Modeling Mechanical Functional Architectures in SysML

Imke Drave
Bernhard Rumpe
Andreas Wortmann
Software Engineering
RWTH Aachen University
http://www.se-rwth.de

Joerg Berroth, Gregor Hoepfner, Georg Jacobs, Kathrin Spuetz, Thilo Zerwas
Institute for Machine Elements and Systems Engineering
RWTH Aachen University
http://imse.rwth-aachen.de

Christian Guist
Jens Kohl
BMW Group
http://www.bmw.de

ABSTRACT

Innovations in Cyber-Physical System (CPS) are driven by functionalities and features. Mechanical Engineering, on the other hand, is mainly concerned with the physical product architecture, *i.e.,* the hierarchical arrangement of physical components and assemblies that forms the product, which is not explicitly linked to these functions. A holistic model-driven engineering approach for CPS, therefore, needs to bridge the gap between functions and the physical product architecture to enable agile development driven by automation. In the theoretical field of mechanical design methodology, functional architectures describe the functionality of the system under development as a hierarchical structure. However, in practice, these are typically not considered let alone modeled. Existing approaches utilizing mechanical functional architectures, however, do not formalize the relation between the functional architecture and the geometric design. Therefore, we conceived a meta-model that defines modeling-languages for modeling functional architectures of mechanical systems and physical solutions, *i.e.,* interconnections of physical effects and geometries, as refinements of the functional components. We have encoded the meta-model as a SysML profile and applied it within an interdisciplinary, industrial project to model an automotive coolant pump. Our contribution signposts the potential of functional structures to not only bridge the gap between function and geometry in mechanics but also to integrate the heterogeneous domains participating in CPS engineering.

CCS CONCEPTS

• **Cyber-Physical Systems**; • **Functions in Mechanical Engineering**; • **Functional Architectures**; • **SysML-Profile**;

KEYWORDS

Systems Engineering, Cyber-Physical Systems, Functional Architecture, Mechanical Design Methodology SysML, SysML-Profile, Product Development Process

MODELS '20, October 18–23, 2020, Virtual Event, Canada

ACM ISBN 978-1-4503-7019-6/20/10…$15.00
https://doi.org/10.1145/3365438.3410938

ACM Reference Format:
Imke Drave, Bernhard Rumpe, Andreas Wortmann, Joerg Berroth, Gregor Hoepfner, Georg Jacobs, Kathrin Spuetz, Thilo Zerwas, Christian Guist, and Jens Kohl. 2020. Modeling Mechanical Functional Architectures in SysML. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3365438.3410938

1 INTRODUCTION

CPS are characterized by the interaction of mechanical, electronic, and software systems [1, 10, 42]. Social and technological challenges [7, 17] as well as the customer's demand for more functionalities and a shorter time-to-market, raise the complexity of engineering such systems [14, 29, 40].

In Mechanical Engineering (ME), engineers integrate the physical components of a product, into assemblies such that geometric constraints, *e.g.,* regarding design space, mounting, and maintenance are satisfied [40]. Further design requirements, such as lifetime requirements, or energy efficiency, contribute to the complexity of mechanical systems. We refer to the hierarchical arrangement of physical components and assemblies that forms the product as the *(physical) product architecture*. Mechanical systems fulfill certain functionalities through physical effects acting between components and assemblies of the physical product. Thus, there is a strong correlation between physical effects and the product architecture. Engineers control the impact of physical effects by manipulating the geometric shape of components or their material, as the effects themselves are set by laws of nature [40]. As a result, mechanical engineers tend to directly design the geometry of components based on given requirements, without explicating the functionality to implement. Therefore, the geometric and physical integration of components into mechanical products has been optimized and the physical product architecture has become the element that structures the development activities in ME [40].

This raises a gap between the functional CPS requirements [14, 29] and the physical product architecture as the physical components are not directly linked to the functions or features they implement. Therefore, reusing existing implementations in other systems is hardly possible and functional testing occurs late in the Product Development Process (PDP), *i.e.,* when changes are cost-intensive.

In Software Engineering (SE), the problem-implementation gap arises whenever the solution to a problem is described at a lower level of abstraction than the problem itself [17]. Model-Driven Engineering (MDE) aims to enable developers to focus on their respective domains by abstracting from the complexities of the

implementation platform [17]. The application of abstraction, separation of concerns, and architectural modeling [8] have shown to reduce this gap effectively [8, 17]. A software architecture describes the functions of the software system under development as a hierarchy of interacting, *i.e.,* message-exchanging, functional components [9]. Among others, formal modeling based on domain-specific adjustable Architecture Description Languages [13, 35, 36] enables verification at early design stages, reuse, and supports automation to enhance agile development in the software domain.

The problem-implementation gap present in automotive software engineering [14, 17, 29] resembles the gap between functional requirements and the product architecture in ME: Functional requirements imposed upon the product are stated by the customer, typically in natural language at a high level of abstraction [29]. Geometric models, such as *e.g.,* Computer-Aided Design (CAD) models, describe the assemblies and components at a level of detail that reaches from the engine as a whole to the screws holding it together. Thus, the product architecture which is part of the mechanical solution domain is described at a lower level of abstraction (the level of screws) than the functional requirements which belong to the mechanical problem domain.

Mechanical design theory considers functional structures, *i.e.,* a hierarchical decomposition of the required system functionality, as a means to systematize the design process in ME [28, 40]. Once formalized as models, functional structures have the potential to not only narrow the gap between functional requirements and the physical product architecture, but also to bridge the gap between the latter and the software architecture. Hence, the contributions of this paper are

(1) a meta-model for functional architectures of CPS from an ME point of view based on [28, 40];
(2) a demonstration of how to encode the meta-model as a SysML profile, that enables mechanical engineers to model these architectures in a way that fosters reuse and early exploration of innovative solutions; and
(3) an example from industry modeling an automotive coolant pump using the profile which emerged as part of an interdisciplinary project to demonstrate the profile's usage and possible benefits.

The contributions aim to signpost the potential of functional architectures not only to systematize the PDP through reuse and automation enabled by formal models but also to enhance collaboration of experts from heterogeneous domains in a holistic CPS engineering approach.

The rest of this paper is structured as follows: Section 2 introduces a running example. Section 3 provides preliminaries regarding the SysML elements extended or reused in in the proposed SysML profile. Section 4 gives insight into functional architectures in mechanical design theory from a language engineering point of view and constitutes the concepts in a meta-model. Section 5 encodes this meta-model as a SysML profile. Section 6 illustrates the results from using the profile for engineering an automotive electrical coolant pump within our project. Section 7 discusses related work and the findings before Section 8 concludes.

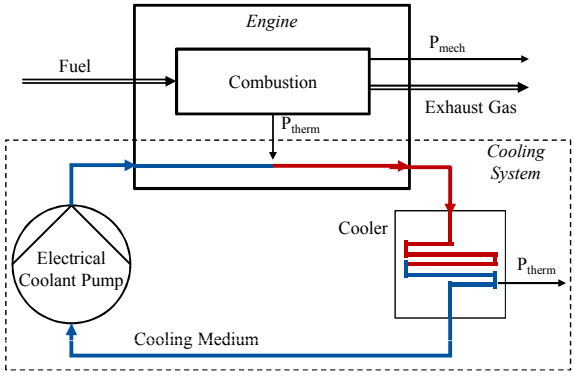

Figure 1: Diagrammatic illustration of an automotive combustion engine with a cooling system.

2 RUNNING EXAMPLE

To illustrate domain and language concepts, we use the following running example from automotive engineering throughout the paper: For propulsion, automotive systems contain a drive system. The main functionality of the drive system is to convert input energy into mechanical energy and to transfer the mechanical energy onto the road, where it causes the vehicle to move. Speaking in terms of components, the engine performs the former, while the drive train and the wheels perform the latter task. Combustion drives, for instance, convert the chemical energy held by fuel into mechanical energy. Electric drives, on the other hand, convert electrical to mechanical energy.

Figure 1 shows the principle set up of a combustion engine diagrammatically. The physical effect that makes combustion engines serve their purpose is the combustion of the fuel that is injected into the engine's cylinders. The combustion converts a portion of the chemical energy held by the fuel into thermal energy which causes the pressure in the combustion chamber to increase. The released exhaust gas holds the rest of the chemical energy. The increasing pressure acts on the surface of the engine's piston as mechanical energy (P_{mech}) causing the piston to move. However, the thermal energy (P_{therm}) is released as heat which causes the engine's temperature to increase. Once a maximum temperature is reached, the engine overheats and stops functioning. To prevent the engine from overheating, it has to be cooled, *i.e.,* the thermal energy released as heat has to be dissipated. Often, water cooling systems, as sketched at the bottom of Figure 1, take on this task. Driven by an electric motor, a cooling medium circulates between the combustion engine and a cooler. By the law of convection [50], the circulating cooling medium absorbs the combustion heat at the engine. The cooler releases the heat absorbed by the cooling medium to the surrounding air. Keeping the cooling medium circulating is required for absorbing the heat, as otherwise convection would not take place. Thus, a cooling medium pump is a necessary component of the cooling system.

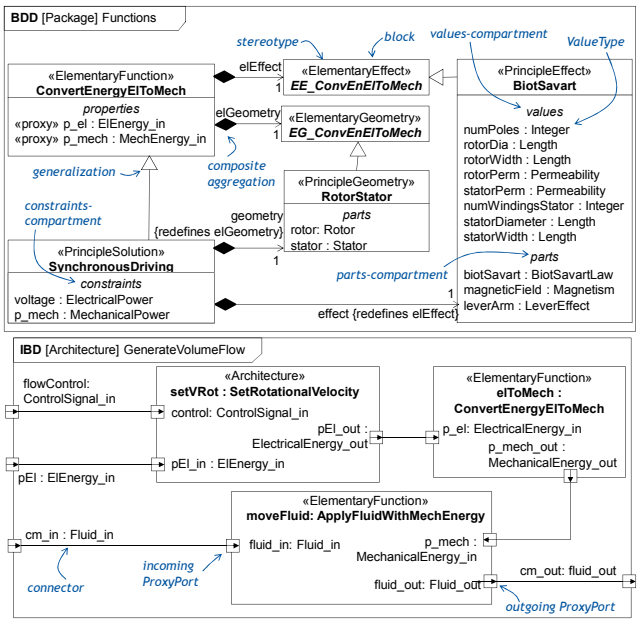

Figure 2: Top: SysML Block Definition Diagram (BDD) of the functional architecture of the running example. Bottom: Internal Block Diagram (IBD) of `GenerateVolumeFlow`. For details on stereotypes and contents see Sections 4 to 6.

3 PRELIMINARIES

The SysML profile introduced in Section 5.2 is tailored for ME and provides a language for explicating the functional structure of a mechanical system, which is understood as a reusable basis of the PDP in [40], in a model. This section briefly summarizes the SysML elements that are extended or reused in the profile.

SysML is a general-purpose modeling language family for systems engineering [33] that reuses and extends a subset of the Unified Modeling Language (UML) 2.5 [32] to represent aspects of system software and hardware in an integrated way [25]. To this end, it comprises four behavior modeling languages, four structure modeling languages, and requirement diagrams. The structure modeling languages comprise BDDs that describe the structure, interfaces, and properties of blocks. IBDs provide a means to describe the internal structure of a block. Figure 2 shows examples for a BDD and an IBD comprising most of the SysML modeling elements reused or specialized in the profile presented in Section 5:

Blocks extend UML classes and are used to model system decomposition, system interaction, and various system properties such as values [33]. The properties of blocks are organized in compartments. The *values*-compartment lists a block's *ValueProperties*, which are *ValueTypes* having composite aggregation, *e.g.,* `numPoles` of the block `BiotSavart`. *PartProperties*, listed in the *parts*-compartment, are blocks that have composite aggregation [33], *e.g.,* `leverArm` of `BiotSavart`. *ConstraintBlocks* are specific blocks used to integrate engineering analyses, *e.g.,* reliability, but also to specify physical constraints as mathematical expressions [33]. *ConstraintProperties* of a block are ConstraintBlocks having composite aggregation, *e.g.,* `voltage` of `SynchronousDriving` in Figure 2. *ConstraintParameters* are the ValueProperties of ConstraintBlocks and represent the variables of such expressions. *ProxyPorts* make features or internal parts of a block available for other components, but do not represent separate parts of the system nor exhibit behavior or comprise internal parts [33]. They are properties of a block typed by *InterfaceBlocks* and identified by the stereotype «proxy», *e.g.,* `p_el` of the block `ConvertEnergyElToMech` typed by the InterfaceBlock `ElEnergy_in`. InterfaceBlocks specify the elements that flow between a block and its environment through *FlowProperties* with direction *in, out* or, *inout* [33]. Section 5 gives more details on InterfaceBlocks and their usage. An internal structure, *i.e.,* the interconnection of a composition of blocks, is modeled by an IBD that belongs to the composed block. The bottom of Figure 2 shows an IBD that models the functional structure of the running example introduced in Section 2. The IBD shows the interaction between the PartProperties of the block `GenerateVolumeFlow` through *Connectors* between the ProxyPorts of the PartProperties. In IBDs, ProxyPorts, typed by InterfaceBlocks that have FlowProperties of only one direction which is not `inout`, hold an arrow showing this unique direction. For example, `p_el` typed by the InterfaceBlock `ElEnergy_in` in the IBD in Figure 2, has only FlowProperties of direction `in`. *Parametric diagrams* are restricted IBDs that show only the usage of ConstraintBlocks. *BindingConnectors* are connectors that specify the equality of the numeric values of the properties at both ends and hold the stereotype «equal» [33]. Being defined on UML [32], SysML offers infrastructure to create profiles by defining stereotypes as extensions of meta-classes or as sub-stereotypes [33].

4 A META-MODEL FOR FUNCTIONAL ARCHITECTURES OF CPS

Ongoing research in ME deals with narrowing the gap between functional requirements and the product architecture. Proclaimed methods differ in terms of terminology and details. Prevalently, these methods describe the product's function as a *functional structure* and use descriptions of *physical effects* as links to geometric components [40, 54]. Design catalogs [28, 44, 53] document recurring elements, such as functions, physical effects, or geometries to enable their systematic variation and rational reuse. In practice, however, mechanical engineers rarely explicate functional structures or utilize design catalogs during development. Mostly, the link between function and geometrical component is kept in the engineer's mind. Formalizing the knowledge from design catalogs and linking the information to detailed models describing physical effects and geometry provides the basis for systematic reuse in an MDE approach. A modeling language that enables to model functions, solutions, geometry and physical effects, where models of solutions comprise the latter two serves this purpose. The formal nature of the modeling language establishes systematic relationships between models of functions and solutions, as well as geometry and physical effect. Models in this language can be (re)used to, *e.g.,* investigate different solutions for products at early development stages by varying solutions to functions. This section summarizes and extends the concepts of [28, 40] and presents a formalizing meta-model [48] which provides the conceptual basis for defining such modeling languages.

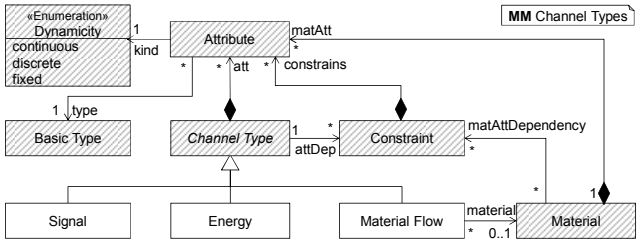

Figure 3: Meta-model that describes the types of functional flows. The shaded classes mark extensions to [28, 40].

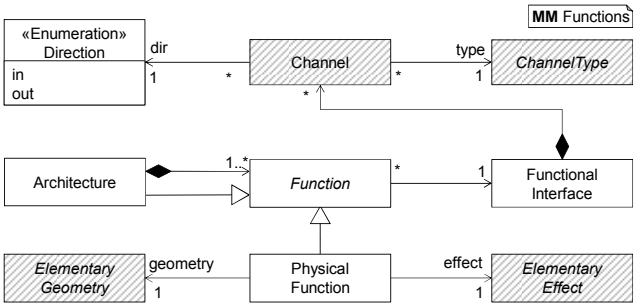

Figure 4: Meta-model of architectures. The shaded classes are extensions of [28, 40].

4.1 Functional Structures

Mechanical design theory gives a definition of function based on the concept that a system or a part of it can be delimited by a boundary, through which physical quantities can enter and leave the system as *functional flows* [40]. The *function* of the delimited system transforms the incoming functional flows to the outgoing flows. Therein, functional flows are classified as flows of *signal*, *energy* and *material* [40, 52]. In the running example (see Section 2), the main function of the pump is to "apply fluid with mechanical energy". The function applies the incoming cooling medium flow with mechanical energy (P_{mech}), such that the cooling medium leaving the system boundary is accelerated. The function of a mechanical system breaks down into several sub-functions linked by functional flows [40]. Functions are referred to as *elementary functions* if the transformation of flows they represent does not physically decompose further [28]. The following section captures and extends these concepts in the meta-models shown in Figure 3 and 4 from a language engineering point of view. The shaded classes mark extensions of [28, 40] to enable the formalization.

Functional Interfaces: To capture kinds of functional flows, our meta-model uses a concept of *channel types*. In an object-oriented fashion [47, 48], *channel type*s specify the type of a functional flow by means of *attributes* and *constraints*. The channel types *energy*, *signal*, and *material flow* [28, 40] comprise attributes representing the characteristics of physical flows of energy or material and of logical signal flows. A material flow may additionally reference a *material* representing the physical material that is flowing in detail, *e.g.,* [31]. Material engineering is out of the scope of this paper and we consider materials solely as types specified by attributes and constraints. Here, constraints are mathematical expressions and represent (physical) dependencies between the attributes of a channel type or a material.

The attributes of channel types have a *basic type* specifying the data type of the attribute. The *dynamicity* specifies how an attribute changes its value during system runtime. For example, theory on software functions often considers signals to change *discrete*ly [11]. Attributes that represent physical characteristics belong to either an energy, or material flow channel type or to a material. These attributes typically represent static, *i.e., fixed*, characteristics of the physical entity, such as *e.g.,* the specific heat capacity of a material, or they represent characteristics that change *continuous*ly at system runtime, *e.g.,* the temperature of a physical part.

Functional Architectures: Describing system functions as a decomposed hierarchy is common to ME [28, 40] and SE [8, 17]. Therein, the functional composition of the sub-functions specifies the behavior of the decomposed function [11, 40]. Each *function* has an *interface* comprising a set of typed *channels* with an unambiguous *direction*, that is either *in* or *out* (*cf.* Figure 4). The meta-model, utilizes the composite pattern [19] to capture an *architecture* as a hierarchical composition of *function*s. The meta-model defines modeling languages for ME, thus, the leaves of a functional architecture are *physical function*s, which capture the elementary functions of [28]. The abstract *elementary effect* and *elementary geometry* bridge the gap between the physical function and its principle solutions since their implementations represent effects and geometries suited to realize the physical function. As detailed in Section 4.2, specifying an interaction of a physical effect and a quantitative geometry adds a behavior to the physical function. Extending the meta-model to capture functional architectures across domains, is possible by integrating description techniques for the behavior of leaf-functions from other domains: Software and control engineering, for example, often utilize various kinds, of automata to specify functional behavior, *e.g.,* [1, 13, 42, 47].

4.2 Effect Catalogs and Principle Solutions

ME created design catalogs to rationalize the PDP by storing proven solutions for recurring design tasks [22, 40]. Various design catalogs exist in ME, *e.g.,* for machine elements [44] or mechanic connections [45]. A popular contribution focuses on physical effects to support engineering solutions that realize a functional structure within the PDP [27, 28, 40]. The catalog comprises 350 physical effects that are mapped to the elementary function they are suited to fulfill [28]. For the conversion of electrical to mechanical energy in the running example (see Section 2), the effect catalog lists, *e.g.,* the *Biot-Savart effect* (*cf.* Figure 11). *Principle solutions* characterize how a physical effect, given a qualitative geometry with certain material properties [40], fulfills a physical function, *i.e.,* the leaf of a functional structure. The principle solution of the physical function "apply fluid with mechanical energy" in the running example (see Section 2) could, for example, specify that the selected effect should be implemented with the principle geometry of a rotating wheel mounted within the cylinder through which the fluid flows, to which Section 5.3 provides further details.

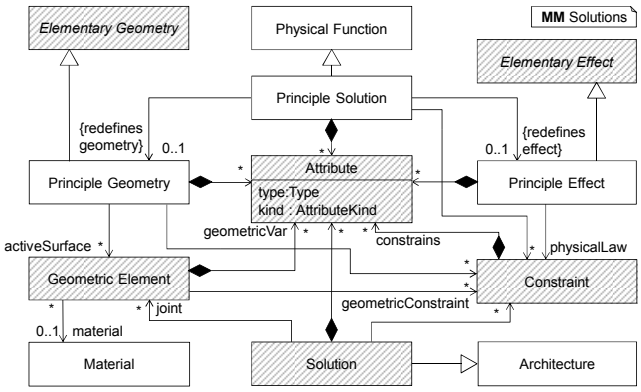

Figure 5: Meta-model of solutions. The shaded classes are extensions to [28, 40].

Meta-model of Solutions. Figure 5 shows the meta-model for principle solutions of physical functions. To enable systematic design of principle solutions to physical functions, the former must be *consistent* to the latter. That is, the specification of the function must be fulfilled by the solution. Therefore, we consider a *principle solution* to refine a *physical function* (*cf.* Figure 4) by selecting a *principle effect* and a *principle geometry*. To support the idea of [28], that elementary functions point to the physical effects suited to realize them, principle effects implement the abstract elementary effect of the physical function. Similarly, suitable principle geometries implement the elementary geometry of the respective physical function. Models of physical functions, principle effects, and principle geometries thereby become reusable. The implementation relation from the elementary geometry and elementary effect to their principle counterparts formalizes the mapping between elementary function and physical effect proposed in [28].

Physical phenomena are described as mathematical formulas over geometric and dynamic variables, captured by the constraints a principle effect is composed of in the meta-model in Figure 5. The lever effect, for example, requires a lever arm to be mounted on a pivot. Putting force on the longer end of the lever arm, by the lever effect, causes a larger force to occur at the other end. The mathematical formula relates the strength of both forces to the lengths of the lever emanating from the pivot. Principle solutions describe how such phenomena can be utilized to fulfill a physical function by relating attributes of the principle effect and the principle geometry through constraints. A physical effect often comprises an interaction of multiple physical laws. Principle effects model physical effects by specifying this interaction as a network of constraints, where each constraint represents a physical law. A principle solution relates the attributes of the principle effect to the attributes of the principle geometry and the attributes of the channel types of the incoming and outgoing channels. The principle effect, thereby, mathematically describes the transformation of the incoming flows to the outgoing flows, *i.e.,* the behavior of the physical function, dependent on the chosen principle geometry. The meta-model considers principle geometries as compositions of *geometric element*s which are types characterized by attributes and constraints between these attributes. The fixed attributes of a geometric element represent static characteristics, *e.g.,* dimensions, continuous attributes specify dynamic characteristics, *e.g.,* velocity, and constraints specify dependencies between these attributes, *e.g.,* that the volume is the product of the dimensions. ME practitioners typically rely on simulation languages to model constraints or physical effects [2, 6, 21, 41, 55], which are tailored particularly for modeling differential equations, and respective tooling often comes with powerful solvers. For geometry, CAD models are the typical choice of the domain experts. Languages encoding the meta-model may integrate such models.

Solutions implement architectures and redefine the inherited physical functions or architectures to principle solutions or solutions, respectively. A solution's components, *i.e.,* (principle) solutions, are interconnected by functional flows as inherited from the architecture. Constraints of a solution express the mathematical dependencies between the attributes of its components.

5 MODELING MECHANICAL FUNCTIONS AND SOLUTIONS IN SYSML

The previous section conceived an expressive meta-model which formalizes and extends the concepts of mechanical design theory [28, 40] from a language engineering point of view. The extension of [28, 40] includes, *e.g.,* the notion of channel types, the systematic mapping of principle solutions to elementary functions as well as the relation between functions and solutions. The latter provides the foundation to systematically define domain-specific modeling languages for the ME domain which enable to model the functional architecture of a mechanical system. In the following, we propose SysML for Functional Mechanical Architectures (SysML4FMArch), a SysML profile which gives a concrete syntax that encodes the meta-model by specifying suitable stereotypes and relations between them.

5.1 Functional Interface

Functions interact by means of signal, material or energy flows [40], which our meta-model captures as channel types. To this effect, SysML4FMArch specifies the stereotypes «Signal», «Energy», and «MaterialFlow», as well as «Material» and «Attribute» which encode the respective elements of the meta-model (*cf.* Figure 3). Attributes are specific ValueTypes with a property of the enumeration type «Dynamicity» which encodes the respective element of the meta-model and indicates how the numeric value of the modeled attribute changes during runtime. The abstract channel type of the meta-model does not have a respective stereotype in SysML4FMArch to assure that functional flows are always classified. The attributes of channel types or materials are represented as «Attribute»-typed ValueProperties. Constraints between these attributes are modeled by ConstraintBlocks and BindingConnectors [33] .

The examples in Figure 6 illustrate this in the BDD and the parametric diagram at the bottom left: A flow of fluid is represented by a «MaterialFlow»-block with three attributes. One is of the real number type `Pressure` and specifies the unit `Pascal` as well as the «Dynamicity» `cont`, which indicates, that ValueProperties typed by `Pressure` change their value continuously during system runtime.

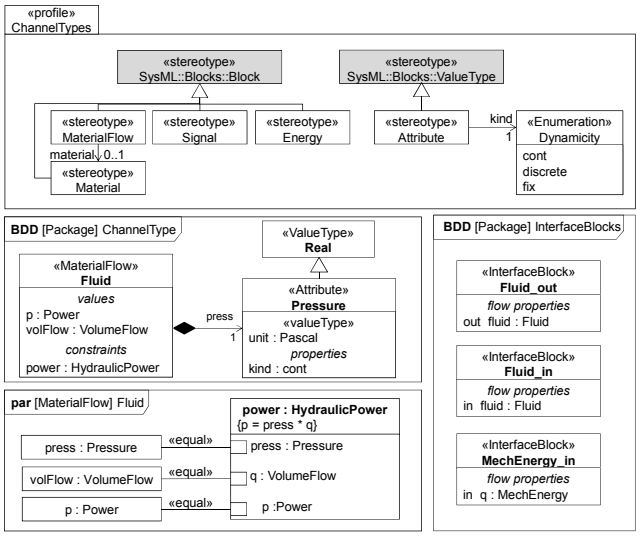

Figure 6: Top: SysML4FMArch's encoding of the meta-model in Figure 3. Bottom: Examples for channel types modeled in SysML4FMArch.

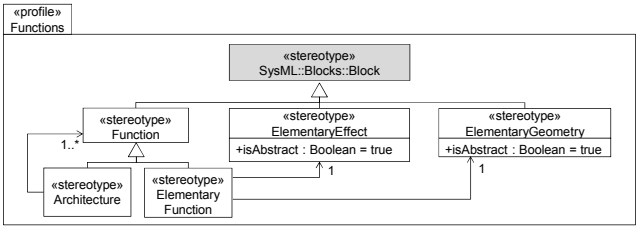

Figure 7: SysML encoding of functions as defined by the meta-model in Figure 4.

The ConstraintBlock `HydraulicPower`, shown at the bottom left, models the physical relationship between the attributes of `Fluid`.

Channels of functional interfaces are represented by ProxyPorts [33]. SysML4FMArch requires the InterfaceBlocks typing these ProxyPorts to have FlowProperties of unambiguous direction, *i.e.,* the usage of direction `inout` or specifying FlowProperties of multiple directions is not allowed. The bottom right of Figure 6 shows examples for InterfaceBlocks to be used for typing ProxyPorts. Note that `Fluid` and `MechEnergy` are models of the physical entities and not of flows of information about these entities.

5.2 Functions

Figure 7 shows how SysML4FMArch encodes the notions of architecture, function, physical function, elementary effect and elementary geometry, that were introduced in Section 4.1.

Architectures and physical functions are encoded as stereotyped blocks (see Figure 7). Since the profile is intended for users with a background in ME, physical functions are encoded by the stereotype «ElementaryFunction» to convey the terminology of [28, 40]. SysML4FMArch provides the «Function»-stereotype that encodes

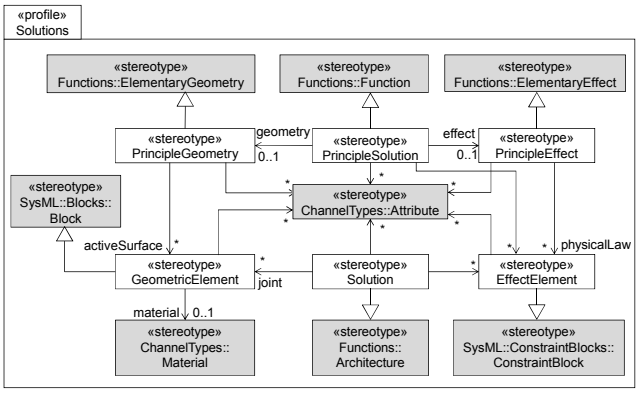

Figure 8: SysML4FMArch encoding of solutions (*cf.* Figure 5).

the notion of function which is abstract in the meta-model. This enables black-box use of functions and allows to postpone specifying a behavior, as it may not be known at early development stages whether the function has to be further decomposed.

To illustrate this, consider the example «Architecture» shown in Figure 2. The IBD at the bottom shows the internal structure of the «Architecture» `GenerateVolumeFlow` which comprises two PartProperties of «ElementaryFunction»-type, *i.e.,* `moveFluid` and `elToMech` as well as the «Architecture» `setVRot`.

The physical functions described in [28] can be digitized by storing respective «ElementaryFunction»-blocks in a SysML ModelLibrary [33]. The general specifications of [28] often need to be refined to integrate them as part of an architecture. Utilizing a specialization of an «ElementaryFunction» allows to refine the function's interface while preserving consistency.

Elementary Effects and Elementary Geometries: A physical function can be realized by selecting a physical effect from a finite list [28, 40]. The interconnection of a physical effect and geometry can be interpreted as the behavior of a physical function. Thus, physical functions comprise an elementary geometry and an elementary effect in the meta-model (*cf.* Section 4.1). SysML4FMArch encodes these elements by abstract blocks with respective stereotypes (*cf.* Figure 7). The abstract «ElementaryGeometry» and «ElementaryEffect» serve as placeholders for the «PrincipleEffect» and «PrincipleGeometry». The latter are implementations of their elementary counterparts that can be selected when creating a «PrincipleSolution» that specializes an «ElementaryFunction» (see Section 5.3). The example at the top of Figure 2 illustrates this: The «PrincipleSolution» `SynchronousDriving` specializes `ConvertEnergyElToMech` which redefines the elementary effect and elementary geometry to their implementations `BioSavart` and `RotorStator`, respectively.

5.3 Solution Architectures

To specify a principle solution for a physical function in an architecture, the engineer chooses implementations of the elementary geometry and elementary effect owned by the physical function and specifies the constraints between the values of both components. Figure 8 shows the encoding of the meta-model in Figure 5 in SysML4FMArch which is detailed in the following paragraphs.

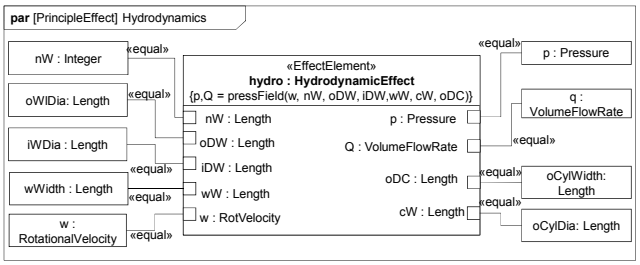

Figure 9: Principle effect representing the cause for turbulences in flowing fluids, modeled in SysML4FMArch.

Effect Elements and Geometric Elements: SysML4FMArch represents geometric elements as blocks with the stereotype «GeometricElement», comprising ValueProperties typed by «Attribute»-ValueTypes. Constraints between attributes of geometric elements are modeled, either as BindingConnectors in case of equalities or as regular ConstraintBlocks in case of more complex mathematical relationships. Effect elements, *i.e.,* physical laws or relations between attributes of principle effects, are modeled as specific SysML ConstraintBlocks with the «EffectElement»-stereotype. Their ValueProperties are typed by «Attribute»-ValueTypes and represent the attributes of the constraint (*cf.* Figure 8). An «EffectElement» may link to a simulation model, which can be realized *e.g.,* as proposed in [25]. In SysML4FMArch, the variables of the simulation are represented by the ConstraintProperties of «EffectElements». The attributes of a geometric element are modeled by the ValueProperties of a «GeometricElement».

Principle Geometry and Principle Effect: Principle geometries comprise the active surfaces between which physical effects come into action. In analogy to the meta-model, a «PrincipleGeometry» therefore comprises «GeometricElement»-PartProperties, each representing an active surface. Since a principle effect is an interaction of physical laws, a «PrincipleEffect» in SysML4FMArch comprises ConstraintProperties typed by an «EffectElement». BindingConnectors connect the «Attribute»-ValueProperties of a «PrincipleEffect» to attributes of its «EffectElement»-ConstraintProperties and, thereby represent equality of the numeric values of the ValueProperties at the connector's ends.

To illustrate this, consider the principle effect modeled in Figure 9. The equation $p \cdot Q = M \cdot \omega$ (1) describes the physical law that causes turbulences, *i.e.,* a rotational velocity, within a flowing fluid [43]. Here, p is the fluid's pressure, Q is the volume flow rate, ω is the rotational velocity, and M is a momentum imposed by a mechanical energy. Thus, the «PrincipleEffect» `Hydrodynamics` has «Attributes» of respective types which all specify the «Dynamicity» `cont` (*e.g.,* Figure 6 shows the definition of `Pressure`). The momentum strongly depends on the geometric setup, through which the fluid is flowing. In the context of the running example (*cf.* Section 2), we assume that the fluid flows through a tubular pipe with a length of `oCylWidth` and a diameter of `oCylDia`. In the pipe, the fluid flows through a paddle wheel with `nW` paddles, an outer diameter of `oWDia`, an inner diameter of `iWDia`, and a width of `wWidth`. These ValueProperties of `Hydrodynamics` represent geometric variables of `fix` kind, as these attributes are assumed to not change their value at system runtime. The «EffectElement» `HydrodynamicEffect` links to a simulation model that calculates a difference `p` between pressures of the incoming and the outgoing fluid, and a volume flow rate `q` according to Equation (1). BindingConnectors between the ConstraintParameters of the ConstraintProperty `hydro` model the physical relationships between the attributes of the principle effect as stated by the physical law modeled by the «EffectElement» `HydrodynamicEffect`.

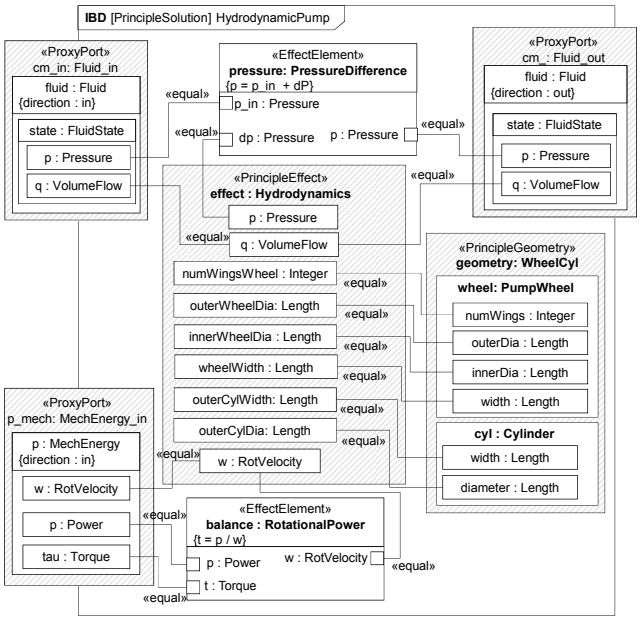

Figure 10: Model of a possible principle solution to realize the elementary function `ApplyFluidWithMechanicalEnergy`.

Solutions and Principle Solutions: A principle solution inherits the functional interface, the elementary geometry and the elementary effect from the physical function it fulfills. By redefining the latter two to concrete implementations, *i.e.,* principle effect and principle geometry, the engineer creates a solution to realize the functionality [28]. The interaction of the principle effect and the principle geometry specifies the behavior of the principle solution. Figure 8 shows how SysML4FMArch encodes this: Principle solutions are represented by blocks with the respective stereotype composed of parts typed by a «PrincipleGeometry» and a «PrincipleEffect» (*cf.* Figure 8). A «PrincipleSolution» must specialize an «ElementaryFunction» and may redefine the inherited «ElementaryGeometry» and «ElementaryEffect» to a «PrincipleGeometry» and a «PrincipleEffect», respectively. The selection of either one may be delayed, indicated by the multiplicity `0..1`. SysML4FMArch uses BindingConnectors to specify the constraints between attributes of principle geometries and principle effects as well as the function's interface. Effectively, this models the behavior of the function by specifying how the function changes the attributes in the representations of the flows that enter or leave the function through its interface. The physical interaction of principle

geometry and principle effect are represented by BindingConnectors between the «EffectElement»-ConstraintProperties and the «GeometricElement»-PartProperties of a «PrincipleSolution». The constraints between ValueProperties of a «Solution» and the contained «PrincipleSolution» components are modeled equivalently.

Figure 10 shows an IBD of `HydrodynamicPump`, which specializes the «ElementaryFunction» `ApplyFluidWithMechEnergy` and inherits its interface. `Hydrodynamics` specializes the «ElementaryEffect» of `ApplyFuidlWithMechEnergy` (both specialization relations are not shown in Figure 10). A simulation linked to the «EffectElement» modeling Equation (1) assumes the fluid to flow through a tubular pipe comprising a paddlewheel. The «PrincipleGeometry» `WheelCyl` specializes the «ElementaryGeometry» of `ApplyFluidWithMechEnergy` and has PartProperties of type `PumpWheel` and `Cylinder`. These represent a pair of active surfaces possible to enforce the represented effect, and assign the attributes of the effect to distinguishable geometric shapes. The pressure of the outgoing fluid is determined as the sum of the pressure of the incoming fluid and the pressure difference which results from the hydrodynamic effect acting on the fluid, which is modeled by the «EffectElement» `PressureDifference`. The rotational velocity, power, and torque imposed by the incoming mechanical energy must obey the law of energy conservation modeled by the «EffectElement» `RotationalPower`.

6 MODELING EXAMPLE: AUTOMOTIVE ELECTRICAL COOLANT PUMP

This section presents an extract from the results of an interdisciplinary industrial project, involving researchers from SE and ME as well as practitioners from the automotive industry. In the project, we have applied SysML4FMArch to model the cooling system for an automotive combustion engine drive train (*cf.* Section 2). This section presents and explains the SysML4FMArch models of the coolant pump, a part of the cooling system, in detail.

6.1 Architecture of the Electric Coolant Pump

The coolant pump's main functionality is to keep the cooling medium flowing which is physically necessary for it to absorb the heat from the engine's cylinders. The IBD of the «Architecture» `GenerateVolumeFlow` in Figure 2 shows the decomposition of this functionality. The architecture has ProxyPorts representing three incoming flows, *i.e.,* `cm_in` which represents an incoming cooling medium, an electrical energy `pEl`, and a signal `flowControl`, as well as `cm_out` which represents the outgoing flow of the cooling medium. Figure 6 shows the InterfaceBlocks for typing the ProxyPorts representing the functional flows of fluid. The other InterfaceBlocks have FlowProperties typed by a «Signal» `ControlSignal` which is defined similar to `Fluid` but specifies the unit `m/s` and the AttributeKind `discrete`, and by an«Energy» `ElEnergy` which represents electrical energy by means of a real number, the unit `Watt`, and the AttributeKind `cont`. The flow `flowControl` represents an information flow (changing its value discretely at runtime) telling how fast the outgoing fluid has to flow, in order to absorb enough heat from the engine, which enters the function `SetRotationalVelocity`. The latter is modeled as «Architecture» that calculates a necessary amount of electrical energy. The outgoing signal flow enters an actuator function which

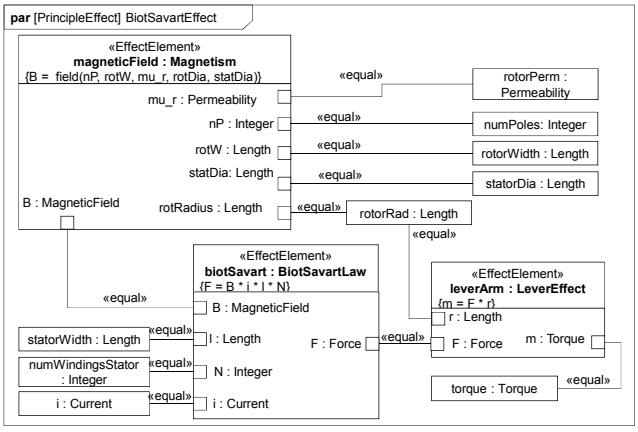

Figure 11: The Biot-Savart effect is an interaction of physical laws: `Magnetism`, the `LeverEffect` and the `BiotSavartLaw`. The figure shows a model of this principle effect to be used in principle solutions of `ConvertEnergyElToMech` .

outputs a flow of electrical energy (`pEl_out`). The «ElementaryFunction» `ConvertEnergyElToMech` represents a physical function that converts the flow of electrical energy into a flow of mechanical energy `p_mech_out`. The physical function represented by the «ElementaryFunction» `ApplyFluidWithMechEnergy` impinges this mechanical energy `p_mech` upon the incoming fluid `cm_in` and resulting in the outgoing flow `fluid_out`.

6.2 Solution-Models

The «Architecture» `GenerateVolumeFlow` comprises two «ElementaryFunctions» for which [28] lists physical effects suitable to realize their functionality. In the solution of this architecture considered here, the hydrodynamic effect provides the acceleration of the fluid specified by `ApplyFluidWithMechEnergy`. Figure 9 shows the SysML4FMArch-model of the principle solution using the hydrodynamic effect which was explained previously in Section 5.3. This section details a principle solution to convert electrical to mechanical energy using the *Biot-Savart-Effect* [24]. This principle solution realizes the elementary function `ConvertEnergyElToMech` in our running example (*cf.* Figure 2).

Principle Solution to Convert Electrical to Mechanical Energy: The BDD in Figure 12 shows the «PrincipleEffect» `BiotSavart` which specializes the «ElementaryEffect» `EE_ConvEnElToMech`. The principle effect is an interaction of multiple physical laws, therefore, `BiotSavart` comprises three «EffectElements», *i.e.,* `Magnetism`, `BiotSavartLaw` and `LeverEffect`, connected by BindingConnectors which represents the following: An electromagnetic coil (stator) is positioned within a magnetic field B. The magnetic field is created by a permanent-magnet (rotor), that is placed at a distance r to a rotation axis such that it may rotate around the stator. Once a voltage implies a current i in the conductor, the Lorentz-force starts acting on the rotor. Due to the *lever-effect*, a mechanical torque M occurs around the rotation axis, causing the rotor to rotate. The rotation reflects the existence of mechanical energy. The physical laws are (1) $B = \mu_0 \cdot \mu_r \cdot H$, (2) $M = F \cdot r$, and (3) $F = B \cdot i \cdot l \cdot N$, where

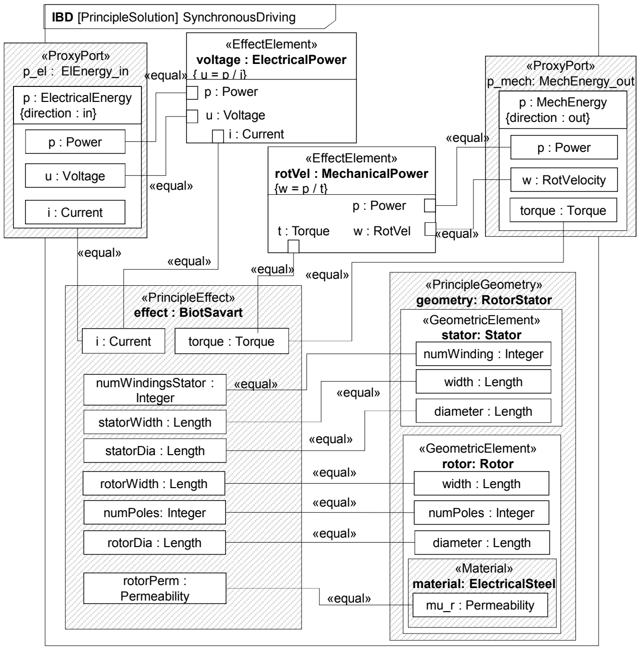

Figure 12: Principle solution of `ConvertEnergyElToMech` using the «PrincipleEffect» `BiotSavart`, which represents the electric drive of the cooling pump.

μ_0 is the vacuum permeability, μ_r is the permeability of the rotor, and H is the magnetic field strength induced by the rotor. Further, l is the length and N the number of windings of the stator. If losses are not considered, electrical input power is equal to mechanical output power (see, *e.g.,* [24] for details).

Figure 11 shows a SysML4FMArch-model of this principle effect: The magnetic field strength H depends on the number of poles `numPoles` and the diameter of the rotor as well as the diameter of the conductor. By means similar to [25], the «EffectElement» `Magnetism` links to a simulation model that calculates the magnetic field `B` from the geometric attributes of the stator, *i.e.,* the conductor and the rotor, according to Equation (1). The «EffectElement» `BiotSavartLaw` models Equation (2): The Lorentz-force `F` depends on the magnetic field `B`, the electric current `i`, the stator's `statorWidth` and the number of windings of the stator `numWindingsStator`. The «EffectElement» `leverEffect` models Equation (3): The torque that acts around the rotation axis depends on the Lorentz-force acting on the rotor and length of the lever arm, *i.e.,* the diameter of the rotor `rotorDia`.

Figure 12 shows a principle solution to apply a fluid with mechanical energy that (re-)uses the «PrincipleEffect» `BiotSavart`: `SynchronousDriving` specializes the «ElementaryFunction» `ConvertEnergyElToMech` (*cf.* Figure 2) and therefore inherits the interface. Further, `SynchronousDriving` specifies the «PrincipleEffect» `BiotSavart` explained above and the «PrincipleGeometry» `RotorStator` which models a geometry comprising a rotor and a stator, both characterized by attributes of `fix` AttributeKind. The BindingConnectors between the attributes of the modeled principle effect and of the geometric elements of the represented principle geometry as well as the attributes of the represented channel types model the equality of their numeric values. The «EffectElements» `ElectricalPower` and `MechanicalPower` represent the physical law of energy conservation.

Solution to Generate a Volume Flow. The models of the principle solutions introduced above can be used to model a solution to the «Architecture» `GenerateVolumeFlow` whose internal structure is modeled in Figure 2. A solution to `GenerateVolumeFlow` is a «Solution»-block that specializes the «Architecture» `GenerateVolumeFlow`. The «PrincipleSolution»-blocks `HydrodynamicPump` and `SynchronousDriving` specialize the «ElementaryFunction»-blocks `ApplyFluidWithMechEnergy` as well as `ConvertEnergyElToMech`, respectively. The latter type the PartProperties `moveFluid` and `elToMech` of `GenerateVolumeFlow`, as shown in Figure 2. A «Solution» to this «Architecture» inherits the interface and the PartProperties of `GenerateVolumeFlow`. By redefining `ConvertEnergyElToMech` to `SynchronousDriving` and `ApplyFluidWithMechEnergy` to `HydrodynamicPump`, this solution integrates these «PrincipleSolutions» and forms a model of the solution to the entire architecture. In this case, the solution models an electrical coolant pump.

7 RELATED WORK AND DISCUSSION

Modeling as the act of describing or prescribing properties of the system under development, is the essential foundation for systematically engineering (cyber-physical) systems. MDE employs formal modeling languages to enable frontloading of analysis and design exploration to reduce engineering costs, facilitate collaboration among domain experts, and supports the synthesis of system parts by automation. Research on this topic is scattered across the domains of CPS engineering, including ME and SE.

Related Work. Ongoing research has produced theories, and modeling languages for engineering software and electronic functions of CPS, *e.g.,* [1, 42], as well as for designing [23, 49], engineering [4], and operating [3] CPS in various domains. Most of these approaches consider modeling only through the lens of software engineering, *i.e.,* for discrete and functional systems. Where continuity and geometry are supported, the theories and languages do not support established processes or modeling concepts from other (*i.e.,* the "physical") domains, such as ME.

In the Focus theory [11], systems are composed of components that realize stream processing functions. As functions communicate via channels only, they can be (de)composed and refined systematically, where refinement considers both, structure and the behavior of components. Applying Focus's notion of refinement within a model-driven functional PDP is subject to ongoing research.

In ME, a variety of design catalogs to aid the design process regarding various aspects [18] exist in the literature, *e.g.,* [40, 46]. Approaches that digitize such catalogs, *e.g.,* [18, 34], focus on making the (extended) information from existing design catalogs accessible by providing digital textual descriptions complemented with mathematical expressions or sketches. Lacking a representation in a formal modeling language that also enables to integrate the information within a functional architecture of a mechanical system hinders to apply these approaches in a model-driven PDP. Modeling languages based on UML or SysML have

emerged or been used in the ME domain, *e.g.,* MechatronicUML [12], SysML4Modelica [5], or SysML4Mechatronics [26], in the field of production systems engineering [16], and in the context of Industry 4.0 [58], *e.g.,* UML4IoT [51]. Neither of these languages enable to relate (elementary) functions and (principle) solutions of mechanical systems as part of a systematic PDP. The FAS-method [30, 56], extended for ME by FAS4M [39] promotes modeling functional architectures for system design and both define respective SysML profiles. As introduced in [37, 38] the latter uses trace links to underly SysML elements with informal sketches of geometric components. The focus of these contributions lies on the connection between requirements and function. In contrast to our approach, principle solutions, here, are described by informal sketches that neither distinguish between principle geometry and principle effect nor enable automatic processing. This prevents utilizing the information from design catalogs such as, *e.g.,* [28], and to compose the physical product architecture of geometric elements related to physical effects by a principle solution. This holds similarly for the techniques proposed in [15, 20, 57, 59]. In particular, the approaches in [15, 20] do not consider functional structures in the sense of [28, 40] (see Section 4.1) and do not systematically establish consistency between function and principle geometry in a model-driven approach. Currently, precise modeling languages tailored to support the PDP based on the foundations of functional architectures established in [27, 28] do not exist. Explicit modeling techniques for the PDP in ME do not support the time-honored paradigms that paved the way for the success of software engineering, such as abstraction, automation, composition, refinement, and separation of concerns.

Discussion. The meta-model and its encoding in SysML emerged during an interdisciplinary project comprising researchers from SE and ME as well as practitioners from the automotive industry. So far, the meta-model captures and extends the notion of functional architectures prevalent in mechanical design theory [28, 40]. Therein, components interact through functional flows, and physical functions are implemented by principle solutions, *i.e.,* an interaction of a physical effect and a geometry. Integrating description techniques for functional behavior prevalent, *e.g.,* in the software [11, 13] and control engineering domains [1, 42], is subject to ongoing research. As SysML is fairly in known in the automotive domain [14, 29] and since there exist modeling tools with integrated model-processing, we encoded the meta-model as a SysML profile. To test the approach in the ME domain, we engineered an automotive coolant pump and modeled its functional architecture as well as the solutions to each function in SysML4FMArch. The model comprises multiple SysML4FMArch-diagrams which were presented exemplary throughout the paper. The systematic relation between functions and solutions in the SysML4FMArch-models enabled to use the tooling effectively for automation during the mechanical design process. For example, we tested the suitability of the chosen principle solutions (*cf.* Figure 10 and Figure 12) by linking virtual simulations to the effect elements in the SysML4FMArch model and for virtual dimensioning of the pump wheel, *i.e.,* the automatic manipulation of values of its geometric attributes, (*cf.* Figure 10), which was enabled by utilizing the automatic model execution functionality of existing SysML tools. Towards the end of the project, the pump wheel was 3D-printed to obtain a prototype of a part of the physical product. Further automation for functional testing and dimensioning as well as digitizing a design catalog such as [28] are subject to ongoing research. However, SysML has its drawbacks, *e.g.,* regarding modeling efficiency, and intuitiveness. SysML's nature as a general purpose language and the inherited UML concepts decrease understanding and ease of use for ME practitioners, as, for them, these concepts are not as intuitive as for SE practitioners. Further, the graphical nature of SysML may hinder manageability of SysML4FMArch models with many attributes. The lack of formal semantics for SysML hinder the implementation of product-specific automatic model processing and tailored model analyses based on mathematical theories.

8 CONCLUSION

This paper formalized and extended the concepts of [28, 40] in a meta-model that defines modeling languages for the ME domain. Further, we encoded the meta-model in the SysML profile SysML4FMArch and employed the language within in an interdisciplinary, industrial project to engineer an automotive coolant pump. As a result, the models could be used for automatic, virtual dimensioning and testing, which holds out the prospect of an agile model-driven PDP supported by automation. While SysML has its drawbacks regarding formality and intuitiveness, the results of the project signpost the potential of utilizing modeling languages for explicating functional architectures of a technical system and making the knowledge of design catalogs assimilable for a holistic MDE approach that narrows the gap between the functional and the product architecture by means of abstraction.

REFERENCES

[1] Rajeev Alur. 2015. *Principles of cyber-physical systems.*

[2] Faysal Andary, Joerg Berroth, and Georg Jacobs. 2019. An Energy-Based Load Distribution Approach for the Application of Gear Mesh Stiffness on Elastic Bodies. *Journal of Mechanical Design* 141, 9 (2019).

[3] Patrick Bareiss, Daniel Schütz, Rafael Priego, Marga Marcos, and Birgit Vogel-Heuser. 2016. A model-based failure recovery approach for automated production systems combining sysml and industrial standards. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA).* IEEE, 1–7.

[4] Luca Berardinelli, Stefan Biffl, Arndt Lüder, Emanuel Mätzler, Tanja Mayerhofer, Manuel Wimmer, and Sabine Wolny. 2016. Cross-disciplinary engineering with AutomationML and SysML. *at-Automatisierungstechnik* 64, 4 (2016), 253–269.

[5] Olaf Berndt, Uwe Freiherr von Lukas, and Arjan Kuijper. 2015. Functional Modelling And Simulation Of Overall System Ship-Virtual Methods For Engineering And Commissioning In Shipbuilding.. In *ECMS.* 347–353.

[6] Joerg Berroth, Georg Jacobs, Tobias Kroll, and Ralf Schelenz. 2016. Investigation on pitch system loads by means of an integral multi body simulation approach. *Journal of Physics: Conference Series* 753 (2016).

[7] Manfred Broy. 2006. Challenges in Automotive Software Engineering. In *Proceeding of the 28th international conference on Software engineering - ICSE '06.*

[8] Manfred Broy. 2007. Model-driven architecture-centric engineering of (embedded) software intensive systems: modeling theories and architectural milestones. *Innovations Syst Softw Eng* 3 (2007).

[9] Manfred Broy. 2010. A Logical Basis for Component-Oriented Software and Systems Engineering. *Comput. J.* 53, 10 (2010).

[10] Manfred Broy, · Heinrich Daembkes, and Janos Sztipanovits. 2019. Editorial to the theme section on model-based design of cyber-physical systems. *Software & Systems Modeling* 18 (2019).

[11] M Broy and Ketil Stølen. 2001. *Specification and development of interactive systems.*

[12] Sven Burmester, Holger Giese, and Matthias Tichy. 2004. Model-driven development of reconfigurable mechatronic systems with mechatronic UML. In *Model Driven Architecture.* Springer, 47–61.

[13] Arvid Butting, Arne Haber, Lars Hermerschmidt, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. 2017. Systematic Language Extension Mechanisms for the MontiArc Architecture Description Language. In *European Conference*

on Modelling Foundations and Applications (ECMFA'17) (Marburg) *(LNCS 10376)*. Springer, 53–70.
[14] Imke Drave, Steffen Hillemacher, Timo Greifenberg, Stefan Kriebel, Evgeny Kusmenko, Matthias Markthaler, Philipp Orth, Karin Samira Salman, Johannes Richenhagen, Bernhard Rumpe, Christoph Schulze, Michael von Wenckstern, and Andreas Wortmann. 2019. SMArDT modeling for automotive software testing.
[15] Martin Eigner, Torsten Gilz, and Radoslav Zafirov. 2012. Proposal for functional product description as part of a PLM solution in interdisciplinary product development.
[16] Stefan Feldmann, Sebastian JI Herzig, Konstantin Kernschmidt, Thomas Wolfenstetter, Daniel Kammerl, Ahsan Qamar, Udo Lindemann, Helmut Krcmar, Christiaan JJ Paredis, and Birgit Vogel-Heuser. 2015. Towards effective management of inconsistencies in model-based engineering of automated production systems. *IFAC-PapersOnLine* 48, 3 (2015), 916–923.
[17] Robert France and Bernhard Rumpe. 2007. Model-Driven Development of Complex Software: A Research Roadmap. In *Future of Software Engineering 2007 at ICSE*.
[18] H.-J Franke, S Löffler, and M Deimel. 2004. Increasing the Efficiency of Design Catalogues By Using Modern Data Processing Techniques. In *DS 32: Proceedings of DESIGN 2004, the 8th International Design Conference, Dubrovnik, Croatia*.
[19] Erich. Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design patterns : elements of reusable object-oriented software*. Addison-Wesley.
[20] Jürgen Gausemeier, Rafal Dorociak, Sebastian Pook, Alexander Nyßen, and Axel Terfloth. 2010. Computer-Aided Cross-Domain Modeling of Mechatronic Systems, Vol. 2.
[21] Reza Golafshan, Georg Jacobs, Matthias Wegerhoff, Pascal Drichel, and Joerg Berroth. 2018. Investigation on the Effects of Structural Dynamics on Rolling Bearing Fault Diagnosis by Means of Multibody Simulation. *International Journal of Rotating Machinery* 2018 (2018), 1–18.
[22] Karl-Heinrich Grote and Erik K. Antonsson. 2009. *Springer Handbook of Mechanical Engineering*. Springer, Berlin.
[23] IEEE Architecture Working Group. 2000. *IEEE Std 1471-2000, Recommended practice for architectural description of software-intensive systems*. Technical Report. IEEE.
[24] Austin Hughes and Bill Drury. 2019. *Electric motors and drives: fundamentals, types and applications*. Newnes.
[25] Thomas Johnson, Aleksandr Kerzhner, Christiaan J.J. Paredis, and Roger Burkhart. 2012. Integrating models and simulations of continuous dynamics into SysML. *Journal of Computing and Information Science in Engineering* 12, 1 (2012).
[26] K. Kernschmidt and B. Vogel-Heuser. 2013. An interdisciplinary SysML based modeling approach for analyzing change influences in production plants to support the engineering. In *2013 IEEE International Conference on Automation Science and Engineering (CASE)*.
[27] Rudolf Koller. 2014. *Konstruktionslehre für den Maschinenbau - Grundlagen zur Neu- und Weiterentwicklung technischer Produkte mit Beispielen* (4 ed.). Springer, Berlin.
[28] Rudolf Koller and Norbert Kastrup. 1998. *Prinziplösungen zur Konstruktion technischer Produkte*. Springer Berlin.
[29] Stefan Kriebel, Matthias Markthaler, Karin Samira Salman, Timo Greifenberg, Steffen Hillemacher, Bernhard Rumpe, Christoph Schulze, Andreas Wortmann, Philipp Orth, and Johannes Richenhagen. 2018. Improving model-based testing in automotive software engineering. In *Proceedings - International Conference on Software Engineering*.
[30] Jesko G. Lamm and Tim Weilkiens. 2014. Method for Deriving Functional Architectures from Use Cases. 17 (2014).
[31] Donald Leo. 2008. *Engineering Analysis of Smart Material Systems*.
[32] Object Management Group. 2015. OMG Unified Modeling Language (OMG UML) Version 2.5.
[33] Object Management Group. 2019. OMG Systems Modeling Language (OMG SysML) Version 1.6.
[34] Johannes Mathias, Tobias Eifler, Roland Engelhardt, Hermann Kloberdanz, Herbert Birkhofer, and Andrea Bohn. 2011. Selection of Physical Effects Based on Disturbances and Robustness Rations in The Early Phases of Robust Design. In *International Conference on Engineering Design*. 11–15.
[35] Nenad Medvidovic, Eric M. Dashofy, and Richard N. Taylor. 2007. Moving architectural description from under the technology lamppost. *Information and Software Technology* 49, 1 (2007), 12–31.
[36] Nenad Medvidovic and Richard N. Taylor. 2000. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering* 26 (2000).
[37] Georg Moeser. 2015. Example on "Usage of Free Sketches in MBSE". (04 2015). https://doi.org/10.5445/IR/1000047231
[38] G. Moeser, A. Albers, and S. Kümpel. 2015. Usage of free sketches in MBSE raising the applicability of Model-Based Systems Engineering for mechanical engineers. In *2015 IEEE International Symposium on Systems Engineering (ISSE)*. 50–55.
[39] Georg Moeser, Christoph Kramer, Martin Grundel, Michael Neubert, Stephan Kümpel, Axel Scheithauer, Sven Kleiner, and Albert Albers. 2015. Fortschrittsbericht zur modellbasierten Unterstützung der Konstrukteurstätigkeit durch FAS4M. In *Tag des Systems Engineering*. Carl Hanser Verlag GmbH & Co. KG, 69–78.
[40] Gerhard Pahl, W. Beitz, Jörg Feldhusen, and Karl-Heinrich Grote. 2007. *Engineering Design - A Systematic Approach* (3 ed.). Springer, London.
[41] Gerwin Pasch, Georg Jacobs, Gregor Höpfner, and Joerg Karl Berroth. 2019. Multi-Domain Simulation for the Assessment of the NVH Behaviour of a Tractor with Hydrostatic-Mechanical Power Split Transmission. *Land.Technik AgEng 2019 : Hannover* 77th International Conference on Agricultural Engineering / VDI-Wissensforum ; Supporters: VDI Max-Eyth Society for Agricultural Engineering (2019).
[42] Claudius Ptolemaeus. 2014. System Design, Modeling, and Simulation using Ptolemy II.
[43] Sulzer Pumps. 2010. *Centrifugal Pump Handbook*. Elsevier.
[44] Karlheinz Roth. 1994. *Konstruieren mit Konstruktionskatalogen - Band I: Konstruktionslehre* (2 ed.). Springer, Berlin.
[45] Karlheinz Roth. 1996. *Konstruieren mit Konstruktionskatalogen - Band III: Verbindungen und Verschlüsse - Lösungsfindung* (2 ed.). Springer, Berlin.
[46] Karlheinz Roth. 2011. Selection of Physical Effects Based on Disturbances and Robustness Rations in The Early Phases of Robust Design. In *International Conference on Engineering Design*.
[47] Bernhard Rumpe. 2016. *Modeling with UML: Language, Concepts, Methods*. Springer International.
[48] Bernhard Rumpe. 2017. *Agile Modeling with UML: Code Generation, Testing, Refactoring*. Springer International.
[49] Chantal Steimer, Jan Fischer, and Jan C Aurich. 2017. Model-based design process for the early phases of manufacturing system planning using SysML. *Procedia CIRP* 60 (2017), 163–168.
[50] K. Stephan. 1994. Thermodynamics. In *Dubbel Handbook of Mechanical Engineering*. Springer London.
[51] Kleanthis Thramboulidis and Foivos Christoulakis. 2016. UML4IoT—A UML-based approach to exploit IoT in cyber-physical manufacturing systems. *Computers in Industry* 82 (2016), 259–272.
[52] Karl Ulrich and Steven Eppinger. 2003. *Product Design and Development* (3 ed.). McGraw-Hill, New York.
[53] VDI. 1982. *VDI 2222 Blatt 2 - Konstruktionsmethodik - Erstellung und Anwendung von Konstruktionskatalogen*. Beuth Verlag, Berlin.
[54] VDI. 1997. *VDI 2222 Blatt 1 - Konstruktionsmethodik - Methodisches Entwickeln von Lösungsprinzipien*. Beuth Verlag, Berlin.
[55] Matthias Wegerhoff, Georg Jacobs, and Pascal Drichel. 2018. Noise, vibration and harshness validation methodology for complex elastic multibody simulation models: With application to an electrified drive train. *Journal of Vibration and Control* 25, 2 (2018).
[56] Tim. Weilkiens, Jesko G. Lamm, Stephan Roth, and Markus Walker. 2015. *Model-based system architecture*.
[57] Stefan Wölkl and Kristina Shea. 2009. A Computational Product Model for Conceptual Design Using SysML.
[58] Andreas Wortmann, Olivier Barais, Benoit Combemale, and Manuel Wimmer. 2020. Modeling Languages in Industry 4.0: an Extended Systematic Mapping Study. *Software and Systems Modeling* 19, 1 (January 2020), 67–94.
[59] Christian Zingel, Albert Albers, Sven Matthiesen, and Michael Maletz. 2012. Experiences and Advancements from One Year of Explorative Application of an Integrated Model-Based Development Technique Using C&C²-A in SysML. *International Journal of Computer Science* 34-39 (2012).