progress of a design. It emphasizes design verification where the focus, besides functional verification, is also on non-functional aspects such as safety, timing or power. Functional coverage tells us about the quality of a testbench and what portion of the design has been activated and tested during the simulation run (controllability). On the one hand, observability shows the ability to observe effects of the simulation (white-box vs. black-box testing). Thus, this metric allows us to answer the crucial question in the verification process "Are we done yet?". Through coverage metrics, we are able to adjust our tests and stimuli to optimize the verification. Moreover, it helps us to reduce debug time and to increase the correlation between specification, design, and verification. Were all design features and requirements identified in the tests? Have there been any lines of code or structures in the design that have never been exercised? We can classify coverage [88] by

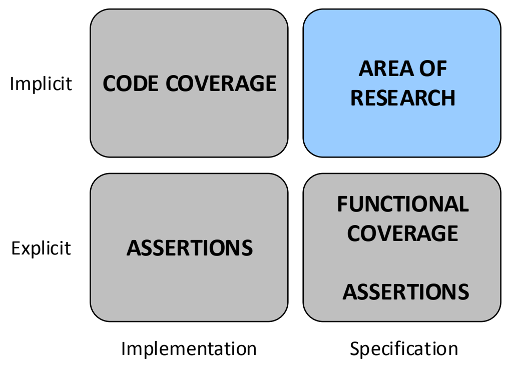


**Figure 2.6:** Implicit/explicit coverage, adapted from [88].

its method of creation (implicit vs. explicit) and their origin of the source (specification vs. implementation), see Fig. 2.6. Line coverage and expression coverage are two examples of an implicit coverage metric and can be automatically derived from the code, whereas functional coverage (explicit coverage metric) has to be defined and implemented by the engineer, derived from the various requirements and the specification document. None of these metrics is sufficient to make a statement about the completeness of the system. As an example, we might achieve 100% code coverage during our simulation runs, but do not know if we verified 100% of our functionality. This is because code coverage does not measure the interaction and behavior between the systems, nor the temporal sequences of functional events. On the other hand, we might achieve 100% functional coverage but only 90% code coverage because an important feature is missing in the test plan or specification, which never reaches the specific part of the code. A complete functional coverage can only be achieved if there are tests for all the features, which are indicated in the specification and the engineer has thought of. Functional coverage distinguishes between two simulation methodologies, direct testing and constrained random verification, whereas the latter is able to achieve a higher distribution over the huge space of the available input stimuli. This mechanism increases the coverage and the ability to find corner cases in the design by creating random tests, which have not been found by direct testing. The coverage space classified by an implicit specification (also known as intelligent verification) is a current academic research area, where the coverage metrics are automatically extracted by a tool and are derived from the design specification. These higher-level functional behaviors cannot be automatically derived from the implementation alone and need the information from the specification as well.

### 2.2.3 Design Frameworks

UML as a generic standard has the benefit that it is supported by various commercial and open-source tools. This strenghtens the interoperability between languages and the growing tool landscape. We emphazise the implementation with the UML editor of Eclipse named Papyrus [28]. This tool already provides the designer with UML extensions such as MARTE, SysML or EAST-ADL. Another advantage of Eclipse is the plugin mechanism, which eases the development of extensions for the UML editor. Sirius [29] is an other open-source tool from Eclipse, which can also be used for the design of complex systems, but its emphasis is more on domain specific languages rather than standardized modeling languages such as UML.

## 2.3 Related European Projects

### 2.3.1 OpenES

To improve European electronics system design productivity (faster time-to-market), design quality (fewer design errors and fewer re-designs) to stay competitive, the OpenES (Catrene, Eureka) consortium joins forces to provide missing links in system-level design and to develop common open solutions based on four pillars:

- Fill gaps in design flows with new interoperable tools and/or improve existing tools/flows ensuring the semantic continuity of the design flow.
- Specifically focus on integral support of both functional and extra-functional requirements from specification to verification, jointly with the use cases defined at system level.
- Raise reuse capabilities from IP to HW/SW subsystem in order to eliminate integration effort by supporting reuse of pre-integrated and pre-verified subsystems.
- Enhance interoperability of models and tools by upgrading and extending existing young open standards (SystemC TLM, SystemC-AMS, IP-XACT).

To demonstrate the efficiency of this approach a new tool in cooperation with CISC Semiconductors is developed in this project. The aim of the tool SHARC is to design, simulate and verify safety-critical systems in the automotive area. Safety turns out to be the key issue for future vehicle development, as the complexity in the automotive area is steadily growing. Especially when systems interact with and have an effect on the physical world, so-called cyber-physical systems, it is not longer sufficient to test a single behavior. The whole system must be validated as early as possible in the development cycle and at any level of granularity. This is also recommended by the ISO262626 standard for automotive E/E systems.

### 2.3.2 eRamp

eRamp, an ENIAC project, is focusing on the rapid introduction and research activities of new production technologies and further exploration of chip packaging technologies for power semiconductors. The goal is to have fast access to reliable prototypes of electronic devices as well as competitive advanced manufacturing of such devices made in Europe. The project partners are investigating and developing new methods for speeding up the start of the production runs for More than Moore (MtM) technologies. New simulation methods and verification technologies are thus essential to enable high performance and high-speed designs to support the ramp up of MtM products. In the course of eRamp, CISC and TU Graz are developing a new simulation environment (SHARC) to speed-up the development and verification of microelectronic embedded systems. With the help of novel methodologies, based on standardized technologies (UML, UVM, SystemC), thousands of different Monte Carlo simulation tasks are distributed and parallelized in a cloud-based infrastructure. This enables a virtually linear speed-up in testing reliable systems.

# 3 Related Work

This chapter presents the current state-of-the-art and related work in dealing with the design of safety-critical systems. We outline other simulation-based approaches in the goal to verify systems regarding functional safety and how hardware evaluation is done in other projects.

## 3.1 Design of Safety-Critical Systems

The effort in today's development of cyber-physical systems is huge, as are the costs. This not only includes the engineering effort, but also the tools involved in the whole process. In the automotive domain of today, there is no single approach that unites the design and verification tools in one single environment, which can also be used throughout the whole development process. In the current development processes several different tools are used such as Matlab Simulink [55] for the functional specification, Doors [71] for requirements specification, document-centric approaches such as Excel for failure mode and effect analysis (FMEA), hardware architectural metrics, hardware/software interface or even the whole design, furthermore Maggilem for traceability and Mentor/-Candence for hardware simulation, just to name a few. For each of these steps, data from the design must be exhaustively exported to other tools and vice versa. System-wide and even cross-domain constraints, such as safety features must be exchanged between tools manually, which results in redundancy and inconsistency. Today's approaches lack a proper design low, a modeling language to communicate between different engineering and management teams and furthermore traceability of requirements, design, tests and results.

The MDE vision emphasizes to increase the abstraction level and automation through modeling, which entails advantages such as reduced development costs, increased quality, and reduction of maintenance costs [7]. Machine-executable models are essential in today's design flows, since companies are seeking for increased automation. Increased automation also means faster development and fewer costs. Multi-view modeling is one approach to deal with complex embedded systems in the development process. Often there are many different experts and tools involved, also from different domains, which help experts in the analysis and decision making during the design. A view that is tailored to their particular tasks helps to gain more insights on the system. In [78] a model integration framework is presented which addresses the issues associated with multi-view modeling. The authors are using the potential of the SysML standard as a general language to present a common model and to create dependencies between various domain-specific tools and languages. To achieve consistency between the views, model transformations are defined that map the interdependent constructs to and from a common SysML model. This paper points out interesting issues in today's development of embedded systems. The overall goal of the authors was to provide consistency across multiple views of a system to meet the objectives of a variety of stakeholders. They also describe the problems when dealing with the integration of simulation models and specifying configuration parameters of the system. To tackle the research question regarding consistency, they propose that a general framework must also take into account the workflow process, to allow consistency to be evaluated. The authors are stating that SysML as general modeling language is too high-level to be used for domain-specific application (simulation of control systems). In contrast to this thesis, they use model-transformation and don't take a modeling languages with several abstraction levels into account.

The authors of [51] aimed at achieving consistency of information between several tools involved in the development process through a single source of information principle. Wanting to achieve dependability (safety, security) in the development process between different teams and stakeholders, they decided against a document-centric approach and used the capabilities of UML and SysML for their design. This, in turn, improved consistency, correctness, and completeness of the entire system under development, which is in line with the objectives of this thesis. In contrast to our approach, the toolchain in this paper focused more on the system and software development and did not take hardware development into account. The authors also propose to update their profile to work efficiently on hardware development as such.

How to use SysML as representation of requirements in the automotive industry and the functional safety standard is discussed in [38],[4],[61]. The authors show how to model the requirements on different abstraction levels in a semi-formal way. Also, the traceability across the different levels in the requirements phase of the safety lifecycle is handled. They point out issues of bidirectional traceability in today's distributed developments and how this information can be used for later assessment of the safety case. The approach of [4] extends SysML to define requirements of safety-critical systems. In [61] also the allocation to structural and behavioral models is taken into account. This approach very well shows how to use SysML for requirements on different abstraction levels. In contrast to this thesis, the used method does not cover the whole design phase of the ISO26262 and confines traceability solely to UML/SysML diagrams. We will build upon this approach to cover all traceability aspects as recommended in the standard. As none of these approaches consider MARTE as detailed modeling language for hardware and software, we will show how to use SysML to maintain the traceability in MARTE models on multiple levels.

Two major European projects that also relate to a model-based design in the automotive domain are the SAFE [74] (Safe Automotive soFtware architEcture) and MEANAD [53] (Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles) projects. The objective of the SAFE project was to define development processes complying with functional safety and to develop new methods for defining safety goals. Furthermore, the aim of this project was to improve dependability from vehicle to component and the early evaluation of the safety architecture. During this project, this has been achieved by defining a meta-model for a model-based safety analysis, which is based on existing technologies (ReqIF, EAST-ADL, and AUTOSAR). They also defined a toolset to capture requirements, modeling and safety analysis based on formal models and to support automated safety analysis to reduce the manual effort. In contrast to the OpenES project, SAFE solely focus on the development and safety analysis of automotive usecases. The OpenES project covers a broader spectrum, which also takes other domains into account. EAST-ADL models are not flexible enough to be applied to other E/E systems and domains such mobile or health-care. Nevertheless, will build upon the outcomes of this project, because several important objectives have been covered in the development of safety-critical systems in the automotive domain.

With the focus on Fully Electric Vehicle (FEV) and to bring the engineering of FEV to a next level, the MAENAD project extended the EAST-ADL2 standard with advanced capabilities to facilitate development of dependable, efficient and affordable products. The three goals of this project were to support the ISO26262 safety standard and to support automatic allocation of safety requirements to components of an evolving architecture and an effective model-based prediction of quality attributes of FEVs (dependability, performance). A further goal was the automated exploration of huge design spaces to achieve an optimal trade-off between dependability, performance and costs. By using a common modeling language in the project, they managed to understand engineering information across different departments and companies, to exchange engineering models between different organizations and to progress jointly on tools and methodologies for modeling, analysis and synthesis. Also, the MAENAD project proposes to use an overall design methodology for FEV development. The MAENAD project covers important aspects in the development of fully electric vehicles, which is also in line with the objectives of this thesis. Therefore, we will use the results in this project to cover certain issues in today's development process.

By combining several UML profiles, as presented in [32], [58] and [15], the authors could increase the potential for easy validation and verification of embedded systems, and facilitated reuse and evolution. Since a single profile may not be adequate to cover all aspects of a multidisciplinary domain, popular UML-profiles, in particular EAST-ADL, MARTE, IP-XACT and SySML have been combined. By following this approaches [15], a unique model can be established which take into account requirements, functional modeling, and the modeling of verification and validation activities. Furthermore, the functional safety standards ISO26262 was also partially addressed in this paper. This is completely in line with the proposed process and methodology in this thesis. However, this approach lacks in simulation capabilities and a proper framework for modeling. While the approach from [15] asks for a framework to combine these profiles, [32] proposes a complete modeling framework, while avoiding specification conflicts. The authors of [58], try to close the gap between UML modeling and simulation (e.g. SystemC) for verification and synthesis, by using code-generation from SysML models. They also point out, that using standardized profiles leads to a higher interoperability between other enterprise tools. In contrast to our thesis they do not take modeling of safety-critical systems into account nor the use of MARTE.

## 3.2 Hardware Evaluation in the ISO26262 Context

The authors of [24] evaluated the hardware fault metrics of a electronic power-assisted steering (EPAS) system by using systematic quantitative Fault Tree Analysis (FTA). Using their methodology, they derived top-level events, which serve as starting point for the analysis. The objective was to build a structure that contains all relevant random hardware failures that could contribute to the top-level event. From the item definition they derived a first fault tree that represents the system structure, where they consider that the failure of a block can occur due to one or two categories: i) fault within the block or ii) fault in the input of the block. These events are described as intermediate level events. At the bottom level of the fault tree, root causes of failures are described. After describing the whole fault tree, they have been able to calculate single-point/residual faults and dual-point faults of their IC. Systematic but also random hardware faults of the safety mechanisms have been considered in the evaluation. Data for FIT rates, possible faults, safety mechanisms or values for diagnostic coverage have been based on references such as ISO26262 or IEC61508. In contrast to our approach, the authors of this paper do not take modeling languages into account, thus it is challenging to reuse or even transform information to other views. Moreover, the do not provide tools for their methods.

How to develop hardware according to functional safety and the ISO26262 is described in [44] and [17]. In [17] the quantitative hardware architecture of an automotive safety microprocessor is evaluated. The data for the diagnostic coverage of the hardware components comes from a commercial EDA environment, and no evidence is given about the correctness. Both approaches neither take modeling approaches into account nor does they recommend safety mechanisms to improve their use cases. Furthermore, they do not use tools to support their methodology.

The authors of [73] present a methodology to evaluate design choices early in the development process. This is done in an iterative way until a specific safe, and cost-effective E/E architecture is derived. This approach was applied in a model-based development process. The models used in this paper are specified as a self-defined metamodel for representing the part of the design artifact. However, this approach neither takes standardized modeling languages into account nor does it use information from standardized hardware IPs. The authors of this paper also recommend using languages like SysML or MARTE.

The authors of [5] present a rapid method for qualitative and quantitative assessment of hardware architectural design in a model-based approach. They are evaluating the design in a top-down manner using fault tree analysis. The architecture is evaluated with the HiP-HOP methodology in an external tool through a defined exchange format. The evaluation of the hardware is more related to the hardware part evaluation and not to the architectural level of advanced SoCs, as this plays a crucial role in the second version of ISO26262 coming in 2017. They also propose a meta-model for modeling hardware designs regarding functional safety in EAST-ADL2 [22], which we will rely on.

In paper [83], the authors apply the ISO26262 standard to several example scenarios involving Li-Ion batteries for plug-in vehicles. They point out occurring problems in today's integration and evaluation of Li-Ion batteries regarding safety. This includes malfunctions of control systems that may have an impact on charging and discharging. Also, the thermal aspect must be taken into account. The paper shows how a Hazard and Risk Analysis (HARA) and Goal Structuring Notation (GSN) can be applied to a Li-Ion battery pack. Furthermore, they define safety requirements during the design process, from safety goals down to the technical safety requirements. The authors also propose to add more details on hardware and software implementation to their models. In contrast to our approach, they do not use UML for their design nor do they use detailed hardware and software models. Nevertheless, the authors point out important aspects in the definition of safety requirements for Li-Ion batteries which are also important for this thesis.

## 3.3 Simulation and Virtual Prototyping

Popular approaches such as [6] and [25] have shown that UML as modeling language can be efficiently used with analysis and verification methods such as FMEA (failure mode and effect analysis), fault tree analysis (FTA) [65], [48], design walkthrough [36], code generation [27], [67], [89] and many more. The drawback of UML, in terms of simulation to verify the system behavior is that code-generation can only be done at a very late stage or even at the end of the design process, when all details are very well known. Later changes in design are costly and result in inconsistent models and reverse engineering, which is an error-prone and cumbersome task. The majority of components in new projects are reused and simply extended by the addition of new features to reduce costs and time-to market. The reuse of whole safety concepts, well-trusted designs and mechanisms is thus becoming more important to reduce the effort required for developing complex systems. This situation prompts the urgent demand for new techniques to simulate the behavior in early development phases by reusing verified system components.

Many approaches try to combine the design with analysis techniques. In particular, the authors of [47] propose a methodological approach for modeling and analyzing integrated safety-relevant automotive ECUs in early stages. The model analysis is performed by additional tools which operate on the derived information from the modeled system. To perform scheduling analysis [40] they also use external simulation tools. The authors also point out the benefits of standardized models by allowing the development of standard modeling and analysis tools. Nevertheless, this approach has the drawback of using an analysis toolchain for the evaluation of the system, thus using model transformation [72]. Since this approach relies on the generic UML profile, they also propose additional profiles such as SysML and MARTE.

In [54] the authors present three different analysis techniques for architectural models described in EAST-ADL, to guarantee the quality in the context of ISO26262. One of the proposed techniques is the simulation of EAST-ADL functions in Simulink. The behavior of each function was linked to FMU or Simulink models to facilitate the simulation. The authors also described mapping rules for the EAST-ADL to Simulink transformation (one-to-one mapping). The results of the simulation have been traced back to the requirements. This approach was applied to an industrial use case of a brake-by-wire system on the design level. In contrast to our approach, however, they use proprietary simulation engines with high license costs and external tools which are not integrated into the design and development flow and do not reach lower abstraction layers.

Several papers have been published about the Gaspard2 design environment [68], [11], [70]. The aim of this environment was to overcome the complexity issues within the Multi-Processor System on Chip (MPSoC) development. It uses the capabilities of the MARTE and the MDE approach, to move from a high-level system representation to an executable platform. Within this developed design flow for MPSoC, they used a compilation chain to transform the high abstraction level models to Cycle Accurate Bit Accurate (CABA) and Timed Programmer View (PVT) SystemC simulation. With the used MARTE representation they separated the application from the hardware architecture by the corresponding allocation. They introduced a deployment profile in MARTE to transform the high abstraction level models into simulation code, also by linking existing code to each elementary component. They forced the use of IP libraries to keep the MPSoC model independent from the compilation target. By using the MARTE standard, they also agree that this approach increases maintainability and simplifies modification. They applied their design flow on a use case of an H.263 video encoder within the Gaspard2 framework. Thanks to the high-level modeling, they highlighted the ease of exploring various configurations, also by using different points of view.

In [59], [60], the authors build on the work of Gaspard2 and present a semi-automated design flow where HW/SW Codesign and the MDE methodologies are merged and exploited to enable a fast design process. The aim of the authors was to help and design complex real-time embedded systems by allocation and binding within design space exploration, schedulability analysis, HW/SW partitioning and estimation techniques. To achieve these goals they agreed on using UML profiles such was SysML and MARTE. By using the Computational Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM) approach and mode-to-model transformation, they described their system at different levels of abstraction, until sufficient details have been added. At the end of the design process, including structural and behavioral models, they generated the corresponding SystemC executable models for simulation and verification. Nevertheless, none of these approaches are taking safety standards in their design processes into account.

COSIDE from COSEDA [21],[13], [52] is a proprietary Electronic System Level (ESL) design tool for design exploration, verification and virtual prototyping of embedded systems. For the modeling of hardware and software, but also analog and mixed-signal systems on the system level, they use the capabilities of SystemC and SystemC AMS. The framework is based on the Eclipse environment and uses a DSL for the graphical representation of their design models. For the verification of system level use cases in the automotive domain, they propose to use Coverage Driven Verification (CDV) within UVM and extensions for SystemC AMS. For the creation of AMS test scenarios, consisting of stimuli generation and response checking, they extended the SystemC AMS standard by new language constructs and generic verification components. Furthermore, they automated the test creation procedure by using the IP-XACT standard, and they claim to support an ISO26262 conform design process. COSIDE seems to be a powerful tool relying on the same simulation engine as SHARC. Nevertheless, they use a DSL approach for their design models. Moreover, the don't provide other reliability analysis techniques besides simulation and virtual prototyping.

To overcome the issues with consistency within design and simulation models in the context of ISO26262, the authors of [81] proposed a model transformation framework between SysML and Matlab/Simulink. They support a consistent and traceable refinement from the early concept phase to software implementation in a bi-directional manner. The authors also claim that a model-based design helps to enable different views for different stakeholders, different levels of abstraction, and central storage of information. Nevertheless, the author's focus was more on the software architecture generation from system design rather than on the requirements for hardware design.

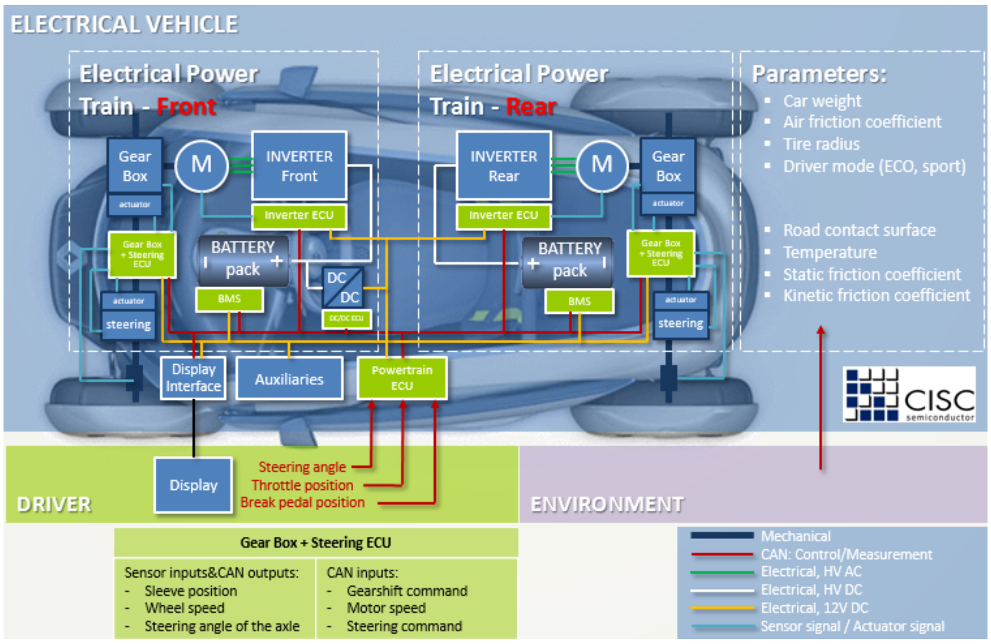
# 4 Verification, Generation and Integration of a Safety-Aware Virtual Prototype

In this chapter, we describe our novel design process named Safety Aware Virtual Prototyp Generation and Evaluation of a System on Chip (SaVeSoC). We will first explain our seamless model-based design flow based on standardized modeling languages. Then we will show how this model-based approach can be used for different verification methods and reliability analysis. Furthermore, our use case, on which the SaVeSoC process has been applied, is presented.

## 4.1 Use Case

Throughout this thesis, we will demonstrate our methodologies on a relevant problem in today's automotive domain, a battery management system for Li-Ion-powered electrical vehicles. This industrial use case was provided by CISC Semiconductor GmbH. It will help to illustrate more fully the innovative capabilities and benefits of our approach. As more and more vehicles are now powered by Li-Ion batteries the challenge for engineers to ensure reliability and fault tolerance is also greatly increasing. It is crucial for ensuring safe operating conditions of a battery that monitoring systems, such as the BMS, measure the voltage, temperature, and current of the battery very precisely. This information must be forwarded to a vehicle-wide controller network to ensure a reliable and fully utilized system. Problems with overheating or even explosions have been frequent in the past. The main cause of these problems was an excessively high energy intake from regenerative braking or harsh environmental conditions. Management systems and mechanisms are thus essential to ensure that persons are not put at risk and that no damage is caused.

The overall system of the eVehicle is depicted in Figure 4.1. For reasons of simplification, we only consider the major components of the electric vehicle, for the analysis of the battery and the BMS. This includes the battery pack in Li-Ion technology, the BMS, which measures voltage and temperature of the battery, an inverter ECU, a controller and the electric motor model. Two main factors influence the behavior of the eVehicle. The driver provides the desired speed (rounds per minute) and the load on the motor shaft. These stimuli can be set according to standardized maneuvers such as the New European Drive Cycle (NEDC), or the newer standards known as the worldwide harmonized light-duty vehicles test procedure (WLTP), which will be introduced in 2017. The controller is a model for a PI state-space controller and maintains a constant speed based on the



| Gear Box + Steering ECU | |
|---|---|
| Sensor inputs&CAN outputs:<br>- Sleeve position<br>- Wheel speed<br>- Steering angle of the axle | CAN inputs:<br>- Gearshift command<br>- Motor speed<br>- Steering command |

**Figure 4.1:** Overall system level description of the electric vehicle use case.

information about the state variables, motor armature current, and motor speed. The inverter model implements an inverter function for a PM-DC motor driving stage. It compares the actual battery voltage and the requested controller voltage to maintain the PM-DC motor terminal voltage. The battery model simulates the behavior of a Li-ion battery pack composed of a defined set of single cell Li-Ion batteries. The appropriate number of single cells is connected in parallel and series to obtain the necessary capacity, maximum current, and terminal voltage. The battery pack's terminal voltage is calculated based on the defined parameters and the battery current. A BMS is connected to the battery to measure voltage, current and temperature of the cells/modules. The BMS computes the state of charge (SOC), State-Of-Health (SOH) and is responsible for cell balancing, cell protection and demand management of the battery. These computed values are processed via a CAN controller as digital values and forwarded to the power train controller. Also, the external load environmental conditions, such as temperature, can be changed during the simulation.

The battery model is a central focus of this thesis. It is a detailed model of a Li-Ion battery pack and a BMS, which computes internally for each timestep the new state of charge based on the previous one. Changes in temperature are also calculated. The temperature can further on be used to calculate the module voltage, taking the current state of charge into account. The batteries' reliability is of utmost importance in a vehicle and avoids harmful effects for occupants in case of a failure. To guarantee safety, BMS are used to control and monitor the behavior of the battery throughout the entire life cycle of a battery. In this paper, the function of a BMS shall be implemented on a microcontroller, which meets the high safety requirements in its implementation.

## 4.2 Safety-Critical Embedded System Design

### 4.2.1 Functional Specification

To overcome the shortcoming of defining hardware platforms in UML and SysML, an extension to UML2 was defined, which provides capabilities to model hardware and software, as well as timing, resource and performance behavior of real-time and embedded systems. This modeling language is called MARTE [63], [77] which follows the philosophy of cyber-physical systems to deal with the whole system rather than with a set of specialized parts. Furthermore, it has been established in the embedded system domain. MARTE is defined as a profile in UML2 and provides additional mechanisms for modeling real-time systems, which are missing in UML. Thanks to the UML extension mechanism, the software resource model (SRM) and hardware resource model (HRM) profiles extend UML2 with concepts for software and hardware. In addition to the concepts of software and hardware resource models, it is possible to allocate software applications to hardware resources with the help of the MARTE allocation mechanism. This is particularly important for schedulability analysis and real-time applications but also multi-core applications. For the modeling of systems on a higher level of abstraction (system level), MARTE provides capabilities with the general resource models (GRM). These models can be used for components, where no early assumptions about implementation in hardware or software can be made. The stereotypes offer concepts to model general platforms for executing real-time embedded applications. This includes the modeling of both, hardware and software. With this package, it is possible to model complete systems on a very high abstraction level. This helps us to model systems very early in the design process, when design choices are still undecided. These models can then be refined in a later step of the design process. The GRM package includes different resource types, representing a physically or logically persistent entity, e.g., ComputingResource or StorageResource. These resources offer services to perform the expected tasks.

A stereotype which helps to simplify the modeling in a component-based approach is GCM (General Component Models). It brings the advantage of describing ports with information about incoming (in), outgoing (out) or bidirectional (inout) communication of the different subsystems. These FlowPorts have been introduced in MARTE to enable a flow-oriented communication paradigm between components.

Especially in the automotive domain, where timing and performance is crucial, modeling languages such as MARTE help to describe these properties of real-time systems. To cover also important properties from the functional safety aspect, we have shown in Paper A how to design safety-critical systems throughout the design phase of the functional safety standard ISO26262 based on UML/MARTE. Our contribution in this work was a mapping approach between the model driven architecture (MDA) and the design phases of the functional safety standard, as depicted in Fig. 4.2. To close the gap between system level design and hardware/software architecture in today's design flows, we defined an additional layer between the *computation independent model* (CIM) and *platform specific model* (PSM). This layer, named *refined PIM*, defines additional structural and behavioral diagrams for a seamless design flow, especially in safety-critical system design. We defined the *refined PSM* to extend the standard by an additional level, since they have not been adequately specified and lacked a formal definition. The resulting four levels of our approach are named *computation independent model* (CIM), *platform independent model* (PIM), *refined PIM* and *platform specific model* (PSM). They represent all major design phases of the functional safety standard such as item definition (Fig. 4.3), preliminary architectural assumptions, system design, down to hardware and software design.

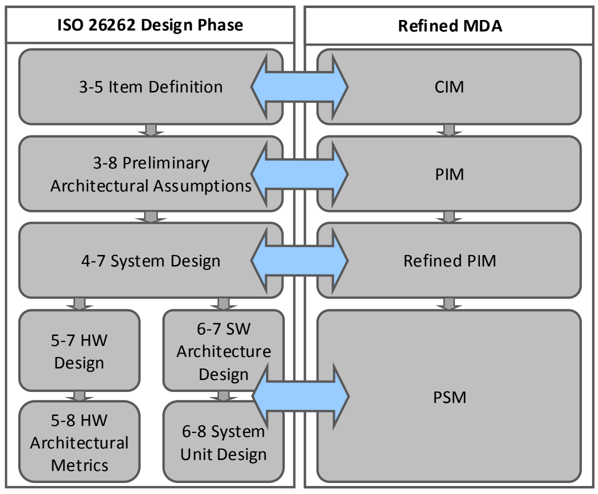


**Figure 4.2:** Mapping approach between the ISO26262 design phases and the MDA, adapted from Paper A.

The CIM level aims at providing a system level view, mainly focusing on its functional structure. Figure 4.3 shows the CIM level in the context of the functional safety standard (item definition), including the information flow between functional blocks and boundaries/interfaces to the environment. With the help of the detailed information of the graphical representation as composite structure diagram, safety goals are derived, which are the basis for the functional Safety Requirement (SR).

The PIM level includes the CIM capabilities with additional behavioral models such as UML state machines or activity diagrams. If the behavior is not directly expressed in behavioral models, the models can reference to existing implementation code (see Chapter 4.3.1). These models can then be used for an early and high-level simulation of the system behavior before making decisions about the hardware platform. We represented the CIM level as preliminary Architectural Assumption (preAA), which is the result of the functional safety concept including functional SR. This level is still independent of the actual implementation; therefore no technical details on the platforms and where the functions are implemented are specified. At this stage, important behavioral diagrams such as the safe state or detailed description about the system reaction and fault reaction time are added and allocated to the structural diagrams of the preAA. These safe states are also defining the timing properties and constraints of the design. This information is later on used for the comparison with the simulation results.

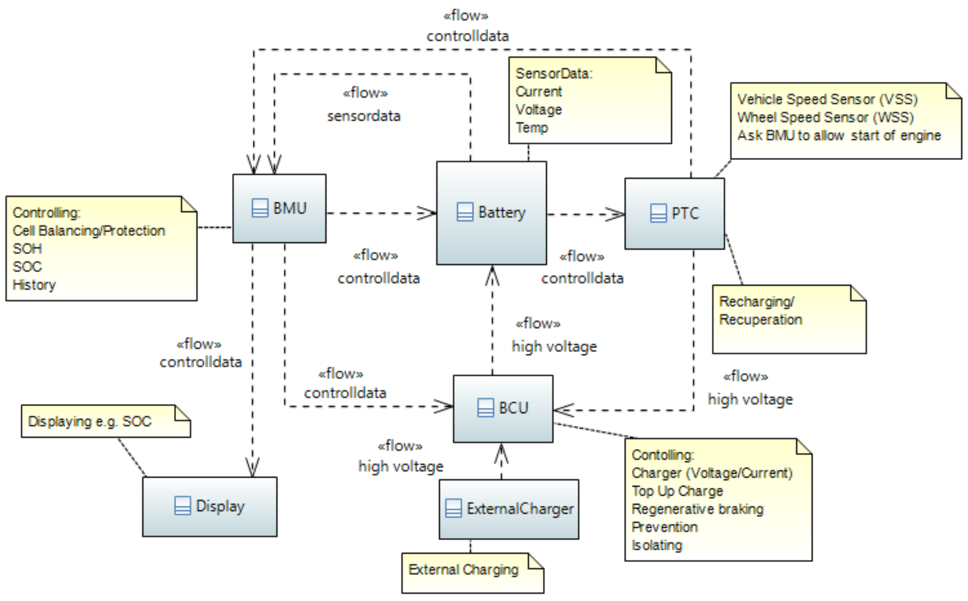


**Figure 4.3:** Definition of the item on CIM level, with functional models in the UML standard, adapted from Paper A.

The refined PIM has been explicitly identified to fit within the sub-system definition concept. It consists in a functional decomposition of the PIM with a granularity detailed enough to allocate each of its blocks to a single hardware or software resource. This results in the conclusion that a PIM functional block cannot be allocated to several execution resources. Communication interfaces have to be defined and functional blocks have to be split into sub-functionalities before switching to the next abstraction level. On this level, we also apply different safety patterns to increase the reliability and availability of the system. This approach is also in line with the decomposition mechanism as defined in ISO26262, by using redundant and diverse hardware, e.g., for redundant and diverse measurements.

The PSM level defines the detailed steps towards hardware and software design. On the hardware side, the models can contain low-level details, such as registers and memory map information. On the software side, the execution platform description can specify OS-specific information such as tasks, scheduling algorithms or middleware services. A more detailed description of the design of the hardware platform is given in Chapter 4.2.2.

Through applying this approach to an industrial use case, we demonstrated how a safety-critical system can be designed and specified, but also how existing verification methods such as fault tree analysis (FTA), failure mode and effect analysis (FMEA), hardware software interface (HIS) can be applied to this structural and behavioral models. Furthermore, SysML was used in addition to MARTE to define safety requirements and to handle the issues with traceability. With the link to MARTE components and behavioral diagrams we achieved a horizontal and vertical traceability, as it is required by the ISO26262 standard.

### 4.2.2 Hardware Design

As mentioned in our previous chapter, UML/MARTE provides several levels of detail for the specification of the hardware platform. The hardware resource model (HRM) package provides several models to describe subsystems such as HwProcessor, HwBus, HwDeviceor HwMemory in a logical and physical way. Although UML/MARTE provides a rich set of different stereotypes to describe the hardware platform, it does not provide the level of detail for the hardware configuration as expected by the OVP methodology. Several mandatory properties such as the BusInterface or Memory mapped or the VLNV principle are not supported in the MARTE standard by now.

To overcome these issues, we decided to rely on the specification of the IP-XACT standard, which helps us to define our platform in a sufficiently high degree of details for the generation of the virtual prototype. Both IP-XACT and OVP rely on the VLNV principle for structuring the models. Therefore, we built on approaches such as [9] to extend the MARTE standard by properties in IP-XACT for hardware description. Figure 4.4 depicts the composed structural architecture model of the platform consisting of a processor, bus, memory, CAN and two ADC. For simplification we only show the major components of the design of the battery management system. The designer can easily compose his system by using the standard models from the MARTE library such as HwProcessor for the Central Processing Unit (CPU), HwI_O or HwComponent for peripherals, HwRam for memory or HwBus for the internal bus or CAN bus. Depending on the nature of the component, the designer can extend the component with specific hardware properties in the IP-XACT standard such as MemoryMaps and BusInterface. The properties from the IP-XACT standard are then shown in the description of the MARTE model. Further extensions for IP-XACT such as VLNV can be added to the hardware components as well.

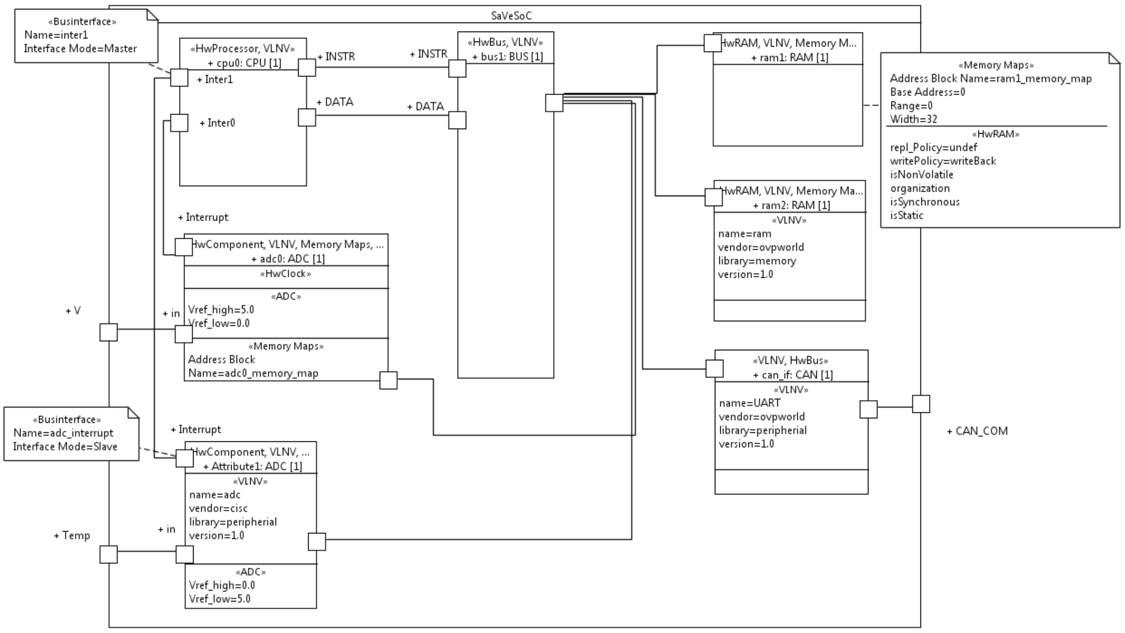


**Figure 4.4:** Hardware platform of SaVeSoC including IP-XACT extensions for MARTE.

By applying this approach, the designer can configure the whole hardware platform, for instance "instructions per seconds" of the processor, which affects the simulation time directly. The designer uses the standard MARTE models from the library and adds additional IP-XACT properties to his models. After the evaluation of the design regarding safety, the description of the platform is converted to a TCL description in the OVP standard. The OVP iGen converter takes the information from the TCL script and generates a full SystemC platform using the modular components of the OVP library. The resulting virtual prototype can then be used for further hardware simulations, such as fault injections on TLM level.

For demonstration purpose, a small application is running on the platform, which computes state of charge, state of health and checks on the plausibility of the incoming measurements. This application is written in c-language and also defined through the allocation mechanism in MARTE. The provided processor is also capable of executing embedded operating systems such as embedded Linux.

## 4.3 Evaluation/Verification of the Design

### 4.3.1 Simulation-based Verification

What makes the functional standard ISO26262 "functional", is the aspect that we analyze the function which shall be implemented on the whole system (in this case the item). This starts with a situation analysis and hazard and risk analysis with the help of the item definition. This requires the development of a system definition, including components, hardware (-parts) and software (-units). Based on the resulting ASIL level of the hazard and risk analysis, the safety goals lead to the functional safety concept, which includes the definition of the functional safety requirements with respect to preliminary architectural assumption (preAA).

Since the step, from the first definition of the system and their boundaries (item definition), to the functional safety concept is a cornerstone towards the development of hardware and software, we support this by our methodology described in Paper F. In this work, the first system design (preAA) is supported through a simulation-based approach. Thanks to early executable models of the functional specification, further technical requirements can be derived. The functional specification of the system is depicted in Figure 4.5. With this methodology, we can evaluate our first design in early design phases and furthermore throughout the entire development process. Since reusability of well-tested

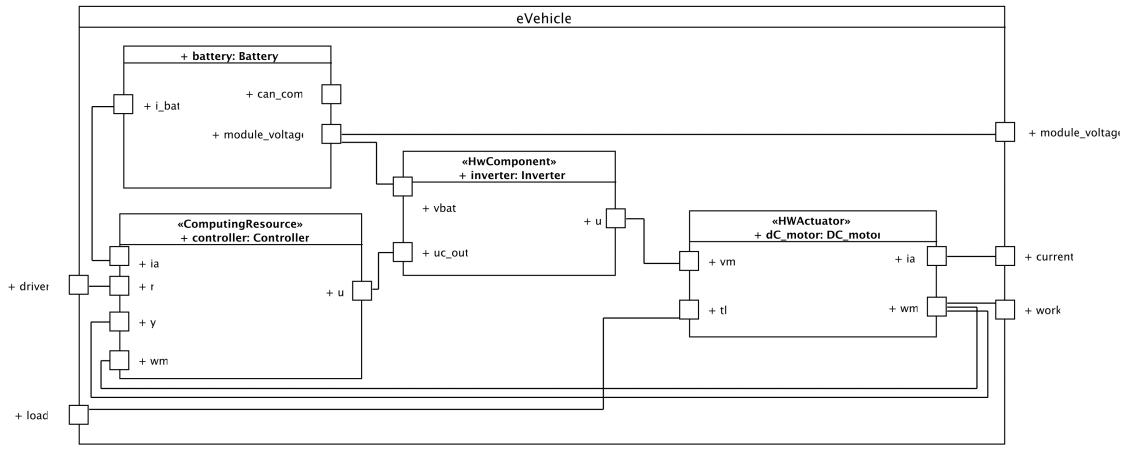


**Figure 4.5:** Functional specification of the preliminary architectural design.

designs, mechanisms or even complete safety concepts is an issue that is currently becoming even more important; we support this artifact through reusable models from our developed System Component Library (SCL). This library includes all major elements for a high-level simulation of digital, but also analog systems, in the automotive domain. It also includes components on different levels of abstraction and different versions, depending on its application and viewpoint. For this approach, we use the capabilities of SystemC for digital systems on register transfer level (RTL) and transaction level modeling (TLM). For analog and mixed signal models we use the extension of SystemC AMS. Since these simulation models rely on the same modeling language, as our whole design flow, they can be easily integrated into our preliminary system design. How the UML/MARTE models are linked to the simulation core in SystemC is described in paper Paper E in detail.

The presented approach can also be underpinned by the statements from the ISO26262 standard in Part4: Product development at the system level [41]:

*"The technical SR shall be specified in accordance with the functional safety concept, the preliminary architectural assumptions of the item and the following system properties: the external interfaces, such as communication and user interface; the constraints, e.g. environmental conditions or functional constraints; and the system configuration requirements. The ability to reconfigure a system for alternative applications is a strategy to reuse existing systems."*

*"The system design shall be verified for compliance and completeness with regard to the technical safety concept using the verification methods e.g. Simulation for ASIL level higher than B."*

As already described in the previous chapter, we made an extension to the requirements of the SysML standard to define also SR. The resulting requirements definition, depicted in Fig. 4.8, helps to define all different requirements from different sources and stakeholders and to keep trace with models, verification tests and results. This becomes even more obvious if we look at the definition of the safety standard Part 8 (Supporting Processes [42]), where it is recommended to use semi-formal notations for requirements specification for ASIL higher than B. Therefore, each functional SR in this approach has several defined constraints for functional and non-functional properties. These constraints are defined in the MARTE Value Specification Language (VSL) and specify the boundaries for fail-safe operations of the system but also environmental conditions and operation modes. These constraints precisely capture the original requirement and open up, through computer readable formalism, the possibility of subsequent computer-aided analysis of the characteristics of the design. The MARTE nfpConstraint is defined by arithmetic, logical or time expressions formed by combining operators such as ('<','≤','=','≠','≥','>') but also 'AND', 'OR' and 'XOR'. The syntax used for these constraints follows the pattern:

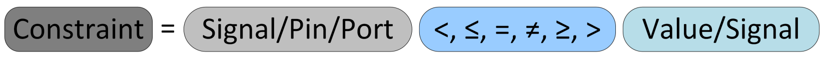


**Figure 4.6:** The constraint pattern.

Multiple constraints can be connected via simple Boolean statements such as:

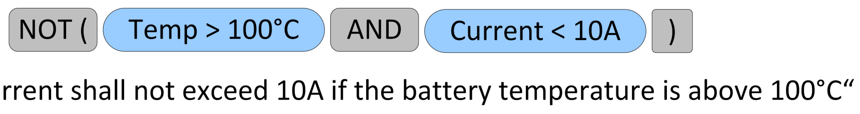

NOT (
Temp > 100°C
AND
Current < 10A
)
„The current shall not exceed 10A if the battery temperature is above 100°C“

**Figure 4.7:** Example constraint derived from a functional safety requirement.

In order to support the specification of the technical SR and furthermore enable the verification in compliance with the technical safety concept, we defined a novel methodology to derive further requirements and inputs from the functional SR in coherence with the early system design (preAA). Using the syntax for safety requirements we are able to generate UVM verification components and whole testbenches from the definition of the functional SR and their constraints. For each constraint of the functional SR, a new UVM validator is added to the ports or one end of the signal. A validator consists of a configurable comparator with the pin/port/signal attached to one input and a reference signal or constant value attached to the second input. The outputs of the comparator can be either 1 (true) or 0 (zero) and are connected via arithmetic or algebraic function blocks to create the Boolean operations. In addition, we use non-safety requirements in the SysML specification to provide stimuli blocks for relevant operating modes and driving maneuvers. Depending on the non-safety requirements and constraints and if the pin/port/signal is an unused input of a block, the testbench generator creates a stimuli block and attaches it. This block generates either values that are within the specifications in order to validate proper operation or it generates invalid stimuli to verify safety mechanisms within the model. More details on this generation are given in Paper F.

To vary the parameters and stimuli of our system and to cover up corner cases we use the benefits of CDV, with its aim to detach from direct user dependent testing [2]. This methodology provides the definition of so called verification goals, which can be verified by smart test scenarios. The intelligence is mainly achieved by creating simulation configurations (stimuli), with respect to some predefined constraints. This concept is widely known as Constraint Random Verification (CRV). CRV mainly consists of two core concepts, which is on one hand, the usage of Markov-chain Monte Carlo to guarantee coverage through probability and on the other hand the processing of constraints with SAT solvers.

As described above, it is important to vary parameters so that many different input combinations can be covered. The defined internal values of the DUT vary according to a predefined probability distribution. In this case, we use Gaussian distribution with the definition of a value of 3 Sigma. This approach covers requirement-based tests as recommend on all ASIL levels.

As there is a trend to more structured, modular, configurable and reusable verification methods, UVM was defined to tackle these challenges. UVM is an Accellera System

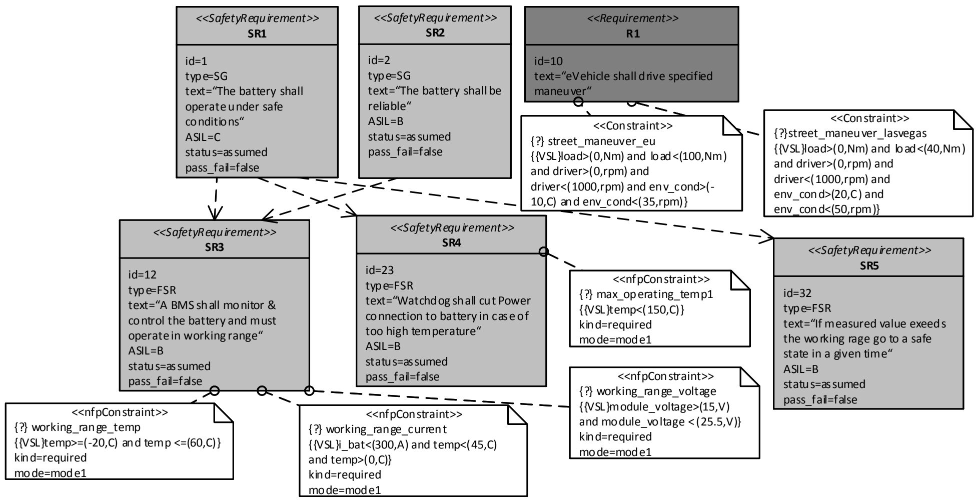

<<SafetyRequirement>>
SR1
id=1
type=SG
text="The battery shall operate under safe conditions"
ASIL=C
status=assumed
pass_fail=false
<<SafetyRequirement>>
SR2
id=2
type=SG
text="The battery shall be reliable"
ASIL=B
status=assumed
pass_fail=false
<<Requirement>>
R1
id=10
text="eVehicle shall drive specified maneuver"
<<Constraint>>
{?} street_maneuver_eu
{{VSL}load>(0,Nm) and load<(100,Nm) and driver>(0,rpm) and driver<(1000,rpm) and env_cond>(-10,C) and env_cond<(35,rpm)}
<<Constraint>>
{?}street_maneuver_lasvegas
{{VSL}load>(0,Nm) and load<(40,Nm) and driver>(0,rpm) and driver<(1000,rpm) and env_cond>(20,C) and env_cond<(50,rpm)}
<<SafetyRequirement>>
SR3
id=12
type=FSR
text="A BMS shall monitor & control the battery and must operate in working range"
ASIL=B
status=assumed
pass_fail=false
<<SafetyRequirement>>
SR4
id=23
type=FSR
text="Watchdog shall cut Power connection to battery in case of too high temperature"
ASIL=B
status=assumed
pass_fail=false
<<nfpConstraint>>
{?} max_operating_temp1
{{VSL}temp<(150,C)}
kind=required
mode=mode1
<<SafetyRequirement>>
SR5
id=32
type=FSR
text="If measured value exeeds the working rage go to a safe state in a given time"
ASIL=B
status=assumed
pass_fail=false
<<nfpConstraint>>
{?} working_range_voltage
{{VSL}module_voltage>(15,V) and module_voltage<(25.5,V)}
kind=required
mode=mode1
<<nfpConstraint>>
{?} working_range_temp
{{VSL}temp>=(-20,C) and temp <=(60,C)}
kind=required
mode=mode1
<<nfpConstraint>>
{?} working_range_current
{{VSL}i_bat<(300,A) and temp<(45,C) and temp>(0,C)}
kind=required
mode=mode1

**Figure 4.8:** Specification of requirements and safety requirements on different abstraction level.

Initiative approved standard methodology for verification and provides a UVM Class Library with all the building blocks needed to quickly develop well-constructed and reusable verification components and environments in SystemVerilog. Furthermore, it provides a well-defined layered architecture to clearly distinguish between the various abstraction levels. UVM is usually developed for SystemVerilog, which allows simulation and verification of digital hardware on RTL level. Since the trend in embedded system design, as well as in verification of safety-critical systems, is going to a higher abstraction level, various approaches tried to connect UVM with SystemC [13],[50],[79]. SystemC allows the modeling and simulation of hardware and software components in a single language and allows the description on TLM but also RTL, which leads to a faster or more detailed simulation depending on the use case. Trough applying methods such as [14] a higher-level verification on system level could be achieved, which defines high-level testbenches that can be reused throughout the whole development process.

## Distribution of Simulation Tasks to a Cloud-based Environment

Today, millions of test kilometers have to be driven to ensure a reliable behavior of the electronic/electrical systems in a car [56]. This procedure has to be done for all the different versions of a car model, each with different features, also affected by environmental conditions. Since many parameters and stimuli data have to be tuned and varied to achieve a high degree of functional coverage, verification implies a huge amount of simulation runs. Without using novel simulation methodologies this can lead to high costs and verification cycles. One standard, which is used in industry to test embedded microelectronic systems, is Universal Verification Methodology (UVM). UVM provides capabilities

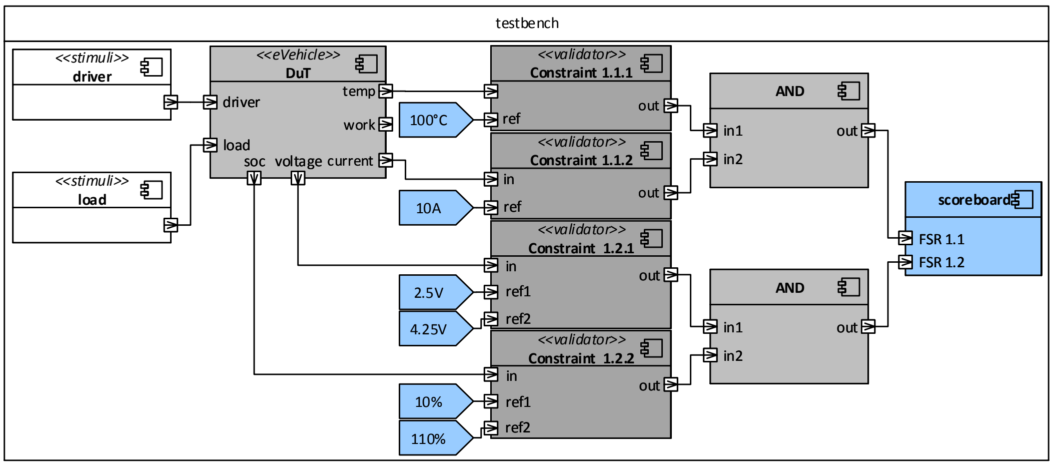


**Figure 4.9:** Automatic generated UVM testbench from safety requirements, adapted from Paper F.

to generate thousands of random test runs with CRV to cover all possible parameters of a system and simulates it in a sequential manner. In a sequential process, each task has to wait until the prior task ends. This has the drawback that verification of a system can take hours, days or even weeks to end. In addition, small companies cannot afford big server farms and internal clusters to test their system in an appropriate amount of time. With our approach we want to speed-up the simulation time within the UVM standard through parallel execution of simulation runs achieved through a novel cloud-based verification pattern.

Cloud computing has led to a paradigm shift in the way software is consumed and delivered. Moreover, it is changing the way systems are developed with tools and complete environments moving to the cloud. By using Software as a Service (SaaS), complete development processes are taking place in the cloud [75], thus bringing benefits such as a faster development and a saving of resources (money, time, effort).

To solve the problems for our large number of simulation tasks we defined an adaptation of the UVM- SystemC layered architecture by introducing messaging patterns from the Enterprise Integration Patterns [37]. As mentioned before, the overall result of a simulated sequence does not affect other configurations in any way, which leads to the possibility of parallel processing of sequences. Due to the fact that the simulation of a sequence is the most time-consuming part of the verification, a (theoretical) linear speedup can be expected. This prediction can also be underpinned by the fact that returning results from single CPUs can be neglected compared to the simulation time. Hence, a cloud-based approach was developed, which is illustrated in its main features in figure 4.10.

The traditional UVM-SystemC architecture consists of five layers, which communicate through standardized interfaces. This allows for a clear distinction between test case def-

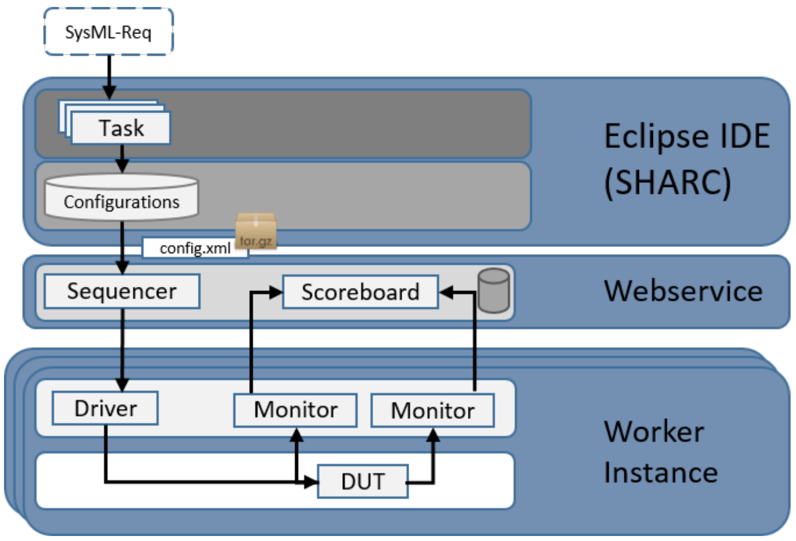


**Figure 4.10:** Layered Architecture of the novel UVM for the Cloud solution.

inition from test scenarios and moreover the actual verification environment (verification IP) including driver, monitor, and configuration. The highest layer defines the current test, which consists of the selection of the testbench stimuli and test sequences. UVM test is the top-level component and has three main functions: initiation of the verification environment, configuration of the environment and applying stimuli by invoking sequences. The functional layer contains the sequencer, which is responsible for the right arbitration and ordering of sequences and their transactions. Another part of this layer is the scoreboard, which collects the observed results from the monitors and checks the behavior of the Design Under Test (DUT). It compares the expected output (golden reference model) with the actual output from the DUT. Furthermore, for self-checking, the collection of functional coverage and pass/fail reports is necessary. The Command layer includes the driver, monitors, and checkers, which are implemented on physical-level. The driver receives the individual sequence-transaction from the Sequencer and forwards it to the DUT. The Monitor samples the data coming from the DUT and is responsible for coverage collection, checking, logging or recording. On the signal level, the lowest layer of this architecture, the testbench is connected, and the signals are sent to the DUT.

To overcome the issues with the traditional UVM approach, we applied the Messaging Pattern from the Enterprise Integration Patterns [37] on the layered architecture from the UVM-SystemC approach. A simple working queue pattern is depicted in Fig. 4.11.

The messaging pattern allows for a many-to-many connection where only one address has to be known to all machines. This machine accepts and distributes messages, takes care of message persistence and can optionally monitor all consumers using a heartbeat signal. Retransmission is also built into the message broker (B). The worker connects

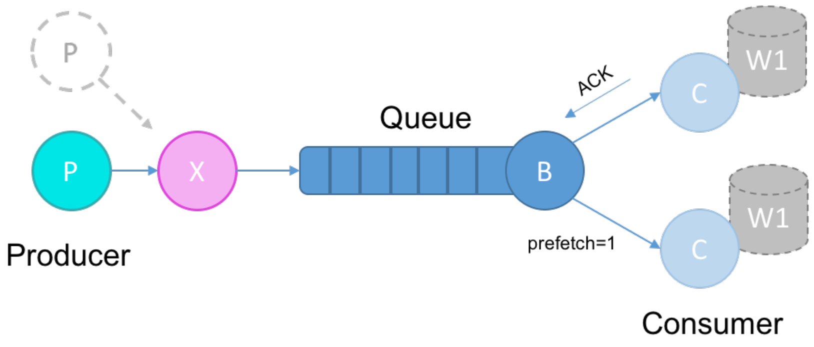


**Figure 4.11:** Simple worker queue from the enterprise integration patterns, adapted from [37].

to the broker (running on the master) and uses a heartbeat signal to indicate it has not crashed or is otherwise unavailable. Worker queues are used to distribute time-consuming tasks among multiple workers in the cloud. This is especially the case if tasks, such as simulation runs, can take hours or days to complete. This way we avoid carrying out a resource-intensive task immediately and can schedule the task to be done later and do not need to wait for the task to complete. Depending on the number of workers, the task will be shared between them. Each of the components of this pattern (publisher, consumer and broker) can be swapped out to a separate machine, when thinking towards a cloud-based cluster.

Through bringing the UVM for SystemC approach to a cloud-based environment by applying enterprise integration patterns, we solved six important issues in the verification of embedded systems:

- **Reduction of simulation-time:** With our cloud-based UVM approach we can reduce the time for simulation of thousands of simulation tasks. The fact that each simulation task is independent of one another, and does not affect other configurations in any way, leads to the possibility for parallel processing of sequences. From our approach, we can expect a (theoretical) linear speedup. Companies also benefit from a faster time-to-market.

- **License costs:** Our framework utilizes the full capacity of the server infrastructure in a very efficient way to scale down the number of licenses. Furthermore, we use open standards such as SystemC to reduce license costs.

- **Flexible infrastructure:** There is no need for companies to invest in big server farms. Through footprint analysis, we are able to predict the time for simulation and therefore can buy simulation time on demand from server and cloud providers. Simulation can be done over night when simulation time is cheaper.

- **Architecture:** Our pattern provides different levels of detail for the architecture of our approach. Therefore it is defined in detail and guarantees who is distributing the simulations tasks, what is done when simulations fail and where the results are collected and analyzed.

- **Efficiency:** A broker implemented in RabbitMQ automatically distributes the tasks to the worker instances to guarantee a high degree of capacity utilization.

### 4.3.2 Evaluation of the Hardware Architecture

In the previous section we have shown how to gain important information for the specification of the technical safety concept through a simulation-based approach in early phases of the development process. An automatic high-level testbench generation in the UVM standard helped to check functional safety requirements and constraints. These high-level testbenches can be used throughout the whole development process, from system design to hardware and software design, to check on consistency with the functional specification. With the help of this approach, a refinement of the preAA to the final system design including the technical safety concept can be achieved.

From the gathered information, we are now able to start the safety-aware hardware design of our platform. This is an important step in the development process and takes a lot of effort, since many different methods and measures have to be applied to the platform to guarantee a reliable product in the end.

Part 5 of the ISO26262 standard [42] handles the product development at the hardware level, which includes the evaluation of the hardware architectural metrics. It evaluates the hardware architecture of the item against the requirements for fault handling. This part includes guidance on avoiding systematic and random hardware failures by means of appropriate safety mechanisms. Each safety-related hardware element is analyzed regarding safe, single point (SPFM), residual and multiple point faults (LFM). It also describes the effectiveness of the hardware architecture in coping with random hardware failures (PMHF). Each hardware part is to be protected by means of safety mechanisms. The diagnostic coverage gives evidence of the effectiveness of these mechanisms. Whether the item (system or array of systems according to ISO26262) passes or fails a given ASIL is also a result of the hardware architectural metrics evaluation. To achieve a certain ASIL, the values from Table 2.1 must be met. It is also important to point out that only safety-related hardware elements that have the potential to contribute significantly to the violation of the safety goal are addressed in this metric. This must be considered in the evaluation of the whole item.

The drawback of this mandatory step is that it is done at a very late phase of the whole development process, where later changes are time-consuming and cause high costs. Furthermore, these safety-related properties, such as the failure rate of hardware components, are published and taken from various standards such as Siemens Handbook SN 29500 [80],

**Table 4.1:** Architectural Metrics - evaluates whether the hardware achieves a certain ASIL, according to ISO26262.

| | ASIL B | ASIL C | ASIL D |
|---|---|---|---|
| SPFM | $\geq 90\%$ | $\geq 97\%$ | $\geq 99\%$ |
| LFM | $\geq 60\%$ | $\geq 80\%$ | $\geq 90\%$ |
| PMHF | $< 10^{-7}h^{-1}$ | $< 10^{-7}h^{-1}$ | $< 10^{-8}h^{-1}$ |

MIL HDBK 338 [87] or IEC 62380 Reliability Handbook [39]. Usually, these data are very general, dependent on the temperature and not applicable to every domain. In some cases, the source is unspecified and principally obtained from field or statistical data. This, in turn, can lead to false consequences, if the failure rate predictions differ significantly from field data.

We propose that safety-related information for hardware-IPs should come from the vendors themselves, as they know the product best. Therefore, our approach allocates safety properties to vendor IPs in a standardized way, so there are no false assumptions about safety-critical hardware.

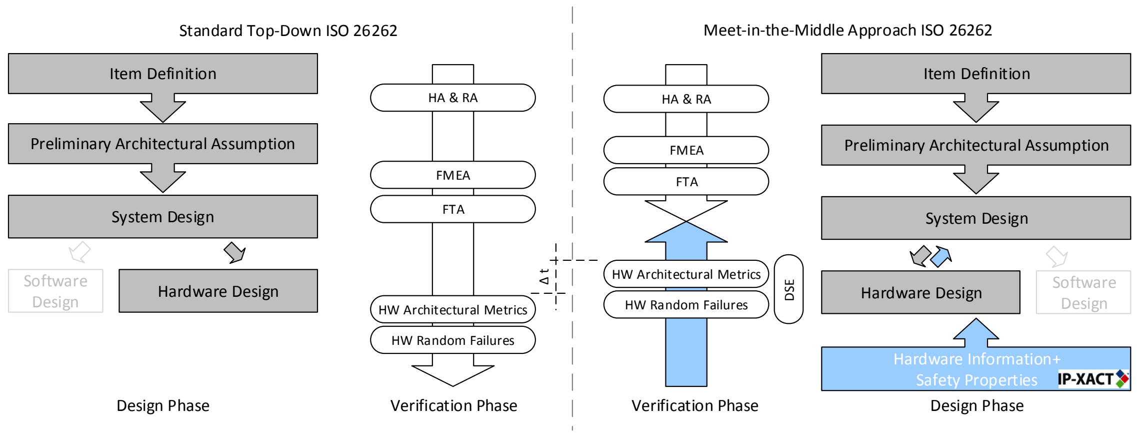


**Figure 4.12:** Meet-in-the-middle approach: Design and verification speedup through failure modes provided by hardware description adapted from Paper B.

Late decisions about hardware characteristics and fault behavior can cause wrong decisions at the system level. Therefore, it is necessary to have information about hardware early to ensure system integrity. Figure 4.12 depicts the new methodology in comparison to the top-down design approach as executed in ISO26262. The left side shows the different abstraction levels of the design phase and their derivations, from item definition to detailed hardware and software design. It also shows recommended verification methodologies, which are to be used on each abstraction level. The arrow shows the chronology of these methodologies in the verification process. The right side depicts our approach by providing safety properties in the IP-XACT standard to speed up the verification and design process of safety-critical systems. This so-called meet-in-the-middle approach makes it possible to evaluate the hardware design round $\delta_t$ earlier, as it is handled in the traditional approach. Time for evaluation is reduced through seamless integration of tools in the design process, a hardware IP library and provided design space exploration. Furthermore, the system design layer benefits from our approach, which allows an earlier verification through methods such as FTA and FMEA. The safety properties provided are defined in a standardized format, which can be used by many verification engineers and provides important inputs to additional stakeholders.

With the described methodology, we are able to provide inputs for various evaluation techniques. In this part of the chapter we will focus on the evaluation of the hardware architectural metrics. To evaluate the hardware according to Clause 6, Clause 7, Clause 8 and Clause 9 in ISO26262, the following equations must be carried out to achieve a given ASIL level. This task must be completed separately for each safety goal and requires seamlessly integrated tools to support this evaluation-process:

**Single Point Fault Metric (SPFM):** The SPFM reflects the robustness of the item when coping with single point and residual faults. This can either be handled by design or proper safety mechanisms. The higher the value of SPFM, the more robust our applied safety mechanism will be. The following equation is used to determine the SPFM:

$$SPFM = 1 - \frac{\sum_{SafetyRelatedHW}(\lambda_{SPF} + \lambda_{RF,est})}{\sum_{SafetyRelatedHW}\lambda} \tag{4.1}$$

$$\lambda_{RF,est} = \lambda \times (1 - \frac{K_{DC,RF}}{100}) \tag{4.2}$$

where $\mathrm{K_{DC,RF}}$ is the diagnostic coverage with respect to residual faults and $\lambda_{\mathrm{RF}}$ is the estimated failure rate with respect to residual faults.

**Latent Fault Metric (LFM):** The LFM reflects the robustness of the item when coping with latent faults. This can either be handled by coverage of faults through proper safety mechanisms or by the driver, recognizing that the fault exists before the violation of the safety goal. The higher the value of LFM, the more robust our applied safety mechanism will be. The following equation is used to determine the LFM:

$$LFM = 1 - \frac{\sum_{SafetyRelatedHW}(\lambda_{MPF,L,est})}{\sum_{SafetyRelatedHW}(\lambda - \lambda_{SPF} - \lambda_{RF})} \tag{4.3}$$

$$\lambda_{MPF,L,est} = \lambda \times (1 - \frac{K_{DC,MPF,L}}{100}) \tag{4.4}$$

where $\mathrm{K}_{\mathrm{DC,MPF}}$ is the diagnostic coverage with respect to multiple point latent faults and $\lambda_{\mathrm{MPF}}$ is the estimated failure rate with respect to multiple point latent faults.

**Probabilistic Metric for Random Hardware Failures (PMHF):** The PMHF evaluates the residual risk of violating a safety goal due to single point faults, residual faults, and plausible dual point faults. It defines the quantitative target values for the maximum probability of the violation. The following equation estimates the failure rate for the failure modes of each hardware part that would cause a single point, residual or dual point fault (ISO 61508):

$$PMHF = \sum \lambda_{SPF} + \sum \lambda_{RF} + \sum \lambda_{MPF,latent} \tag{4.5}$$

### 4.3.3 Reuse of Safety Artifacts in Hardware IPs

Trawling through datasheets to determine failure rates for hardware components is a cumbersome task. The information about hardware safety properties mostly comes from the vendors themselves. Currently, there is no standardized way to provide information about safety in IPs to tool vendors (EDA) or system integrators. To do this and subsequently achieve interoperability and reuseability with other tools, we propose an extension to a well-known format in industry: IP-XACT. IP-XACT is a standard (IEEE 1685) driven by Accellera and its format is used for documenting IPs using meta data. The data are used for configuring, integrating and verifying IPs in advanced SoC design and interfacing tools. The specifications are derived from the requirements of the industry to enable an efficient design of electronic systems. The 1.4 release of the IP-XACT format also includes implementation models on RTL and TLM level. This format also supports the data exchange through a common structured data management. Today IP-XACT is used by many different major tool vendors in the embedded system domain, and several European research projects are working on extensions and standardization, such as [16], [74] or [20].

An IP-XACT model can consist of different files in relation to the IP, such as design files, behavioral models, simulation files and results. It also consists of detailed information about the hardware such as parameters, ports, memory or configuration. The aim of the standard is to support a component-based design of hardware and to enable the reuse and assembly of hardware components like cores (processors, co-processors, DSPs), peripherals (memories, DMA controllers, timers, UARTs) and buses (simple buses, multi-layer buses, cross bars, network on chip).

Additionally, the IP-XACT format also provides vendor extensions to support user-defined features. Vendor-specific IP meta data can be stored in a vendorExtension element. These extensions can be applied to several elements in the hierarchical manner of the IP-XACT format (components, bus interfaces, registers, etc.). We use the capabilities of IP-XACT to add safety properties to the different elements of the IP. The vendor extensions are composed in a hierarchical manner. The root container can contain one or several vendor extensions. For our purpose, we add the following properties to the elements:

**Failure rate (FR)** - is usually known by the vendor of the component. It is a result of field return and statistical data, where expert judgment can also be considered.
**Failure modes (FM)** - describes the different modes where a failure can occur. The failure modes depend on the application in which the element is used.
**Safety mechanism (SM)** - implemented mechanism to detect and control faults. It prevents faults from violating the safety goal. If a fault is detected, a safe state is initiated.
**Diagnostic coverage (DC)** - is the effectiveness of the internal safety mechanism implemented to cover single point, residual or latent faults.

Safety properties can be defined on the top level or on a very detailed level (subsystem level) of the different components. The level on which the failure modes and safety mechanisms are described depends on two factors:

- If a failure mode is comprehensive across several components and cannot be assigned to a dedicated unit, it must be described on a higher level.
- Safety-related data is very sensitive information and needs years of research and field tests. Therefore, it must be protected because of proprietary reasons. Depending on the use case and how much information one wants to relinquish, very detailed information or only top-level information is used to describe the safety of an IP.

Just like the majority of the approaches in this domain [74], we describe our safety properties not on hardware detailed level, but rather on an architectural level. The benefit here is that we achieve a much higher level of abstraction, which is closer to the system design. It also leads to a faster evaluation of the hardware design and brings important inputs for potential faults in system and software design. More information about this approach is given in Paper B.

## 4.4 Generation and Integration of SaVeSoC

### 4.4.1 Safety Aware Virtual Prototype

In the previous section, we described how to design our system in the UML standardized modeling language using extensions such as MARTE and SysML. We also showed how to derive technical requirements from a first functional specification in simulation, which brings important inputs for our final system design. After the design of our hardware platform, we are able to evaluate our architecture including design space exploration regarding the requirements for functional safety. With the help of our extensions for safety-properties such as failure modes and FIT rates in the IP-XACT standard, we are able to execute mandatory methods such as hardware architectural evaluation, FTA or FMEA on the hardware level. All these methods help us to strengthen the reliability of our system and furthermore bring evidence for the technical correctness of the hardware design, which is also strictly required by the safety case in the end.

Besides the before mentioned verification methodologies, the functional safety standard also recommends to use design walk through and design inspection for the platform, but more importantly hardware prototyping and simulation for higher ASIL levels. This virtual prototype can then be used for further hardware verification such as fault injection tests, which is key nowadays for testing the dependability of the system. Several research institutes are now working on executing fault injection, also on a higher abstraction level such as TLM. This has the advantage that this method can be applied on faster simulation models without losing information from the more detailed models (RTL).

Virtual prototyping has the benefit that embedded software can be tested much earlier before a first real hardware prototype is available. Also the hardware/software interface can be tested towards consistency. Changes on a virtual hardware design are much faster than changes on the real platform, which takes weeks or months of redesign and production, which in turn has an impact on time-to-market. With intensive simulation, corner cases but also long-term reliability errors can be encountered, which also prevents costly product recalls. Environmental impacts on the virtual prototype can be simulated and reproduced, where real testbeds are not capable of this kind of verification. Instead of building several physical prototypes, different hardware design alternatives can be easily explored through virtual prototyping. At the end of the development phase, the final prototype can be tested towards consistency, correctness, and completeness with the functional specification. The drawback is that, a complex VP is not developed overnight. It takes a lot of effort, experienced designers and engineers to build a so-called digital twin of the actual hardware. The VP should have a modular architecture and be flexible in creating the platform. Depending on the test application, it should also provide different levels of abstraction to distinguish between several levels of detail, since simulation on a detailed level can consume immense computing power and time. Furthermore, it should ease the way to verify the hardware platform and the embedded software and not require months of building the virtual prototype for testing. Our proposal is thus to reuse models for virtual hardware prototyping from open libraries such as Open Virtual Platform (OVP), [64]. OVP comes with a growing model library, which offers processor cores, memory, and various peripherals. The idea is a modular design with the combination of components in a so-called virtual platform. The generated platform can be simulated with the OVPsim API. OVP was founded by Imperas, its commercial brother, and can be used freely for non-commercial/academic use.

One of the outlined goals in this work is to develop and test embedded software within the development phase, on a realistic hardware prototype of the target system, before the actual platform is available. Embedded software is often written in a desktop environment, using a general-purpose operating system of the host system. This approach often differs significantly from the target platform and parts of the written software need to be adjusted. One way to deal with this problem is the usage of an Instruction Set Simulator (ISS) and hardware visualization. Due to a variety of vendor components within SoC design, the simulation can be very difficult. Hardware emulators are very popular for this issue, but require the detailed RTL description of the developed system, which is a contradiction to the outlined goals. OVP makes it possible to create virtual platform models, with SystemC TLM 2.0 support. OVP models can be executed much faster than their counterparts developed in RTL since their level of abstraction is higher but still appropriate for modeling purposes.

The instruction set simulator OVPsim is released for 32 bit Windows and Linux and is a just-in-time code morphing simulator engine, which means that the target instructions are translated to x86 host instructions. This causes a significant speed-up since the simulation can be highly optimized after that.

OVPs model Generator iGen was written to build simulation models through a Tool Command Language (TCL) script, which contains used platform components and their connection. TCL files and IP-XACT hardware descriptions, rely on the Vendor Library Name Version (VLNV) principle, a standardization of the Spirit consortium. VLNV establishes a unique identification for models by providing the parameters Vendor, Library, Name, and Version. The directory structure of the Imperas model library was designed in a similar manner so that the iGen converter retrieves the necessary information for the instantiated models of the platform and generates a SystemC description. The OVP library comes with different peripheral models from different vendors. These models implement communication interfaces, such as UARTs, I2C, Ethernet or Controller Area Network (CAN), which is of special interest in the automotive area. Those interfaces work purely digital such that message exchange can be established to other control units. The task of the BMS is to monitor battery temperature and voltage, which are analog values in SystemC AMS and to compute the state of charge and state of health. Therefore, the usage of an Analog Digital Converter (ADC) is necessary, to build an accurate system description.

### 4.4.2 Configuration and Seamless Integration of Virtual Prototypes

Since OVP consists more or less of a set of SystemC files including a TCL script, one goal of this thesis is to include the whole generation of the virtual prototype into our design flow for safety-critical systems. A further goal is the design and configuration of the VP using a graphical modeling standard, in this case UML/MARTE. This approach brings the advantage to evaluate and analyze the configuration of the hardware design before the actual prototype is generated from UML models. OVP itself does not support verification regarding safety, nor is OVP now embedded in a seamless design flow.

Since our whole methodology (from functional specification to hardware and software design) relies on the same modeling language and furthermore the same hardware description language respectively system-modeling language, we are easily able to reapply our generated safety-aware virtual prototype for the functional specification of the system design. After generation, the hardware description of the SaVeSoC platform with the whole interface specification is added to the UML model library and can be reapplied to the system design including the generated simulation files in SystemC. This saves time in terms of integration effort, when testing the virtual prototype on the functionality of the whole system, as recommended in ISO26262. System-level testbenches can also be reused for the verification of the entire system including the integrated VP. The whole process

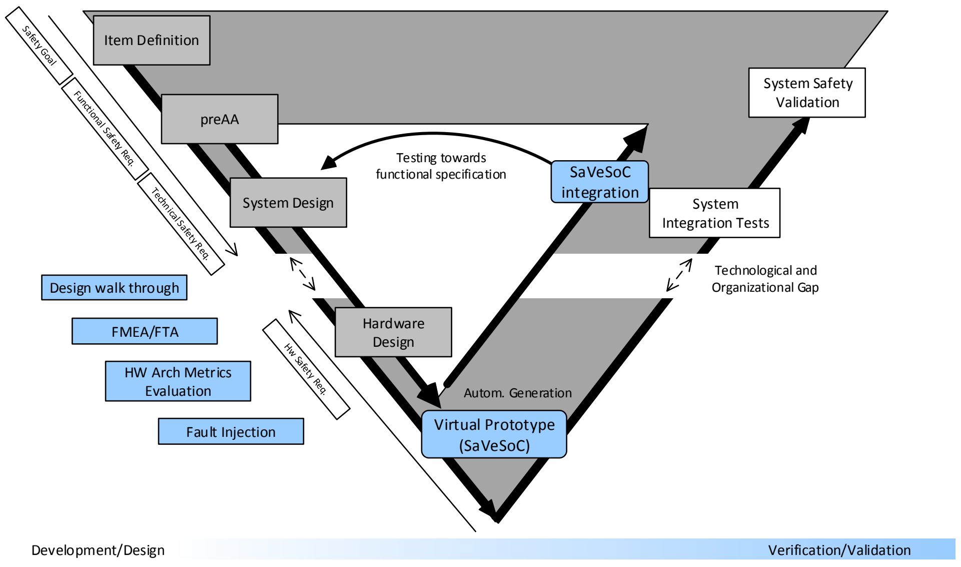


**Figure 4.13:** Process of SaVeSoC integration into functional specification.

is depicted in Figure 4.13. By adding a smaller V-model to the traditional approach, we are closing the technological and organizational gap between system design and hardware development, which exists in today's tool flows. Since our design and simulation languages in use share a seamless development flow, no information transfer is needed between those design levels. Changing requirements in the specification can also be easily and efficiently tested for the virtual prototype.

Figure 4.14 depicts the resulting system-level description including the battery and newly defined BMS hardware. The battery component is no longer a black box where the functionality is described in SystemC. It is now a white box, where the detailed hardware of the battery component is specified. It consists of the SaVeSoC element, which is the hardware platform of the BMS including an application for plausibility checks. It measures the voltage and temperature of the battery pack over two ADC and computes the state of charge and state of health. The interfaces of the battery model remained the same since no changes have been made to the interface description. Since we are relying on the same simulation engine, the virtual prototype can now be easily tested at a higher level, which also speeds up the overall simulation time. A challenge when integrating the VP to the functional specification was the communication interface between the functional simulation environment of SystemC AMS and the OVP platform because these platforms are mainly used to process data measured from embedded sensors. We thus implemented communication channels, which guarantee data exchange between different SystemC dialects. The authors of [49] rely on the idea that the simulation engine is

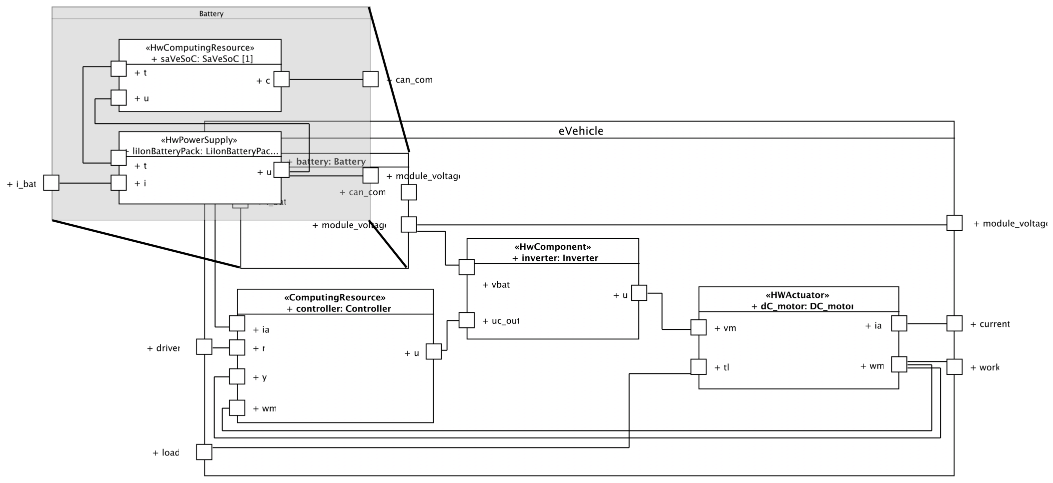


**Figure 4.14:** SaVeSoC integration into functional specification.

executed in a single process, with the SystemC and OVP simulator running in different software threads. The authors mainly focused on the capability to co-simulate SystemC RTL models, with either Quick Emulator (QEMU) [69] or OVP. To avoid overheads from the use of sockets they established the communication channel via a shared memory and synchronization mechanism. Furthermore they developed a SystemC bridge to enable the connections to the external hardware simulator. Since the original component library is not meant to be cycle accurate, the main focus was set to establish the communication between the existing SystemC AMS components and the TLM2.0 models provided in the OVP. Paper [23] describes the main properties of both sides and how synchronization is performed internally. SystemC AMS provides so-called converter ports to establish a connection between Timed Data Format (TDF) modules and an ordinary SystemC signal. In the event of access to such a port, the AMS kernel triggers an interrupt, which causes a context switch to the SystemC/OVP simulator. The crucial part of the implementation was, therefore, the conversion from SystemC AMS Linear Signal Flow (LSF) to TLM and how to handle the data stream of an arbitrary LSF module to the ADC peripheral of the OVP platform.

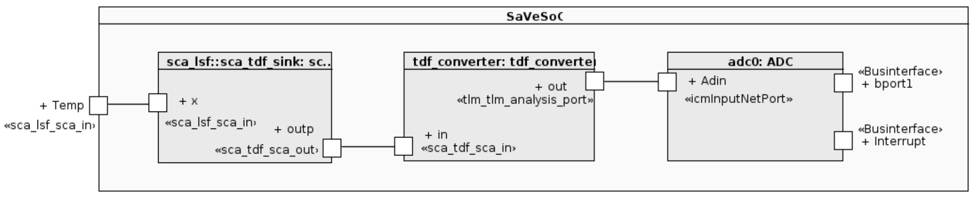


**Figure 4.15:** Communication channel to interact between SystemC AMS and OVP TLM2.0.

Figure 4.15 shows the used components for converting the initial LSF signal to a TLM signal. The *sca_lsf::sca_tdf_sink* is a component of the SystemC AMS library, used to sample arbitrary input data and convert it to tdf. The self-defined *adc_module* processes the tdf signal so that it is written to the *Adin* port of the *adc0* within its processing() procedure. The *adc0* is part of the OVP library and was adapted slightly to meet our needs. The port of the ADC is implemented with a call back function, which triggers the conversion of the ADC. Afterwards it can be read with the implemented driver, executed on the CPU. Since the OVP module requires the usage of a certain *tlm_signal_port* as TLM target socket for the communication, we adapted and advanced the approach of [23] to meet our needs. Secondly, we omitted the suggested internal First In First Out (FIFO) regarding data loss. This can be reasoned by reference to the constant sampling rate of the AMS simulation. A fixed size FIFO would nevertheless lead to data loss if the processor does not execute sufficient instructions per second.

# 5 SaVeSoC Implementation

This chapter gives a brief overview on the technologies and software projects that resulted in the SHARC framework. In the frame of the SHARC project, the SaVeSoC process with all its novel methodologies was implemented.

## 5.1 SHARC Framework

### 5.1.1 System Component Library

To avoid the design and simulation of larger systems from scratch and furthermore achieve reusability, our developed methodology provides as core component a System Component Library (SCL) (Fig. 5.1 a)). This library includes all major components for the simulation of systems from different domains e.g. automotive, mobile computing or multimedia. It also includes components in different versions and more importantly on different abstraction levels. These models serve on one hand as the starting-point for future development and on the other hand as a golden reference for integration aspects. The components are modeled as UML-Class in a composite structure diagram as depicted in Fig.5.2 b). The UML-class owns the attributes and properties of the component. Our example shows a UML-Class named Li_IonBatteryPack, tagged with HRM *PowerSupply.* To describe the inputs and outputs of the battery, the ports are tagged with MARTE FlowPorts. The stereotype PowerSupply allows us to define different configurations for the simulation e.g. multiplicity (number of cells), power supply, capacity or frequency of the battery. Besides this also non-functional properties for power like energy consumption or dissipation are used for the parametrization of the battery. The mapping between MARTE and SystemC is described by the work done in publication Paper E.

To raise the reusability and provide good support for developers, the SCL is built as an Eclipse plugin. New models can be generated and added to the library. Updates for components can be easily checked by updating the library from the server. This helps to support design teams by adding new components and keeps the library consistent.

The executable models used in our approach are models on different abstraction levels and in different versions. The generic models have the potential to be used in various domains and support reusability. The detailed models are refinements of the generic models and have the purpose to be used in special domains. Dependent on the domain

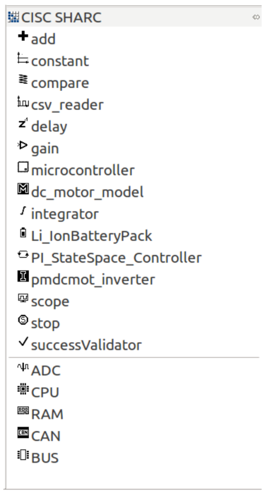


**Figure 5.1:** a) SCL pallet.

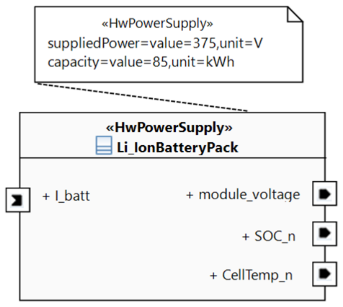


**Figure 5.2:** b) Example component from the SCL (battery model).

and abstraction level, the models are built in SystemC TLM and SystemC AMS. All the information for the configuration of the system is extracted from the UML-files by the framework at runtime. To speed up the simulation time, we created already compiled binary files from implementation code. This has the advantage of being able to parametrize or even reconfigure systems and components without the need for recompiling the code every time the system is simulated. This approach was evaluated regarding performance and accuracy towards a state of the art simulation approach such as Matlab Simulink in publication Paper E.

### 5.1.2 SysCore

Another core element of the SHARC framework is named SysCore (short for SystemC core). It was developed as part of the OpenES and eRamp project at the Technical University of Graz concerning modularity and simplicity, with support from Markus Schuss and Martin Schachner. Since the SysCore was intended to be also used in distributed environments, the goal was to rely on view external libraries. Moreover, the goal was to reduce the footprint in memory (RAM and disc space). It is made up of four major components:

- The Main File
- The Config Store
- The Parser
- The Factory

The *Main File* parses the arguments of the UML design and writes the information to the central config store element (*configstore*). It calls the parser and starts the simulation. The *configstore* is a singleton which stores all the information and is accessible by all plugins and components.

The *Factory* is responsible for the discovery of all local and global plugins which are registered by their name. Each plugin contains four basic functions that are called by the parser when required (*createObject*, *createPort*, *createTimestep*, *createComposition*).

The most complex component of the SysCore is the Parser. Its purpose is to translate an UML model defined in one or more files to a single SystemC AMS model. This is done at run-time and does not require compilation of the resulting model. When the parser is first created, it requires the model name for the top model as well as the name of the UML file that contains the model. The involved steps of this approach are depicted in Fig. 5.3. While most of the process is straight forward, there is one potential "loop" if a class has attributes other than ports (usually properties). This node is treated as the new root node, and the submodel is created before moving on to the connection creation. Each submodel may contain any number of submodels of its own. Therefore this step may repeat any number of times. It is important however to know that it is possible in UML to model a class that contains a property of the type of that very same class. This way the parser would not terminate. This is why such constructs must be avoided. We will now elaborate a little further the steps involved in the process.

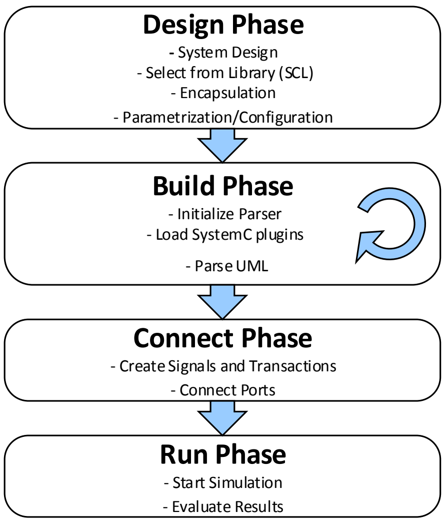


**Figure 5.3:** Parser methodology for executable SystemC models from MARTE design.

The first part of our methodology is described as the ***Design Phase***. The designer creates an UML top class such as the eVehicle example in Fig. 4.5. This class describes the overall structural architecture of the system. It is composed of the different instances provided by the SCL. They can be easily included and excluded from the top-class by using the drag and drop mechanism. These sub-systems are connected to each other by ports according to their specification. Through the MARTE flow-port capability, these ports are checked in advance by the framework concerning the correct direction of the data flow e.g. two output-ports are connected together. To trace the data flow, *Scope*-models are added to the system and connected to the ports. These *Scopes* write the monitored data to the trace files to compare the results from different tests and test benches in a latter step.

Our framework also provides a mechanism to encapsulate existing components and to raise the abstraction level of the whole system. Smaller systems that model the interior design of the class can be merged to a more simplified model. This helps the designer to have a better view of the system, without having too many details in the models on system-level. This mechanism allows us to abstract the complexity of components.

The whole eVehicle system also referred to in our case DUT, is provided with several connectors. These signals required for testing and debugging are brought out to the ports of the top-class itself. This has the advantage of connecting test benches to the DUT for testing various scenarios of the electric car and also for monitoring the performance. The outcome of the design step is a netlist that also serves as configuration and parametrization for the simulation. It is the starting point for the *Build Phase*.

The heart of the ***Build Phase*** is the self-defined parser methodology. The purpose of the parser is to translate a UML model defined in one or more files to a single SystemC system. This is done at run-time and does not require compilation of the resulting model. When the parser is initialized, it needs the name of the top-class of the model in the diagram as well as the name of the UML file that contains the model. Starting from the root node of the UML model, each child of a node is parsed and returned. As single systems can be composed of more detailed sub-systems, we had to define a loop to find all properties and ports of each root note. Each node found in the UML file is treated as the new root node, and each sub-system is created before moving on to the *Connect Phase*. Each system may contain any number of subsystems. This is why this step is done in several iterations till all properties of the root node are found. In order to keep the framework extensible, a DLL-based plugin system is used. The information is stored in the *configctore* element.

In the ***Connector Phase*** phase the connector objects are created to link the different instances in the *Build Phase*. Depending on their nature, the connector objects can be signals or transactions. It is important to notice, however, that UML allows multiple 1:1 connections per port, SystemC merely allows a port to be bound once, but a signal may connect any number of ports (basically 1:n as only one driver is allowed per signal). As a means of handling these issues, both ends of each connector are tagged by an ID. Instead of creating new signals for connecting to a used port, the old signal is reused.

After all nodes, ports, and properties of the UML file are found by the parser, the SystemC instances created and connected, the simulation is started (***Run Phase***). The results of the Scopes are saved as trace file. This file contains all the relevant information required for the verification of the model or to evaluate the behavior of the model for different parameters and/or implementations for the system. Besides this, logic can also be added to the system to react to certain events such as stopping the simulation in case of a signal violating the given constraints or the system running out of energy (for a battery powered system). An implemented dialog is also used to configure the settings for the simulation such as duration or timestep (resolution).

Using the Toem Impulse plugin [84] for Eclipse the results are presented in a graphical form. The results can be displayed in the desired manner, dependent on the nature of the simulation e.g. analog interpolation for real values and numeric representation of digital signals in a hierarchy that allows for easy interpretations. The results may be verified against the known or expected behavior of the (physical) system modeled. If the system behaves as expected, it can be used for further analysis or verification (e.g. as a golden reference model or synthesizable).

## 5.2 SHARC IDE

The software for the user side of the simulation framework depicted in 5.4 is based on the open source Eclipse environment. It is built around existing plugins such as Papyrus [33] (UML editor) and Impulse [84] (for visualizing the trace files of the simulation) and plugins for online and offline simulation. Papyrus was extended by a number of plugins to only show relevant information by using a custom theme as well as to allow the user to easily instantiate predefined library components. This abstraction of information is especially important to allow designers to only use a predefined set of diagrams and components, since UML comes with a variety of different ways to describe the structure and behavior of the system. The setting of constraints helps users to have a common and consistent way to describe crucial parts of the design. As the Eclipse platform runtime is easily extendable via plugins, new components can be added later on and managed via the build-in software updater.

The plugins specifically created for SysCore can be split into two principal sections:

- *Visual Plugins:* These plugins contain mainly papyrus extensions like the theme, information abstraction as well as the palette plugin which allows the user to instantiate new components for the library.
- *Simulation Plugins:* These plugins are used to either simulate a created design offline or as either a single simulation or batch online. Additionally, the creation of test benches for an existing design including verification of outputs.

Fig 5.5 shows an overview of the software stack used in the SHARC framework. It

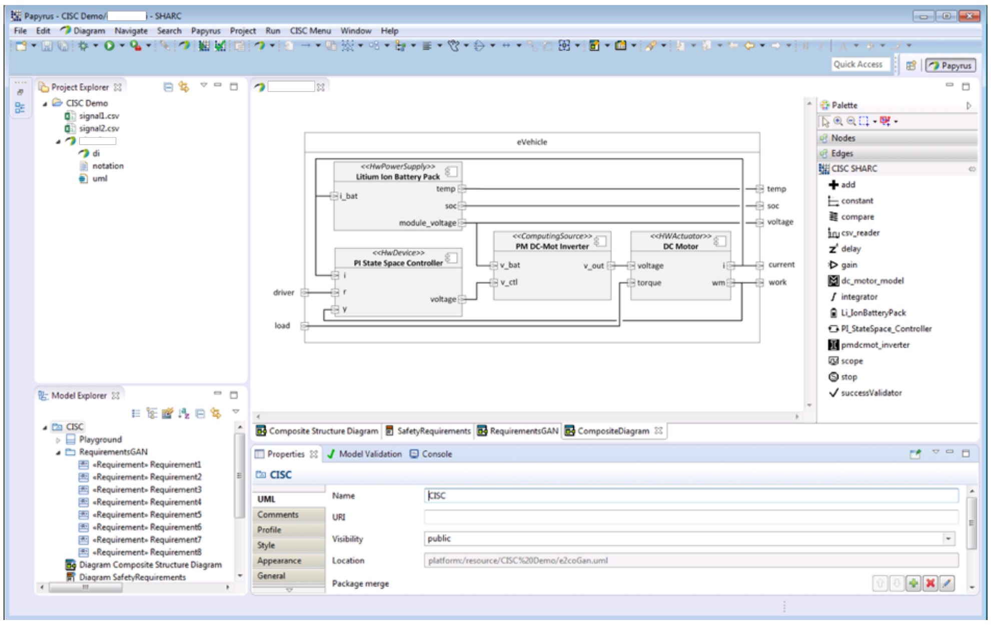

Papyrus - CISC Demo/ - SHARC
File Edit Diagram Navigate Search Papyrus Project Run CISC Menu Window Help
Project Explorer
CISC Demo
signal1.csv
signal2.csv
di
notation
uml
eVehicle
<<HwPowerSupply>> Litium Ion Battery Pack
<<HwDevice>> PI State Space Controller
<<ComputingSource>> PM DC-Mot Inverter
<<HWActuator>> DC Motor
i_bat
temp
soc
module_voltage
v_bat
v_out
v_ctl
voltage
torque
wm
driver
load
current
work
Palette
Nodes
Edges
CISC SHARC
add
constant
compare
csv_reader
delay
gain
dc_motor_model
integrator
Li_IonBatteryPack
PI_StateSpace_Controller
pmdcmot_inverter
scope
stop
successValidator
Model Explorer
CISC
Playground
RequirementsGAN
«Requirement» Requirement1
«Requirement» Requirement2
«Requirement» Requirement3
«Requirement» Requirement4
«Requirement» Requirement5
«Requirement» Requirement6
«Requirement» Requirement7
«Requirement» Requirement8
Diagram Composite Structure Diagram
Diagram SafetyRequirements
Composite Structure Diagram
SafetyRequirements
RequirementsGAN
CompositeDiagram
Properties
Model Validation
Console
CISC
UML
Comments
Profile
Style
Appearance
General
Name CISC
URI
Visibility public
Location platform:/resource/CISC%20Demo/e2coGan.uml
Package merge

**Figure 5.4:** Screenshot of the SHARC IDE including Papyrus UML editor and Impulse plugins.

lists the different plugins and technologies used and where they are implemented. This includes the plugins for the graphical user IDE and the distributed simulation environment including master and worker. Figure 5.6 shows a more detailed view of the C part of the software. It shows the data flow from the UML files to the final results. The highlighted entries are files available for download using the web-interface.

## 5.3 Distributed Simulation Environment

Using the described technologies in the previous sections, an initial concept and final design was created. The master handles the retransmission if a message could not be transmitted e.g., in case of a breakdown of a worker, as well as the overall scheduling, which would not have been feasible using only a webservice. Figure 5.7 illustrates the overall flow of information from the creation of the workpackages by the user to the upload of the results by the worker. The workers shown in this diagram may be in a public or private cloud, physical machines running the software bare metal or a mix of all three.

As Amazon is currently the largest provider of cloud services worldwide, it was chosen as the target platform, but due to the running cost of hosting instances on EC2 it was decided that a local alternative was required during the development phase as well as an option for customers requiring that the data stays in-house. OpenStack can be accessed via an API compatible to Amazons and was therefore chosen as development platform for

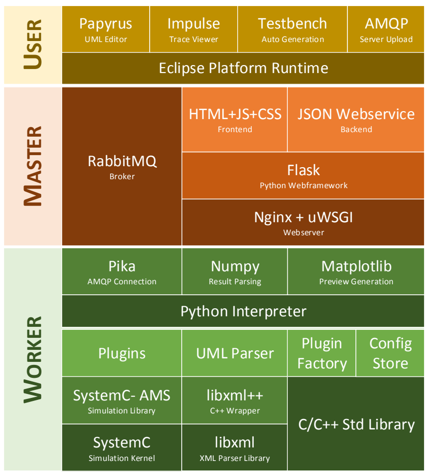


**Figure 5.5:** Software stack of the SHARC environment, adapted from [76].

the framework. For the purpose of development an All-In-One solution was chosen for the OpenStack deployment, and due to the constraints in the network setup, the setup was configured by hand. The bulk of the communication is based on AMQP (with RabbitMQ as its implementation) due to the build-in redundancy and robustness. The message throughput is a magnitude higher than the expected workload of the final system. The web interface is currently hosted by a Nginx due to its small size and high performance although Apache could also be used with uWSGI (as currently done by OpenStack Horizon).

Using the worker queue pattern described in section 4.3.1 the worker has been implemented in Python using Pika for AMQP communication. The worker connects to the broker running on the master and uses a heartbeat signal to indicate it has not crashed or is otherwise unavailable. The simulation is executed as a child process so the worker can keep sending heartbeat signals during that time. As the simulation core is unable to use multiple threads for simulation the ideal setup for a worker is a virtual machine that has only one CPU. Should such a setup be unavailable or physical machines be used instead it is, however, possible to spawn multiple worker threads on a single machine as well. Workers do not require a public IP address or connection to the internet; they only need to be able to connect the master and the webserver (which can run on a single machine). The framework does not impose any limitations on the number of workers. Usually there are as many workers as possible at any given time. They can, however, be easily spawned on demand and destroyed if no longer needed (if necessary even during a running simulation as it will be rescheduled by the master in that case). Workers do not need a persistent state as all information for simulation is gathered from the webserver (UML workpackage) and the master (SysCore configuration).

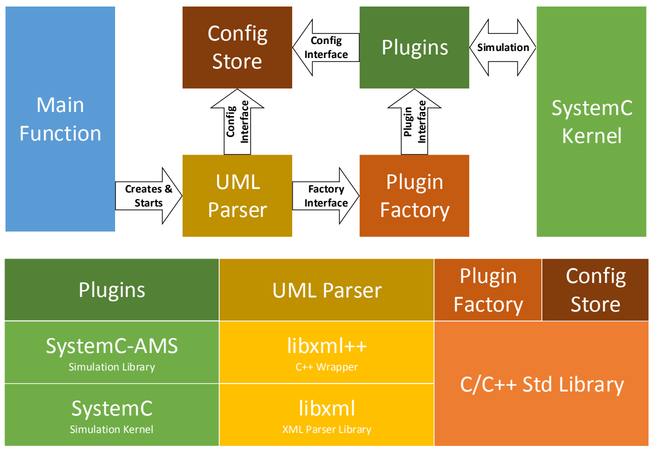


**Figure 5.6:** Interactions between the SHARC software components, adapted from [76].

The master is responsible for accepting and distributing simulations tasks to the workers. As with the workers, the master does not store information (other than the persistent queue) and therefore needs only a minimal configuration. Most Linux distributions include a package for RabbitMQ as well as Erlang (the programming language used for RabbitMQ).

The Webserver is currently the only server that actually stores data. It is written entirely using Python and Flask. Once again Ubuntu 14.04 with python 2.7.6 was used and Flask 0.10.1 was installed using pip (Pythons own package manager). In order to increase performance, Nginx (1.4.6) and uWSGI (1.9.17) are used to balance the load among several Flask instances running in parallel. Standard load balancing setups can be used to increase the number of webservers but the current setup uses only one such server. The webserver also hosts a SQL database (using MariaDB version 5.5.46, a MySQL fork) which would need to be replicated among all webservers as well or deployed on a separate set of database servers to improve the performance even further. Every simulation task can have a single workpackage (an archive containing all UML files required for the simulation (including any IP libraries needed) assigned to it which is the stored on the webserver using a file upload. This feature does not scale as well, but could be replaced by an OpenStack Swift object storage server/cluster. The current storage backend, however, only writes the uploaded file to the filesystem of the webserver. The webserver hosts two types of sites: The JSON API for interaction with other software (e.g., optional Eclipse plugins) as well as a human readable HTML website. The HTML portion uses Twitter Bootstrap for a reactive layout and has been tested on several desktop PCs, notebooks and mobile devices (such as smartphones). For every task the workers create simulation results and store the trace files, the configuration used and the output logs of the simulation core to the server

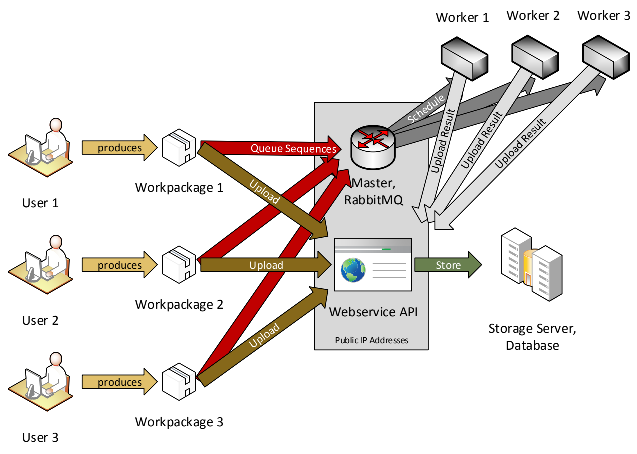


**Figure 5.7:** Concept of the distributed simulation environment, adapted from [76].

using the file upload feature as well.

While the results of the simulation are not relevant for this work, the achieved speedup is highly relevant. Due to the limited amount of overlapping resources (mainly network and hard-disk) the simulations are not affecting each other even when using three workers on the quad core machine used for testing. One core was reserved for the overhead of the virtual machine of the master and webserver as well as the overhead from the OpenStack installation on the machine. This resulted in a virtually linear speedup when using more than one worker as long as the number of simulations is reasonably high. For single simulations, there is, of course, no speedup as the workers cannot share a single run.

# 6 SaVeSoC Evaluation

In this chapter, first we will show how the global design flow, developed as part of OpenES, has been evaluated regarding quantified metrics by all partners in the project. Secondly, we present the results of the standalone SaVeSoC process evaluation and its benefits when dealing with the design and verification of safety-critical embedded systems.

## 6.1 Evaluation of the advanced OpenES Design Flow

To goal in the OpenES project was to define a global design flow by all partners in the project. By leveraging the common definition and understanding of defined abstraction levels, sub flows of the case studies could be included into a common reference flow. The aim was to combine the models and methods developed during the project into an advanced system design methodology and its design flow. At the end this newly defined design flow shall be evaluated regarding quantitative measures to show the benefits of this approach. This has been done by executing the design flow on several industrial case studies:

- Software-defined-radio-based application for security domain
- Advanced software-defined-radio system on a chip in the car entertainment domain
- Advanced set-top-box for consumer electronics
- Verification of a battery management system in the automotive domain
- Multimedia use case

For these five cases studies (Fig. 6.2), the OpenES design flow was applied in the development process. To evaluate, validate and measure the efficiency of the advanced design flow, metrics have been defined to quantify the results. Each partner in the consortium defined their own metrics depending on their focus in the project. Nevertheless, there have been several overlaps between the partners to select some of them as primary metrics. These metrics have been compared to the effort spend without the advanced technologies and tools from the OpenES project. Since the complexity of the case studies (e.g., complexity 140%) have a higher level compared to the traditional approach (complexity 100%), these aspects must be taken into account in the evaluation. Furthermore, the partners distinguished between two abstraction levels, system level and architectural level. The evaluation of the OpenES design flow has been done in "Task 4.3 - Proof of whole Concepts" and is documented in deliverable D4.3 as part of the OpenES project. The evaluation results can be obtained in detail from Table 6.1.

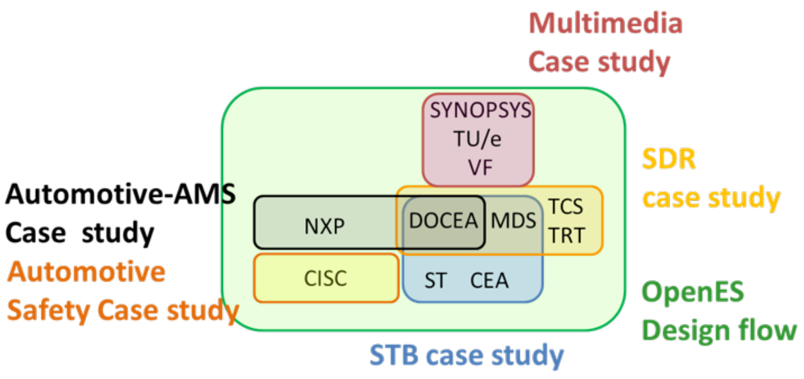


**Figure 6.1:** Collaboration matrix: collaboration between case study holder, tools and methodology provider in OpenES.

### 6.1.1 Definition of metrics

In order to measure the efficiency of the design flow three aspects have to be evaluated:

- Quality of the design
- Design effort
- Verification effort

Quality, design and verification effort are one of the main criteria for a successful design project and reliable and safe products. Requirements management is one aspect which improves quality by ensuring that the complete customer requirements are captured in a machine-readable way and verified against the actual implementation. Especially in the automotive domain quality is a main concern since standards such as ISO26262 require traceability between requirements, design, tests and results. Furthermore, a qualified design flow is one of the major requirements in a safety lifecycle. Of course, requirement management will consume extra effort at the start of the project. However, quality of the design can be increased by capturing all relevant customer requirements and duplication of tests can be avoided. Moreover, automation of test and verification can be achieved.

Partner 1 benefited from the OpenES flow by combining mixed-signal modeling with virtual prototyping, which allows a more extensive simulation of use-cases. They experienced a better controlled process to deliver higher-quality production tests with reduced risks. Moreover, compliance with lifetime safety standards (ISO26262) could be achieved

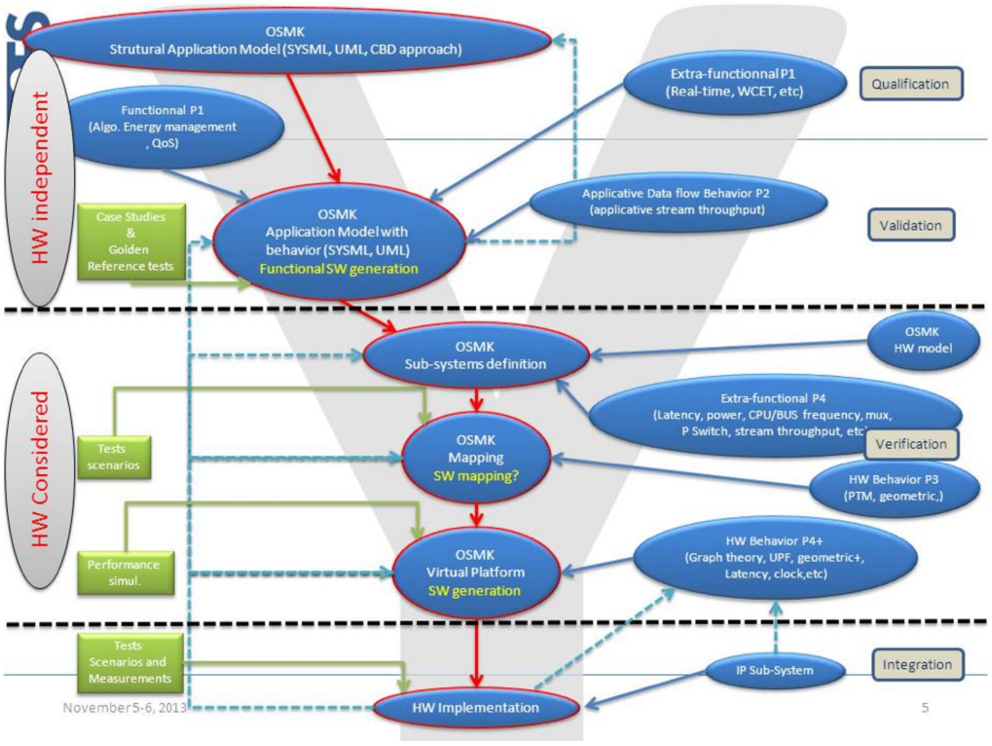


**Figure 6.2:** Global design flow as defined in the OpenES project, adapted from OpenES deliverable D4.1.

by advanced functional testing. This could not be handled without the fast executable models and defined interfaces. Thus, the proposed modeling approaches improve both quality and verification effort. By introducing mixed signal modeling, it required more effort at the start of the project. However during the verification of the design, the models reduced simulation time and enabled simulation of more use cases.

Furthermore, by using the IP-XACT standard for the hardware description a flexible configuration of the virtual prototype could be achieved, thus providing consistency with the specification and actual implementation. Furthermore, it allows a consistent memory map of the design and consistency by generating UVM-based verification views, which reduces verification effort. This enabled a faster and more complete verification, which helped to reduce the verification effort by 16%, measured over several projects. Moreover automatically generated documentation can be generated from the specification.

Partner 2 gained from the OpenES design flow by using novel modeling techniques for the integration of HW/SW IP subsystems into an advanced SoC infrastructure. Processing requirements of the subsystems have been captured in the models to assess whether they will meet the real-time requirements when integrating the subsystem in a SoC infrastructure. Therefore, a virtual prototype was developed to test towards these requirements. Thus, no extensive profiling of use cases for the SoC integrator was required, which made a more focused verification of critical use cases possible. System quality was improved through reduced change of system failures in the field, so the need for over-design. This approach led to a reduced design and verification effort by 55% (subsystem dimensioning and configuration).

Another advantage of the OpenES design flow is the capability to provide knowledge sharing across different domains and stakeholders, by bringing modeling and simulation to a higher abstraction level. Information in spreadsheets, block-diagrams or ad-hoc simulations by architects, which need to be shared between the teams are no longer efficient enough given the high complexity and performance of the targeted design. With a standard-based, earlier and high-level flow, information can be shared between platform architects, software designers, verification and validation engineers. Including the support to refinement or generation of design and testcases.

| | | | | | | |
|---|---|---|---|---|---|---|
| Complexity Reference Project | 100% | | | | | |
| Complexity Case Study | 128% | | | | | |
| | | | Final Savings (%) | | | |
| Partner | Partner 1 | Partner 2 | Partner 3 | Partner 4 | Partner 5 | Partner 6 |
| Metric | | | | | | |
| Reusability Aspect | 16 | N/A | 23 | N/A | 20 | N/A |
| Overall System Design | N/A | 16 | 15 | 36 | N/A | N/A |
| Performance Acceleration | N/A | N/A | 60 | 50 | 25,71 | N/A |
| Maschine readable models, incl. extra-functional prop. | N/A | 6 | 12,5 | 16,7 | 26,66 | N/A |
| Savings System Level (%) | 16 | 55 | 15 | 11 | 26 | 25 |
| Savings Architecture Level (%) | N/A | N/A | 33 | 23 | N/A | N/A |
| Overall System Level (%) | | | 24,67 | | | |
| Overall Architecture Level (%) | | | 28 | | | |

**Table 6.1:** Metrics definition and final savings from the OpenES design flow.

Partner 3 gained an overall of 27% in effort on system and architecture level by the newly developed methodologies in OpenES. They experienced an additional effort of 268% applied on the more complex case study used in the project, which could not be handled without the developed technologies. A significant design improvement is also to be expected by the definition of a common semantic and automated tools, which give access to the impact of performance/power consumption. The saved effort is expected to grow in the future, since the effort using the new methodologies will decrease, due to experience.

The savings of partners 4 have been revealed as 11% at the system level and up to 28% at the architecture level, demonstrated on the more complex case study. They could improve their global consistency of developments by using model-driven engineering techniques. Moreover, they observed savings up to one month compared to manual fixes of consistency bugs in IP APIs. Furthermore, they gained savings by code generation (from 20000 to 100000 lines of code) and automatic document generation per IP. Through extensions of the modeling standards at architecture level, post-silicon reuse of virtual prototypes could be achieved. Extra effort was spent in developing the modeling infrastructure, but higher savings in terms of effort using the more complex case study could be observed.

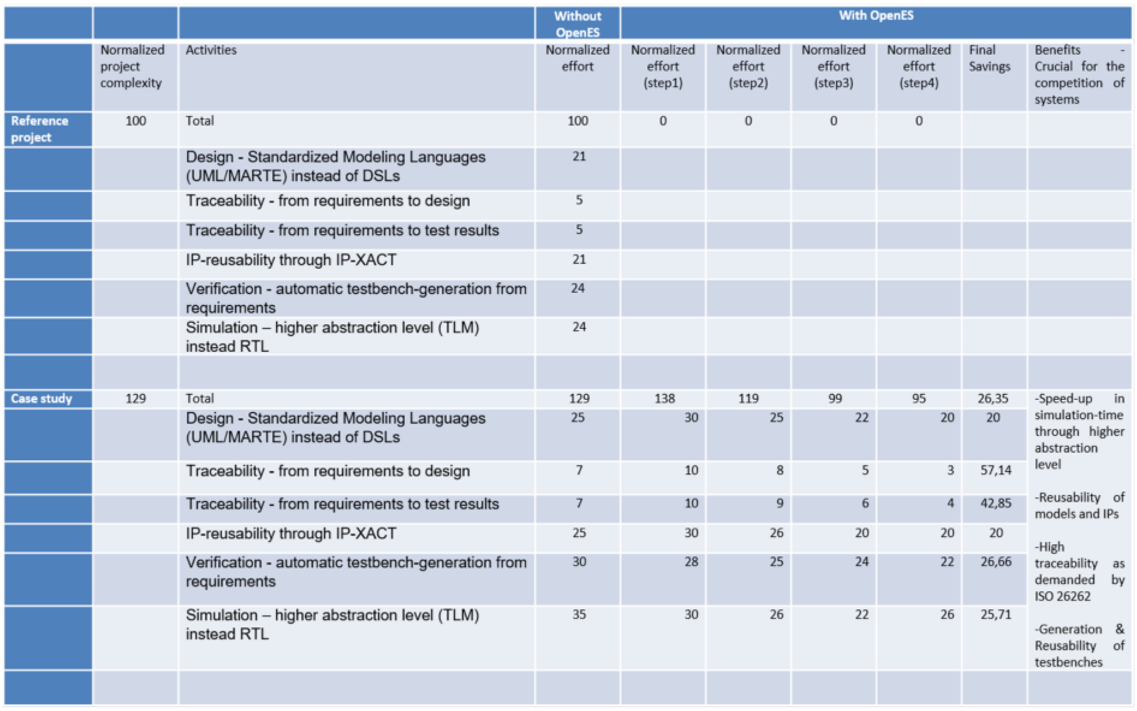
Table 6.1 shows the final savings on architecture and system level from the involved partners in the case study evaluation. From each independent partner evaluation four major metrics could be derived, which represent the comprehensive benefits of the OpenES design flow. All in all, the case study used for the evaluation was 128% more complex than the reference case used in current state-of-the-art projects. Because of the new technologies and tools embedded in a seamless design flow, the high increase of complexity in future projects could be handled. Overall the project consortium achieved savings reaching from 25% on system level and 28% on architectural level. The savings are expected to grow in the future since the effort in using the new design flow will decrease as the new design flow establishes.

## 6.2 Evaluation of the SaVeSoC Design Flow

With the tool SHARC we achieve a tight and seamless integration of analytical methods and simulation-based verification in the design flow of ISO26262. To evaluate design at all stages, a simulation-based verification of UML/MARTE design models on preliminary Architectural assumption (preAA) level was proposed with reusable components from the SCL. The properties of the models are all taken from the standard definition for UML/MARTE system, hardware and software models. In order to bring the components of the SCL to life, they are linked to executable models in SystemC TLM or SystemC AMS.

Based on the functional SR from the functional safety concept, defined as SysML models, and the information from the preAA, further requirements were obtained for the technical safety concept. Through taking also non-functional properties (timing, power, thermal) into account, the functional SR were refined and the technical SR have been defined. Furthermore inputs for final system design were obtained, before costly implementation of faulty design.

Testbenches in the UVM, to test the design on preAA level through simulation are automatically generated from the information and constraints of the functional SR defined in SysML. Furthermore, constraint random verification helps to cover all possible parameters and variants of the system, but also to vary environmental conditions, to find corner cases. These testbenches can be used throughout the whole development cycle towards the final



| | | | Without OpenES | With OpenES | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Normalized project complexity | Activities | Normalized effort | Normalized effort (step1) | Normalized effort (step2) | Normalized effort (step3) | Normalized effort (step4) | Final Savings | Benefits - Crucial for the competition of systems |
| Reference project | 100 | Total | 100 | 0 | 0 | 0 | 0 | | |
| | | Design - Standardized Modeling Languages (UML/MARTE) instead of DSLs | 21 | | | | | | |
| | | Traceability - from requirements to design | 5 | | | | | | |
| | | Traceability - from requirements to test results | 5 | | | | | | |
| | | IP-reusability through IP-XACT | 21 | | | | | | |
| | | Verification - automatic testbench-generation from requirements | 24 | | | | | | |
| | | Simulation – higher abstraction level (TLM) instead RTL | 24 | | | | | | |
| | | | | | | | | | |
| Case study | 129 | Total | 129 | 138 | 119 | 99 | 95 | 26,35 | -Speed-up in simulation-time through higher abstraction level |
| | | Design - Standardized Modeling Languages (UML/MARTE) instead of DSLs | 25 | 30 | 25 | 22 | 20 | 20 | |
| | | Traceability - from requirements to design | 7 | 10 | 8 | 5 | 3 | 57,14 | |
| | | Traceability - from requirements to test results | 7 | 10 | 9 | 6 | 4 | 42,85 | -Reusability of models and IPs |
| | | IP-reusability through IP-XACT | 25 | 30 | 26 | 20 | 20 | 20 | |
| | | Verification - automatic testbench-generation from requirements | 30 | 28 | 25 | 24 | 22 | 26,66 | -High traceability as demanded by ISO 26262 |
| | | Simulation – higher abstraction level (TLM) instead RTL | 35 | 30 | 26 | 22 | 26 | 25,71 | -Generation & Reusability of testbenches |
| | | | | | | | | | |

**Table 6.2:** Table with reference projects and case study measures, adapted from OpenES deliverable D4.3.[16].

system integration and validation. By generating a first virtual prototype from the existing hardware description in UML/MARTE, the process of testing different configurations of hardware prototypes and their verification against the functional specification could be advanced.

A mature goal of CISC in the context of OpenES was to move from a DSL, like it was used in CISC's deprecated tool System Architecture Designer (SyAD), towards a common design language (UML/MARTE) which is also used by many companies and partners in this field. Therefore, we observed and measured the effort spend from moving to another design language but also increasing the abstraction level (system level) in the team. Another focus was to enhance the traceability from requirements to design, but also from tests back to requirements, since this is now mandatory and required to conform to standards such as the ISO26262 for functional safety. This includes the definition of requirements and constraints in a machine-readable semi-formal way. Due to using IP-XACT in CISC's IP-library (SCL) we are now able to reuse models throughout the whole lifecycle of ISO26262 on different levels of abstraction, but also across different design and simulation tools. Furthermore, we leveraged the reusability aspect by extending IP-XACT with safety properties, which brings an enhancement and speed-up in the design but furthermore in verification/evaluation of safety-goals. CISC's developed testbench generator to automatically create UVM like verification components to evaluate functional and non-functional properties brings high savings in the evaluation of the first system design. Moreover, generation and integration of virtual prototypes from the hardware specification in the IP-XACT standard in different configuration, allows efficient testing towards the system safety requirements. Furthermore, using simulation on TLM level, instead RTL level on system level, brings an important speed-up in simulation of thousands of tasks.

The whole approach, implemented in SHARC, has been evaluated by four embedded design experts, including probands from CISC Semiconductor GmbH and Technical University of Graz. The criteria for the evaluation were the quality of the design (regarding functional safety) and the design and verification effort. These criteria have been derived from the requirements of the industry (time-to-market), design experts (usability, simplicity) and standards (ISO26262). Subsequently, the results have been compared regarding quality and effort in CISC's previous design tool SyAD. The results can be obtained from 6.2. Column *Without OpenES* indicates the effort spent (100%) without the newly developed methodologies and tools. The new case study in the project was 29% more complex (129%) than the case study applied to SyAD. In contrast to SyAD's emphasis on being a tool for co-simulation of heterogeneous embedded systems, SHARC's focus is on safety-criticality of embedded systems, which requires different verification techniques on several levels of detail. Also, the high traceability aspect between requirements, design, and tests was not given in SyAD, which is one of the core elements in ISO26262. Thus, we have to admit that the comparison between those tools was not straightforward but gave a good indication and pointed out the benefits of the new tool, the process and its methodologies regarding design and verification of embedded safety-critical systems. By establishing new methods for requirements management, higher quality of safety aware design could be achieved, thus avoiding duplication of tests. Moreover, the number of test runs could also be increased by fast simulation models, thus enhancing functional coverage. Reduction of verification effort by automatically generating test benches derived from safety requirements and their reusability. Furthermore, the sharing of knowledge across different domains and stakeholders by bringing modeling and simulation to a higher abstraction level. The newly developed methodologies in OpenES including the OpenES design flow bring an average saving of 26%, applied to the new and more complex case study. This value will grow in the future since the effort using the new methodologies will decrease.

# 7 Conclusions

This chapter concludes this doctoral thesis by briefly summarizing the contributions and potential future work and research.

## 7.1 Summary and Conclusion

In this thesis we presented a seamless design and verification process for safety-critical systems. A standardized modeling language based on UML was used to represent the design flow, from functional specification down to hardware and software. This model-based approach eases the communication between different stakeholders involved in the development process and serves as a single-source of information. Through tight integration of recommended safety analysis methods such as FTA, FMEA, hardware architectural metrics and simulation-based verification, we achieved consistency, correctness and completeness throughout the development process. The hardware architecture was evaluated by extensions to a well-known hardware description in the industry, IP-XACT. Existing and reusable hardware description was used for system design and integration. Our tool-aided method helped to speed up the evaluation process, and to reduce costs through reusability. The evaluated hardware description was then used to automatically generate a safety aware hardware virtual prototype, which was used to test correctness regarding the functional specification. This closes the technological and organizational gap in today's toolchain of safety-critical system development. Furthermore, this early virtual prototype can be used for fault-injection tests, as recommended by the functional safety standard. In addition our approach was developed as a plugin for the Eclipse, with the result that every Papyrus UML editor can be used for safety aware development of cyber-physical systems, simply by adding our plugin. This tool is named SHARC (Simulation and verification of HierARChical embedded microelectronic systems) it is to be published for download and is also used for educational purposes.

## 7.2 Future Work

This doctoral thesis contributes to state-of-the-art safety-critical system development in the automotive domain, it does not claim to be exhaustive or the holistic solution. Therefore, there is space for future work and further improvements and the following paragraphs propose ideas and directions for future work with respect to the major contributions.

### 7.2.1 Fault Injection on TLM Level

One of the most popular reliability analysis is fault injection. For ASILs higher or equal C, it is one of the highly recommended testing methods in the functional safety standards ISO26262. Fault injection testing can be applied at different abstraction levels such as physical, software-based, fault emulation or model-based just to name a few. Since they operate on different abstraction levels, fault injection techniques can be applied at different development stages. When we talk about fault injections testing in SystemC, usually this is done on Register Transfer Level (RTL). This is a broad area of research, and there are several different fault injection techniques published on this topic in international and scientific papers. Since our first virtual prototype is generated at a higher level than RTL, we propose to use fault injection on Transaction Level Modeling (TLM). This approach would shorten the time for simulation runs, thus increasing the coverage for the same simulation time. Several research institutions are currently working on a solution for fault injections tests on TLM level [82], without losing coverage compared to RTL. As a result, these techniques could also be integrated into our design and verification process for safety-critical systems.

### 7.2.2 Security

For the acceptance and use of Cyber-Physical System (CPS) and Internet of Things (IoT) in the automotive domain, issues of security play an increasing role. The danger of cyber attacks in automotive is present as we can see from the example in [90], where hackers took over control of a JEEP Cherokee to point out vulnerabilities and weaknesses in today's road vehicles. A proverb says *"there is no guaranteed safety without security"* [8]. This has also been recognized by several research institutes [86] but also international standardization bodies [42], where they try to integrate security aspects in safety standards. This is a highly advanced and difficult task since security threats can be occur in different ways. Security analysis must take into account software, hardware, communication channels and interfaces, remote access or even physical damage. It is very challenging to apply security analysis in early stages of development where hardware platform and interfaces are not specified in detail or do not even exist. This would be a potential research topic to be integrated into our proposed design and verification flow.

# 8 Publications

In the course of this doctoral thesis scientific essays have been published in several highly rated domain-specific conferences, book chapters and workshops. Most notably (in chronological order):

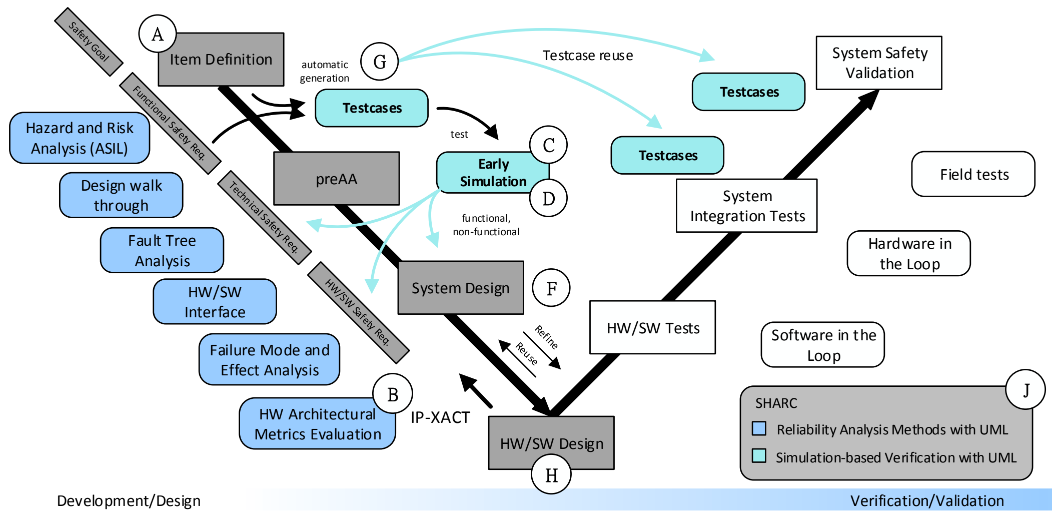


**Figure 8.1:** Overview of the contributions of this thesis that include techniques and tools that are applied during development to enhance the safety and reliability of the complete system.

A. Ralph Weissnegger, Christian Kreiner, Markus Pistauer, Kay Römer, and Christian Steger. “A Novel Design Method for Automotive Safety-Critical Systems based on UML/MARTE.” in: *Proceedings of the 2015 Forum on specification & Design Languages (FDL)*. Barcelona, Spain, 2015, pp. 177–184. ISBN: 9791092279092

B. Ralph Weissnegger, Markus Pistauer, Christian Kreiner, Kay Römer, and Christian Steger. “A novel method to speed-up the evaluation of cyber-physical systems (ISO 26262).” In: *12th International Workshop on Intelligent Solutions in Embedded Systems (WISES) 2015, Ancona, Italy, October 29-30, 2015*. Ed. by Massimo Conti and Simone Orcioni. IEEE, 2015, pp. 109–114. ISBN: 978-8-8875-4808-2. URL: `http://ieeexplore.ieee.org/xpl/freeabs{\_}all.jsp?arnumber=7356991`

C. Ralph Weissnegger, Markus Schuss, Christian Kreiner, Markus Pistauer, Kay Römer, and Christian Steger. "Simulation-based Verification of Automotive Safety-critical Systems Based on EAST-ADL." in: *Procedia Computer Science - The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016).* Vol. 83. Madrid, Spain, 2016, pp. 245–252. DOI: `10.1016/j.procs.2016.04.122`

D. Ralph Weissnegger, Markus Schuß, Martin Schachner, Kay Römer, Christian Steger, and Markus Pistauer. "A Novel Simulation-based Verification Pattern for Parallel Executions in the Cloud." In: *Proceedings of the 21st European Conference on Pattern Languages of Programs.* EuroPlop '16. New York, NY, USA: ACM, 2016, 20:1–20:9. ISBN: 978-1-4503-4074-8. DOI: `10.1145/3011784.3011806`. URL: `http://doi.acm.org/10.1145/3011784.3011806`

E. Ralph Weissnegger, Markus Pistauer, Christian Kreiner, Markus Schuß, Kay Römer, and Christian Steger. "Automatic Testbench Generation for Simulation-based Verification of Safety-critical Systems in UML." in: *Proceedings of the 6th International Joint Conference on Pervasive and Embedded Computing and Communication Systems {(PECCS} 2016).* Ed. by Andreas Ahrens and César Benavente-Peces. Lisbon, Portugal: SciTePress, 2016, pp. 70–75. ISBN: 978-989-758-195-3. DOI: `10.5220/0005997700700075`. URL: `http://dx.doi.org/10.5220/0005997700700075` (not included in thesis)

F. Ralph Weissnegger, Markus Schuß, Christian Kreiner, Markus Pistauer, Römer Kay, and Christian Steger. "Bringing UML / MARTE to life : Simulation-based Verification of Safety-Critical Systems." In: *2016 Forum on Specification and Design Languages (FDL).* Bremen, Germany, 2016

G. Ralph Weissnegger, Markus Schuß, Christian Kreiner, Markus Pistauer, Kay Römer, and Christian Steger. "Seamless Integrated Simulation in Design and Verification Flow for Safety-Critical Systems." In: *Computer Safety, Reliability, and Security: SAFECOMP 2016 Workshops, ASSURE, DECSoS, SASSUR, and TIPS Proceedings.* Ed. by Amund Skavhaug, Jérémie Guiochet, Erwin Schoitsch, and Friedemann Bitsch. Trondheim, Norway: Springer International Publishing, 2016, pp. 359–370. ISBN: 978-3-319-45480-1. DOI: `10.1007/978-3-319-45480-1-29`. URL: `http://dx.doi.org/10.1007/978-3-319-45480-1{\%}5C{\%}5C{\_}29`

H. Ralph Weissnegger, Martin Schachner, Christian Kreiner, Markus Pistauer, Kay Römer, and Christian Steger. "SaVeSoC - Safety Aware Virtual Prototype Generation and Evaluation of a System on Chip." In: *Procceeding of the 7th International Symposium on Model-driven Approaches for Simulation Engineering, accepted.* Virginia Beach (US), 2017

I. Ralph Weissnegger, Markus Pistauer, Martin Schachner, Christian Kreiner, Kay Römer, and Christian Steger. "Generation and Verification of a Safety Aware Virtual Prototype in the Automotive Domain." In: *Handbook of Research on Solutions for*

*Cyber-Physical Systems Ubiquity, accepted* (2017) (not included in thesis)

J. Ralph Weissnegger, Christian Kreiner, Markus Pistauer, Römer Kay, and Christian Steger. "SHARC - Simulation and Verification of Hierarchical Embedded Microelectronic Systems." In: *Procedia Computer Science - The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2017).* Vol. 00. 2017. Funchal, Madeira, 2017

Additionally, this thesis is supplemented by the following peer-reviewed fast abstract and student forum papers:

K. Ralph Weissnegger, Markus Pistauer, and Christian Steger. "High Level Simulation of Cyber-Physical Systems." In: *The 10th International Conference on Scientific Computing in Electrical Engineering SCEE 2014,Program & Book of Abstracts.* Wuppertal, Germany, 2014, pp. 57–58

L. Ralph Weissnegger, Markus Pistauer, Christian Kreiner, Römer Kay, and Christian Steger. "A Novel Method for Fast Evaluation of Cyber-Physical Systems in Compliance with Functional Safety." In: *Euromicro Conference in Software Engineering and Advanced Applications - Proceedings of the Work in Progress Session.* Funchal, Madeira: Johannes Kepler University Linz, 2015, pp. 24–25

M. Ralph Weissnegger, Markus Pistauer, Kay Römer, and Christian Steger. "Simulation-based Verification of Automotive Safety-critical Systems Based on UML." in: *Microelectronic Systems Symposium (MESS).* vol. 83. Vienna, Austria, 2016, p. 32

# A Novel Design Method for Automotive Safety-Critical Systems based on UML/MARTE

Ralph Weissnegger*†, Markus Pistauer†, Christian Kreiner*, Kay Römer* and Christian Steger*
*Institute for Technical Informatics
Graz University of Technology (TU Graz), Austria
Email: (ralph.weissnegger, christian.kreiner, roemer, steger)@tugraz.at
†CISC Semiconductor GmbH, Klagenfurt, Austria
Email: m.pistauer@cisc.at

***Abstract*—The complexity of electric/electronic systems in today's vehicles is steadily growing. New challenges arise through highly distributed systems which interact with and have an impact on the physical world, so-called cyber-physical systems. There is a need for modeling languages like UML/MARTE to support engineers and managers throughout the whole design process to reduce costs and time to market. Especially when it comes to safety-critical systems, safety aspects must be handled on various abstraction levels from high level system description to detailed modeling of hardware and software. Not only functional but also non-functional requirements need to be taken into account here. In this paper, we present a seamless model-driven architecture approach to model safety-critical systems throughout the whole design phase of the functional safety standard ISO 26262. Furthermore, SysML is used to extend MARTE with semi-formal requirements to handle the issue with traceability. In order to demonstrate its efficiency, this methodology is applied to an industrial use case of a battery management system. The results show that MARTE is very suitable for modeling systems at any level of granularity in the automotive area, in compliance with functional safety.**

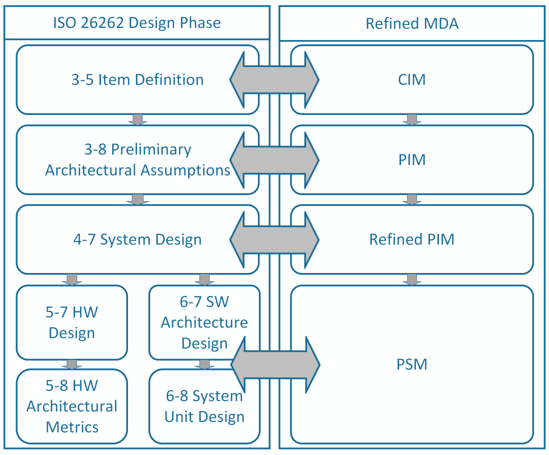


Fig. 1. ISO 26262 design phase to MDA mapping

## I. Introduction

Today's cars consist of highly complex E/E systems with sensors and actuators networking with each other, in fact a car is now more or less a smartphone on wheels. It can be observed that there is a shift towards fully E/E cars, since traditional combustion engines are slowly disappearing. The sensing and controlling of these systems is the work of the highly distributed electrical control units (ECU) and it's no surprise that up to 100 of these micro-controller are currently integrated in an electric vehicle [1], [2].

Recent trends in the in-vehicle E/E architecture and new applications brought a rapid shift towards multicore, heterogeneous, networked, and reconfigurable systems. The design and development of such systems is extremely complex and imposes an enormous challenge for designers (hard- and software) from different domains in designing their applications. The next generation of such systems should be able to run in parallel on different parts (ECUs and/or processors, DSPs within a multicore architecture of a single ECU) of the system. Applications are going towards multimedia, infotainment, advanced driver assistance systems (ADAS), navigation and many more. This has an impact on design, development and management and in turn increases production costs and time to market. This has been acknowledged also on European industry level [3], [4].

With the growing complexity in the automotive area one aspect is turning out to be the key issue for future vehicle development: safety. This is especially the case whenever systems interact with and have an effect on the physical world, so-called cyber-physical systems, it is not longer sufficient to test a single behavior. The whole system must be validated as early as possible in the development cycle and at any level of granularity. This is also recommended by the ISO 26262 [5] standard for automotive E/E systems. The ISO 26262 is an adaption of the functional safety standard IEC 61508 and compliance is currently required for OEMs and suppliers of E/E systems. The ISO 26262 supports managers and engineers throughout the whole product lifecycle on different abstraction levels. As a variety of system assumptions and design solutions needs to be taken into consideration, a model-based approach is an important basis for engineers and multiple stakeholders. It helps designers to have a quick and augmented view of the system and provides an effective way for communication, especially if systems are very complex and involve a number of teams in the design.

A way to model such systems is MARTE. This is an extended profile to UML2 and provides capabilities for modeling hard- and software, as well as timing and performance behavior. It is used at present by many semiconductor vendors and suppliers [6]. Today, MARTE is not very common in the automotive domain but with the newly electrification of vehicles and thus more and more components are related to E/E systems, MARTE could well help to save development costs and time in the future. Furthermore, it is the driven system-design language in the European Catrene-project OpenES [7]. OpenES is a European initiative to fill the gaps in today's system-design and to develop common solutions to stay competitive. A special focus is given to integral support for functional, but also non-functional requirements such as timing, thermal issues and power.

Another aspect of UML is that it is now supported by several commercial and open-source tools like Eclipse's Papyrus [8], that helps designers to model systems in UML and extensions like MARTE or SysML.

In this paper we present a way to model safety-critical systems on different levels of abstractions. We show that there are existing modeling languages that provides us with capabilities to represent the whole design flow of the functional safety standard ISO 26262 without compromises and helps us with additional features for safety analysis. We therefore use a refinement of the Model Driven Architecture (MDA), elaborated in the OpenES-project. We show that we can map the whole process from item definition and system design to hardware and software separation, to the model-based approach depicted in Fig.1. In our approach, each level in the ISO 26262 has an equivalent level in the MDA. This helps designers and engineers to keep a consistent view on all levels of the design phase. Furthermore we use the capabilities of SysML to model each requirement on different abstraction levels in the requirements phase to have a seamless allocation to our models and diagrams in the design phase. We also show that MARTE is very suitable for designing complex systems in the automotive area.

The paper is organized as follows: Section 2 presents the state of the art and related work. A short overview of functional safety and the safety lifecycle is given in Section 3. Section 4 describes the model driven architecture approach and the modeling languages in use. Section 5 presents our methodology applied to a case study for a battery management system. This is followed-up by the conclusion in Section 6.

## II. Related Work

How to use MARTE in a co-design process is discussed in several papers [9], [10], [11]. They show how MARTE complies with the model-driven architecture and the defined abstraction levels. In these papers the issue of how to model hardware and software on different abstraction levels in the design process is also discussed. The paper authors also address the issues of modeling extra-functional properties and the mapping from software applications to platforms. However, they do not consider traceability to SysML requirements, nor do they take modeling of safety constraints into account.

The authors of [12] present a concept how to apply the ISO 26262 in the development of a safety critical system. They address the system level as contained in part 3 (concept phase) and part 4 (product development at the system level) of the functional safety standard, but do not take detailed hardware or software modeling into account. Furthermore they do not use standards like UML for system-modeling, nor are they able to maintain a seamless flow throughout the design phase. This approach also does not show how to add behavioral diagrams to the flow, nor are safe states or other behavioral functions defined. Traceability to structural and behavioral diagrams is only partially covered.

How to use SysML as representation of requirements in the automotive industry and the functional safety standard is discussed in [13], [14], [15]. The authors show how to model the requirements on different abstraction levels in a semi-formal way. Also the traceability between the different levels in the requirements phase of the safety lifecycle are handled. One approach [14] extends SysML to define requirements of safety-critical systems. In [15] also the allocation to structural and behavioral models is taken into account. This approach shows how to use SysML for requirements on different abstraction levels very well. Unfortunately this method does not cover the whole design phase of the ISO 26262 and confines traceability solely to UML/SysML diagrams. We will build upon this approach in our paper to cover all traceability aspects as recommended in the standard. As none of these approaches consider MARTE as detailed modeling language for hardware and software, we will show how to use SysML to maintain the traceability to MARTE models on multiple levels.

One language that is established in the automotive area is EAST-ADL [16]. EAST-ADL is a language for the development of vehicle embedded electronic systems and in combination with AUTOSAR, the initiative to standardize software development. The language was developed in the context of the ITEA cooperative project EAST-EEA and further projects like ATESST [17] and MEANAD [18]. Since EAST-ADL is included in Eclipse Papyrus also SysML requirements models can be used to define requirements [19].

The language is structured in five abstraction layers, each with a corresponding system behavior: vehicle level, analysis level, design level, implementation level and operational level. EAST-ADL is built on top of AUTOSAR and covers only the abstraction levels from vehicle to design level [20]. The implementation and operational levels are modeled in AUTOSAR which makes the top down traceability and also the traceability to the requirements, as specified is required by the ISO 26262 standard, very cumbersome and error prone. One purpose of the ATESST and MEANAD project was to provide capabilities to map the functional safety standard to EAST-ADL abstraction levels. These levels are not in compliance with the model driven architecture, nor does this approach address all the design-levels of ISO 26262. Furthermore this language lacks of referencing timing and performance properties or other extra-functional properties, which are addressed in detail in MARTE.

As more and more systems in the automotive domain are now related to real-time and embedded systems, bringing the safety standard to MARTE is a next step in the development of safety-critical systems.

## III. Functional Safety

ISO 26262 is an adaption of the function safety standard IEC 61508 for automotive E/E systems. Since ISO 26262 is treated as state of the art in court, OEMs and their suppliers are required to comply with this standard today. It addresses hazards caused by safety related E/E systems due to malfunction and covers functional safety aspects through the whole lifecycle. It governs the identification, design, implementation and testing in form of an industry-standard V-model, called automotive safety lifecycle. The standards also provides an Automotive Safety Integrity Level (ASIL) analysis to specify the items necessary safety requirements for the development to hardware and software components. A safety goal is derived for each hazardous event with an ASIL-classification. The safety goals are the source for the whole chain of the safety lifecycle. This in turn results in a safety case, to show that the system is acceptably safe. The safety case is used to collect and present evidence, to support safety claims and arguments. In this work we address the left side of the V-model, from concept phase to product development. Since the right side addresses verification, testing and production it is not handled by our approach and is beyond the scope of this work. We use an MDA approach to demonstrate how we can model safety aspects in the design phase of the automotive safety standard.

## IV. Model-Driven Architecture

### A. Modeling languages

UML is a modeling standard of the OMG (Object Management Group) [21]. It is a graphical representation for specification and documentation of software and other systems. It delivers a complete view of the system, their individual components and the interaction between them. UML has different types of structural (e.g. Class, Component, Composite Structure) and behavioral (e.g. Activity, UseCase, State Machine) diagrams. Although this language is well suited for developing software, it provides no capabilities to design hardware and non-functional properties.

Another approach is SysML [22] as domain-specific modeling language. It uses a subset of UML2 and provides additional extensions to describe complex systems in system engineering. SysML supports UML2 by two additional diagrams (requirements, parametric) for requirements-engineering and performance-analysis. It provides a good mechanism for allocating requirements to components or behavioral diagrams, but is inaccurate in modeling hardware and resources.

MARTE was defined as an adaption from the OMG to address the shortcomings of modeling platforms in UML. MARTE [23], [24] is a domain-specific modeling language intended for model-based design and analysis of real-time and embedded software of cyber-physical systems. MARTE is defined as a profile in UML2 and provides additional mechanisms for modeling real-time systems, which are missing in UML. MARTE has the advantage of precise hardware and software resources in the form of *HRM* and *SRM* stereotypes. In addition it is possible to allocate software applications to hardware resources with the help of the MARTE allocation mechanism. MARTE follows the philosophy of cyber-physical systems to deal with whole systems rather than a set of specialized parts. This is also recommended by the ISO 26262 for the design of safety-critical systems.

The MDA approach has gained more importance as a result of a trend to pursue more formal modeling languages and greater exploitation. We can see from the definition of the different sub-profiles that MDA is also anchored in the MARTE language. The importance of this development has also been acknowledged on European level, where MARTE and the MDA approach have a significant part in the Catrene project OpenES. For our approach we only use standardized MARTE elements.

### B. Refined model-driven architecture

Since the levels of the standard MDA by the OMG were not adequately specified and lacked formal definition in the OpenES-project, partners elaborated a refinement of the MDA-approach (Fig. 1). This figure illustrates our mapping between the different levels in the design phase of ISO 26262 and the levels in the MDA. In our methodology, each level of the functional safety standard has an equivalent level in the MDA approach. The detailed definition of each level is given below:

**Computation Independent Model (CIM)** - aims at providing a system level view, mainly focusing on its functional structure. It does not specify any information on how the functionality will be implemented. In particular there is no hardware software identification. Moreover, this kind of model is not precise enough to execute models. Despite the lack of an explicit or implicit model of computation, CIM can include system level use cases showing synchronous and asynchronous communications between the different functional blocks. CIM is too abstract to specify any non-functional properties.

**Platform Independent Model (PIM)** - includes CIM capabilities with additional behavioral models like UML state machines or activity diagrams. Moreover, non-functional properties like timing, power, thermal issues or safety can be expressed at this level of refinement. PIM are not fully executable models, only a part of the whole system can be detailed more precisely. If the behavior is not directly expressed in diagrams, the models can be referenced to existing implementation code. These models can be used for an early and high-level simulation of the system-behavior. A PIM shows that part of the specification, which does not change from one platform to another.

**Refined PIM** - has been explicitly identified to fit with the OpenES sub-system definition concept. It consists in a functional decomposition of the PIM with a granularity detailed enough to allocate each of its blocks to a single hardware or software execution resource. In other words, a PIM functional block cannot be allocated on several execution resources. The functional blocks should be split beforehand into different sub-functionalities. Furthermore, their communication interfaces should be identified before mapping them onto different execution resources.

**Platform Specific Model (PSM)** - encompasses several aspects. It should first contain elements that will describe the execution platform, including hardware and software execution resources. On the hardware side, the model can contain low level details, such as registers and memory map information. On the software side, the execution platform description can specify OS-specific information, such as tasks, scheduling algorithms or middleware services. Complementary to those platform description aspects, the PSM can contain a new refinement of the PIM model where platform independent functional components are transformed into platform specific components, with explicit references to the execution resources services. The PSM part can be partially generated by the Allocation Model described below.

**Allocation Model** - is an intermediate step between the refined PIM and the full PSM. It expresses how refined platform-independent functional components can be allocated onto hardware or software execution resources. It implies that part of the PSM already exists, to identify and reference those execution resources. It can be the input of extra-functional properties analysis tools to verify if a given mapping will allow meeting expected extra functional property constraints. It can also be the source of code generation or model transformation to obtain detailed platform-specific application model. Since the allocation model is more like a link to a higher detailed model, it is not a separate level in our approach.

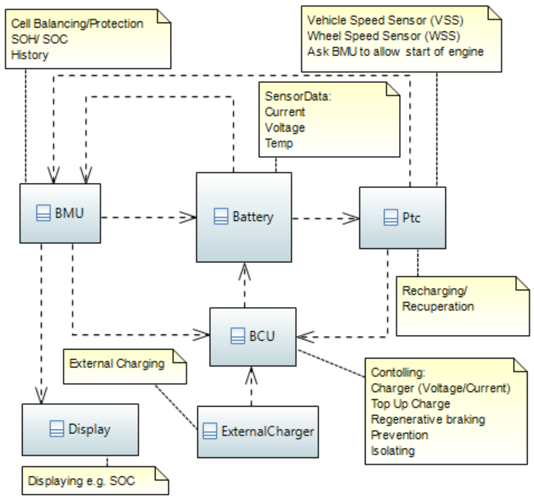


Fig. 2. **Item definition :** high-level functional view

## V. Example Case Study: Battery Management System

To show how we can use MARTE and the MDA approach through the whole design phase of the functional safety standard we demonstrate this by an industrial example of a battery management system with recuperation features provided by CISC Semiconductor. As more and more vehicles are now powered by Li-Ion-batteries, the challenge for engineers to ensure reliability and fault-tolerance of batteries is also greatly increasing. Problems with overheating or even explosions have been frequent in the past. The mainly cause of these problems was excessively high energy intake from regenerative braking or harsh environmental conditions. Management systems and mechanisms are thus essential to assure that persons are not put at risk and that no damage is caused. Safety mechanisms such as redundant and diverse measurements of the temperature and voltage of battery-cells decrease the occurrence of single-point, residual and multiple-point faults. Also the multiple and diverse calculation of sensor-data is an important measurement at a high integrity level. Since it is beyond the scope of this paper to examine all safety aspects of the item, we focus here on monitoring the state of the battery. The parts of the Battery Management System are explained in detail below:

**Battery Monitoring Unit (BMU)** - is the main controller of the battery. It measures different values coming from sensors of the battery-cells. The BMU computes the State-Of-Charge (SOC), State-Of-Health (SOH) and is responsible for cell balancing, cell protection and demand management of the battery. It also controls a hardware switch, which connects the battery to the electric motor.

**Battery Control Unit (BCU)** - controls the voltage and current profile of the charger output during the charging process. Besides controlling the charge of the external charger it monitors the regenerative braking charges and dumps it when the battery is fully loaded. If a fault occurs the battery can be isolated or the BCU sends a signal to the PTC to restrict the speed limit.
**Power Train Controller (PTC)** - is the main contactor between the battery and the electric motor. It controls vehicle and wheel speed and instructs the BMU, which monitors the state of the battery, to start the motor.
**Battery** - consists of 12 cells, connected in series. Each cell has a temperature sensor and connections to measure the voltage.
**Display** - shows the current SOC and SOH and warns the driver if a critical threshold is reached.

The first and most essential step in the development process in the context of the functional safety standard is to define the item. The definition of an item is a system or array of systems to implement a function at vehicle level to that the safety standard is applied. A system is a set of elements containing at least a sensor, controller and actuator, whereby an element can be a hardware or a software part. Figure 2 shows the item definition modeled as functional blocks by means of a UML composite structure diagram and informational flows. This provides a good view of the whole item at CIM-level with boundaries and interfaces to the environment. On this level we show how the functional blocks communicate with each other, but do not specify how the functionality will be implemented. A safety goal is derived with the help of the item definition and the hazard analysis and risk assessment, for each hazard. Each safety goal has its own ASIL-level that classifies the severity, exposure and controllability of the operating scenario. For this example an ASIL C safety goal was stated: *"excessive battery temperature must be avoided"*. The safety goals provide the basis for the functional safety requirements (FSR) specified in the functional safety concept (FSC), (ISO 26262, Part 3-8). A derived FSR for this example would be *"A BMS shall monitor and control the battery. The battery must operate in working range"*.
The ISO 26262 standard recommends different methods and design techniques to achieve safety on certain ASIL-levels. In this example we choose the heterogeneous duplex pattern to increase the reliability and availability of the system. The heterogeneous duplex pattern uses extra and diverse hardware components and has the advantage of being able to handle not only random but also systematic faults. Different hardware components will lead to the hardware reacting in different ways and also increasing the coverage of common cause failures. As our item has an ASIL C classification, the standard recommends using two independent and diverse signals to control the battery-cells, to achieve redundancy and diversity. We combine this with the decomposition-mechanism of the ISO 26262, which allows us to decompose our ASIL C sensor into two ASIL B(C) sensors in order to have the same classification but also to increase reliability by using independent redundant and diverse measurements. In the next step we make a refinement of our previously defined FSR to *"A safe state will be switched to, if the two measurements deliver different values of the battery-cells (plausibility-check)"*. The safe state is the desired behavior of the system in the event of a fault. It ensures the safe operation of a system. This corresponds in our case to a safe state such as *"Reduce the power (degradation function) or even shutdown the connection from the battery to the motor"*.

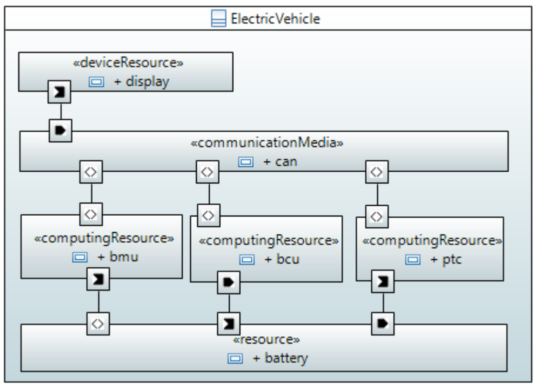


Fig. 3. **Architectural assumption:** a pre-version of the system design, no information about the used platform is given

The result of the functional safety concept is the preliminary architectural assumption illustrated in Fig.3, a pre-version of the system design, which is the actual solution to the functional requirements. In this diagram the relationship between the different components is more explicitly expressed and gives a more detailed view of the system. Moreover MARTE flow-ports (in,out,in_out) of component instances are also present, as well as connectors between them. On this abstraction level (PIM) we are independent of the actual implementation and therefore do not specify any technical details nor platform on which the function may be implemented. With the use of the MARTE general resource model (GRM) we are able to make our first assumptions regarding the system design. With the stereotype *"communicationMedia"*, properties such as capacity or transmission-mode for the CAN bus can be defined. The *"computingResource"* stereotypes is used in order to also model processing resources at a very high level of abstraction with no concern about the details of CPU speed or memory capacity.

At this stage additional behavioral models like activity diagrams or state machines are also added to describe the behavior of the system. These behavioral diagrams are allocated to the blocks of the architectural assumption. In Fig.4 the safe state is modeled by means of nodes and

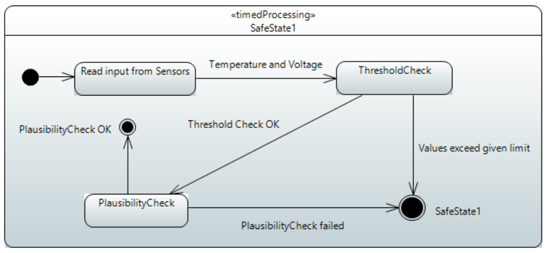


Fig. 4. **Safe state:** activity diagram with extra-functional timing properties

edges. First assumptions regarding timing constraints can also be considered in the activity diagram with the help of *"timedProcessing"* stereotypes. This is not shown in the figure but it is specified by a duration of *"value=30;unit=ms"* in the value specification modeling format (VSL). A more precise timing behavior of each node in the diagram can be made in a subsequent refined step, where more detail is given from the hardware software interface (HSI). An example would be the detailed description about system reaction or fault reaction time. It is the advantage of MARTE to capture timing information by means of qualitative and quantitative annotations in the description of the behavior. These expected behavior specifications can provide important inputs to perform model validation in later phases of the process. Another aspect of behavioral diagrams and timing constraints is to use them later for the generation of testbenches for simulation-based verification purposes. Furthermore they can be used for comparison with simulation results.

Now that the pre-version of our system design has been completed, the MARTE-models are coupled with preexisting implementation-models written in SystemC-TLM. This allows us to use a high-level simulation to have a closer look at the dependencies between the different components. For this purpose we use SHARC [25], an Eclipse-based tool under development for modeling and simulation of cyber-physical systems at different levels of abstraction. SHARC is an enhancement of SyAD/SIMBA [26] and uses co-simulation of various distributed components written in SystemC, Matlab or VHDL. It also allows us to switch to lower implementation levels of single components in the system, such as RTL-level simulation. This is particularly important for the verification of hardware safety mechanisms with methods like fault-injection and also recommended in the hardware design verification methods by the ISO 26262. The outcome of the preliminary architectural assumption, HSI, FMEDA/FTA and the hardware architectural metrics results in achieving the technical safety requirements (TSR). A FMEA/FTA on UML-models can be performed by approaches described in [27] or [28]. In this case-study example the TSR1 is defined as, *"Plausibility-checks every 30 ms of two analog sensors (temperature and voltage). If the difference of* 10 °C *is above the tolerance threshold for more than a certain time, go to*

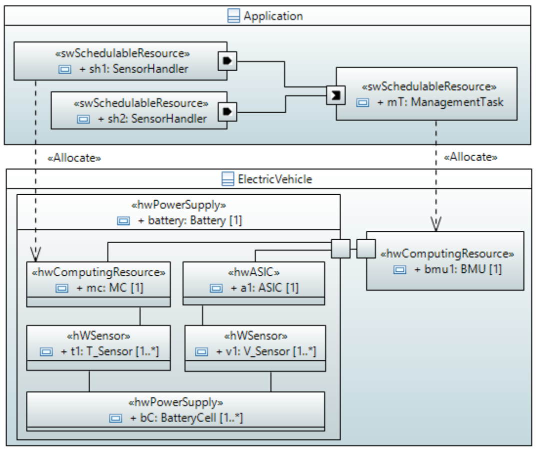


Fig. 5. **System design:** applications are allocated to the used hardware platforms

*safe state"*. This in turn results in a first version of our system design. At this stage we make our first assumptions about what we are going to realize in hardware and software. As defined in the OpenES refined PIM, the systems must be split to subsystems if no allocation from each block to hard- or software is possible (atomic). We now refine the architecture by adding two ASIL B(C) hardware sensors, as a result of our foregoing step where the ASIL C sensor was split into two ASIL B(C) sensors with the decomposition-mechanism. We achieve a vertical traceability throughout the ISO26262 levels by aggregating the components in the UML class diagram. For the modeling of our hardware-platform we use the MARTE hardware resource model (HRM). In Fig.5 the battery is split into a multiplicity of battery-cells, each linked to a voltage and temperature sensor tagged with MARTE *"hwSensor"*. Included are also two processing units, a micro-controller and an ASIC tagged with *"hwComputingResource"* and *"hwASIC"*, which are specializations of *"hwResource"* stereotype. These two processing units are used to calculate the data coming from the sensors, independent and redundant. Furthermore, they increase the fault tolerance of the system. To show that we have now a clear separation between software and hardware, we allocate the sensorhandler- and the management-task to our computing resources, micro-controller, respectively battery management unit. At this point where applications are allocated to platforms, a schedulability analysis in MARTE like described in [29] can be performed. This enables an early analysis of design alternatives before committing to a particular design for implementation. This is especially important in the design of safety-critical systems where resource-handling must be carried out very carefully. It must be ensured that common tasks like multimedia applications are not influencing or locking resources from safety-critical tasks that must perform in a given time. A

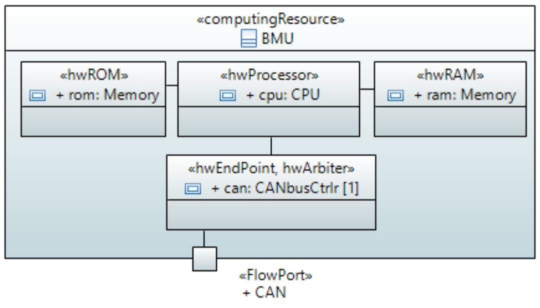


Fig. 6. **Hardware-description:** detailed structure of the battery management unit

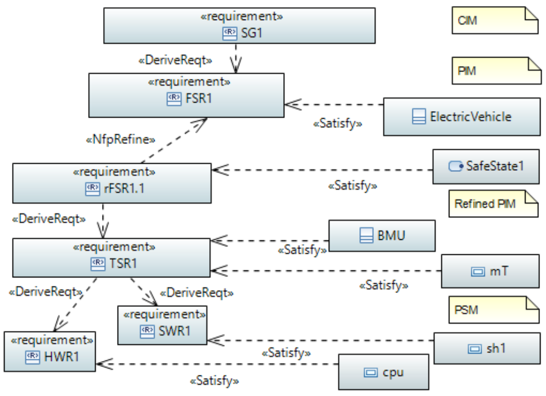


Fig. 7. **Requirements allocation to components and diagrams:** vertical and horizontal traceability

task may be switched to another processing unit that needs his own protected memory. This approach allows to analyze worst-case scenarios of different tasks allocated to processor cores.

Now that the functional blocks of the system are split into hardware and software components the definition of the requirements for hardware and software can be made. The final level (PSM) of the design approach, the hardware and software architectural design, is derived from these requirements.

The MARTE profile provides a set of concepts for hardware modeling that can be used to define very detailed models of computing hardware. In this example the hardware is designed by means of composite structure diagrams to graphically show the inputs/outputs and interfaces, depicted in Fig.6. The BMU designed on system level is now split into four detailed components: CPU, ROM, RAM and CAN. With the help of MARTE *HRM*, the blocks are tagged with specialized stereotypes to specify the properties. Also additional safety relevant properties required for the hardware design such as failure rate, safety-related or not, hardware safety mechanism and associated diagnostic coverage can be annotated to the hardware description. In this approach we also use a self-defined MARTE profile, for describing the hardware in the IP-XACT [30] standard. The software tasks are described as usual in the software design by traditional class diagrams. The MARTE software resource model (*SRM*) and also the high level application model (HLAM) are sufficient for defining constraints to our system including message size or memory size. As the next step operations are added to the management task for the BMU such as ThresholdCheck(), Display() or PlausibilityCheck(). Also timing-attributes tagged with MARTE timing notations are added to the task. As more and more details are attached, this models are latter used for automatic code generation for hard- and software in SystemC [31]. The Gaspard2 framework uses MARTE models to generate RTL code for synthesis or TLM code for simulation on a higher level of abstraction. This closes the gap from early safety-critical requirement analysis and system specification to simulation and synthesis.

A decisive issue which is often mentioned in the ISO 26262 standard is assuring traceability. Traceability starts with the safety goal and runs through the entire requirement and design phase, from derived requirements like FSR and TSR to behavioral and structural models. Traceability must be assured at each level of the lifecycle to support not only engineers and managers from different domains but also the argumentation in the safety case. In our approach the requirements tree is modeled in SysML and the relationship between each requirement and level is linked with SysML *"derived"* stereotypes. If the description of the requirement is not detailed enough, another requirement can be linked with *"refine"* stereotype. In Fig.7 we show that we not only achieve vertical, but also horizontal traceability through the whole lifecycle by allocating each requirement to the associated component or diagram. Another approach is to represent SysML requirements in a specified spreadsheet-like table-view in Papyrus. This provides a good overview of all requirements and their *"satisfiedBy"* relationship. This table is fully dynamic and immediately updated if relationships between requirements and models are added or modified. It also scales up for larger systems.

## VI. Conclusion

In this paper we demonstrated a methodology to model safety-critical systems at any level of granularity in the design phase of the functional safety standard ISO 26262, with the model-driven architecture approach (MDA). The specification and refinement of the MDA-levels were elaborated in the European Catrene-project, OpenES. A UML-profile MARTE for real-time and embedded systems was used to model safety aspects on all abstraction levels of the design-phase of the ISO 26262. We show that MARTE is very suitable for modeling E/E systems in the automotive area without using any extended self-defined UML-profiles. Through the use of standardized modeling languages we achieve a high reusability with other tools in this domain. Furthermore, SysML was used for horizontal and vertical traceability of requirements to components and behavioral models. With the link to MARTE diagrams we achieve a very high traceability level as demanded by ISO 26262. We showed the efficiency of this approach by applying the methodology to a battery management system. Future work will deal with the simulation of design models for verification of safety-critical systems. With the tool SHARC, MARTE-models will be linked to implementation models in SystemC or Matlab on various abstraction levels. In a next step, behavioral models will be used to automatically generate testbenches for simulation based verification.

## Acknowledgment

The approach presented above was experimented in the OpenES Catrene- project: CA703 - 2013 research program supported by CISC Semiconductor.

## References

[1] R. N. Charette, "This Car Runs on Code - IEEE Spectrum," 2009. [Online]. Available: http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code

[2] Etas, "Electronic Control Unit ( ECU ) - Webinar Basics of Automotive ECU ETAS Embedded Systems Consulting Embedded Software , AUTOSAR and Safety Consulting Chandrashekara N ( ETAS / ESC ) Lead Consultant Embedded Systems," pp. 1–30, 2014.

[3] Gartner, "Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020," 2013. [Online]. Available: http://www.gartner.com/newsroom/id/2636073

[4] ARTEMIS Industry Association, "2014 MultiAnnual Strategic Research and Innovation Agenda for the ECSEL Joint Undertaking," Tech. Rep., 2014. [Online]. Available: http://www.smart-systems-integration.org/public/documents/publications/2014ecselmasriapartc.pdf

[5] ISO, "Road vehicles Functional safety Part 1: Vocabulary," 2011. [Online]. Available: https://www.iso.org/obp/ui/\#iso:std:iso:26262:-1:ed-1:v1:en

[6] J. Medina, "The UML Profile for MARTE: modelling predictable real-time systems with UML."

[7] Catrene, "OpenES CATRENE Project: CA703 - 2013," 2013. [Online]. Available: http://www.ecsi.org/openes

[8] Eclipse, "Papyrus," 2015. [Online]. Available: https://www.eclipse.org/papyrus/

[9] L. G. Murillo, M. Mura, and M. Prevostini, "MDE Support for HW/SW Codesign: A UML-based Design Flow," *Advances in Design Methods from Modeling Languages for Embedded Systems and SoCs*, vol. 63, pp. 197–212, 2010.

[10] J. Vidal, F. D. Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet, "A co-design approach for embedded system modeling and code generation with UML and MARTE," *2009 Design, Automation & Test in Europe Conference & Exhibition*, 2009.

[11] A. Koudri and D. Aulagnier, "Using marte in a co-design methodology," *UML Workshop at Date'08*, 2008.

[12] W. Taylor, G. Krithivasan, and J. J. Nelson, "System safety and ISO 26262 compliance for automotive lithium-ion batteries," *2012 IEEE Symposium on Product Compliance Engineering, ISPCE 2012 - Proceedings*, pp. 6–11, 2012.

[13] D. D. Ward and I. Ibarra, "Development Phase in Accordance with ISO 26262," *System Safety Conference incorporating the Cyber Security Conference 2013, 8th IET International*, pp. 1–6, 2013.

[14] M. Adedjouma, H. Dubois, K. Maaziz, and F. Terrier, "A Model-Driven Requirement Engineering Process Compliant with Automotive Domain Standards," *Third Workshop on Model Driven Tool and Process Integration (MDTPI), Paris, France, June 16, 2010 Proceedings*, 2010.

[15] L. Muat, M. Hübl, A. Buzo, G. Pelz, L. Musat, M. Huebl, A. Buzoee, G. Pelz, S. Kandl, and P. Puschner, "Semi-formal Representation of Requirements for Automotive Solutions using SysML," in *2014 Forum on Specification & Design Languages (FDL)*, 2014.

[16] "EAST-ADL Association," 2013. [Online]. Available: http://www.east-adl.info/

[17] "ATESST," 2010. [Online]. Available: http://www.atesst.org

[18] "maenad.eu," 2014. [Online]. Available: http://www.maenad.eu/

[19] J. L. Boulanger and Q. Van Dao, "Experiences from a model-based methodology for embedded electronic software in automobile," *2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications, ICTTA*, pp. 1–6, 2008.

[20] P. Cuenot, D. Chen, S. Gérard, H. Lönn, M. O. Reiser, D. Servat, C. J. Sjöstedt, R. T. Kolagari, M. Törngren, and M. Weber, "Managing complexity of automotive electronics using the EAST-ADL," *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, no. Iceccs, pp. 353–358, 2007.

[21] "Object Management Group (OMG)," 2015. [Online]. Available: http://www.omg.org/

[22] "SysML.org: SysML Open Source Specification Project," 2014. [Online]. Available: http://sysml.org/

[23] "The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems — www.omgwiki.org/marte," 2013. [Online]. Available: http://www.omgmarte.org/

[24] B. Selić and S. Gérard, *Modeling and analysis of real-time and embedded systems with UML and MARTE*, 2014.

[25] R. Weissnegger, M. Pistauer, and C. Steger, "Program & Book of Abstracts," in *The 10th International Conference on Scientific Computing in Electrical Engineering SCEE 2014,Program & Book of Abstracts*, Wuppertal, 2014, pp. 57–58.

[26] C. Trummer, C. M. Kirchsteiger, C. Steger, R. Weiß, M. Pistauer, and D. Dalton, "Automated simulation-based verification of power requirements for systems-on-chips," *Proceedings of the 13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2010*, pp. 8–11, 2010.

[27] F. Mhenni and N. Nguyen, "Automatic Fault Tree Generation From SysML System Models," 2014.

[28] H. Kim, W. E. Wong, V. Debroy, and D. Bae, "Bridging the Gap between Fault Trees and UML State Machine Diagrams for Safety Analysis," *2010 Asia Pacific Software Engineering Conference*, pp. 196–205, 2010.

[29] L. S. Indrusiak, I. Quadri, I. Gray, N. Audsley, and A. Sadovykh, "A MARTE Subset to Enable Application-Platform Co-simulation and Schedulability Analysis of NoC- based Embedded Systems."

[30] SPIRIT, "IEEE SA - 1685-2009 - IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows." [Online]. Available: http://standards.ieee.org/findstds/standard/1685-2009.html

[31] E. Piel, R. B. Atitallah, P. Marquet, S. Meftali, S. Niar, A. Etien, J. J.-L. Dekeyser, P. Boulet, and I. Europe, "Gaspard2: from marte to systemc simulation," *DATE'08 Workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile*, vol. 8, pp. 1–6, 2008.

# A Novel Method to Speed-Up the Evaluation of Cyber-Physical Systems (ISO 26262)

Ralph Weissnegger*†, Markus Pistauer†, Christian Kreiner*, Kay Römer* and Christian Steger*
*Institute for Technical Informatics
Graz University of Technology (TU Graz), Austria
Email: (ralph.weissnegger, christian.kreiner, roemer, steger)@tugraz.at
†CISC Semiconductor GmbH, Klagenfurt, Austria
Email: m.pistauer@cisc.at

***Abstract*—The development of electric/electronic systems of today's vehicles is becoming more and more complex. New challenges are arising through highly distributed systems, so-called cyber-physical systems, which interact with and have an impact on the physical world. Methods and tools are thus essential to support the development process, especially when systems are safety-critical and demand reliability. In this paper, we present a novel method to decrease the design effort and speed up the verification of hardware. Our approach helps to avoid building safety-critical systems from scratch using industry standards like IP-XACT and UML/MARTE. Furthermore, our tool-aided method supports designers in making design-decisions for hardware very early in the development process. To demonstrate its efficiency, our methodology is applied to an industrial use-case of a battery management system. The results show that using our approach, it is possible to decrease development time and effort in the development of safety-critical systems.**

## I. Introduction

Today's cars consist of highly complex E/E systems with sensors and actuators networking with each other, in fact a car is now more or less a smartphone on wheels. It can be observed that there is a shift towards fully E/E cars, since traditional combustion engines are slowly disappearing. The sensing and controlling of these systems is the work of the highly distributed electrical control units (ECU) and it is no surprise that up to 100 of these micro-controller are currently integrated in an electric vehicle [1].

With the growing complexity in the automotive area one aspect turns out to be the key issue for future vehicle development: safety. This is especially the case whenever systems, so-called cyber-physical systems, interact with and have an effect on the physical world. It is no longer sufficient to test a single behavior. The whole system must be validated as early as possible in the development cycle and at any level of granularity. This is also recommended by the ISO 26262 standard [2] for automotive E/E systems, an adaption of the functional safety standard IEC 61508. Since the ISO 26262 is today treated as state-of-the-art in court, OEMs and suppliers are required to comply with this standard. The ISO 26262 helps managers and engineers throughout the whole product lifecycle in identifying safety aspects on different abstraction levels. The standard also provides an Automotive Safety Integrity Level (ASIL) analysis to specify the item's necessary safety requirements to avoid unreasonable risk due to malfunction. The ASILs are divided into 3 classes: Severity (S0-S3), Probability (E0-E4) and Controllability (C0-C3). After determination, the product is developed according to recommended methods and measures to its ASIL.

In this work we address Part 5 of the ISO 26262 standard: product development at the hardware level. This part includes guidance to avoid systematic and random hardware failures by means of appropriate safety mechanisms. Each safety-related hardware element is analyzed regarding safe, single-point (SPFM), residual and multiple-point faults (LFM). It also describes the effectiveness of the hardware architecture to cope with random hardware failures (PMHF). The diagnostic coverage gives evidence of the effectiveness of its safety-mechanism. Whether the item passes or fails a given ASIL is also a result of the architectural metrics evaluation. To achieve a certain ASIL, the values from Table I must be met. It is also important to point out that only safety-related hardware elements that have the potential to contribute significantly to the violation of the safety goal are addressed in this metric. This must be considered in the evaluation of the whole item.

TABLE I
ARCHITECTURAL METRICS - EVALUATES WHETHER THE HARDWARE ACHIEVES A CERTAIN ASIL

| | ASIL B | ASIL C | ASIL D |
|---|---|---|---|
| SPFM | >90% | >97% | >99% |
| LFM | >60% | >80% | >90% |
| PMHF | <100FITs | <100 FITs | <10Fits |

Safety-related properties like failure rate of components are published in various standards such as Siemens SN 29500 or IEC 62380 [3]. Usually this data is very general and not applicable to every domain. In some cases the source is unspecified and principally obtained from field or statistical data. This in turn can lead to false consequences, if the failure rate predictions differ significantly from field data. We propose that safety-related information for hardware-IPs should come from the vendor himself, as he knows the product best. Therefore our approach allocates safety-properties to vendor-IP in a standardized way, so there are no false assumptions about safety-critical hardware.

Since a variety of system assumptions and design solutions need to be taken into consideration, a model-based approach is an important basis for engineers and multiple stakeholders. It helps designers to gain a quick and augmented view of the system and provides an effective way for communication, especially if systems are very complex and involve a number of teams in the design.

One way to model real-time and embedded systems is MARTE. It is an extension of UML2 and provides capabilities to model hard- and software, as well as timing and performance behavior. It is nowadays used by many semiconductor vendors and suppliers [4] and is the driven system-design language in the European Catrene-project named OpenES [5]. The OpenES is a European initiative to fill the gaps in today's system-design and to develop common solutions to stay competitive. A special focus is given on integral support for functional, but also extra-functional requirements like timing, thermal and power. Another emphasis is given on the enhancement of interoperability of models and tools by upgrading and extending open standards like IP-XACT [6].

Usually the evaluation of SPFM, LFM and PHMF of hardware architectural design in compliance with functional safety is done in a top-down manner. We propose a meet-in-the-middle approach to speed up the hardware evaluation of bigger systems with the help of industry standards like IP-XACT. We extend the XML-based files with safety-properties for hardware-components. After generating MARTE hardware models from IP-XACT files, the components are integrated into the whole system-design. The user determines which elements are safety-related depending on the item and safety goal. A developed Eclipse-Plugin helps to evaluate the hardware design of the whole system with safety-properties provided by our approach. Furthermore FMEA or Fault Tree Analysis (FTA) can be performed through failure modes provided by our extended hardware description. An additional design space exploration (DSE) on the system helps to take design-decisions more easily. If a certain ASIL-level of the system is not reached, the tool proposes safety mechanisms with higher diagnostic coverage or elements with lower failure rate. With this approach we demonstrate how the efficiency of verifying safety-critical systems with existing technologies and standards can be increased.

## II. Related Work

Since the design of cyber-physical systems is getting more complex, it needs novel methods to avoid building safety-critical systems from scratch. This chapter describes the previous work done in this area.

How to develop hardware according to functional safety and the ISO 26262 is described in [7] and [8]. In [8] the quantitative hardware architecture of an automotive safety microprocessor is evaluated. The data for the diagnostic coverage of the hardware components comes from a commercial EDA environment and no evidence is given about the correctness. Both approaches neither take modeling approaches into account nor do they recommend safety mechanisms to improve their use-cases. Also, they do not use tools to support their methodology.

The authors of [9] present a methodology to evaluate design choices early in the development process. This is done in an iterative way until a specific safe and cost-effective E/E architecture is derived. This approach was applied in a model-based development process. The models used in this paper are specified as a self-defined metamodel for representing the part of the design artifact. However, this approach neither takes standardized modeling languages into account nor does is use information from standardized hardware IPs. The authors of this paper also recommend using languages like SysML or MARTE.

How to use MARTE in the development process of E/E systems is discussed in several papers [10], [11], [12]. They show how MARTE complies with the model-driven architecture and the CIM/PIM/PSM levels. Although the MARTE language is very capable of modeling E/E systems on different abstraction levels, it lacks in a definition for non-functional properties in safety.

The vendorExtensions [13] of the IP-XACT standard allows vendor-specific definitions of non-functional properties. Properties for timing and power are already partially defined in the standard. The investigation of safety properties in IP-XACT showed that there are no indicators for safety in the standards right now.

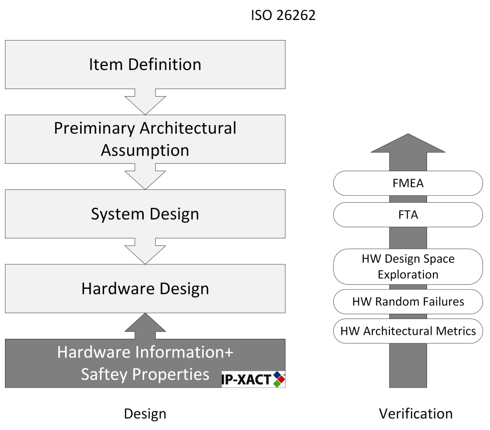


Fig. 1. **Meet-in-the-middle approach:** Design and Verification-speedup through Failure-Modes provided by hardware description

## III. Functional Safety in Hardware

Figure 1 depicts our approach to speed up the verification process for hardware by providing safety-properties in IP-XACT. The left side of this figure shows the traditional development-process of safety-critical hardware (top-down) enhanced through our bottom-up approach. This leads to a speed up in design and verification through providing information about hardware and safety-properties in an early phase of development (meet-in-the-middle). The right side of this figure shows the different analysis-techniques which benefits from our approach. To evaluate the hardware according to Clause 7, Clause 8 and Clause 9 in ISO26262 [2] following equations must be carried out to achieve a given ASIL-Level. This task must be done for each safety goal separately and asks for tools to support this evaluation:

**Single Point Fault Metric (SPFM):** The SPFM reflects the robustness of the item to cope with single-point and residual faults. This can either be handled by design or proper safety mechanisms. The higher the value of SPFM the more robust our applied safety mechanism will be. The following equation is used to determine the SPFM:

$$SPFM = 1 - \frac{\sum_{SafetyRelatedHW}(\lambda_{SPF} + \lambda_{RF,est})}{\sum_{SafetyRelatedHW}\lambda} \quad (1)$$

$$\lambda_{RF,est} = \lambda \times (1 - \frac{K_{DC,RF}}{100}) \quad (2)$$

where Kdc,rf is the diagnostic coverage with respect to residual faults and lambda,rf is the estimated failure rate with respect to residual faults.

**Latent Fault Metric (LFM):** The LFM reflects the robustness of the item to cope with latent faults. This can either be handled by coverage of faults through proper safety mechanisms or by the driver, recognizing that the fault exists before the violation of the safety goal. The higher the value of LFM, the more robust our applied safety mechanism will be. The following equation is used to determine the LFM:

$$LFM = 1 - \frac{\sum_{SafetyRelatedHW}(\lambda_{MPF,L,est})}{\sum_{SafetyRelatedHW}(\lambda - \lambda_{SPF} - \lambda_{RF})} \quad (3)$$

$$\lambda_{MPF,L,est} = \lambda \times (1 - \frac{K_{DC,MPF,L}}{100}) \quad (4)$$

where Kdc,mpf,l is the diagnostic coverage with respect to multiple point latent faults and lambda is the estimated failure rate with respect to multiple point latent faults.

**Probabilistic Metric for Random Hardware Failures (PMHF):** The PMHF evaluates the residual risk of violating a safety goal due to single-point faults, residual faults, and plausible dual-point faults. It defines the quantitative target values for the maximum probability of the violation. The following equation estimates the failure rate for the failure modes of each hardware part that would cause a single-point, residual or dual-point fault (ISO 61508).

$$PMHF = \sum \lambda_{SPF} + \sum \lambda_{RF} + \sum \lambda_{MPF,latent} \quad (5)$$

## IV. Safety Properties of Hardware-IPs

Crawling through datasheets to determine failure rates for hardware components is a cumbersome task. The information of hardware safety properties mostly comes from the vendors themselves. Currently there is no standardized way to provide information about safety in IPs to tool vendors (EDA) or system-integrators. To do this and furthermore achieve inter-operability and reuseability with other tools we propose an extension to a well known format in industry: IP-XACT. IP-XACT is a standard (IEEE 1685) driven by Accellera and its format is used for documenting IPs using meta-data. The data is used for configuring, integrating and verifying IPs in advanced SoC design- and interfacing-tools. The specifications are derived from the requirements of the industry to enable an efficient design of electronic systems. The 1.4 release of the IP-XACT format also includes implementation-models on RTL and TLM level. Furthermore this format supports the data exchange through a common structured data management. Nowadays IP-XACT is used by many different major tool-vendors and there are no other standards compared to it, neither in USA nor Japan.

An IP-XACT model can consist of different files in relation to the IP, like design files, behavioral models, simulation files and results. It also consists of detailed information about the hardware like parameters, ports, memory or configuration. The aim of the standard is to support a component-based design of the hardware and enable the re-use and assembly of HW-components like cores (processors, co-processors, DSPs), peripherals (memories, DMA controllers, timers, UARTs) and buses (simple buses, multi-layer buses, cross bars, network on chip).

Additionally the IP-XACT format also provides vendor extensions to support user-defined features. Vendor specific IP meta-data can be stored in a vendorExtension element. These extensions can be applied to several elements of the IP-XACT format (components, bus-interfaces, registers, etc.). We use the capabilities of IP-XACT to add safety properties to the different elements of the IP. The vendor extensions are composed in a hierarchical manner. The root container can contain one or several vendor extensions. For our purpose we add following properties to the elements:
**Failure-Rate (FR)** - is usually known by the vendor of the component. It is a result of field return and statistical data, where expert judgment can also be considered.
**Failure Modes (FM)** - describes the different modes where a failure can occur. The failure modes depend on the application in which the element is used.
**Safety-Mechanism (SM)** - Implemented mechanism to detect and control faults. It prevents faults from violating the safety goal. If a fault is detected, a safe state is initiated.
**Diagnostic Coverage (DC)** - is the effectiveness of the internal safety mechanism implemented to cover single-point, residual or latent faults.
Since not all safety properties for the evaluation of the

single-points and latent faults are related to the hardware IP, but rather to a given safety-goal, no further properties are described in IP-XACT. This is done in a next step, where the hardware elements are modeled in UML/MARTE.

## V. Safety Properties in Modeling

To address the shortcomings of modeling platforms in UML, MARTE was defined as an adaption from the OMG. MARTE [14], [15] is a domain-specific modeling language intended for model-based design and analysis of real-time and embedded software of cyper-physical systems. MARTE is defined as a profile in UML2 and provides additional mechanisms for modeling real-time systems which are missing in UML. MARTE has the advantage to precisely defining Hardware and Software Resources in form of *HRM* and *SRM* stereotypes. Furthermore it is possible to allocate software applications to hardware resources with the help of the MARTE allocation mechanism. MARTE follows the philosophy of cyber-physical systems to deal with whole systems rather than a set of specialized parts. This is also recommended by the ISO 26262 for the design of safety-critical systems.

Unfortunately the profile does not provide the degree of precision to describe properties for safety. Therefore, we propose an extension-profile for MARTE hardware resource model (HRM) -components for safety-properties to evaluate the architectural metrics. The user is able to choose the safety-related components and their failure modes, depending on their corresponding safety goal and ASIL. One of the key benefits of the profile mechanism is that it ensures tool interoperability if the profile is compatible with the extended base UML concept. This reduces both training and tooling costs [15].

For this purpose we created a self-defined profile named *SafetyProfile* which is a specialization of the existing UML/MARTE concept. Since our stereotype *SafetyProperties* is a subclass of a stereotype that extends the UML element concept, it can only by annotated to elements of the same kind. The extended MARTE-Hardware profile *SafetyProperties* consist of 4 attributes: Safety-Related, Failure-Rate, Failure-Modes and Safety-Mechanisms. The attribute Safety-Related describes whether the component significantly contributes to the violation of the safety-goal. This depends on the outcome of the hazard and risk analysis and therefore cannot be allocated to the hardware description of the component. The FR is annotated as numeric attribute from type integer. The FM includes two further attributes, the name of the failure and the safety-mechanism. The attribute SM is linked to the definition of the safety-mechanisms including the name and corresponding diagnostic coverage. This profile is attached to the hardware components (HRM) of the item to facilitate our calculations for the architectural metrics of our use-case.

## VI. Example Case Study: Battery Management System

To show the novelty and benefits, we apply our methodology to an industrial use case, a Battery Management System (BMS) with regenerative braking features provided by CISC Semiconductors [4]. As more and more vehicles are now powered by Li-Ion-batteries, the challenge for engineers to ensure reliability and fault-tolerance is also greatly increasing. Problems with overheating or even explosions have been frequent in the past. The main cause of these problems was an excessively high energy intake from regenerative braking or harsh environmental conditions. Management systems and mechanisms are thus essential to assure that persons are not put at risk and that no damage is caused. This calls for safety mechanisms to reduce the number of single-point, residual and risks from latent faults in the hardware architecture. The architecture of the item is depicted in Fig. 2 and explained in detail below:

**LiIon-Battery -** consists of 12 cells, connected in series. Each cell has a temperature sensor and connections to measure the voltage. The battery is tagged with *HwPowerSupply* stereotype.
**Battery Monitoring Unit (BMU) -** is the main controller of the battery. It measures different values coming from sensors of the battery-cells. The BMU computes the State-Of-Charge (SOC), State-Of-Health (SOH) and is responsible for cell balancing, cell protection and demand management of the battery. It also controls a hardware switch, which connects the battery to the electric motor. The BMU includes 6 Temperatur-sensors, 12 VoltageSensors, 2 Actuators, 2 ProcessingUnits, RAM, ROM and a BusInterface. The data coming from the BMU is collected and forwarded to the Power Train Controller (PTC) that controls vehicle and wheel speed.
**Sensors -** monitor the battery on temperature and voltage. To maintain redundancy and diversity the voltage is additionally monitored beside the temperature. This provides a more robust design and increases the reliability. The data coming from the sensors are forwarded to the processing units. The sensors are tagged with MARTE *HwSensor* stereotype.
**ProcessingUnit/ASIC -** collect the data from the sensors for threshold- and plausibility-checks. In addition to the processing unit, an ASIC is included to compute the data independent and divers. This in turn reduces the risk of random and systematic faults. Different hardware components lead to different ways in which the hardware reacts and increases the coverage for common cause failures. Sterotypes used are *HwProcessor* and *HwAsic*.
**Memory -** of the micro-controller consists of RAM and ROM. Both components are protected by error detection correction codes (EDC). This safety mechanism helps to detect single-bit and some all-bit failures in words. By checking redundant bits, corruption in words can be determined. Tagged with *HwRam* and *HwRom*.

**Bus-Interface -** a controller area network (CAN) is used to connect the different ECUs with each other in a bus-system. A read back of sent message safety mechanism is used for a more reliable communication. The Bus-interface uses the *HwBus* stereotype.
**Actuators -** cut the connection to the motor, if the temperature of the battery exceeds the maximum threshold. The output is monitored by the processing unit. Tagged with *HwActuator*.

Since it is beyond the scope of this work to examine all safety aspects of the item, we focus here on the battery and the BMU. In order to maintain a coherent picture of the paper we exclude the CAN and PTC from our analysis.

### A. Eclipse-based HW-Architecture Evaluation

Collecting information and metrics for safety purposes is a cumbersome and error-prone task. Engineers and Tool-providers must look into standards like the Siemens SN 29500 or IEC 62380 to obtain the values for hardware-properties like FIT-rate or diagnostic coverage for safety mechanisms. This is sometimes only an estimation of the values and leads to errors in the calculation. We propose that vendors provide their information of IPs and non-functional properties like safety in a standardized format. A format that has already been established in the semiconductor industry like IP-XACT. This saves time and costs in the analysis and the development of new systems. We show this by means of an exemplary

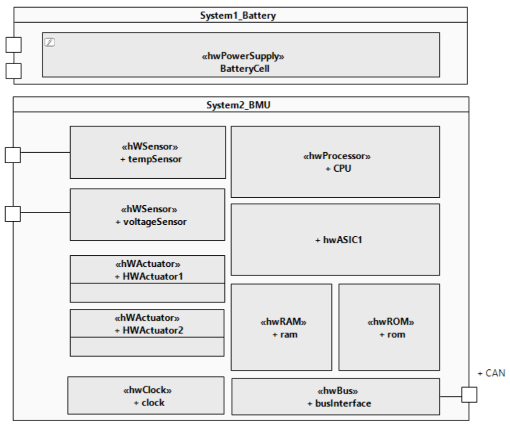


Fig. 2. **Hardware architecture:** modeled in MARTE

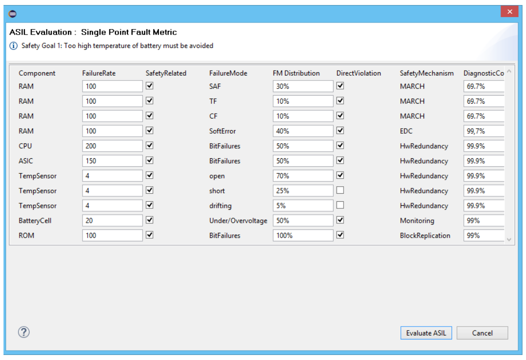
Memory element where we apply two safety-mechanism, error code detection (EDC) and MARCH test. This safety-mechanisms protect our memory against stuck at faults (SAF) and coupling faults (CF). These safety properties are now described with the help of IP-XACT vendorExtensions:

```
<spirit:component>
 <spirit:vendor>-</spirit:vendor>
 <spirit:library>library<spirit:library>
 <spirit:name>component</spirit:name>
 <spirit:version>1.0<spirit:version>
 <spirit:vendorExtensions>
  <cisc:memory>
   <safety:memorySafetyProperties>
    <safety:memorySafetyPropertie>
     <safety:fr>100</safety:fr>
     <safety:fms>
      <safety:fm1>
       <safety:name>SAF</safety:name>
       <safety:sm>SM1</safety:sm>
      </safety:fm1>
      <safety:fm2>
       <safety:name>CF</safety:name>
       <safety:sm>SM1</safety:sm>
      </safety:fm2>
      ...
     </safety:fms>
      <safety:sm>
       <safety:sm1>
        <safety:name>MARCH</safety:name>
        <safety:dc>96</safety:dc>
       </safety:sm1>
       <safety:sm2>
        <safety:name>EDC</safety:name>
        <safety:dc>99</safety:dc>
       </safety:sm1>
      </safety:sm>
      ...
    </safety:memorySafetyPropertie>
   </safety:memorySafetyProperties>
  </cisc:memory>
 </spirit:vendorExtensions>
<spirit:component
```

We developed an Eclipse-plugin to extract information from IP-XACT files into our modeling-environment [16]. This helps us to speed up the design-process, with already fully configured IPs in the IP-XACT standard. The designer is not required to build his system from scratch. He chooses existing hardware-components in different versions and from different vendors. Because the system design is modeled in MARTE, there is no need to say that also pure MARTE HRM-models can be used for the design with additional features from the *SafetyProfile*.

To manage the IPs we use the SHARC IP-library. This database helps us to reuse components for our design, furthermore it provides design space exploration (DSE) to the designers. The SHARC-DSE checks the HW-design if the safety requirements are fulfilled or if further improvements are necessary. The DSE proposes different components with a higher FIT-Rate or safety-mechanisms with better diagnostic coverage for a more reliable design.

In addition to the annotated *SafetyProfile*, a wizard helps us to evaluate our hardware design by the equations mentioned in Chapter III. This wizard is developed as Eclipse-Plugin as depicted in Fig. 3. This plugin provides an overall view of all components of the item to the user. The elements in the wizard are automatically filled by the provided information of the *SafetyProfile*. The designer is able to complete the missing information for the analysis and chooses the safety-related components. The user also defines whether the failure mode has the potential to directly violate the safety goal in absence of a safety mechanism. Regarding this information the wizard evaluates if the hardware design fulfills all requirements for a given safety goal and ASIL. With the help of the SHARC IP-Library the designer is able to choose from different hardware versions and vendors, if the design does not reach a given ASIL. Since the architectural metrics depend on the safety



ASIL Evaluation : Single Point Fault Metric

Safety Goal 1: Too high temperature of battery must be avoided

| Component | FailureRate | SafetyRelated | FailureMode | FM Distribution | DirectViolation | SafetyMechanism | DiagnosticCo |
|---|---|---|---|---|---|---|---|
| RAM | 100 | ✓ | SAF | 30% | ✓ | MARCH | 69.7% |
| RAM | 100 | ✓ | TF | 10% | ✓ | MARCH | 69.7% |
| RAM | 100 | ✓ | CF | 10% | ✓ | MARCH | 69.7% |
| RAM | 100 | ✓ | SoftError | 40% | ✓ | EDC | 99,7% |
| CPU | 200 | ✓ | BitFailures | 50% | ✓ | HwRedundancy | 99.9% |
| ASIC | 150 | ✓ | BitFailures | 50% | ✓ | HwRedundancy | 99.9% |
| TempSensor | 4 | ✓ | open | 70% | ✓ | HwRedundancy | 99.9% |
| TempSensor | 4 | ✓ | short | 25% | ☐ | HwRedundancy | 99.9% |
| TempSensor | 4 | ✓ | drifting | 5% | ☐ | HwRedundancy | 99.9% |
| BatteryCell | 20 | ✓ | Under/Overvoltage | 50% | ✓ | Monitoring | 99% |
| ROM | 100 | ✓ | BitFailures | 100% | ✓ | BlockReplication | 99% |

Fig. 3. **Eclipse-Plugin:** Single Point Fault Metric evaluation of safety goal

goal, each wizard is called by the safety-goal modeled as SysML-requirement. The results of the hardware architectural metrics are annotated to the requirements to archive traceability as recommended by the ISO 26262 standard. How to model safety-requirements in SysML in compliance with ISO26262 was handled in our previous work [17].

## VII. Conclusion

In this paper we presented a novel methodology to evaluate the hardware architecture of safety-critical systems. Through applying the methodology on an industrial use-case of a Battery Management System (BMU), we showed how we increase the productivity and speed up the verification process. To promote the development of safety-critical systems, components are required with already implemented safety-mechanisms. However, these components must be verified on system-level. With the use of IP-XACT, already existing hardware is partially integrated into the system-design. Additional properties for safety help to speed up the evaluation. Furthermore, our tool-aided method helps the designer to take design decisions for hardware-parts very early in the design process. Experiments showed that our methodology helps to verify these systems much faster. Development costs are drastically reduced through reusability. Furthermore, the designer's effort decrease, if components with better characteristics are proposed by our methodology. The whole process is included in an Eclipse-based tool named SHARC [18]. In a next step, the efficiency of the safety mechanisms will be verified by fault-injection tests in simulation. MARTE-models will be linked to implementation models in different languages (SystemC(-AMS), Matlab, VHDL) with the tool SHARC for co-simulation.

## Acknowledgment

The approach presented above is experimented in the OpenES CATRENE Project: CA703 - 2013 research program supported by the FFG (Austrian Research Promotion Agency), project-number 843380 in tight cooperation with CISC Semiconductor.

## References

[1] Etas, "Electronic Control Unit ( ECU ) - Webinar Basics of Automotive ECU ETAS Embedded Systems Consulting Embedded Software , AUTOSAR and Safety Consulting Chandrashekara N ( ETAS / ESC ) Lead Consultant Embedded Systems," pp. 1–30, 2014.
[2] ISO, *ISO 26262: Road vehicles - Functional safety - Part 1-10*, 2011.
[3] J. Vigen and O. No, "IEC TR 62380 - Reliability data handbook - Universal model for reliability prediction of electronic components," vol. 2004, 2005.
[4] J. Medina, "The UML Profile for MARTE: modelling predictable real-time systems with UML," Tech. Rep. [Online]. Available: http://www.artist-embedded.org/docs/Events/2011/ModelsforSA/01-MARTE-SAM-Julio/Medina.pdf
[5] Catrene, "OpenES CATRENE Project: CA703 - 2013," 2013. [Online]. Available: http://www.ecsi.org/openes
[6] SPIRIT, "IEEE SA - 1685-2009 - IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows." [Online]. Available: http://standards.ieee.org/findstds/standard/1685-2009.html
[7] S.-H. Jeon, J.-H. Cho, Y. Jung, S. Park, and T.-M. Han, "Automotive hardware development according to ISO 26262," *13th International Conference on Advanced Communication Technology (ICACT2011)*, pp. 588–592, 2011.
[8] Y.-c. Chang, L.-r. Huang, H.-c. Liu, C.-j. Yang, and C.-t. Chiu, "Assessing Automotive Functional Safety Microprocessor with ISO 26262 Hardware Requirements," pp. 3–6, 2014.
[9] V. Rupanov, C. Buckl, L. Fiege, M. Armbruster, A. Knoll, and G. Spiegelberg, "Early safety evaluation of design decisions in E/E architecture according to ISO 26262," *Proceedings of the 3rd international ACM SIGSOFT symposium on Architecting Critical Systems - ISARCS '12*, p. 1, 2012.
[10] L. G. Murillo, M. Mura, and M. Prevostini, "MDE Support for HW/SW Codesign: A UML-based Design Flow," *Advances in Design Methods from Modeling Languages for Embedded Systems and SoCs*, vol. 63, pp. 197–212, 2010.
[11] J. Vidal, F. D. Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet, "A co-design approach for embedded system modeling and code generation with UML and MARTE," *2009 Design, Automation & Test in Europe Conference & Exhibition*, 2009.
[12] A. Koudri and D. Aulagnier, "Using marte in a co-design methodology," *UML Workshop at Date'08*, 2008.
[13] A. S. Initiative, "Accellera Vendor Extensions for IEEE 1685-2009," Tech. Rep. September, 2013.
[14] "The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems — www.omgwiki.org/marte," 2013. [Online]. Available: http://www.omgmarte.org/
[15] B. Selić and S. Gérard, *Modeling and analysis of real-time and embedded systems with UML and MARTE*, 2014.
[16] C. André, F. Mallet, A. M. Khan, and R. de Simone, "Modeling SPIRIT IP-XACT with UML MARTE," *Proc. DATE Workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile*, 2008.
[17] R. Weissnegger, C. Kreiner, M. Pistauer, R. Kay, and C. Steger, "A Novel Design Method for Automotive Safety-Critical Systems based on UML/MARTE," in *Proceedings of the 2015 Forum on specification & Design Languages*, 2015, pp. 177–184.
[18] R. Weissnegger, M. Pistauer, and C. Steger, "Program & Book of Abstracts," in *The 10th International Conference on Scientific Computing in Electrical Engineering SCEE 2014,Program & Book of Abstracts*, Wuppertal, 2014, pp. 57–58.

Available online at www.sciencedirect.com
SciVerse ScienceDirect
Procedia Computer Science 00 (2016) 000–000
Procedia Computer Science
www.elsevier.com/locate/procedia

The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016)

# Simulation-based Verification of Automotive Safety-Critical Systems based on EAST-ADL

Ralph Weissnegger[a,b,*], Markus Schuss[a], Christian Kreiner[a], Markus Pistauer[b], Kay Römer[a], Christian Steger[a]

[a]*Institute for Technical Informatics, Graz University of Technology (TU Graz), Austria*
[b]*CISC Semiconductor GmbH, Klagenfurt, Austria*

**Abstract**

The increasing amount of assistance features in today's vehicles to ensure safe and reliable operation, imply increasingly complex systems. New challenges are arising due to highly heterogeneous and distributed systems which interact with and have an impact on the physical world, so called cyber-physical systems. Since millions of test kilometers must be driven to ensure a reliable system, simulation-based verification is becoming more important to reduce costs and time-to-market. This situation prompts the urgent demand for new techniques to simulate the behavior in early development phases by reusing verified system components. Best combined within a model-based approach that both unites different stakeholders and helps non-specialists to understand problems in the design. In this paper, we present a novel method for simulation-based verification of automotive UML/EAST-ADL design models. To demonstrate its benefits, our methodology is applied in an industrial use case of a battery management system.

*Keywords:* UML; EAST-ADL; automotive; ISO26262; simulation; verification; SystemC;

## 1. Introduction

Today's cars consist of highly complex electric/electronic (E/E) systems with sensors and actuators networking with each other. In fact a car is now more or less a smartphone on wheels. It can be observed that there is a shift towards fully E/E cars, since electric cars are getting more popular. The sensing and controlling of these systems is the work of the highly distributed electrical control units (ECU) and it is no surprise that more than 200 of these micro-controllers are currently integrated in a modern electric vehicle[1]. Since the electrification in the automotive domain continuous, new challenges in the development process are arising. This is especially the case where multiple stakeholders including specialists for hardware, software and system design have to work together with safety engineers to ensure a reliable and safe system. A model-based approach helps non safety-specialist to also understand problems in the design and development of safety-critical systems. One modeling languages which has established

* Corresponding author: Ralph Weissnegger
*E-mail address:* ralph.weissnegger@tugraz.at

itself in the automotive domain is EAST-ADL[2]. It allows the detailed design of automotive E/E systems on different levels of abstraction. The last two layers conform with the AUTOSAR standard. Furthermore, they are in line with the development process of safety-critical systems according to ISO26262[3] and allow the evaluation and formal verification of design models. Since millions of test kilometer must be driven to ensure a reliable system, simulation is becoming more and more important[4], because it is no longer possible to cover the costs of physical tests. A drawback in today's development process is that simulation tools are often detached from the design tools and require cumbersome imports and exports of files between the environments. It is important that simulation tools are tightly and seamlessly integrated into the design and development process[5], meeting the requirements of ISO26262. Furthermore, approaches are needed that allow a high traceability to requirements and make it possible to derive requirements from simulation-results. This must be done as early as possible and on different abstraction levels.

In this work, we present a model-based simulation framework for the verification of E/E systems in the automotive domain. We link quickly executable simulation models, implemented in SystemC (-TLM) and SystemC-AMS, with EAST-ADL design models. The level of granularity of the models can be easily switched depending on the complexity. Using these reusable components, we achieve an early behavior simulation of the whole system. The result is a tool-aided methodology built as an Eclipse plugin in Papyrus[6], which makes it easy to verify the behavior of automotive safety-critical systems.

## 2. Related Work

An approach for generating simulation models from EAST-ADL architecture models was presented in[7]. In this work, several architecture levels of EAST-ADL have been mapped to abstraction levels of SystemC-TLM. The architecture of an automotive use case was presented on analysis and design level. For the expression of the behavior, the authors used SystemC code and state machines. This approach works very well for the digital domain, but lacks proper definition needed for analog and mixed-signal components. Through the use of code generators, it is possible to achieve synthesis of very detailed EAST-ADL models. It would also benefit of analyze and verification mechanisms for their simulations.

The authors of[8] presented three different analysis techniques for architectural models described in EAST-ADL, to guarantee the quality in the context of ISO26262. One of the proposed techniques is the simulation of EAST-ADL functions in Simulink. The behavior of each function was linked to FMU or Simulink models to facilitate the simulation. The authors also described mapping rules for the EAST-ADL to Simulink transformation (one-to-one mapping). The results of the simulation have been traced back to the requirements. This approach was applied to an industrial use case of a brake-by-wire system on Design Level. However, in contrast to our approach, they use proprietary simulation engines with high license costs and external tools which are not integrated into the design and development flow.

The authors of[9] demonstrated how to use MARTE for hardware design and simulation. They introduced a step-by-step methodology for hardware modeling with Hardware Resource Models (HRM) stereotypes. The platform models are refined until the final platform class is reached. In a later step, these models are used to generate code with the help of a Java plugin. A tool called Simics was used to facilitate the simulation. Instead of using the whole MARTE spectrum for simulation, this approach only uses HRM models for code generation of very detailed platforms instead of system level design.

## 3. Model-based System Design

Model-based design plays an ever increasing role in today's development to deal with complex systems. Organization of specialized people in projects of a certain size requires a lot of effort. Therefore, it is becoming increasingly important that stakeholders from different domains, e.g. hardware, software, safety or even security can efficiently work together. Particularly in the evaluation of safety-critical systems, safety specialists need a entire view of the system, that includes all domains of the system. Best combined in a tool where even entire processes like the ISO26262 can be addressed.

One modeling-language which has established itself in the automotive domain is EAST-ADL. It allows the capturing of detailed automotive electric and electronic systems on five layers of abstraction, each with a clear separation of concerns: *Vehicle* , *Analysis*, *Design* , *Implementation* and *Operational Level*. Besides structural aspects, this modeling language allows the expression of behavior, requirements, verification and validation. The highest level is the *Vehicle Level*, that describes electronic features to allow integration of product variability. The *Analysis Level* includes the Functional Analysis Architecture (FAA), which allows an abstract functional representation of the architecture (what the system shall do), in relation with the features from the *Vehicle Level*. The *Design Level* allows the decomposition of models in the FAA to Functional Design Architecture (FDA) models and Hardware Design Architecture (HDA) models. Within these models the functional representation of the architecture can be allocated onto the hardware platforms. The applications are represented by *DesignFunctionTypes* with annotated behavior and configurations. The hardware components are modeled by *Sensors*, *Nodes* (ECUs), *Actuators* and *HardwarePortConnector* (Buses) and more. They are interconnected by *IOHardwarePins* or *CommunicationHardwarePin* and wired by *HardwareConnectors*. The last two layers (Implementation, Operational) are the realization of the implementation in AUTOSAR. Therefore, the models on these levels are compliant with the AUTOSAR specifications. The behavior of components on all these levels are not explicitly addressed in EAST-ADL. It can be either expressed by behavioral diagrams (state machines, activity diagram) or externally in tools like Matlab.

EAST-ADL as automotive modeling language also addresses parts of the functional safety standard ISO2626, which was one of the outcomes of international projects like ATESST[10] and MEANAD[11]. This enables the language for safety-analysis like Fault Tree (FTA) or Failure Mode and Effect Analysis (FMEA), but also for defining safety-requirements and to achieving high traceability to models and behavioral diagrams. Furthermore this language provides means to describe validation and verification activities by *VVCases*. Since EAST-ADL is included in an Eclipse UML2 Editor called Papyrus[6], it makes it easy to design complex systems without licensing costs.

In the next section, we present the simulation core and how to execute EAST-ADL models with behavioral languages such as SystemC and SystemC-AMS.

## 4. Executable Models

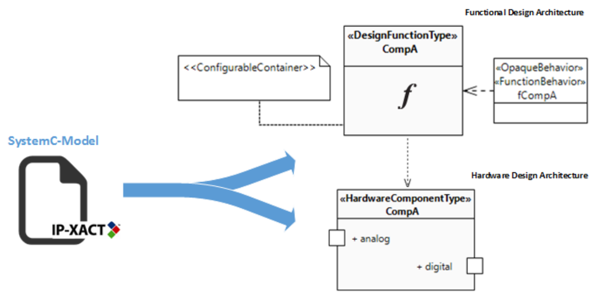
SystemC is defined by Accelera, a standards organization in the area of electronic design automation (EDA). It is an open standard modeling language and has been also approved by the IEEE standards association. SystemC is defined by several levels of abstraction. On the transaction level modeling level (TLM), a very high level simulation, the focus lies on communication and functionality. This serves as a golden reference for lower level hardware models (RTL). The RTL level included very detailed models where the components are connected through signals with pins. SystemC enables the design teams to have a fundamental understanding of the system at an early stage of the design process. Due to its high flexibility, it enables the representation of a complete system.

SystemC tries to bridge the gap between hardware description language (HDL) and object-oriented language (OOP). While SystemC is commonly used in the context of a system on chip or to model several components in a system, it is usually limited to the digital domain. In order to address complex systems with digital and analog parts, an extension was introduced called SystemC-AMS. This extension enables the simulation of continuous time, discrete time and discrete event behavior of analog/mixed-signals simultaneously. Nevertheless, SystemC lacks a visual representation for interacting with different stakeholders and their requirements. Because of its C++ background, SystemC is object-oriented and has a lot of similarities with UML and EAST-ADL, that supports the part,port and connector principle. This has also been acknowledged in publications such as[7] and makes the linking with EAST-ADL models intuitive. Because of this and its wide acceptance in the industry as well as availability, SystemC was chosen as the primary simulation language in our approach. Our methodology bridges the gap between model-driven design in EAST-ADL and executable models in SystemC.

The executable models used in our approach are models on different abstraction levels and in different versions. The generic models have the potential to be used in various domains and support reusability. The detailed models are refinements of the generic models and are to be used in special domains. Dependent on the domain and abstraction level, the models are built in SystemC(-TLM) and SystemC(-AMS).

## 5. System Libary

To avoid the design and simulation of larger systems from scratch and to achieve reusability, our methodology provides a SystemComponentLibrary (SCL). This library includes all major components for the simulation of systems in the automotive domain. Furthermore, it includes components in different versions and on different abstraction levels. A component in our library consists of two models, a structural model that describes the hardware structure and a functional model that describes the behavior. Both models have the same name but are tagged with a different type. The structural model is described by the EAST-ADL stereotype *HardwareComponentType* and owns the digital and analog ports of the hardware, tagged by *IOHardwarePin*. This mechanism also makes it possible to detach the digital components from the analog components. The behavior of the component is tagged by *DesignFunctionType*. This model owns the *FunctionBehavior*, which contains the kind and path to the behavioral description (SystemC model). In addition, the *DesignFunctionType* is tagged with *ConfigurableContainer*, which defines parameter and values of the component. These parameters may vary depending on the use case and verification methodology. To illustrate the togetherness, the functional model is allocated on the structural model with *FunctionAllocation*. Figure 1(a) depicts this approach. Since the creation of models for the SCL is a cumbersome task, a Text-to-Model converter helps to generate EAST-ADL models from SystemC code or even IP-XACT files. This converter generates the structural and behavioral models, with ports, parameters and allocation to the files. The mapping from SystemC/-AMS to EAST-ADL models is described in Table 1(b).



(a)

| EAST-ADL | | SystemC | |
|---|---|---|---|
| Stereotype/ Attribute | Type | Class/ Attribute | DataType |
| HWComponentType | | sc_module | |
| IOHardwarePin | | | |
| type/direction | analog in | sca_in | |
| | analog out | sca_out | |
| | digital in | sc_in | |
| | digital out | sc_out | |
| HardwarePortConnector | | sc_module | |
| busSpeed | float | txrate | double |
| busType | | bustype | |
| DesignFunctionType | | sc_module | |
| FunctionBehavior | | class.dll | |
| ConfigurableContainer | | | |
| Type | | type | |
| Attribute | | attribute | |

(b)

Fig. 1. (a) SystemComponentLibrary (SCL) text-to-model converter (b) mapping of EAST-ADL to SystemC models

To increase the reusability and provide good support for developers, the SystemComponentLibrary is built as an Eclipse plugin. New models can be generated and added to the library. Updates for components can be easily checked by updating the library from the server. This helps to support design teams by adding new components, and keeps the library consistent.

## 6. Methodology

Our methodology for the execution of EAST-ADL models is composed of four phases as depicted in Fig.2: Design-Phase, Build-Phase, Connect-Phase and Run-Phase.

### 6.1. Design Phase

The first part in our methodology is the system design. The designer creates an EAST-ADL design level class where the top-level is modeled. This class describes the overall architecture of the system. It is composed of the functional and hardware architecture model instances from the SystemComponentLibrary. These sub-systems are connected together by ports according to their specification. Due to the EAST-ADL port capability, these ports are checked in advanced by the framework in the correct direction of the dataflow or type (digital, analog). To trace the dataflow, *Scope* models are added to the functional design and connected to functional ports. These *Scopes* collect the monitored data and stores them in trace files to compare the results from different tests and testbenches in a latter step. Our framework also provides a mechanism to encapsulate existing components and to raise the abstraction level of the whole system. Smaller systems that model the interior design of the class can be merged to a more simplified model. This helps the designer to have a better view of the system, without having too many details in the models on system level. This mechanism allows us to abstract the complexity of components. The whole eVehicle system, also referred to in our case as Design Under Test (DUT), is provided with several connectors. These signals required for testing and debugging are brought out to the ports of the top class itself. This has the advantage of connecting testbenches to the DUT for testing various scenarios of the electric car and also for monitoring performance. The outcome of the design step is a netlist that also serves as configuration and parametrization for the simulation. It is the starting point for the build phase.

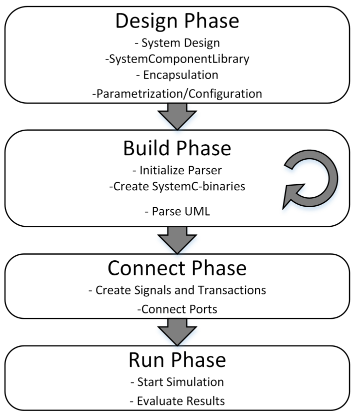


Fig. 2. Methodology for executable SystemC models from EAST-ADL design

### 6.2. *Build Phase*

The heart of our Build-Phase is the self-defined parser-methodology. The purpose of the parser is to translate a UML model defined in one or more files to a single SystemC system. This is done at run-time and does not require compilation of the resulting model. When the parser is initialized it requires the name of the top-class of the model in the diagram as well as the name of the UML file that contains the model. Starting from the root node of the UML model, each child of a node is parsed and returned. As single systems can be composed of more detailed sub-systems, we had to define a loop to find all properties and ports of each root note. Each node found in the UML file is treated as the new root node and each sub-system is created before moving on to the Connect-Phase. Each system may contain any number of sub-systems, therefore this step is done in several iterations till all properties of the root node are found. In order to keep the framework extensible a DLL-based plugin system is used. All the information is stored in a ConfigStore map.

### 6.3. *Connect Phase*

In this phase, the connector objects are created to link the different instances in the build phase. Depending on their nature, the connector objects can be signals or transactions. It is important to notice, however, that UML allows multiple 1:1 connections per port, SystemC merely allows a port to be bound once but a signal may connect any number of ports (basically 1:n as only one driver is allowed per signal). As a means of handling these issues, both ends of each connector are tagged using an ID. Instead of creating new signals for connecting to a used port, the old signal is reused.

### 6.4. Run Phase

After all nodes, ports and properties of the UML file have been found by the parser, and the SystemC instances have been created and connected, the simulation is started. The results of the monitored signals via *Scopes* are saved as trace files. These files contain all the relevant information required for the verification of the model or to evaluate the behavior of the model for different parameters and/or implementations for the system. Besides this, a logic can also be added to the system to react to certain events such as stopping the simulation in case of a signal, violating the given constraints, or the system running out of energy (for a battery powered system). An implemented dialogue is also used to configure the settings for the simulation such as duration or timestep (resolution).

Using the Toem Impulse plugin[12] for Eclipse, the results are presented in a graphical form. The results can be displayed in the desired manner, dependent on the nature of the simulation, e.g. analog interpolation for real values and numeric representation for digital signals in a hierarchy that allows for easy interpretations. The results may be verified against the known or expected behavior of the (physical) system modeled. If the system behaves as expected, it can be used for further analysis or verification (e.g as a golden reference model or synthesis-able).

## 7. Example Case Study: Electric Vehicle Simulation

We have applied our methodology to an industrial use case, an electric vehicle (eVehicle) system provided by CISC Semiconductor, to more fully illustrate its innovative capabilities and benefits. As more and more vehicles are now powered by Li-ion batteries, the challenge for engineers to ensure reliability and fault tolerance is also greatly increasing. It is crucial that the battery management systems (BMS) measure voltage, temperature and current of the battery very precisely to ensure safe operating conditions. This information must be forwarded to a system wide controller network to ensure a reliable and fully utilized system. Problems with overheating or even explosions have been frequent in the past. The main cause of these problems was an excessively high energy intake from regenerative braking or harsh environmental conditions. Management systems and mechanisms are thus essential to assure that persons are not put at risk and that no damage is caused. The overall system model of the eVehicle is depicted in Fig.3. It is composed of the *battery*, *controller*, *inverter*, *dc-motor*, *power train controller (PTC)* and the *battery management unit (BMU)*. The *driver* provides the desired speed for the eVehicle. This can be set according to standardized maneuvers such as the New European Drive Cycle (NEDC). The *controller* is a model for a PI state-space controller and maintains a constant speed based on the information about the state variables, motor armature current and motor-speed. The *inverter* model implements an inverter function for a PM-DC motor driving stage. It compares the actual battery voltage and the requested controller voltage to maintain the PM-DC motor terminal voltage. The *battery* model simulates the behavior of a Li-ion battery pack composed of a defined set of single cell Li-ion batteries. The appropriate number of single cells is connected in parallel and series to obtain the necessary capacity and terminal voltage. The battery pack's terminal voltage is calculated based on the defined parameter and the battery current. A BMU is connected to the battery to measure voltage, current and temperature of the cells/modules. The BMU computes the SOC, State-Of-Health (SOH) and is responsible for cell balancing, cell protection and demand management of the battery. These computed values are then processed via a CAN controller as digital values to the power train controller. In addition, the external load environmental conditions like temperature can be changed during the simulation.

The *Design Level* of the Design Under Test (DUT) contains the *Functional*, *Analog* and *Digital Design Architecture*. On the functional level, the components are tagged with *DesignFunctionType* which describes the behavior of the hardware. To provide stimuli and monitoring function, ports are brought outside of the DUT to connect various verification components like driver, monitor or scoreboard. Every *DesignFunctionType* on this functional level is allocated on its hardware counterpart on the hardware/digital architecture level as described in Section 5. The hardware components are tagged with stereotypes from the EAST profile such as *Node*, *Sensor*, *Actuator* and *ElectricComponent*. Depending on the nature of the port, the *IOHardwarePinKind* attribute is set to analog or digital. Each component is configured and parametrized through *ConfigurableContainer*.

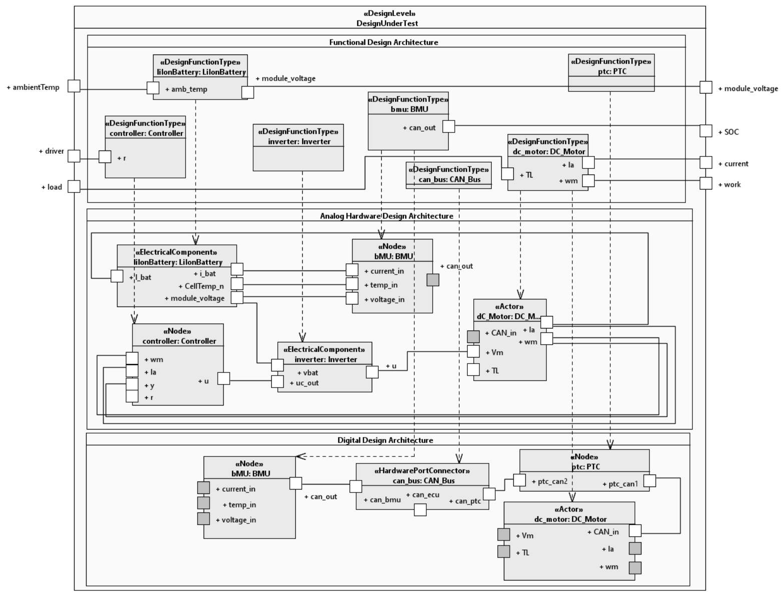

«DesignLevel»
DesignUnderTest
Functional Design Architecture
«DesignFunctionType» lilonBattery: LilonBattery
+ amb_temp
+ module_voltage
«DesignFunctionType» ptc: PTC
«DesignFunctionType» bmu: BMU
+ can_out
«DesignFunctionType» controller: Controller
+ r
«DesignFunctionType» inverter: Inverter
«DesignFunctionType» dc_motor: DC_Motor
+ Ia
+ TL
+ wm
«DesignFunctionType» can_bus: CAN_Bus
+ ambientTemp
+ driver
+ load
+ module_voltage
+ SOC
+ current
+ work
Analog Hardware Design Architecture
«ElectricalComponent» lilonBattery: LilonBattery
+ i_bat
+ CellTemp_n
+ module_voltage
«Node» bMU: BMU
+ current_in
+ temp_in
+ voltage_in
+ can_out
«Actor» dC_Motor: DC_M...
+ CAN_in
+ Ia
+ wm
+ Vm
+ TL
«Node» controller: Controller
+ wm
+ Ia
+ y
+ r
+ u
«ElectricalComponent» inverter: Inverter
+ vbat
+ uc_out
+ u
Digital Design Architecture
«Node» bMU: BMU
+ current_in
+ temp_in
+ voltage_in
+ can_out
«HardwarePortConnector» can_bus: CAN_Bus
+ can_ecu
+ can_bmu
+ can_ptc
«Node» ptc: PTC
+ ptc_can2
+ ptc_can1
«Actor» dc_motor: DC_Motor
+ Vm
+ TL
+ CAN_in
+ Ia
+ wm

Fig. 3. EAST-ADL Design Level of the eVehicle simulation

## 8. Results

The results of our simulation-based verification within EAST-ADL are depicted in Fig.4. It shows the analog and digital signals which are monitored by Universal Verification Methodology (UVM) components. The DUT was stimulated by a driver with different driving scenarios. The exact simulation was also built as a Simulink model to compare our results with a golden reference model. The output of the Simulink run is referenced as *golden_ref* signal. Signal *deviation* shows the difference between signal *work* of both simulations engines. Only when it comes to a step in signal *load* there is a peak in the deviation of about 0.5 percent. This occurs because the SystemC simulation kernel requires an additional delay at this step, where Simulink, with its centralized timing solver, does not. This produces a short shift between both signals but ends up, after a timestep, with the same results. The average error between the SystemC and Simulink signal is 0.0081 percent. Both simulations have the same accuracy with a fixed timestep of $1 \times 10^{-3}$s.

## 9. Conclusions

In this paper, we presented a model-based simulation framework for verification of electric/electronic systems. We used the capabilities of EAST-ADL for a model-based design, to simulate analog and digital components in the automotive domain. With the help of our tool-aided methodology, we achieve simulation of systems seamlessly integrated into the design flow of ISO26262. Especially regarding functional safety, a model-based approach helps safety engineers to have an augmented view, to understand problems in the design and development of safety-critical systems. Through EAST-ADL models, the behavioral models in SystemC can be configured or even reconfigured for different testcases. A model library was introduced that helps to speed up the design and development process and raises reusability. With the help of UVM-like components, the whole system can be verified by methods like constraint random verification. Due to the Eclipse plugin mechanism, every Papyrus editor is now capable of executing their EAST-ADL models by installing our plugin. This tool called SHARC (Simulation and verification of HierARChical

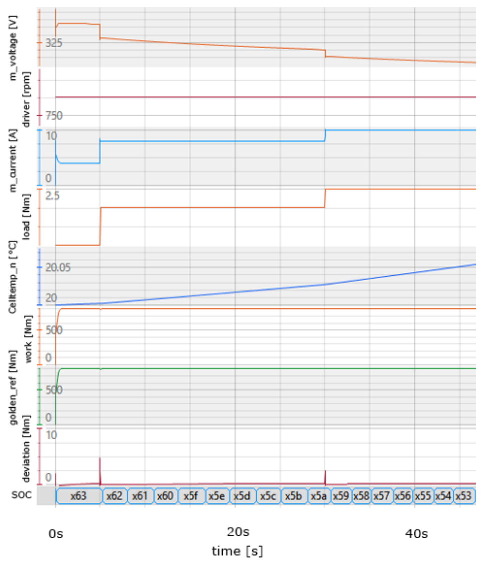


Fig. 4. Output-trace of the eVehicle simulation, the signals are compared to a golden reference

(embedded) systems) will be published for download and also used for educational purpose. To show the benefit of our framework, the tool-aided methodology was applied to an automotive use case of a battery management system. Another focus will be the resource handling of behavioral models for multi- and many-core applications. Because of its small memory footprint and fast execution time, this simulation environment will be used additionally for parameter variation in cloud-based environments.

## Acknowledgments

The approach presented above is an experiment undertaken in the framework of OpenES CATRENE Project: CA703 - 2013 research program supported by the FFG (Austrian Research Promotion Agency), project-number 843380 in tight cooperation with CISC Semiconductor.

## References

1. ETAS Embedded Systems Consulting: Electronic Control Unit ( ECU ) - Webinar Basics of Automotive ECU 2014;:1–30.
2. EAST-ADL Association. 2014. URL: `http://www.east-adl.info/`.
3. ISO 26262, . Road vehicles-functional safety-Part 5: Product development at the hardware level 2011;.
4. Maurer, M., Gerdes, J.C., Lenz, B., Winner, H.. Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte. Springer Open; Springer Berlin Heidelberg; 2015. ISBN 9783662458549.
5. Weissnegger, R., Kreiner, C., Pistauer, M., Römer, K., Steger, C.. A Novel Design Method for Automotive Safety-Critical Systems based on UML/MARTE. In: *Proceedings of the 2015 Forum on specification & Design Languages*. Barcelona, Spain; 2015:177–184.
6. Eclipse, . Papyrus. 2015. URL: `https://www.eclipse.org/papyrus/`.
7. Weiss, G., Zeller, M., Eilers, D., Knorr, R.. Approach for Iterative Validation of Automotive Embedded Systems. *Models 2010 ACES-MB Workshop Proceedings* 2010;:69–83.
8. Marinescu, R., Kaijser, H., Mikucionis, M., David, A., Seceleanu, C., Henrik, L.. Analyzing Idustrial Architectural Models by Simulation and Model-Checking. *Formal Techniques for Safety-Critical Systems* 2014;419:189–205. doi:10.1007/978-3-319-05416-2.
9. Taha, S., Radermacher, A., Gérard, S.. An Entirely Model-Based Framework for Hardware Design and Simulation. *International Federation for Information Processing (IFIP)* 2010;:31–42.
10. ATESST. 2010. URL: `http://www.atesst.org`.
11. maenad.eu. 2014. URL: `http://www.maenad.eu/`.
12. Toem Impulse. 2016. URL: `http://toem.de/index.php/projects/impulse`.

# Parallel Executions in the Cloud

Ralph WEISSNEGGER, Markus SCHUß, Martin SCHACHNER, Kay RÖMER
and Christian STEGER, Graz University of Technology, Austria
Markus PISTAUER, CISC Semiconductor, Klagenfurt, Austria

Simulation-based verification is one of the most essential verification-methods in today's development of embedded systems. To ensure a reliable system, not only functional but also non-functional properties like timing, power, thermal or safety must be taken into account. These properties must also be verified concerning standards like the ISO26262 for functional safety in the automotive domain. Since millions of test kilometre have to be driven to ensure a reliable system, simulation is becoming more and more important, since the costs for physical tests cannot be handled anymore. One verification methods which has been established in the field of embedded systems is the Universal Verification Methodology (UVM). However, this method has the drawback of consuming too much time when executing thousands of simulations with varying parameters in a sequential manner. Therefore, it needs new methodologies to speed-up this verification process through parallelization. In this paper, we present a novel approach which extends the layered pattern of UVM with message patterns used in today's cloud computing. This helps design and verification engineers in the embedded system domain to gain their simulation results much faster. The result of this work is a complete verification environment, which uses the full potential of our newly defined verification pattern.

CCS Concepts: **•Computing methodologies → Modeling and simulation; •Computer systems organization → Embedded systems; Reliability;**

Additional Key Words and Phrases: Message Patterns, Verification, UVM Functional Safety, Embedded Systems, UML

## 1. INTRODUCTION

Verification is one of the most essential concepts in the overall development process. Since many (in)famous examples have shown that redesigns or, even worse, call-backs cause high costs, system designers have investigated many efforts to avoid faulty designs. This of course belongs also to the automotive domain where failure or malfunction of the system can lead to severe damage to people and environment or can even lead to death. This has been also recognized by several safety standard such as the ISO26262 for electric/electronic systems in cars, where these issues are covered by a proper design, requirements and verification flow, so-called safety lifecycle.

All in all it is essential to start system verification in early phases and reuse them throughout the whole design phase to keep track with the system requirements. Furthermore, to reuse these testbenches at the end of the safety lifecycle, for verification of system integration and validation tests. Best combined within an approach to automatically derive testbenches from the definition of safety goals and requirements as presented in [Weissnegger et al. 2016b]. This approach helps designers to keep a high traceability from requirements to design, tests and their final results.

*EuroPLoP '16*, July 06-10, 2016, Kaufbeuren, Germany
© 2016 ACM. ISBN 978-1-4503-4074-8/16/07... $15.00
DOI: http://dx.doi.org/10.1145/3011784.3011806

the aid to use concepts from the Universal Verification Methodology (UVM). UVMs [uvm 2016] core functionality makes it possible to generate test environments that allow a prediction which parts of the design can be verified from given test scenarios. This approach is available with the usage of the so-called constrained random verification stimulus (CRV) principle. UVM becomes even more powerful in combination SystemC, to verify the system from an early functional specification down to hardware development on register transfer level (RTL).

One drawback of CRV verification within UVM is the huge amount of simulation runs. Many parameters and stimuli data have to be tuned and varied to make a statement about the reliability of the system. This follows in thousands of simulation runs and is very time consuming. Therefore it needs novel approaches to use UVM for parallel execution of simulation runs.

In this work, we use the standard UVM layered architecture pattern in combination with the enterprise integration pattern to facilitate our thousands of simulation-runs in the cloud such as Amazon AWS [ama 2016] or Microsoft Azure [azu 2016]. The result is a complete framework which uses the layered architecture pattern of UVM with the benefits of message patterns.

## 2. UVM

As there is a trend to more structured, modular, configurable and reusable verification methods, UVM was defined to tackle these challenges. UVM is an Accellera System Initiative approved standard methodology for verification and provides a UVM Class Library with all the building blocks, which are needed to quickly develop well-constructed and reusable verification components and environments in SystemVerilog. Furthermore it provides a well defined layered architecture as depicted in Fig. 1, to clearly distinguish between the various abstraction levels. UVM is usually developed for SystemVer-

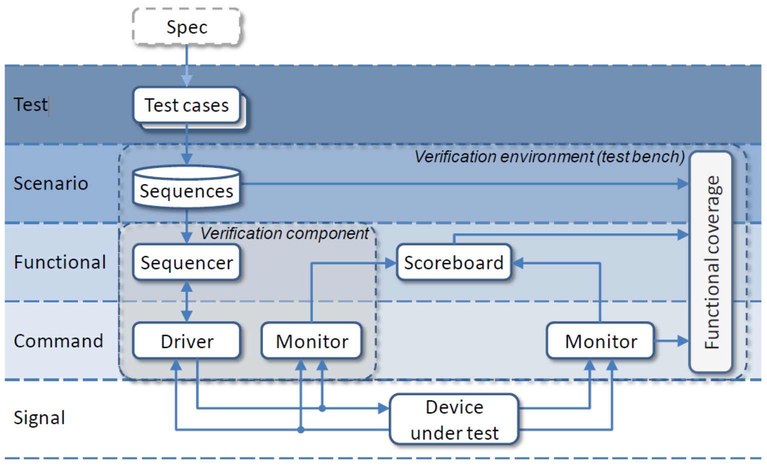


Fig. 1. Traditional UVM layered architecture for SystemC [Barnasconi and Curie 2014]

ilog which allows simulation and verification of digital hardware on RTL level. Since the trend in the embedded system design as well in the verification of safety-critical systems is going to a higher abstraction level, various approaches tried to connect UVM with SystemC [acc 2016]. SystemC allows the modeling and simulation of hardware- and software-components in a single language and allows the simulation depending on the use case. Therefore, we will build on the approach of [Barnasconi and Curie 2014] which defines a methodology to use UVM for SystemC.

The UVM-SystemC architecture consists of 5 Layers, which communicate through standard interfaces with each other. This approach allows to clearly distinguish between test case definition from test-scenarios and also the actual verification environment (testbench) on which the sequences are executed. The highest layer (Test layer) defines the current tests, which consists of the selection of the testbench and defines the test sequences. UVM Test is the top-level component and has three main functions. It instantiates the verification environment, configures the environment (via factory pattern) and applies stimulus by invoking sequences. The scenario layer contains the sequences, which contain the behavior for generating stimulus. On this layer the actual test sequence(s) are generated. The next layer (Functional) contains the Sequencer, which is responsible for the right arbitration and ordering of sequences and their transactions. It controls the transaction flow from multiple stimulus sequences. Another part of this layer is the Scoreboard, which main functionality is to check the behavior of the design under test (DuT). It compares the expected output (golden reference model) with the actual output. Furthermore it is necessary for self-checking mechanism, the collection of functional coverage and pass/fail reports. The Command layer includes the driver, monitors and checkers, which are implemented on physical-level. The driver receives the individual sequence-transaction from the Sequencer and applies (drives) it to the DuT (from TLM to RTL level). The Monitor samples the data coming from the DuT and is responsible for coverage collection, checking, logging or recording. On the Signal level, the lowest layer of this architecture, the testbench is connected and the signals are send to the DuT.

## 3. UVM FOR THE CLOUD PATTERN

### 3.1 Context

Today, millions of test kilometer have to be driven to ensure a reliable behavior of the electronic/electrica systems in an car [Maurer et al. 2015]. These procedure has to be done for all the different version of a car-model, each with different features. One standard which is used in industry to test embedded microelectronic systems is UVM. UVM provides capabilities to generate thousand of random test runs with CRV to cover all possible parameters of a system and simulates it in a sequential manner.

### 3.2 Problem

CRV verification within UVM needs a lot of time and resources. Many parameters and stimuli-data have to be tuned and varied to make a statement about the reliability of the system. In a sequential process, each task has to wait until the prior task ends. This has the drawback that verification of a system can take hours, days or even weeks to end. Furthermore, it brings a huge amount of license costs with it, with which companies nowadays have to struggle. In addition, small companies cannot afford big server farms and internal clusters to test their system in an appropriate amount of time. With our approach, we want to speed-up the simulation time within the UVM standard.

### 3.3 Forces

—**Computation Power:** We could use faster processors and server architectures to reduce the runtime of sequential simulation runs, but acquisition and maintenance are associated with high costs.

—**Internal server farm:** Companies can invest in internal server farms for parallel processing, but this brings also high investments with it. Furthermore, it can not be guaranteed that all servers are fully utilized the whole time.

utilization nor maintainable.

—**Loss of tasks:** Problems within the execution or termination of tasks can lead to loss of results and important data. This cannot be done manually and must be automatically ensured by the framework.

—**Architecture:** To ensure a smooth workflow for the verification of thousands of simulations, the architecture and interfaces of the approach must be clearly defined.

—**Time-to-market:** Simulation tasks which takes weeks or even month to finish also have impact on time-to-market. This must be reduced to guarantee deadlines for products.

—**License costs:** Long simulation runs bring a huge amount of license costs with it, with which companies nowadays have to struggle.

## 3.4 Solution

To solve the problems for our large number of simulation tasks we made an adaption of the UVM-SystemC layered architecture by introducing messaging pattern from the Enterprise Integration Patterns [Hohpe and Woolf 2003]. As mentioned before, the overall result of a simulated sequence does not effect other configurations in any way, which leads to the possibility for the parallel processing of sequences. Due to the fact that the simulation of a sequence is the most time consuming part of the verification, a theoretical linear speedup can be expected. This prediction can also be underpinned with the fact that returning results from single CPUs can be neglected compared to the simulation time. From this point of view a cloud based approach was developed, which can be obtained in its main features from figure 3.

3.4.1 *Messaging Pattern.* The messaging pattern allows for a many to many connection where only one address has to be known to all machines. This machine accepts and distributes messages, takes care of message persistence and can optionally monitor all consumers using a heartbeat signal. Retransmission is also build into the message broker (B). The worker connects to the broker (running on the master) and uses a heartbeat signal to indicate it has not crashed or is otherwise unavailable. Worker queues are used to distribute time-consuming tasks among multiple workers in the cloud. This

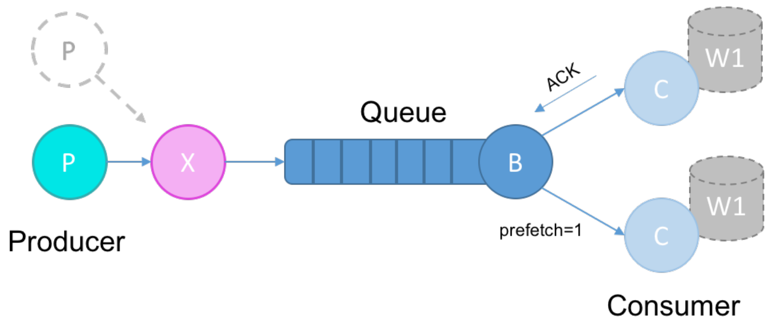


Fig. 2. AMQP worker queue from the enterprise integration patterns

is especially the case if tasks such as simulation-runs can take hours or days to complete. Thereby we avoid doing a resource-intensive task immediately and can schedule the task to be done later and do between them.

A simple working queue pattern is depicted in Fig. 2. The Producer P is a program that creates and sends messages over an exchange (X) to a queue (Queue). The queue stores the messages in a kind of infinite buffer and is not bound to any limits. It is also possible that multiple producers send messages to one queue. A broker accepts and forwards the messages to the consumer (C). A consumer is a program that waits for the messages to receive. Each of the component of this pattern (publisher, consumer and broker) can be swapped out to a separate machine, when thinking towards a cloud-based cluster.

A problem of this architecture can be the loss of tasks. This can be solved by acknowledgments. Acknowledgments have the benefit, that if a worker dies the task will be delivered to another worker. An acknowledge is send by the consumer, if a message is received and processed. If the broker does not receive the acknowledgement the task will be put back into the queue and redelivered to another consumer. This mechanism ensures that no task disappears. This implementation also allows bigger tasks to complete because it resigns of timeout-limits. To make our architecture even more reliable the queue is declared as durable. This means that no tasks are lost, even when the server stops or get killed. Using the quality of service (QoS) method only one message is consumed at a time and none are prefeched from the broker. This ensures that every worker only consumes one task and only after its completion consumes the next. Otherwise, ever n-th worker would get every n-th message (independent of the amount of time it requires to finish a given task).

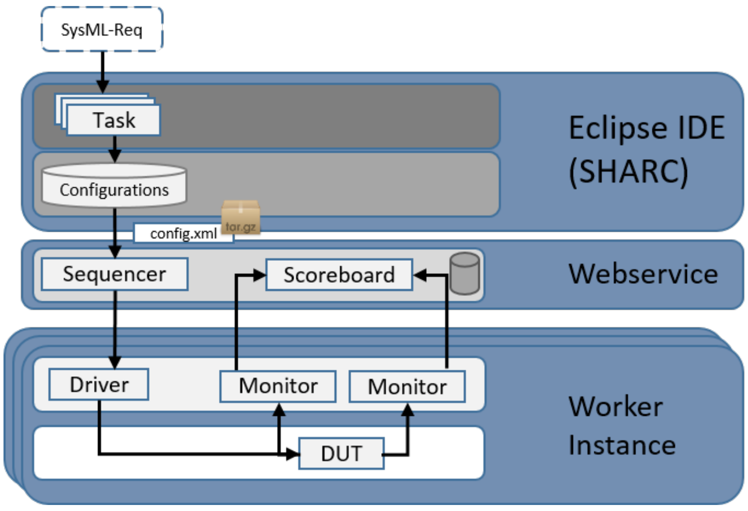


Fig. 3. Layered Architecture of the novel UVM for the Cloud solution

3.4.2 *Adaption of the UVM layered architecture.* In the first step of the verification process, usually the design or verification engineer is building the testbenches depending on the safety requirements he wants to test. In our approach he is supported through our automatic testbench generator. It helps the designer to build very fast verification environments by reusing UVM verification component. The automatic testbench generator takes the safety requirements and constraints defined in the SysML to simulate different scenarios depending on the use-case, the user has to define a preferred amount of sequences. The utilization of these scenarios is done within an override file, which contains a specific amount of override elements. The result is a verification environment model which consists of the outlined components on command layer, such as drivers and monitors, and their according connections to the chosen DuT. All this steps are done in our developed Eclipse framework called SHARC (Simulation and verification of HierARChical microelectronic embedded systems) [Weissnegger et al. 2016a]. However, from that point of view, the created test instance is only capable to simulate one certain scenario, which has been predefined through the default values in the verification environment description. Therefore, a XML schema was defined, which consists of a config and a further override. The aim of

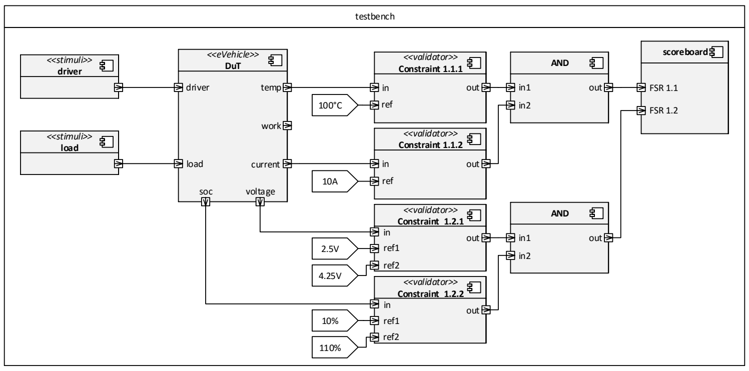


Fig. 4. Automatic generation of UVM verification components around the DuT with UML models

the config description is the name provision of the root model, which should be simulated. Furthermore this includes also the duration of the simulation as well as its timestep. The second element is primarily used to redefine certain values in terms of verification issues. This might for instance be the case, if we want to see the influence of a certain parameter and perform various reruns with changing values. The configuration files are stored in a queue that is processed from a master instance. Depending on the amount of worker the simulation tasks are distributed to the worker instances. The configuration file is passed to the simulation core and utilizes duration, timestep, inner values such as constants and the signal descriptions for instantiated drivers.

Since a simulation is processed on a worker instance, the created bundle has to be deployed from a centralized master. Therefore it is necessary to create an archive containing all referenced files such as the description of the DuT, testbench with stimuli and used component-library. This created package is then sent to a webserver where the worker instances can access them, if necessary. This avoids needless traffic, as well as it leads to a transparent storage of task descriptions. Furthermore, this approach allows to rerun simulations once they have been executed.

Using a large amount of machines not only results in a new linear amount of speedup but also in a linear amount of more data. While it is still possible to evaluate the results of a single simulation locally this gets impractical fast once there are hundreds or thousands of results to analyse. While a location (such as a network share) which is accessible by all machines would be enough to merely store the simulation data it is usually a better idea to store them on a server which also has the actually stores data. It includes the sequencer (master) which distributed the tasks to the worker and a scoreboard which compares the simulation results from a database. Using the webservice API of the framework a new task is created and using the file upload feature a new workpackage including config and package file is uploaded for this task. The master distributes the tasks to the workers. The simulation-results, configuration used and output logs for each task and simulation are send back to the server and stored. The data is then presented by the webserver in a human readable fashion.

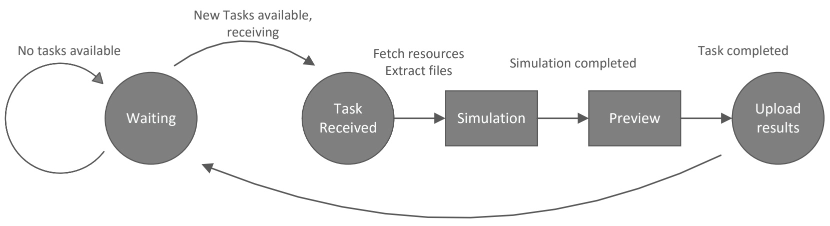


Fig. 5. Worker state machine

The workers are responsible for the execution of the simulation runs and send the results back to the master. The workers connect to the broker running on the master and uses a heartbeat signal to indicate they are available. Figure 5 shows a state diagram which corresponds to the workers behavior. The simulation is executed as a child process so the workers can keep sending heartbeat signals during that time. As the simulation core (SystemC) is unable to use multiple threads for simulation, the ideal setup for a worker is a virtual machine that has only one CPU. Should such a setup be unavailable or physical machines be used instead, it is however possible to spawn multiple worker threads on a single machine as well. The framework does not impose any limitations on the number of workers, usually there are as many workers as possible at any given time, they can however be easily spawned on demand and destroyed if no longer needed (if necessary even during a running simulation as it will be rescheduled by the master in that case). Workers do not need a persistent state as all information for simulation is gathered from the webserver and the master.

## 3.5 Consequences

### 3.5.1 *Benefits*

—**Reduction of simulation-time:** With our cloud-based UVM approach we can reduce the time for simulation of thousand of simulation tasks. Due to the fact that each simulations task is independent from each other and does not effect other configurations in any way, leads to the possibility for the parallel processing of sequences. From our approach, we can expect a theoretical linear speedup. Furthermore, companies benefit from a faster time-to-market.

—**License costs:** Our framework utilizes the full capacity of the server infrastructure in a very efficient way to scale down the number of licenses. Furthermore we use open standards such as SystemC to reduce license costs.

—**Flexible infrastructure:** There is no need for companies to invest in big server farms. Through footprint analysis we are able to predict the time for simulation and therefore can buy simulation

tion time is cheaper.

—**Architecture:** Our pattern provides different levels of detail for the architecture of our approach. Therefore it is defined in detail and guaranteed who is distributing the simulations tasks, what is done when simulations fail and where the results are collected and analysed.

—**Efficiency:** A broker implemented in RabbitMQ automatically distributes the tasks to the worker instances to guarantee an efficient degree of capacity utilization.

—**Loss of tasks:** Through using message acknowledgements we guarantee that tasks are never lost. An acknowledgement is sent back to the consumer to tell that a particular message has been received, processed and that it can be deleted.

#### 3.5.2 *Liabilities*

—**Adapt the UVM standard:** The verification components have to be customized and also have to take over other tasks. The layers architecture pattern from the standard UVM approach has to be adapted. New layers have to be defined or merged.

—**Contract with server providers:** Their must be an existing contract to cloud providers such as Amazon Web Services (AWS) or Microsoft Azure.

—**Security:** Security need to be considered in more detail when sensible data leaves the companies infrastructure. This part will be not considered in our work.

## 4. CONCLUSION

In this paper we presented a novel layered architecture pattern approach to facilitate the parallel execution of thousands of simulations runs, which usually takes weeks or month. The result was an extension of the traditional UVM pattern with message patterns of the Enterprise Integration Patterns. Through applying this new defined pattern to a whole verification environment, we achieved a linear speed-up in the simulation. This framework can now be used to verify the behaviour of embedded, but most notably safety-critical systems in various domains such as automotive, aviation, health-care and many others in a reasonable time. With the help of UVM like components, the whole system can be verified by methods such as constraint random verification which also covers corner-cases. With this new pattern we achieve a speed-up in the execution of intensive simulation tasks and therefore reduce verification costs which leads to a faster time-to-market. The implementation of our approach was done with the help of rabbitMQ [rab 2016] and the AMQP protocol [amq 2016]. AMQP raises the level of security, reliability and performance, since it allows to specify which messages will be received and where from. The execution environment can be switched between cloud solutions such as Microsoft Azure and Amazon AWS, but also internal cluster. The whole environment, including our graphical design and verification tool SHARC, will be published for download and also be used for educational purpose. A next step will be the integration of parallelize-able SystemC code which is under investigation in several research groups.

## 5. ACKNOWLEDGEMENTS

A part of the work has been performed in the project eRamp (Grant Agreement N621270), co-funded by grants from Austria, Germany, Slovakia and the ENIAC Joint Undertaking. Furthermore, we would also like to express our greatest gratitude to Dirk Schnelle-Walka, our shepherd who provided many insightful advice for the patterns in this paper.

2016. Accellera System Initiative Standards. (2016).

2016. Advanced Message Queuing Protocol, AMQP. (2016). https://www.amqp.org/

2016. Amazon Web Services. (2016). https://aws.amazon.com/

2016. Microsoft Azure. (2016). https://azure.microsoft.com/

2016. RabbitMQ. (2016). http://www.rabbitmq.com/

2016. UVM (Universal Verification Methodology). (2016). http://accellera.org/downloads/standards/uvm

Martin Barnasconi and Marie Curie. 2014. Advancing system-level verification using UVM in SystemC. *Design and Verification US (DVCon)* (2014).

Gregor Hohpe and Bobby Woolf. 2003. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

M Maurer, J C Gerdes, B Lenz, and H Winner. 2015. *Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte*. Springer Berlin Heidelberg. 446–448 pages.

Ralph Weissnegger, Markus Schuß, Christian Kreiner, Markus Pistauer, Römer Kay, and Christian Steger. 2016a. Bringing UML / MARTE to life : Simulation-based Verification of Safety-Critical Systems. *2016 Forum on Specification and Design Languages (FDL)* (2016).

Ralph Weissnegger, Markus Schuß, Christian Kreiner, Markus Pistauer, Kay Römer, and Christian Steger. 2016b. *Seamless Integrated Simulation in Design and Verification Flow for Safety-Critical Systems*. Springer International Publishing, Cham, 359–370. DOI:http://dx.doi.org/10.1007/978-3-319-45480-1_29

# Bringing UML/MARTE to life: Simulation-based Verification of Safety-Critical Systems

Ralph Weissnegger*†, Markus Schuß*, Christian Kreiner*, Markus Pistauer†, Kay Römer* and Christian Steger*
*Institute for Technical Informatics
Graz University of Technology (TU Graz), Austria
Email: {ralph.weissnegger, markus.schuss, christian.kreiner, roemer, steger}@tugraz.at
†CISC Semiconductor GmbH, Klagenfurt, Austria
Email: m.pistauer@cisc.at

***Abstract*—The complexity of systems that interact with and have an impact on the physical world, so-called cyber-physical systems, is steadily growing. This is preeminently the case in the automotive domain where we are experiencing an electrification of the modern car, also caused by the trend to electric vehicles. Early verification of these highly heterogeneous sub-systems on system level is an important task in today's development-effort and is also demanded by several safety standards (e.g. ISO26262). Code-generation from design-models and furthermore simulation to verify the system-behavior is done either at a very late stage, or even at the end of the design process. This situation prompts the urgent demand for new techniques to simulate the behavior in early development-phases by reusing verified system-components. In this paper, we present a novel method supported through a model-based simulation framework based on a standardized modeling language (UML/MARTE). Here we show that UML supports both, analytical methods such as FMEA, FTA and simulation for safety-critical systems in one methodology and tool. Our approach allows early simulation of UML/MARTE design models within the design flow of ISO26262. To demonstrate its benefits, our methodology is applied to an industrial use-case of a battery management system. Results show significant improvements compared to other state-of-the-art approaches.**

## I. Introduction

In our fast-paced modern world the proportion of embedded systems in various domains is increasing enormously. This is also a fact in the automotive industry. It can be observed that there is a shift towards fully electric/electronic (e/e) systems also caused by the trend towards electric vehicles. The industry is facing with new problems through the emergence of many new features in cars that are also influencing each other [1]. This raises the complexity levels in the design and development of large systems and imposes an enormous effort for engineers from different domains in developing their applications. Old approaches are becoming less effective and this is precipitating the needs for a paradigm change in design and verification of larger systems. It is no longer sufficient to test the behavior of single sub-systems, the focus must be on the system as a whole and not on parts taken separately - "the whole is more than the sum of its parts". This familiar phrase is also a recommendation in the functional safety standard ISO26262 for automotive e/e systems in road vehicles [2]. It is an approach that must be introduced very early in the design and development process to avoid later changes that causes costs and time-delays.

A model-based approach is an important basis for engineers and multiple stakeholders in their quest to overcome these complexity-issues. It helps designers to gain a quick and augmented view of the system and provides an effective means of communication, especially if systems are very complex and involve a number of teams in the design. It also helps in coping with the huge amount of requirements that must now be faced.

A model-language which is already established in the embedded system domain is MARTE [3]. It is an extension of UML2 and provides capabilities to model hard- and software, as well as timing, resource and performance-behavior. It is used by many semiconductor vendors and suppliers today and is the driven system-design language in the European Catrene-project named OpenES [4]. The OpenES is a European initiative to fill the gaps in today's system-design and to develop common solutions to stay competitive.

Another advantage of UML/MARTE is that it is currently supported by several commercial and open-source tools like Eclipse's Papyrus [5], that helps designers to model systems in UML and extensions like SysML or MARTE. As a result of its outstanding composition involving several levels of detail in compliance with the Model Driven Architecture (MDA) MARTE helps to specify the system on every abstraction, up to very detailed platform specific models. With UML/MARTE it is possible to model the whole design flow within the functional safety standard ISO26262 as shown in [6]. Also various recommended verification methods like Fault Tree Analysis (FTA), Failure Mode and Effect Analysis (FMEA) or hardware evaluation can be applied on these models [7],[8]. Another way is to use these models to create systems from specifications with code-generators for synthesis from VHDL models or simulation-models in SystemC. The drawback of this approach is that simulation of systems is done very late in the design process when changes are very costly. Also later changes in code are resulting in inconsistent models. Furthermore reverse-engineering is error-prone and cumbersome. Another issue is reusability of already verified system components that cannot be left ignored. The majority of components in new projects are reused and simply extended by the addition of new features to save costs and time-to-market. This calls for new ways of

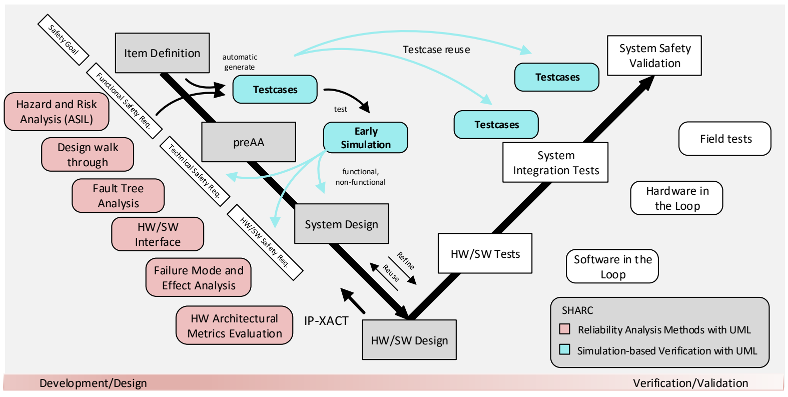


Fig. 1. Simulation-based verification, tight and seamless integrated in the design flow of ISO26262

testing advanced features in today's development process.

We present a novel methodology to support MARTE with fast executable models in this work. We link the implementation of several hardware and software systems written in SystemC (-TLM) and SystemC-AMS, with MARTE design-models on different abstraction levels for this purpose. Since our modeling methodology has a clear graphical representation of composition and supports the part, ports and connector concepts, it is well suited for system-design and system-configuration. The level of granularity of the models can be easily switched depending on the complexity. Simple models serve as starting point for the highly detailed models, that are later on used as golden reference or even synthesis. Through these capabilities we achieve an early behavior simulation of the whole system. Subsystems can be composed to nested components to reduce the complexity. Whole system can be easily integrated in a testbench (UVM) and parametrized or even completely reconfigured for experiments under various conditions. Our approach closes the gap between early design and simulation of safety-critical systems, tight and seamless integrated in the design flow of ISO26262, depicted in Fig. 1. From the results of the simulation we obtain more information about the expected behavior (timing, power, thermal) of the system and can derive further requirements for safety. We demonstrate the efficiency of our framework, with an electric vehicle use case on system level.

## II. Related Work

How MARTE is used in the development process of E/E systems is presented in several papers [9],[10],[11]. They show how MARTE complies with the model-driven architecture and the computation independent model (CIM), platform independent model (PIM) and platform specific model (PSM) levels. In these papers it is also discussed how to model hardware and software on different abstraction levels in the design process. They also address the issues of modeling non-functional properties and the mapping from software applications to platforms. The drawback of these approaches are that code-generation of MARTE models is done very late in the design process and also reverse engineering is an error-prone and cumbersome task.

The authors of [12] demonstrated how to use MARTE for hardware design and simulation. They introduced a step-by-step methodology for hardware modeling with Hardware Resource Models (HRM) stereotypes. The platform models are refined until the final platform class is reached. In a later step, these models are used to generate code with the help of a Java plugin. A tool called Simics was used to facilitate the simulation. Instead of using the whole MARTE spectrum for simulation this approach only uses HRM models for code generation of very detailed platforms.

How to generate SystemC (-TLM) and C++ code from UML/SysML models for HW/SW co-design was presented in [13]. To facilitate the mapping between UML/SysML to hardware and software three profiles for UML have been introduced, e.g. for synthesizable SystemC, synthesis extensions and furthermore C integration. This paper also showed a design flow from UML to code generation for hardware and software. The authors demonstrated their approach by using two case-studies. The first use case focus on co-modeling and co-simulation to indicate the performance of their concept. The second one shows the design flow in real applications and the refinement to FPGAs. This approach would benefit from a more detailed UML-profile for hardware and software such as MARTE.

## III. Model-Based System Design

MARTE [3], [14] is a domain-specific modeling language and was defined as an adaption from the OMG to address the shortcomings of modeling platforms in UML. It is intended for model-based design and analysis of real-time and embedded software of cyber-physical systems. MARTE is defined as a profile in UML2 and provides additional mechanisms for modeling real-time systems, which are missing in UML. Thanks to

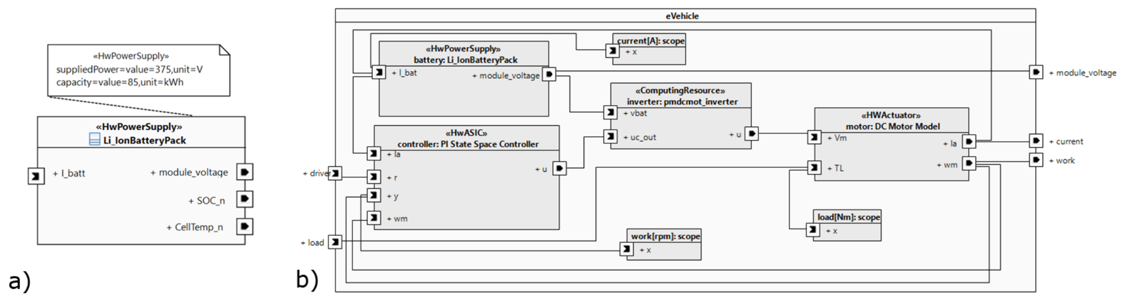


Fig. 2. **a) depicts an example component of the SystemLibrary with non-functional properties b) shows the system-design consisting of heterogeneous sub-systems connected through ports**

the UML extension mechanism, the software resource model (SRM) and hardware resource model (HRM) profiles extend UML2 with concepts for software and hardware. The purpose of the SRM packages is to design software of real-time and embedded applications. It consists of the *SW_ResourceCore* which provides the basic software resource concept. The HRM is an extension to UML and serves as a description for existing and the conception of new hardware platforms. These description can be made of different levels of granularity. The HRM is grouping most hardware concepts under a hierarchical taxonomy. It is composed of a logical view (HwLogical) that classifies hardware resource depending on their functional behavior and a physical view (HwPhysical) that focus on the physical nature. The HWLogical model provides a classification for different hardware entities such as computing, storage or communication. This includes stereotypes like *HwASIC*, *HwProcessor*, *HwBus*, *HwDevice* or *HwMemory*. All the stereotypes defined in the HRM package are organized under a tree of inheritances from more generic sterotypes. This has the advantage, that if no stereotype suits to a used hardware component, a more generic stereotype may fit instead. As an example, a *HwSensor* inherits the properties from *HwI/O*, and is furthermore a specialization of *Hw_Device*. In contrast, the HWPhysical package contains stereotypes as physical components. They describe their shape, size, position within platform, power consumption or other physical properties.

In addition it is possible to allocate software applications to hardware resources with the help of the MARTE allocation mechanism. For the modeling of systems on a higher level of abstraction, MARTE provides capabilities with the general resource models (GRM). These models can be used for components, where no early assumptions about implementation in hardware or software can be made. The stereotypes offer concepts to model general platforms for executing real-time embedded applications. This includes the modeling of both, hardware and software. With this package it is possible to model complete systems on a very high abstraction-level. This helps us to model systems very early in the design process, when design choices are still undecided. The GRM package includes different *resource* types, representing a physically or logically persistent entity e.g. *ComputingResource* or *StorageResource*. These Resources offer services to perform the expected tasks.

Another stereotype which helps to simplify the modeling in a component-based approach is GCM (General Component Models). It brings the advantage of describing ports with information about incoming (in), outgoing (out) or bidirectional (inout) communication of the different subsystems. These *FlowPorts* have been introduced in MARTE to enable a flow-oriented communication paradigm between components.

## IV. System-Libary

To avoid the design and simulation of larger systems from scratch and furthermore achieve reusability, our methodology provides a SystemComponentLibrary. This library includes all major components for a high-level simulation of systems from different domains e.g. automotive, mobile computing or multi-media. It also includes components in different versions and on different abstraction levels. These models serve on one hand as the starting-point for future development and furthermore as a golden reference for integration aspects. The components are modeled as UML-Class in a composite structure diagram as depicted in Fig.2 a). The UML-class owns the attributes and properties of the component. Our example shows a UML-Class named Li_IonBatteryPack, tagged with HRM *PowerSupply*. To describe the inputs and outputs of the battery, the ports are tagged with MARTE FlowPorts. The stereotype PowerSupply allows us to define different configurations for the simulation e.g. multiplicity (number of cells), powersupply, capacity or frequency of the battery. Besides this also non-functional properties for power like energy consumption or dissipation are used for the parametrization of the battery. The mapping between MARTE and SystemC is decribed by an example of a battery in TableI.

To raise the reusability and provide a good support for developers, the SystemComponentLibrary is built as an Eclipse-plugin. New models can be generated and added to the library. Updates for components can be easily checked by updating the library from the server. This helps to support design-teams by adding new components and keeps the library consistent.

In order to bring the components of the SystemComponentLibrary to life, they are linked to executable models in SystemC(-TLM) or SystemC-AMS. These steps towards high-level simulation of cyber-physical systems are described in the next chapter.

TABLE I
MARTE TO SYSTEMC MAPPING

| MARTE | | | SystemC | | |
|---|---|---|---|---|---|
| Name | Stereotype/ Attribute | Type | Name | Class/ Attribute | DataType |
| LiIonBattery | HwPowerSupply | | li_ion_battery | sc_module | |
| | suppliedPower | NFP_Power | | vBatt | double |
| | capacity | NFP_Energy | | capacityCell | double |
| | r_Conditions | Env_Conditions | | ambientTemp | double |
| | staticDissipation | NFP_Power | | dissiCell | double |
| | frequency | NFP_Frequency | | sampleTime | double |
| | resMult | NFP_Integer | | numberCells | int |
| Connector | DefaultLink | | | sca_signal | |
| CAN_Bus | HwBus | | can_bus | sc_module | |
| | adressWidth | NFP_DataSize | | addr_width | sc_bv |
| | wordWidth | NFP_DataSize | | data_width | sc_bv |
| | bandWidth | NFP_DataTxRate | | txrate | int |
| I_batt | FlowPort | | ibat_lsf_in | sca_in | |
| module_voltage | FlowPort | | vbat_lsf_out | sca_out | |
| SOC_n | FlowPort | | soc_n_lsf_out | sca_out | |
| CellTemp_n | FlowPort | | cellT_n_lsf_out | sca_out | |

## V. Executable SystemC-Models

SystemC is a system-level modeling language for the development of software and hardware models on different levels of granularity, from high level simulation to register transfer level (RTL). It is defined by Accelera, a standards organization in the area of electronic design automation (EDA). SystemC is an open standard and has been also approved by the IEEE standards association. On the transaction level modeling level(TLM), a very high level simulation, the focus lies on communication and functionality. This serves as a golden reference for lower level hardware models. This very detailed level (RTL) is pin accurate and the focus lies on signals. SystemC enables the design-teams to have a fundamental understanding of the system at an early stage of the design process. Through its high flexibility it enables to represent a complete system.

SystemC tries to bridge the gap between hardware description language (HDL) and object-oriented language (OOP). While SystemC is commonly used in the context of a system on chip or to model several components in a system, it is usually limited to the digital domain. In order to address complex systems with digital and analog parts, an extension was introduced under the name SystemC-AMS. This enables the simulation of continuous-time, discrete-time and discrete-event behavior of analog/mixed-signals simultaneously. Nevertheless, SystemC lacks a visual representation for interacting with different stakeholders and their requirements. Because of its C++ background, SystemC is object-oriented and has a lot of similarities with UML and MARTE. This has also been acknowledged by several publications [15],[16],[17] and makes the linking with MARTE models intuitive. Because of this and its wide acceptance in the industry and availability, SystemC was chosen as the primary simulation language in our approach. Our methodology bridges the gap between model-driven design in MARTE and executable models in SystemC.

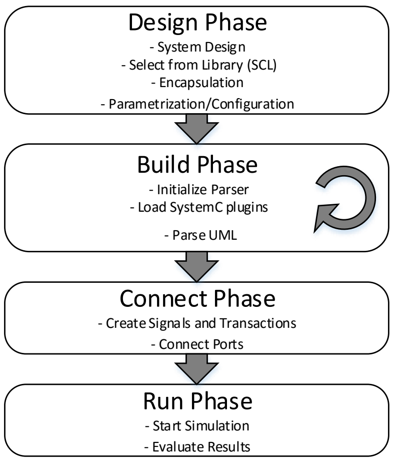


Fig. 3. Methodology for executable SystemC-models from MARTE design

The executable models used in our approach are models on different abstraction levels and in different versions. The generic models have the potential to be used in various domains and support reusability. The detailed models are refinements of the generic models and have the purpose to be used in special domains. Dependent on the domain and abstraction level, the models are built in SystemC(-TLM) and SystemC(-AMS). All the information for the configuration of the system is extracted from the UML-files by the framework at runtime. To speed up the simulation time, we created already compiled binary files from implementation code. This has the advantage of being able to parametrize or even reconfigure systems and components without the need for recompiling the code every time the system is simulated.

## VI. Methodology

Our methodology is composed of four phases as depicted in Fig.3: Design-Phase, Build-Phase, Connect-Phase and Run-Phase.

### A. Design-Phase

The first part in our methodology is the system-design. The designer creates an UML top-class in our example eVehicle, depicted in Fig. 2. This class describes the overall structural architecture of the system. It is composed of the different instances provided by the SystemComponentLibrary. They can be easily included- and excluded from the top-class by using the drag and drop mechanism. These sub-systems are connected together by ports according to their specification. Through the MARTE flow-port capability, these ports are checked in advanced by the framework on the correct direction of the dataflow e.g. two output-ports are connected together. To trace the dataflow, *Scope*-models are added to the system and connected to the ports. These *Scopes* write the monitored data to the trace-files to compare the results from different tests and testbenches in a latter step.

Our framework also provides a mechanism to encapsulate existing components and to raise the abstraction level of the whole system. Smaller systems that model the interior design of the class can be merged to a more simplified model. This helps the designer to have a better view of the system, without having too many details in the models on system-level. This mechanism allows us to abstract the complexity of components.

The whole eVehicle system, also referred to in our case Design Under Test (DUT), is provided with several connectors. These signals required for testing and debugging are brought out to the ports of the top-class itself. This has the advantage of connecting testbenches to the DUT for testing various scenarios of the electric car and also for monitoring the performance. The outcome of the design step is a netlist that also serves as configuration and parametrization for the simulation. It is the starting point for the build-phase.

### B. Build-Phase

The heart of our Build-Phase is the self-defined parser-methodology. The purpose of the parser is to translate a UML model defined in one or more files to a single SystemC system. This is done at run-time and does not require compilation of the resulting model. When the parser is initialized it needs the name of the top-class of the model in the diagram as well as the name of the UML-file that contains the model. Starting from the root node of the UML model, each child of a node is parsed and returned. As single systems can be composed of more detailed sub-systems, we had to define a loop to find all properties and ports of each root note. Each node found in the UML-file is treated as the new root node and each sub-system is created before moving on to the Connect-Phase. Each system may contain any number of sub-systems, therefore this step is done in several iterations till all properties of the root node are found. In order to keep the framework extensible a DLL-based plugin system is used. All the information is stored in a ConfigStore map.

### C. Connect-Phase

In this phase the connector-objects are created to link the different instances in the Build-Phase. Depending on their nature, the connector-objects can be signals or transactions. It is important to notice, however, that UML allows multiple 1:1 connections per port, SystemC merely allows a port to be bound once but a signal may connect any number of ports (basically 1:n as only one driver is allowed per signal). As a means of handling these issues both ends of each connector are tagged by an ID. Instead of creating new signals for connecting to a used port, the old signal is reused.

### D. Run-Phase

After all nodes, ports and properties of the UML-file are found by the parser, the SystemC instances created and connected, the simulation is started. A result of the Scopes is saved as a trace-file. This file contains all the relevant information required for the verification of the model or to evaluate the behavior of the model for different parameters and/or implementations for the system. Besides this, a logic can also be added to the system to react to certain events such as stopping the simulation in case of a signal violating the given constraints or the system running out of energy (for a battery powered system). An implemented dialog is also used to configure the settings for the simulation such as duration or timestep (resolution).

Using the Toem Impulse plugin [18] for Eclipse the results are presented in a graphical form. The results can be displayed in the desired manner, dependent on the nature of the simulation e.g. analog interpolation for real values and numeric representation for digital signals in a hierarchy that allows for easy interpretations. The results may be verified against the known or expected behavior of the (physical) system modeled. If the system behaves as expected, it can be used for further analysis or verification (e.g. as a golden reference model or synthesis-able).

## VII. Example Case Study: Electric Vehicle Simulation

We have applied our methodology to an industrial use case, an electric vehicle (eVehicle) system provided by CISC Semiconductor GmbH, to more fully illustrate its innovative capabilities and benefits. As more and more vehicles are now powered by Li-Ion-batteries, the challenge for engineers to ensure reliability and fault-tolerance is also greatly increasing. Problems with overheating or even explosions have been frequent in the past. The main cause of these problems was an excessively high energy intake from regenerative braking or harsh environmental conditions. Management systems and mechanisms are thus essential to ensure that persons are not put at risk and that no damage is caused. This calls for safety mechanisms to reduce the number of faults.

The overall system model of the eVehicle is depicted in Fig.2 b). It is composed of the driver, battery, controller, inverter, dc-motor and external load. The *driver* provides the desired speed for the eVehicle. The speed be set according to standardized maneuvers (NEDC). The *controller* is a model for a PI state-space controller and maintains a constant speed based on the information about the state variables motor armature current and motor-speed. It is realized as ASIC and is configured by the MARTE HwASIC stereotype. The *inverter*-model implements an inverter function for a PM-DC motor driving stage. It is comparing the actual battery voltage and the requested controller voltage to maintain the PM DC motor terminal voltage. It is annotated with ComputingResource stereotype. The *battery*-model simulates the behavior of a LiIon-battery pack composed of a defined set of single-cell LiIon batteries. The appropriate number of single cells is connected in parallel and series to obtain the necessary capacity and terminal voltage. The battery-pack terminal voltage is calculated based on the defined parameter and the battery current. It also considers the temperature of the module and its State-Of-Charge (SOC). The Battery Model also includes a Battery Management System (BMS), which measures the values coming from the included sensors of the battery-cell. The BMS computes the SOC, State-Of-Health (SOH) and is responsible for cell balancing, cell protection and demand management of the battery. In addition the external load and temperature of the environment can be changed during the simulation.

## VIII. Results

In order to also illustrate the effectiveness of our extended Papyrus-framework (Simulation and verification of HierARChical embedded microelectronic systems called SHARC) we compared our results with a state-of-the-art tool in this domain: Matlab Simulink. The exact eVehicle, built as a Simulink-simulation, was used as golden reference to compare the execution-time and memory-footprint for benchmarking. Fig.5 presents the monitored signals from the simulation, verified by UVM-like components. The output of the Simulink run is referenced as *golden_ref* signal. The last signal (*deviation*) shows the difference between signal *work* of both simulations. Only when it comes to a step in signal *load* there is a peak in the deviation with about 0,5 percent. This occurs because the SystemC simulation kernel requires an additional delay at this step where Simulink with his centralized timing solver, does not. This produces a short shift between both signals but ends up, after a timestep, in the same results. The average error is 0,0081 percent. Both simulations have the same accuracy with a fixed timestep of $1 \times 10^{-3}$s.

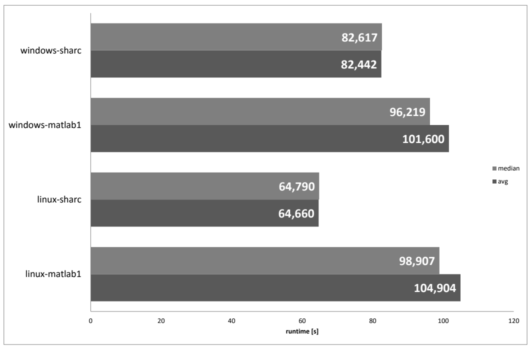


Fig. 4. Benchmark results from Matlab and SHARC simulation

Instead of only measuring the accuracy of our framework, we furthermore compared the time needed for simulation against Simulink. To guarantee a fair comparison in relation to the size of the environment, both SHARC and Simulink have been executed ”headless” (without graphical user interface (GUI)). As there is no discrepancy whether SHARC is executed with or without GUI regarding simulation time, we executed Simulink without display, desktop, splash nor jvm. Figure 4 shows the results of the benchmarks, executed on Windows (Windows 7) and Linux (Ubuntu 16.04). The electric vehicle use case includes 10 runs with different parameter sets, where the goal is to find the optimum set which also satisfies the safety constraints and requirements. This is not limited to 10 runs but rather results in thousands of simulation tasks, which have to be executed to derive further requirements and outcomes. For demonstration reasons we have limited our simulations to 10 runs. More information on this approach can be found in publication [19].

The average simulation time of 10 runs in Matlab needs 104,904 seconds to execute our electric vehicle use case on Linux. Compared to our methodology and lightweight tooling it only takes 64,66 seconds of simulation time. This is an average saving of 38 percent in simulation time, which also results in reduction of license and infrastructure costs. In the first of the 10 runs there is a major difference in the execution-time of both tools. This drift is a result of the high memory-usage (>500MByte) of Simulink, by contrast, our tool uses less than 10MByte of memory for the same simulation. After the first run the execution time improves.

Another issue is that Matlab requires the connection to a license server, which also causes fluctuation in simulation time. As we are working with the GNU license we do not have to take license costs nor bandwidth problems with license servers into account. All simulations have been executed on an Intel Core i5-3550 @ 3.3GHz, 8GB RAM, HDD: ST500DM002 machine.

Our framework has also the benefit to run several simulations in parallel on distributed simulation cores (cloud-based simulation) without running several instances of the tool-GUIs. Our lightweight tooling will have a major impact for future cloud-based solutions (small simulation core, no license costs).

## IX. Conclusion

In this paper we presented a model-based simulation framework for fast and effective simulation of safety-critical systems. We used UML/MARTE for a graphical representation of components, because of its great capabilities for modeling both software and hardware. MARTE helped us to model systems from different domains and on different abstraction levels. To bring MARTE to life, SystemC implementation-models have been linked to the modeling environment with the help of our methodology. This includes both, digital and also analog-mixed signal simulation with SystemC. A model-library was introduced that helps to speed up the design and development process and raises reusability. Because of using UML/MARTE models, our approach can be easily and seamlessly integrated the design flow of safety-critical systems (ISO26262), so that the UML models can be used for FMEA/FTA but also simulation. Further requirements for safety are derived from the results of the whole system simulation in early phases (timing, power, thermal). Complex systems can be tested by parameter variation and testbenches without recompiling the whole system for fast verification. The detailed specification of models in MARTE for hardware and software helped

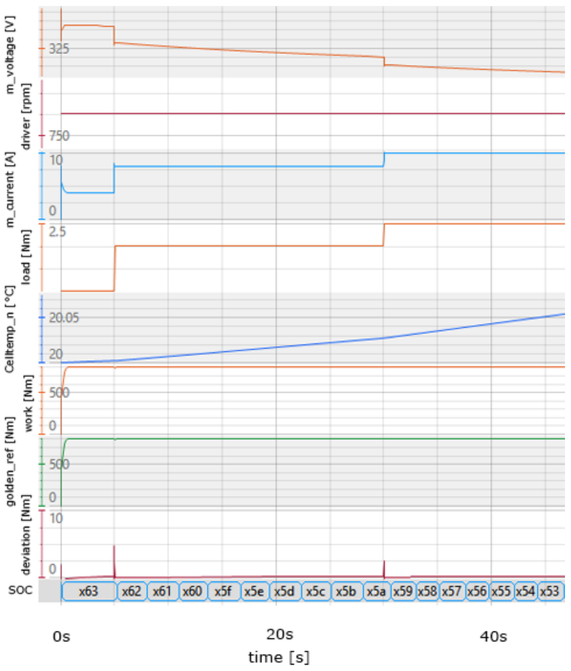


Fig. 5. Output-trace of the eVehicle simulation, the signals are compared to a golden reference

us to configure and parametrize our system. Because our approach was developed as a plugin in Eclipse, every Papyrus UML editor is now capable of simulation by installing our plugin. This tool will be published for download and also be used for educational purpose. To show the efficiency this framework was tested regarding accuracy and simulation time by a complex example from the automotive industry. In a next step, the verification of the system will be improved by UVM testbenches (constraint random verification) written in SystemC. The reusability aspect will also be improved by relying on standards such as IP-XACT. Because our simulation framework shows best results concerning small memory-footprint and fast execution-time it will be used additionally for parameter variation in cloud-based environments.

## Acknowledgment

The approach presented above is experimented in the OpenES CATRENE Project: CA703 - 2013 research program supported by the FFG (Austrian Research Promotion Agency), project-number 843380 in tight cooperation with CISC Semiconductor.

## References

[1] R. N. Charette, “This Car Runs on Code - IEEE Spectrum,” 2009. [Online]. Available: http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code

[2] ISO, “Functional Safety ISO26262 - Part 4: Product development at the system level,” vol. 2011, pp. 1–35, 2011.

[3] “The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems,” 2015. [Online]. Available: http://www.omgmarte.org/

[4] Catrene, “OpenES CATRENE Project: CA703,” 2016. [Online]. Available: http://www.ecsi.org/openes

[5] Eclipse, “Papyrus,” 2015. [Online]. Available: https://www.eclipse.org/papyrus/

[6] R. Weissnegger, C. Kreiner, M. Pistauer, K. Römer, and C. Steger, “A Novel Design Method for Automotive Safety-Critical Systems based on UML/MARTE,” in *Proceedings of the 2015 Forum on specification & Design Languages*, Barcelona, Spain, 2015, pp. 177–184.

[7] F. Mhenni and N. Nguyen, “Automatic Fault Tree Generation From SysML System Models,” *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), Besancon, France*, 2014.

[8] R. Weissnegger, M. Pistauer, C. Kreiner, K. Römer, and C. Steger, “A novel method to speed-up the evaluation of cyber-physical systems (ISO 26262),” in *Intelligent Solutions in Embedded Systems (WISES), 2015 12th International Workshop on*, 2015, pp. 109–114.

[9] L. G. Murillo, M. Mura, and M. Prevostini, “MDE Support for HW/SW Codesign: A UML-based Design Flow,” *Advances in Design Methods from Modeling Languages for Embedded Systems and SoCs*, pp. 197–212, 2010.

[10] J. Vidal, F. D. Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet, “A co-design approach for embedded system modeling and code generation with UML and MARTE,” *2009 Design, Automation & Test in Europe Conference & Exhibition*, 2009.

[11] A. Koudri and D. Aulagnier, “Using marte in a co-design methodology,” *UML Workshop at Date*, 2008.

[12] S. Taha, A. Radermacher, and S. Gérard, “An Entirely Model-Based Framework for Hardware Design and Simulation,” *International Federation for Information Processing (IFIP)*, pp. 31–42, 2010.

[13] F. Mischkalla, D. H. D. He, and W. Mueller, “Closing the gap between UML-based modeling, simulation and synthesis of combined HW/SW systems,” *Design, Automation and Test in Europe Conference Exhibition (DATE)*, 2010.

[14] B. Selić and S. Gérard, *Modeling and analysis of real-time and embedded systems with UML and MARTE*, 2014.

[15] L. Murillo, M. Mura, and M. Prevostini, “Semi-automated Hw/Sw Co-design for embedded systems: from MARTE models to SystemC simulators,” *2009 Forum on Specification & Design Languages (FDL)*, 2009.

[16] P. Peñil, E. Villar, H. Posadas, and J. Medina, “Executable SystemC specification of the MARTE generic concurrent and communication resources under different Models of Computation,” *Proc. of Satellite Workshop of the 21st Euromicro Conference on Real-Time Systems*, 2009.

[17] É. Piel, R. B. Atitallah, P. Marquet, S. Meftali, S. Niar, A. Etien, J. J.-L. Dekeyser, P. Boulet, and I. Europe, “Gaspard2: from marte to systemc simulation,” *DATE’08 Workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile*, vol. 8, pp. 1–6, 2008.

[18] “Toem Impulse,” 2016. [Online]. Available: http://toem.de/index.php/projects/impulse

[19] R. Weissnegger, M. Schuß, M. Schachner, M. Pistauer, K. Römer, and C. Steger, “A Novel Simulation-based Verification Pattern for Parallel Executions in the Cloud,” in *21st European Conference on Pattern Languages of Programs Proceedings*, 2016.

# Seamless integrated Simulation in Design and Verification Flow for Safety-Critical Systems

Ralph Weissnegger, Markus Schuß, Christian Kreiner,
Markus Pistauer, Kay Römer, and Christian Steger

Institute for Technical Informatics, Graz University of Technology (TU Graz), Austria
CISC Semiconductor GmbH, Klagenfurt, Austria
{ralph.weissnegger,markus.schuss,christian.kreiner,roemer,
steger}@tugraz.at,markus.pistauer@cisc.at

**Abstract.** In the automotive domain, safety plays an ever increasing role in the development of future vehicles. Since the automotive market is heading towards fully automated driving cars, the amount of new assistance features for ensuring safe and reliable operations is rising. Today, requirements, design and verification must follow the stringent specifications from standards such as ISO26262 for functional safety. Thus, simulation in early design design phases is key to develop safe and reliable systems and to reduce costs and time-to-market. UML as a model-based approach, helps to overcome the complexity issues of safety-critical systems and improves the communication between different stakeholders (e.g. hardware, software, safety, security). In this paper, we present a novel methodology to automatically generate testbenches for simulation based verification starting from a first safety analysis and derived safety requirements. Through early simulation of UML/MARTE models with constraint random stimuli and parameters we are able to derive further requirements for safety-critical system development. Furthermore, our approach is compliant with the requirements, design and verification flow of ISO26262. We will show the benefits by applying our methodology to an industrial use case of a battery management system.

**Keywords:** ISO26262, safety, automotive, process, UML, MARTE, verification, simulation, model-based

## 1 Introduction

In the world of today, the increasing number of new assistance features for ensuring safe and reliable operation in modern vehicles, also have the implication of increasingly complex systems. The development and verification effort of these highly complex systems in an ever increasing and more elaborate task, since the amount of electric/electronic (e/e) components is steadily growing. In safety terms, these systems must fulfill standards such as ISO26262 [4] (functional safety standard for road vehicles). Therefore, OEMs and their suppliers are required to develop and test their systems according to certain levels, alias ASIL levels.

In the effort to cope with the high complexity in the design of safety-critical systems, a model-based approach helps to unite stakeholders from different domains. Furthermore it supports non-safety specialists in understanding the problems of the design of safety-critical systems. In addition to this, provides great help in coping with the vast range of requirements that must currently be met. MARTE was introduced as an extension of UML2 to overcome the high complexity in the design of real-time and embedded systems. MARTE provides capabilities to model hardware and software, as also timing, resource and performance behavior. It is used by many semiconductor vendors and suppliers and is the driving system-design language in the European Catrene project entitled OpenES [3].

Simulation plays an ever increasing and important role in the verification of the modern car because of its advantages in easily varying the virtual environment and also representing the car in different variations, and this not least from an economic perspective. These tests can be monitored and reproduced every time. Another advantage of simulation is not only can it be run day and night, but also massively in parallel.

In this work, we present a novel methodology to simulate and verify MARTE designs supported through our Eclipse framework called SHARC [1] (Simulation and verification of HierARChical embedded microelectronic systems). With the help of our library, we link fast executable digital, analog mixed signal and mechanical simulation-models with MARTE design models. These simulation-models are implemented in open-source languages such as SystemC (-TLM) and SystemC-AMS. Through these reusable components we achieve an early behavior simulation of the whole system. The advantage of our approach is that design models are tightly and seamlessly integrated into the design flow of ISO26262. From this early system level simulation we are able to obtain further requirements for the design of hardware and software for real-time applications (timing, power, thermal). With our proposed solution there is no need to switch between several design or verification tools. Both state-of-the-art analytical methods and simulation-based verification can be handled by using MARTE, SysML and our approach. Tests derived from safety requirements can be reused throughout the entire development cycle until final system integration and validation. We use constraint random verification, as defined in the UVM standard, to cover all possible parameters and various variants of a vehicle. Any shortcomings in the design can thus be detected much earlier in the development process to reduce costs and time-to-market.

## 2 Related Work

Popular approaches [5],[10],[8] have shown that analysis and verification of UML models with methods methods such as failure mode and effect analysis (FMEA), fault tree analysis (FTA), design space exploration (DSE), design walk through, hardware architectural metrics evaluation or even code-generation are very efficient for testing safety-critical systems. The drawback of UML, in terms of code-generation and simulation to verify the system-behavior is that this is done at a very late stage or even at the end of the design process when all details are well known. Later changes in design are costly, they result in inconsistent models and furthermore reverse engineering is an error prone and cumbersome task. The majority of components in new projects are reused and simply extended by the addition of new features to reduce costs and time-to market. The reuse of complete safety concepts, well-trusted designs and mechanisms is thus growing more important as a means to reduce the effort in developing complex systems. This situation prompts the urgent demand for new techniques to simulate the behavior in early development phases by reusing verified system components.

In [9] the authors presented three different analysis techniques for architectural models described in EAST-ADL, to guarantee the quality in the context of ISO26262. One of the proposed techniques is the simulation of EAST-ADL functions in Simulink. The behavior of each function was linked to FMU or Simulink models to facilitate the simulation. The authors also described mapping rules for the EAST-ADL to Simulink transformation (one-to-one mapping). The results of the simulation have been traced back to the requirements. This approach was applied to an industrial use case of a brake-by-wire system on design level. In contrast to our approach, however, they use proprietary simulation engines with high license costs and external tools which are not integrated into the design and development flow.

The authors of [11] demonstrated how to use MARTE for hardware design and simulation. They introduced a step-by- step methodology for hardware modeling with Hardware Resource Models (HRM) stereotypes. The platform models are refined until the final platform class is reached. In a later step, these models are used to generate code with the help of a Java plugin. A tool under the name Simics was used to facilitate the simulation. Instead of using the whole MARTE spectrum for simulation, this approach only uses HRM models for code generation of very detailed platforms instead of system level design.

In [6] the authors presented a simulation-based methodology for requirements verification of SoC designs. This automatically generated a white-box and black-box verification platform from requirements specified in textual specification format. During a simulation-based verification these very fication platforms are simulated together with the SoC design to verify whether or not they fulfill the given requirements. Lexical, syntax and semantic analysis were used to parse textural requirements into a semi-formal format. This approach would benefit from a standardized format such as SysML to de- fine the requirements in tight interaction with the system design. Furthermore, this approach cannot be adapted to an industrial use case.

## 3 Methodology

Since the design and development of safety-critical systems is a cumbersome and costly task, it needs novel methods to test evaluate the design both in the early phases and also during and throughout the entire development process. The reusability of well-tested designs, mechanisms or even complete safety concepts is an issue that is currently becoming ever more prominent. Against this background we thus propose simulation-based verification of UML/MARTE design models on the preliminary architectural assumption (preAA) level, depicted in 1. For this simulation we are using our reusable components from our System Component Library (SCL). This library includes all major components for a high level simulation of systems from different domains e.g. automotive, mobile computing, health care or multimedia. It also includes components in different versions and on different abstraction levels. These models serve on the one hand as the starting-point for future developments and furthermore as the verified and golden reference for integration aspects. The properties of the models are all taken from the standard definition for UML/MARTE system, hardware and software models. In order to bring the components of the SCL to life, they are linked to executable models in SystemC(-TLM) or SystemC-AMS. More information on this methodology is given in [12] and [13] . Based on the functional

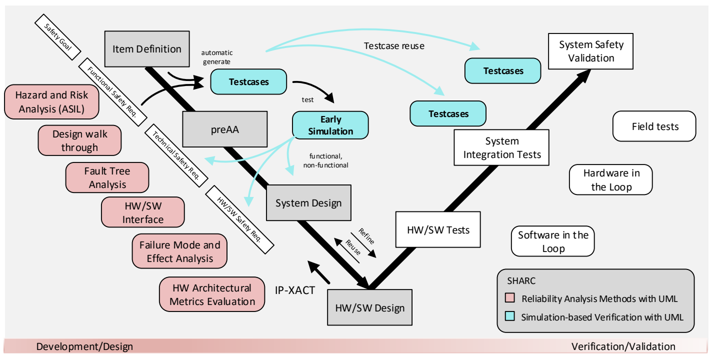


**Fig. 1.** Seamless integration of simulation-based verification in the ISO26262 design flow

SRs from the functional safety concept, defined as SysML models, and the information from the preAA we are able to obtain further requirements for the technical safety concept, described in chapter 3.1. By also taking non-functional properties (timing, power, thermal) into account, we are able to refine the functional SR and to define the technical SR. Furthermore we are able to obtain inputs for our final system design before the step of costly implementation of faulty design is taken.

Testbenches in the Universal Verification Methodology (UVM), to test the design on preAA level through simulation are automatically generated from the information and constraints of the functional SR defined in SysML. Furthermore constraint random verification helps to cover all possible parameters and variants of the system, but also to vary environmental conditions, to find corner cases.

These testbenches can be used throughout the whole development cycle through to the final system integration and validation.

### 3.1 UVM Testbench generation from SysML requirements

We use a simple semi-formal language to define our requirements as approaches such as [6] have shown that informal languages can be too ambiguous for our application. The ISO26262 also promotes the view that informal languages should only be used for applications with low ASIL levels such as A and B and highly recommends the use of semi-formal requirements specifications for higher safety goals such as C and D. We thus we decided to use the benefits of the UML profile SysML for the definition of the requirements. As SysML for requirements lacks in proper definition for safety, we defined an extension as depicted in Fig. 2. Besides standard attributes id and text, following attributes such as type ( functional SR, technical SR, hardware SR, software SR), status (proposed, assumed, accepted, reviewed), ASIL level, and pass/fail have been added to the definition. Attributes such as id, text, status and ASIL level are also recommended by the ISO26262 standard. Each safety goal in our approach is therefore clearly defined by our extension for safety requirements. As mentioned in

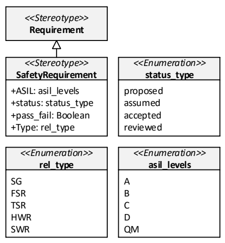
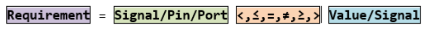
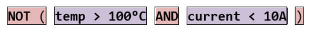


**Fig. 2.** An extension to the SysML profile to cope with safety requirements and to achieve traceability

the previous chapter, the top level safety requirements (Safety Goals) are derived from the hazard and risk analysis. These safety goals lead to the definition of the functional safety concept. Here the functional SR are derived from the safety goals in conjunction with the preAA. At least one functional SR shall be specified for each safety goal, but also one functional SR can also be valid for several safety goals. Each functional SR is described by the defined attributes in our extension for safety requirements. Furthermore each functional SR in our approach has several defined constraints for functional and non-functional properties. These constraints are defined in the MARTE value specification language (VSL) and specify the boundaries for a fail-safe operation of the system. These constraint precisely captures the original requirement and opening up, through computer readable formalism, the possibility of subsequent computer-aided analysis of the characteristics of this design. The MARTE *nfpConstraint* are defined by arithmetic; logical or time expressions formed by combining operators such as ('<','≤','=','≠','≥','>') but also 'and', 'or' and 'xor'. The syntax used for our constraints follows the following patterns:



Multiple constraints can be connected via simple Boolean statements such as:



*"The current shall not exceed 10A if the battery temperature is above 100°C"*

The technical SR can be derived after the systematic specification of the functional SR and design of the preAA with the help of our SCL. The ISO26262 specifies the technical safety requirements as following [4]:

*"The technical SR shall be specified in accordance with the functional safety concept, the preliminary architectural assumptions of the item and the following system properties: the external interfaces, such as communication and user interface; the constraints, e.g. environmental conditions or functional constraints; and the system configuration requirements. The ability to reconfigure a system for alternative applications is a strategy to reuse existing systems."*
*"Safety Mechanisms: The technical safety requirements shall specify the response of the system or elements to stimuli that affect the achievement of safety goals. This includes failures and relevant combinations of stimuli in combination with each relevant operating mode and defined system state."*
*"The system design shall be verified for compliance and completeness with regard to the technical safety concept using the verification methods e.g. Simulation for ASIL level higher than B."*

In order to support the specification of the technical SR and furthermore enable the verification in compliance with the technical safety concept, we defined a novel methodology to derive further requirements and inputs from the functional SR in coherence with the early system design (preAA). Using the syntax for safety requirements we are able to generate UVM verification components and whole testbenches from the definition of the functional SR and their constraints. For each constraint of the functional SR, a new UVM validator is added on the ports or one end of the signal. A validator consists of a configurable comparator with the pin/port/signal attached to one input and a reference signal or constant value attached to the second input. The outputs of the comparator can be either 1 (true) or 0 (zero) and are connected via arithmetic or algebraic function blocks to create the boolean operations. In addition we use non safety requirements in the SysML specification to provide stimuli blocks for relevant operating modes and driving maneuvers. Depending on the non safety requirements and constraints and if the pin/port/signal is an unused input of a block the testbench generator creates a stimuli block and attaches it. This block generates either values that are within the specifications in order to validate proper operation or to generate invalid stimuli to verify safety mechanisms within the model. To vary the parameters and stimuli of our system and to cover up corner cases we use the benefits of Coverage-Driven Verification (CDV), with its aim to detach from direct - user depended - testing [2]. This methodology provides the definition of so called verification goals, which can be verified by smart test scenarios. The intelligence is mainly achieved by creating simulation configurations (stimuli), with respect to some predefined constraints. This concept is widely known as Constraint Random Verification (CRV) [7]. CRV mainly consists of two core concepts, which is on one hand the usage of Markov-chain Monte Carlo to guarantee coverage through probability and on the other hand the processing of constraints with SAT solvers. As described above, it is important to vary parameters such that many different input combinations can be covered. The defined internal values of the DUT vary according to a predefined probability distribution. In this case we use Gaussian distribution with the definition of a value of 3 sigma.

## 4 Usecase: Battery Management System

We have applied our methodology to an industrial use case, an electric vehicle (eVehicle) system provided by CISC Semiconductor , to more fully illustrate its innovative capabilities and benefits. As more and more vehicles are now powered by Li-ion batteries, the challenge for engineers to ensure reliability and fault tolerance is also greatly increasing. It is crucial for ensuring safe operating conditions that the battery management systems (BMS) measure voltage, temperature and current of the battery very precisely. This information must be forwarded to a vehicle wide controller network to ensure a reliable and fully utilized system. Problems with overheating or even explosions have been frequent in the past. The main cause of these problems was an excessively high energy intake from regenerative braking or harsh environmental conditions. Management systems and mechanisms are thus essential to assure that persons are not put at risk and that no damage is caused. The overall system model of the eVehicle is depicted in Fig. 3. This model gives an early view of the system on preAA design level with little to no assumption about the actual hardware. It is composed of the *battery*, *controller*, *inverter*, *dc-motor* and the *battery management unit (BMU)*. The *BMU* is included in the *battery* model. The *driver* provides the desired speed for the eVehicle. This can be set according to standardized maneuvers such as the New European Drive Cycle (NEDC). The *controller* is a model for a PI state-space controller and maintains a constant speed based on the information about the state variables, motor armature current and motor-speed.

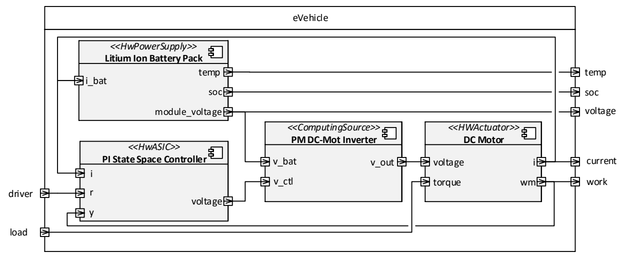


**Fig. 3.** Design under test (DUT): ports on the outside of the eVehicle class enables the connection to verification components

The *inverter* model implements an inverter function for a PM-DC motor driving stage. It compares the actual battery voltage and the requested controller voltage to maintain the PM-DC motor terminal voltage. The *battery* model simulates the behavior of a Li-ion battery pack composed of a defined set of single cell Li-ion batteries. The appropriate number of single cells is connected in parallel and series to obtain the necessary capacity, maximum current and terminal voltage. The battery pack's terminal voltage is calculated based on the defined parameter and the battery current. A BMU is connected to the battery to measure voltage, current and temperature of the cells/modules. The BMU computes the SOC, State-Of-Health (SOH) and is responsible for cell balancing, cell protection and demand management of the battery. These computed values can then processed via a CAN controller as digital values to the power train controller. In addition, the external load environmental conditions such as temperature can be changed during the simulation.

In a next step the functional SR are derived from the definition of the safety goals. An example for this would be to reuse the battery pack from a prior design which has known operating conditions and test if it is powerful enough to power the motor chosen for the new design (using a preliminary specifications provided by the manufacturer).

– The maximum operation temperature allowed for the battery cells is 100°C, therefore this temperature shall never be reached.
– Due to the choice of battery the maximum current drawn from the cells shall not exceed 10A.
– The cell/module voltage shall remain between 2.5V (empty) and 4.25V (maximum charging voltage)
– The state of charge for the individual cells shall not be lower than 10% nor higher than 110% of design capacity.

While textual or informal definition is easy to read, according to ISO26262 a semi-formal notation for requirements specifications is best qualified for ASIL levels higher than B, shown in our requirements diagram in Fig.4.

These requirements and constraints can be used to test only the battery to be included as DUT. As i_bat is modeled as an input (e.g., a current sense ADC

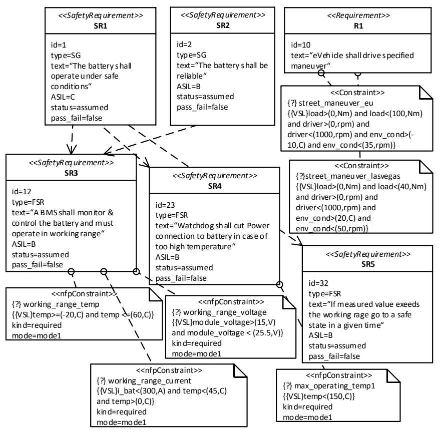

<<SafetyRequirement>>
SR1
id=1
type=SG
text="The battery shall operate under safe conditions"
ASIL=C
status=assumed
pass_fail=false
<<SafetyRequirement>>
SR2
id=2
type=SG
text="The battery shall be reliable"
ASIL=B
status=assumed
pass_fail=false
<<Requirement>>
R1
id=10
text="eVehicle shall drive specified maneuver"
<<Constraint>>
{?} street_maneuver_eu
{{VSL}load>(0,Nm) and load<(100,Nm) and driver>(0,rpm) and driver<(1000,rpm) and env_cond>(-10,C) and env_cond<(35,rpm)}
<<Constraint>>
{?}street_maneuver_lasvegas
{{VSL}load>(0,Nm) and load<(40,Nm) and driver>(0,rpm) and driver<(1000,rpm) and env_cond>(20,C) and env_cond<(50,rpm)}
<<SafetyRequirement>>
SR3
id=12
type=FSR
text="A BMS shall monitor & control the battery and must operate in working range"
ASIL=B
status=assumed
pass_fail=false
<<SafetyRequirement>>
SR4
id=23
type=FSR
text="Watchdog shall cut Power connection to battery in case of too high temperature"
ASIL=B
status=assumed
pass_fail=false
<<SafetyRequirement>>
SR5
id=32
type=FSR
text="If measured value exeeds the working rage go to a safe state in a given time"
ASIL=B
status=assumed
pass_fail=false
<<nfpConstraint>>
{?} working_range_temp
{{VSL}temp>=(-20,C) and temp <=(60,C)}
kind=required
mode=mode1
<<nfpConstraint>>
{?} working_range_voltage
{{VSL}module_voltage>(15,V) and module_voltage < (25.5,V)}
kind=required
mode=mode1
<<nfpConstraint>>
{?} working_range_current
{{VSL}i_bat<(300,A) and temp<(45,C) and temp>(0,C)}
kind=required
mode=mode1
<<nfpConstraint>>
{?} max_operating_temp1
{{VSL}temp<(150,C)}
kind=required
mode=mode1

**Fig. 4.** Definition of safety and non-safety requirements to derive automatically testbenches for verification

in the BMU) and temp as output, it would merely sweep the current from 0-10A (0 as no lower boundary was defined) and evaluate if the temperature, voltage and SoC remain within their respective bounds.

We use non-safety requirements to define driving maneuvers with an assumed load and different environmental conditions. By this means we can automatically create a testbench for the entire design as shown in Fig.5, including stimuli, validators and scoreboards. The validator verifies that an input signal does not exceed a given threshold or remains bounded between two limits. This can basically be represented as a comparator with a user configurable operation ('<','≤','=','≠','≥','>') and one or two constants. Nevertheless, these descried thresholds are not stringed constants. The constraints can also describe temporal parameters as a certain peak current may be drawn from the battery but not for a prolonged period of time. The validator components are provided by our SCL and exists in many common configurations. Each validator has a boolean out-

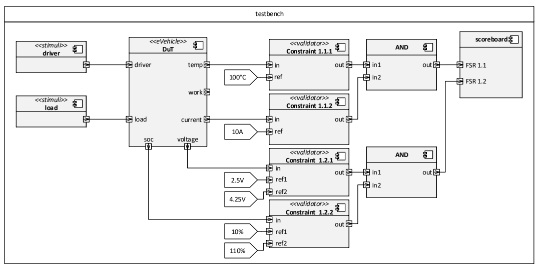


**Fig. 5.** Automatically generated testbench for the eVehicle model using UVM components derived from specification

put that indicates if the constraints have been violated by the monitored signal. The scoreboard can be configured to terminate the simulation upon violation or continue and flag the simulation accordingly. Using our tool these problematic simulations can be filtered and re-run using more output or a smaller timestep to gain more insight into the problem. Testbenches from each design step can be reused for the further steps in order to improve test coverage. This means that while the preAA will not contain exact timing information for every specific subsystem, its testbenches can still be reused in later design phases to verify that the overall system is still behaving as initially intended. This design approach incorporates elements from the test-driven, continuous integration design flows commonly used in agile software development in the sense that for each step the constraints from parent and current level serve as unit-tests. The scoreboard is used to check every commit/change for errors. This is also useful in case of refactoring e.g., if for reasons of supply problems a part/component has to be replaced rather late in the design.

As mentioned previously stimuli are required in order to correctly evaluate the overall design (integration testing) and not only individual components (unit testing). While it would be possible to automatically generate the stimuli for the overall systems from the constraints (e.g., linear search of the entire value space for an input in correlation with each other input trying to find corner cases that best test the design) most of them would not represent any realistic environment. For this reason we decided to use non-safety requirements and derive stimuli from these. Using the eVehicle as an example this could be a standardized driving maneuver using a number of predefined locations for environment parameters (e.g., ambient temperature and humidity). It is also important to test if the designed safety mechanisms and safe states operate as designed. For this reason we could either define stimuli that provoke the triggering of a mechanism (e.g., driving at full speed for a prolonged period of time under high ambient temperature to test if the system can prevent overheating) or due fault injection. To the terms of software development this would represent a form of mutant testing where a deliberate fault is simulated in order to verify that a safe state can be reached. This is especially useful if existing designs are reused or a fault tree is given by the vendor to define the stimuli. The traces of our simulation-based verification within UML/MARTE are depicted in Fig. 6. This shows the analog signals such as module_voltage, driver, module_current, load, celltemperature and work, which are monitored by our UVM components. The DUT was stimulated by a driver with the *street_maneuver_eu* driving scenarios. Only one run, with a specific configuration, is shown in this figure. As the number of the simulation tasks for different parameter configurations can be relatively high and are independent of each other, we use a cloud-based solution for UVM [14] in order to parallelize our simulations to a very significant extent and gain a virtually linear acceleration. This provides a flexible way to allocate several worker instances to speed-up the time needed to simulate thousands of tasks.

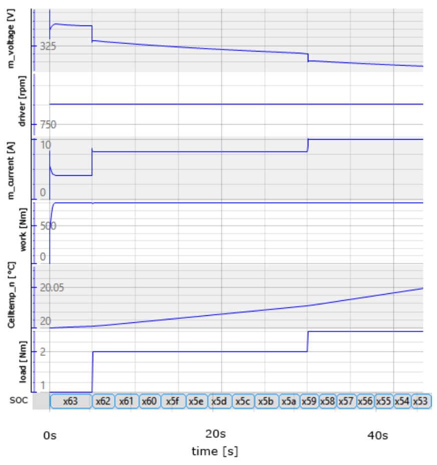


**Fig. 6.** This figure presents traces from one run of our seamless simulation-based verification methodology, e.g. temperature, voltage, current and SOC of the battery

## 5 Conclusions

In this paper we presented a simulation-based verification methodology tightly and seamlessly integrated in the safety-lifecycle (v-model) of the functional safety standard ISO26262. Our tool-aided methodology can be used from early system design, throughout the entire safety-lifecycle through to system integration and validation. Since millions of testkilometers now need to be managed, our simulation-based and constraint random approach helps to cover up a high percentage of possibilities. From an early safety analysis in conjunction with the early system design, testbenches have been automatically generated to test the preliminary architectural design. From this early analysis further technical but also hardware and software requirements have been derived. Furthermore, our approach provided important inputs for the more detailed system design. These testbenches have been generated from the requirements and constraints defined in the semi-formal SysML/ MARTE format with our extension for safety requirements and can be used throughout the entire safety-lifecycle. We used standardized UVM components and the benefits of constraint random verification to provide different stimuli and configurations to find corner cases in our system. To randomly stimulate our UML/ MARTE design models, these models have been linked to fast-executable analog, digital but also mechanical implementation models in SystemC (-AMS). This framework was tested by a complex example from the automotive industry in order to demonstrate its efficiency. The use cases showed how to define constraints in the MARTE constraint language and to generate verification components to automatically test the current preliminary design. In addition our approach was developed as a plugin in Eclipse, with the result that every Papyrus UML editor is now capable of simulation simply by installing our plugin. This tool will be published for download and is also to be used for educational purposes. Further work will include the definition of safe-states and timing behavior and the generating of testbenches from it. Furthermore, sequence diagrams will be used for the generation of the test stimuli.

## References

1. CISC Semiconductor GmbH, `https://www.cisc.at/`
2. Accellera: Universal Verification Methodology (UVM) 1.2 User's Guide. Tech. rep., Accellera (may 2015)
3. Catrene: OpenES CATRENE Project: CA703 (2016), `http://www.ecsi.org/openes`
4. ISO: Functional Safety ISO26262 - Part 4: Product development at the system level 2011, 1–35 (2011)
5. Kim, H., Wong, W.E., Debroy, V., Bae, D.: Bridging the Gap between Fault Trees and UML State Machine Diagrams for Safety Analysis. 2010 Asia Pacific Software Engineering Conference pp. 196–205 (2010)
6. Kirchsteiger, C.M., Grinschgl, J., Trummer, C., Steger, C., Weiß, R., Pistauer, M.: Automatic test generation from semi-formal specifications for functional verification of system-on-chip designs. 2008 IEEE International Systems Conference Proceedings, SysCon 2008 pp. 421–428 (2008)
7. Kitchen, N., Kuehlmann, A.: Stimulus Generation for Constrained Random Simulation. In: Proceedings of the 2007 IEEE/ACM International Conference on Computer- aided Design. pp. 258–265. Piscataway, NJ, USA (nov 2007)
8. Mader, R., Armengaud, E., Leitner, A., Kreiner, C., Bourrouilh, Q., Grießnig, G., Steger, C., Weiß, R.: Computer Safety, Reliability, and Security: 30th International Conference,SAFECOMP 2011, Naples, Italy, September 19-22, 2011. Proceedings. chap. Computer-A, pp. 113–127. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
9. Marinescu, R., Kaijser, H., Mikucionis, M., David, A., Seceleanu, C., Henrik, L.: Analyzing Idustrial Architectural Models by Simulation and Model-Checking. Formal Techniques for Safety-Critical Systems 419, 189–205 (2014)
10. Mhenni, F., Nguyen, N.: Automatic Fault Tree Generation From SysML System Models. 2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), Besancon, France (2014)
11. Taha, S., Radermacher, A., Gérard, S.: An Entirely Model-Based Framework for Hardware Design and Simulation. International Federation for Information Processing (IFIP) pp. 31–42 (2010)
12. Weissnegger, R., Kreiner, C., Pistauer, M., Römer, K., Steger, C.: A Novel Design Method for Automotive Safety-Critical Systems based on UML/MARTE. In: Proceedings of the 2015 Forum on specification & Design Languages. pp. 177–184. Barcelona, Spain (2015)
13. Weissnegger, R., Schuss, M., Kreiner, C., Pistauer, M., Römer, K., Steger, C.: Simulation-based Verification of Automotive Safety-critical Systems Based on EAST-ADL. Procedia Computer Science 83, 245–252 (2016)
14. Weissnegger, R., Schuß, M., Schachner, M., Pistauer, M., Römer, K., Steger, C.: A Novel Simulation-based Verification Pattern for Parallel Executions in the Cloud. In: 21st European Conference on Pattern Languages of Programs Proceedings (2016)

# SAVESOC - SAFETY AWARE VIRTUAL PROTOTYPE GENERATION AND EVALUATION OF A SYSTEM ON CHIP

Ralph Weissnegger
Markus Pistauer
CISC Semiconductor GmbH
Klagenfurt, Austria
{r.weissnegger, m.pistauer}@cisc.at

Martin Schachner
Christian Kreiner
Kay Römer
Christian Steger
Institute for Technical Informatics
Graz University of Technology (TU Graz), Austria
{schachner, christian.kreiner, roemer, steger}@tugraz.at

## ABSTRACT

The electrification of today's vehicles and the high amount of new assistance features imply more and more complex systems. The sensing and controlling of these systems is the work of the highly distributed and connected electronic control units. To keep pace with the fast growing automotive market, reusability of components and features is today the key to reduce costs and time-to-market. Especially when systems are safety-critical and demand reliability, new methods and tools are thus essential to support the reusability aspect in the development process. A model-based approach, in conjunction, moreover helps to communicate between different stakeholders, provides different views and serves as a central storage of information. Through applying reliability analysis and simulation-based verification methods on our hardware model and furthermore automatic generation of a first virtual prototype, we are able to reduce the tools involved, thus resulting in correctness, completeness and consistency of the entire system.

**Keywords:** UML, functional safety, ISO26262, cyber-physical system, virtual prototyping.

## 1 INTRODUCTION

In the world of today, the amount of embedded electrical/electronic (E/E) systems in various domains is highly increasing to a very great extent. When we think about the complexity of the past few years, it is apparent that new applications have emerged in which systems are not only interacting with each other but also have impact on the physical world, the so-called cyber-physical systems. Depending on their application, they must fulfill different requirements ranging from timing constraints, performance behavior, low power consumption, thermal or even working capability under different environmental conditions. The point here is, we live in a world where cyber-physical systems are ubiquitous, they have impact on our daily lives and the malfunction of these systems can lead to severe damage or injury to people. We must thus assure the dependability of these systems.

This is even more obvious when we turn to the automotive domain. It can be observed that there is a shift towards fully E/E systems resulting from the trend to electric vehicles.The sensing and controlling is the work of the highly distributed electronic control units (ECU) and it is no surprise, that through all these new features in cars, more than 100 of these microcontrollers (Charette 2009) are currently integrated in a modern car. This situation has also an impact on the amount of software in cars today, which can total 150 million lines of code (Eitdigital 2016). The industry is facing new problems through the emergence of many new (assistance-) features that are also influencing each other. In turn, this raises the complexity level in the design, development and verification of complex systems and imposes an enormous effort for engineers in developing their applications. In terms of safety, these systems must fulfill standards such as the ISO26262 (functional safety for road vehicles), (ISO 26262 2011). Since this standard is now treated as state of the art in court, OEMs and their suppliers are required to develop and test their systems towards the recommended measures and methods.

When we discuss the design of a system, a single modeling language immediately springs mind, the Unified Modeling Language (UML), (Group 2015). Having the roots in the software domain, UML paved the way and established a model-based thinking in various engineering domains, far across the borders of conventional software design. Since UML comes with several extensions such as MARTE (OMG 2016), SysML (Omg 2015) or EAST-ADL (EAST-ADL 2017), furthermore engineers from different domains can use the full potential of an object-oriented approach. MARTE was introduced to overcome the enormous complexity issues in the design of real-time and embedded systems. It provides capabilities to model hardware, software as also as system design.

Since a state of the art car of today exists not only in one single version, but rather in several hundreds of variants all with different features, each of these must be exhaustively tested to fulfill the standards. Millions of test kilometers must be driven to ensure the reliability of a car and it is neither economic nor safe to test them in a real environment (Maurer, Gerdes, Lenz, and Winner 2015). Simulation plays an ever increasing and important role in the verification of the modern car because of its advantage in easily varying the virtual environment and to representing the car in different variations, not least from an economical perspective. Simulations can be done in early development phases, where the detailed implementation of a function is still undecided and furthermore, on platform specific models where the hardware and software are explicitly defined (virtual prototype). Applied verification methods and tests can be monitored, reproduced and rerun every time. Another advantage of simulation is that it cannot only be run day and night, but also massively in parallel. A specification and simulation language, which shares the same philosophy as the UML/MARTE approach, is SystemC (Accellera Systems Initiative 2017). Like UML, it shares the MDA (Model Driven Architecture) approach, starting from a computational independent system design, down to hardware and software design.

The ISO26262 recommends different verification methodologies used for the hardware platform use. These includes design walk through, FTA/ FMEA, hardware architectural metrics evaluation but more importantly, hardware prototyping and simulation for higher ASIL levels. These simulations, including virtual prototyping, can then be used for further hardware verification methods such as fault injection test, which is currently the key for testing the reliability of the hardware. Several research institutes are now working on executing fault injection, also on higher abstraction level such as transaction level modeling (TLM). This has the advantage that this method can be applied on faster simulation models without losing information from the more detailed models (RTL, register transfer level). Virtual prototyping has the benefit that embedded software can be tested much earlier, before a first real hardware prototype is available. Changes on a virtual hardware design are much faster than changes on the real platform, which takes weeks or months of redesign and production, which in turn has impact on time to market. With intensive simulation, corner cases but also long term reliability errors can be encountered, which also prevents costly product recalls. Environmental impacts on the virtual prototype can be simulated and reproduced, where real testbeds are not capable of this kind of verification. The drawback, a complex virtual prototype (VP) is not developed overnight. It takes a lot of effort, experienced designers and engineers to build a so called digital twin of the actual hardware. Our proposal is thus to reuse models for virtual hardware prototyping from open libraries such as Open Virtual Platform (OVP), (OVP 2017).

In this work we present a novel design and development flow for a safety aware virtual prototype. This design flow is conform to the ISO26262 standard and meets all its requirements to produce a reliable product in the end. The whole system, from a first functional specification down to hardware design, is specified by standardized modeling languages. Before the virtual prototype is generated from the hardware specification, several reliability analysis methods are applied and executed on this specification. This allows an early evaluation of the hardware design regarding safety, before testing the prototype in simulation. Furthermore, we show the integration of the generated virtual prototype into the system design, to verify the interfaces and its functionality. The whole methodology is available through the developed design and verification framework named SHARC (Simulation and Verification of Hierarchical Embedded Microelectronic Systems), (CISC 2017).

## 2 RELATED WORK

The authors of (Macher, Stolz, Armengaud, and Kreiner 2015) aimed at achieving consistency of information between several tools involved in the development process, through to a single source of information principle. In the goal of achieving dependability (safety, security) in the development process between different teams and stakeholders, they decided against a document-centric approach and used the capabilities of UML and SysML for their design. This in turn improved consistency, correctness, and completeness of the entire system under development. The toolchain in this approach focused more on the system and software development and did not take hardware development into account. The authors also propose to update their profile in order to work efficiently on hardware development as such.

To overcome the issues with consistency within design and simulation models, the authors of (Sporer, Macher, Armengaud, and Kreiner 2015) proposed a model transformation framework between SysML and Matlab/Simulink. They support a consistent and traceable refinement from the early concept phase through to software implementation and this in a bidirectional manner. The authors also claim that a model-based design helps to enable different views for different stakeholders, different levels of abstraction, and central storage of information. Nevertheless, the author's focus was more on the software architecture generation from system design rather than on the requirements for hardware design.

Popular approaches (Adler, Domis, Höfig, Kemmann, Kuhn, Schwinn, and Trapp 2011) and (David, Idasiak, and Kratz 2009) have shown that UML as modeling language can be efficiently used with analysis and verification methods such as FMEA (failure mode and effect analysis), fault tree analysis (FTA), design walk through (Gvero, 2013), code-generation and many more. The drawback of UML, in terms of simulation to verify the system behavior is, that code-generation can only be done at a very late stage or even at the end of the design process, when all details are very well known. Later changes in design are costly and result in inconsistent models and furthermore reverse-engineering is an error prone and cumbersome task. The majority of components in new projects are reused and simply extended by the addition of new features to reduce costs and time-to market. The reuse of whole safety concepts, well-trusted designs and mechanisms is thus becoming more important to reduce the effort required for developing complex systems. This situation prompts the urgent demand for new techniques to simulate the behavior in early development-phases by reusing verified system components.

## 3 USE CASE

Throughout this paper we will demonstrate our methodologies on a relevant problem in today's automotive domain, a battery management system for Li-ion powered electrical vehicles. This industrial use case was provided by CISC Semiconductor GmbH, based in Austria and the United States. This use case will help to illustrate more fully the innovative capabilities and benefits of our approach. As more and more vehicles are now powered by Li-ion batteries, the challenge for engineers to ensure reliability and fault tolerance is also greatly increasing. It is crucial for ensuring safe operating conditions of a battery that monitoring systems such as the battery management systems (BMS) measure the voltage, temperature and current of the battery very precisely. This information must be forwarded to a vehicle-wide controller network to ensure a reliable and fully utilized system. Problems with overheating or even explosions have been frequent in the past. The main cause of these problems was an excessively high energy intake from regenerative braking or harsh environmental conditions. Management systems and mechanisms are thus essential to assure that persons are not put at risk and that no damage is caused.

To achieve a first executable specification of our use case, we used the methodologies provided in (Weissnegger, Schuß, Kreiner, Pistauer, Kay, and Steger 2016), to connect the UML/MARTE design models with reusable simulation models (model library) in SystemC (-TLM)/ SystemC-AMS. These functional models are early executable models on a high level of abstraction and can be refined, depending on their purpose, through to more detailed models. The provided model library contains analog and digital models to gain an early and fast evaluation of the functional specification on system level. With this approach moreover, we eliminate the tedious task of exporting our gathered data to other simulation tools such as Matlab/Simulink and leave the UML/MARTE design as a single source of information. Furthermore, we rely on standardized and open modeling and simulation languages and keep the costs for licenses low.

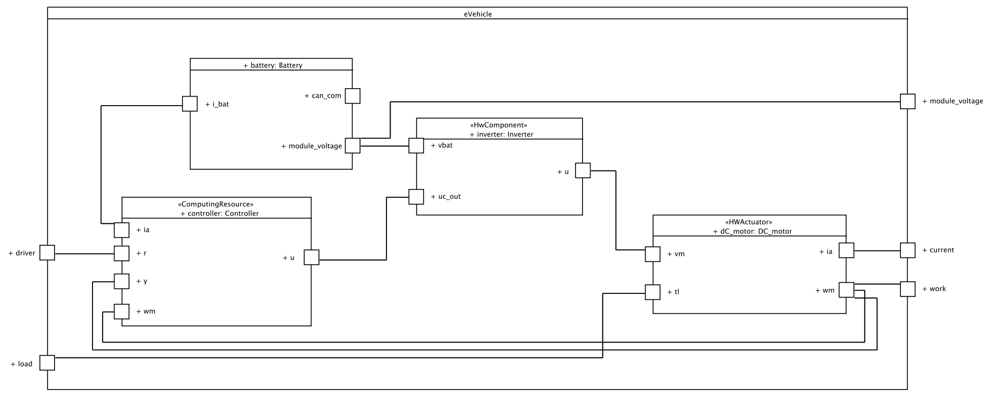


Figure 1: Overall system level description of the electric vehicle use case with UML/MARTE.

The overall system of the eVehicle is depicted in Figure 1. For reasons of simplification, we only consider the major components of the electric vehicle, for the analysis of the battery and the BMS. This includes the battery pack in Li-ion technology, the BMS which measures voltage and temperature of the battery, an inverter ECU, a controller and the electric motor model. Two main factors influence the behavior of the eVehicle. The driver provides the desired speed (rounds per minutes) and on the other hand the load on the motor shaft. These stimuli can be set according to standardized maneuvers such as the New European Drive Cycle (NEDC), or the newer standards known as the worldwide harmonized light-duty vehicles test procedure (WLTP), which will be introduced in 2017.

## 4 SPECIFICATION OF THE HARDWARE PLATFORM

From the gathered information of our functional and executable specification we now delve deeper into the hardware design. Since MARTE and SystemC share the same philosophy to move from system to hardware and software development, we are now able to specify, through refinement, the internal architecture of our BMS platform. As the major goal of this work is to build a ISO26262 safety aware platform, this is an important step in the development process and takes a lot of effort into account (Kreiner 2015), since many different methods and measures have to be applied to the hardware platform to guarantee the final result of a reliable and safe product.

Since the OVP virtual prototype consists more or less of a set of SystemC -(TLM) files including a TCL script, one goal of this work is to include the whole generation of the virtual prototype into a seamless design flow for safety critical systems. Furthermore, the design and configuration of the VP on a graphical modeling standard, in this case UML/MARTE. This approach brings the advantage of being able to evaluate and analyze the configuration of the hardware design before the actual prototype is generated from UML models. OVP itself does not support verification regarding safety, nor is OVP till now embedded in a seamless design flow.

As mentioned in the previous chapter, UML/MARTE provides several levels of detail for the specification of the hardware platform. The hardware resource model (HRM) package provides several models to describe subsystems such as HwProcessor, HwBus, HwDevice or HwMemory in a logical and physical way. Although, UML/MARTE provide a rich set of different stereotypes to describe the hardware platform, it does not provide the level of detail for the hardware configuration as expected by the OVP methodology. Several mandatory properties such as the BusInterface or Memory mapped or the VLNV (vendor, library, name, version) principle are by now not supported in the MARTE standard.

To overcome these issues, we rely on the specification of the IP-XACT standard, which helps us to define our platform with a sufficiently high degree of detail for the generation of the virtual prototype. Both, IP-XACT and OVP rely on the VLNV principle for structuring the models. We thus built on approaches such as (André, Mallet, Khan, and de Simone 2008) to extend the MARTE standard by properties in IP-XACT for hardware description.

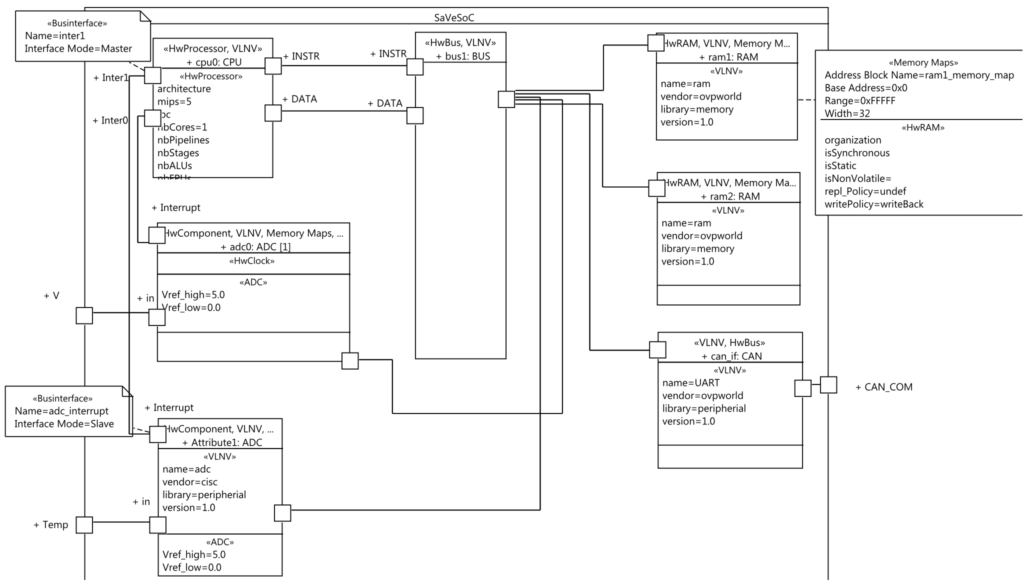


Figure 2: Hardware platform of SaVeSoC including IP-XACT extensions for MARTE.

Figure 2 depicts the composed structural architecture model of the platform consisting of a processor, bus, memory, CAN and two ADCs. For simplification we only show the major components of the design of the battery management system. The designer can easily compose his system by using the standard models from the MARTE library such as HwProcessor for the CPU, HwI_O or HwComponent for peripherals, HwRam for memory or HwBus for the internal bus or CAN bus. Depending on the nature of the component, the designer is able to extend the component with specific hardware properties in the IP-XACT standard such as MemoryMaps and BusInterface. The properties from the IP-XACT standard are additionally shown in the description of the MARTE model. Further extensions for IP-XACT such as VLNV can be added to the hardware components as well.

## 5 GENERATION AND EVALUATION OF THE VIRTUAL PROTOTYPE

### 5.1 Evaluation of the Hardware Configuration

Based on the specification of our hardware platform in Fig. 2 we are able to perform different evaluation methods on our hardware design. As mentioned in the previous chapters, there are several methods which can be used to evaluate the reliability and safety of our hardware architecture, one of these is the hardware architectural metrics evaluation. The hardware architectural metrics are handled in Part 5 (ISO 26262 2011) of the ISO26262, product development at the hardware level. This evaluates the hardware architecture of the item against the requirements for fault handling. This part also includes guidance on avoiding systematic and random hardware failures by means of appropriate safety mechanisms. Each safety-related hardware element is analyzed regarding single point (SPFM), residual and multiple point faults (LFM). It also describes the effectiveness of the hardware architecture in coping with random hardware failures (PMHF). Each hardware part is to be protected by means of safety-mechanisms. The diagnostic coverage gives evidence of the effectiveness of these mechanisms. Whether the item (system or array of systems according to ISO26262) passes or fails a given ASIL is also a result of the hardware architectural metrics evaluation. In order to achieve a specific ASIL, the values from Table 1 must be met (FIT- failure in time).

Table 1: Architectural Metrics - evaluates whether the hardware achieves a certain ASIL, according to ISO26262.

| | ASIL B | ASIL C | ASIL D |
|---|---|---|---|
| SPFM | $\geq 90\%$ | $\geq 97\%$ | $\geq 99\%$ |
| LFM | $\geq 60\%$ | $\geq 80\%$ | $\geq 90\%$ |
| PMHF | $< 10^{-7}h^{-1}$ | $< 10^{-7}h^{-1}$ | $< 10^{-8}h^{-1}$ |

To perform this evaluation on our hardware specification, we built on approaches such as (Weissnegger, Pistauer, Kreiner, Römer, and Steger 2015), (Weissnegger, Pistauer, Kreiner, Kay, and Steger 2015) and (Das and Taylor 2016). These approaches use the capabilities of UML/MARTE and IP-XACT to perform several quantified methods to evaluate the reliability of the hardware platform. Furthermore, they defined an extension in IP-XACT for fault models, which can be reused for different hardware platform configurations. These reuse strategies are commonly used in industry and are known as clone and own. The existing and reused safety models can give important feedback through an early safety assessment for a modified platform or even a new product. Changes and adaption on the overall platform must be taken into account, of course when reusing safety artifacts during development. Since the main contribution of this paper is the generation and integration of the virtual prototype into a seamless design and development flow, we do not go into detail on the hardware evaluation. A more detailed information is given in (Weissnegger, Pistauer, Kreiner, Römer, and Steger 2015).

## 5.2 Generation of the Virtual Prototype

One of the outlined goals in this work is to develop and test embedded software within the development phases, on a realistic hardware prototype of the target system, before the actual platform is available. Embedded software is often written in a desktop environment, on a general purpose operating system of the host system. This approach often differs often significantly from the target platform and parts of the written software need to be adjusted. One way to deal with this problem is the usage of an instruction set simulator (ISS) and hardware visualization. Due to a vendor variety of components within SoC design, the simulation can be very difficult. Hardware emulators are very popular for this issue, but require the detailed RTL description of the developed system, which is a contradiction to the outlined goal of a homogeneous development environment. OVP makes it possible to create virtual platform models, with SystemC TLM 2.0 support. OVP models can be executed much faster than their counterparts developed in RTL, since their level of abstraction is higher but still appropriate for modeling purposes.

Relying on the ideas of model driven engineering the virtual hardware prototypes is modeled in a graphical way. After a comprehensive hardware safety analysis, the developed system design is moreover generated and integrated into the existing modeling framework. Therefore a methodology was defined, which converts the according UML platform description into a proprietary TCL file, containing the necessary information to generate source code with OVPs iGen tool. For the graphical platform description we rely on the capabilities of UML/MARTE. Additional properties, which exceeded MARTEs capabilities got defined by additional IP-XACT stereotypes. Both TCL and IP-XACT are relying on the VLNV principle. Compiled to a shared object the virtual platform can be simulated along with other components from the provided library. This approach guarantees a seamless integration of platforms into the existing simulation framework and enable the co-simulation of a virtual hardware platform together with a model of the physical environment in which it is embedded. The OVP iGen converter takes the information from the TCL script and generates a full SystemC TLM2.0 platform using the modular components of the OVP library. The resulting virtual prototype can then be used for further hardware simulations, such as fault-injections on TLM level.

## 5.3 Integration and Verification of the Virtual Prototype

Since our whole methodology (from functional specification through to hardware and software design) relies on the same modeling language and furthermore the same hardware description language respectively system-modeling language, we are easily able to reapply our generated safety aware virtual prototype into the functional specification of the system design. After generation, the hardware description of the SaVeSoC platform with the whole interface specification is added to the UML model library and can be reapplied to the system design including the generated simulation files in SystemC. This saves time in terms of integration effort, when testing the virtual prototype on the functionality of the whole system, as recommended in ISO26262. System-level testbenches can also be reused for the verification of the entire system including the integrated VP. The whole process is depicted in Figure 3. By adding a smaller V-model to the traditional approach, we are closing the technological and organizational gap between system design and hardware development which exists in today's tool flows. Since our design and simulation languages in use, share a seamless development flow, no information transfer is needed between those design levels. Changing requirements in the specification can also be easily and efficient tested for the virtual prototype.

Figure 4 depicts the resulting system level description including the battery and new defined BMS hardware. The battery component is no longer a black box where the functionality is described in SystemC. It is now a white box, where the detailed hardware of the battery component is specified. It consists of the SaVeSoC element, which is the hardware platform of the BMS including an application for plausibility checks. It measures the voltage and temperature of the batterypack over two ADC and computes the state of charge

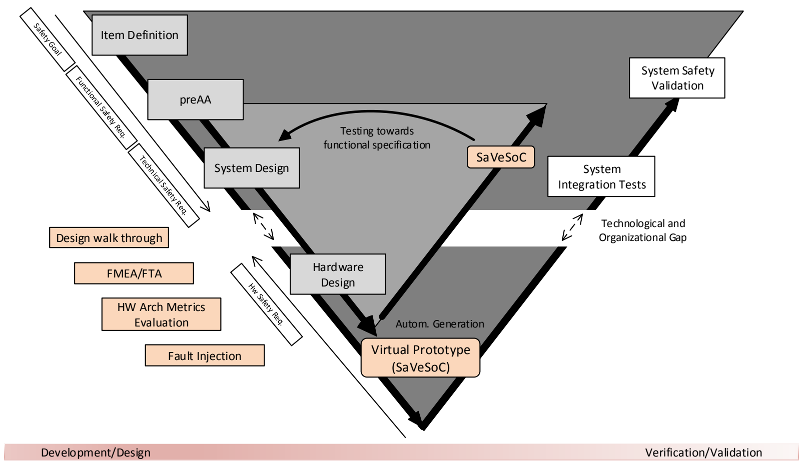


Figure 3: Process of SaVeSoC integration into functional specification.

and stage of health. The interfaces of the battery model remained the same, since no changes have been made to the interface description. Since we are relying on the same simulation engine, the virtual prototype can now be easily tested on a higher level environment, which also speeds up the overall simulation time. A challenge when integrating the VP to the functional specification was the communication interface between the functional simulation environment of SystemC-AMS and the OVP platform, because these platforms are mainly used to process data measured from embedded sensors. We thus implemented communication channels which guarantee data exchanges between different SystemC dialects. The authors of (Lonardi and Pravadelli 2015) rely on the idea that the simulation engine is executed in a single process, with the SystemC and OVP simulator running in different software threads. The authors mainly focused on the capability to co-simulate SystemC RTL models, with either QEMU or OVP. To avoid overheads from the use of sockets, they established the communication channel via a shared memory and synchronization mechanism. Furthermore they developed a SystemC bridge to enable the connections to the external hardware simulator. Since the original component library is not meant to be cycle accurate, the main focus was set to establish the communication between the existing SystemC-AMS components and the TLM2.0 models provided in the OVP. The paper (Damm, Grimm, Haas, Herrholz, and Nebel 2008) describes the main properties of both sides and how synchronization is performed internally. SystemC AMS provides so-called converter ports to establish a connection between timed data flow (TDF) modules and an ordinary SystemC signal. In the event of an access to such a port, the AMS kernel triggers an interrupt, which causes a context switch to the SystemC/OVP simulator. The crucial part of the implementation was therefore the conversion from SystemC-AMS linear signal flow (LSF) to TLM and how to handle the data stream of an arbitrary LSF module to the ADC peripheral of the OVP platform.

Figure 5 shows the used components for converting the initial LSF signal to a TLM signal. The sca_lsf::sca_-tdf_sink is a component of the SystemC AMS library, used to sample arbitrary input data and convert it to TDF. The self-defined adc_module processes the TDF signal so that it is written to the Adin port of the adc0 within its processing() procedure. The adc0 is part of the OVP library, and was adapted slightly to meet our needs. The port of the ADC is implemented with a call back function, which triggers the conversion of the ADC. Afterwards it can be read with the implemented driver, executed on the CPU. Since the OVP module requires the usage of a certail tlm_signal_port as TLM target socket for the communication, we adapted and advanced the approach of (Damm, Grimm, Haas, Herrholz, and Nebel 2008) to meet our needs.

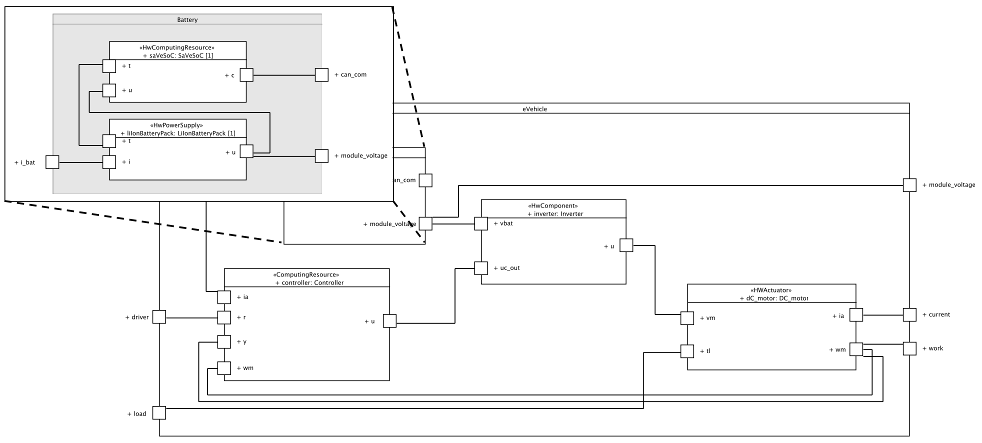


Figure 4: Virtual Prototype integration into the functional specification of the system design to verify the functionality.

Secondly, we omitted the suggested internal FIFO regarding data loss. This can be reasoned by reference to the constant sampling rate of the AMS simulation. A fixed size FIFO would nevertheless lead to data loss if the processor does not execute sufficient instructions per second.

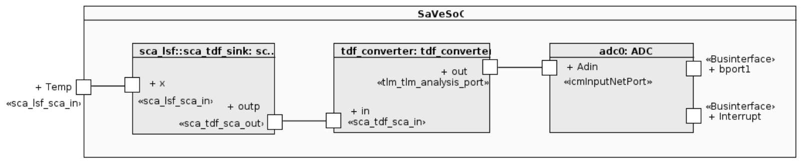


Figure 5: Communication channel for the interaction between SystemC AMS and OVP TLM2.0.

## 6 CONCLUSION

In this paper we presented a seamless design and verification process for safety-critical systems. A standardized modeling language based on UML was used to represent the design flow, from functional specification down to hardware and software. This model-based approach eases the communication between different stakeholders involved in the development process and serves as a single-source of information. Through tight integration of recommended safety analysis methods such as FTA, FMEA, hardware architectural metrics and simulation-based verification, we achieved consistency, correctness and completeness throughout the development process. The hardware architecture was evaluated by extensions to a well-known hardware description in the industry, IP-XACT. Existing and reusable hardware description was used for system design and integration. Our tool-aided method helped to speed up the evaluation process, and to reduce costs through reusability. The evaluated hardware description was then used to automatically generate a safety aware hardware virtual prototype, which was used to test correctness regarding the functional specification. This closes the technological and organizational gap in today's toolchain of safety-critical system development. Furthermore, this early virtual prototype can be used for fault-injection tests, as recommended by the functional safety standard. In addition our approach was developed as a plugin for the Eclipse, with the result that every Papyrus UML editor can be used for safety aware development of cyber-physical systems, simply by adding our plugin. This tool is named SHARC (Simulation and verification of HierARChical embedded microelectronic systems) is to be published for download and is also used for educational purposes. Further work will include the automatic generation of TLM fault-injection tests for the generated virtual prototype.

**ACKNOWLEDGMENTS**

A part of the work has been performed in the project eRamp (Grant Agreement N621270), co-funded by grants from Austria, Germany, Slovakia and the ENIAC Joint Undertaking. Some parts have also been experimented in the OpenES CATRENE Project: CA703 - 2013 research program supported by the FFG (Austrian Research Promotion Agency), project-number 843380 in tight cooperation with CISC Semiconductor.

## REFERENCES

Accellera Systems Initiative 2017. “SystemC”. http://www.accellera.org/. Accessed February, 2017.

Adler, R., D. Domis, K. Höfig, S. Kemmann, T. Kuhn, J. P. Schwinn, and M. Trapp. 2011. “Integration of component fault trees into the UML”. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* vol. 6627 LNCS, pp. 312–327.

André, C., F. Mallet, A. M. Khan, and R. de Simone. 2008. “Modeling SPIRIT IP-XACT with UML MARTE”. *Proc. DATE Workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile*.

Charette, Robert N. 2009. “This Car Runs on Code - IEEE Spectrum”. http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code.

CISC 2017. “CISC Semiconductor GmbH”. https://www.cisc.at/. Accessed February, 2017.

Damm, M., C. Grimm, J. Haas, A. Herrholz, and W. Nebel. 2008, sep. “Connecting SystemC-AMS models with OSCI TLM 2.0 models using temporal decoupling”. In *2008 Forum on Specification, Verification and Design Languages*, pp. 25–30.

Das, N., and W. Taylor. 2016. “Quantified fault tree techniques for calculating hardware fault metrics according to ISO 26262”. In *2016 IEEE Symposium on Product Compliance Engineering (ISPCE)*, pp. 1–8.

David, P., V. Idasiak, and F. Kratz. 2009. “Towards a better interaction between design and dependability analysis: FMEA derived from UML/SysML models”. *Safety, Reliability and Risk Analysis: Theory, Methods and Applications - Proceedings of the Joint ESREL and SRA-Europe Conference* vol. 3 (August 2015), pp. 2259–2266.

EAST-ADL 2017. “EAST-ADL Association”. http://www.east-adl.info/.

Eitdigital 2016. “FORD GT - Lines of code”. https://www.eitdigital.eu/news-events/blog/article/guess-what-requires-150-million-lines-of-code/.

Group, O. M. 2015. “OMG Unified Modeling Language TM ( OMG UML ), Superstructure v.2.5”. *InformatikSpektrum* vol. 21 (May), pp. 758.

ISO 26262 2011. “Road vehicles-functional safety-Part 5: Product development at the hardware level”.

Kreiner, C. 2015. "Trident Architectural Views: A Pattern for Dependable Systems Design". In *Proceedings of the 20th European Conference on Pattern Languages of Programs*, EuroPLoP '15, pp. 18:1—-18:9. New York, NY, USA, ACM.

Lonardi, A., and G. Pravadelli. 2015. *On the Co-simulation of SystemC with QEMU and OVP Virtual Platforms*, pp. 110–128. Cham, Springer International Publishing.

Macher, G., M. Stolz, E. Armengaud, and C. Kreiner. 2015. "Filling the gap between automotive systems, safety, and software engineering". *e & i Elektrotechnik und Informationstechnik* vol. 132 (3), pp. 142–148.

Maurer, M., J. C. Gerdes, B. Lenz, and H. Winner. 2015. *Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte*. Springer Open. Springer Berlin Heidelberg.

Omg 2015. "OMG Systems Modeling Language ( OMG SysML ™ ) v.1.4". *Source* (June), pp. 260.

OMG 2016. "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems". Technical report, Object Management Group.

OVP 2017. "Open Virtual Platform". http://www.ovpworld.org.

Sporer, H., G. Macher, E. Armengaud, and C. Kreiner. 2015. "Incorporation of Model-Based System and Software Development Environments". *Proceedings - 41st Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2015*, pp. 177–180.

Weissnegger, R., M. Pistauer, C. Kreiner, R. Kay, and C. Steger. 2015. "A Novel Method for Fast Evaluation of Cyber-Physical Systems in Compliance with Functional Safety". In *Euromicro Conference in Software Engineering and Advanced Applications - Proceedings of the Work in Progress Session*, pp. 24–25. Funchal, Madeira, Johannes Kepler University Linz.

Weissnegger, R., M. Pistauer, C. Kreiner, K. Römer, and C. Steger. 2015. "A novel method to speed-up the evaluation of cyber-physical systems (ISO 26262)". In *2015 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES)*, pp. 109–114. Ancona, Italy.

Weissnegger, R., M. Schuß, C. Kreiner, M. Pistauer, R. Kay, and C. Steger. 2016. "Bringing UML / MARTE to life : Simulation-based Verification of Safety-Critical Systems". In *2016 Forum on Specification and Design Languages (FDL)*. Bremen, Germany.

## AUTHOR BIOGRAPHIES

**RALPH WEISSNEGGER** received his Bachelor's and Master's degree in Telematics (Information and Computer Engineering) from Graz University of Technology, Austria, in 2013. Since 2014 he is with the Institute for Technical Informatics at Graz University of Technology where he is working towards his Ph.D in Electrical Engineering. His research interests include design and verification of HW/SW codesigns, especially safety-critical systems. His Ph.D is done in tight cooperation with CISC Semiconductor GmbH, r.weissnegger@cisc.at.

**MARKUS PISTAUER** (CEO, Member IEEE) holds a Master degree in Electrical and Electronic Engineering (1991) and a Ph.D. degree in Electronic and Control Engineering (1995), both from Graz University of Technology, Austria. From 1995 to 1999 he worked at Siemens AG (Semiconductor Division, now Infineon Technologies) and also as Professor at University of Applied Sciences, Carinthia. He has founded CISC Semiconductor in 1999 where he acts as CEO and in 2012 CISC Semiconductor Corp. in Mountain View, CA, USA. He is author and co-author of more than 70 papers published and holds several patents in the area of embedded systems.

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2016) 000–000

Procedia
Computer Science

www.elsevier.com/locate/procedia

The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017)

# SHARC - Simulation and Verification of Hierarchical Embedded Microelectronic Systems

Ralph Weissnegger[a,b,], Christian Kreiner[b], Markus Pistauer[a],
Kay Römer[b], Christian Steger[b]

[a]*CISC Semiconductor GmbH, Klagenfurt, Austria*
[b]*Graz University of Technology (TU Graz), Austria*

**Abstract**

The modern automotive market is heading towards fully automated self-driving cars. Following this evolution, the amount of new assistance features for ensuring safe and reliable operations is rising, thus the design and verification of electric/electronic systems is becoming more and more complex. Simulation-based verification is key nowadays to test the reliability of a system, since the costs for physical tests cannot be handled anymore. Current tools and design flows hit the limits of complexity and therefore are not capable to efficiently address software and hardware design and optimization in a joint way. Furthermore, the technological, organizational and design gap in today's flows are not covered by current methods and tools. To cope with the high complexity in the integration of embedded systems, the use of advanced methods and design tools is more relevant than ever. In this work, we present a design, simulation and verification framework named SHARC. This framework allows an efficient verification of safety-critical networked embedded systems regarding functional safety (ISO 26262). Moreover, we achieve to merge a simulation-based approach, including virtual prototyping, with quantified reliability analysis without losing consistency.

*Keywords:* UML; Virtual Prototype; Simulation; MDA; Functional Safety; ISO26262.

## 1. Introduction

In the automotive domain, safety plays an ever increasing role in the development of future vehicles. Today, requirements, design and verification must follow stringed specifications from standards such as ISO26262 for functional safety. The sensing and controlling is the work of the highly distributed electronic control units (ECU) and it is no surprise, that through all these new features in cars, more than 100 of these microcontrollers[1] are currently integrated in a modern car. This situation has also an impact on the amount of software in cars today, which can total 150 million lines of code[2]. Since 60% of vehicle recalls are nowadays due to software defects[3], new methods such as virtual prototyping (VP) helps to test embedded software in much earlier design phases, before a first real hardware prototype is available. Changes on a virtual hardware design are much faster than changes on the real platform.

*E-mail address:* r.weissnegger@cisc.at

which takes weeks or months of redesign and production, which in turn has impact on time to market. With intensive simulation, corner cases but also long term reliability errors can be encountered, which also prevents costly product recalls. Environmental impacts on the virtual prototype can be simulated and reproduced, where real testbeds are not capable of this kind of verification. The drawback, a complex VP is not developed overnight. It takes a lot of effort, experienced designers and engineers to build a so called digital twin of the actual hardware. Our proposal is thus to reuse models for virtual hardware prototyping from open libraries such as Open Virtual Platform (OVP),[4].

In this work we present a novel design and verification framework named SHARC (Simulation and Verification of Hierarchical Embedded Microelectronic Systems),[5]. In particular, we will focus on a design and verification flow for a safety aware VP (SaVeSoC - Safety aware Virtual Prototype evaluation and verification of a System on Chip). This design flow is conform to the ISO26262 standard and meets all its requirements to produce a reliable product in the end. The whole system, from a first functional specification down to hardware design, is specified by standardized modeling languages (UML/MARTE). Before the VP is generated from the hardware specification, several reliability analysis methods are applied and executed on this specification. This allows an early evaluation of the hardware design regarding safety, before testing the prototype in simulation. Furthermore, we show the integration of the generated VP into the system design, to verify the interfaces and its functionality.

The whole process is depicted in Figure 1. By adding a smaller V-model to the traditional approach, we are closing the technological and organizational gap between system design and hardware development which exists in today's tool flows. Since our used design and simulation languages share a seamless development flow, no information transfer is needed between those design levels. Also changing requirements in the specification can be easily and efficient tested towards the VP.

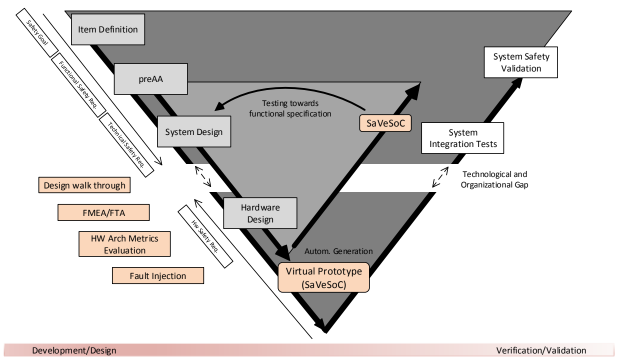


Fig. 1. Process of SaVeSoC integration into functional specification

## 2. Related Work

Popular approaches such as[6] and[7] have shown that UML as modeling language can be efficiently used with analysis and verification methods such as failure mode and effect analysis (FMEA), fault tree analysis (FTA)[8],[9], design walk through[10], code-generation[11],[12],[13] and many more. The drawback of UML, in terms of simulation to verify the system behavior is, that code-generation can only be done at a very late stage or even at the end of the design process, when all details are very well known. Later changes in design are costly and result in inconsistent models and furthermore reverse-engineering is an error prone and cumbersome task. The majority of components in new projects are reused and simply extended by the addition of new features to reduce costs and time-to market. The reuse of whole safety concepts, well-trusted designs and mechanisms is thus becoming more important to reduce the effort required for developing complex systems. This situation prompts the urgent demand for new techniques to simulate the behavior in early development-phases by reusing verified system components.

Two major European projects that also relate to a model-based design in the automotive domain are the SAFE[14] (Safe Automotive soFtware architEcture) and MEANAD[15] (Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles) project. The objective of the SAFE project was to define development processes complying with functional safety and to develop new methods for defining safety goals. Furthermore, the aim of this project was to improve dependability from vehicle to component and the early evaluation of safety architecture. During this project, this has been achieved by defining a meta-model for a model-based safety analysis, which is based on existing technologies (ReqIF, EAST-ADL and AUTOSAR). With the scope on Fully Electric Vehicle (FEV) and to bring the engineering of FEV to a next level, the MAENAD project extended the EAST-ADL2 standard with advanced capabilities to facilitate development of dependable, efficient and affordable products. By using a common modeling language in the project, they achieved to understand engineering information across different departments and companies, to exchange engineering models between different organizations and to progress jointly on tools and methodologies for modeling, analysis and synthesis. In addition, the MAENAD project proposes to use an overall design methodology for FEV development.

## 3. Use Case

Throughout this paper we will demonstrate our methodologies on a relevant problem in today's automotive domain, a battery management system (BMS) for Li-Ion powered electrical vehicles. This industrial use case was provided by CISC Semiconductor GmbH, based in Austria and the United States. This use case will help to illustrate more fully the innovative capabilities and benefits of our approach. As more and more vehicles are now powered by Li-Ion batteries, the challenge for engineers to ensure reliability and fault tolerance is also greatly increasing. It is crucial for ensuring safe operating conditions of a battery that monitoring systems such as the BMS measure the voltage, temperature and current of the battery very precisely. This information must be forwarded to a vehicle-wide controller network to ensure a reliable and fully utilized system. Problems with overheating or even explosions have been frequent in the past. The main cause of these problems was an excessively high energy intake from regenerative braking or harsh environmental conditions. Management systems and mechanisms are thus essential to assure that persons are not put at risk and that no damage is caused. For reasons of simplification, we only consider the major components of the electric vehicle, for the analysis of the battery and the BMS. This includes the battery pack in Li-Ion technology, the BMS which measures voltage and temperature of the battery, an inverter ECU, a controller and the electric motor model.

To achieve a first executable specification of our use case, we used our methodologies provided in[16], to connect the UML/MARTE design models with reusable simulation models (system component library) in SystemC with extensions such as transaction level modeling (TLM) and analog mixes signal (AMS). These functional models are early executable models on a high level of abstraction and can be refined, depending on their purpose, through more detailed models. The provided model library contains analog and digital models to gain an early and fast evaluation of the functional specification on system level. This functional specification has been tested by the automatic generated testbenches described in[17]. With this approach, we moreover eliminate the tedious task of exporting our gathered data to other simulation tools such as Matlab/Simulink and leave the UML/MARTE design as a single source of information. Furthermore, we rely on standardized and open modeling and simulation languages and keep the costs for licenses low.

## 4. Specification and Configuration the Hardware Platform

From the gathered information of our functional and executable specification we now delve deeper into the hardware design. Since MARTE and SystemC share the same philosophy to move from system to hardware and software development, we are now able to specify, through refinement, the internal architecture of our BMS platform. As the major goal of this work is to build a ISO26262 safety aware platform, this is an important step in the development process and takes a lot of effort into account[18], since many different methods and measures have to be applied to the hardware platform to guarantee a reliable and safe product in the end.

A goal of this work is to include the whole generation of the VP into a seamless design flow for safety critical systems. Furthermore, the design and configuration of the VP on a graphical modeling standard, in this case

UML/MARTE. This approach brings the advantage to evaluate and analyze the configuration of the hardware design before the actual prototype is generated from UML models. OVP itself does not support verification regarding safety, nor is OVP till now embedded in a seamless design flow.

Although, UML/MARTE provide a rich set of different stereotypes to describe the hardware platform (HwBus, HwMemory, HwProcessor), it does not provide the level of detail for the hardware configuration as expected by the OVP methodology. Several mandatory properties such as the BusInterface, MemoryMaps or the VLNV principle are by now not supported in the MARTE standard. To overcome this issues, we rely on the specification of the IP-XACT standard, which helps us to define our platform in a sufficient high degree of detail for the generation of the VP. Both, IP-XACT and OVP rely on the VLNV principle for structuring the models. Therefore, we built on approaches such as[19] to extend the MARTE standard by properties in IP-XACT for hardware description. Figure 2 shows a snapshot

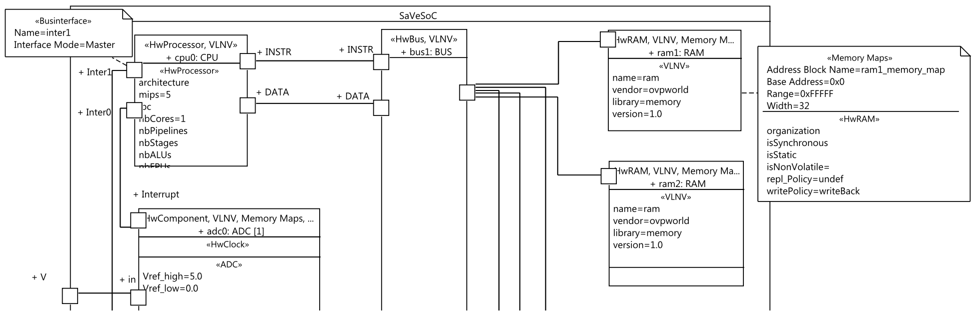


Fig. 2. Hardware platform of SaVeSoC including IP-XACT extensions for UML/MARTE.

of the composed structural architecture model of the platform consisting of a processor, bus, memory, CAN and two ADC. For simplification we only show the major components of the design of the battery management system. The designer can easily compose his system by using the standard models from the MARTE library. Depending on the nature of the component, the designer is able to extend the component with specific hardware properties in the IP-XACT standard such as MemoryMaps and BusInterface. The properties from the IP-XACT standard are shown in the description of the MARTE model. Further extensions for IP-XACT such as VLNV can be added to the hardware components as well.

## 5. Evaluation of the Virtual Prototype

### *5.1. Evaluation of the Hardware Configuration*

Based on the specification of our hardware platform in Fig. 2 we are able to perform different evaluation methods on our hardware design. As mentioned in the previous chapters, there are several methods which can be used to evaluate the reliability and safety of our hardware architecture, one of these is the hardware architectural metrics evaluation. The hardware architectural metrics are handled in Part 5[20] of the ISO26262, product development at the hardware level. This evaluates the hardware architecture of the item against the requirements for fault handling. This part also includes guidance on avoiding systematic and random hardware failures by means of appropriate safety mechanisms. Each safety-related hardware element is analyzed regarding single point (SPFM), residual and multiple point faults (LFM). It also describes the effectiveness of the hardware architecture in coping with random hardware failures (PMHF). Each hardware part is to be protected by means of safety-mechanisms. The diagnostic coverage gives evidence of the effectiveness of these mechanisms. Whether the item (system or array of systems according to ISO26262) passes or fails a given ASIL is also a result of the hardware architectural metrics evaluation. In order to achieve a specific ASIL, the values from Table 1 must be met (FIT- failure in time).

To perform this evaluation on our hardware specification, we built on approaches such as[21],[22] and[23]. These approaches use the capabilities of UML/MARTE and IP-XACT to perform several quantified methods to evaluate the reliability of the hardware platform. By adding following safety properties as extension to the IP-XACT standard they achieved reusability of hardware safety artifacts:

**Failure rate (FR)** - is usually known by the vendor of the component. It is a result of field return and statistical data, where expert judgment can also be considered.
**Failure modes (FM)** - describes the different modes where a failure can occur. The failure modes depend on the application in which the element is used.
**Safety mechanism (SM)** - Implemented mechanism to detect and control faults. It prevents faults from violating the safety goal. If a fault is detected, a safe state is initiated.
**Diagnostic coverage (DC)** - is the effectiveness of the internal safety mechanism implemented to cover single point, residual or latent faults.

These reuse strategies are commonly used in industry and are known as clone and own. The existing and reused safety models can give important feedback through an early safety assessment for a modified platform or even a new product. Changes and adaption on the overall platform must be taken into account, of course when reusing safety artifacts during development. Since the main contribution of this paper is the generation and integration of the virtual prototype into a seamless design and development flow, we do not go into detail on the hardware evaluation. A more detailed description of the developed approach is given in[21].

Table 1. Architectural Metrics - evaluates whether the hardware achieves a certain ASIL, according to ISO26262.

| | ASIL B | ASIL C | ASIL D |
|---|---|---|---|
| SPFM | $\geq 90\%$ | $\geq 97\%$ | $\geq 99\%$ |
| LFM | $\geq 60\%$ | $\geq 80\%$ | $\geq 90\%$ |
| PMHF | $< 10^{-7}h^{-1}$ | $< 10^{-7}h^{-1}$ | $< 10^{-8}h^{-1}$ |

### *5.2. Generation, Integration and Verification of the Virtual Prototype*

One of the outlined goals in this work is to develop and test embedded software within the development phases, on a realistic hardware prototype of the target system, before the actual platform is available. By applying a model based approach in UML/MARTE, the designer is able to configure the whole hardware platform, for instance instructions per seconds of the processor, which affects the simulation time directly. The designer uses the standard MARTE models from the library and adds additional IP-XACT properties to his models. After the evaluation of the design regarding SPFM, LFM and PMHF, the description of the platform is converted to a TCL description in the OVP standard. The OVP iGen converter takes the information from the TCL script and generates a full SystemC- platform using the modular components of the OVP library. The resulting virtual prototype can then be used for further hardware simulations, such as fault-injections on TLM level[24,25]. Since our whole methodology (from functional specification through to hardware and software design) relies on the same modeling language and furthermore the same hardware description language respectively system-modeling language, we are easily able to reapply our generated safety aware VP into the functional specification of the system design. After generation, the hardware description of the SaVeSoC platform with the whole interface specification is added to the UML model library and can be reapplied to the system design including the generated simulation files in SystemC. This saves time in terms of integration effort, when testing the virtual prototype on the functionality of the whole system, as recommended by ISO26262. System level testbenches can also be reused for the verification of the entire system including the integrated VP. Figure 3 depicts the resulting system level description including the battery and new defined BMS hardware. The battery component is no longer a black box where the functionality is described in SystemC. It is now a white box, where the detailed hardware of the battery component is specified. It consists of the SaVeSoC element, which is the hardware platform of the BMS including an application for plausibility checks. It measures the voltage and temperature of the batterypack over two ADC and computes the state of charge and stage of health. Since we are relying on the same simulation engine, the virtual prototype can now be easily tested on a higher abstraction level, which also speeds up the overall simulation time. A challenge when integrating the VP to the functional specification was the communication interface between the functional simulation environment of SystemC-AMS and the OVP platform, because these platforms are mainly used to process data measured from embedded sensors. We thus implemented communication channels which guarantee data exchanges between different SystemC dialects (SystemC TLM2.0, SystemC AMS).

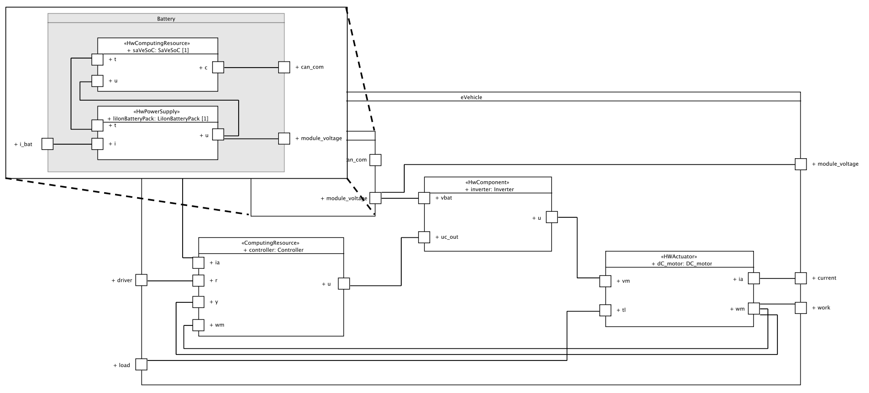


Fig. 3. Virtual Prototype integration into the functional specification of the system design to verify its functionality.

## *5.3. Distribution of Simulation Tasks to a Cloud-based Environment*

To verify the functionality, we integrated our generated VP into the functional specification of our system design. This specification is tested with the constrained random verification (CRV) principle in the Universal Verification Methodlogy (UVM) standard and measured towards functional coverage. Functional coverage is defined as a metric which is used to determine the completeness and verification process of the design. It emphasizes design verification where the focus, besides functional process of a design, is also on non-functional aspects such as safety, timing or power. Functional coverage tells us about the quality of a testbench and what portion of the design has been activated and tested during the simulation run (controllability). On the other hand, observability shows the ability to observe effects of the simulation (white-box vs. black-box testing). Thus, this metric allows us to answer the crucial question in the verification process "Are we done, yet?". We can classify coverage[26] by their method of creation (implicit vs. explicit) and their origin of source (specification vs. implementation). Line coverage and expression coverage are two examples of an implicit coverage metric and can be automatically derived from the code, whereas functional coverage (explicit coverage metric) has to be defined and implemented by the engineer, derived from the various requirements and the specification document. Functional coverage is distinguished between two simulation methodologies, direct testing and constrained random verification, whereas the later one is able to achieve a higher distribution over the huge space of the available input stimuli. This mechanism increases the coverage and the ability to find corner cases in the design, by creating random tests, which not have been found by direct testing. The coverage space classified by a implicit specification (also known as intelligent verification) is a current academic research area, where the coverage metrics are automatically extracted by a tool and are derived from the design specification. These higher-level functional behaviors cannot be automatically derived from the implementation alone and need the information from the specification as well.

To satisfy the demand on a high functional coverage, we automatically derive testbenches from semi-formal safety requirements (extension to SysML) in early phases of development. With this approach we are able to cover all possible parameters and various variants of a vehicle. Any shortcomings in the design can thus be deteted much ealier in the development process to reduce costs and time-to-market. More detailed of this approach is given in publication[17].

Since automatic generation of testbenches in combination with CRV produces a high amount of simulation tasks we developed an extension to the traditional UVM layered architecture for parallel execution in a cloud-based environment[27]. Since the overall result of a simulated sequence does not affect other configurations in any way, parallel processing of various sequences is possible. Through applying message patterns from the Enterprise Integration Patterns[28] on the UVM standard and furthermore integration into a whole framework we could speed up the verification process of complex embedded systems. Due to the fact that simulation of sequences is the most time consuming part of the verification, a theoretical linear speedup can be expected. The execution environment can be switched between private/internal clusters, but also public cloud solutions such as Amazon AWS[29] or Microsoft Azure[30]. The benefits of this approach are reduction of simulation time, license costs, flexible infrastructure, defined levels of abstraction, efficient degree of capacity utilization and prevention of data loss.

Figure 4 depicts our design, simulation and verification framework SHARC. It is based on the Eclipse UML editor Papyrus[31] and allows the execution of UML/MARTE models by using our system component library. This library includes digital and analog simulations models which can communicate through defined interfaces. The use case of a BMS, shown in the center of this figure, is designed and configured with the help of executable UML/MARTE models.

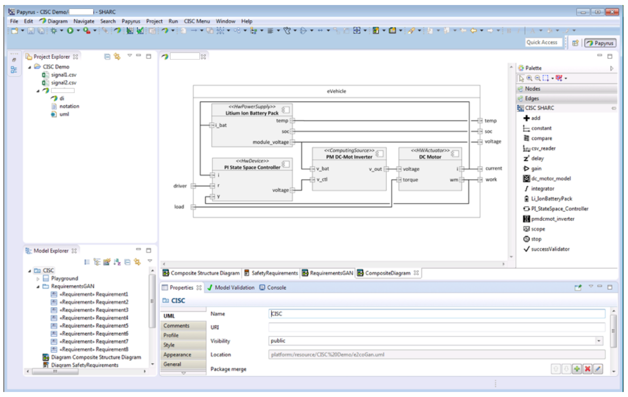


Fig. 4. Screenshot of SHARC, a design and simulation framework for the verification of (safety-critical) embedded systems in the automotive domain.

## 6. Conclusion

In this work we presented a novel design and simulation framework for the verification of safety-critical systems. A standardized modeling language based on UML was used to represent the design flow, from functional specification down to hardware and software, which is in conformance with the functional safety standard (ISO26262). This model-based approach eases the communication between different stakeholders involved in the development process and serves as a single-source of information. Through tight integration of recommended safety analysis methods such as FTA, FMEA, hardware architectural metrics and simulation-based verification, we achieved consistency, correctness, and completeness throughout the development process. The used UML profile MARTE and extensions to IP-XACT helped to specify and evaluate the hardware in early development stages. From this specification a virtual prototype was generated, which can be tested by fault injection techniques on a higher abstraction level. Moreover, we showed the seamless integration of the virtual prototype into the functional specification to verify its functionality. Through relying on a cloud-based and distributed environment we are able to speed-up the verification process and by using our approach, we achieve to close the technological and organizational gap in today's toolchain of safety-critical system development.

## Acknowledgments

A part of the work has been performed in the project eRamp (Grant Agreement N621270), co-funded by grants from Austria, Germany, Slovakia and the ENIAC Joint Undertaking. Some parts have also been experimented in the

OpenES CATRENE Project: CA703 - 2013 research program supported by the FFG (Austrian Research Promotion Agency), project-number 843380 in tight cooperation with CISC Semiconductor.

## References

1. Charette, R.. This Car Runs on Code - IEEE Spectrum. 2009.
2. Eitdigital, . FORD GT - Lines of code. 2016.
3. SRR, . 2016 Automotive Warranty and Recall Report: New Insights For the Road Ahead. Tech. Rep.; SRR; 2016.
4. OVP, . Open Virtual Platform. 2016. URL: `http://www.ovpworld.org`.
5. CISC, . CISC Semiconductor GmbH. 2017. URL: `https://www.cisc.at/`.
6. Adler, R., Domis, D., Höfig, K., Kemmann, S., Kuhn, T., Schwinn, J.P., et al. Integration of component fault trees into the UML. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 2011;:312–327.
7. David, P., Idasiak, V., Kratz, F.. Towards a better interaction between design and dependability analysis: FMEA derived from UML/SysML models. Safety, Reliability and Risk Analysis: Theory, Methods and Applications - Proceedings of the Joint ESREL and SRA-Europe Conference 2009;(August 2015):2259–2266.
8. Pai, G.J., Dugan, J.B.. Automatic synthesis of dynamic fault trees from UML system models. In: 13th International Symposium on Software Reliability Engineering, 2002. Proceedings. 2002, p. 243–254.
9. Lauer, C., German, R., Pollmer, J.. Fault Tree Synthesis from UML Models for Reliability Analysis at Early Design Stages. SIGSOFT Softw Eng Notes 2011;36(1):1–8.
10. Gvero, I.. Computers As Components, 3rd Edition: Principles of Embedded Computing System Design by Marilyn Wolf. SIGSOFT Softw Eng Notes 2013;38(5):67–68.
11. Ebeid, E., Quaglia, D., Fummi, F.. Generation of SystemC/TLM code from UML/MARTE sequence diagrams for verification. In: 2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS). 2012, p. 187–190.
12. Peñil, P., Villar, E., Posadas, H., Medina, J.. Executable SystemC specification of the MARTE generic concurrent and communication resources under different Models of Computation. Proc of Satellite Workshop of the the 21st Euromicro Conference on Real-Time Systems 2009;.
13. Nguyen, M.D., Thalmaier, M., Wedler, M., Stoffel, D., Kunz, W., Bormann, J.. Advances in Design Methods from Modeling Languages for Embedded Systems and SoC's. Advances in Design Methods from Modeling Languages for Embedded Systems and SoC's 2010;63:197–212.
14. SAFE Project. 2014. URL: `http://www.safe-project.eu/`.
15. maenad.eu. 2014. URL: `http://www.maenad.eu/`.
16. Weissnegger, R., Schuß, M., Kreiner, C., Pistauer, M., Kay, R., Steger, C.. Bringing UML / MARTE to life : Simulation-based Verification of Safety-Critical Systems. In: 2016 Forum on Specification and Design Languages (FDL). Bremen, Germany; 2016,.
17. Weissnegger, R., Schuß, M., Kreiner, C., Pistauer, M., Römer, K., Steger, C.. Seamless Integrated Simulation in Design and Verification Flow for Safety-Critical Systems. In: Computer Safety, Reliability, and Security: SAFECOMP 2016 Workshops, ASSURE, DECSoS, SASSUR, and TIPS Proceedings. Trondheim, Norway; 2016, p. 359–370.
18. Kreiner, C.. Trident Architectural Views: A Pattern for Dependable Systems Design. In: Proceedings of the 20th European Conference on Pattern Languages of Programs. EuroPLoP '15; New York, NY, USA: ACM. ISBN 978-1-4503-3847-9; 2015, p. 18:1—-18:9.
19. André, C., Mallet, F., Khan, A.M., de Simone, R.. Modeling SPIRIT IP-XACT with UML MARTE. Proc DATE Workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile 2008;.
20. ISO 26262, . Road vehicles-functional safety-Part 5: Product development at the hardware level. 2011.
21. Weissnegger, R., Pistauer, M., Kreiner, C., Römer, K., Steger, C.. A novel method to speed-up the evaluation of cyber-physical systems (ISO 26262). In: Conti, M., Orcioni, S., editors. 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES) 2015, Ancona, Italy, October 29-30, 2015. IEEE. ISBN 978-8-8875-4808-2; 2015, p. 109–114.
22. Weissnegger, R., Pistauer, M., Kreiner, C., Kay, R., Steger, C.. A Novel Method for Fast Evaluation of Cyber-Physical Systems in Compliance with Functional Safety. In: Euromicro Conference in Software Engineering and Advanced Applications - Proceedings of the Work in Progress Session. Funchal, Madeira: Johannes Kepler University Linz; 2015, p. 24–25.
23. Das, N., Taylor, W.. Quantified fault tree techniques for calculating hardware fault metrics according to ISO 26262. In: 2016 IEEE Symposium on Product Compliance Engineering (ISPCE). 2016, p. 1–8.
24. Tabacaru, B.A., Chaari, M., Ecker, W., Kruse, T., Novello, C.. Speeding up safety verification by fault abstraction and simulation to transaction level. In: 2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC). 2016, p. 1–6.
25. Tabacaru, B.a., Chaari, M., Ecker, W., Kruse, T.. Fault-Injection Techniques for TLM-Based Virtual Prototypes; 2015.
26. Verification Academy, . Coverage Cookbook. 2012.
27. Weissnegger, R., Schuß, M., Schachner, M., Römer, K., Steger, C., Pistauer, M.. A Novel Simulation-based Verification Pattern for Parallel Executions in the Cloud. In: Proceedings of the 21st European Conference on Pattern Languages of Programs. EuroPlop '16; ACM; 2016, p. 20:1—-20:9.
28. Hohpe, G., Woolf, B.. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.; 2003.
29. Amazon Web Services. 2016. URL: `https://aws.amazon.com/`.
30. Microsoft Azure. 2016. URL: `https://azure.microsoft.com/`.
31. Eclipse, . Papyrus. 2017. URL: `https://www.eclipse.org/papyrus/`.

# Bibliography

[1] *A Guide To The Project Management Body Of Knowledge (PMBOK Guides)*. Project Management Institute, 2004. ISBN: 193069945X, 9781933890517.

[2] Accellera. *Universal Verification Methodology (UVM) 1.2 User's Guide*. Tech. rep. Accellera, 2015.

[3] Accellera Systems Initiative. *SystemC*. 2016. URL: `http://www.accellera.org/downloads/standards/systemc/` (visited on 2017-05-01).

[4] Morayo Adedjouma; Hubert Dubois; Kamel Maaziz; and François Terrier. "A Model-Driven Requirement Engineering Process Compliant with Automotive Domain Standards." In: *Third Workshop on Model Driven Tool and Process Integration (MDTPI), Paris, France, June 16, 2010 Proceedings* (2010).

[5] Nico Adler; Stefan Otten; Markus Mohrhard; and Klaus D. Muller-Glaser. "Rapid safety evaluation of hardware architectural designs compliant with ISO 26262." In: *2013 International Symposium on Rapid System Prototyping (RSP)* (2013), pp. 66–72. DOI: `10.1109/RSP.2013.6683960`. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6683960`.

[6] Rasmus Adler; Dominik Domis; Kai Höfig; Sören Kemmann; Thomas Kuhn; Jean Pascal Schwinn; and Mario Trapp. "Integration of component fault trees into the UML." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6627 LNCS (2011), pp. 312–327. ISSN: 03029743. DOI: `10.1007/978-3-642-21210-930`.

[7] Sami Alajrami; Barbara Gallina; and Alexander Romanovsky. "EXE-SPEM: Towards Cloud-Based Executable Software Process Models." In: *4th International Conference on Model-Driven Engineering and Software Development*. 2016. URL: `http://www.es.mdh.se/publications/4186`.

[8] Altera. *No Safety without Security*. 2016. URL: `http://systemdesign.altera.com/no-safety-without-security-iot/` (visited on 2017-02-05).

[9] Charles André; Frédéric Mallet; Aamir Mehmood Khan; and Robert de Simone. "Modeling SPIRIT IP-XACT with UML MARTE." In: *Proc. DATE Workshop on*

*Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile* (2008).

[10] *ATESST*. 2010. URL: http://www.atesst.org (visited on 2017-05-01).

[11] Rabie Ben Atitallah; Eric Piel; Julien Taillard; Smail Niar; and Jean Luc Dekeyser. "From high level MPSoC description to SystemC code generation." In: *International ModEasy'07 Workshop in conjunction with Forum on specification and Design Languages (FDL'07)* (2007).

[12] Michael Azoff. "MDD in Modern Web-based Systems X EXECUTIVE SUMMARY Model Driven Development, CA Gen, and Mission-critical Applications." In: (2008). URL: www.butlergroup.com.

[13] M Barnasconi; M Dietrich; K Einwich; T Vortler; R Lucas; J P Chaput; F Pecheux; Z Wang; P Cuenot; I Neumann; T Nguyen; and R Lucas. "UVM-SystemC-AMS Framework for System-Level Verification and Validation of Automotive Use Cases." In: *Design Test, IEEE* PP.99 (2015), p. 1.

[14] Martin Barnasconi and Marie Curie. "Advancing system-level verification using UVM in SystemC." In: *Design and Verification US (DVCon)* (2014).

[15] Jean Louis Boulanger and Quang Van Dao. "Experiences from a model-based methodology for embedded electronic software in automobile." In: *2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications, ICTTA* (2008), pp. 1–6. DOI: 10.1109/ICTTA.2008.4530259.

[16] Catrene. *OpenES CATRENE Project: CA703*. 2016. URL: http://www.ecsi.org/openes (visited on 2017-05-01).

[17] Yung-chang Chang; Li-ren Huang; Hsing-chuang Liu; Chih-jen Yang; and Ching-te Chiu. "Assessing Automotive Functional Safety Microprocessor with ISO 26262 Hardware Requirements." In: *Technical Papers of 2014 International Symposium on VLSI Design, Automation and Test* (2014), pp. 3–6.

[18] Robert N. Charette. *This Car Runs on Code - IEEE Spectrum*. 2009. URL: http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code (visited on 2017-05-01).

[19] CISC. *CISC Semiconductor GmbH*. 2017. URL: https://www.cisc.at/ (visited on 2017-05-01).

[20] *COMPLEX / ECSI*. 2016. URL: http://ecsi.org/complex (visited on 2016-01-10).

[21] *COSEDA - COSIDE*. 2017. (Visited on 2017-03-10).

[22] Philippe Cuenot; Nico Adler; and Stefan Otten. *Safe Automotive soFtware architEcture ( SAFE ) Proposal for extension of Meta model for hardware modeling.* 2013.

[23] M Damm; C Grimm; J Haas; A Herrholz; and W Nebel. "Connecting SystemC-AMS models with OSCI TLM 2.0 models using temporal decoupling." In: *2008 Forum on Specification, Verification and Design Languages.* 2008, pp. 25–30. DOI: 10.1109/FDL.2008.4641416.

[24] N Das and W Taylor. "Quantified fault tree techniques for calculating hardware fault metrics according to ISO 26262." In: *2016 IEEE Symposium on Product Compliance Engineering (ISPCE).* 2016, pp. 1–8. DOI: 10.1109/ISPCE.2016.7492848.

[25] P David; V Idasiak; and F Kratz. "Towards a better interaction between design and dependability analysis: FMEA derived from UML/SysML models." In: *Safety, Reliability and Risk Analysis: Theory, Methods and Applications - Proceedings of the Joint ESREL and SRA-Europe Conference* 3.August 2015 (2009), pp. 2259–2266.

[26] EAST-ADL. *EAST-ADL Association.* 2016. URL: http://www.east-adl.info/.

[27] E Ebeid; D Quaglia; and F Fummi. "Generation of SystemC/TLM code from UML/MARTE sequence diagrams for verification." In: *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS).* 2012, pp. 187–190. DOI: 10.1109/DDECS.2012.6219051.

[28] Eclipse. *Papyrus.* 2017. URL: https://www.eclipse.org/papyrus/ (visited on 2017-05-01).

[29] *Eclipse Sirius.* 2017. URL: http://www.eclipse.org/sirius/ (visited on 2017-03-03).

[30] Eitdigital. *FORD GT - Lines of code.* 2016. URL: https://www.eitdigital.eu/news-events/blog/article/guess-what-requires-150-million-lines-of-code/.

[31] ENIAC. *eRamp - Excellence in Speed and Reliability for More than Moore Technologies.* URL: https://www.infineon.com/cms/en/product/promopages/eramp/ (visited on 2017-02-01).

[32] Huascar Espinoza; Daniela Cancila; Bran Selic; and Sébastien Gérard. "Challenges in combining SysML and MARTE for model-based design of embedded systems." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5562 LNCS (2009), pp. 98–113. ISSN: 03029743. DOI: 10.1007/978-3-642-02674-4_8.

[33] Sébastien Gérard; Cédric Dumoulin; Patrick Tessier; and Bran Selic. "Papyrus: A UML2 Tool for Domain-Specific langauge Modeling." In: *Model-Based Engineering*

*of Embedded Real-Time Systems*. Vol. 6100. Lecture Notes in Computer Science. Springer, 2011. Chap. 19, 361 {\textendash} 367.

[34] "Graz University of Technology - Information and Computer Engineering (ICE)." PhD thesis. URL: http://ice.tugraz.at.

[35] Object Management Group. "OMG Unified Modeling Language TM ( OMG UML ), Superstructure v.2.5." In: *InformatikSpektrum* 21.May (2015), p. 758. ISSN: 08950695. DOI: 10.1007/s002870050092. URL: http://www.omg.org/spec/UML/2.5/.

[36] Igor Gvero. "Computers As Components, 3rd Edition: Principles of Embedded Computing System Design by Marilyn Wolf." In: *SIGSOFT Softw. Eng. Notes* 38.5 (2013), pp. 67–68. ISSN: 0163-5948. DOI: 10.1145/2507288.2507292. URL: http://doi.acm.org/10.1145/2507288.2507292.

[37] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN: 0321200683.

[38] Ireri Ibarra and D.D. Ward. "Development Phase in Accordance with ISO 26262." In: *8th IET International System Safety Conference incorporating the Cyber Security Conference 2013* (2013), pp. 5.1–5.1. DOI: 10.1049/cp.2013.1718. URL: http://digital-library.theiet.org/content/conferences/10.1049/cp.2013.1718.

[39] "IEC TR 62380 - Reliability data handbook - Universal model for reliability prediction of electronic components." In: (2005).

[40] Leandro Soares Indrusiak; Imran Quadri; Ian Gray; Neil Audsley; and Andrey Sadovykh. "A MARTE Subset to Enable Application-Platform Co-simulation and Schedulability Analysis of NoC- based Embedded Systems." 2011.

[41] ISO. "Functional Safety ISO26262 - Part 4: Product development at the system level." In: 2011 (2011), pp. 1–35.

[42] ISO 26262. *Road vehicles-functional safety-Part 5: Product development at the hardware level.* 2011.

[43] "ISO/IEC/IEEE Systems and software engineering – Architecture description." In: *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)* (2011), pp. 1–46. DOI: 10.1109/IEEESTD.2011.6129467.

[44] Seo-Hyun Jeon; Jin-Hee Cho; Yangjae Jung; Sachoun Park; and Tae-Man Han. "Automotive hardware development according to ISO 26262." In: *13th International Conference on Advanced Communication Technology (ICACT2011)* (2011), pp. 588–592. ISSN: 1738-9445.

[45] Christian Kreiner. "Trident Architectural Views: A Pattern for Dependable Systems Design." In: *Proceedings of the 20th European Conference on Pattern Languages of Programs*. EuroPLoP '15. ACM, 2015, 18:1–18:9. ISBN: 978-1-4503-3847-9. DOI: 10.1145/2855321.2855340. URL: http://doi.acm.org/10.1145/2855321.2855340.

[46] P Kruchten. "Architectural blueprints–the" 4+ 1" view model of software architecture." In: *IEEE Software* 12.November (1995), pp. 42–50. ISSN: 07407459. DOI: 10.1145/216591.216611. arXiv: EEESoftware.

[47] C Lauer; R German; and J Pollmer. "Modeling and Analysis of Advanced Automotive ECU Architectures at Early Design Stages Using EMF and Model Transformation." In: *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement Companion*. 2010, pp. 18–23. DOI: 10.1109/SSIRI-C.2010.37.

[48] Christoph Lauer; Reinhard German; and Jens Pollmer. "Fault Tree Synthesis from UML Models for Reliability Analysis at Early Design Stages." In: *SIGSOFT Softw. Eng. Notes* 36.1 (2011), pp. 1–8. ISSN: 0163-5948. DOI: 10.1145/1921532.1921558. URL: http://doi.acm.org/10.1145/1921532.1921558.

[49] Alessandro Lonardi and Graziano Pravadelli. "On the Co-simulation of SystemC with QEMU and OVP Virtual Platforms." In: *VLSI-SoC: Internet of Things Foundations: 22nd IFIP WG 10.5/IEEE International Conference on Very Large Scale Integration, VLSI-SoC 2014, Playa del Carmen, Mexico, October 6-8, 2014, Revised Selected Papers*. Ed. by Luc Claesen; Maria-Teresa Sanz-Pascual; Ricardo Reis; and Arturo Sarmiento-Reyes. Springer International Publishing, 2015, pp. 110–128. ISBN: 978-3-319-25279-7. DOI: 10.1007/978-3-319-25279-7_7. URL: http://dx.doi.org/10.1007/978-3-319-25279-77.

[50] Ronan Lucas; Philippe Cuenot; Marie-Minerve Louërat; Yao Li; Zhi Wang; Jean-Paul Chaput; François Pêcheux; Ramy Iskander; Martin Barnasconi; and Thilo Vörtler. "Generation of UVM compliant Test Benches for Automotive Systems using IP-XACT with UVM-SystemC and SystemC AMS*." In: *DVCon Europe*. 2014, pp. 1–8.

[51] Georg Macher; Michael Stolz; Eric Armengaud; and Christian Kreiner. "Filling the gap between automotive systems, safety, and software engineering." In: *e & i Elektrotechnik und Informationstechnik* 132.3 (2015), pp. 142–148. ISSN: 0932-383X. DOI: 10.1007/s00502-015-0301-x. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-84938097514{\&}partnerID=tZOtx3y1.

[52] T Machne; Z Wang; B Vernay; L Andrade; C Ben Aoun; J P Chaput; M M Louërat; F Pêcheux; A Krust; G Schröpfer; M Barnasconi; K Einwich; F Cenni; and O Guillaume. "UVM-SystemC-AMS based framework for the correct by construction design

of MEMS in their real heterogeneous application context." In: *2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 2014, pp. 862–865. DOI: `10.1109/ICECS.2014.7050122`.

[53] MAENAD. *maenad.eu.* 2014. URL: `http://www.maenad.eu/` (visited on 2017-03-01).

[54] Raluca Marinescu; Henrik Kaijser; Marius Mikucionis; Alexandre David; Cristina Seceleanu; and Loenn Henrik. "Analyzing Idustrial Architectural Models by Simulation and Model-Checking." In: *Formal Techniques for Safety-Critical Systems* 419 (2014), pp. 189–205. DOI: `10.1007/978-3-319-05416-2`.

[55] Matlab. *MATLAB - The Language of Technical Computing.* URL: `http://www.mathworks.com/products/matlab/index.html?s{\_}tid=gn{\_}loc{\_}drop` (visited on 2017-05-01).

[56] M Maurer; J C Gerdes; B Lenz; and H Winner. *Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte.* Springer Open. Springer Berlin Heidelberg, 2015, pp. 446–448. ISBN: 9783662458549.

[57] J Miller and J Mukerji. "MDA Guide Version 1.0. 1." In: *Object Management Group* June (2003). URL: `http://scholar.google.com/scholar?hl=en{\&}btnG=Search{\&}q=intitle:MDA+Guide+Version+1.0.1{\#}0`.

[58] F. Mischkalla; Da He Da He; and W. Mueller. "Closing the gap between UML-based modeling, simulation and synthesis of combined HW/SW systems." In: *Design, Automation and Test in Europe Conference Exhibition (DATE)* (2010). ISSN: 1530-1591. DOI: `10.1109/DATE.2010.5456990`.

[59] L.G. Murillo; M. Mura; and M. Prevostini. "Semi-automated Hw/Sw Co-design for embedded systems: from MARTE models to SystemC simulators." In: *2009 Forum on Specification & Design Languages (FDL)* (2009). ISSN: 1636-9874.

[60] Luis Gabriel Murillo; Marcello Mura; and Mauro Prevostini. "MDE Support for HW/SW Codesign: A UML-based Design Flow." In: *Advances in Design Methods from Modeling Languages for Embedded Systems and SoC's.* Vol. 63. 2010, pp. 19–37. ISBN: 978-90-481-9303-5. DOI: `10.1007/978-90-481-9304-2`. URL: `http://www.springerlink.com/index/10.1007/978-90-481-9304-2`.

[61] Liana Musat; Markus Hübl; Andi Buzo; Georg Pelz; Susanne Kandl; and Peter Puschner. "Semi-formal Representation of Requirements for Automotive Solutions using SysML." In: *Proceedings of the Forum on Specification and Design Languages (FDL 2014).* 2014, pp. 1–8.

[62] Omg. *OMG Systems Modeling Language ( OMG SysML ™ ) v.1.4.* 2015. URL: `http://www.omg.org/spec/SysML/1.2/PDF/`.

[63] OMG. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems.* Tech. rep. Object Management Group, 2016.

[64] OVP. *Open Virtual Platform.* 2016. URL: `http://www.ovpworld.org`.

[65] G J Pai and J B Dugan. "Automatic synthesis of dynamic fault trees from UML system models." In: *13th International Symposium on Software Reliability Engineering, 2002. Proceedings.* 2002, pp. 243–254. DOI: `10.1109/ISSRE.2002.1173261`.

[66] *Pegasus Project.* 2017. URL: `http://www.pegasus-projekt.info/` (visited on 2017-04-04).

[67] P Peñil; E Villar; H Posadas; and J Medina. "Executable SystemC specification of the MARTE generic concurrent and communication resources under different Models of Computation." In: *Proc. of Satellite Workshop of the the 21st Euromicro Conference on Real-Time Systems* (2009).

[68] Éric Piel; Rabie Ben Atitallah; Philippe Marquet; Samy Meftali; Smail Niar; Anne Etien; J.L. Jean-Luc Dekeyser; Pierre Boulet; and I.L.N. Europe. "Gaspard2: from marte to systemc simulation." In: *DATE'08 Workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile* 8 (2008), pp. 1–6.

[69] *QEMU-Project.* 2017. URL: `http://www.qemu-project.org` (visited on 2017-03-10).

[70] Imran Rafiq Quadri; Abdoulaye Gamati; Pierre Boulet; Jean-luc Dekeyser; Imran Rafiq Quadri; and Abdoulaye Gamati. "Modeling of Configurations for Embedded System Implementations in MARTE Modeling of Configurations for Embedded System Implementations in MARTE." In: *Modeling of Configurations for Embedded System Implementations in MARTE. 1st workshop on Model Based Engineering for Embedded Systems Design - Design, Automation and Test in Europe (DATE 2010).* 2010.

[71] *Rational Doors.* 2017. (Visited on 2017-03-03).

[72] A E Rugina; K Kanoun; and M Kaâniche. "The ADAPT Tool: From AADL Architectural Models to Stochastic Petri Nets through Model Transformation." In: *2008 Seventh European Dependable Computing Conference.* 2008, pp. 85–90. DOI: `10.1109/EDCC-7.2008.14`.

[73] Vladimir Rupanov; Christian Buckl; Ludger Fiege; Michael Armbruster; Alois Knoll; and Gernot Spiegelberg. "Early safety evaluation of design decisions in E/E architecture according to ISO 26262." In: *Proceedings of the 3rd international ACM SIGSOFT symposium on Architecting Critical Systems - ISARCS '12* (2012), p. 1. DOI: `10.1145/2304656.2304658`.

[74] *SAFE Project.* 2014. URL: `http://www.safe-project.eu/` (visited on 2017-03-01).

[75] M Schmidberger and M Schmidberger. "Software Engineering as a Service for HPC." In: *2012 11th International Symposium on Parallel and Distributed Computing*. 2012, pp. 34–39. DOI: 10.1109/ISPDC.2012.13.

[76] Markus Schuß. "Design and Implementation of a Distributed Simulation Framework based on Cloud Computing." Master thesis. Graz University of Technology (TU Graz), 2016.

[77] Bran Selić and Sébastien Gérard. *Modeling and analysis of real-time and embedded systems with UML and MARTE*. 2014, Online Ressource (314 pages). ISBN: 978-012-416-656-1. DOI: 10.1016/B978-0-12-416619-6.00008-0.

[78] Aditya A Shah; Alexsandr A Kerzhner; Dirk Schaefer; and Christiaan Paredis. "Multi-View Modeling to Support Embedded Systems Engineering in SysML." In: *Lecture Notes in Computer Science* 5765 (2010), pp. 580–601. DOI: 10.1007/978-3-642-17322-6_25. URL: http://opus.bath.ac.uk/46114/.

[79] Adam Sherer. *Accellera's UVM in SystemC Standardization: Going Universal for ESL*. \url{http://accellera.org/resources/articles/accelleras-uvm-in-systemc-standardization-going-universal-for-esl}. 2015.

[80] Siemens. *SN 29500-1 Expected values, general. (January 2004).*

[81] Harald Sporer; Georg Macher; Eric Armengaud; and Christian Kreiner. "Incorporation of Model-Based System and Software Development Environments." In: *Proceedings - 41st Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2015* (2015), pp. 177–180. DOI: 10.1109/SEAA.2015.65.

[82] Bogdan-andrei Tabacaru; Moomen Chaari; Wolfgang Ecker; and Thomas Kruse. "Fault-Injection Techniques for TLM-Based Virtual Prototypes." 2015.

[83] William Taylor; Gokul Krithivasan; and Jody J. Nelson. "System safety and ISO 26262 compliance for automotive lithium-ion batteries." In: *2012 IEEE Symposium on Product Compliance Engineering, ISPCE 2012 - Proceedings* (2012), pp. 6–11. DOI: 10.1109/ISPCE.2012.6398297.

[84] *Toem Impulse*. 2017. URL: http://toem.de/index.php/projects/impulse (visited on 2017-05-01).

[85] *Tsummit*. 2017. URL: http://tsummit.org (visited on 2017-03-03).

[86] TuGraz. *Institute for Technical Informatics*. 2017. (Visited on 2017-02-05).

[87] *United States. Dept. of Defense : Electronic Reliability Design Handbook*. Electronic Reliability Design Handbook Bd. 1. Department of Defense, 1988.

[88] Verification Academy. *Coverage Cookbook*. 2012.

[89] J. Vidal; F. De Lamotte; G. Gogniat; P. Soulard; and J.-P. Diguet. “A co-design approach for embedded system modeling and code generation with UML and MARTE.” In: *2009 Design, Automation & Test in Europe Conference & Exhibition* (2009). ISSN: 1530-1591. DOI: `10.1109/DATE.2009.5090662`.

[90] WIRED. *The Jeep Hackers Are Back to Prove Car Hacking Can Get Much Worse.* 2016. URL: `https://www.wired.com/2016/08/jeep-hackers-return-high-speed-steering-acceleration-hacks/` (visited on 2017-02-05).