[KMG+23] S. Kriebel, M. Markthaler, C. Granrath, J. Richenhagen, B. Rumpe:
Modeling Hardware and Software Integration by an Advanced Digital Twin for Cyber-Physical Systems:
Applied to the Automotive Domain.
In: Handbook of Model-Based Systems Engineering, A. Madni, N. Augustine, M. Sievers (Eds.), pp. 1-38,
Springer International Publishing, Januar 2023.
www.se-rwth.de/publications/

Modeling Hardware and Software Integration by an Advanced Digital Twin for Cyber-Physical Systems: Applied to the Automotive Domain

Applied to the Automotive Domain

S. Kriebel, M. Markthaler, C. Granrath, J. Richenhagen, and B. Rumpe

Contents

S. Kriebel (✉)
FEV.io GmbH, Aachen, Germany

BMW Group, Munich, Germany
e-mail: kriebel@fev.io

M. Markthaler
BMW Group, Munich, Germany

Software Engineering, RWTH Aachen University, Aachen, Germany

C. Granrath
FEV.io GmbH, Aachen, Germany

Mechatronics in Mobile Propulsion, RWTH Aachen University, Aachen, Germany

J. Richenhagen
FEV.io GmbH, Aachen, Germany

B. Rumpe
Software Engineering, RWTH Aachen University, Aachen, Germany

A. Madni et al. (eds.), *Handbook of Model-Based Systems Engineering*,
https://doi.org/10.1007/978-3-030-27486-3_21-1

Abstract

Systems engineering deals with the development of complex systems where complexity is usually product domain-specific. Thus, it often fails to a large extent to integrate the mechanical and electrical engineering disciplines with the computer science discipline, including cultural issues. However, the success of future cyber-physical systems depends not only on hardware functionality but also progressively on its integration into distributed software functionality. The presented Advanced Digital Twin combines the prerequisites for efficient hardware and software integration, particularly for large and complex systems, like cyber-physical systems. Therefore, it is based on an Advanced System Model which comprises the respective architectural designs needed by the domains involved. Based on this resilient integrated architecture, the emerging artifacts can be reused which makes the application of model-based techniques economically reasonable. This enables automated quality checks, simulations, application of artificial intelligence, and big data analysis and serves as a thread through necessary cultural changes to set up a cross-functional and t-shaped collaboration.

Keywords

Digital twin · Model-based systems engineering · Cyber-physical system · Hardware-software integration · System architecture · Functional architecture · Logical architecture · Technical architecture · Automation · Reuse · Diversity · Cultural change

Introduction

It is common sense that for at least the last 30 years, mechanical systems are becoming more and more complex. Some say they follow Moore's law which is the observation that the number of transistors in dense integrated circuits doubles at least every 2 years. However, for mechanical systems, this doesn't mean that they double parts every 2 years. For cost reasons, it is even vice versa. Actually, it means the number of provided functions increases exponentially. These functions are provided jointly by contributions from the mechanic, the electrical/electronic (e/e), and the computer science domains. In the following, such combined systems are called cyber-physical.

Cyber-physical systems (CPS) (Broy et al. 2012; Kirchhof et al. 2020) comprise the most fascinating innovative technical products existing, like cars, planes, trains, ships, spacecrafts, satellites, etc., including cyber functions like traffic control, system-to-system communication, and customer information. This means they are pervasive and contribute to a huge extent to our economic power and advancement.

Unfortunately, mechanical systems engineering today still seems not well prepared to cope with the volatile, uncertain, complex, and ambiguous ("VUCA") product requirements of today's CPS.

In the automotive industry, for example, the VUCA gap addressed in this article is not only caused by modern functions, e.g., driver assistance, communication, and connectivity like network functionality (car2car) or near environment and backend communication (car2x). But it also addresses the real new challenge of autonomous driving with its huge bunch of necessary safe near-field and far-field environmental cognition and communication functions which shall replace the driver one day.

Additionally, a lot of former pure mechanical functions are today controlled by software as well, e.g., suspension, steering, braking, and combustion. Furthermore, these functions are progressively protected by safety and diagnostic functions. For increased customer comfort and product quality, additional functions are required like data analytics, big data evaluation, and artificial intelligence. How to cope with all these functions? How to reliably manage their interactions and dependencies?

Well-known and trained product development cycles for mechanical systems are not entirely matching the product and market needs any longer. Task forces are implemented more and more often as a tried and tested but cost-intensive survival technique of traditional players to achieve the market-driven delivery date. Collateral new market players are coming up with new product and process visions and are changing the game. Market pressure increases. The applied survival techniques are usually matching the schedules but hardly the cost and quality targets.

Economic Darwinism (Read 2010) forces mechanical industries to check their current product development processes for their applicability to CPS development, i.e., intensive and reliable integration of hardware and software. In order not to become a Dinosaur in economic Darwinism, the approach presented in this chapter introduces a way to continuously adapt the respective product development process and the required collaboration culture to the current state of the art in CPS Engineering (CPSE) and the new market conditions.

A comprehensive integrated modeling approach is introduced which facilitates hardware and software integration by an Advanced System Model. It also enables the optimization of time and budget required for the industrialization of a CPS by a specifically branched out Advanced Digital Twin. In addition, it increases product quality by arrogating preventive measures and facilitates the efficient handling of possibly necessary retrospective measures. The approach is based on function-oriented systems engineering combined with a solution-oriented model-based appendage similar to (Kriebel et al. 2017; Kriebel et al. 2018; Drave et al. 2019).

The integrated modeling approach presented in this chapter is named model-based cyber-physical systems engineering (MBCPSE). It merges the respective advantages of the underlying disciplines of mechanical and electrical engineering as well as computer science to overcome the technical and cultural gaps in cooperation and the increasing complexity of CPS. It shall be applied as a domain overlapping integrated development approach as neither a CPS nor the application of MBCPSE can be regarded as domain-specific any longer.

The current challenges of product development processes as well as existing approaches are presented in section "State of the Art" referring to the automotive domain as a well-known example. The details of MBCPSE based on the Advanced System Model and the resulting Advanced Digital Twin are shown in section "Best Practice Approach". The required modeling is illustrated in section "Illustrative Examples" by using an exemplary use case. Finally, in section "Chapter Summary and Expected Advances", the content is summarized, and expected future advances are outlined.

State of the Art

MBCPSE comprises the entire system life cycle starting from the concept phase and structuring the development phase to system design and analysis to the final disposal process (ISO International Organization for Standardization 2015; Walden et al. 2015). Furthermore, MBCPSE is an approach for accomplishing cross-disciplinary systems development. For this purpose, MBCPSE combines a set of cross-disciplinary architectures with separated development steps and artifacts. Consequently, MBCPSE enables a new approach to systems engineering for complex systems like CPS.

Systems Engineering: Overcoming the Differences?

As mentioned before, MBCPSE focuses on complex systems like CPS, in this article the automotive vehicle as a well-known example. A closer look at the term CPS shows that it addresses the areas "cyber," "physical," and "system." A system is defined as a "combination of interacting elements organized to achieve one or more stated purposes" (ISO International Organization for Standardization 2017).

Cyber relates to software systems and is characterized by the culture of computers and information technology, hence the discipline of computer science. Physical relates to the operation of natural forces generally, hence the disciplines of mechanical and electrical engineering. A CPS unites all three disciplines and therefore results in a high degree of system complexity.

Each discipline involved has its different approach to problem-solving and decision-making, i.e., to culture and psychology. In a CPS, these differences often lead to misunderstandings, frustration, delays, and consequently increasing development time and high costs (Kriebel 2018). To avoid these issues, it is necessary to identify potential impediments in order to avoid them but also to focus on the advantages of each discipline in order to keep their benefits. Therefore, the differences in these disciplines between skills and handling complexity are outlined in the following section.

Different Skills and Knowledge

To relate to and to differentiate these three disciplines from one another, it is essential to understand what mechanical engineers, electrical engineers, and computer scientists learn throughout their studies and particularly what they do professionally. The disciplines share many of the same intentions and problem-solving processes but in different dimensions. To identify and solve problems, all of them use many of the same mathematical and scientific principles.

The main difference is that in computer science, the focus is less on physical elements, but on dealing with a huge variety of functional solutions and innovations. Complexity handling techniques such as modeling are used to define suitable boundary conditions. Thus, computer science developers learn to continuously optimize boundary conditions and to efficiently manage the evolution of versions and variants. On the other hand, in mechanical and electrical engineering, the physical products with their physical laws define the boundary conditions. Once developed, they are fixed, and the products are continuously optimized within constant boundary conditions. Engineering developers learn to optimize within given boundary conditions and to manage efficiently the evolution of parameters.

However, the increasing proportion of software in physical products demands a product development process integrating the needs of cyber systems development. To cope with these innovative needs, the development of CPS must merge the advantages of each discipline. MBCPSE facilitates the separated application of the different skills and knowledge as outlined in section "Best practice Approach".

Different Handling in Complexity

Besides different skills and knowledge, the disciplines use different methods to handle complexity based on what developers learned. Electrical and mechanical engineering usually use physical systems decomposition, which is based on (geometrical) components. For this, the decomposition starts with the entire system that is iteratively decomposed into subsystems until the emerging components are manageable to be developed independently. For subsequent technical modeling, CATIA (Dassault Systemes 2020) and MATLAB/Simulink (MathWorks 2019) are common and often optimized by simulations.

In contrast, computer scientists usually decompose the system according to its functional behavior. For this, computer scientists recursively break down the system at different levels of abstraction, until these become simple enough to be solved independently. For the subsequent structural and behavioral modeling, the Unified Modeling Language (UML) (OMG Object Management Group 2017) or Systems Modeling Language (SysML) (OMG Systems Modeling Language (OMG SysML) 2017) is common and often optimized by model-checking and code generation tools.

Figure 1 shows a selection of strengths of the product development techniques of computer science and mechanical engineering. They are assigned roughly to the well-known basic V-model (Boehm 1979; Scheithauer and Forsberg 2013).

However, an applicable solution for any systems engineering approach for complex systems must combine all three disciplines with all its characteristics

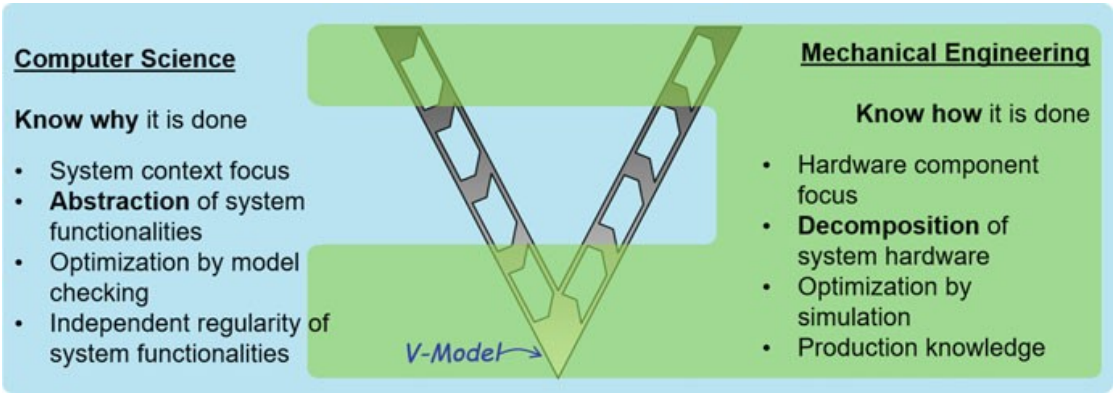

Fig. 1 Strengths of computer science and mechanical engineering in the context of the basic V-model

mentioned above. Mechanical and electrical engineers focus on hardware components, deal with complexity by decomposition of system hardware, optimize the components by simulations, and have particular knowledge about hardware production systems.

In contrast, computer scientists focus on the system context, deal with complexity by the abstraction of system functionalities optimized by model-checking, and have particular knowledge about the reuse of independent system functionalities. MBCPSE incorporates the shown different handling of complexity as outlined in section "Best practice Approach".

Model-Based: Unite Engineering Solutions?

Models are found in every named discipline, and modeling takes place even when the experts are not aware of it. According to (Stachowiak 1973), a model represents an original, includes only relevant ("seeming") properties, and always fulfills a purpose with respect to the original. For example, in mechanical engineering, the finite element method is model-based and aims on the simulation of the behavior of a physical system.

Electrical engineering, for example, uses circuit models for a representation of the electrical system and differential models to describe the aspects of electromagnetism. Compared to these physical models, the models in computer science mainly serve the abstraction of these technical details to concentrate on the functional behavior and describe the logical system, i.e., the operating principle.

Regardless of its purpose, the role of modeling and the quality of models are extremely important not only within the disciplines but for the development of a CPS in general. The integration of the different models is difficult, respectively not possible, because a common "semantic linkage" is missing. The term semantic linkage encompasses linguistic aspects, such as expressions and the understanding of words, and also models, architectures, and their elements. For a successful model-based project, a common spoken language is not enough. In addition to a glossary for

the precise definition of often-used terms (Walden et al. 2015), the models, architectures, and their elements such as interfaces, information, etc. must also be defined and their relationships precisely specified. Otherwise, the misunderstandings and frustration mentioned above are likely to occur and jeopardize the project success. Further, if there might be a semantic linkage available, integration of the different models fails technically due to different modeling languages, modeling guidelines, modeling focuses, and not aligned architectural principles and interfaces. The effort in the particular modeling is cost-intensive, while the models are not usable for an integrated modeling approach. Additionally, the reuse of the models is limited, and therefore the return of investment is little.

To meet this challenge, MBCPSE integrates all models in a comprehensive development model, the Advanced System Model introduced in section "Best practice Approach". The applied model integration is according to well-known methods of computer science. The embeddedness with methods of computer science was chosen because the consistent use of complex models has improved the quality and efficiency of software development in computer science over the last decades (Rumpe 2017). This provides a consistent model-based methodology and aims at a significant raise of the return of investment.

Related Work

This section lists the relevant existing approaches and provides a brief evaluation of their applicability to CPSE. It also provides an opportunity to acknowledge the influence of existing research in the field. Even though the automotive industry has recognized mastering the complexity of CPS, new approaches are still required in the future (Broy 2006; Whittle et al. 2014; Giusto, and R. S., and S. M 2016). Current literature reflects that text-based systems engineering is still the status quo (Giusto, and R. S., and S. M 2016; Liebel et al. 2019). Nevertheless, several approaches exist for the model-based specification of CPS but are not applied as a standard methodology today.

One of the best-known standards in the field of systems engineering is part of the ISO standard 15,288 systems engineering-system life cycle processes (ISO International Organization for Standardization 2015). The standard can be understood as a collection of activities that are necessary to achieve the desired result. The philosophy is based on the systems engineer's ability to divide the given descriptions into a sequence of activities that are applicable to the problem. At the same time, logical sequences for determining the sequences of necessary activities are not specified.

Besides the ISO standard 15,288, well-known examples in the automotive industry include OOSEM (Walden et al. 2015), EAST-ADL (EAST-ADL Association 2013), REMsES (Braun et al. 2014), and SPES 2020 (Pohl et al. 2012). These approaches combine abstraction and decomposition steps which imply for their applicability that either all individuals or teams have to provide profound integrated knowledge in all involved disciplines. This is, of course, possible in smaller professional expert environments but does not usually scale to the needs of the industrial

product development of large and complex systems. In fact, for applicability reasons, these approaches seem to focus on a maximum of two disciplines. In contrast, MBCPSE uses separated steps for abstraction and decomposition and documents them thoroughly. This allows exploiting the experts' knowledge of all three involved disciplines, mechanical engineering, electrical engineering, and computer science in separate steps as outlined in section "Best practice Approach". Of course, there is the need for mutual understanding when defining interfaces and interferences of the CPS. However, this applies not to all individuals and teams involved in the project but to a smaller number of necessarily experienced architects.

The object-oriented systems engineering method (OOSEM) (ISO International Organization for Standardization 2015) is a stepwise approach to system design and specification. In the first step, the needs of the stakeholders are analyzed. In a second step, the system requirements are analyzed, followed by the definition of a logical architecture. In the last step, possible physical system architectures are considered. All the mentioned steps can be applied to different system hierarchies and run through iteratively. Overall, OOSEM represents a collection of activities for each phase of the four-pronged process, which can be used to specify the desired system. Procedures for the execution of the individual steps must be defined by the user in an individual application-specific way. Hence, there is no detailed description of the resulting views. Another difference is that the four steps combine abstraction and decomposition and the approach is not domain-specific.

EAST-ADL (Electronics Architecture and Software Technology-Architecture Description Language) is a domain-specific language in the automotive industry for software and system modeling (EAST-ADL Association 2013). The focus is on the specification of electrical and electronic vehicle architectures and software (EAST-ADL Association 2013). The four abstraction levels of vehicle, analysis, design, and implementation, system descriptions describe the tasks and communication of the system (EAST-ADL Association 2013). In EAST-ADL, the refinement mixes abstraction and decomposition, making it difficult to revise the specification/architecture in the event of a functional or technical change. In addition, the focus is strongly on e/e and software, which limits the integration of the mechanical approach.

Another MBSE approach is REMsES (Requirements Engineering and Management for software-intensive Embedded Systems) (Braun et al. 2014). REMsES uses concepts of the UML for the specification of software-intensive embedded systems. The three abstract and system decomposition layers are divided into a system level, function group level, and hardware and software level and separate the solution and problem level (Braun et al. 2014). Orthogonal to this abstraction is a categorization with views to the categories context, requirements, and draft (Braun et al. 2014). REMsES is similar to the approach presented here, especially in prioritizing the functional basis before the technical solution. However, REMsES mainly focuses on the computer science and e/e disciplines and combines abstraction and decomposition steps.

A similar methodology is SPES 2020 (Pohl et al. 2012). Similar to the approach of REMsES, the "decomposition layers" are arranged orthogonally to the

"viewpoints" requirements, functions, logic, and technology. The decomposition layers are not predefined besides by a structural characteristic from the layer above. SPES 2020 is the only approach that combines all three disciplines, but the mechanical, electrical, and technical parts are separated at an early stage, and no order of the steps is given.

Summed up there are a number of model-based approaches available. However, they seem to be useful in academics or in an expert environment but are not meeting the entire needs for an industrial scalable CPS development as discussed above. Therefore, MBCPSE was developed and applied in the last couple of years as a best of breed approach within the disciplines involved.

Best Practice Approach

There are plenty of differences in engineering and management between mechanical engineering, electrical engineering, and computer science. However, the combination of these three disciplines is fundamental for the interdisciplinary development of CPS as outlined in the previous sections. In the following, MBCPSE is presented as an integrated best practice approach in industrial development which makes it different from the approaches presented in section "State of the Art".

Systems Engineering: Merging Engineering Disciplines

A generally known, widely spread, and well-proven approach to systems engineering is the basic V-model mentioned in section "Different Handling in Complexity", especially in the automotive industry. It starts with a concept phase, from which requirements are derived, followed by design and realization of the system. In these phases, the system is iteratively decomposed and abstracted in each step. Often the V-model is used to structure the development process by product decomposition layers but unfortunately neglects documentation at least in daily routines. This includes requirements and architectural views as well as linked test cases.

The V-model is usually augmented with plenty of domain-specific standards focusing in more details on relevant procedures. The ISO 26262 (ISO International Organization for Standardization 2018), for example, is the automotive functional safety standard and therefore embedded in established development methods for automotive vehicles. The aim of the ISO 26262 is to minimize risks and therefore classifies the risks of failure probabilities. To calculate the failure probabilities, the ISO 26262 proposes to use models describing the failure aspects. It consequently demands intense documentation as well as systematic linking between requirements, architecture, and test cases. The ISO 26262 is derived from the IEC 61508 (ISO International Organization for Standardization 2018), well-known in electrical engineering since 1998.

MBCPSE provides various model-based views as presented in section "The Advanced System Model". Thus, it facilitates the integration of specific procedures

like the failure aspects of the ISO 26262 in the modeling and uses parts of the V-model for structuring the decomposition layer and the respective artifacts. Additionally, it enables systematic derivation of test cases out of requirements and architectural models. It shall be emphasized that the V-model is not used in MBCPSE as a process model. The applied principal architectural methods are decomposition and abstraction as introduced in the following. Every step of decomposition and abstraction is executed separately, inspired by the well-known OSI model (Zimmermann 1980) where application and communication are consequently separated. The separated decomposition and abstraction steps greatly simplify documentation of different architectural views on the CPS.

Starting with the system architecture as the prevailing geometric structure in a mechanical environment, the functional and logical architectures, prevailing in a computer science environment, are developed in separate steps but with a clear plan for integration. This enables to assign the necessary tasks to specialists of the respective disciplines. The technical architecture integrates the results to a joint product view and specifies the (hardware and software) component architecture which prepares for industrialization.

This defined self-contained granularity facilitates particularly the application of agile frameworks like the Scaled Agile Framework (SAFe) (Knaster and Leffingwell 2019) and its suitable agile methods as a process model. In this way, a modern and sustainable systems engineering culture can be progressively built which is attractive to young and experienced specialists as they can evolve continuously within the disciplines during product development.

Decomposition

The purpose of decomposition as an architectural tool is to divide the complexity of a system into less complex subsystems. This is usually done starting from the root system element, going through one or more intermediate layers depending on the system's complexity until a leaf system element is reached. The appropriate number of decomposition layers is dependent on the complexity of the product. Thus, it is possible that a vehicle is decomposed into three to five layers but an aircraft or spacecraft into more than eight layers. The number of layers is variable and can be unsymmetrical, if necessary. The resulting view is the **system architecture** of the product which is structured by **system elements** of types root, node, and leaf as presented subsequently.

Root: In MBCPSE, the decomposition of the system starts with layer 1 which represents the entire product. It contains one system element of type "root" as it is the starting point for the entire product decomposition. In the following, the term CPS is used for the product which is intended to be handed over to the customer. Hence, the boundaries of the CPS are specified by the root. External functionalities like backend support are regarded as part of the CPS environment but not of the CPS itself. Consequently, the CPS can operate in a higher system environment in a system of systems manner. However, it is perceived as a product on its own by the customer. The backend part of the root specifies the interface at product level and is specified in

more details by the subsequent layers ending at component level. The customer relevant part of the root comprises the necessary content for the product manual.

Node: Usually, several connected intermediate layers are required to enable functionalities in a CPS. For example, the acceleration of an electric vehicle is a complex interaction of the chassis, the drive train, the power electronics, the electric machine, and the high-voltage storage. In MBCPSE, the class of these intermediate decomposition layers is called "nodes." Depending on the complexity of a node, additional decomposition according to the system architecture may lead to subordinate nodes.

Leaf: The system elements of the last decomposition layer are of type "leaf." They specify the lowest entities of the system architecture of the CPS under concern, the components. Leaves are usually subject for subcontracting or internal manufacturing when the CPS is industrialized. Figure 2 illustrates a cutout of a vehicle system architecture derived from the decomposition focusing on the torque functionality as an example.

The expression "function list" in the blocks shown in Fig. 2 may surprise in the context of a mechanical decomposition. However, it was chosen intentionally to emphasize that each single element of the system architecture has to provide a specific functionality regardless of its type, i.e., root, node, or leaf. The complete but concise specification of the system element functionality is one of the crucial points in systems engineering and, of course, even more so when dealing with CPS.

Each system element in a layer *n* can be assigned requirements from layer *n-1*. Consequently, not all requirements have to be modeled or implemented in one specific layer. As an example, the vehicle functions can be experienced by the clients, i.e., the final customers, whereas timing relevant security and safety functions are usually occurring only in successional layers. In the same way, mechanical

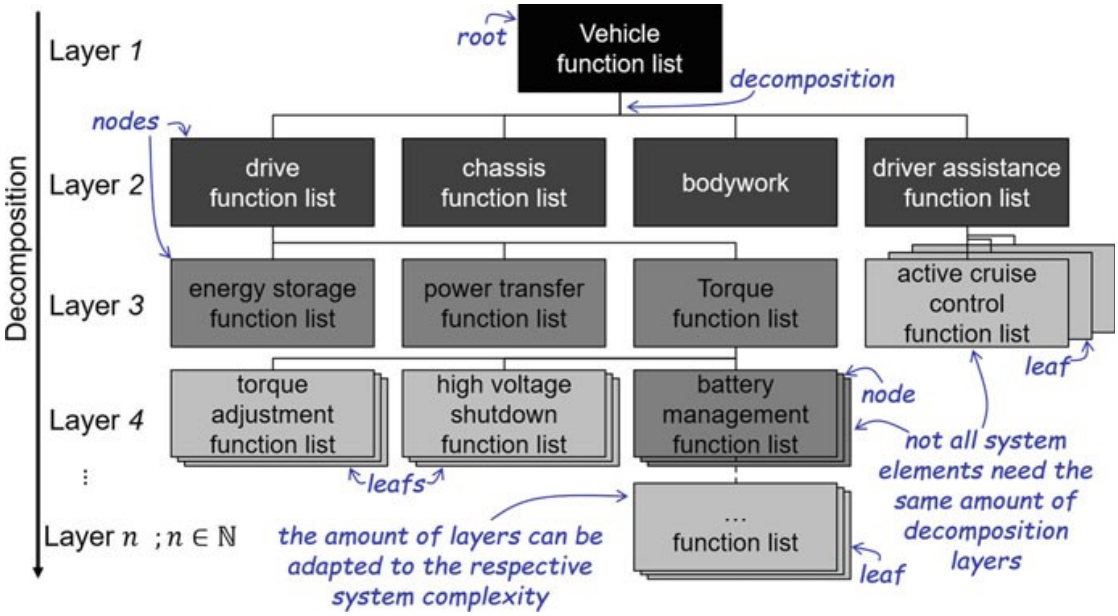

Fig. 2 Cutout of a vehicle system architecture focusing on torque functionality

requirements can be dealt with. For example, a power train momentum is specified in layer n and is further specified in layer $n + 1$ to decide whether the momentum is generated electrically or conventionally by a combustion engine. Of course, they can use the same gearbox specified as well on layer $n + 1$ as contributing to the power train in level n. Using the hierarchical decomposition of system elements persistently in a function-oriented manner facilitates an efficient handling of variants.

Out of experience, system inconsistencies usually occur already when specifying the system elements and their dependencies. The documentation seems to be well established for root and leaf elements but not for the nodes. This may also be due to the compulsory approval and homologation procedures which focus basically on the entire product or the relevant components as the specification of node system elements is often regarded as internal knowledge. Therefore, it occurs that specification of nodes is documented coarsely as this provides apparent efficiencies. However, if particularly the node system elements of the system architecture are not subject to a thorough change management, the connection between root and leaf is lost. Possible side effects through changing cannot be evaluated and tested as traceability is not established. Knowledge heroes who know why the system is designed in a certain way and task forces are necessary for the inevitable reengineering but, unfortunately, more budget and project time as well. Even worse are the effects when system inconsistencies are not detected at all and the CPS operation ends up with failure and loss, e.g., the maiden Flight 501 of the European Launcher Ariane 5 on June 4, 1996 (European Space Agency 2020).

At this point, MBCPSE emphasizes the function-oriented approach according to the advantages of the engineering disciplines involved and pointed out in Fig. 1. The knowledge of the heroes mentioned above shall be documented by means well established in computer science to make CPS organizations sustainably learn their know-whys to be independent from individuals and well prepared for necessary changes. The architectural principle for the anchoring of the know-why is the abstraction presented in the following section.

Abstraction

Based on the system architecture derived from the decomposition presented in the previous section, the functionality of each system element outlines its requirements. In particular, the task, the behavior, the interfaces, and the dependencies are specified. Referring to Fig. 1 and the previous section, the mechanical engineering know-how of the CPS domain was used to derive the system architecture. The abstraction introduced in this section applies computer science methods to document the different views of functionality, as it is not enough to specify the know-why in a sole collection of textual requirements. Too many aspects of the desired functionality would not be specified in this way.

Computer science has developed in the last decades several (model-based) methods and procedures to solve this issue, in response to the software crisis in the late 1960s and 1970s (Randell 2020; Dahl et al. 1972) and when focusing on new development processes and process improvement methods in the 1980s (Humphrey 2002) and 1990s (Booch et al. 2005). These fundamental contributions to today's

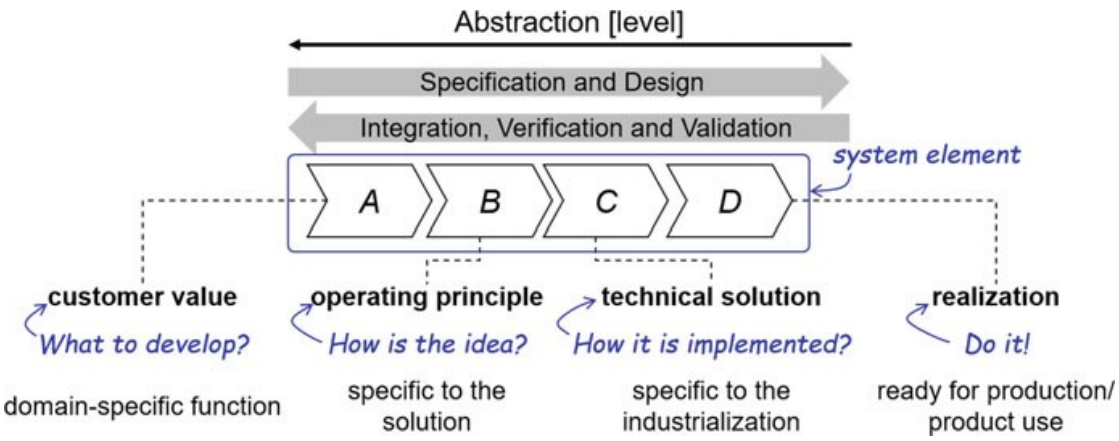

Fig. 3 Specification of a system element: abstraction levels and their purpose

state-of-the art software engineering were also used for the development of agile methods, e.g., SAFe and the agile manifesto in 2001 (Manifesto 2001).

Within MBCPSE, the functionality is used to integrate decomposition and abstraction. In contrast to the variable number of decomposition layers, the number of abstraction levels is defined to four, as shown in Fig. 3. These four abstraction levels are worked out for each system element as follows.

Customer Value: The purpose of level *A* is to specify the customer value of a system element by the required functions, resulting in a **function list**. The environmental conditions and the context conditions are determined for each function. It is important to point out that these functions are independent of the disciplines required to develop them. In other words, those functions can be mechanical, mechatronics, and software functions. For this purpose, level *A* specification elements are realized by.

- SysML use case diagrams (OMG Systems Modeling Language (OMG SysML) 2017) to provide an overview of the entire system at a high level of abstraction with services provided from the user's point of view.
- Textual requirements to supplement the use case diagrams.

The model includes both the diagrams and the textual requirements. The textual requirements supplement the diagrams with information that cannot or are not intended to be modeled. Consequently, there must be no redundancy between the diagrams and the textual requirements. If requirements are required outside the model, they are exported from the model and are not adapted since the model is the single source of truth.

Operating Principle: Each function of the function list created at level *A* is further specified by its operating principle at level *B* (see Fig. 3). The required functions are broken down into subordinated functions which are either local, i.e., within the system element, or functions of a child system element allocated in the subsequent decomposition layer. The resulting functions are then logically integrated at the function level. To achieve functional consistency at the system element level,

all logically integrated functions of the function list have to be integrated into the **logical architecture**.

If possible, the operating principle is not intended to anticipate a technical solution, although certain product conditions may apply. Such product conditions can be rather simple but may have a major impact, e.g., a car has four wheels and is powered by an (electric) engine and not by a turbine. If necessary, they are added as (textual) requirements. Level *B* specification elements are realized by.

- SysML activity diagrams (OMG Systems Modeling Language (OMG SysML) 2017) to model sequences of actions.
- SysML state charts (OMG Systems Modeling Language (OMG SysML) 2017) to model states of the system under consideration and SysML sequence diagrams (OMG Systems Modeling Language (OMG SysML) 2017) to model the flow of a use case with the focus on interactions.
- Textual requirements to supplement the SysML diagrams.

Technical Solution: At level *C*, the function-oriented view of the operating principle is mapped to the mechanical view of the system, i.e., the functions of the function list are assigned to the provided **technical elements.** In other words, it is identified how the function and its operating principle are technically realized.

Technical elements are the abstract representation of one possible industrialization. This implies that technical elements correspond to system elements. However, it is possible that several system elements are bundled in one technical element or that one system element is distributed over several technical elements in one variant.

For example, a torque adjustment includes the functionality, electronics for regulation, mechanical parts such as the electrical machine, and software for intelligent control. All these technical elements can be mapped onto one system element, e.g., a component. Or the electronics and software could be outsourced to another system element for electromagnetic compatibility reasons. Since the maintenance of architectures involves effort, it is recommended to match the technical elements 1:1 with the system elements. Otherwise, both the technical and the system architectures have to be maintained separately, which leads to a significant additional effort and is prone to inconsistencies.

The required technical elements are further specified locally, i.e., within the decomposition layer, or using the functionality of a child technical element allocated in the subsequent decomposition layer. Furthermore, the technical elements determine which elements are realized in hardware or implemented in software, if the operating principle in the respective layer is already detailed enough. The resulting **technical architecture** integrates all technical elements to achieve technical consistency. Level *C* specification elements are realized by.

- SysML block definition diagrams to describe the relationships between the technical elements.

- SysML internal block diagrams to describe the interfaces and information flows between the technical elements.
- Textual requirements to supplement the SysML diagrams.

Realization: Based on the technical solution, the realization of the technical elements is specified at level *D*. The main focus of this level is to specify and document the necessary details for hardware and software in order to enable their efficient industrialization. The structure is called **component architecture** and remains local within the decomposition layer of the technical element.

The required components are further specified locally, i.e., within the technical element, or using the functionality of a child technical element allocated in the subsequent decomposition layer. The component architecture is consolidated by the resulting technical solution which integrates all technical elements at the same decomposition layer to achieve technical consistency.

The transparent and documented refinement and decision-making process from level *A* to level *D* result in the integrated specification of the system element. This helps to react quickly to change requests in any circumstance and is reusable. In the case of new technical findings, another type of realization can be quickly chosen without having to change the basic functionality of the technical elements. By this, local changes can be tested and approved locally when not changing the interfaces. This effect has a huge impact on possible cost reductions when industrializing the product.

The Advanced System Model

The Advanced System Model is the essential systems engineering element within MBCPSE. It provides all necessary architectural views and information of the CPS. It comprises the system architecture, the functional architecture, the logical

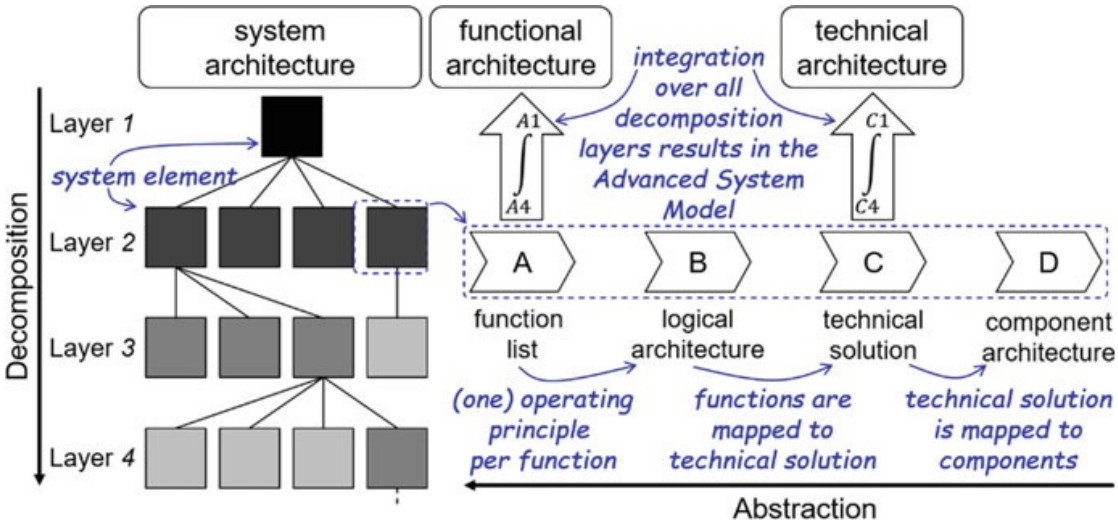

Fig. 4 The Advanced System Model

architecture, the technical architecture, and the component architecture and parts of the e/e architecture, as shown in Fig. 4. Thus, the Advanced System Model provides the **semantic link** for the integration of all the disciplines involved as mentioned in section "Model-Based: Unite Engineering Solutions?".

Each system element is defined by the decomposition presented in section "Decomposition" and specified by the abstraction introduced in section "Abstraction". Figure 4 illustrates the orthogonality of abstraction and decomposition of the Advanced System Model. The development of the advanced system architecture is outlined in this section.

In order to initially apply MBCPSE within an organization which has already developed a prevailing **system architecture**, this structure can be modeled as the system architecture of MBCPSE by an initial decomposition of the CPS as presented in section "Decomposition". The system architecture represents a primary abstract mechanical decomposition of the CPS and provides the **system elements** of types root, node, and leaf. The requirements for each system element are clustered according to its functionality. As an example, the decomposition of the root vehicle function list in Fig. 2 is partitioned into the nodes drive function list, chassis function list, body function list, and driver assistance function list. The further decomposed node (sub-)functions are then partitioned into further subfunctions until reaching the leaf functions in the leaf system element which provide pure hardware and software functions. Hence, a **function list** structures and specifies a system element. Consequently, the mechanical hardware focus is integrated with the functional focus of computer science, which provides the necessary **semantic linkage** between the two disciplines.

In this way, the **customer value** (level *A*) of the CPS is specified by several function lists, one for each system element. However, this would lead to system inconsistency as the specified function lists are worked out independently and are still not consolidated, i.e., there is no integration plan or architecture applied yet. To derive the **functional architecture** from the set of function lists, they have to be made consistent. Within MBCPSE, the principle of **parental communication** is applied. Each function of a function list shall be fully specified either internally within the system element or using the functions of a child system element only. In other words, each function serves the customer, the parent system element (see section "Decomposition"), or other functions of the same system element.

It is essential that the functions of different system elements of the same layer do not communicate directly but are controlled by the parent function of the layer above. The objective is again system consistency. The functions of the system element in layer n would not notice the information exchange of the functions of different system elements in layer $n + 1$ which may cause a state change of the CPS, and the entire system would risk inconsistent behavior. Needless to emphasize, this is one of the main sources for task forces, additional cost for rework, and system failure. In addition, applying the parental communication principle provides a design criterion for the functional architecture. The number of indirect communications between functions of the same decomposition layer via their parent functions shall be as small as possible.

The functional architecture may give the impression of over-specification or being too complex at first glimpse. Still, it provides system consistency and facilitates (re-)structuring and (re-)integration (particularly after changing) of the respective functions. This means individual parts of the system can be flexibly separated and reused without having to detach them from the entire network. Depending on the quality of the functional architecture, reuse can be deployed to a large extent at any layer. This provides a quite large leverage for cost savings at any stage of the product life cycle. Particularly in the automotive domain, this feature additionally contributes to customer satisfaction and risk minimization when applied to necessary changes in operative use.

Furthermore, the quality of the functional architecture can be measured by the parental communication principle. If, for example, an active cruise control function in Fig. 2 directly communicates with the torque adjustment function, this communication must communicate over five system elements. Consequently, it must be evaluated whether these functions are optimally deployed in the functional architecture or the system architecture, respectively.

For the specification of the customer value, it is also necessary to determine the customer. There are different kinds of customers possible. Most important is the end user, of course, when specifying the root system element. Specifying the other system elements, the functions of the function list of the parent system element are regarded as a customer. The third kind of customer is external stakeholders like legal regulations, standard specifications, and company standards which can occur in all decomposition layers. This also means that requirements can either be propagated through the Advanced System Model or be directly assigned to one system element. Requirement's consistency has to be checked continuously throughout the sequence of development phases, e.g., sprints or milestones.

Corresponding to the functional architecture, the underlying **operating principle** of each function (level *B*) shall be made consistent. This leads to the **logical architecture** of each function list, respectively the system element. The logical architecture also clusters and aligns various model elements of the different functions specified. Required data are to be aligned in resolution, timing, accuracy, repetition frequency, etc., e.g., speed information and engine state. This facilitates identifying commonalities and redundancies as well as reducing the development effort. Thus, information and data consistency are provided already at an early phase of the product development to avoid later cost-intensive troubleshooting when integrating the components. The operating principle shall be highly independent of technical influences to facilitate reuse.

Therefore, the principle of the **lowest technical definition** is introduced. It means that a detailed technical solution of a system element shall be defined by the child system elements in the lowest decomposition layer possible. The system elements of higher decomposition layers can, of course, limit the spectrum of technical solutions when technically necessary. As an example, the principle of the brake system of a vehicle has to be decided already at the root level as it has a major impact on all other parts of the system when using a brake parachute instead of an (electromagnetic) friction brake. It might be different, for example, for the power train when using an

integrated combustion engine and/or an electric power train with or without fuel cells. If the different types of power trains fit into the same geometric and electrical interfaces, they can be gathered together as variants of a system element in the Advanced System Model.

It appears that in organizations developing complex systems, the hierarchical structure of the organization is mirrored to the system architecture in order to have assigned a one-on-one responsibility (Manifesto 2001). Of course, this effect can facilitate efficiencies by easier decision-making and correspondingly adapted processes. Nevertheless, it impedes overlapping innovation and technology changes as well as entraps to neglect thorough documentation. Particularly the structured and integrated modeling of the functional and logical architecture is a prerequisite for sustainable and efficient systems engineering.

At this stage of MBCPSE, the system architecture, the functional architecture, and the logical architecture of the CPS are consistently modeled and well documented. However, the system architecture is still a primary abstract mechanical decomposition as mentioned at the beginning of this section and has to be further technically detailed to be applicable for industrialization. The system architecture is therefore mapped to the **technical architecture**, i.e., the system elements are mapped to **technical elements** in all decomposition layers as outlined in the following. This implies that there are technical elements of type root, node, and leaf in analogy to the system elements and the mapping remains in the same decomposition layer.

There are three possibilities for how the mapping can take place. Firstly, the system elements can be mapped 1:1 to the technical elements. This means the technical solution for all system elements which is the entire CPS is defined from the beginning. This is suitable if the product doesn't change or innovate but in the given system elements. Secondly, the system elements can be mapped n:1 to the technical elements. This means a technical element integrates a number of system elements. This contributes particularly to savings of material costs. However, the additional cost for the more complex integration may not be planned. Thirdly, the system elements are mapped 1:m to the technical elements. This means the system is further decomposed and additional technical interfaces are introduced. It has to be particularly made sure that these interfaces are not affecting the functional and technical consistency of the CPS.

It is important to mention that all three mapping possibilities can be applied to different parts of the system architecture as this is a major asset of MBCPSE. There may be less innovative parts within the CPS that are already well established for industrialization where modeling efficiencies can support cost savings. For these parts, the details of the technical elements are integrated into the technical solution of the system elements. The principle of the lowest technical definition is broken intentionally for cost savings. For more innovative parts of the system architecture, the functional and logical architecture can be used to check whether the technical innovation under concern can be consistently integrated. However, the principle of the lowest technical definition shall be applied in general, allowing the mentioned exceptions. For CPS consistency reasons, the mapping of system elements to

technical elements has to be checked when changing interfaces of the technical architecture.

After the mapping of the system elements to the technical elements, the technical elements are further specified locally and by the technical elements of the child technical element of the next lower decomposition layer. In this way, a consistent and complete technical architecture is created. If the technical element is of type leaf, the technical solution at level *C* is further detailed to the **component architecture**, which comprises the realization at level *D* for each technical element. The component architecture contains the requirements for subcontracting the components production and assembly.

According to the technical architecture and the component architecture, the industrialization of all parts of the CPS, inclusively their assembly, shall be made consistent. For all mechanical technical elements, the overlapping of constructed space is avoided by the thorough model-based specification of the technical architecture and the subsequent computer-aided geometric integration. For the mechatronics components, including pure computational components, additionally, the **e/e architecture** has to be developed.

In any case, the e/e architecture is a topic in CPS development on its own. However, the thorough specification of the functional and logical architecture at levels *A* and *B* and the technical and geometric architecture at levels *C* and *D* provide all necessary specifications to measure and evaluate possible e/e solutions. It is worth to be mentioned again that without a complete and consistent functional and logical architecture, the performance of an e/e architecture can only be evaluated after industrialization when finally approving the CPS itself, accompanied with additional time and budget needs for the required rework.

The abstraction and decomposition steps are performed and repeated multiple times until the CPS is fully specified and designed. If the requirements of any abstraction level in any decomposition layer cannot be met, the higher-level or higher-layer solution shall be reworked iteratively. This accelerates the development and helps to achieve an early cross-system consensus for the entire system, with all its system elements because the specification is consolidated before industrialization. Like this, the consistency and completeness of the Advanced System Model can be evaluated in early development phases. This early evaluation of the entire CPS can be done by continuously testing the CPS against the model views of the Advanced System Model.

Summed up, the Advanced System Model is the overarching element of MBCPSE. It comprises the required architectural views for the disciplines involved with an integrated semantic linkage of all models. Different technological solutions and variants for a deliberate technical element of the technical architecture can be easily added to the Advanced System Model as long as its interface is not changed.

Furthermore, the Advanced System Model extends the analysis, organization, checking, and derivation of elements for automated procedures. This includes simulations, test case generation, and model-checking as well as failure mode and effects analysis (FMEA). As the CPS is entirely and consistently structured by the technical architecture, all engineering procedures can be applied to a specific

technical element. This facilitates the assignment of specialized teams and increases the quality of the results. Hence, it connects people through common interfaces and languages. In addition, the unified data sets enable big data approaches, e.g., digital shadowing (Bibow et al. 2020). With digital shadowing, the system and functional behavior can be checked according to its data without a predefined use case and requirement such as defined in the customer value. The behavior is tested on stored data and use cases from the field which is not feasible in this number of use cases by individuals. For this and further applications, structured data as presented in MBCPSE is necessary.

This consistent model view of MBCPSE facilitates a so-called 150% parts list. Of course, not all variants of the Advanced System Model are industrialized in one specific CPS version. Functional variants can be efficiently managed due to the system view of the functional architecture. Mechanical and electrical variants can be efficiently managed due to the technical architecture at level *C* and the component architecture at level *D*. Hardware and software solutions within the component architecture are to be realized specifically only. For this purpose, the realization uses different tools to (generate) code, design hardware, and integrate software and hardware.

The decision which variants shall be deployed is made for the industrialization of a specific version of the CPS for which the Advanced Digital Twin is derived from the Advanced System Model as outlined in the following section.

The Advanced Digital Twin

"A digital twin of a system consists of a set of models of the system, a set of digital shadows, and provides a set of services to use the data and models purposefully with respect to the original system" (Bibow et al. 2020).

The digital twin introduced in this section is the outcome of MBCPSE and comprises the complete, consolidated, and comprehensive model-based documentation of the specified version of the CPS to be industrialized. The set of models of the CPS is structured by semantically linked architectures which specify the relevant characteristics and functions of the CPS, as outlined in section "The Advanced System Model". Further, it provides a set of services to use data and models purposefully. For this, the Advanced Digital Twin of MBCPSE is derived from the Advanced System Model, presented in the previous section. As an enormous advantage, MBCPSE integrates the different architectures explicitly. Thus, each discipline integrates its architectural and technical assets, contributing the respective added value.

The functional and logical architectures are structured according to the system architecture, i.e., to the system elements, as this is ensured by the semantic linkage (see section "Systems Engineering: Merging Engineering Disciplines"). They provide the architectural aspects respectively the behavioral models representing the computer science point of view. The functional architecture models the consolidated and consistent customer value. It enables deriving specific development artifacts

such as test cases for system and acceptance testing. The logical architecture specifies, consolidates, and integrates the respective operating principles. It facilitates, for example, the development of interface testing.

The technical and component architecture are initially structured according to the system architecture, i.e., to the system elements, as well. The system elements are then further mapped to the technical elements to decouple the mechanical and electrical point of view from the functional point of view. With this structured and reversible decoupling, MBCPSE provides a semantic linkage between the computer science and engineering disciplines. The technical elements provide the architectural aspects, respectively, the mechanical and electrical models, representing the engineering point of view, including the e/e architecture for the design of the vehicle electrical system. The technical architecture models the consolidated and consistent technical solution. It enables deriving specific development artifacts such as test cases for system test, e.g., crash test and onboard communication as well as component testing. The component architecture further specifies, consolidates, and integrates the respective requirements for subcontracting and industrialization.

Nonetheless, as already mentioned in section "Introduction", modeling has no economic purpose on its own. The nontechnical objectives of modeling a digital twin are to minimize the required time and budget for industrialization and operative use of the CPS. Thus, generating an **Advanced Digital Twin** is the phase of MBCPSE where the not neglectable additional effort for modeling the Advanced System Model shall pay off. Particularly, multiple generations of different Advanced Digital Twins, i.e., the reuse of the Advanced System Model, lead to a major increase in cost savings and customer satisfaction.

Based on a complete, consistent, and tested Advanced System Model, an Advanced Digital Twin can be efficiently branched out. For each technical element, the required variants are selected to set up the Advanced Digital Twin.

The maturity and quality of the technical elements may vary depending on the number of previous usages in earlier industrializations. However, adequate risk management covers possible maturity and quality issues by introducing effective measures for risk mitigation. The technical architecture shall not be changed at branching out or afterward. With such basic conditions, the respective industrialization shall remain in time and budget.

Since all product-relevant data is proven to be complete and coherent as well as available in a common pattern, i.e., semantically linked, the Advanced Digital Twin intrinsically provides consistency for integrated engineering, production, and operational approaches, as shown in Fig. 5.

Figure 5 shows how all elements of MBCPSE are connected, providing the systems engineering framework. It shows how decomposition and abstraction span the documentation of the CPS. Based on the structure of the basic V-model, the system elements are well defined by the Advanced System Model. The diagonal in Fig. 5 shows the apparent cost-saving mentioned in section "The Advanced System Model", i.e., the decomposition and abstraction steps are worked out and documented in one step. This doesn't enable iterative development due to the complexity of each step. Hence, the "diagonal development" skips at least 75% of

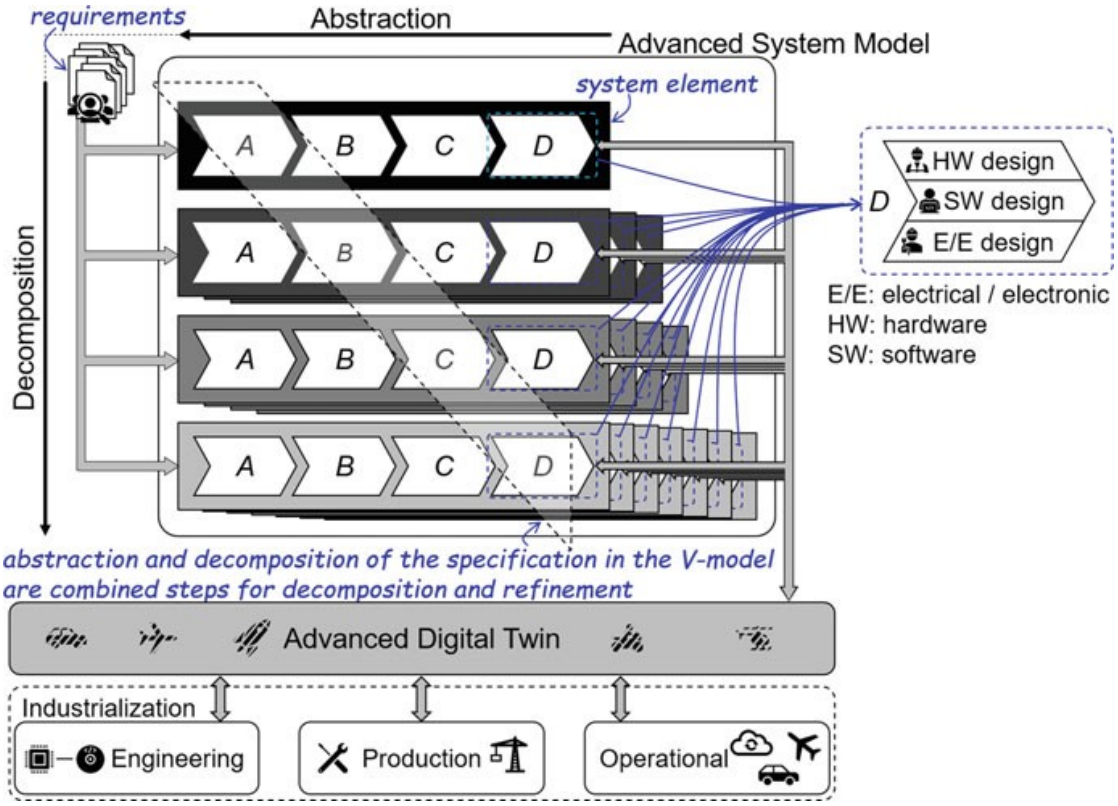

Fig. 5 The Advanced Digital Twin is branched out from the reusable Advanced System Model and provides all information and documentation for an efficient industrialization of one specific version of the CPS

the development steps and therefore seems to be efficient, but it is not sustainable. The similarity to the so-called waterfall model is obvious. With such coarse documentation, the reusability of the development artifacts is not feasible with respect to robust testing and approval.

In contrast to the diagonal path, the separated steps of abstraction and decomposition in MBCPSE is not only about refining but also about being able to pull the elements apart again for modifications or technical replacements. Therefore, it is necessary to specify the entire abstraction and decomposition matrix of Fig. 5 with all its system elements.

Furthermore, the different system elements of the Advanced System Model, e.g., layer *2*/level *A* and layer *3*/level *C*, may be subject to different development styles, e.g., agile or conventional, as long as its interfaces and documentation results are not affected. A further crucial point is, of course, that all structural changes are to be done within the Advanced System Model and not within the Advanced Digital Twin, i.e., after having branched it out.

Model-Based: More than a Description

The sustainable management of any applied systems engineering and development methodology needs a cross-disciplinary base for communication. This cross-disciplinary base is preferably model-based to have a reliable basis for agile and fast processes. Reliable means that the model is the single source of truth and the necessary information for the system is included in the model. Since the costs and benefits of modeling must always be weighed up, the additional information has to be limited. This counts, of course, also for textual requirements.

Furthermore, it is important that once the modeling has started, the information from the model remains model-based. This means that a model is never converted into textual requirements and vice versa. This rule is applied because the conversion from model to text and back to model bears high risks of redundancies as well as the loss of information and must be avoided. Nevertheless, this procedure is still a common practice in model-based approaches. The requirements are adopted and satisfied depending on the depth of detail of the layer and level. Consequently, the model is not overloaded with unnecessary information and can be reused. Such a model has to support the top-down approach to reduce complexity and to enable an intellectual cross-disciplinary understanding of complex correlations. For this reason, the model needs a language that is

- Unambiguous.
- Domain-specific.
- Machine-readable.
- Understandable for people working in systems development.

An unambiguous and common language is necessary for the cross-disciplinary approach since the operating principles and technical solutions need to be understandable and comparable for every discipline. Otherwise, the interdisciplinary cooperation with its interaction between the disciplines will be lost within single models. The practical application of modeling shows that previously separated tasks elaborated separated teams building silos inside an organization are solved together in a cross-functional team in order to clarify all open questions, which are triggered by needs of explicit modeling. Hence, modeling especially fosters a cultural change from the strict division of labor into more agile organizations where each team member establishes its t-shaped profile through collaboration with others (Johnston 1978). A useful starting point for an unambiguous and common language is general-purpose modeling languages like SysML (OMG Systems Modeling Language (OMG SysML) 2017).

Based on a systems engineering method and an established modeling method, efficiencies can be improved with a domain-specific language (DSL). A DSL is less expressive than general-purpose modeling languages and limits the modeling options, but it is expressive enough to represent and address the problems and solutions for the specific domain (Brambilla et al. 2012). However, if the specification of a DSL is considered, experience with MBSE is required. This is because the

scope and requirements of the desired DSL have to be communicated to experienced DSL developers. With the DSL, fewer redundancies, declarative descriptions, easier readability, and easier learnability due to the limited language set increases efficiency.

Additionally, the DSL has to be machine-readable to support all parties involved in the project with automated steps to monetize the development more efficiently. An illustrative example is the checking of functional dependencies using model-checking, automated functional safety checks, or automated test case creation (Drave et al. 2019; Drave et al. 2018; Hölldobler et al. 2019).

The unambiguous, domain-specific, and machine-readable language shall also be legible and comprehensible for people working actively in systems development. The project participants include modelers, system engineers, testers, functional safety engineers, and developers from various disciplines. All these groups must be able to rely on the information and automatically generated elements from the model. Furthermore, the model represents the status and process of the current project situation that affects various involved groups. A change in the operating principle possibly affects the subsequent development steps as well as integrating, verifying, and validating steps.

Illustrative Examples

The following section illustrates a best practice modeling approach for specifying CPS as presented in section "Best Practice Approach". As an example, the development of a Lane Keep Assist (LKA) function is shown. The LKA is a (product) function that is developed across domains and therefore has interdisciplinary requirements. For this purpose, the levels of abstraction according to section "Abstraction" respectively different views within the chosen SysML framework are exemplified.

The decomposition layer can either be of type node which means that the LKA function is part of a CPS or of type root which means the LKA function is the CPS by itself. It cannot be of type leaf as the components are not specified sufficiently. In any case, as an example, the following modeling represents an arbitrary decomposition layer representing the LKA perspective which would have to be integrated into an Advanced System Model of a CPS at the adequate decomposition layer. In other words, the function LKA is integrated firstly into the functional architecture and then secondly in the system architecture.

The content represents a sample version of the development artifacts and aims to illustrate the modeling methodology of MBCPSE without claiming completeness of the function information.

ID	Text	Allocation
SYS1	1 Customer Value	
SYS2	The system shall activate the functionality only when it is explicity requested by the driver during the operation of the vehicle.	System
SYS3	The system shall inform the driver at all times if the functionality is active, on stand-by or deactivated	System

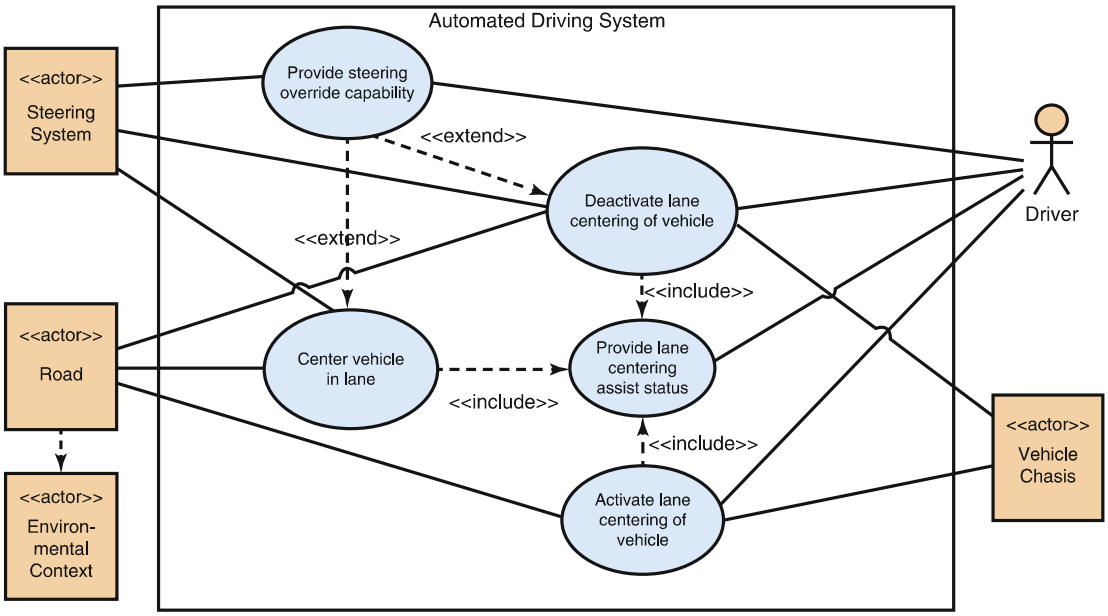

Fig. 6 The customer value of the Lane Keep Assist is modeled by use case diagrams augmented with textual requirements

Level A: Customer Value

The customer value of abstraction is expressed as a black box view and represents the most abstract description of the function. A consideration of the inner operation of the function to fulfill its requirements is not part of this level. The focus is on the definition of clear function boundaries, the identification of all interacting functions and stakeholders, and the specification of black box requirements for the function. These requirements form the central basis for the further development of the function, since the fulfillment of these customer (refer to section 3.1.3) requirements contributes decisively to the success and acceptance of the function to be developed. Further, the requirements of the customer value provide a founded and reusable test basis for system and acceptance tests (Fig. 6).

The LKA function interacts with the driver, the environment (e.g., the road), and the components of the vehicle, such as the steering system, to center the vehicle within the lane. In addition, the actors impose requirements on the function for activation and deactivation, display of the current status, and manual override of the function. The model-based specification is done with a SysML use case diagram and allows good visualization of the interrelations and dependencies of actuators and use cases. The specification of requirements has to be done as part of the diagrams as

ID	Text	Allocation
SYS4	2 Operating Principle	
SYS6	The system shall activate the functionality only if the lane centering assist request is true and the vehicle speed is greater than or equal to the minimum vehicle speed.	System
SYS7	The system shall deactivate the functionality if the driver override signal is true.	System

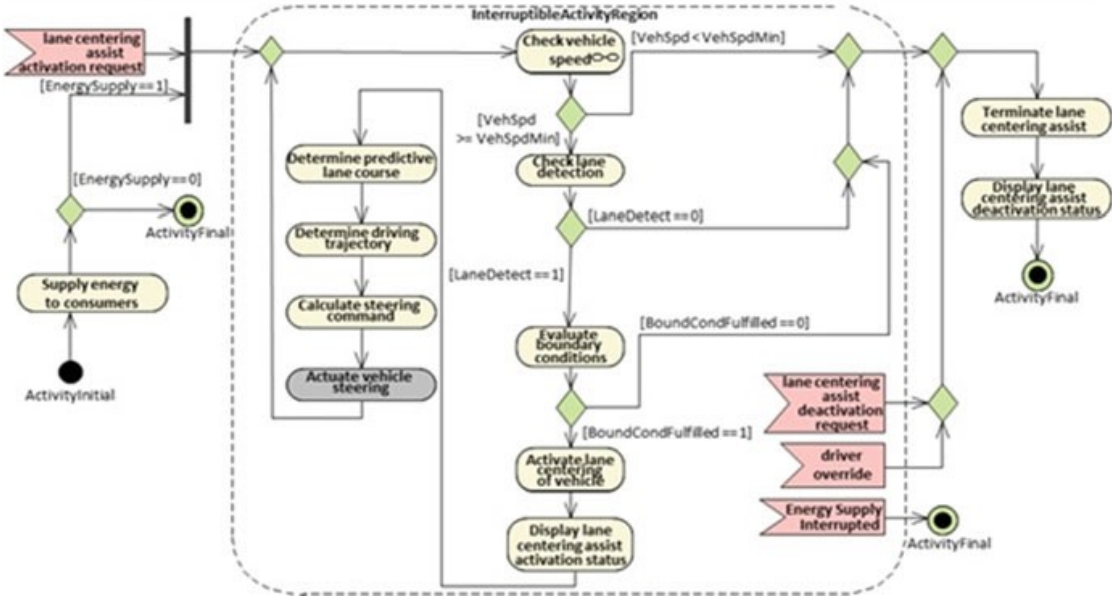

Fig. 7 The operating principle of the Lane Keep Assist is modeled by activity diagrams augmented with textual requirements

well as textual. As an optional aspect, there are advantages for machine post-processing if textual requirements are specified in a formalized manner.

Level *B*: Operating Principle

The next step of abstraction is the specification of the operating principle at level *B*. It is expressed in a first phase as a **white box view** and provides a solution-neutral description of the function, the LKA in this case. Consequently, the inner operation of the function is considered without predefining its technical implementation. Especially functional dependencies, decision paths, function states, and information flows are focused by this form of specification. For the considered example, the operating principle is modeled in the form of an activity diagram and again described by additional textual requirements (see Fig. 7).

It is especially important to note that there shall be only one master of the specification. Even if additional textual requirements exist, they have to be a part of the function model and shall be automatically extractable from the function model. This avoids version conflicts and reduces consolidation and consistency issues, i.e., the well-known but inefficient additional rework effort which comes along with additive cost can be avoided. The actions included in the activity diagram

are formulated solution-neutral to enable its reuse independently of the technical implementation.

For example, the determination and verification of the vehicle speed can be performed by GPS measurement or by means of built-in sensors but is not further specified in detail within the LKA. Figure 7 illustrates actions according to the places of execution in different colors. The highlighting allows identifying those actions that are executed by different components on leaf-level but are necessary for the modeled functionality. Overall, this model-based description represents an ideal functional behavior without considering any kind of functional misbehavior.

The essential substitute behavior in case of **functional misbehavior** is specified in the second phase of the operating principle. In order to consider different kinds of functional misbehavior, a stepwise extension (in accordance to (Granrath et al. 2019)) of the previously modeled desired behavior is added to the model of the first phase (see Fig. 7).

The first aspect is the function **diagnostics**. To ensure that the LKA works fine, all sensors shall be clean. Therefore, an example of the diagnostic aspect is the detection of a not properly cleaned sensor, which requires cleaning to ensure impeccable operation.

The second aspect considered is **degradation**. Here, the ideal behavior with maximum availability of the function is maintained by suitable substitute reactions. An example is the prevention of termination of the LKA due to a not detectable lane, for which purpose the pursuit of another vehicle can be used to stay in line.

The third aspect is the consideration of **safety**-relevant topics. For example, the plausibility check of the current position of the vehicle must be performed for the LKA in order to ensure the safety of the occupants and the environment.

The fourth aspect is **security**. This factor is of particular importance in the field of automotive driving, as there is a high risk of external control of the own or other vehicles, e.g., the LKA must ensure that the leading vehicle is trustworthy; otherwise, the function shall be terminated. If this aspect is not given, there would be a high risk in the vehicle pursuit mode that a risky and unwanted behavior occurs.

Besides the presented supplementary aspects of the operating principle, this level shall also be used to specify additional functionality necessary for the application of **big data** analyses and incorporated **artificial intelligence** algorithms.

The presented stepwise model-based system specification allows especially good complexity handling by continuously adding new aspects during development. In addition, this procedure allows for good cross-functional, i.e., interdisciplinary, collaboration, because depending on the development aspects under consideration, the necessary experts can be involved in a targeted manner, thus reducing high communication efforts caused by excessively large teams.

Level *C*: Technical Solution

The third level is the technical solution. This level is again divided into two phases. The first phase addresses the assignment of the system elements specified by the

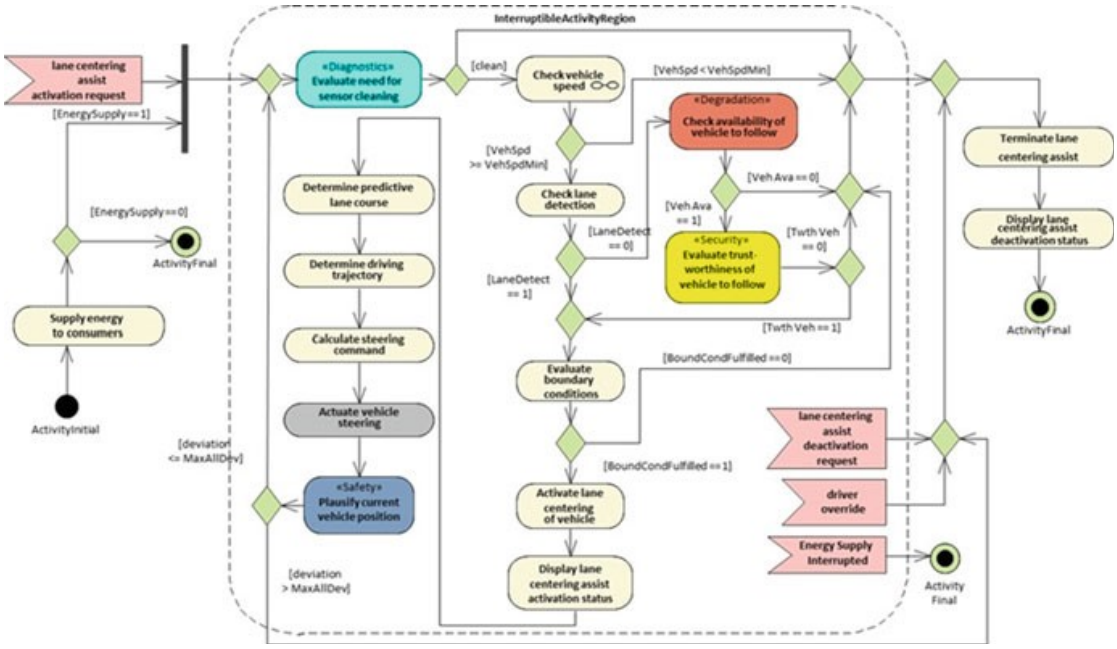

Fig. 8 The operating principle model of the Lane Keep Assist extended with functional misbehavior

functional architecture to the technical elements of the technical architecture within the respective decomposition layer. For this purpose, the functional actions with the corresponding functional requirements are allocated to the full extent. This can be realized, for example, by partitions in the activity diagram of the operating principle. Consequently, in this phase, the system architecture in the form of interacting technical elements is further specified. In other words, the desired functional behavior is assigned to each technical element, and, even more important, the behavior of all technical elements is consistent with the specified operating principle.

However, the technical implementation of the technical elements is still not necessarily defined in this phase, following the rule of the lowest technical definition set up in section "Abstraction". This facilitates improved reuse of specification artifacts since the detailed specification of the technical elements in this phase is still independent of the realization. It allows an efficient mastering of the management of system variants and versions.

In the case of the LKA, the functional actions for the determination of the vehicle speed and for the digitalization of the measured raw values are assigned to a technical element "environment sensing" (see Fig. 8). For the realization of interactions between these technical elements, control flows and object flows are used, whereas the latter only represent information flows (Fig. 9).

In the second phase, the system architecture for the considered decomposition layer is defined if necessary, for further system design, again following the rule of the lowest technical definition. Each defined technical element is assigned either a technical element of the next decomposition layer or a unique type of realization intended for the product, i.e., a component. Thus, in the case of the LKA, the decision is made that the functional architecture element environment sensing is

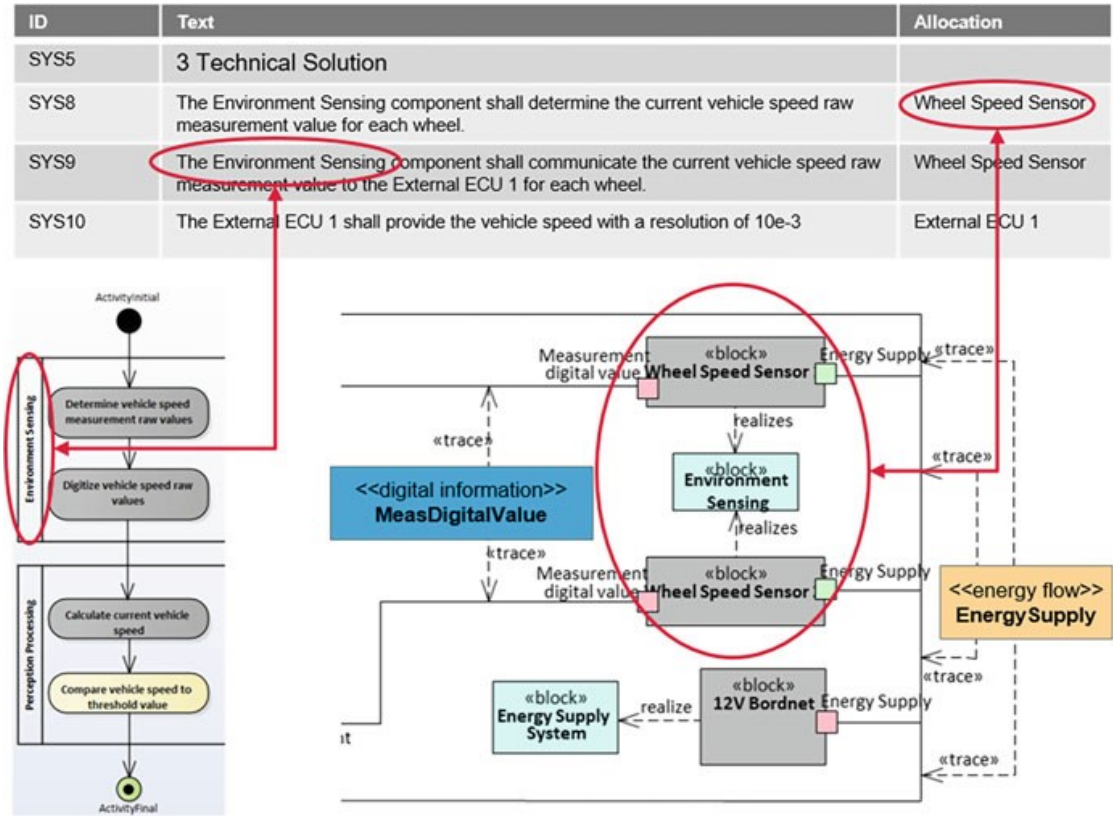

Fig. 9 The technical solution of the Lane Keep Assist is modeled by activity diagrams and internal block diagrams augmented with textual requirements with the objective to provide the functional behavior specified by the functional architecture and mapped to the technical elements

executed by physical components (e.g., two wheel speed sensors). More detailed consideration and specification of the wheel speed sensor component are given as part of the following decomposition layer of type "leaf." However, the functional and electrical interfaces have to be specified here.

To illustrate the system architecture, a representation in an internal block diagram is used. Functional and nonfunctional information requirements (e.g., resolution of information, timing) are additionally specified in textual form. After completion of the architecture definition, which contains decisions about the components to be technically realized and their interfaces, the details of the implementation and manufacturing of the individual components are defined. This includes all relevant functional and technical details of software and hardware components. For the realization of a software component, for example, the integration target to be used must be defined. For hardware components, details about materials like dimensions, resistance, and stability or manufacturing processes to be used have to be specified, for example.

Before the transition to the realization and the concluding industrialization of the system, the technical elements of the technical architecture defined here are mutually consolidated first and then considered and specified individually. When working on the decomposition layer of type leaf, a unique realization must be defined for each

(hardware or software) component. After completion of the final specification, level *D* follows.

Level D: Realization

The last level of the abstraction is the detailed specification of each component of the system architecture assigned to the decomposition layer under concern, independent from whether it is realized in hardware or in software. The detailed specification is then used as the base for industrialization, i.e., as a specification for internal production or for subcontracting.

For this purpose, software models, e.g., with SysML or UML for discrete functionalities in combination with MATLAB/Simulink for continuous functionalities, and hardware models with computer-aided design are created. Based on the software models, software code can be generated automatically or completed by handwritten code. Once the code is generated, it is linked to the technical elements of the technical solution at level *C* as a valid realization and can be verified and validated virtually using the SysML specification.

It is important that the models used in this development step are completely different from the models made with SysML in the previous abstraction levels. No shortcuts are possible as the purpose of the models is different!

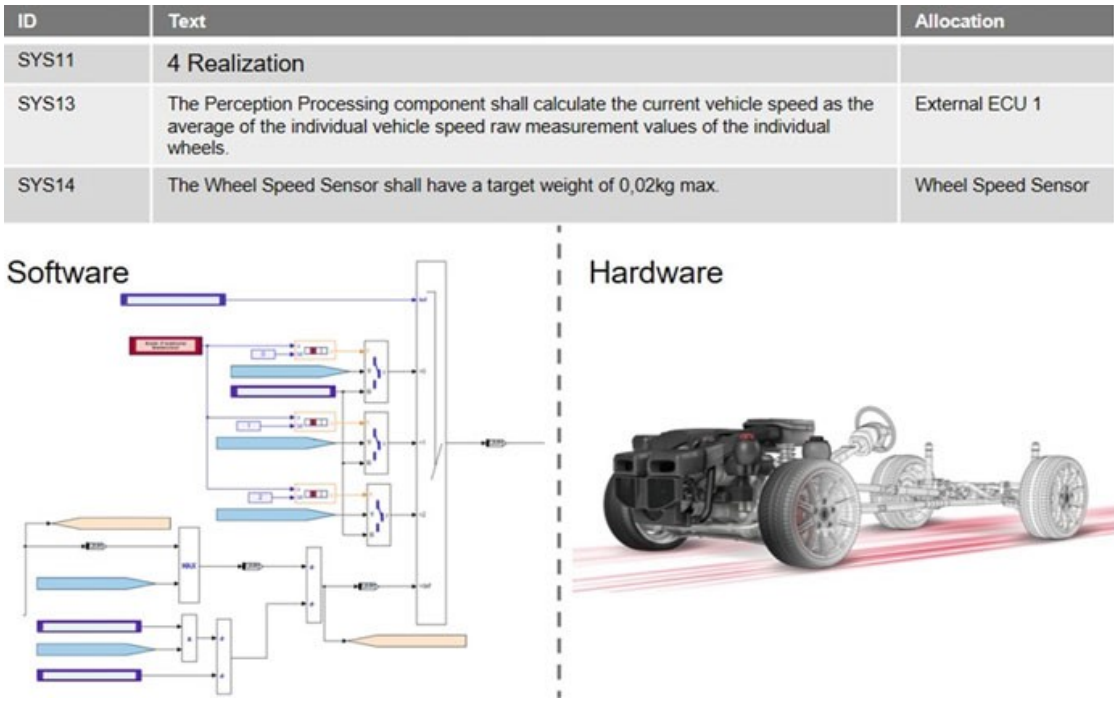

Fig. 10 The realization models the implementation and production results of a technical element augmented with textual requirements and specifies the requirements for industrialization

In close conjunction with the software realization, the hardware is designed and virtually tested. Based on the virtual designs, first prototypes are produced and evaluated. The easily adaptable solutions fit into the previously specified system architecture due to the clearly defined interfaces at all (abstraction) levels and (decomposition) layers. As a result, integration, variation, and validation are facilitated and reduce the possibility of unpleasant surprises due to unpredictable interfaces and communication problems. Each software and hardware prototype is verified and validated with test cases derived from the specification on the corresponding levels and layers within the Advanced System Model. The integration of the components is similarly verified and validated with test cases from preceding specified elements, e.g., the integration of the electric drive system is tested against the test cases of the operating principle to confirm the maturity of functions (Kriebel et al. 2018; Zimmermann 1980) (Fig. 10).

Chapter Summary and Expected Advances

The approach presented in this chapter is called MBCPSE. This is a rather bulky abbreviation for something rather smooth and intelligent. However, each of the three master issues integrated, i.e., model-based (MB), cyber-physical (CP), and systems engineering (SE), demands particular prerequisites, comes along with different requirements, and provides different possibilities. The core is an integrated systems engineering approach, particularly when developing and industrializing a cyber-physical system (CPS). As shown in section "State of the Art", there are multiple systems engineering approaches available in the disciplines of mechanical and electrical engineering as well as in computer science.

Each discipline focuses mainly on its specific requirements, product maturity steps, and culture. The necessary partner disciplines are hardly integrated with respect to intermediate results, milestones, reuse, error tolerance, and decision-making – not to mention the different ways of supplier management. This issue may be made more tangible by checking how and by whom decisions are made in the respective professional environment. It is an easy way to determine the core discipline and may lead the attention to other disciplines, as "diversity" is not meant to be only gender-specific or referring to nationalities. Diversity is indeed also focusing on disciplines as pointed out in the description of t-shaped profiles and cross-functional teams (Johnston 1978).

Before starting to create complex systems, it has to be made sure the people involved in the project are well trained on the same systems engineering approach and have the same understanding, the documentation is well defined, and most importantly they use the same language with respect to the systems engineering approach applied. This can be evaluated by asking a team to set up a glossary with common terms and definitions starting with "function," "model," and "system."

To handle very complex systems like today's CPS, MBCPSE provides an adjustable approach applicable for existing and completely new CPS developments. The adaptability is achieved through system elements derived from mechanical

decomposition, their functional abstraction, their mapping onto technical elements, and the respective properly organized architecture, i.e., the architectures are semantically linked.

With the decomposition introduced in section "Decomposition", a reduction in complexity is achieved by dividing the system into system elements in subordinated layers. Starting from the root system element, the system is decomposed layer by layer into different nodes until the leaves are assigned to component elements. This approach ensures that all system-relevant relationships of the CPS are consistently and completely specified and documented and comprehensively modeled. Thus, side effects can be understood early in the development process or when changes are to be made within the system elements.

The root, each node, and each leaf demand specific functional requirements of the system, i.e., behavior and interfaces. To meet these functional requirements, they are clustered in functions for each system element which are specified by abstraction, as pointed out in section "Abstraction". The carried-out abstraction steps are separated from the decomposition steps to provide the semantic linkage, which integrates the engineering point of view with the computer science point of view. Like this, the specific design steps of each discipline can be performed independently and distinguished in terms of modeling and comprehensibility. Further, it provides the possibility to avoid side effects when modifying a function of a system element. Hence, the semantic linkage provides the mechanisms for an efficient local adaptation reinforced by the principles of model-based systems engineering.

The abstraction is divided into four levels and starts with level *A* the customer value. At level *A*, the environment and context conditions for each function are specified, resulting in the function list mentioned above. These functions are further broken down into internal functions and functions provided by a child system element resulting in the operating principle at level *B*. The operating principle provides a solution-neutral description of the function. At the next level, i.e., level *C* (technical solution), the functions of the operating principle are further technically specified and mapped to the technical elements which are used for industrialization. Subsequently, at level *D* (realization), this technical solution is further refined and documented for component realization in order to enable efficient industrialization. The results of all decomposition and abstraction steps span the Advanced System Model presented in section "The Advanced System Model".

By this, the Advanced System Model comprises the system architecture, the functional and logical architecture, as well as the technical and component architecture. As these architectures are semantically linked, the Advanced System Model provides all necessary documentation consistently for its augmentation with further relevant integrated system parts, e.g., the vehicle electric system or the integrated cooling system.

It may be criticized that MBCPSE leads to a large number of documents to be managed. However, based on a sound systems engineering approach, this helps to cope with the high complexity of a CPS as its architecture comprises the function-oriented view of a computer scientist (abstraction levels) and the engineering views of a mechanical and electrical engineer (decomposition layers, abstraction levels

C and *D*). Particularly it replaces the rather unstructured discussion during requirement elicitation which causes many industry projects to be delayed already in early project phases. All developers are guided by a systematic approach in an integrated architectural framework which is semantically linked, the Advanced System Model. There is an additional positive effect caused by the MBCPSE documentation. As all model interfaces of a CPS are well defined, the different technical elements are behaving like black boxes to each other. Thus, they can be internally changed without causing side effects in other parts of the CPS as long as their interface is not changed and behaves as before. All interface tests and other parts of the CPS can be (re)used as outlined before.

A further advantage is the complete and consistent Advanced Digital Twin obtained from the Advanced System Model as detailed in section "The Advanced Digital Twin". Since the Advanced Digital Twin contains all implemented architectures of the Advanced System Model, it creates a consistency for integrated engineering, production, and operational approaches. This facilitates, for example, industrialization, appropriate, flexible, and efficient responses to product changes and development of interface testing. Nevertheless, modeling a digital twin is always cost-intensive and needs special knowledge and, as a matter of fact, a lot of experiences as discussed in section "Model-Based: More than a Description". Domain experts in the mechanical industry, e.g., the automotive industry, often have little knowledge about standardized modeling languages for discrete systems like SysML and UML but excellent knowledge in modeling dynamical systems, e.g., with MATLAB/Simulink. Vice versa, computer scientists are usually not prepared to specify CPS beyond the functional architecture. Since experts who master mechanical and electrical engineering as well as computer science are rare, large CPS projects usually fail to meet their objectives as pointed out in section "Introduction".

Nonetheless, due to the high dependency of the functional and the technical architecture, MBCPSE also requires a few experts who master all three disciplines, the CPS architects. These CPS architects are the key players for a successful system architecture. However, the CPS architects shall not have to deal with the specific technical details but with the interfaces and the interferences of the parts of the Advanced System Model only, as outlined in section 3.1.3. The details of the functional architecture are assigned to a cross-functional team with a focus on computer science knowledge and the development of the technical architecture to a cross-functional team with a focus on engineering expertise. The realization of hardware and software is assigned to the respective expert teams. This leads to an easier exchange of expert knowledge. Furthermore, the independence of all development results within the CPS architecture enables simultaneous engineering and continuous integration which are the prerequisites for agile development, e.g., according to the SAFe framework. These modern working methods appeal to young people and emerging young disciplines, and these will again further develop the system's method.

In order to achieve such an integrated development environment, the system architecture of the Advanced Digital Twin enables implementing MBCPSE in every organization, by modeling the prevailing mechanical structure of the CPS.

Development processes can be optimized in accordance with the system architecture as long as product changes stay within the underlying framework, e.g., technology updates or innovations. Next to the process, often the hierarchical organization is also following the system architecture. This may impede flexibility for necessary product advancements and innovations which can be taken further when applying MBCPSE, as it enables the adaptation of product, organization, and culture within an existing system architecture.

What at first sight seems like an anachronism, because modern approaches start on a greenfield with the functional architecture and leave the system architecture open at the beginning (see section "Related Work"), is at second sight a suitable approach to address existing organizations (brownfield systems) (Hopkins and Jenkins 2008). However, the adoption of a prevailing system architecture, which often reflects the organizational structure (Colfer and Baldwin 2016), limits the functional and technical architecture. This circumstance can be overcome by consistently applying MBCPSE starting with the prevailing system architecture, semantically linking it to the functional architecture, and resulting in a technical architecture for industrialization (see Fig. 4).

The function-oriented view will additionally provide new potentials for efficiencies which can lead to a completely different approach to structure processes and organization. In conclusion, it can be observed that since the last decades the product decomposition in engineering domains is usually structured by geometric interfaces. Due to countless optimization steps, this led to mirrored structures in organization and responsibilities. All other "minor" disciplines involved had to follow this prevailing structure and to bow their architectural needs and requirements to the still ongoing economic success story. This subordination led to relatively weak results in the product contributions of the electric engineering and computer science domain.

If the entire development organization shall be changed to follow an effective function-oriented approach, MBCPSE supports this objective by smoothly transferring the system architecture to the technical architecture. The iterative and agile application of MBCPSE development cycles leads to a function-oriented product development process. At this stage, the functions are developed at level *A* and level *B*, and level *C* and level *D* take continuously over for the technical solution. Hence, the technical architecture substitutes the not any longer needed initial system architecture. The functional architecture is free of initial technical limitations, as required by the mentioned greenfield approaches. It is a question of economic efficiency to adapt the organization to the function-oriented systems engineering approach.

In any case, such an extensive technical change implies an extensive cultural change that takes a serious amount of time and budget. In addition, it also requires a huge amount of discipline and management commitment to stay with the applied architecture and to incorporate upcoming changes properly. However, the application of MBCPSE definitely pays off when the Advanced System Model is reused in order to develop a couple of Advanced Digital Twins, by which the time to market is reduced significantly and task forces are avoided. Most importantly, it prevents an organization from becoming a Dinosaur. Modern agile processes are implemented,

and the continuous achievement of reachable targets is mutually recognized and appreciated. The efficiency and effectiveness of each individual work are increasing as personnel continuously evolve within a cooperative agile culture.

Last but not least, the customer will appreciate the consistent and reliable behavior of the CPS which can be functionally continuously maintained and extended over the air avoiding cost-intensive major defects and failure.

Cross-References

- ▶ Condition Based Maintenance using Digital Twins
- ▶ Exploiting Digital Twins in MBSE to Enhance System Modeling and Life Cycle Coverage
- ▶ MBSE for Systems and SoS Integration / MBSE for Network Systems
- ▶ Model-Based Hardware-Software Integration
- ▶ Model-Based Requirement Elicitation and Tradeoff Analysis

References

P. Bibow, M. Dalibor, C. Hopmann, B. Mainz, B. Rumpe, D. Schmalzing, M. Schmitz, A. Wortmann Model-Driven Development of a Digital Twin for Injection Molding, International Conference on Advanced Information Systems Engineering (CAiSE'20), ser. Lecture Notes in Computer Science, S. Dustdar, E. Yu, C. Salinesi, D. Rieu, V. Pant, 12127. Springer International Publishing, 2020. 85–100.

B. W. Boehm, "Guidelines for Verifying and Validating Software Requirements and Design Specifications, Euro IFIP 79, P. A. Samet North Holland, 1979, 711–719.

G. Booch, J. Rumbaugh, and I. Jacobson, The unified modeling language user guide: Covers UML 2.0 ; thoroughly updated, the ultimate tutorial to the UML from the original designers, 2nd ed., ser. Safari Books Online. Upper Saddle River, NJ: Addison-Wesley, 2005. [Online]. Available: http://proquest.tech.safaribooksonline.de/032126797426

M. Brambilla, M. Wimmer, and J. Cabot, Model-driven software engineering in practice, ser. Synthesis lectures on software engineering. San Rafael, Calif.Morgan & Claypool, 2012, 1.

P. Braun, M. Broy, F. Houdek, M. Kirchmayr, M. Müller, B. Penzenstadler, K. Pohl, and T. Weyer, "Guiding requirements engineering for software-intensive embedded systems in the automotive industry: The REMsES approach," Computer Science - Research and Development, vol. 29, no. 1, pp. 21–43, 2014.

M. Broy, "Challenges in Automotive Software EngineeringProceedings of the 28th International Conference on Software Engineering, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 33–42.

M. Broy, M. V. Cengarle, and E. Geisberger, "Cyber-Physical Systems: Imminent ChallengesLarge-scale complex IT systems, ser. Lecture Notes in Computer Science, R. Calinescu D. Garlan, Berlin: Springer, 2012,. 7539, 1–28.

L. J. Colfer and C. Y. Baldwin, "The mirroring hypothesis: Theory, evidence, and exceptions," Industrial and Corporate Change, vol. 25, no. 5, pp. 709–738, 2016.

O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, Structured programming. GBR: Academic Press Ltd, 1972.

Dassault Systemes. 03.11.2020. Catia - Computer Aided Three-Dimensional Interactive Application: https://www.3ds.com/de/produkte-und-services/catia/.

I. Drave, T. Greifenberg, S. Hillemacher, S. Kriebel, E. Kusmenko, M. Markthaler, P. Orth, K. S. Salman, J. Richenhagen, B. Rumpe, C. Schulze, M. Wenckstern, and A. Wortmann, “SMArDT modeling for automotive software testing,” Software: Practice and Experience, vol. 49, no. 2, pp. 301–328, 2019.

I. Drave, T. Greifenberg, S. Hillemacher, S. Kriebel, M. Markthaler, B. Rumpe, and A. Wortmann,“Model-Based Testing of Software-Based System Functions,” Conference on Software Engineering and Advanced Applications (SEAA’18), 2018, 146–153.

EAST-ADL Association. 2013. EAST-ADL Domain Model Specification version V2.1.12. [Online]. Available: http://www.east-adl.info/Specification.html

European Space Agency. 03.11.2020. Ariane-5: Learning from Flight 501 and Preparing for 502. [Online]. Available: www.esa.int/esapub/bulletin/bullet89/dalma89.htm

P. Giusto, R. S., and S. M., “Modeling and Analysis of Automotive Systems: Current Approaches and Future Trends, Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development. SCITEPRESS - Science and Technology Publications, 2016, pp. 704–710.

Granrath, C., et al. The next generation of electrified powertrains: Smart digital systems engineering for safe and reliable products,SIA PARIS 2019 - Power Train & Electronics, 2019.

K. Hölldobler, J. Michael, J. O. Ringert, B. Rumpe, and A. Wortmann, "Innovations in model-based software and systems engineering," The Journal of Object Technology, vol. 18, no. 1, pp. 1–60, 2019.

R. Hopkins K. Jenkins, Eating the IT elephant: Moving from greenfield development to brownfield, ser. Safari Books Online. Upper Saddle River, N.J: IBM Press/Pearson plc, 2008.

W. S. Humphrey, Managing the software process, 28th ed., ser. The SEI series in software engineering. Boston: Addison-Wesley, 2002.

ISO International Organization for Standardization, “ISO/IEC/IEEE 15288–1:2015–05 Systems and software engineering: System life cycle processes,” Berlin, 2015.

ISO International Organization for Standardization, “ISO/IEC/IEEE 24765:2017–09: Systems and software engineering | Vocabulary,” Berlin, 2017.

ISO International Organization for Standardization. 2018. ISO 26262-10:2018: Road vehicles - Functional safety - Part 10: Guidelines on ISO 26262 Berlin

D. L. Johnston, “Scientists become managers-the ‘t’-shaped man,” IEEE Engineering Management Review, 6, 3, 67–68, 1978.

J. C. Kirchhof, J. Michael, B. Rumpe, S. Varga, A. Wortmann,“Model-driven Digital Twin Construction: Synthesizing the Integration of Cyber-Physical Systems with Their Information Systems,” Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. ACM, 2020, 90–101.

R. Knaster and D. Leffingwell, SAFe 4.5 distilled: Applying the scaled agile framework for lean enterprises. Boston: Addison-Wesley, 2019.

S. Kriebel. 2018. Pains in Modeling: SysML-based Deployment in an Engineering Domain: Invited Talk at 1st Workshop on Pains in Model-Driven Engineering Practice, Conference on Model Driven Engineering Languages and Systems (MODELS' 18): https://sites.google.com/view/pains-2018/home. Copenhagen, Denmark.

S. Kriebel, V. Moyses, G. Strobl, J. Richenhagen, P. Orth, S. Pischinger, C. Schulze, T. Greifenberg, and B. Rumpe, "The next generation of BMW's electrified powertrains: Providing software features quickly by model-based system design,"26th Aachen colloquium automobile and engine technology, 2017.

S. Kriebel, J. Richenhagen, C. Granrath, and C. Kugler, “Systems engineering with SysML the path to the future?” MTZ worldwide, 79, 5, 44–47, 2018.

G. Liebel, M. Tichy, and E. Knauss, “Use, potential, and showstoppers of models in automotive requirements engineering,” Software & Systems Modeling, vol. 18, no. 4, pp. 2587–2607, 2019.

A. Manifesto,Agile manifesto,Haettu, 14, 2012, 2001.

MathWorks. 02.07.2019. Simulink - Simulation und Model-Based Design: https://de.mathworks.com/products/simulink.html.

OMG Object Management Group. OMG Unified Modeling Language, v2.5.1: Version 2.5.1 2017. [Online]. Available: http://www.omg.org/spec/UML/2.5.1
OMG Systems Modeling Language (OMG SysML): Version 1.5. 2017. [Online]. Available: http://www.omg.org/spec/SysML/1.5/
K. Pohl, H. Hönninger, R. Achatz, and M. Broy, Model-based engineering of embedded systems: The SPES 2020 methodology. Berlin and Heidelberg: Springer, 2012.
B. Randell. 03.11.2020. NATO Software Engineering Conference 1968. Schloss Dagstuhl. [Online]. Available: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/NATOReports/
C. Read, "Dinosaurs and economic Darwinism," in The rise and fall of an economic empire: With lessons for aspiring economies. London: Palgrave Macmillan UK, 2010, pp. 256–270.
B. Rumpe, Agile Modeling with UML: Code generation, Testing, Refactoring. Springer International, 2017.
D. Scheithauer K. Forsberg, "4.5.3 V-Model Views," INCOSE International Symposium, 23, 1,. 502–516, 2013.
H. Stachowiak, "Allgemeine Modelltheorie," Wien: Springer, 1973.
D. D. Walden, G. J. Roedler, K. Forsberg, R. D. Hamelin, and T. M. Shortell, Systems engineering handbook: A guide for system life cycle processes and activities, 4th ed. Hoboken, NJ: Wiley, 2015.
J. Whittle, J. Hutchinson, and M. Rouncefield, "The state of practice in model-driven engineering," IEEE Software, vol. 31, no. 3, pp. 79–85, 2014.
H. Zimmermann, "OSI reference model-the ISO model of architecture for open systems interconnection," IEEE Transactions on Communications, vol. 28, no. 4, pp. 425–432, 1980.

Stefan Kriebel is technical director in the Business Unit Intelligent Mobility & Software and responsible for Future Mobility SW & EE Platforms at FEV Europe GmbH, Aachen, Germany. He studied Computer Science at TU Munich and received his Ph.D. from TU Munich with his thesis done on behalf of the Joint Research Centre (JRC) of the European Commission in Ispra, Italy, on time series analysis with artificial neural networks. He was with the BMW Group in various innovative projects assigned as responsible manager to several fields like electric power train, driving dynamics software, driver assistance software, and systems engineering. Earlier he was responsible manager for the software test of smart cards at Giesecke & Devrient GmbH, Munich, and with the Software Quality Department of the European Space Agency (ESA) at the European Space Research and Technology Centre (ESTEC), Noordwijk, Netherlands.

His main interest is state-of-the-art systems engineering integrating engineering and computer science disciplines, enhanced by digital transformation, agile transformation, and cyber-physical systems engineering. He is the author and co-author of several articles on these topics.

Matthias Markthaler is a function specialist at the BMW Group responsible for the test and integration of the vehicle energy system and the e/e architecture. He is currently pursuing a Ph.D. degree with his thesis on "model-based method for automated test case creation in the automotive industry based on a systems engineering approach" on behalf of the BMW Group as a Ph.D. fellow and the Department of Software Engineering at the RWTH Aachen University.

Since 2016, he has been working in different departments of integration and testing at the BMW Group. During this time, he was jointly responsible for the development and rollout of a model-based systems engineering method in the area of the electric drive system. He studied at the Federal University of São João del-Rei, Brazil, and at the Munich University of Applied Sciences, Germany, where he received his B.Sc. and M.Sc. degree in electrical engineering and information technology. His current research interests include the cooperation of the different disciplines in model-based systems engineering, the transformation to model-driven systems engineering, and cyber-physical systems engineering. He is the author and co-author of several articles on these topics.

Christian Granrath received his B.Sc. degree in mechanical engineering in 2014 and his M.Sc. degree in energy engineering in 2016 from RWTH Aachen University, Aachen, Germany. He is currently pursuing the Ph.D. degree in software and systems engineering at the Junior Professorship for Mechatronic Systems for Combustion Engines, RWTH Aachen University. As group leader at RWTH Aachen University, he is supporting the lecture "Software Development for Combustion Engines." In 2019, as a research associate at the University of Applied Sciences Aachen, he conducted a scientific training in systems engineering and agile development within the project "ERASMUS+ UNITED" to realize a knowledge transfer between European and Indonesian universities. His research interests include the fields of model-based and feature-driven systems engineering, agile software engineering, software architecture development and evaluation, as well as simulation model development for XiL applications in the automotive domain.

Johannes Richenhagen is vice president of Intelligent Mobility & Software at FEV Europe GmbH, where he previously held a number of responsible management positions. He studied mechanical engineering at the RWTH Aachen University, Germany, where he also received his Ph.D. with a thesis "Control Software Development for Automotive Powertrains with Agile Methods." He is assigned as associate lecturer on Software Development at the Junior Professorship for Mechatronic Systems for Combustion Engines, RWTH Aachen University.

His main interest is state-of-the-art systems engineering integrating engineering and computer science disciplines, enhanced by digital transformation, agile transformation, and cyber-physical systems engineering. He is the author and coauthor of several articles on these topics.

Bernhard Rumpe is heading the Software Engineering Department at the RWTH Aachen University, Germany. Earlier he had positions at INRIA/IRISA, Rennes, Colorado State University, TU Braunschweig, Vanderbilt University, Nashville, and TU Munich.

His main interests are rigorous and practical software and systems development methods based on adequate modeling techniques. This includes agile development methods like XP and SCRUM as well as model engineering based on UML-like notations and domain-specific languages. He has contributed to many modeling techniques, including the UML standardization. He also applies modeling, for example, to autonomous cars, human brain simulation, BIM energy management, juristical contract digitalization, production automation, cloud, and many more. In his projects, he intensively collaborates with all large German car manufacturers, energy companies, insurance and banking companies, a major aircraft company, a space company, as well as innovative start-ups in the IT-related domains.

He is author and editor of 36 books and editor in chief of the Springer international journal *Software and Systems Modeling* (www.sosym.org). His newest books *Agile Modeling with UML: Code Generation, Testing, Refactoring* and *Engineering Modeling Languages: Turning Domain Knowledge into Tools* were published in 2016 and 2017, respectively.