Gregor Hoepfner[1]
Institute for Machine Elements
and Systems Engineering,
RWTH Aachen University,
Schinkelstraße 10,
52062 Aachen, Germany
e-mail: gregor.hoepfner@imse.rwth-aachen.de

Imke Nachmann
Software Engineering,
RWTH Aachen University,
Ahornstraße 55,
52062 Aachen, Germany
e-mail: nachmann@se-rwth.de

Thilo Zerwas
Institute for Machine Elements
and Systems Engineering,
RWTH Aachen University,
Schinkelstraße 10,
52062 Aachen, Germany
e-mail: thilo.zerwas@imse.rwth-aachen.de

Joerg K. Berroth
Institute for Machine Elements and Systems
Engineering,
RWTH Aachen University,
Schinkelstraße 10,
52062 Aachen, Germany
e-mail: joerg.berroth@imse.rwth-aachen.de

Jens Kohl
BMW Group AG,
Knorrstraße 147,
80788 Munich, Germany
e-mail: jens.kohl@bmw.de

Christian Guist
BMW Group AG,
Knorrstraße 147,
80788 Munich, Germany
e-mail: christian.guist@bmw.de

Bernhard Rumpe
Chair of Software Engineering,
RWTH Aachen University,
Ahornstraße 55,
52062 Aachen, Germany
e-mail: rumpe@se-rwth.de

Georg Jacobs
Institute for Machine Elements and Systems
Engineering,
RWTH Aachen University,
Schinkelstraße 10,
52062 Aachen, Germany
e-mail: georg.jacobs@imse.rwth-aachen.de

Towards a Holistic and Functional Model-Based Design Method for Mechatronic Cyber-Physical Systems

Engineering cyber-physical systems (CPS) is complex and time-consuming due to the heterogeneity of the involved engineering domains and the high number of physical and logical interactions of their subsystems. Model-based systems engineering (MBSE) approaches tackle the complexity of developing CPS by formally and explicitly modeling subsystems and their interactions. Newer approaches also integrate domain-specific models and modeling languages to cover different aspects of CPS. However, MBSE approaches are currently not fully applicable for CPS development since they do not integrate formal models for physical and mechanical behavior to an extent that allows to seamlessly link mechanical models to the digital models and reuse them. In this paper, we discuss the challenges arising from the missing integration of physics into MBSE and introduce a model-based methodology capable of integrating physical functions and effects into an MBSE approach on a level where detailed physical effects are considered. Our approach offers a fully virtual, model-based development methodology covering the whole development process for the development of CPS. Evaluating this methodology on a real automotive use case demonstrates benefits regarding virtual development and functional testing of CPS. It shows potentials regarding automated development and continuous integration of the whole CPS including all domains. As an outlook of this paper, we discuss potential further research topics extending our development workflow. [DOI: 10.1115/1.4056807]

Keywords: cyber-physical system design and operation, functional modeling, information management, knowledge engineering, model-based systems engineering

Introduction

Missing Cross-Domain Collaboration Makes Cyber-Physical Systems Development Complex and Time-Consuming. In various areas of industry, such as automotive or aerospace, nowadays more and more cyber-physical systems (CPS) are developed. CPS consist of mechanical, electrical, electronic, and software components leading to a large net of both physical and logical

[1]Corresponding author.

Contributed by the Computers and Information Division of ASME for publication in the Journal of Computing and Information Science in Engineering. Manuscript received April 14, 2022; final manuscript received January 23, 2023; published online March 29, 2023. Assoc. Editor: Chris Paredis.

interdependencies between the components and domains. Hence, systems are getting more and more complex, while a shorter time to market demands faster development of CPS [1].

The general complexity of CPS and their cross-domain character makes engineering CPS challenging. In addition, CPS often take over safety-relevant tasks. Therefore, possible faulty behavior has to be detected, the fault's root cause has to be inferred and failures have to be prevented or mitigated [2]. Failures often result from missing consideration of interactions between subsystems. This is especially critical in CPS due to the high amount of cross-domain interdependencies. Hence, guaranteeing function fulfillment for CPS without failures of single subsystems or the overall system results in cost- and time-intensive development processes.

A major reason for missing consideration of interactions is different development processes in the domains. Historically, each domain has developed its systems in a different manner and on different platforms. Due to the different character of the domain systems—e.g., high lead times in mechanics or the possibility to update in software—different development processes have evolved. Systems engineering (SE) is an approach to overcome these domain processes and identify interdependencies in complex systems. SE faces many challenges ranging from the need for collaboration of experts from different backgrounds to the assurance of function fulfillment at the system level with high efficiency and quality [3].

Complex systems require digital SE approaches supported by automation to provide verification and validation of systems while remaining efficient. Due to the heterogeneous character of CPS, experts from various domains need to collaborate to provide functional verification and validation at the system level. However, validation of mechanical components and systems is done predominantly on physical prototypes since current virtual methods do not cover all relevant domains. A virtual verification and validation of the full CPS including mechanics is thus not possible and a common collaboration platform to describe the entire CPS including all domains in a formal, i.e., mathematical and thereby machine-interpretable, manner is not given. As virtual validation is often faster than using physical prototypes, using formal models in all domains can accelerate development.

One reason for missing formality in mechanics is a conceptual gap between the problem domain and the solution domain that has to be overcome [4,5]. The conceptual gap is especially high for mechanical engineering, as the product is geometry-centric, while CPS requirements are functional.

Model-Based Systems Engineering Including Mechanical Functions Offers a Facilitation of the Cyber-Physical Systems Development Process. Overcoming the conceptual gap in mechanical engineering has been investigated in the past centuries. The ideas of mechanical design theory [6,7] provide a foundation for systems engineering. Especially, Ref. [6] comprises the concept of functional decomposition and the reuse of mechanical solutions, to realize mechanical functions. Elementary mechanical solutions are structured based on physical effects, which act between components' active surfaces and are called principle solutions.

Software engineering has successfully overcome the conceptual gap using model-based software engineering approaches [5,8]. Several studies have investigated the benefits of model-based software development [9,10] and show that considering systems as networks of interacting encapsulated subsystems leads to enhanced system quality especially when utilized at the early stages of development.

Model-based systems engineering (MBSE) has developed to be a cross-domain approach, in which models instead of documents become the primary development artifacts for the entire system and all domains. Formal models of systems enable automatic verification, validation, and testing and also analyses and transformations [11].

However, mechanical functions are not yet well integrated into MBSE. Encapsulation of functions in software engineering can be done without the consideration of physical and geometric boundaries, as only information is transferred between functions. In mechanical systems, one function is realized by multiple physical components and additional flows, i.e., energy and material, are transferred. Hence, functional cutting requires the consideration of physical and geometric restrictions from components. Thus, the link between component and function is not directly given. So far, this has kept mechanical engineering from formal functional modeling. Integrating mechanics into formal modeling is the next step in providing a cross-domain description of CPS and their interdependencies.

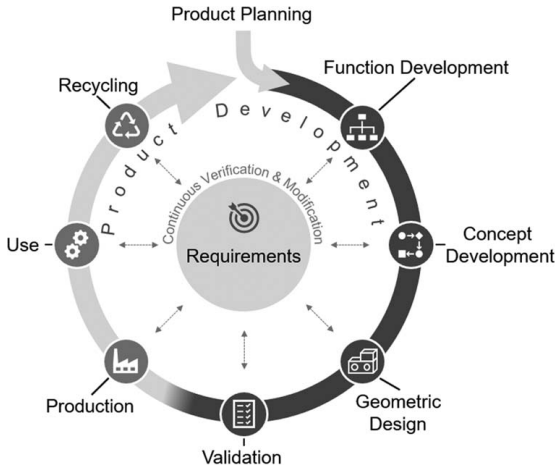

Fig. 1 Steps of the product life cycle

The proposed methodology of [6] has the potential to overcome this challenge. It is compatible with the idea of MBSE and has the potential to provide—to the far of our knowledge—the first approach for fully formalizing mechanical functions and integrating mechanics in common CPS development. However, it is not yet formalized fully.

Still, a gap arises in mechanical systems from restrictions set by laws of nature and suitable formal modeling techniques to integrate physical models to an extent that they can be reused, and tested continuously within development processes. To consider the interplay of physical effects and geometric components inherent to mechanical systems, the existing methods, concepts, and languages of MBSE need extensions.

The methodological weaknesses in holistic CPS engineering give rise to challenges in all phases of the product lifecycle, increasing problems arising from the enlarging complexity of today's CPS.

This contribution aims to present a novel approach that formalizes mechanical development to overcome the conceptual gap in mechanical engineering and to integrate mechanics into MBSE, with a focus on the specific challenges in the design phase.

Overview of the Contribution. In Sec. 2, an automotive-industry-specific review identifies current challenges for product development arising from MBSE approaches missing the mechanical domain. Section 3 presents the state of research regarding model-based development methods showing that mechanical functions are not properly integrated into current MBSE approaches. In Sec. 4, we present our approach. Section 5 discusses the potentials of the approach and in Sec. 6 open points towards an industrialized approach are given before Sec. 7 concludes.

Challenges in Mechatronic Cyber-Physical Systems Development

The complexity of engineering CPS increases by the need for collaborative and distributed development [12,13]. This demands for

adapting the methods established in each of the domains to cross-domain collaboration. Especially, integrating the mechanical domain reveals several challenges that affect the entire product life cycle. These challenges are discussed in the following section.

The Product Lifecycle From a Mechanical Perspective. The product lifecycle includes all phases that a product experiences: Product planning and development, production, use, as well as recycling (cf. Fig. 1). Product development significantly determines how a product can be manufactured, used, and recycled as well as to which degree it meets the customer's requirements [7].

In mechanical engineering, product development includes five steps that can be run in parallel and iteratively. **Requirements** are modified throughout development based on findings and changing constraints and serve as a reference for all development decisions. **Function development** is concerned with specifying the system as a hierarchical network of functions that interact through function flows. In **concept development**, engineers seek solutions for each (sub-)function while satisfying requirements. In **geometric design**, engineers evolve the selected solutions into components and assemblies. The activities during **validation** integrate all elements of the system and ensure that the whole system meets all customer requirements [7].

Challenges in Today's Development Process. The increasing complexity of modern systems on the one hand and strict requirements on legislation, functional safety, sustainability, and economic constraints on the other hand pose new, fundamental challenges for the product lifecycle [7]. Within an interdisciplinary research project involving experts from the automotive industry, we have identified three major challenges the automotive industry faces throughout the product lifecycle:

Challenge 1: Nowadays, innovation is driven more by functions and features than physical components

Current technological developments often extend well-established systems with new, additional functionalities, e.g., interconnectivity or autonomous driving in automotive systems. The new functions are often not enabled by a specific mechanical component, but by the interaction of multiple, heterogeneous subsystems [7]. While in other domains there are approaches to describe interfaces in a sufficient manner [14], especially in mechanical engineering, development teams are usually organized according to geometric modules and distinguished by geometric interfaces. These geometry-specific development teams can improve functions of the individual module incrementally but enabling innovative system functions and features requires a function-oriented, system-wide approach. The authors in Ref. [15] state that systems designed in organizations are usually a copy of the organizations structure. Therefore, in geometrically organized teams, systems may be often designed with a focus on geometry.

Challenge 2: A holistic workflow throughout product development linking the development artifacts is missing

Increasing system complexity has led to an increased number of development artifacts, i.e., documents and models that are created throughout the product lifecycle to provide descriptions of the system under development from various viewpoints. Due to the large number and varying formats of these artifacts, it is an effortful and time-consuming process for humans to maintain consistency. Further, updates to one artifact require updating depending artifacts manually. However, changes in artifacts often cannot be tracked and propagated automatically, as documents are not available in a machine-interpretable format. As artifacts continue to proliferate and become more complex, traditional document-based lifecycle management approaches reach their limits and the time to keep overall consistency and non-redundancy of the product data increases significantly, decelerating product development.

Challenge 3: Demand on accelerated development and increasing product quality

Globalized competition forces companies to offer new products with more functionality in a shorter time span, creating the need to decrease the development time significantly. At the same time, the quality demands of customers and legislators increase [7]. The high-quality demands challenge to identify and correct as many failures as possible, creating iterations in product development, which are time-consuming. Hence, to reduce development time, failures have to be identified early on to prevent such iterations.

Consequences of the Challenges Throughout the Development Process. The challenges listed above impact the entire product lifecycle. To outline all the manifestations of each challenge on the mechanical domain at the respective stages of the lifecycle in more detail, we have aggregated them in Table 1. Three major manifestations can be identified.

The *conceptual gap* between functions and components is a major obstacle in mechanics that has not yet been fully overcome. On the one hand, the functional view of mechanical products is often too weakly developed: there is a lack of a systematic description of functional requirements, from which an architecture of product functions can be derived and function-oriented solution concepts can be identified. On the other hand, a strong, formal linkage of this functional view with the existing architecture of physical components is still missing.

In addition to the linking of functions and physical components, the fundamental and consistent *connection of all relevant data and development artifacts* is missing. A wide variety of models and data sets are generated in all lifecycle phases from development through production to use. To ensure consistency as well as efficient change and further development cycles, all relevant dependencies between the artifacts must be formally mapped and usable.

A major use case for linked artifacts is *virtual verification and validation*. On the one hand, this lacks a functionally testable description of the subsystems including their functional interfaces, and on the other hand, there is a lack of completeness on the part of the functional requirements against which testing is performed. Due to this deficit, full virtual validation is not possible up to now, and instead performed by time-consuming and cost-intensive field tests.

State of Research

To solve the three identified challenges described in Sec. 2 for modern product development, we identify three major fields of research.

Regarding a holistic workflow linking artifacts in product development (challenge 2), explicit modeling of interdependencies between development artifacts of different domains is key. Software engineering has developed model-based software engineering methodologies to master the complexity of their respective systems [16,17].

To improve function development for mechanics (challenge 1), a function-oriented development method for mechanics helps to close the gap between functional requirements and components.

Accelerated product development with increasing product quality (challenge 3) requires virtual identification of errors and holistic virtual validation using simulations before using physical prototypes, as iteration processes are faster in a virtual way. Thus, formal modeling also including mechanics is an enabler, as the system description becomes machine-readable and thereby opens for (semi-)automated verification and validation.

We describe the research fields model-based approaches, function-oriented development, and formal modeling in mechanics in the following subsections.

Key Concepts of Model-Based Approaches in the Software Domain. Software engineering has developed model-based

Table 1 Challenges in cps development throughout the product life cycle

Product life cycle phases	Challenges		
	1. Function-driven innovation	2. Missing holistic workflow	3. Accelerated development
Requirements	Requirements are often collected in documents that specify geometries instead of an unambiguous description of the functional behavior.	Requirements are usually not linked to each other or other development artifacts hindering consistency and transparency. If linked in requirements engineering tools, consistency is often not ensured.	To reduce iterations and late changes, it must be possible to identify relevant requirements as early as possible and change them efficiently later on.
Function development	Mechanical engineers have centered their development on physical components, complicating the collaboration with function-oriented domains.	For parallel and virtual development of CPS, its functions must be known with their interdependencies and interactions, for which there is no formal method today.	Functional interfaces between development teams are not always fully defined and sometimes do not match organization structures, impeding the system integration process.
Concept development	Concept development in industry is often still based on the individual knowledge of experts and does often not consider a wide range of concepts.	Concepts are often described by heterogeneous and unlinked development artifacts, e.g., sketches, CAD models, or analytical equations.	Reusing concepts in future projects without reusing the same components is often difficult and slows down concept development.
Geometric design and optimization	Allocating functions in a mechanical component architecture is complex and often not methodically supported. Thus, formal function-oriented development is mostly done on the top-system-level.	Artifacts for geometric design, e.g., CAE models, are often not linked to other artifacts from other phases in a way, that virtual validation and data consistency can be ensured automatically.	Reduced development time restricts the number of manual design iterations and increases the need for machine-readable descriptions of physical concepts.
Verification and validation	The typical component-oriented development focuses on verifying non-functional requirements instead of functions, which can only be validated at the overall system level.	The behavior of components and their interactions rely on physical effects which, up to now, cannot be modeled entirely. Thus, full validation is only possible in field tests and not yet virtually.	
Production	Predicting the impact of changes from production to functional behavior and vice versa is complex and has to be verified in real hardware due to missing links between both domains.	Due to shorter time to market, product changes are often developed and introduced to production in parallel. Keeping production and product consistent is effortful and time-consuming, without seamless linking.	
Use	Using field and service data to update existing and improve future product functions requires linking the collected data with development artifacts.	Continuous optimization of products after market-launch is an advantage for software systems but is inherently difficult for mechanical systems.	
Recycling and afterlife	To identify the most sustainable recycling decision for parts or functions of a product instance and accelerate reuse in second life scenarios, evaluation of field data instance within the virtual models from product development is required.		

approaches to specify software systems. The following summarizes the key techniques they apply [8,18,19]:

Abstraction from technical details allows to capture domain concepts comprehensibly and also facilitates collaborative development in (spatially) distributed teams.

Formality by employing domain-specific languages enables automatic verification, analyses, and transformation of artifacts to enhance efficiency at all development stages by detecting possible system failures in an automated way. Techniques for model-based testing enable fast verification and validation.

Encapsulation defines reusable components which communicate solely via defined interfaces while shielding their internal behavior from the outside. Reusing these components and their composition reduces development time and costs significantly since only the interface and the interplay with the rest of the system have to be validated. Encapsulation enables individual development of the single components. Through *Composition*, systems become modular as composition of their subsystems.

Abstraction, Formality, Encapsulation, and Composition facilitate the development of large complex software systems in distributed teams. They have proven to enhance efficiency through the reuse of components significantly and enable parallelization of development tasks due to clear interface definitions [16,17].

Function-Oriented Development. Theories for automatic validation and verification typically understand systems as a network of interacting functions [8,12,20]. Languages with semantics built with these theories do have physical aspects of CPS in mind, and consider continuity, or geometry [21] but do not yet integrate physical aspects from mechanical engineering and software engineering.

The FOCUS theory [8,22] regards systems as networks of interacting, stream processing functions, where a stream is a function from time domain to a set of messages. Modularity is achieved because components encapsulate a functional behavior and interact via their interfaces only. So far, FOCUS does regard continuous streams [23] but has not yet been applied for modeling functional behavior that arises from physical effects.

The SMArDT method [24] finds its applications in automotive software engineering, and in particular testing [9,25]. The method relies on formalized models in the systems modeling language (SysML) that represent the system under development on four levels of abstraction. The method, however, does not yet offer modeling techniques to include functions and solutions considered in the mechanical domain.

In the past decades, SE methods have evolved to MBSE methods especially in the context of space missions. Lately, in NASA's Europa Project, efforts have been made to design an architecture including requirements, functions, and concepts to realize MBSE in SysML and link it with further design artifacts [26,27]. This approach has mainly been applied to electric and software components or focused higher-level mechanical systems—e.g., for obtaining mass equipment lists [28]—instead of a detailed focus on the mechanical domain and mechanical behavior.

As mentioned in Sec. 1, Refs. [6,29] offer the potential to integrate mechanics into common CPS development. They establish a functional architecture from which solutions can be derived as realizations of the functions. It is postulated that every mechanical behavior can be described by a network of elementary functions. The elementary functions describe transformations of functional flows, which are not further decomposed and are connected via incoming and outgoing functional flows. The author in Ref. [6]

proposes that in the mechanical domain physical effects realize functions. The effects act between two active surfaces from specific materials. Supplementing a suitable physical effect by active surfaces and specifying their material, yields a principle solution. In the remainder of this paper, we follow this definition. The significant discovery is that the sets of both elementary functions and physical effects are finite. In Ref. [29], both sets and their links are listed in a document-based catalog [6,29].

In Ref. [29] principle solutions are often depicted by a sketch and textual description, or by an equation. This representation does not meet the demands of model-based CPS engineering, as sketches and descriptions are informal and not machine-readable. This prevents verification and validation support through automation and hinders reusing functions and principle solutions.

Formal Modeling of Functions in Mechanical Domain. In recent research, there have been several approaches to formalize the concept of principle solutions for MBSE or describe mechanical realizations of a function with SysML. The authors in Ref. [21] integrate manual sketches of concepts into SysML. However, since sketches are no formal description, the same disadvantages continue to exist as with [6,29]. Other approaches describe the realization of a function by physical components [13,30,31]. Thereby, similarities of components that fulfill the function with the same principle but slightly modified geometry are not formally modeled. In addition, engineers need to design the component geometry in detail before testing if the realization meets the requirements. Effectively, this results in delayed collaboration with experts of other domains.

Another approach for modeling principle solutions using equations for physical laws and parameters for geometric and physical quantities is proposed in Ref. [32]. It remains unclear whether the principle solution will be linked to other development artifacts, e.g., requirements, functions, components, or simulation models. Hence, the principle solution cannot be validated directly with behavior describing models [33–35] against functions or requirements.

A promising approach capturing the concepts of functional structures, elementary functions, principle solutions, and their constituents together with their relations from a language engineering point of view is presented in Ref. [4]. The proposed meta-model specifies an abstract syntax for modeling languages that enable to capture these elements in a formal, machine-processable model whose elements are open for reuse. Also, the authors have encoded the meta-model as a SysML profile which demonstrates its applicability for holistic CPS development conforming to the SysML standard. The approach relies on reusing existing models for an automated validation. The design of geometric components, however, has not yet been integrated into a development methodology.

All in all, to address the identified challenges, a model-based, function-oriented approach is required, that enables modeling according to the four key techniques for all domains involved in a CPS. To the best of our knowledge, mechanics have not yet been integrated into model-based approaches in a sense that formal modeling of mechanical functions and solutions is possible. An integrated approach to model, test, and reuse mechanical solutions efficiently is missing. So, as to the best of our knowledge [4,36,37] showed the first approach for a fully formalized approach covering all domains. In the remainder of this paper, we adapt from [4,36,37] and extend them to a full methodology covering the whole lifecycle of CPS.

Approach

To enable efficient communication and documentation between experts from heterogeneous backgrounds regarding the state of development of CPS, engineers need system specifications, i.e., descriptions of the system that are precise and detailed, yet abstract from the details that do not contribute to the understanding. In MBSE approaches these descriptions are given as models [22].

Models need to be linked to prevent ambiguities and redundancy. Creating such models requires a paradigm that is understood by the involved domains, allows to modularize the engineering task to facilitate development in distributed teams and in parallel, and, at the same time, admits realizations by different technical means [5,22]. The *functional specification-paradigm* seems to do the trick since it is present in mechanical engineering [6,7], software engineering [8], as well as electrical and control engineering [12,20]: By transforming incoming to outgoing flows of energy, material, and information, a system defines a stream-processing [8] function. To perform such transformations, the system encapsulates a physical and a computational structure. Through its interfaces, the system is connected to its context.

The paradigm suggests a development process guided by three principles: (1) *Functions* are a universal construct and well-understood among the domains involved in development. Functions provide a universal, commonly understood, and formal description of the system under development, (2) *underspecification* [38] allows to add previously absent information as soon as it is available and to regard uncertainty in the system, product variability, the degrees-of-freedom for customizing a product, as well as behavioral non-determinism that occurs during system operation, and (3) *composition* [8] which allows dividing a complex engineering task into smaller sub-problems that can be solved individually possibly by reusing existing solutions.

Following the paradigm, our approach models *requirements* and derives a decomposed hierarchy of interacting *functions* from them. Functions are realized through *principle solutions* as stated in Ref. [6] comprising a physical effect and a set of active surfaces. In the *product*, the active surfaces are composed to geometric components.

The following sections demonstrate our approach that follows the above ideas and paradigms based on a running example. The approach is first explained on a design methodological level followed by a presentation of a formal modeling method. The modeling languages and methods proposed in Refs. [4,36,37] are used to relate the models across the stages systematically and yield a framework for specifying and testing the system seamlessly.

Running Example. We use the cooling circuit of an automotive combustion engine as a running example, cf. Fig. 2: the vehicle fulfills the main function of locomotion. For this purpose, the subsystem drivetrain supplies mechanical energy that is conducted to the wheels and transferred to the road. In vehicles with combustion engines, this involves converting chemical energy of the fuel into mechanical energy. For this purpose, the physical effect of combustion in the engine generates a thermal expansion of the fuel-air mixture, accelerating the piston, which transfers mechanical energy to the further drive system.

During the combustion process, some chemical energy is converted into thermal energy, which is induced into the engine's

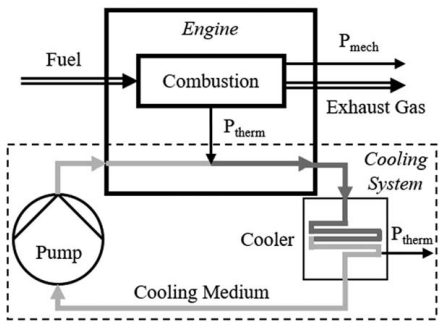

Fig. 2 Running example: The cooling circuit of a combustion engine, source: Ref. [4]

components, increasing their temperatures. The rising engine temperature is critical for both the component's material and the combustion process. Therefore, combustion engines are usually equipped with a liquid-based cooling system to regulate temperature. A cooling medium circulates, absorbs heat from the engine, and releases via the cooler to the environment. The cooling medium is conveyed by a pump to ensure its motion despite friction and pressure losses.

The coolant pump of the example system is operated electrically and can thus be set to a certain rotational speed by a control unit. For simplicity, the engine itself is reduced to the subsystems cylinder head (CH) and crankcase (CC), which both require different temperature levels. The coolant pump and cooling circuit can be assumed as mechatronic functions and systems of the whole thermal management system, which itself provides cyber-physical functions, e.g., remote control of engine temperatures or over the air update. For developing the cyber-physical functions an understanding of all subfunctions is required, including physical functions.

Design Methodology. For developing a system seamlessly, it is required to transform system requirements to a reliable product that satisfies such requirements. To ease this transformation process, design methodology has developed a method of first developing functions from requirements, find suitable solutions for each function, and then develop a product from these solutions [7]. Our approach is based on this method and described in Fig. 3. In our approach, functional and design requirements are developed from stakeholder requirements. We derive functions from functional requirements and decompose the functions into sub-functions and elementary functions, which are not further decomposed, following the approach of Ref. [6]. For each function, a solution is developed. For elementary functions, these are principle solutions. Principle solutions consist of physical effects acting between active surfaces. From the active surfaces, components are finally designed. The methodological approach for each of the layers *requirements*, *functions*, *principle solutions*, and *product* is described in the following subsections.

Requirements. Meeting the stakeholder requirements determines the quality of the final product and therefore requires thorough verification and validation of the implemented requirements on all system levels.

Different stakeholders can state additional requirements at any time during the development process. Such requirements are usually broken down to technical requirements. An efficient development process requires repeatable verification and validation of the current state of the development with respect to the growing set of technical requirements at all times. For different types of requirements, verification, and validation steps may vary. Therefore, our approach relies on early stage requirements structuring.

We differentiate between *functional requirements* and *design requirements* that arise during all stages of the product's life cycle [39]. Functional requirements, on the one hand, express expectations about the desired behavior of the system. Design requirements, on the other hand, express constraints as structural or geometric expectations, e.g., on the value ranges of system parameters.

Regarding the running example a functional requirement is that the cooling system shall be able to remove heat from the engine, if the engine's temperatures rise too high—a functional description of intended behavior. A design requirement is given by the optimal range of the cylinder head's operating temperature being given between 120 °C and 130 °C.

Functions. After developing and structuring requirements, functional decomposition is the next step. Here, the product relevant functions are defined and decomposed to sub-functions. Functions in this method describe transformation operations of the functional flows energy, material, and information, that the system under development shall be able to perform. Functions therefore consist of (1) interfaces comprised of the incoming and outgoing typed flows of energy, material, and information, and (2) a functional behavior, that defines how the input is transformed to the output such that, e.g., the conservation laws are obeyed [4].

As stated in Refs. [6,40], the behavior of a function is either defined through a composition of sub-functions or by a formula

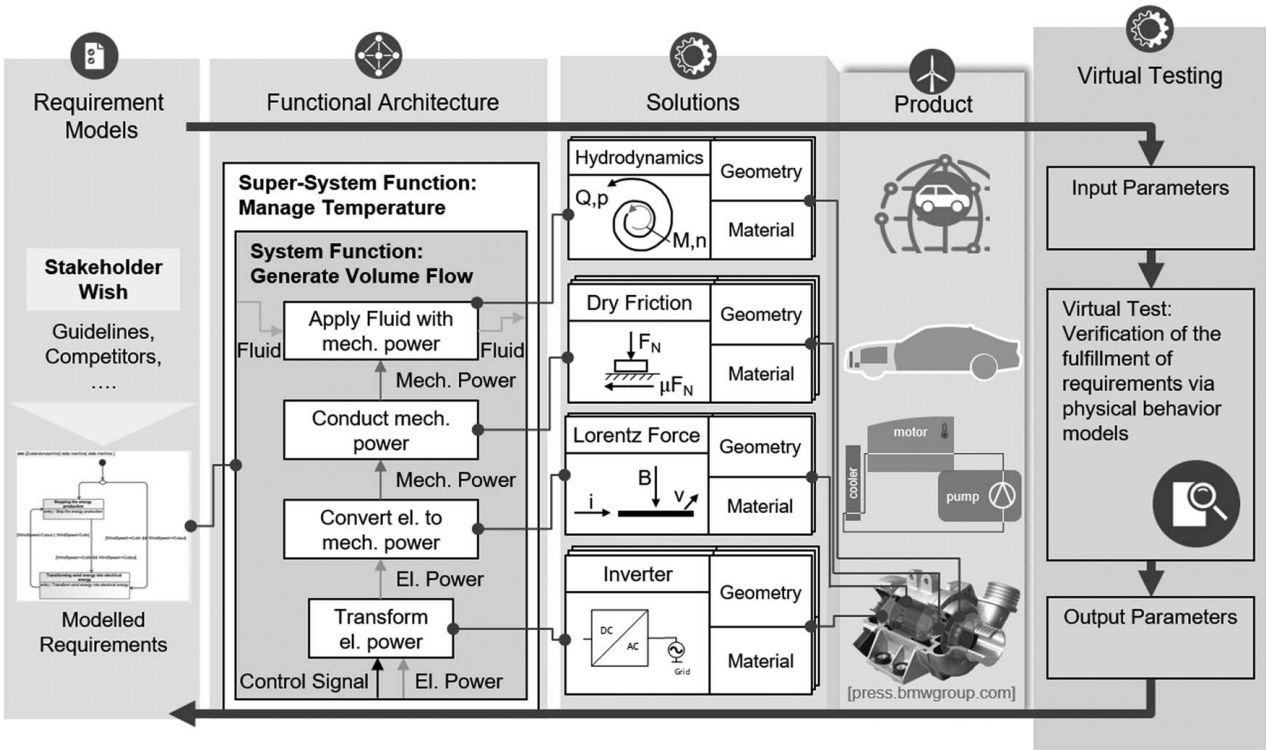

Fig. 3 Function-oriented development methodology

relating the input to the output. Therefore, and following [6], we define the two types of functions *elementary function* and *architecture*. An architecture may consist of further architectures and elementary functions and thereby its behavior is combined from the combination of the sub-functions behaviors. Elementary functions are not further decomposed and define the lowest level of the functional architecture. According to Ref. [6], the number of elementary functions for physical flows is finite and can therefore be catalogized. However, functions do not describe, how the intended behavior is realized in a system, but what behavior the system can provide. Therefore, by abstracting from details of the geometric shape of the components, the functions are specified independently of a specific solution and technical domain.

Principle Solutions. The functional architecture specifies a system in terms of interacting functions abstracting from possible realizations of the function in the technical product. The realization is domain-specific and described in the solution product's solution layer. In the solution layer, it is determined how an intended functional behavior is achieved. Mechanical solutions achieve an intended functional behavior by utilizing physical effects determined by physical laws. Physical effects determine how an incoming functional flow is converted into an outgoing functional flow. The physical effects act in a defined way between geometric surfaces, so called active surfaces. The combination of physical effect and active surfaces is the *principle solution* and describes how an elementary function is physically realized in a product [6,36]. In a principle solution, the physical effect is described by one or multiple interacting physical laws and is specified through mathematical equations. The variables in these equations often require parameters of active surfaces (geometric or material ones), quantifiers of the incoming or outgoing types of flows, or natural constants to be quantitatively determined [4,36]. These parameters given by the equations are the ones that should be focused on first in the design, as they are the key parameters for functional behavior of the system.

As described, the principle solutions of a system are structured according to the system's functional architecture, not according to the components in the product structure. However, it considers the active surfaces of the product. Principle solutions thereby provide a bridge between the function space and the final mechanical components. And, since the number of physical effects is finite [6], cataloging principle solutions can support accelerating concept development by reusing known principle solutions.

An example for a principle solution in the running example is the centrifugal pump impeller. The impeller realizes the elementary function apply fluid with mechanical energy. This function has a fluid and a rotational mechanical energy (i.e., torque and rotational velocity) as input and provides an accelerated fluid as output. In the centrifugal pump impeller, this is realized by using the physical effect of centrifugal forces (as listed in Ref. [6]). Other possibilities are, e.g., Bernoulli's principle or electro-kinetic effect. For the centrifugal force, different possibilities of the set of active surfaces include pumping impellers, whose inlets and outlets can be varied axially and tangentially. Exemplarily, we chose a setup, where a rotating pump wheel with paddles accelerates the fluid and leads it to a tangential outlet in a surrounding cylinder. The relevant geometric parameters are then according to the physical law the inner and outer diameters as well as the width of the wheel. The complete principle solution of the centrifugal pump wheel comprises the physical effect of centrifugal force and the two active surfaces pump impeller and outer cylinder. However, it does not determine mechanical components. The impeller surface has to be integrated into an impeller wheel, which might consist of further active surfaces as a tight fight that connects it with a shaft. The cylinder might be integrated into, for example, the pump housing as a component. Thus, the principle solution only determines the active surfaces, not the mechanical components.

Product. In the product layer, the final realization of the physical system is described, i.e., mechanical components and assemblies. Hence, the product layer does not provide a functional view but a component structured view on the system.

The link between the principle solutions and the product layer is the active surfaces. In the aforementioned layers, active surfaces are created to fulfill functions, and in the product layer, they are integrated into components, as a component consists of multiple active surfaces, which are connected by supporting structure. Thus, the main development task in the product layer is deciding which active surfaces to aggregate in which component and then to generate supporting structure that withstands the physical loads acting on the active surfaces. This process can either be done manually by a mechanical designer or automated, e.g., by generative design tools.

The aggregation of active surfaces to components underlies multiple restrictions, which are on the one hand functional ones, e.g., kinematics of the active surfaces. As mentioned before, principle solutions consist of two active surfaces, between which physical effects act. Typically, these active surfaces are integrated into different components, as often different kinematic states are required for them. For example, if two active surfaces need to rotate in opposite directions, they may not be integrated into the same component. On the other hand, design restrictions influence the aggregation, which may be for example production restrictions or building space of components.

As a result of separating principle solutions' active surfaces to components, there is a many to many-to-many relationship between components and principle solutions (and thereby functions). This many-to-many-relationship between component and function may result in a product structure that does not relate to the functional architecture.

The decision which active surfaces are aggregated in which mechanical component can provide new requirements to the system and thereby create new functions and principle solutions. For example, components that shall be rotating need to be supported, i.e., forces and moments need to be taken up. This creates a new functional requirement, a new function (support and lock degrees-of-freedom) and new principle solutions (bearings), which then create new components. Hence, the described process from requirements via function and solution to product is not a linear but an iterative one.

Considering the running example, examples for components aggregating active surfaces are the pump wheel and the pump housing. The pump wheel integrates the active surface impeller wheel from the aforementioned principle solution centrifugal pump wheel. Furthermore, it may integrate a shaft-hub-connection, so that it can be linked to a shaft. The pump housing integrates the tangential outlet of the pump, as well as, e.g., bearing seats. As the pump wheel rotates in the housing, the two active surfaces impeller wheel and tangential outlet cannot be realized in the same component, which would hinder relative movement.

Physical Behavior Models for Testing. Principle solutions describe how a function is realized within a system physically. To validate this physical behavior, physical behavior models are required to test the behavior of a principle solution against the requirements. Such models are already strongly used in mechanical engineering. As these models are physically oriented (and thereby function-oriented), they can be assigned to principle solutions already and thereby be used for early stage functional testing. For virtual verification, often different models are combined to workflows. In such workflows, outputs and inputs of different models are linked with each other. Thereby, complex requirements can be verified by estimating behavior.

Modeling Methodology. In the previous subsection, we explained the general design methodology. For applying this methodology and create a digital, model-based design method, we use

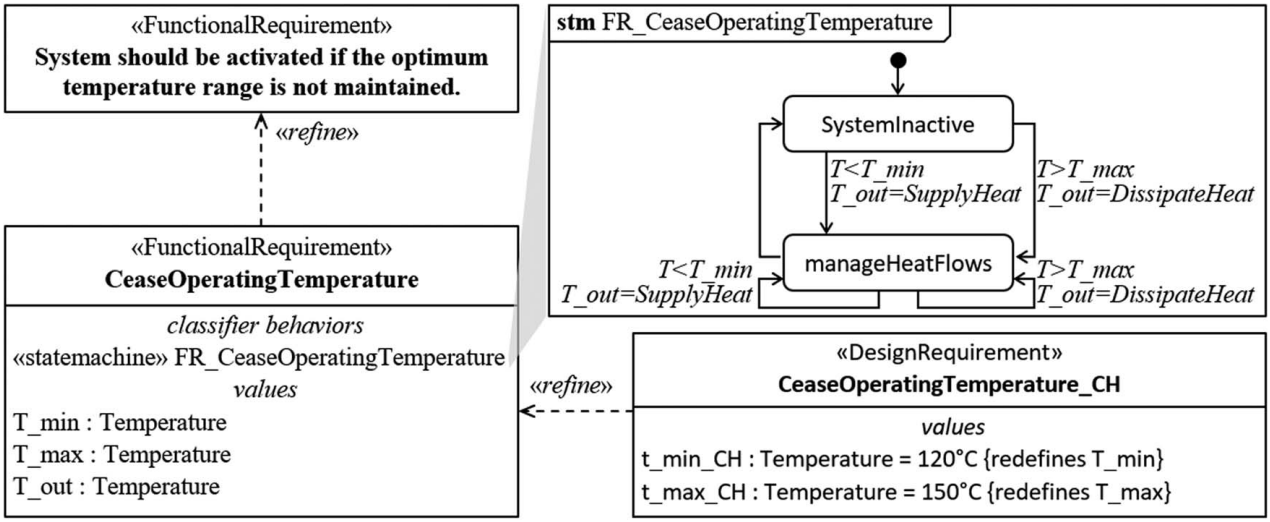

Fig. 4 Model of the functional requirement *CeaseOperatingTemperature*

SysML to model the described elements of systems, and thereby provide a seamless workflow of a physical system from requirements to product. This modeling is described in the following subsection.

Requirements. To ensure requirement satisfaction on all system levels, the requirements need to be stated such that the models representing the state of development can be verified and validated repeatable with respect to these requirements.

Therefore, in our approach requirements are structured as described before and modeled in a predefined modeling language suitable for the respective type of requirement.

As mentioned before, we differentiate between *functional requirements* and *design requirements*. The former express expectations about the desired behavior of the system. Suitable modeling languages for behavioral are, e.g., SysML state machines or activity diagrams [9,39].

Design requirements, on the other hand, express constraints as structural or geometric expectations, e.g., on the value ranges of system parameters. Expressions, such as mathematical (in-)equations or logical formulae seem suitable to model these requirements.

In the running example, a SysML-Block with a respective stereotype, value properties, and a classifier behavior defined through a state machine defines the fundamental functional requirement on the behavior of the cooling system, which is shown in Fig. 4. The cooling system alternates between the two states active and inactive. As long as the engine's temperature is within an optimal range defined through the parameters *T*_max and *T*_min, the cooling system remains inactive. Once the temperature rises above or below the operating temperature thresholds, the system changes its state to active and adjusts the temperature back into the operating range, see Fig. 4.

Considering the running example, a design requirement is given by the expectation that throughout the system's lifecycle, the optimal range of the cylinder head's operating temperature is between 120 °C and 130 °C. This design requirement is defined by a block with the respective stereotype. It refines the functional requirement described above and provides specific values for *T*_min and *T*_max. The functional requirement given in Fig. 4, is therefore open for reuse in different engines. For reuse, developers just define another design requirement to refine the functional requirement and set the engine-specific numeric values of these parameters. Conversely, the definition of the design requirement can be reused in other functional requirements that reference the operating temperature of the cylinder by referencing the parameters. Since parameters are assigned a unique type, consistency, e.g., in units is ensured [36].

The proposed modeling methods of requirements enable reuse and link requirements among each other. Techniques to analyze state charts, and activity diagrams may be applied and allow formal verification and validation of elements linked to these requirements.

Functions. A function performs transformation of energy, material, and information.

For modeling functions, Ref. [4] offers blocks with respective stereotypes. The functional constituents of an architecture, are specified through SysML's composition association. The interaction of sub-functions is defined through connectors that link the interfaces of these functions in an internal block diagram (IBD) of the architecture. Herein, typed flows model energy, material, or information in terms of physical quantities. This notion is compatible with [40] which equips the functional models in our approach with formal semantics that allows analyzing these models regarding consistency with the requirements. SysML's typing mechanism assures that connectors exist only between ports of equal types. Using SysML relations, functions can be linked to the requirements they are supposed to fulfill. These links enable tracing requirements to functions and, as proposed in Refs. [8,40], enable function verification with respect to the requirements. The link to the functional requirement is provided by a *satisfy* relationship in SysML.

Figure 5 shows the functional architecture of the running example: The architecture *ManageHeatFlows* has an interface that comprises incoming flows of energy of types *ThermalEnergy* and *ElectricalEnergy*, and an outgoing flow of energy also of type *ThermalEnergy*. These are modeled by typed *ProxyPorts*. The functional behavior of *ManageHeatFlows* is given through the composition of four functions, i.e., the three elementary functions *ControlHeatFlows*, *DistributeHeatFlows*, *SeparateFluidAndThermalEnergy*, and an architecture *GenerateVolumeFlow*.

Therein, *ControlHeatFlows* monitors the temperatures of the engine and calculates a control value that influences the volume flow of the coolant exiting the function *GenerateVolumeFlow*. In *DistributeHeatFlows,* the coolant absorbs thermal energy from the combustion process. *SeparateFluidAndThermalEnergy* releases heat from the coolant.

Principle Solutions. The functional architecture specifies a system in terms of interacting functions abstracting from possible realizations. Mechanical realizations of these functions achieve an intended behavior, determined by physical laws, that realize the specified functionality by acting in a defined way between geometric surfaces.

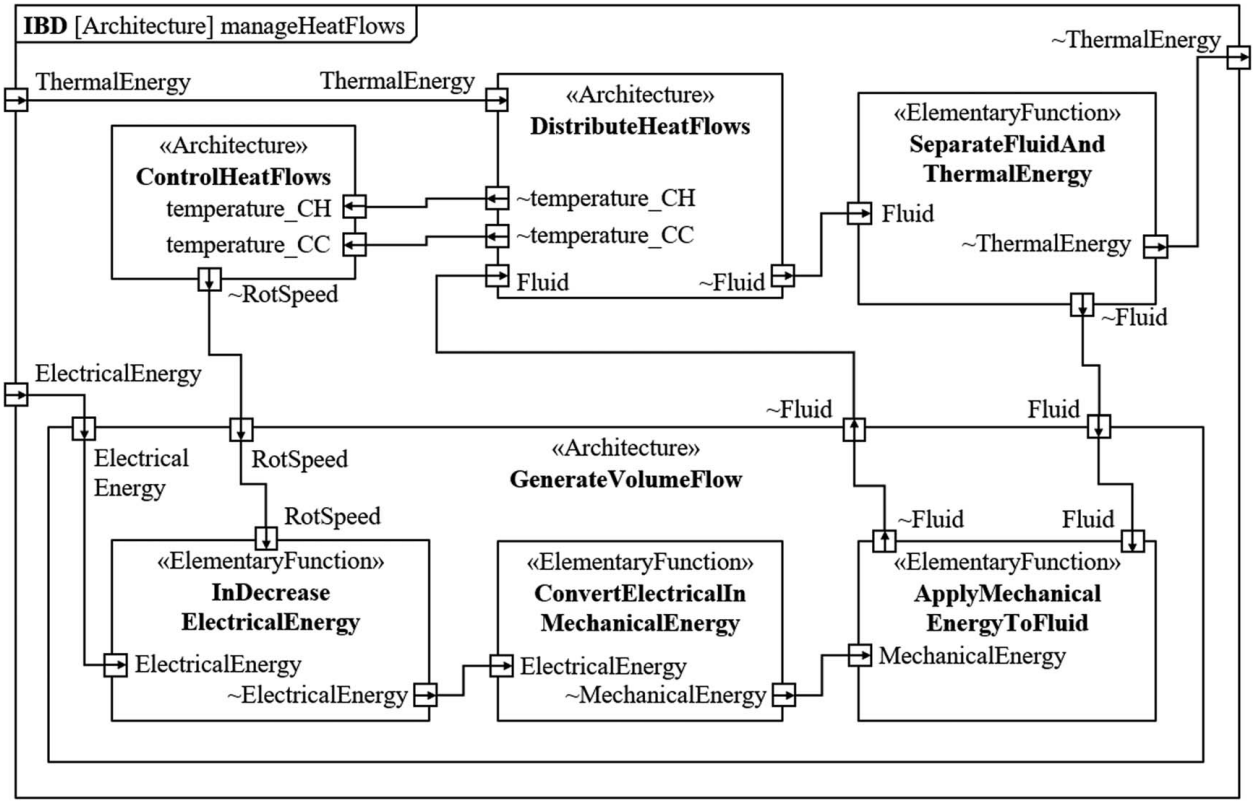

Fig. 5 Functional architecture of the example system

Suitable to link models within our approach seamlessly is the object-oriented principle of inheritance already provided by SysML: principle solutions specialize elementary functions and therefore inherit all the properties specified in the elementary function, i.e., the interface and the behavior [4]. As a specialization, the principle solution *adds* to the functional specification a definition of active surfaces and the equations defining a physical effect. This narrows the solution space, as the addition imposes further technical constraints on valid implementations. Definitions of physical effects, sets of active surfaces, and materials may be reused from model libraries.

Principle solutions and their constituents, active surfaces, physical effects, and material are modeled through blocks with the respective stereotypes. System parameters and natural constants are typed value properties. Since value properties are typed, searching for sets of active surfaces that are compatible with the equations that define a physical effect can be automated [4]. As all elements within principle solutions are typed, the principle solution is modeled modularly. This allows the creation of libraries for, e.g., active surfaces, physical effects, and full principle solutions, in which the typed elements can be stored and provided for reuse in other systems.

In the running example, one elementary function in the architecture *GenerateVolumeFlow* is *ApplyMechanicalEnergyToFluid*. A conceptual realization of this elementary function (i.e., how mechanical energy is applied to a fluid) is defined in the principle solution *CentrifugalPumpWheel*, cf. Fig. 6.

Among others, Ref. [6] lists the physical effect *CentrifugalForce* to realize this function. The modularity of the architecture facilitates exploring suitable technical concepts by exchange, provided a library of models of these effects. *PumpWheel* and *Cylinder* specify active surfaces through geometric parameters (e.g., the width of the pump wheel) and design parameters (e.g., the optimum volume flow). To verify the concept with respect to the functional requirements, materialistic properties apart from amongst others the density of the fluid are not decisive, and therefore not considered in this model.

The modeling technique presented so far allows specifying a system seamlessly from requirements, over functions to principle solutions. The formal grounding of the models and their relations provides the prerequisites for the formal and objective verification of concepts with respect to the requirements at any time during the development. The following section details how active surfaces are integrated into the product to extend the seamless approach to include models of the physical product.

Product. As described before, mechanical components integrate active surfaces from principle solutions. We use the SysML composition to model the relation between active surfaces and components. Thereby, the sets of active surfaces from different principle solutions are reorganized. One component may integrate multiple active surfaces from different principle solutions, and the active surfaces from one principle solution may be integrated into different components, cf. Fig. 7.

Existing tools such as the Cameo Systems Modeler allow linking parameters from external models with SysML Blocks. We use this infrastructure and model the active surfaces as shell surfaces in computer-aided design (CAD) that refine the parametric description in SysML via the interface described in Ref. [37]. Hence, integrating active surfaces to components as described above, initially yields a CAD component model consisting of unconnected shell surfaces. Shell surfaces are then transformed into full volume CAD components by designing the supporting structure between the active surfaces, either manually or automatically, e.g., using generative engineering.

Physical Behavior Models for Testing. The principle solution as proposed above provides a simple functional description of a mechanical solution. However, it is still to demonstrate that all requirements of a principle solution are satisfied, not only the functional requirements but also the design requirements, e.g., lifetime. For testing a solution against design requirements, other physical models than just the physical effect equations are required. In our

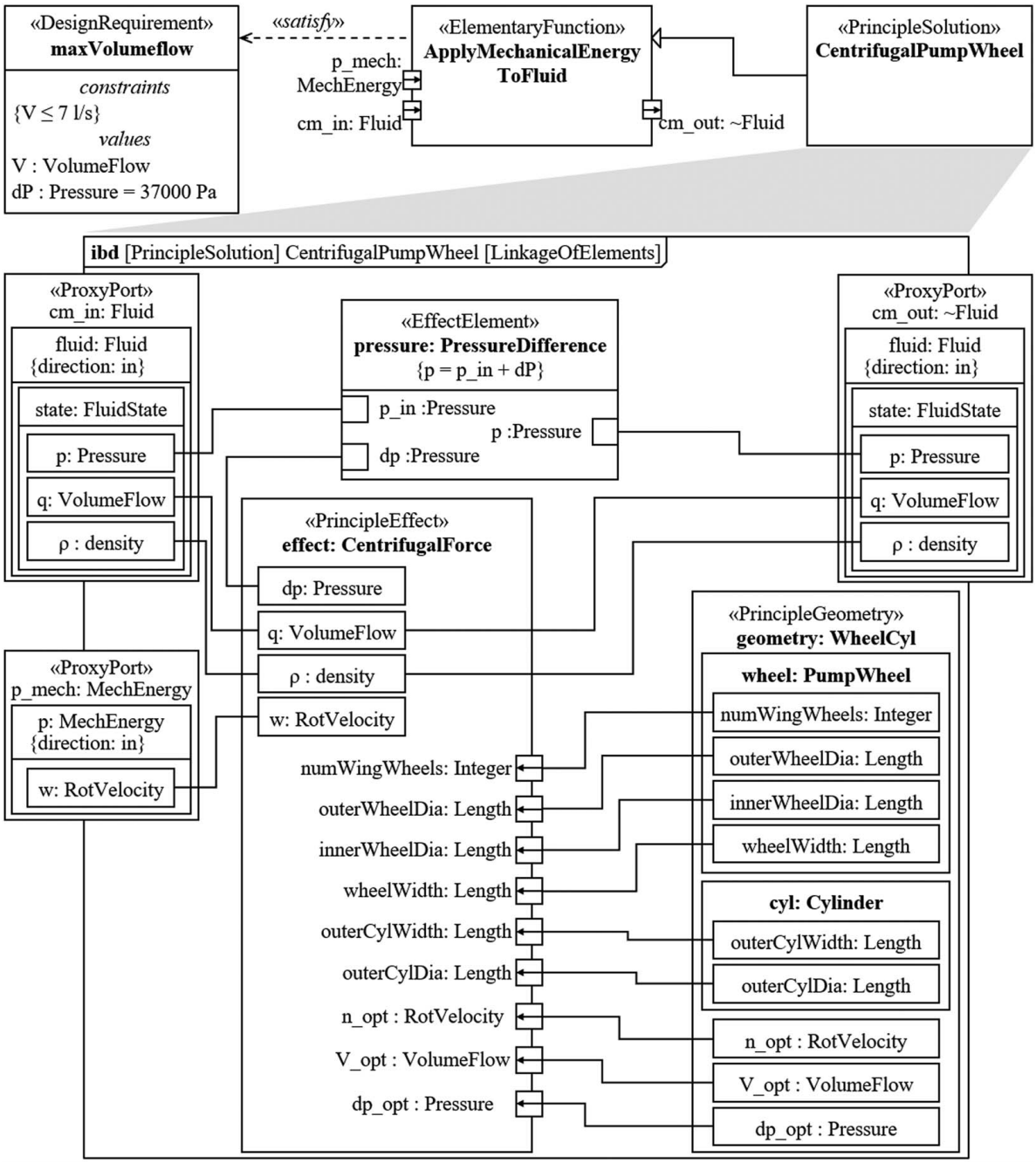

Fig. 6 Requirements are satisfied by elementary functions and their principle solutions whose elements, parameters, and relationships are visible in the internal block diagram

running example, the equation given in the physical effect of *CentrifugalPumpWheel* cannot estimate the lifetime, as the principle solution describes the functional relationship between input and output. Mechanical engineering has already developed powerful simulation methods based on physical behavior models and guidelines for testing mechanical solutions against design requirements [34,35]. These are usually created for one purpose, e.g., lifetime validation. Therefore, simulation models of multiple purposes may exist for a solution [41].

Physical behavior models are used for testing, not for providing a functional description. The models are therefore not directly integrated into the functional architecture. Instead, in our approach, sequential testing workflows are defined and linked to the solutions via trace links [37]. In a testing workflow, an input is converted to an output by one or multiple models estimating the physical behavior of the function under test. The output is then compared to numerical values in the design requirement. We employ SysML activity diagrams to model the test procedures, where the control flow describes the execution order of simulations, and the object flow describes the passing of parameter values between the simulations [37].

For providing executable tests, the physical behavior models, e.g., simulation models need to be integrated into the activity diagrams. Physical behavior models are usually modeled in domain-specific tools. Here, a tool-specific interface is required between the SysML editor and the domain-specific tool. Some SysML editors provide an execution engine, enabling the integration, and execution of models in external tools such as MATLAB. In this case, an activity contains an executable link to a physical behavior model. The activity has predefined and typed inputs and outputs. The object flow between the activities represents their data dependencies and automatic execution of the activity diagram propagates the values consistently between them.

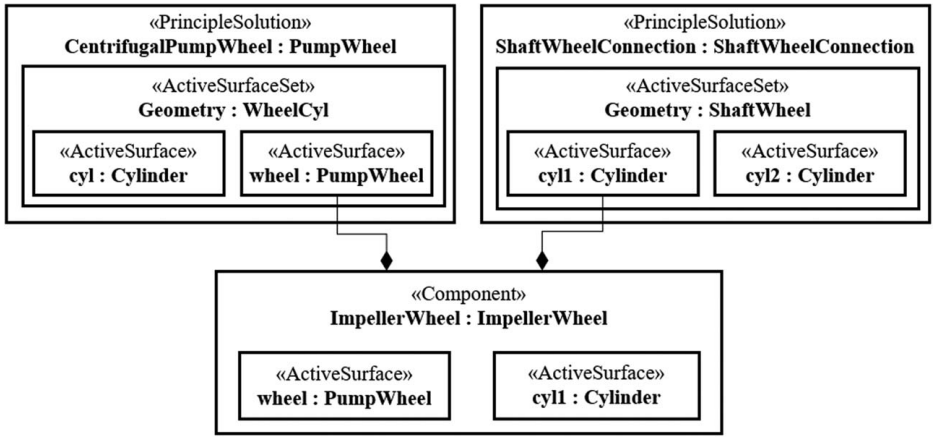

Fig. 7 Concept of mapping active surfaces to components

Using this modeling technique, models of principle solutions may be enhanced by predefined test activities that represent the norms or guidelines for validating design requirements. Added to a library they are open for reuse and provide automatic and repeated test execution at any time.

During development, physical behavior models are not only used for testing but also for design. Design guidelines help engineers to find a suitable set of parameters for a principle solution. From these guidelines, engineers can infer the typed value properties of a principle solution by giving input and required output values of a principle solution. For example, the value properties of the active surfaces from Fig. 6 as *innerWheelDiameter* and *outerWheelDiameter* are estimated by giving rotational speed, fluid pressure, and volume flow of the *CentrifugalPumpWheel*. The guideline provides values for the geometric attributes which we model as default values of the value properties. Based on these values, the geometric size of the active surfaces can be estimated by an activity that implements the procedure prescribed in the guideline. For standardized principle solutions such as centrifugal pumps or gear stages, these guidelines help towards finding suitable geometric parameter values. Design guidelines can also be modeled as activity diagrams and are in general similar to test activities. However, the result of such guidelines is not a parameter that is compared to a requirement, but geometric and physical parameter values of the principle solution [37].

The combination of tests and design guidelines allows for (semi-) automated development of solutions. A change in requirements triggers the execution of all test activities. In case of failed tests, design activities may be executed and the system can be adapted to the new conditions by estimating new sets of parameter values.

In this section, we presented an approach to describe the mechanical functions of a CPS from requirements via functions and principle solutions to components including defined tests for solutions by integrating physical behavior models. In the next section, we discuss the approach and its potentials.

Discussion & Evaluation

This section discusses the contribution of the presented approach (Sec. 4) regarding overcoming the identified challenges (Sec. 2).

Challenge 1: Function-driven innovation

The functional specification paradigm of our approach establishes consistency among requirements and physical components by modeling a functional architecture and principle solutions, overcoming geometry orientation.

For this purpose, the approach first differentiates between design requirements and functional requirements which facilitates linking them to further models of the system. Functional requirements are translated into a specification of the functional behavior of the system under development. Therefore, in the sense of Ref. [6], the function complied by the future system is defined by a functional architecture consisting of compositions of (elementary) functions wherein each function transforms flows of energy, material, and information. These functions interact by exchanging flows. Through links, the artifacts describing the functions are connected to the functional requirements they are supposed to fulfill. The architecture is modular in a mechanical [6] as well as a software [8] sense that a solution for a composed function is given by the composition of solutions to its constituents. This structures the upcoming development activities. Principle solutions specialize elementary functions by adding physical effects, active surfaces, and material properties realizing the specified functionality. Physical effects, active surfaces, and material properties are described by equations or parameters. Principle solutions do not refer to entire components, but to specific active surfaces that are related to functions and are initially independent of components. Components aggregate these active surfaces. The functional reference is consistently maintained, starting from functional requirements, via the functional architecture and principle solutions to the active surface of the component.

Thus, with our approach, innovative, cross-domain functions can be developed independent of component centricity and boundaries.

Challenge 2: Missing holistic workflow

Our approach provides a formally grounded modeling technique to specify and link the development artifacts of the mechanical domain seamlessly wherein many of the required tasks during development can be automated.

For this purpose, the requirements that are often textual in mechanical engineering are modeled using an explicit modeling language. Dependencies and refinements of requirements are modeled explicitly, which facilitates reusing requirement sets. These requirements are translated into a functional architecture which describes the system's functionality through the composition of (elementary) functions. These functions are linked by functional flows explicating interdependencies of functions. Functions that fulfill functional requirements are linked to these via satisfy relationships so that a continuous development process is ensured.

Principle solutions are deduced from elementary functions by generalization so that the functional interface and also the dependencies between the functions are inherited by the architecture of the principle solutions. The principle solution consists, among other things, of active surfaces, which are subsequently aggregated into components. Thus, the modeling technique of our approach makes it possible to develop requirements, functions, principle solutions, and components in a continuous and parameter-based manner. This provides transparency when analyzing the finally realized

component and automatically provides information about a component's important surfaces, acting physics and interactions, realized functions, and attached requirements. This transparency accelerates product development, as the required artifacts are always visible and accessible. It also allows for automated analyses of products.

Information or data obtained during other life phases such as production, use or recycling may be integrated for continuous optimization during the development time. For this purpose, their demands are explicated, e.g., as requirements, and added to the system model that describes the system under development as part of its continuous evolution. This allows optimizing components, e.g., for manufacturability and recycling, as well as dimensioning with regard to field data.

Challenge 3: Accelerated development

Through formal modeling, our approach facilitates reuse of models and automated, continuous validation.

Explicating and linking requirements forms the basis for verifying and validating functions, principle solutions, and components in the development. In the case of adaptations, the formally modeled requirements, functions, and principle solutions can be reused with high transparency of dependencies, which saves time and resources. For the functional architecture, our approach uses elementary functions [6], which exist as a finite set and can be provided via a library for the system under development. This also applies to principle solutions: Active surfaces and the finite set of physical effects can be reused both individually and in combination. Since [6] covers all possible physical effects for realizing an elementary function, possible principle solutions can be explored to identify optimal concepts to realize a specified functionality. As principle solutions are explicitly modeled in terms of parameters, they can be tested virtually against the requirements continuously from early development stages. By aggregating components from the active surfaces and parameters of the pre-verified principle solutions, components can be further designed in earlier stages. Various computer-aided engineering (CAE) models allow designing or testing the components, which use the component parameters or estimate them appropriately. Due to the described physical constraints, the training effort for using the method is seen as a lower problem than in other modeling approaches, where reuse has not shown the expected benefits. A benefit of our approach is that the reuse of models follows a physical logic, which supports engineers in their way of thinking and thereby may reduce training efforts.

Through the functional specification paradigm, our approach offers continuous validation throughout the entire development process by means of automation on the level of the full system, not only sub systems. This paves the way for (semi-)automated validation and verification, reducing the need for physical prototypes. Since information or data of other life phases such as production and use were also considered at an early stage through explicit modeling, we expect a smoother start of production with higher component quality.

Outlook: Roadmap Towards a Holistic and Functional Model-Based Design Method for Mechatronic Cyber-Physical Systems

The proposed approach provides a function-oriented design method for CPS including mechanical systems. However, we identified further open points to provide an industrialized, holistic approach. In this section, we give a roadmap towards such an approach from a research perspective.

(1) Scaling for broad industrial application.

For broad industrial usage and truly accelerated product development, the proposed methods have to be put into easy-to-use tools and tailored languages represented in an easy-to-understand way. Therefore, a strong interconnection between geometry-oriented CAD tools and function-oriented systems modelers is required. Also, the interconnectivity of simulation models in the system context has to be simplified. To link such models and provide efficient virtual testing for the system a parameter-based integration of simulation models is required, where simulation models can be easily linked to functional models via their parameters.

A further important point for industrial application is thorough training of people. Engineers need to consequently think in systems and functions in order to develop systems in a holistic manner.

(2) Machine-readable requirements.

We have discussed that formal requirements modeling is one central element of our approach. Requirements contain specific knowledge and either arise in a specific project or are established in a fundamental way, e.g., to represent the constraints of production facilities. Therefore, companies usually save all requirements and try to reuse them in subsequent projects, thus, taking relevant knowledge into account from the very beginning. With the rising number of requirements and the higher complexity of technical systems, human-based requirements management is reaching its limits. Novel requirement management tools already employ artificial intelligence such as, e.g., natural language processing for analyzing, structuring, and checking consistency of informal requirements [42,43]. Combined with formal modeling techniques for deriving and specifying system functions from these requirements, this overcomes the identified possible missing links between development artifacts for requirements. In this way, engineering knowledge can be stored in requirements in the long term and used efficiently.

(3) A function-oriented design method based on active surfaces.

As discussed, for function-oriented design, the component is not the smallest entity, but the active surfaces are. Today's design methods lack a methodology to design components with a focus on their active surfaces. Such a method has to support in identifying required surfaces based on chosen principle solutions. These surfaces may then be assigned to mechanical components. The components' supporting structure may then be designed automatically, e.g., by generative design, when active surfaces are given. Considering the active surface as the center of geometric design can anchor function-oriented thinking also for mechanical engineers.

(4) Digital libraries for function and solution models.

A modeling approach such as the one presented allows for modeling a system from a function-oriented view, that abstracts from the physical solutions offered in existing libraries such as the MODELICA standard library [44]. The system may be fully specified including all functions and their interactions using the modeling language given in Ref. [4]. However, especially for mechanical functions, the amount of undefined interactions is large. These interactions are predefined by solution-dependent physical restrictions. To handle the complexity of interactions, predefined function, and solution libraries can support engineering. A transformation of [29] may provide digital functions and solutions, which can be integrated into functional architectures. Functional flows, that are through physical restrictions linked to the function, may be included. Such a digital catalog may also include engineering models, linked to the respective solution. A strong concept for reuse of functions and solutions enables efficient modeling, evolving, and maintaining a system's functional architecture throughout the entire lifecycle and helps improving quality. Function and solution libraries help engineers to save knowledge, accelerate development, and increase product quality.

(5) Systematic integration of engineering models and tests.

Fully virtual system development requires virtual design, and virtual testing of systems. Especially for the later steps,

domain-specific engineering models are used. On the one hand, the functional architecture provides a formal specification of the system under development that enables techniques such as model checking for the formal verification of requirements. On the other hand, the functional architecture serves as an organizing element to trace domain-specific models that describe details of physical implementations at lower abstraction levels. This requires a systematic method to integrate engineering models and predefined tests. On the language side, this requires systematics on how to integrate an external domain model in a reusable way into the system model. Also, methods of test case generations may be evaluated in this context. For a complete integration, interfaces and documentation of the models have to be developed and categorization mechanisms to be defined. Then, changes in the system model can be propagated to domain models (semi-) automatically. Virtual testing is hastened, as models and tests are directly available in the system model and do not need to be defined manually per tool and test.

Validating products holistically requires models from all stages of the product life cycle, as each step provides relevant information for testing requirement satisfaction of the product. Our approach gives a framework to integrate all such models and link them to the solution as all kinds of models may be integrated. In addition, physical hardware tests have to be integrated into the approach, to deal with effects that are not predictable or formalized yet.

(6) Refining architectures to digital twins.

We consider a digital twin to be "a set of models of the system, a set of digital shadows and their aggregation and abstraction collected from a system, and a set of services that allow using the data and models purposefully with respect to the original system" [45]. The digital twin of a system, therefore, comprises all required kinds of models to mirror a system during life [46]. Digital twins are used diversely from design-space exploration [47] to the monitoring of systems [48] and optimizing [49] system behavior.

The models in the digital twin are based on behavior describing models from product development. However, they are often simplified to provide information in a shorter calculation time. The functional architecture can yield a framework for the digital twin models. It provides the system's architecture and can help to identify the models for specific digital twin purposes. It is to be investigated if and how the digital twin can be directly derived from the system models including links to domain-specific models. Also including feedback (e.g., sensor data) from lifetime to such models within a system model may be investigated.

Deriving digital twins from functional models provides a strong link between product development, production, and use of systems, and thereby is one enabler of a holistic workflow through the product lifecycle.

Conclusion

In this paper, we presented a novel approach for a functional, model-based design method of mechatronic cyber-physical systems. Based on the derivation and discussion of current challenges in the industry regarding mechatronic CPS development as well as the discussion of a comprehensive literature survey of the current state of research, a novel development approach was presented that closes the gap between formal, functional modeling, and mechanical development. It thereby enables the formal specification and virtual testing of systems including mechanical functions.

The foundation of the approach is to develop mechanical functions formally. The approach introduces a link between requirements and products via functions and principle solutions and thereby formalizes the studies of Koller [6,29]. It provides a mathematical, generally understandable description of mechanical solutions. Furthermore, it integrates engineering models and is, to the far of our knowledge, the first approach to provide functional development and testing for mechanical functions at every stage of development.

Thus, our approach enables a full specification of CPS also including the mechanical domain on all stages of the product development in a formal way. It thereby enables a fully virtual, model-based methodology covering the whole development process for the development of CPS and linking all data objects seamlessly. Additionally, the method allows for a complete virtual testing of CPS including all domains and interdependencies by organizing models and linking them in a function-oriented way.

We demonstrated the approach using the running example of an electric coolant pump of a combustion engine's coolant circuit. The results show how a seamless description of a mechatronic system as the coolant pump is realizable with our approach from requirements via system functions and solutions to product and how functional testing and automated design may be realized. Based on the results and the identified challenges in product development, a roadmap on future research topics has been defined to extend the approach towards a fully seamless product lifecycle.

For product development, this includes automated requirement modeling, reusable libraries for functions and solutions, automated verification and validation of functions and solutions by theorem proving and simulation models, and generation of components from their active surfaces.

As a link towards the further steps of the product lifecycle, the derivation of digital twins from functional models in product development is to be investigated. A broad use in the industry then requires the support of the method via tools allowing for scalability.

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The authors attest that all data for this study are included in the paper.

Nomenclature

CAD = computer-aided design
CAE = computer-aided engineering
CC = crankcase
CH = cylinder head
CPS = cyber-physical systems
FE = finite elements
IBD = internal block diagram
MBSE = model-based systems engineering
OEM = original equipment manufacturer
SE = systems engineering
SysML = systems modeling language

References

[1] Törngren, M., and Sellgren, U., 2018, "Complexity Challenges in Development of Cyber-Physical Systems," *Principles of Modeling*, M Lohstroh, P Derler, and M Sirjani, eds., Springer International Publishing, Cham, pp. 478–503.

[2] Olivain, N., Tiefenbacher, P., and Kohl, J., 2021, "Bayesian Structural Learning for an Improved Diagnosis of Cyber-Physical Systems," arXiv:2104.00987.

[3] SEBoK Editorial Board, 2021, *The Guide to the Systems Engineering Body of Knowledge (SEBoK), v. 2.5*, RJ Cloutier, ed., INCOSE Systems Engineering Research Center, Hoboken, NJ.

[4] Drave, I., Rumpe, B., Wortmann, A., Berroth, J., Hoepfner, G., Jacobs, G., Spuetz, K., Zerwas, T., Guist, C., and Kohl, J., 2020, "Modeling Mechanical Functional Architectures in SysML," Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, Virtual, Oct. 16–23, ACM, New York, NY, pp. 79–89.

[5] France, R., and Rumpe, B., 2007, "Model-Driven Development of Complex Software: A Research Roadmap," Future of Software Engineering (FOSE '07), Minneapolis, MN, May 23–25, IEEE, pp. 37–54.

[6] Koller, R., 1998, *Konstruktionslehre für den Maschinenbau: Grundlagen zur Neu- und Weiterentwicklung technischer Produkte mit Beispielen*, Springer Berlin, Berlin, Heidelberg.
[7] Pahl, G., Beitz, W., Feldhusen, J., and Grote, K.-H., 2007, *Engineering Design: A Systematic Approach*, 3rd ed, Springer, London.
[8] Broy, M., 2018, "On Architecture Specification," *SOFSEM 2018: Theory and Practice of Computer Science*, AM Tjoa, L Bellatreche, S Biffl, J van Leeuwen, and J Wiedermann, eds., Springer International Publishing, Cham, pp. 19–39.
[9] Markthaler, M., Kriebel, S., Salman, K. S., Greifenberg, T., Hillemacher, S., Rumpe, B., Schulze, C., Wortmann, A., Orth, P., and Richenhagen, J., 2018, "Improving Model-Based Testing in Automotive Software Engineering," 2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), Gothenburg, Sweden, May 30–June 1, pp. 172–180.
[10] Endres, A., and Rombach, D., 2003, *A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories*, Pearson Addison Wesley, Harlow.
[11] INCOSE, 2007, *INCOSE Systems Engineering Vision 2020*, INCOSE, San Diego, CA.
[12] Alur, R., 2015, *Principles of Cyber-Physical Systems*, The MIT Press, Cambridge, MA.
[13] Eigner, M., Gilz, T., and Zafirov, R., 2012, "Proposal for Functional Product Description as Part of a PLM Solution in Interdisciplinary Product Development," DS 70: Proceedings of DESIGN 2012, the 12th International Design Conference, Dubrovnik, Croatia, May 21–24.
[14] Wheatcraft, L. S., 2010, "Everything You Wanted to Know About Interfaces, But Were Afraid to Ask," Incose Int. Symp., **20**(1), pp. 1132–1149.
[15] Conway, M. E., 1968, "How Do Committees Invent?," Datamation Mag., **14**(4), pp. 28–31.
[16] Stahl, T., Völter, M., Bettin, J., Czarnecki, K., and Stockfleth, B. v., 2006, *Model-Driven Software Development: Technology, Engineering, Management*, Wiley, Chichester.
[17] Selic, B., 2003, "The Pragmatics of Model-Driven Development," IEEE Softw., **20**(5), pp. 19–25.
[18] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., 1994, *Design Patterns: Elements of Reusable Object-Oriented Software*, Pearson International, Reading, MA.
[19] Hölldobler, K., Michael, J., Ringert, J. O., Rumpe, B., and Wortmann, A., 2019, "Innovations in Model-Based Software And Systems Engineering," JOT, **18**(1), p. 1.
[20] Ptolemaeus, C., 2014, *System Design, Modeling, and Simulation Using Ptolemy II*, Ptolemy.org, Berkeley.
[21] Moeser, G., Albers, A., and Kumpel, S., 2015, "Usage of Free Sketches in MBSE Raising the Applicability of Model-Based Systems Engineering for Mechanical Engineers," 2015 IEEE International Symposium on Systems Engineering (ISSE), Rome, Italy, Sept. 28–30, IEEE, pp. 50–55.
[22] Broy, M., and Stølen, K., 2001, *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*, Springer, New York.
[23] Broy, M., 2012, "System Behaviour Models With Discrete and Dense Time," *Advances in Real-Time Systems*, S Chakraborty, and J Eberspächer, eds., Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 3–25.
[24] Drave, I., Hillemacher, S., Greifenberg, T., Kriebel, S., Kusmenko, E., Markthaler, M., Orth, P., et al., 2019, "SMArDT Modeling for Automotive Software Testing," Softw. Pract. Exper., **49**(2), pp. 301–328.
[25] Ebert, R., Jolianis, J., Kriebel, S., Markthaler, M., Pruenster, B., Rumpe, B., and Salman, K. S., 2019, "Applying Product Line Testing for the Electric Drive System," Proceedings of the 23rd International Systems and Software Product Line Conference—Volume A, ACM, New York, NY, pp. 14–24.
[26] Bayer, T., Day, J., Dodd, E., Jones-Wilson, L., Rivera, A., Shougarian, N., Susca, S., and Wagner, D., 2021, "Europa Clipper: MBSE Proving Ground," 2021 IEEE Aerospace Conference, Big Sky, MT, Mar. 6–13
[27] Dubos, G., Schreiner, S., Wagner D, A., Jones, G., Kerzhner, A., and Kaderka, J., 2016, "Architecture Modeling on the Europa Project," AIAA SPACE 2016, Long Beach, CA, Sept. 13–16.
[28] Chung, S. H., Bayer, T. J., Cole, B., Cooke, B., Dekens, F., Delp, C., and Lam, D., 2012, "Model-Based Systems Engineering Approach to Managing Mass Margin," 5th International Workshop on Systems & Concurrent Engineering for Space Applications, Lisbon, Portugal, Oct. 17–19.
[29] Koller, R., and Kastrup, N., 1998, *Prinziplösungen zur Konstruktion technischer Produkte*, 2nd ed., Springer Berlin Heidelberg, Berlin, Heidelberg.
[30] Gausemeier, J., Dorociak, R., Pook, S., Nyßen, A., and Terfloth, A., 2010, "Computer-Aided Cross-Domain Modeling of Mechatronic Systems," DS 60: Proceedings of DESIGN 2010, the 11th International Design Conference, Dubrovnik, Croatia, May 17–20, pp. 723–732.
[31] Moeser, G., Kramer, C., Grundel, M., Neubert, M., Kümpel, S., Scheithauer, A., Kleiner, S., and Albers, A., 2015, "Fortschrittsbericht zur modellbasierten Unterstützung der Konstrukteurstätigkeit durch FAS4M," *Tag des Systems Engineering*, S. O Schulze, and M. Christian, eds., Carl Hanser Verlag, Munich, pp. 69–78.
[32] Wölkl, S., and Shea, K., 2009, "A Computational Product Model for Conceptual Design Using SysML," ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, San Diego, CA, Aug. 30–Sept. 2, pp. 635–645.
[33] Berroth, J., Jacobs, G., Kroll, T., and Schelenz, R., 2016, "Investigation on Pitch System Loads by Means of an Integral Multi Body Simulation Approach," J. Phys.: Conf. Ser., **753**(11), p. 112002.
[34] Pasch, G., Jacobs, G., and Berroth, J., 2020, "NVH-Systemsimulation eines Traktors mit hydrostatisch-mechanischem Leistungsverzweigungsgetriebe," Landtechnik, **75**(4), pp. 301–315.
[35] Jaeger, M., Drichel, P., Schröder, M., Berroth, J., Jacobs, G., and Hameyer, K., 2020, "Die Kopplung elektrotechnischer und strukturdynamischer Domänen zu einem NVH-Systemmodell eines elektrischen Antriebsstrangs," Elektrotech. Inftech., **137**(4–5), pp. 258–265.
[36] Zerwas, T., Jacobs, G., Spütz, K., Hoepfner, G., Drave, I., Berroth, J., Guist, C., Konrad, C., Rumpe, B., and Kohl, J., 2021, "Mechanical Concept Development Using Principle Solution Models," IOP Conf. Ser.: Mater. Sci. Eng., **1097**(1), p. 12001.
[37] Hoepfner, G., Jacobs, G., Zerwas, T., Drave, I., Berroth, J., Guist, C., Rumpe, B., and Kohl, J., 2021, "Model-Based Design Workflows for Cyber-Physical Systems Applied to an Electric-Mechanical Coolant Pump," IOP Conf. Ser.: Mater. Sci. Eng., **1097**(1), p. 12004.
[38] Rumpe, B., 2016, *Modeling With UML*, Springer International Publishing, Cham.
[39] Glinz, M., 2002, "Statecharts For Requirements Specification—As Simple As Possible, As Rich As Needed," Proceedings of the ICSE 2002 International Workshop on Scenarios and State Machines: Models, Algorithms and Tools, Orlando, FL, May 20.
[40] Broy, M., 2010, *Cyber-Physical Systems*, Springer Berlin Heidelberg, Berlin/Heidelberg.
[41] Stachowiak, H., 1973, *Allgemeine Modelltheorie*, Springer, Wien.
[42] Perini, A., Susi, A., and Avesani, P., 2013, "A Machine Learning Approach to Software Requirements Prioritization," IEEE Trans. Softw. Eng., **39**(4), pp. 445–461.
[43] Juhnke, K., Nikic, A., and Tichy, M., 2021, "Clustering Natural Language Test Case Instructions as Input for Deriving Automotive Testing DSLs," J. Object Technol., **20**(3), p. 5:1.
[44] Modelica Association, 2020, *Modelica Standard Library—Version 4.0.0*, Modelica Association, Linköping, Sweden.
[45] Dalibor, M., Michael, J., Rumpe, B., Varga, S., and Wortmann, A., 2020, "Towards a Model-Driven Architecture for Interactive Digital Twin Cockpits," *Conceptual Modeling*, G Dobbie, U Frank, G Kappel, SW Liddle, and HC Mayr, eds., Springer International Publishing, Cham.
[46] Negri, E., Fumagalli, L., and Macchi, M., 2017, "A Review of the Roles of Digital Twin in CPS-Based Production Systems," Procedia Manuf., **11**, pp. 939–948.
[47] Allemang, R., Spottswood, M., and Eason, T., 2014, "A Principal Component Analysis (PCA) Decomposition Based Validation Metric for Use With Full Field Measurement Situations," *Model Validation and Uncertainty Quantification*, 3, HS Atamturktur, B Moaveni, C Papadimitriou, and T Schoenherr, eds., Springer International Publishing, Cham, pp. 249–263.
[48] Golafshan, R., Dascaliuc, C., Jacobs, G., Roth, D., Berroth, J., and Neumann, S., 2021, "Damage Diagnosis of Cardan Shafts in Mobile Mining Machines Using Vibration Analysis," IOP Conf. Ser.: Mater. Sci. Eng., **1097**(1), p. 12019.
[49] Bibow, P., Dalibor, M., Hopmann, C., Mainz, B., Rumpe, B., Schmalzing, D., Schmitz, M., and Wortmann, A., 2020, "Model-Driven Development of a Digital Twin for Injection Molding," *Advanced Information Systems Engineering*, S. Dustdar, E. Yu, C. Salinesi, D. Rieu, and V. Pant, eds., Springer International Publishing, Cham, pp. 85–100.