modelling

MDPI

Article

Software Architecture Modeling of AUTOSAR-Based Multi-Core Mixed-Critical Electric Powertrain Controller

Sakthivel Manikandan Sundharam [1], Padma Iyenghar [2,3,*] and Elke Pulvermueller [3]

[1] Delphi Technologies, 4940 Luxembourg, Luxembourg; sakthivelmanikandan@gmail.com
[2] Innotec GmbH, Erlenweg 12, 49324 Melle, Germany
[3] Software Engineering Research Group, University of Onsnabrueck, Wachsbleiche 27, 49090 Osnabrück, Germany; elke.pulvermueller@informatik.uni-osnabrueck.de
* Correspondence: padma.iyenghar@innotecsafety.com

Abstract: In this paper, we present a transition journey of automotive software architecture design from using legacy approaches and toolchains to employing new modeling capabilities in the recent releases of Matlab/Simulink (M/S). We present the seamless approach that we have employed for the software architecture modeling of a mixed-critical electric powertrain controller which runs on a multi-core hardware platform. With our approach, we can achieve bidirectional traceability along with a powerful authoring process, implement a detailed model-based software architecture design of AUTOSAR system including a detailed data dictionary, and carry out umpteen number of proof-of-concept studies, *what-if* scenario simulations and performance tuning of safety software. In this context, we discuss an industrial case study employing valuable lessons learned, our experience reports providing novel insights and best practices followed.

Keywords: model-driven software engineering; software architecture modeling; systems engineering; modeling tool; best practices; electric vehicle powertrain; AUTOSAR

Citation: Sundharam, S.M.; Iyenghar, P.; Pulvermueller, E. Software Architecture Modeling of AUTOSAR-Based Multi-Core Mixed-Critical Electric Powertrain Controller. *Modelling* **2021**, *2*, 706–727. https://doi.org/10.3390/modelling2040038

Academic Editors: Ludovico Iovino and Amleto Di Salle

Received: 23 September 2021
Accepted: 2 December 2021
Published: 4 December 2021

1. Introduction

In the last decade, Model-Driven Architecture (MDA) introduced by the Object Management Group (OMG) is considered as the next paradigm shift in software and systems engineering. Model-driven approaches aim to shift development focus from programming language codes to models expressed in proper domain-specific modeling languages. Thus, models can be understood, automatically manipulated by automated processes, or transformed into other artifacts. However, one must admit that the shift from model-based (models used as mere diagrams) to a completely model-driven methodology (models used as central artifacts) in real-life projects in the industry has not yet taken place [1]. Towards this direction, we touch upon the advances in model-driven engineering solutions for software architectures in automotive domain. In this paper, we discuss the state-of-the-practice in automotive organizations, the main pitfalls in the state-of-the-practice and our proposed approach. Based on our real-life project experiences involving development of electric powertrain software for an OEM (in the last twelve months), we present an industrial case study employing valuable lessons learned, our experience reports providing novel insights and best practices followed.

1.1. State-of-the-Practice in Automotive Organizations

The global Electric-Vehicle (EV) industry continues to expand rapidly, bolstered by the increase in demand for fuel efficient, high performance and low emission vehicles. Such an increase in supply-chain complexity is paralleled by the increasing complexity of software implementations [2]. Keeping this in mind, the Automotive Open System Architecture (AUTOSAR) [3] has been created as a worldwide development partnership of vehicle manufacturers, suppliers, service providers and companies from the automotive

electronics, semiconductor, and software industries. To achieve the technical goals of modularity, scalability, transferability, and function reusability, AUTOSAR provides a common software infrastructure based on standardized interfaces for the different layers. While doing so, AUTOSAR employs component-based software architecture, for the design and implementation of automotive software systems. In this context, a majority of the state-of-the-practice in the automotive Original Equipment Manufacturers (OEMs) for AUTOSAR-based, model-driven Embedded Software Engineering (ESE) is split between two main domains. They are the Unified Modeling Language (UML) [4]/Systems Modeling Language (SysML) [5] domain and the Matlab/Simulink (M/S) domain [6].

In the race to provide model-based tool support (e.g., architecture design, automatic code generation) for AUTOSAR-based ESE in the UML/SysML domain, UML tools such as Enterprise architect [7] and IBM Rhapsody [8] emerged as front-runners. For instance, AUTOSAR-related SysML profiles for the architectural description of an AUTOSAR model that uses the native AUTOSAR concepts is supported by Rhapsody. However, the state-of-the-practice in the automotive industry is that, UML/SysML is used only at higher abstraction levels.

For instance, UML/SysML is used to create descriptive UML models that describe the overall software and system architecture (cf. Figure 1(1)). During the last decade, we are one among the several automotive organizations who have followed such an approach for our automotive OEM client projects. This involved a considerable amount of time spent in training personnel in the UML/SysML domain. Additionally, it resulted in significant investment costs in tools and training, without satisfactory results.

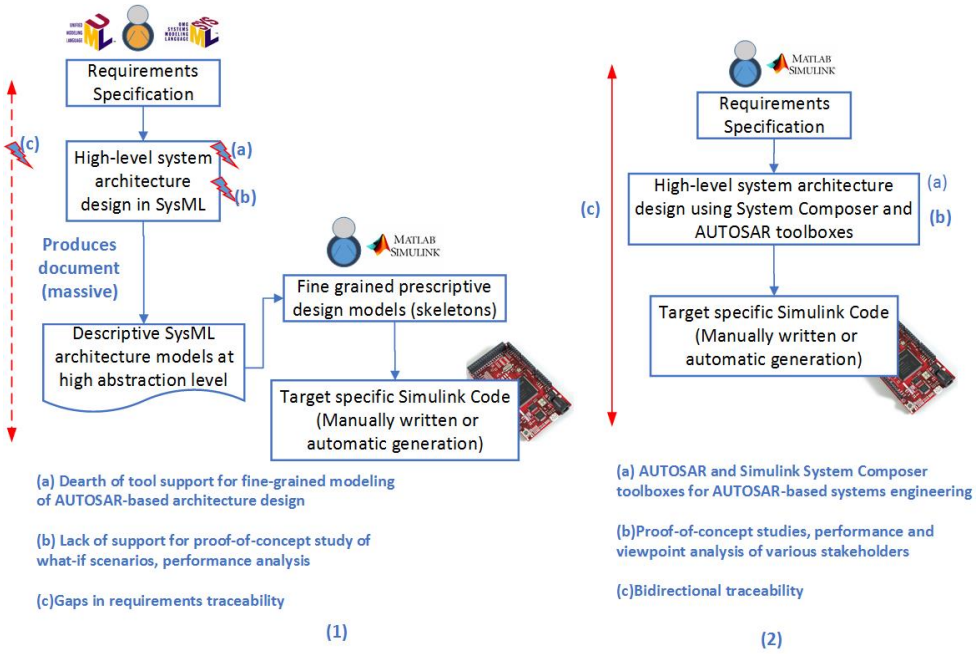

Figure 1. Steps for AUTOSAR-based architecture modeling of automotive embedded software: (**1**) State-of-the-practice steps followed by automotive OEMs and (**2**) Our approach (in the last twelve months) in a real-life automotive project of a multi-core, mixed-critical electric powertrain controller

Further, even though such models are at a higher abstraction level, the document containing these models is massive. From these diagrams, a more fine-grained architecture of prescriptive models are produced manually, but not using UML/SysML. These high level diagrams are used to automatically create, for instance, Simulink skeletons where Simulink code is manually written or automatically generated. Although this is the case in the majority of cases in the automotive industry, some even have simpler processes with less use of UML.

Moreover, automotive software is loaded with non-functional requirements (e.g., timing and memory constraints). During the software architecture design of such systems, numerous such parameters need to be taken into consideration, optimized, and fine-tuned (e.g., CPU load vs. Safety constraint). However, UML/SysML tools lack support for such trade-off analysis among non-functional requirements (e.g., performance parameters). Further, in the development of safety critical automotive software, it is imperative to have bidirectional traceability. Though some island solutions exist for achieving traceability between UML/SysML tool and requirement management tools, they do not support bidirectional traceability as a comprehensive solution. Thus, in this competition towards tool support for automotive software/system architecture design, UML/SysML tools are treated only as diagram tools.

Main Pitfalls in State-of-the-Art Practice

The approach (Hereafter referred to as legacy approach) described so far, of employing two different modeling domains for AUTOSAR-based architecture modeling of automotive embedded software, which is by nature highly complex, is not a seamless solution (cf. Figure 1(1)) and has the following main pitfalls, namely;

- Dearth of *feature rich* tool support for detailed modeling and software architecture design of AUTOSAR-based systems (e.g., no detailed data dictionary);
- Lack of support (e.g., simulations) for proof-of-concept study of *what-if* scenarios and trade-off analysis of performance requirements (e.g., timing, memory, energy, and safety) during early architectural design;
- Lack of bidirectional requirements traceability and a comprehensive authoring process.

In our real-life project experience involving development of electric powertrain software (cf. Section 1.3) for an OEM.Initially, we have employed the legacy approach in Figure 1(1) and faced the challenges listed above.

1.2. Proposed Approach

On the other hand, new modeling capabilities were introduced in the recent releases of M/S (ca. early 2019). With the above gaps in the background, we have made a transition journey of *Software architecture modeling of AUTOSAR-based, multi-core, mixed-critical electric powertrain controller* from using the legacy approach (cf. Figure 1(1)) to the approach illustrated in (cf. Figure 1(2)). In the approach in Figure 1(2),

- Employing AUTOSAR specific block set in the latest releases of M/S, we were able to implement a detailed model-based software architecture design of AUTOSAR system including a detailed data dictionary;
- Making use of the Simulink System Composer toolbox in the recent releases of M/S, we have created *custom-defined profiles* to support trade-off analysis of safety software (e.g., safety vs. timing);
- Using powerful simulations supported by M/S toolboxes, we were also able to carry out umpteen proof-of-concept studies and *what-if* scenario simulations;
- With the help of a state-of-the-art requirements management tool plugin (Polarion for Simulink [9]) and custom-defined scripts, we achieved seamless bidirectional traceability between requirements and architecture design. This is also supported by a comprehensive authoring process.

In this context, this paper presents an industrial case study employing valuable lessons learned, our experience reports providing novel insights and best practices followed. The remainder of the paper is organized as follows. Following this introduction, a brief introduction about electric powertrain of a motor vehicle (running example) is provided in Section 1.3. Background and related work is presented in Section 2. In Section 3 we discuss the pitfalls in the legacy approach and best practices followed in the new approach. In Section 4, we discuss the lessons learned in the new approach. A summary and conclusion is provided in Section 5.

1.3. Electric Powertrain Example

A brief background about an electric powertrain is provided in this section, which will be used as a running example throughout this paper. The *powertrain* of a motor vehicle comprises the main components that generate and deliver power (i.e., provides power to the vehicle). The most recent developments in powertrain are driven by the electrification of it in multiple components. All-electric vehicles eliminate the Internal Combustion Engine (ICE) altogether, meaning they rely solely on electric motors for propulsion. The main components in the electric powertrain are battery pack (providing Direct Current (DC)), electric motor, on board charger, battery, thermal management systems and AC-DC/DC-DC converters [10]. The main components in the electric power train are shown (Source: https://evreporter.com/ev-powertrain-components/ (accessed on 1 December 2021). Interested readers may refer to this source for details) about Figure 2.

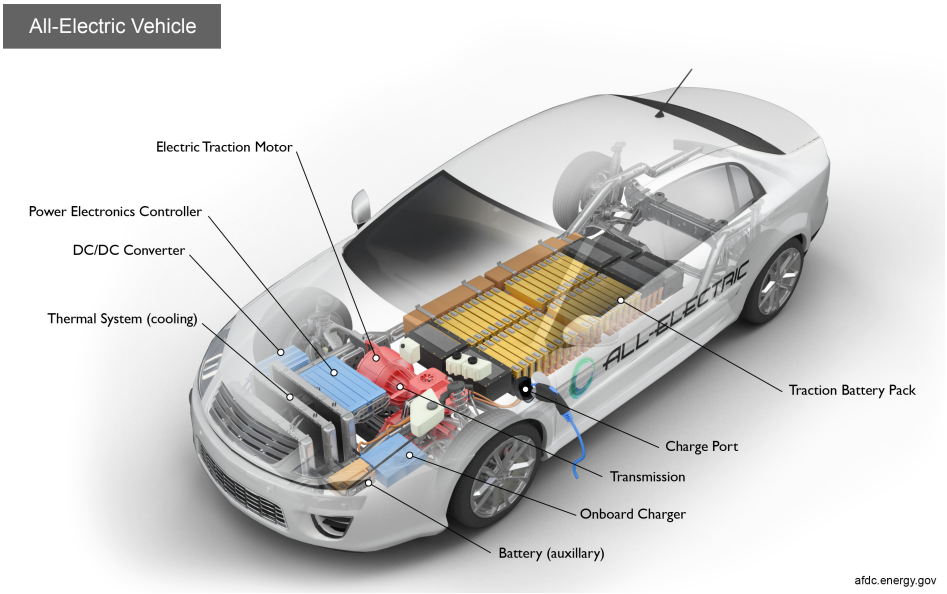

Figure 2. Typical components in an all electric vehicle.

The electric powertrain software is of mixed Automotive Safety Integrity Level (ASIL) criticality levels [11], ranging from A to D (D-highest criticality). Further, this software runs on multi-core hardware platform (Example: https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/ accessed on 1 December 2021), which is being used increasingly in the automotive sector. The electrical powertrain system is driven and controlled by a liquid-cooled high voltage inverter. The inverter control software of the powertrain was initially developed using the legacy approach and then switched to using M/S in our live/client project. Examples from this project are used in Sections 3 and 4. However, the figures used in this paper are not original work items from our live project, but an equivalent representation of them.

2. Background and Related Work

Background and related work pertaining to modeling alternatives for automotive embedded software systems is described in Section 2.1. In Section 2.3, background and related work pertaining to Requirements Management (RM) is presented.

2.1. Modeling Automotive Embedded Software Systems

Today, there are up to 100 Electronic Control Unit (ECU) (An embedded system that controls one or more of the electrical systems or subsystems in a vehicle) developed by different manufacturers, integrated in a modern car, wherein the ECU software is also becoming more and more complex. To address the increasing complexity in development of such systems, Model Driven Development (MDD) [12], is considered as the next paradigm

shift. In MDD, the requirements are specified as models at a higher abstraction level (e.g., using UML/SysML, M/S). They are then refined, starting from higher and moving to lower levels of abstraction, via model transformations. Note that, apart from UML/SysML and M/S domains, the Rubus Component Model (RCM) and EAST-ADL [13] are among other solutions used by a smaller community in the vehicular domain [14].

2.2. Modeling and Simulation-Based Techniques Using UML and SysML

In the direction of adoption of a model-driven approach and the use of simulation-based techniques, significant effort has been spent in the last decade for easing the development and the simulation of complex systems using SysML models such as [15–20] to mention a few.

In [15], a SaaS-based automated framework to build and execute distributed simulations from System models is presented. In [16], a model-driven distributed simulation engineering approach is presented. The proposed approach is framed around the development process defined by a "DSEEP" standard, as applied to distributed simulations based on the high-level architecture and is focused on a chain of automated model transformations. Case studies used in [16] are based on the example application of the proposed approach to development of a high-level architecture based distributed simulation of a space system.

Similarly, Ref. [17] aims to remedy the deficiencies regarding the missing representation of hardware and mechanics artifacts within the E/E-System design which are carried out by common approaches utilizing tools, such as Enterprise Architect or Artisan Studio, to model the E/E-System design in SysML or a kind of UML2 profile. Therefore, a model-based domain-specific language was developed that describes the system in a more comprehensive way. Additionally, the proposed approach claims to make it easier for domain experts, who are not that familiar with UML or SysML, to create an architectural design. Furthermore, the already existing SysML models are not ignored in the presented methodology, but supported through a translator, which converts the DSL model into a SysML representation. In [18], the main objective is the integration of safety analysis in a SysML-based systems engineering approach in order to make it more effective and efficient. The proposed methodology named safety integration in systems engineering (SafeSysE) is applied to a real case study from the aeronautics domain: electromechanical actuator (EMA). A SysML-based methodology for mechatronic systems architectural design is proposed in [19]. This methodology consists of two phases: a black box analysis with an external point of view that provides a comprehensive and consistent set of requirements, and a white box analysis that progressively leads to the internal architecture and behavior of the system. Although [15–19] are extensive studies carried out using SysML domain, none of them consider a user case on high level automotive architecture design and implementation using AUTOSAR.

However, in [20], a practical approach of employing SysML for embedded automotive systems for specific Valeo product lines is elaborated. This paper clearly identifies that, while SysML is a leading topic for Systems Engineering in all domains, there is no pragmatic implementation of SE for automotive embedded systems and products. Although the paper itself provides only a proposal and the paper claims no theoretical novelty, neither does it claim to be on the leading edge of SysML modeling. Finally, the paper proposes that it may help leverage synergies of cross-domain expertise and needs. Though such proposals are more than a decade old, until now there has been no concrete studies concentrating on application of SysML for concrete real-life industrial use cases in embedded automotive domain, especially employing the very relevant AUOTSAR standard for the automotive domain, which is the main topic addressed in this paper.

The aim of reference [21] is to start laying the foundations of modeling and simulation-based systems engineering, which may lead to a textbook in the future. In particular, topics such as methods to support system modeling (chapters 2–4), domain-specific languages (chapters 6 and 10) and model-driven simulation engineering discussed (chapters 6 and

7) in this book are relevant to the work presented in this paper. Chapters 2–4 discuss the process of modeling a system and how modeling and simulation can be applied in this process and how to optimize the process. For example, chapter 2 proposes the alignment of processes, methods, and means that are already used by both system engineers and simulation practitioners. Aligning the processes and using system architecture artifacts as a common repository result in a consistent model that can be executed as a simulation, providing additional numerical insight and eventually resulting in high-quality, trustworthy systems as required by the customer. This idea is relevant to our work; however, the entire system architecture design is carried out by a system architect in our client project. Hence, there is no significant need for alignment of process and system architecture artifacts. Whereas, in our approach the system architect is able to carry out such simulations (e.g., see chapter 3.2 early performance analysis) for instance to evaluate what-if scenarios and provide early performance results, resulting in high-quality and trustworthy systems as required by the customer as mentioned in chapter 2 of [21]. Moreover, it is important to note that, in our work, we concentrate on the capabilities of UML and M/S for detailed architecture modeling of the use case at hand and the best practices involved in this, rather than the process involved.

In chapter 6 in [21], the authors address the use of distributed simulation techniques to model the inherently distributed architecture of a complex system. The chapter proposes a method that exploits principles, standards, and tools introduced in the model-driven engineering filed and supports automated generation of high-level architecture-based distributed simulations from system models specified by the use of SysML. Further, in chapter 7 in [21], model-driven methods to enable simulation-based analysis of complex systems is analyzed. For instance, these chapters discuss how the latest research insights drive the ideas forward to use models and simulation to drive the engineering of systems. For instance, they deal with the collaborative and concurrent design of spacecrafts with respect to modeling and simulation. Looking at the systems engineering life cycle, they discuss expectations and advantages of model-driven design approaches, and highlight the potential of formalized design for continuous verification and validation. The chapter closes with a vision on promising ideas to integrate modeling and simulation as a fast and easy-to-use means of supporting the spacecraft design process. This is also very relevant work, where we address the point of early performance analysis to support continuous design refinement and early performance analysis. In our work, this aspect is addressed in chapter 3.2, where we elaborately demonstrate about analysis of what-if scenarios in early model-based performance analysis with real-life examples. In summary, the reference [21] provides elaborate and significant contributions pertaining to modeling and simulation-based systems engineering. However, none of the chapters in [21] deal with a specific use case study of software architecture modeling of AUTOSAR-based automotive real-life use case, which is one of the main contributing topics of our work.

In [22], a guideline for interoperability modeling, model transformation and simulation is proposed for Supply Chain of Information and Communication Technology (SC-ICT) sector. The authors claim that the approach fits in with the industry 4.0 principles, especially it takes an interest in Cyber Physical Systems (CPS), data analysis, and IoT. Although Model-Driven Interoperability (MDI) is a topic of its own, it would be interesting to compare any take away from this work in [22] to a real-life AUTOSAR-based project such as one we have described. For instance, the article [22] provides some perspectives on the interest of MDISE in the frame of future CPS. Similarly, for migration from one architecture to another, virtual organizations using MDSE described in [23], will help a continuum in modeling and simulations. Thus, the proposed approach in [23] and the proposed library of model templates presenting classical activities of data exchange for CPS and model transformation engine for the creation of a reference framework in [22], may be extended and employed for simulation studies of real-life projects discussed in our paper.

2.2.1. UML/SysML Tools for AUTOSAR-Based Architecture Modeling

UML/SysML tools such as EA, Rhapsody (and others) are well established in the software development process. Sometimes the modeling guidelines are following a custom modeling, e.g., with specific profile. For instance, AUTOSAR-related SysML profiles for the architectural description of an AUTOSAR model that uses the native AUTOSAR concepts is supported by Rhapsody. However, a complaint about these profiles, from system architects in the automotive industry, is that, these profiles are not comprehensive enough to capture all the design artifacts of the AUTOSAR-based software system. Thereby, in the state-of-the-practice in the automotive industry, the UML/SysML tools are used only at higher abstraction levels. They are mostly used to create descriptive UML models that describe the overall software and system architecture. These higher level models are then used to create fine-grained prescriptive models in other development tools such as M/S [24,25].

UML supports the use of specific profiles for performance analysis (e.g., time, energy, and safety), apart from generic system and software modeling. Some examples of employing UML for MDD and examining quality properties such as timing and reliability are available in [26]. In [27], an approach towards early synthesis of timing models in AUTOSAR-based automotive embedded software systems is discussed. In this work, the workflow proposes to create timing annotated AUTOSAR-based models in Rhapsody using AUTOSAR-timing specifications and synthesis of a timing analysis model corresponding to the AUTOSAR-design model. However, the timing analysis itself is to be carried out in some other tool, not Rhapsody. Except for the above work, such performance aspects pertaining to AUTOSAR-based systems are not found in the literature. This is because the UML/SysML tools are primarily not equipped to provide fine-grained descriptive models, which may be used to carry out such performance analysis in the AUTOSAR-domain. Thus, these solutions may be best suited for generic-UML/SysML modeling and not for AUTOSAR-based automotive embedded software architecture design and analysis.

2.2.2. Matlab/Simulink (M/S) Tool for AUTOSAR-Based Architecture Modeling

M/S is a popular example for a modeling tool with non-UML modeling language. It is an established tool in the industry, including the automotive domain [28,29], especially focusing on physics-based modeling and control loops. The modern automotive comprising hybrid powertrains are complex multi-domain systems. As M/S provides extensive model-based design capabilities together with block sets for simulation, it appears to emerge as a sought after tool to capture electrical, mechanical, hydraulic components together with the control systems in a single environment [30].

In this context, new modeling capabilities such as AUTOSAR Blockset [31] and Simulink System Composer [32] toolboxes, in the recent releases of M/S, add sophisticated systems engineering capabilities for AUTOSAR-based systems. However, there are no studies published yet, in general, on the applicability and usage of these new capabilities. In this context, this paper is a first-of-its-kind in employing new capabilities of M/S toolboxes, to a state-of-the-art, real-life project on AUTOSAR-based electric vehicle powertrain architecture modeling, and outlining the lessons learnt and best practices involved.

2.2.3. AUTOSAR Framework

A promising approach in the automotive domain is the standardization of the software architecture used in ECU development [33]. A comprehensive and well- established solution used in the automotive sector is the AUTOSAR standard. It emphasizes to shift the ECU development from an ECU-centric approach to a functionality-based approach.

AUTOSAR uses a component-based, layered software architecture, with central modeling elements called *Software Components* (SWCs or SW-Cs). The SWCs describe a completed, self-contained set of functionality. The AUTOSAR methodology describes various steps, namely, *System configuration, ECU configuration and component implementation* involved in the development process. It also describes the artifacts created and interchanged between the steps. In between these steps, the ARXML file format [34] is used for the exchange

of development artifacts, which is an XML-based file format. The functionality-based approach aims to specify the functions of the complete vehicle first in the so-called *system configuration*, and afterwards extract specifications for the suppliers to implement an ECU. This way, the automotive software can be interchanged on a function level instead of the ECU level, which increases its reusability.

The various components of the AUTOSAR framework are illustrated together with the mapping of software components to ECUs, in the system configuration step, in Figure 3. The software components (seen at the top of Figure 3, e.g., *SW-C1*) are used to structure the AUTOSAR model and group functionality into individual components. These components can be connected together, oblivious of the hardware they will be running on. This is handled by the *Virtual Function Bus* (VFB), which provides an abstraction layer for the SWC to SWC communication. Components distributed over different ECUs however, may use the network bus for communication. This is determined automatically by the *Run-Time Environment* (RTE), which is a communication interface for the software components.

The lower part of the Figure 3 represents the mapping of ECUs to SW-Cs in the system configuration step. Here, the ECUs 1, 2, . . . , n are seen communicating over a network bus (e.g., FlexRay, CAN). In each ECU (e.g., ECU 1 in lower part of Figure 3), the RTE provides interfaces between SW-Cs (e.g., AUTOSAR SW-C 1 and AUTOSAR SW-C 2 in ECU 1) and between SW-C and basic software (BSW). Furthermore, it provides the BSW services (as API abstraction) to SW-C.

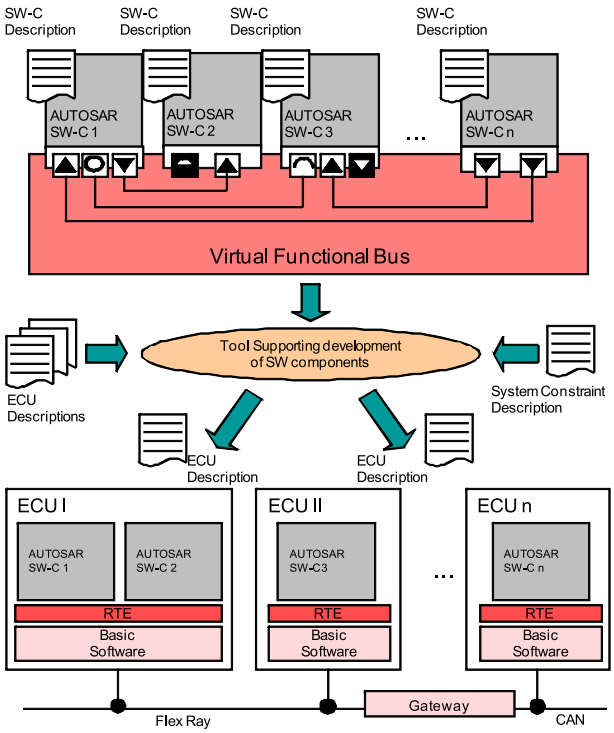

Figure 3. Mapping of software components to ECUs.

The underlying software functions which implement the given requirements are contained inside the SW-Cs. These are later on implemented manually by the software developers. The RTE and *Basic Software* (BSW) which are provided by third-party AUTOSAR software vendors are at the disposal of the developer for communication and hardware abstraction. The inner functionality of the application and sensor/actuator SWCs is defined in *Internal Behavior* elements. They encapsulate *Runnable Entities*, which correspond to atomic functions on the code level that are implemented later in the development process. In [30], a study on design and analysis of battery-aware automotive climate control for electric vehicles is presented. However, a discussion on the best practices involved during

AUTOSAR-based modeling of system architecture design of electric vehicle components (e.g., powertrain) is not available in the literature.

2.3. Requirements Management (RM) Tools

Requirement Management (RM) tools like DOORS [35], Polarion [9] and Eclipse-based ProR [36] are some state-of-the-art RM tools. These tools are much more powerful than textual editors for managing requirements. In these tools, requirements are handled like objects in Object Oriented Programming (OOP). Further, the Requirements Interchange Format (ReqIF) [37] has become a standard in the requirements domain for exchanging requirements between RM tools. Within ReqIF files, requirements are stored as *SpecObjects*, requirements types as *SpecObjectTypes* and links as *SpecRelations*. In practice, ReqIF even allows that requirements on different refinement levels, like user requirements and a system specification, is managed by different partners in different RM tools.

However, establishing traceability beyond requirements into design, implementation, and verification artifacts can become a challenging task. For instance, during modeling of architectural design of an automotive use case, the requirements are in RM tools (Doors, Polarion) and the architectural design is in tools such as M/S, Rhapsody, or Enterprise Architect. Thus specialized tool-plugins compatible with MDD tools are required to establish requirements traceability in both ways (forward and backward), also referred as bidirectional traceability.

Bidirectional traceability is a key notion of all process assessment and improvement models. Traceability helps impact analysis and change management—when requirements are changed, the affected work items are easier to identify. A *good* software tool should specify, interlink, analyze, manage, display, and report the requirements traceability, giving clear visibility throughout the development process. The tool can easily identify if a requirement is floating and unrelated, for example, if it is an orphan item due to a missing upstream requirement or is missing a coverage towards a downstream requirement. In a safety related project, fulfilling the requirements traceability can help decide whether the safety case for the product has been achieved [38].

Even though there exists RM-plugins for EA, such as [39], we have used EA as a software architecture modeling tool to draw UML/SysML diagrams and then publishing them manually to the requirements database. Hence, while employing the legacy approach, we have manually stored the UML/SysML diagrams in the Polarion RM tool. In our new approach, we have employed the Polarion extension for M/S [9] to establish bidirectional traceability and authoring. We were not able to practice bidirectional traceability between requirements to architecture, with the help of legacy tool chain as this was not well suited to AUTOSAR compliant architecture.

3. Pitfalls in Legacy Approach and Best Practices in New Approach

In this section, we deal with the shortcomings of the legacy approach (cf. Figure 1), in three main groups as discussed in Section 1. Accordingly, the shortcomings in *tool support for modeling AUTOSAR-based software architecture design* in the legacy approach and how they are overcome in the new approach (cf. Figure 1(2)) is discussed in Section 3.1. Next, a novel example of a performance trade-off study carried out using the M/S tool, to aid in system architecture design-decisions regarding trade-off analysis in configuring *program flow monitoring* is discussed in detail in Section 3.2. In Section 3.3, we discuss regarding bidirectional traceability.

3.1. Modeling AUTOSAR-Based Software Architecture

3.1.1. Using SysML/EA

Data dictionaries in embedded automotive applications are used for storing data elements and definitions supporting functional and software development. A robust model-based development process should be driven by a data dictionary that can interact with tools supporting a model-based environment and provide a linkage between different

stages of the development process. A data dictionary could significantly increase automation of the modeling, validation, and coding stages of the process [40]. Given the fact that, automotive systems are highly complex systems with ports and interfaces running to several thousands, it is intuitive to perceive that support for data dictionary in the system architecture design phase is imperative.

Modeling guidelines of AUTOSAR basic software (cf. lower part of Figure 3) using UML tools, such as EA, are provided by AUTOSAR [41]. Adhering to these guidelines and applying the legacy approach, we have initially developed the powertrain system architecture design in EA tool. An example of the motor control static architecture design created using SysML in EA is shown in Figure 4.

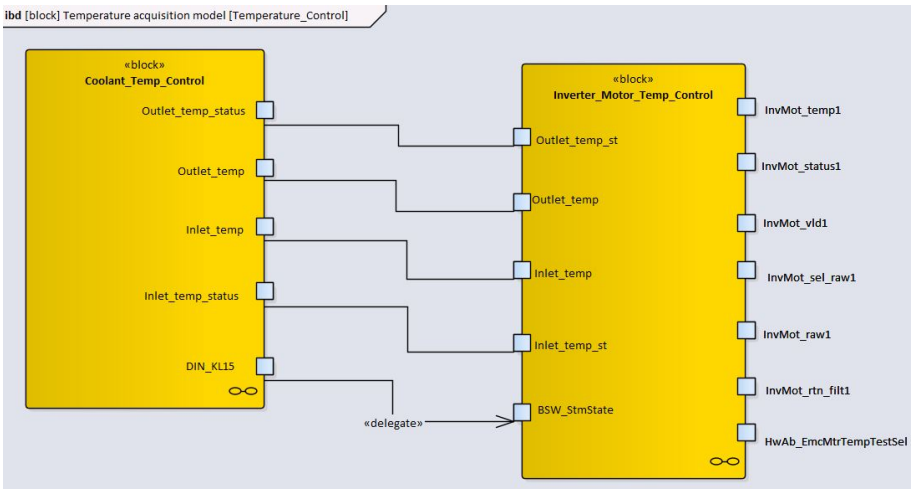

Figure 4. Architecture design using SysML in EA.

However, as seen in Figure 4, there are only names of the SysML blocks and ports listed. There is no data dictionary clearly defined. This is a big disadvantage to not only the system architect, but also for the entire software development team as they are unable to have critical software development data readily available together with the system architecture.

3.1.2. Using System Composer and AUTOSAR Blockset in M/S

On the other hand the same system architecture is modeled using the System Composer and AUTOSAR blockset in M/S tool in our new approach. One can clearly see the support for data dictionary in this example (highlighted in the insert of Figure 5).

All the port details, their type that must be implemented (boolean in this example) are available. Thus, we were not only able to produce fully AUTOSAR compliant system/software architecture designs, but also their datasets well-defined and clearly usable for the code development activity.

With the help of examples in Figures 4 and 5, we have demonstrated that only a static architecture model was possible to be defined without any interfaces name and no data dictionary in SysML/EA tool. This implies that the modeling capabilities of SysML using EA tool may be suitable for generic architecture modeling and not for highly detailed and fine grained AUTOSAR-architecture modeling (i.e., viewpoint of a software architect). As per the Automotive Software Performance Improvement and Capability determination (ASPICE) process model, we have learned clearly that even at architectural modeling level (in ASPICE terminology SWE.2), it is necessary that an architect defines partially or fully the data dictionary information. This is imperative to validate the software architecture during the validation phase called software integration test (in ASPICE terminology, SWE.5).

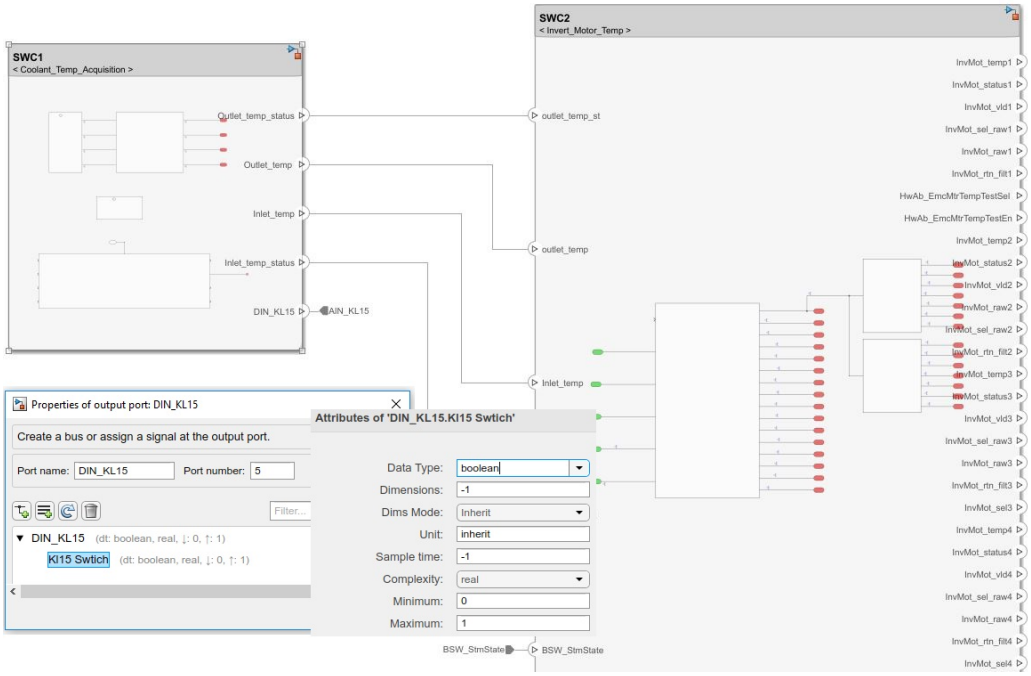

Figure 5. Architecture design using System Composer and AUTOSAR blockset in M/S with data dictionary.

The validation engineer is able to write the test cases if, and only if, the architect provides the data dictionary part during SWE.2. For instance, during software integration test, the validation engineer is mandated to check for valid data range. Clearly, this is feasible only when the modeling environment provides such a possibility. Otherwise, this has to be documented manually using the requirements database. If this is the case, the validation engineer often follows-up with the software architect for each and every test case he writes. It is obvious that this is not an efficient solution, which was our case when employing the legacy approach (cf. Figure 1). Thus, the best practices in the above context are:

- **Best Practice-1 (BP-1)**: Modeling of AUTOSAR-based system architecture using AUTOSAR blockset together with System composer toolbox in recent releases of M/S;
- **BP-2**: Creating fine-grained AUTOSAR architecture models using Simulink System Composer data dictionary support.

3.2. Early Model-Based Performance Analysis

In electric vehicles, the design is now arranged around a battery and one or more motors rather than an internal combustion engine. Thus, the design space is much larger and more varied than before. Engineers are required to approach analysis and simulation from the system level because of the way different physics affect each other in these vehicles. This means that the engineers need to look at the complexity more at the systems level and bring more and more systems together in the earlier design process.

For the architecture design of complex system such as electric powertrain, choosing appropriate design parameters and making early design choices, optimization of various parameters, such as energy vs. timing and CPU load vs. safety constraint, is an imperative and challenging task. These are some examples of *what-if* scenarios which may be analyzed during early phases of system architecture modeling and design using our proposed approach. In the following, we introduce the concept of a functional safety mechanism [11] in AUTOSAR-based systems, namely *program flow check*. We discuss a scenario of *timing overhead calculation* in *program flow check* (also interchangeably referred as *logical supervision*) involving the AUTOSAR-Watchdog manager (WdgM) [42].

In industry practice, during the early system design phase, the system architect follows a thumb rule (e.g., of assigning one monitoring call per supervised function), for configuration of program flow check. As there is no tooling to determine the optimal and/or feasible number of checkpoints, this practice of rule of thumb leads to a costly re-work of system architecture design. For instance, the significant timing overhead introduced by the monitoring calls are only detected later in testing and timing analysis phases (e.g., when the supervised function misses a timing deadline). We illustrate, with a simple example, an effective evaluation of this *timing overhead* introduced by the monitoring functions during program flow check, using both M/S basic functions and model-based configuration using the *profiles* introduced in the new releases of M/S in the *System Composer* toolbox.

3.2.1. AUTOSAR Watchdog Manager (WdgM)

The Watchdog Manager is a basic software module at the service layer of the standardized basic software architecture of AUTOSAR [42]. The WdgM is able to supervise the program execution, abstracting from the triggering of hardware watchdog entities. When the WdgM detects a violation of the configured temporal and/or logical constraints on program execution, it takes a number of configurable actions to recover from this failure. Here, we take an example of *logical supervision* which is a fundamental technique for checking the correct execution of embedded system software.

3.2.2. Supervised Entities, Checkpoints and Program Flow Graph

Supervised entities are the units of supervision for the WdgM module. Important places in a supervised entity are defined as *checkpoints*. The code of supervised entities calls the WdgM when they have reached a checkpoint. For every Logical Supervision, there is a graph of checkpoints connected by transitions. The graph abstracts the behavior of the supervised entity for the WdgM module. The checkpoints and the transitions form a *program flow graph* for the supervised entity. The graphs are defined statically during the configuration of the WdgM, thereby denoting the execution order of the monitored code. At runtime, the supervised entities call the WdgM API when they have reached a checkpoint. Then, the WdgM verify that the sequence of the received checkpoints, starting with any initial checkpoint and finishing with any final checkpoint, is correct. Thus, the WdgM traces the checkpoints using this graph, determining the validity of an execution sequence. Please note that, in AUTOSAR-basic software configuration together with system architecture design, the configuration of checkpoints is carried out together with tools such as DaVinci configurator [43].

3.2.3. Granularity of Checkpoints

This is not fixed by the WdgM. For instance, this is left to be determined by the system architect during the system design phase. In the control flow of a supervised entity, when every checkpoint is reached, the activity is reported to WdgM, resulting in a (access) time overhead. Few coarse-grained checkpoints limits the detection abilities of the WdgM. High granularity of checkpoints causes a complex and large configuration of the WdgM, thereby also introducing a significant overhead in time. Thus, it is imperative at the early system design stage to configure a (pareto-) optimal number of checkpoints in the system architecture. This activity needs to be performed, such that the granularity of checkpoints does not limit the detection ability of the WdgM and at the same time does not result in a significant timing overhead leading to deadlines being missed by the supervised entity. Missing deadlines, in this case, may lead to a compromise of the systems functional safety aspect (e.g., contributing to missing the Fault Tolerant Time Interval-FTTI [11]).

3.2.4. Battery Management System Example

In the electric powertrain, a *Battery Management System* component (cf. Figure 6) integrates different components capable of calculating battery parameters including cell voltage, battery temperature, and battery current. These parameters must be measured

and controlled to improve the capability of state tracking in real-life applications [10]. In this context, let us consider an example of a temperature control *module* (also known as *task*) in the *battery management system* in Figure 6 comprising several *functions* (also known as *runnable*). Assigning one checkpoint per runnable (i.e., functions in the task), results in eight checkpoints (Note that an entry and exit checkpoint is always required here, which is a limitation of the configuration tool) as seen in Figure 7(2). The possible *correct* program flow sequences are shown in Figure 7(3). Note that the transitions T1…T9 specify the program flow between the checkpoints. The transitions for possible execution sequences in Figure 7((3.1)–(3.3)) are represented in a matrix as seen in Figure 7(4). For this example, let us consider that each checkpoint access requires an access time of 5 ms.

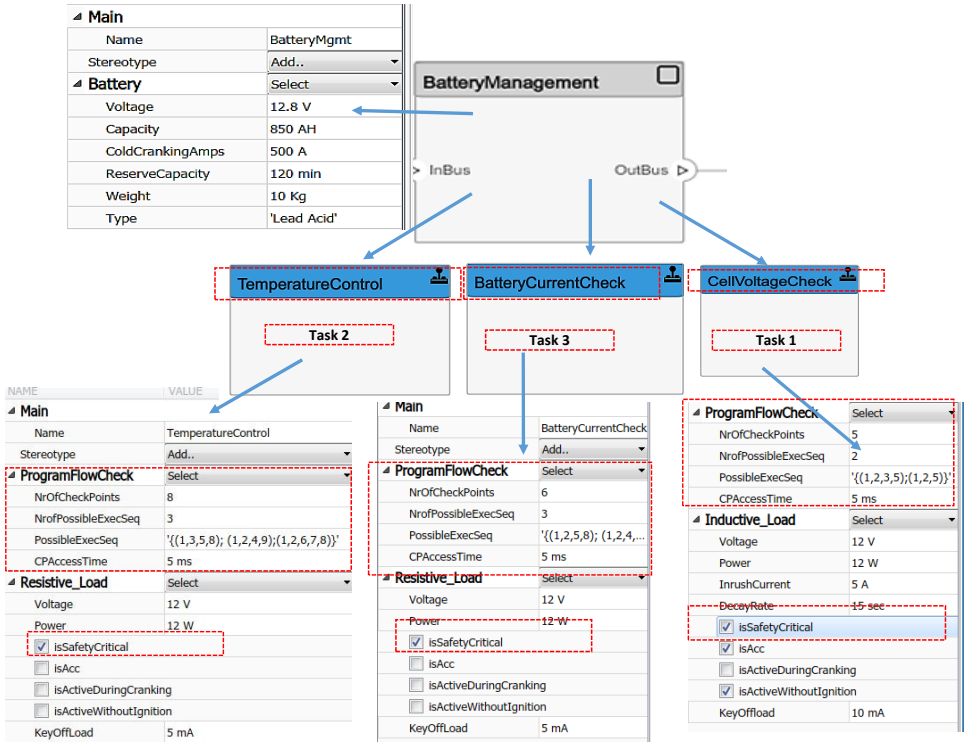

Figure 6. Model-based configuration of program flow check parameters with *ProgramFlowCheck* profile.

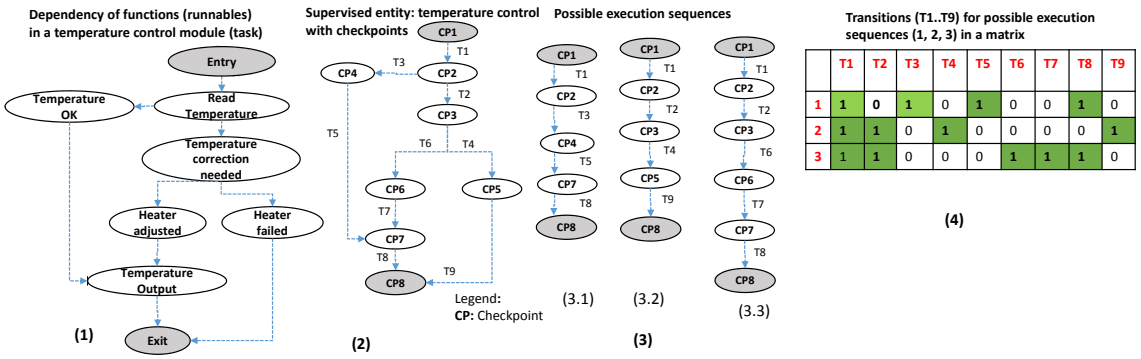

Figure 7. Program flow monitoring configuration of a temperature control module (**1**): Functions (runnables) in this module (task), (**2**): Possible checkpoints of the supervised entity and (**3**): Possible execution sequences of the supervised entity and (**4**): Dependencies of functions/CPs-denoted by transitions among them; for possible (correct) execution sequences.

3.2.5. Estimation Using Matlab Script

With a powerful computation engine and data structures, M/S enables such calculation in a straight forward way. The estimation of the total checkpoint access time of execution sequences in Figure 7((3.1)–(3.3)) is calculated as 25 ms, 25 ms, and 30 ms, respec-

tively. Note that for such a small example, this can also be calculated manually. However, for a complex architecture design involving several hundreds of components, a script written in Matlab involving its powerful computation engine, may be employed. In our simple Matlab script, we have used a n∗n matrix to represent the checkpoints in each correct program flow sequence to estimate the total CP access time for this temperature control task.

3.2.6. Model-Based Estimation

For the aforementioned example, we have also carried out a model-based estimation of the CP access times using custom-defined profile and stereotypes employing the newly introduced SC [32] toolbox. For this example, we have created a *ProgramFlowCheck* profile, which is a custom-defined profile comprising stereotypes such as *NrOfCheckPoints*, *NrOfPossibleExecSeq*, *PossibleExecSeq* and *CPAccessTime*, as seen in Figure 6.

A profile created using SC toolbox can be exported and saved in an XML file for import in any project. Using the input parameters for three different tasks in the example described above, namely, *TemperatureControl*, *BatteryCurrentCheck* and *CellVoltageCheck* we have estimated the *checkpoint access time* overhead per task and per program flow sequence of the tasks as seen in Figure 8. The *iterate* method of the specification API in system composer toolbox is used to iterate through each element of the model and run analysis using the stereotype properties. This is a highly extensible approach and can be employed iteratively changing configuration parameters, if required (e.g., CP access time from hardware measurements/tests).

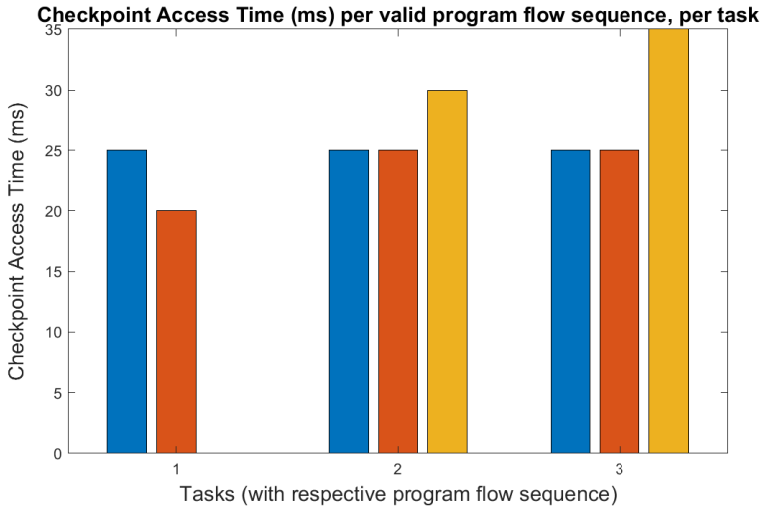

Figure 8. CP access time per program flow sequence, per task.

Determining the timing overhead at such an early stage (i.e., here during configuration itself) enables the system architect to make an informed decision about the trade-off between *meeting a timing deadline of a task* versus *enabling fine-grained program flow monitoring for tasks*. This is an extremely critical point to address during the system architecture design, as the granularity of the checkpoints and their overhead must not lead to a missed deadline such that the system is unable to move to a safe state.

In comparison to the above approach, in the legacy approach such custom-made profiles can be created and UML/SysML models can be annotated with performance attributes. However, the UML/SysML modeling tools lack features such as powerful computation engines. Hence, these annotated performance models have to be exported in one or the other format (e.g., Excel file, XML files, etc.) to performance analysis tools to carry out performance analysis. In the literature, synthesis of such performance models for performance analysis in specific tools can be found in [27,44,45].

3.2.7. Concept Phase of the Project—An Example

During the concept phase of this client project, we were asked to present and discuss with OEM our technical concepts around AUTOSAR and multi-core challenges. Some challenging concepts include the Diagnostics Event Management (DEM) between cores and NVM storage between different safety level cores (QM to ASIL D) concepts. All these concepts are related to non-functional properties such as functional safety and timing constraints. With the custom metamodel support by system composer and AUTOSAR basic software blocks, we were able to model these non-functional properties (e.g., by specifying them in stereotypes and profiles in System Composer), similar to the program flow check example in Section 3.2.6.

Moreover, M/S also supports extensive simulation and analysis. In the case of legacy approach, this was a missing feature. In the newly adopted approach, simulation and analysis facilitated, to a greater extent, in determining the non-functional property defects at an earlier stage of the development. Finally, the whole approach helps to generate a customized Software Architecture Document more conveniently and that drives the software design and further stages of development life-cycle. Thus, the best practice in the above context is:

- **BP-3**: Early model-based performance and trade-off analysis of non-functional requirements using custom-defined profiles (e.g., employing M/S-SC toolbox [32]).

3.3. Bidirectional Traceability

ASPICE [46] is an internationally accepted process model that defines best practices for software and embedded systems development for the automotive industry. It provides guidelines to improve the software development processes and to assess suppliers, and is widely adopted by the automotive industry. In ASPICE, a measure of process performance data is to have defined input and output work products available, during every stage of development. Thus, adhering to ASPICE, during the model-based architecture system design, the input is the requirements specification and the output is a model-based architectural design. Further, one of the best practices listed by ASPICE is to establish bidirectional traceability between software requirements and elements of the (model-based) software architectural design. Thus, there arises a need for a tool plug-in between requirements management tool (e.g., DOORS, Polarion) and model-based system architecture design tool (Rhapsody, Matlab/Simulink) which would aid in achieving bidirectional traceability.

In the legacy approach, we have employed model-based software architecture modeling tools for diagrams (e.g., EA) and then we take them to the requirements database manually. By this approach, we were not able to establish the bidirectional traceability between requirements and architecture design, especially in the AUTOSAR context. This is because, the legacy toolchain itself was not well suited to AUTOSAR compliant architecture.

During the requirement analysis phase, typically a tool allows the user to create various types of requirements, such as software, system, electrical, etc., from stakeholder requirements documents or from customer requirement database with the help of ReqIF exports (cf. Section 2.3). Although we create requirements, the typical expectations to proceed into development of the requirements are how well the requirements database can connect and synchronize to architectural and design environments (modeling tools). Additionally, how well the modeling tool supports in publishing the model onto requirements database along with tracking of requirements on modeling environment.

As depicted in Figure 9, the modeling environment provides navigable links to the requirements database. Additionally, we were able to enter comments and status during the requirement analysis phase or approval for any modification to the requirement, enabling proper notification and maintenance in a database. As a good practice, the tool exports the traceability information into a report. Irrespective of how good the tool is, its best usage can be obtained when its features are well understood and applied fully. A requirements management tool also ensures that all users are following the same requirements flow based on a specified configuration as well as the flow implemented in the tools.

3.3.1. Publishing and Authoring

With the help of Siemens Polarion ALM Connector for Simulink [9], we are able to link the numerous requirements to model-based architecture and tracking works intuitively in both directions (cf. Figure 10). This seamless approach also helped us to publish the software architecture from the design environment to the requirement database in a seamless fashion. The update of requirements, which is an obvious situation during concept phase of the automotive projects, can easily be pushed to design. Likewise, the updated architecture can also be published back to the requirements database. Further, this plug-in has provided us bidirectional authoring and update process together with establishing bidirectional traceability with ease of use, which was not the case in the legacy approach.

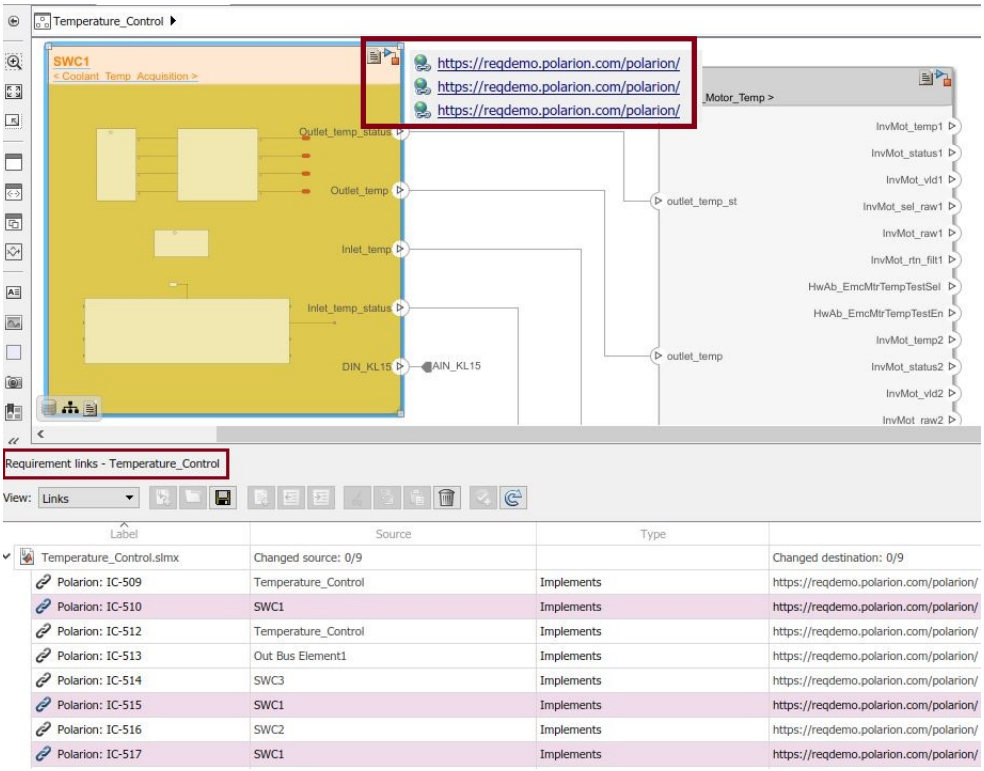

Figure 9. Requirements traceability-requirements database has navigable link to track all the requirements driving the corresponding architectural block in the model. Additionally, the block lists all the requirements corresponding to a software component (SWC1) and from modeling environment, we can track back to requirements database.

3.3.2. Basic Software Configuration Using ARXML Schema

There are several AUTOSAR basic software configuration tools such as DaVinci toolchain [43] in the market. The configuration schema can be exported as a ARXML file and imported in M/S for a seamless integration and update whenever there is an update in the basic software modules. Such a seamless integration was not possible in EA.

During SWE.2 software architectural design, software architect comes up with software component architecture design comprising interfaces (list of inputs and outputs), expected dynamic behavior (SWC period, precedence constraints, etc.), memory allocation and budget. With the help of the proposed approach, he provides the aforementioned information fully or partially in the form of architectural design, which can be further exported as ARXML file. With the legacy approach, we did not find options to export as ARXML. This ARXML file of the component SWC can then be imported at DaVinci developer (or similar AUTOSAR SWC importers) to exchange the above said architectural details to developers.

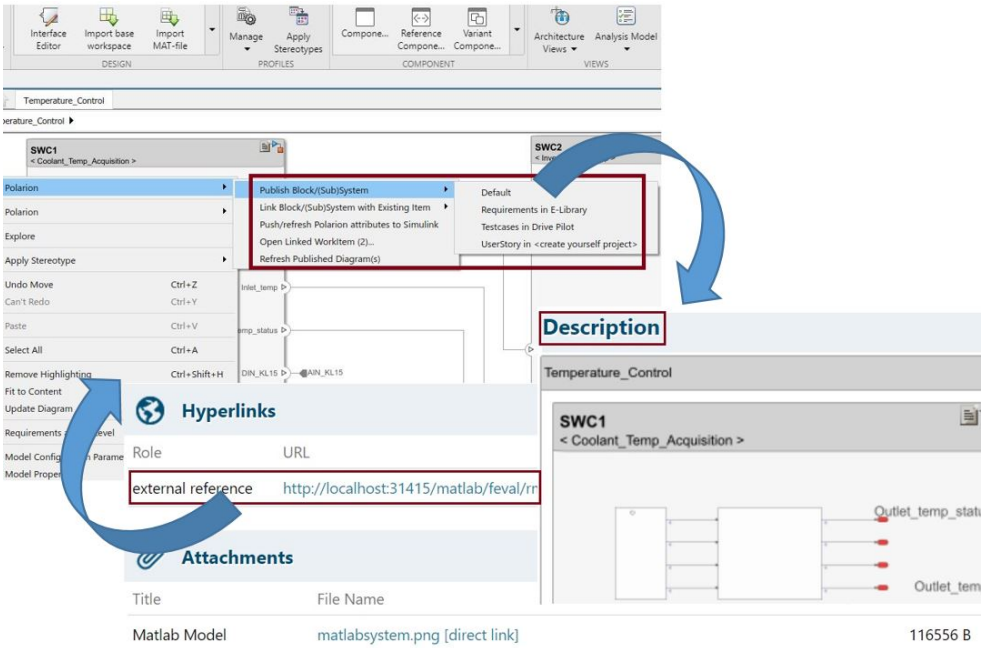

Figure 10. Architectural model authoring and update-practiced in the new approach provides a seamless way to publish the design to requirements database. Additionally, for existing requirements, it provides an option to link them. Whenever the model becomes updated due to maturity of the project, the refresh option could update the same model onto the requirements database. Inversely, any requirement attributes changed on the requirements database can easily be pushed back to the modeling environment.

For developers, it is a highly beneficial workflow as he receives all the necessary information about interfaces (along with data dictionary), and the dynamic behavior of the SWC. This way, the proposed workflow has significantly reduced the development time. In contrast, using the legacy approach, a significant amount of time was spent by the developers to understand the architectural considerations. Thus, the best practices from our lessons learned in the above context are:

- **BP-4**: Employing seamless approach for high-level system and software architectural design with data dictionary support, helps to establish bidirectional traceability between modeling environment and the requirements database. Tracking of requirements back and forth between both the environments to verify fulfillment of requirements;
- **BP-5**: Use of a state-of-the-art plug-in between requirements management tool (Polarion) and System architecture design tool (M/S-SC) to publish requirements and design on to requirements database. Additionally, the approach updates both requirements and design whenever adapted for changes due to technical analysis and discussions in a more efficient way;
- **BP-6**: Import and export of ARXMLs between architectural modeling environment to Basic software (BSW) configuration and development tool-chain to reduce ambiguity on architectural considerations and development time.

4. Lessons Learned

The automotive industry works with the so-called samples (A, B, C . . .), providing components under development as prototypes with increasing functionality and integrating them in test vehicles. As a supplier, during the A-Sample phase, we are expected to deliver functional prototypes usually with limited drivability and low degree of maturity to the OEMs. The A-Sample phase lasts on an average for a year, during which we have held a series of discussions, about architecture modeling, analysis, and design concepts for the electric vehicle powertrain project, with the OEM. It was evident during this phase, that

the UML/SysML tool we have used, in the legacy approach, cannot be used to demonstrate the proof-of-concepts of our design approach. On the other hand, employing tools from two different domains, i.e., the UML/SysML tool for high-level architectural design and M/S for detailed design and analysis, lead to model-to-model gaps. With the introduction of system architecture modeling capabilities in M/S (AUTOSAR toolbox, Simulink System Composer [31,32]) we have adopted **the new approach of using a single tool (M/S) for modeling system architecture and detailed design**.

Having the overall system architecture and detailed design in a single tool has helped us to **carry out several early model-based performance and viewpoint analysis with one global model**. In the design of complex systems, such as the electric vehicle power-train, there are several stakeholders and their viewpoints need to be satisfied. For instance, the *safety engineer's* objective and guarantees in such a system design would be to incorporate 100% program flow check for ASIL-D (the highest criticality) components. On the other hand, the control engineer's objective would be to incorporate, for instance, *minimal settling time* and *reduced overshoot time* of a control parameter. A software engineer's viewpoint would be to deliver a schedule software with CPU load less than, say 70% of a core in a multi-core hardware.

In this context, the early model-based program flow monitoring analysis we have carried out (cf. Section 3.2.6), which is a safety engineer's viewpoint analysis, gives us an estimate on the total CP access time per valid program flow sequence per task (cf. Figure 8). However, the results from timing analysis (carried out in specialized timing analysis tools [47]) was that the deadline for a *temperature control task* (task 2 in Figure 8) has not been met. This was because of the total CP access time of 30 ms overshooting the allowed timing budget of 20 ms for total CP access time for this temperature control task.

In the above scenario, the *temperature control* task being a ASIL-D task, it was mandatory to have the program flow check configured for this task. However, we also had to guarantee that the total CP access time does not exceed 20 ms (which was the allowed budget; but in Figure 8 task 2 we can see it is 30 ms). In order to satisfy the different stakeholders viewpoints and objectives (in this case the *safety engineer* and the *software engineer*), this ASIL-D task was replaced and decomposed into two ASIL-A tasks. The ASIL-A tasks are of lower criticality and are not mandatory to have program flow checks (as in the case of ASIL-D tasks). Had we carried out the model-based performance analysis described in Section 3.2.6 already at early design stage, we could have circumvented this issue. Thus, **early model-based performance and viewpoint analysis enables us to satisfy several stakeholder viewpoints guarantees**. In certain cases, such solutions may be *pareto-optimal solution* (e.g., timing vs. safety), but is still an acceptable solution.

During the B-Sample phase, the supplier is expected to deliver functional, basic prototypes with full drivability, and a high level of maturity. Similarly, during the C-Sample phase, the supplier is to deliver fully functional sample manufactured with series production tools. There are several AUTOSAR authoring tools (AAT), such as DaVinci configurator [43] and EBTresos studio from Electrobit (https://www.elektrobit.com/, accessed on 1 December 2021). In all the sample phases of the EV powertrain project, we have employed the DaVinci configurator tool for basic software (BSW) configuration. The main advantage in employing the proposed approach was that we are able to link to model-based development tools via ARXML file generated from the both tools. The AUTOSAR ARXML importer imports the AUTOSAR software component description files into a Simulink model and vice versa. The importer creates an initial Simulink representation of each imported AUTOSAR software component, with an initial, default mapping of Simulink model elements to AUTOSAR component elements. The initial representation provides a starting point for further AUTOSAR configuration and model-based design. As with a Simulink originated design, we were able to develop the component design and behavior in Simulink-for the imported ARXML file components. This **integration and linking of AUTOSAR authoring tools and M/S with the aid of ARXML files enabled us to meet the (B-Sample and C-Sample) project deadlines in a seamless fashion**. On the

contrary, such facilities are not available in Enterprise Architect UML/SysML tool which we have employed in the legacy approach for modeling system architecture design for the same project.

Finally, traceability comes at a cost in terms of effort in definition, training, reviews and software tools. However, **the benefits of bidirectional requirements traceability, in such complex projects, far outweigh the investment, which is essential for automotive products development**. Further, traceability is mandated by the ISO 26262 standard [11] for a safety-related project. In our EV powertrain project, bidirectional requirements traceability and authoring process using Polarion tool [9], together with M/S, has provided a common development flow for all levels of our organization, as well as real-time data for an informed decision-making. Thereby, no requirement was missed, since they were all linked to the final confirmation. Not only that the impact of request for a change was dynamically analyzed, it was possible to check if the product (A, B, C ... samples) has satisfied all requirements. This served as a measure to gauge the readiness of the EV powertrain product deliverable. Requirements flow and requirements management process are the essential elements that contribute to product quality while ensuring the target goals for functional safety products are achieved and covered by the required verification methods. Having a good requirements management system as a quality process can help organizations achieve higher goals such as the ISO 9001 certification [38].

5. Summary and Conclusions

The standard solution to the complexity constraint has been to model the portion of the system that, if understood, will lead to solution of the problem. A descriptive model (e.g., using UML/SysML) of such a system may provide a high-level overview of the system in an intuitive and easily understood way. This finds wide usage in software engineering and other disciplines. This approach has been successfully used for modeling highly complex software systems, such as aerospace and defense domain [15,16]. On the other hand, such extensive case studies are lacking in the AUTOSAR-based automotive domain, especially concentrating on concrete, real-life projects [20].

However, a significant drawback is that this approach does not work well for complex systems (e.g., automotive systems), because the system is too complex to descriptively model completely or accurately. For such systems, a more fine grained architecture of prescriptive models (e.g., using M/S) are better suited. At this juncture, several automotive companies are facing this dilemma with the usage of UML/SysML for AUTOSAR-based software architecture modeling and torn between the worlds of UML/SysML for producing high level architecture descriptive diagrams and M/S domain for producing fine-grained prescriptive Simulink models from these diagrams.

Addressing this predicament in our organization, we have made a transition in the last twelve months of automotive software architecture design from using legacy approaches/tool chains to employing new modeling capabilities in the recent releases of Matlab/Simulink; for producing both model-based high-level system architecture design and also fine grained prescriptive models. We have elaborated on our approach with the help of a real-life experiences on software architecture modeling of multi-core, mixed-critical, electric powertrain controller project. With our approach of using a single tool and a global model for architecture design, we have eliminated model-to-model gaps arising from involving two different domains. Further, we have been able to carry out umpteen proof-of-concept studies, what-if scenario simulations and especially early model-based performance and viewpoint analysis for various stakeholders. In this context, we have also summarized the best practices followed and valuable lessons learned.

We believe that this first-of-its-kind article with best practices and lessons learned from a state-of-the-art electric vehicle powertrain project will be beneficial for the model-based automotive software engineering community at large.

In this paper we have focused on the static view of the software architecture, such as interface definition, requirements traceability and connections between blocks. In future

work, we plan to investigate on aspects pertaining to dynamic architecture, which is not supported by the M/S tool chain. For example, we are developing a model-based exporter/importer tool plugin for automated export and import of timing annotated M/S model to specialized timing analysis tools (e.g., Gliwa T1 tool [47]) and roundtrip of timing analysis results (from specialized analysis tools) back to M/S tool, respectively.

Author Contributions: All authors contributed to the conception, general idea of the paper to report best practices from transition journey and industry experience. In particular, the overall project work of system architecture design of the inverter control software of the powertrain using legacy and new approach was lead by the author S.M.S., during his stint as a Software Architect at Delphi Technologies. Examples from this project are used throughout this paper. However, the figures used in this paper are not original work items from the live/client project, but an equivalent representation of them. P.I. contributed to the development of proof-of-concept studies and what-if scenarios, with her background and expertise in early performance modeling and analysis of embedded software. In particular, she developed the example scenarios for early model-based performance analysis with a functional safety example of program flow monitoring. Currently, the author P.I. is a functional safety engineer at Innotec GmbH. However, during this work, her stint at the University of Osnabrueck was supported by the head of the Software Engineering research Group, E.P. All authors have read and agreed to the published version of the manuscript.

Funding: A significant part of the time during writing this paper, P.I. was with the University of Osnabrueck in the group of E.P. During this stint, P.I. was supported by grants ZF4447201BZ7 and KF2312004KM4 from BMWi-ZIM co-operation, Germany.

Conflicts of Interest: The authors declare that they have no conflict of interest/competing interests to declare that are relevant to the content of this article.

References

1. van der Sanden, B.; Li, Y.; van den Aker, J.; Akesson, B.; Bijlsma, T.; Hendriks, M.; Triantafyllidis, K.; Verriet, J.; Voeten, J.; Basten, T. Model-Driven System-Performance Engineering for Cyber-Physical Systems: Industry Session Paper. In Proceedings of the 2021 International Conference on Embedded Software (EMSOFT), Austin, TX, USA, 10–15 October 2021; pp. 11–22.
2. Sangiovanni-Vincentelli, A.; Natale, M.D. Embedded System Design for Automotive Applications. *Computer* **2007**, *40*, 42–51.
3. AUTomotive Open System ARchitecture (AUTOSAR). Available online: https://www.autosar.org/ (accessed on 21 September 2021).
4. UML Specification. Available online: https://www.omg.org/spec/UML/About-UML/ (accessed on 21 September 2021).
5. SysML Specification. Available online: http://www.omgsysml.org/ (accessed on 21 September 2021).
6. Mathworks Products. Available online: https://www.mathworks.com/ (accessed on 21 September 2021).
7. Enterprise Architect Tool. Available online: https://www.sparxsystems.com.au/products/ea/index.html (accessed on 21 September 2021).
8. IBM Software. IBM Rational Rhapsody Developer. Available online: https://www.ibm.com/products/systems-design-rhapsody (accessed on 21 September 2021).
9. Polarion ALM Connector for Simulink. Available online: https://extensions.polarion.com/extensions/173-polarion-connector-for-simulink (accessed on 21 September 2021).
10. Martinez, L.R.; Prieto, M.D. *New Trends in Electrical Vehicle Powertrains*, 1st ed.; Intech Open: London, UK, 2019.
11. ISO26262-Road Vehicles Functional Safety Standard. Available online: https://www.iso.org/standard/68383.html (accessed on 21 September 2021).
12. OMG. Object Management Group. Available online: https://www.omg.org/ (accessed on 21 September 2021).
13. Bucaioni, A.; Cicchetti, A.; Ciccozzi, F.; Mubeen, S.; Sjödin, M. A Metamodel for the Rubus Component Model: Extensions for Timing and Model Transformation From EAST-ADL. *IEEE Access* **2017**, *5*, 9005–9020.
14. Mubeen, S.; Nolte, T.; Sjödin, M.; Lundbäck, J.; Lundbäck, K.L. Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints. *J. Softw. Syst. Model.* **2019**, *18*, 36–69.
15. Bocciarelli, P.; D'Ambrogio, A.; Giglio, A.; Gianni, D. A SaaS-based automated framework to build and execute distributed simulations from SysML models. In Proceedings of the Winter Simulations Conference (WSC), Washington, DC, USA, 8–11 December 2013; pp. 1371–1382.
16. Bocciarelli, P.; D'Ambrogio, A.; Giglio, A.; Paglia, E. Model-Driven Distributed Simulation Engineering. In Proceedings of the Winter Simulation Conference (WSC '19), National Harbor, MD, USA, 8–11 December 2019; IEEE Press: Piscataway, NJ, USA, 2019; pp. 75–89.
17. Sporer, H. *A Model-Based Domain-Specific Language Approach for the Automotive E/E-System Design*; Association for Computing Machinery: New York, NY, USA, 2015; pp. 357–362.

18. Mhenni, F.; Nguyen, N.; Choley, J.Y. SafeSysE: A Safety Analysis Integration in Systems Engineering Approach. *IEEE Syst. J.* **2018**, *12*, 161–172.
19. Mhenni, F.; Choley, J.Y.; Penas, O.; Plateaux, R.; Hammadi, M. A SysML-based methodology for mechatronic systems architectural design. *Adv. Eng. Inform.* **2014**, *28*, 218–231.
20. Andrianarison, E.; Piques, J.D. SysML for embedded automotive Systems: A practical approach. In Proceedings of the ERTS2 2010, Embedded Real Time Software & Systems, Toulouse, France, 19–21 May 2010.
21. Daniele, G.; Andrea, D.; Andrea, S.T. *Modeling and Simulation-Based Systems Engineering Handbook*; CRC Press: Boca Raton, FL, USA, 2015.
22. Zacharewicz, G.; Daclin, N.; Doumeingts, G.; Haidar, H. Model Driven Interoperability for System Engineering. *Modelling* **2020**, *1*, 94–121.
23. Ducq, Y.; Chen, D.; Alix, T. Principles of Servitization and Definition of an Architecture for Model Driven Service System Engineering. In *Enterprise Interoperability*; van Sinderen, M., Johnson, P., Xu, X., Doumeingts, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 117–128.
24. Sundharam, S.M.; Havet, L.; Altmeyer, S.; Navet, N. A model-based development environment for rapid-prototyping of latency-sensitive automotive control software. In Proceedings of the Sixth International Symposium on Embedded Computing and System Design (ISED), Patna, India, 15–17 December 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 228–233.
25. Sundharam, S.M.; Navet, N.; Altmeyer, S.; Havet, L. A Model-Driven Co-Design Framework for Fusing Control and Scheduling Viewpoints. *Sensors* **2018**, *18*, 628.
26. Petriu, D.C. Software Model-based Performance Analysis. In *Model-Driven Engineering for Distributed Real-Time Systems*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2013; pp. 139–166.
27. Iyenghar, P.; Huning, L.; Pulvermüller, E. Early Synthesis of Timing Models in AUTOSAR-based Automotive Embedded Software Systems. In Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development, Valletta, Malta, 25–27 February 2020; pp. 26–38.
28. Jianqiang, W.; Shengbo, L.; Xiaoyu, H.; Keqiang, L. Driving simulation platform applied to develop driving assistance systems. *IET Intell. Transp. Syst.* **2010**, *4*, 121–127.
29. Franco, F.R.; Neme, J.H.; Santos, M.M.; da Rosa, J.N.; Dal Fabbro, I.M. Workflow and toolchain for developing the automotive software according AUTOSAR standard at a Virtual-ECU. In Proceedings of the IEEE 25th International Symposium on Industrial Electronics (ISIE), Santa Clara, CA, USA, 8–10 June 2016; pp. 869–875.
30. Vatanparvar, K.; Faruque, M.A.A. Design and Analysis of Battery-Aware Automotive Climate Control for Electric Vehicles. *ACM Trans. Embed. Comput. Syst.* **2018**, *17*, 1–22.
31. Mathworks AUTOSAR Blockset. Available online: https://www.mathworks.com/products/autosar.html (accessed on 21 September 2021).
32. System Composer Toolbox. Available online: https://www.mathworks.com/products/system-composer.html (accessed on 21 September 2021).
33. Navet, N.; Simonot-Lion, F. (Eds.) *Automotive Embedded Systems Handbook*; CRC Press: Boca Raton, FL, USA, 2009.
34. AUTOSAR Release 4.4.0: Methodology and Templates. Available online: https://www.autosar.org/standards/classic-platform/classic-platform-440/ (accessed on 21 September 2021).
35. DOORS Requirements Management Tool. Available online: https://www.ibm.com/products/requirements-management (accessed on 21 September 2021).
36. Requirements Management for Eclipse. Available online: https://www.eclipse.org/rmf/ (accessed on 21 September 2021).
37. ReqIF. Requirements Interchange Format. Available online: https://www.omg.org/spec/ReqIF/About-ReqIF/ (accessed on 21 September 2021).
38. Best Practices for Traceability of Functional Safety Requirements in Automotive. Available online: https://hosteddocs.emediausa.com/synopsys-asset-2-aug-2019.pdf (accessed on 21 September 2021).
39. Polarion Extension for Enterprise Architect. Available online: https://extensions.polarion.com/extensions/167-polarion-connector-for-enterprise-architect (accessed on 21 September 2021).
40. Vitkin, L.; Fallahi, A. *The Role of the Data Dictionary in the Model-Based Development Process*; SAE Technical Paper; SAE International: Warrendale, PA, USA, 2009.
41. Modeling Guidelines of Basic Software Enterprise Architect UML Models, AUTOSAR CP Release 4.3.0. Available online: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TR_BSWUMLModelModelingGuide.pdf (accessed on 21 September 2021).
42. Specification of Watchdog Manager-AUTOSAR CP Release 4.3.1. Available online: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_WatchdogManager.pdf (accessed on 21 September 2021).
43. DaVinci Configurator. Available online: https://www.vector.com/de/en/products/products-a-z/software/davinci-configurator-pro/ (accessed on 21 September 2021).
44. Iyenghar, P.; Huning, L.; Pulvermüller, E. Model-Based Timing Analysis of Automotive Use Case Developed in UML. In *Proceedings of the Evaluation of Novel Approaches to Software Engineering-15th International Conference, ENASE 2020, Prague, Czech Republic, 5–6 May 2020*; Ali, R., Kaindl, H., Maciaszek, L.A., Eds.; Revised Selected Papers; Springer: Berlin/Heidelberg, Germany, 2020; Volume 1375, pp. 360–385.

45. Iyenghar, P.; Pulvermueller, E. A Model-Driven Workflow for Energy-Aware Scheduling Analysis of IoT-Enabled Use Cases. *IEEE Internet Things J.* **2018**, *5*, 4914–4925.
46. Automotive Software Performance Improvement and Capability Determinitation (ASPICE). Available online: http://www.automotivespice.com/ (accessed on 21 September 2021).
47. GLIWA Embedded Systems, Timing Suite T1. Available online: https://www.gliwa.com/index.php?page=products_T1&lang=eng (accessed on 21 September 2021).