Alexander Hayward, Helmut Schmidt University Hamburg
Marian Daun, University of Duisburg-Essen
Ana Petrovska, Technical University of Munich
Wolfgang Böhm, Technical University of Munich
Lisa Krajinski, University of Duisburg-Essen
Alexander Fay, Helmut Schmidt University Hamburg

4

Function Modeling for Collaborative Embedded Systems

The evolution from traditional embedded systems to dynamically interacting, collaborative embedded systems increases the complexity and the number of requirements involved in the model-based development process. In this chapter, we present the new aspects that need to be considered when modeling functions for collaborative embedded systems and collaborative system groups, such as the relationship between functions of a single system and functions resulting from the interplay of multiple systems. These different aspects are represented by a formal, domain-independent metamodel. To aid understanding, we also apply the metamodel to two different use cases.

4.1 Introduction

In modern development methodologies for complex systems, the modeling of functions represents a historically grown and proven way of dealing with large quantities of requirements that need to be taken into account. A function can be used to describe the purpose of a system at different levels of detail.

The SPES projects

The SPES2020 and SPES_XT projects (cf. [Pohl et al. 2012, Pohl et al. 2016]) have already developed a comprehensive set of science-based methods for modeling and analyzing functions of embedded systems, with a special focus on consistency and semantic coherence as part of a comprehensive methodological framework. The methods are based on the assumption that the embedded systems under development are to be integrated into a static context that is well known at the time of development.

Goal fulfillment via collaboration

The additional assumption considered in CrESt—that individual systems no longer achieve the goals¹ associated with them alone, but rather by collaboration with other systems—results in a range of new challenges for which the existing SPES modeling framework is no longer sufficient and needs to be extended.

CES and CSG

A collaborative embedded system (CES), collaborating with other CESs that may be instances of different system types, should be able to achieve goals that 1) the CES could not achieve alone, or 2) could be achieved more easily or better by combining their functions with other CESs. For collaboration, the participating CESs form a common group, referred to as a collaborative system group (CSG). Since a CSG constitutes itself dynamically at runtime and its members, goals, and functions can change, methods for mastering the complexity are particularly necessary for modeling functions at CSG level.

Outline

In this chapter, we describe the new aspects that have to be considered when modeling functions for both CESs as well as the resulting CSG. To describe these aspects systematically, we use a metamodel. With regard to the derivation of this metamodel, Section 4.2 describes the requirements and aspects on which it is based. In Section 4.3, we provide further background information. We then present the metamodel on a domain-independent level in Section 4.4 and evaluate it in Section 4.5. To enable a better understanding of the metamodel, we apply it to two use cases in Section 4.6, and this is

---

¹ See Chapter 2 for a detailed discussion

followed by related work in Section 4.7 and the conclusion in Section 4.8.

4.2 Methodological Approach

Model-based continuous function-centered engineering processes are already established in engineering practice [Daun et al. 2019a]. To support function-centered development, this chapter proposes a function metamodel. In order to achieve the goal of defining a uniform modeling methodology for functions, we have applied the following research methodology. To ensure applicability to a variety of domains and in various contexts, first, we have gathered requirements from academia and industry, using common requirements elicitation techniques such as interviews, workshops, and in-depth discussions. In addition, we have investigated engineering methods of different domains to foster function-centered engineering. As a result, we have derived a set of seven high-level requirements. For details of the requirements elicitation phase and the results, please refer to [Ludewig et al. 2018]. The following is a brief outline of the major requirements:

Deriving requirements

Requirements 4-1: *Requirements for the function metamodel*

- ❑ **Req.1:** It must be possible to model functions on different abstraction layers. Composition and decomposition of functions must consider the relationship between single functions of CESs and overall functions on the CSG layer. On the highest layer of abstraction, a function can be understood as the emerging result of individual contributing functions. On the lowest abstraction layer, it must be possible to model an atomic function and its contribution to the overall function.
- ❑ **Req.2:** It must be possible to model which overall function of a CSG an individual function of a CES can contribute to and how it can do so. For this purpose, the modeling process must consider whether different inputs and outputs of functions can be connected with those of other functions with regard to compatibility.
- ❑ **Req.3:** Due to characteristics of openness and dynamicity, functions—as well as their connections to each other—may vary or change over time. These possible changes in individual functions may affect the overall function. Therefore, the possible inputs and outputs of functions—on different abstraction layers—that vary over time must be considered in the modeling process.

- ❑ **Req.4:** Functions transform input into output to achieve a goal or meet a requirement. It must be possible to model the relationship between a goal (or a requirement) and its possible solution provided by a function. This modeling must consider priorities of goals and functions as well as conflicts between them. This modeling must also include dependencies between different functions.
- ❑ **Req.5:** In case of a failure or of an error in individual functions, compensation strategies are necessary. Possible functional errors or failures must be considered in modeling to make them detectable. Therefore, relationships between the function model and other system models must be considered.
- ❑ **Req.6:** Since different CSG functions are realized by different systems, potentially from different manufacturers, modeling approaches must ensure that inputs and outputs of different functions are compatible and suitable/consistent with each other.
- ❑ **Req.7:** CESs must provide the functionality to allow the CSG to be restructured at runtime. This leads to functions being related to different states. These states must represent, for example, when a function is accessible and when the transformation from input to output is not available.

*Metamodel
development*

We subsequently developed a metamodel iteratively to satisfy these requirements. We conducted workshops with the stakeholders to negotiate and re-iterate the metamodel as long as necessary to achieve a final, agreed version that fits all purposes for functional modeling and analysis of CESs.

The following is an example of the analysis of the individual methods investigated and the resulting rationale for the relevance of the method for the function metamodel. For demonstration purposes, we use a method for modeling the goals of CESs and CSGs.

Example 4-2: Functional aspects of the goal modeling method

Method name: GoalBasedSystemGroupEngineering

Metamodel extension required: Consider the relationship between goals and functions.

Reason: This method focusses on the definition of goals for the individual CESs as well as for the CSG. In the relationship between the CSG goals and the CES goals, it must be ensured that every CSG goal can be operationalized. Therefore, goals are refined into tasks, which represent abstract definitions of functions to be implemented.

4.3 Background

Our work builds on results from the SPES projects that provide a framework that enables seamless model-based engineering of embedded systems (cf. [Pohl et al. 2016]). The SPES modeling framework includes a *functional viewpoint* for specifying the system's functionality. The system's functionality is elicited from the requirements in a preceding requirements viewpoint. Our metamodel builds on this background work.

*SPES modeling
framework*

To a large extent, the SPES modeling framework is based on a formal theory called FOCUS, which provides models and formalisms for specifications and the development of distributed interactive and dynamic systems. FOCUS establishes a formal semantics that serves as a common ground for also giving means to functional behavior. In FOCUS (cf. [Broy and Stølen 2012], and [Broy 2014]), a system's interface is determined by the system's boundary. The *syntactic interface* describes the set of input and output channels and the message types that these channels transport across the system's boundary. The system's functionality is described by the *interface behavior*, which can be observed at the system's boundary and which is defined by the history of streams of messages across the input and output channels. The histories of the streams of messages across the input and output channels capture the system's interface behavior. Accordingly, the interface behavior models system functionality that can be observed.

FOCUS theory

4.4 Metamodel for Functions of CESs and CSGs

The metamodel for functions is given in [Figure 4-3](#), which shows the aspects to be considered when modeling functions of CESs and CSGs. Based on the requirements, we have identified five major aspects that need to be considered when modeling functions for CESs and CSGs. These aspects are detailed in the subsections below:

Five major aspects

- ❑ First, a differentiation between individual collaborating systems, not collaborating systems, and collaborative system groups is necessary. The CESs can partake in a CSG to fulfill their purposes as well as to contribute to the goals of the CSG. Whether a function belongs to either a CES or the CSG influences whether that function exists on its own or only in the collaborative interplay.

- ❑ Second, the term function must be defined, thereby placing a particular emphasis on its behavior and its interfaces. It is important to identify how these aspects relate to the function's contribution to the collaboration of the CSG.
- ❑ Third, as collaborative systems are inherently goal-oriented, system goals must be considered. System goals represent the established realization of stakeholder goals, which are elicited during requirements engineering. This means that a CES takes part in a collaboration only if this fulfills a certain purpose (i.e., the system goal), depending on the needs of the CSG and the other CESs. The systems offer different functions to optimize goal fulfillment of the individual CESs as well as the overall CSG.
- ❑ Fourth, roles play a vital part in the engineering of collaborative systems, as the functions a system offers and requires depend on the role the system takes in a certain collaboration.
- ❑ Fifth, context and adaptivity must be considered. CESs and CSGs operate in an open, dynamic context. The dynamicity of the operational context is the main trigger for the adaptation of the entire CSG, which might result in reconfiguration of the individual CESs and thus impact their functional interplay.

4.4.1 Systems, CESs, and CSGs

Collaborating and not collaborating systems

In this section, we introduce the relationship between a system and the collaboration of systems. Therefore, we start—as in traditional system analysis—by separating a *system* and its *context*. For a system, we must distinguish whether it is a *not collaborating system*, a *collaborating system*, or a *CSG*, always viewed at a certain point in time.

Belonging to a certain CSG depends on time

A not collaborating system does not collaborate with other systems in a given CSG at a current time t . We can distinguish between *CESs* and *non-CESs*. While a non-CES cannot collaborate in a CSG at any time², a CES can become a collaborating system for a CSG at a later point in time during the runtime of the system. A collaborating system is part of a CSG, which consists of multiple collaborating systems. Within a CSG, the CESs work together, provide their functions to each other, and share information to promote common CSG goals.

---

² Note: CESs and non-CESs are always related to a specific type of CSG: a system can be collaborating with respect to a given type of CSG and a non-CES for another type of CSG.

Fig. 4-3: Metamodel for functions of CESS and CSGs [Hayward et al. 2020]

Functional architecture Every system (i.e., each CES, each CSG, each non-CES) has a *functional architecture* that contains all the functions of the system and describes how the functions interact with each other to achieve *goals*. This means that each individual CES consists of a functional architecture (cf. [Pohl et al. 2016]) which is therefore part of the larger functional architecture of the CSG.

4.4.2 Functions

Functions have interfaces Systems can be described on different levels of detail by their *functions* [VDI 2221]. Functions describe the behavior of a system through the interrelation between input and output variables [VDI 2222]. A function has a *syntactic interface*, through which it can take up information, material, or energy, transform it and output it again. Depending on the domain, the understanding of the term function can vary slightly in detail but this definition is valid at this general level [Eisenbart et al. 2012].

Functional decomposition In the classical design methodology according to [Pahl et al. 2013] as well as in today's model-based system development [Vogelsang 2015], [Meissner et al. 2014], functions are derived from requirements lists and models during an early design phase to capture the required functionality of the CES. Since specific solution principles can be derived only to a very limited extent based on these abstract functions, the functions are further decomposed into sub-functions, which can also be further subdivided, thus forming a hierarchy. These sub-functions again have interfaces through which they are connected. The functional hierarchy is called *functional architecture*. The functions and the resulting functional architecture can be used to describe what a system should be able to do. Additionally, the *interface behavior* can be used to describe which states, state transitions, and functional dependencies functions have [Eisenbart et al. 2012].

Functions and collaboration A CES no longer performs certain functions alone; it performs them in collaboration with other CESs. For this purpose, the CESs form a CSG and thereby provide their functions (*CES function*) to each other to achieve a common goal. The CSG can be considered as a system on its own again with components and functions (*CSG function*). These functions of the CSG are realized by CES functions and result from the interplay between the collaborating systems. While the functions of individual CESs can be modeled and realized during design time, the functions of the CSG are only constituted through collaboration between several CESs at runtime. By modeling CSG functions, we can

indicate the specific contributions involved that CESs have to provide to achieve common goals of the CSG.

In addition to these two specializations of the function concept, the metamodel further distinguishes between *system function* and *collaboration function*. A system function contains the individual contribution of a CES within the collaboration. In a broader sense, a system function also includes other internal functions of the CES that contribute, albeit indirectly, to the fulfillment of goals. Collaboration functions, on the other hand, comprise a set of functions that are assumed to be available in all CESs participating in the CSG to enable collaboration in general and independently of the specific form and goal of the CSG. These collaboration functions include functions for the perception of and communication with other CESs, the negotiation of goals, the comparison of required and existing functions, or the adaptation of the CESs' behavior to meet the conditions of the CSG.

*System function and
collaboration function*

4.4.3 Goal Contribution and Fulfillment

Systems have *goals* associated with them. A goal is thus defined as a condition or situation the system wants to achieve or a behavior the system wants to exhibit. This holds for an individual CES as well as for a CSG. As mentioned in the previous sections, the fulfillment of these goals is always realized through functions and their implementation with the help of algorithms. This explicit manifestation of the stakeholder goals enables systems to fulfill the goals planned at the time of development during operation. We have to consider several situations according to system goals.

Goals

The goals of the CESs concerned may differ from each other and from the goals of the CSG and may even be conflicting goals. In the case of (partly) different or contradictory goals, and in order to form a functioning CSG, the following must be negotiated: different goals between CESs collaborating in the CSG, differences between CES and CSG goals, and the way each CES contributes to the achievement of the CSG goals. Finally, the individual goals must be adopted in order to reach a consistent goal system within the CSG and its collaborating systems. As a consequence, it must be possible for the CESs to change their goals according to the results of the negotiations. In this sense, goals are considered dynamic at runtime.

Negotiating goals

We differentiate between goals that will never be changed (*hard goals*) and goals that may be changed (*soft goals*) in order for a CES to

*Hard goals and soft
goals*

actively contribute to the CSG goal system³ in a consistent manner. To this end, we have extended the function metamodel from SPES to match these additional requirements.

4.4.4 Roles

*Roles enable assignment
of responsibilities*

Another concept that supports modeling and implementation of CSGs is that of *roles*. Within a CSG, different functions are needed to achieve the CSG goals. Roles can be used to define, within the CSG, which collaborating system is responsible for which CES functions and thus for which goals. CESs assume roles when they join the CSG. A single CES can potentially assume one or more roles within a CSG at the same time. The roles allow the definition of the necessary CES functions and thus the necessary behavior of the individual collaborating systems [Weiß et al. 2019, Regnat et al. 2019].

*Potential roles and
current role*

A CES that has assumed a certain role within the CSG (*current role*) is responsible for the role-related functions. If a CES leaves a CSG (e.g., intentionally or due to an error), but the functions associated with its role must still be provided, it may be necessary for another CES, which has the necessary functions, to change its current role. This *role change* is only possible if the functions of this subsequent CES allow (*potential role*), it to assume the role from the leaving CES. These processes are only possible if the CESs involved in the CSG have a common understanding of the roles to be assumed.

4.4.5 Context and Adaptivity

System boundary

A system is separated from its context and other systems by its *system boundary*. The system boundary defines whether an object belongs to the system or is outside of it. However, because of the dynamicity of the CSG, the boundary, the behavior, and the structure of the CSG may change over time. Consequently, we had to extend the metamodel to cope with such situations — namely, a potential CES that is outside the CSG at a given point in time may enter the CSG and therefore become part of the CSG structure (i.e., will be inside the CSG boundary).

Context

The *context* of the system describes the environmental surrounding that is not part of the system. The surrounding includes

---

³ Note: The terms hard goal and soft goal are used differently here as compared to goal modeling literature: hard goals are not subject to negotiation and are in a sense “static,” while soft goals may be dynamically negotiated and changed in order to cooperate in a CSG.

persons, groups, organizations, processes, events, documents, functions of other systems, etc. In other words, the context is a perceivable part of the environment that consists of all the objects relevant to the system. Context is everything that is relevant to a system but remains external to its range of action [Lalanda et al. 2013]. Separating the system and its context means distinguishing between changeable and unchangeable variables [Pohl 2010]. Consequently, the context consists of all objects relevant to the system but outside the system's boundary.

Modern systems, such as CESs and CSGs, operate in a changing, uncertain, and dynamic context. In addition to the dynamicity of the context in which the systems operate, the structure of the system itself is also dynamic. Consequently, as briefly explained earlier in this section and in Section 4.4.1, the structure and the behavior of the system fluctuate over time — namely, a CES that is not collaborating at time x might become a collaborating system at a definite time $t + x$ in the future by joining a specific CSG; and vice versa, a collaborating system of a CSG might, over time, leave the group. This directly impacts and changes the boundary of a system, which is no longer static, changes at runtime, and goes beyond what was defined during the system's design.

Context and system boundaries change

Consequently, the systems must be able to adapt in order to deal with the dynamicity and the runtime changes that might originate externally from the context in which the system operates, as well as internally from the system itself. We refer to these uncertainties and changes that trigger the adaptivity as *trigger events*. What we distinguish as internal or external events depends specifically on whether the system under consideration is a CES or a CSG. From the point of view of an entire CSG, internal trigger events could refer to, for example, the changes in CESs that collaborate, as role changes. In contrast, an external trigger event could be a CES from the context requesting to join and share its functionality with the CSG. However, from the view of an individual CES, the changing of a role, which is an internal trigger from the perspective of the CSG, can be considered an external trigger for the CES. In contrast, an example of internal triggers for the CESs are sensor uncertainties, such as sensor ambiguity, sensor imprecision, or even complete sensor failure, which could potentially lead the complete CES to a non-deterministic or faulty behavior.

Trigger events

In a nutshell, the general idea behind adaptivity is the ability to change the system's observable behavior, structure, and realization [Broy 2017], [Krupitzer et al. 2015], [Giese et al 2013] as a response

Adaptivity

to the internal and external events in order for the systems to continue meeting their functional specifications while preserving the performance (or another quality objective) — despite all the changes that the system may encounter during runtime [Petrovska and Pretschner 2019].

Adaptation logic

The adaptivity is enabled by the *adaptation logic*, which is a necessary precondition for a system to adapt to these changing situations. If the adaptation is triggered manually by an external user or administrator (a human assumes the role of an adaptation logic), then this is referred to as a system reconfiguration. In contrast, if the adaptation is triggered and executed by the system itself, in an automated manner, without any user interaction, then we call it self-adaptation [Petrovska et al. 2020]. Specifically, in our metamodel, we consider the adaptation logic to be a collaboration function which adapts the functions and the behavior of the CESs and the CSG through the collaboration of the systems.

4.5 Evaluation of the Metamodel

In this section, we will briefly outline how the proposed function metamodel fulfills the requirements from Section 4.2. Further evaluation is subsequently given in Section 4.6 by showing the applicability of the proposed function metamodel.

4.5.1 Abstraction

Requirements 4-4: Req.1

It must be possible to model functions on different abstraction layers. Composition and decomposition of functions must consider the relationship between single functions of CESs and overall functions on the CSG layer. On the highest layer of abstraction, a function can be understood as the emerging result of individual contributing functions. On the lowest abstraction layer, it must be possible to model an atomic function and its contribution to the overall function.

This requirement is fulfilled because a *function* is composed of other functions, thereby allowing the description of functionality at different levels of granularity. Furthermore, the separation between *CSG function* and *CES function* introduces another abstraction layer, as a function belongs either primarily to the overall CSG or to an individual CES. However, in both cases, the functions must be

implemented in a CES, as the CSG relies on the CESs for any kind of resource. In addition, the distinction between *collaboration function* and *system function* also indicates different levels of granularity to describe functional properties.

4.5.2 Relationships between Functions

Requirements 4-5: Req.2

It must be possible to model which overall function of a CSG an individual function of a CES can contribute to and how it can do so. For this purpose, the modeling process must consider whether different inputs and outputs of functions can be connected with those of other functions with regard to compatibility.

The aforementioned differentiation between *CSG function* and *CES function* allows us to define which CSG function *is realized by* which CES functions, and which CES function *realizes* which CSG functions.

4.5.3 Openness and Dynamicity

Requirements 4-6: Req.3

Due to characteristics of openness and dynamicity, functions—as well as their connections to each other—may vary or change over time. These possible changes in individual functions may affect the overall function. Therefore, the possible inputs and outputs of functions—on different abstraction layers—that vary over time must be considered in the modeling process.

To address this requirement, the *adaptation logic* reacts to *trigger events* in the *context* and *adapts* the *behavior* of a *function*. A change in an individual function also affects other functions of the CES or the CSG (see Sections 4.5.1 and 4.5.2). In particular, the composition of the *functional architecture* of any kind of *system* may be changed.

4.5.4 Goal Contributions

Requirements 4-7: Req.4

Functions transform input into output to achieve a goal or meet a requirement. It must be possible to model the relationship between a goal (or a requirement) and its possible solution provided by a function. This modeling must consider priorities of goals and functions as well as conflicts between them. This modeling must also include dependencies between different functions.

A *goal* is defined as either a *hard goal* or a *soft goal*; each can be decomposed and be related to each other. Each *goal* is implemented by at least one *system function*, while any *function* can contribute to any goal. In addition, a *collaboration function* may change a *soft goal*.

4.5.5 Relationships Between Functions and Systems

Requirements 4-8: Req.5

In case of a failure or of an error in individual functions, compensation strategies are necessary. Possible functional errors or failures must be considered in modeling to make them detectable. Therefore, relationships between the function model and other system models must be considered.

As mentioned earlier, *functions* and *systems* can be directly related by means of the *functional architecture*. Furthermore, the metamodel differentiates between the *CSG* and *CSG functions*, and between *CESs* and *CES functions*.

4.5.6 Input/Output Compatibility

Requirements 4-9: Req.6

Since different CSG functions are realized by different systems, potentially from different manufacturers, modeling approaches must ensure that inputs and outputs of different functions are compatible and suitable/consistent with each other.

Each *function* is defined by its *behavior* and its *interface*. This allows us to check the compatibility of functions. Furthermore, as outlined above, sophisticated relationships between functions, systems, CES functions, and CSG functions can be defined.

4.5.7 Runtime Restructuring

Requirements 4-10: Req.7

CEs must provide the functionality to allow the CSG to be restructured at runtime. This leads to functions being related to different states. These states must represent, for example, when a function is accessible and when the transformation from input to output is not available.

The *adaptation logic* allows restructuring functions and functional architectures by *adapting* the *behavior* of individual functions (see Req.3). This leads to different states of a function being associated with different situations and compositions of the CSG. Furthermore, the concept of *roles* allows restructuring of the CSG by means of the assignment of roles involved. Therefore, individual CEs conduct *role changes*, which influence the function and thereby again restructure the overall functional architecture.

4.6 Application of the Metamodel

In this section, we demonstrate the applicability of the metamodel using examples from the two use cases of the adaptable and flexible factory and autonomous transport robots.

4.6.1 Example from the Adaptable and Flexible Factory

For the adaptable and flexible factory, let us consider the scenario of order-driven production. In this scenario, there are several heterogeneous modules within a factory. These modules are equipped with different functions and can contribute to the production of products. Exemplary functions are drilling, milling, or even turning of materials. In addition, assembly operations can be used to assemble different workpieces or to execute optical quality checks. Depending on the product to be manufactured for a customer, different functions, and thus contributions from different modules are required. Those modules that can contribute to the production form a CSG in which they provide their functions to each other in order to achieve the overall goal of fulfilling the production order. At the end of the production, the modules leave the CSG again.

Both centralized and decentralized coordination paradigms are conceivable for this CSG. In the case of centralized coordination, there is a single module that, as coordinator, decides after receiving a

*Centralized and
decentralized
coordination*

production order which module is supposed to contribute to the production with which functions. In the case of decentralized coordination, the modules autonomously negotiate their possible contributions to the fulfillment of the order. Other mixed forms of centralized and decentralized coordination are also possible.

Regardless of the process of forming the CSG, the modules can be considered as CESs in accordance with the metamodel before the production starts and thus before the CSG is formed. By forming a CSG, these CESs become collaborating systems and, depending on their functional properties, assume one (or even several) roles in the CSG. Roles to be filled in the factory are, for example, material processing, assembly, transport, quality inspection and, in the case of centralized coordination, the coordinator. The required product can only be manufactured once all the roles required for an order have been assigned to the modules forming the CSG.

Modules assume roles

A module can only assume a role if it has the necessary functions. When all necessary roles have been assigned to modules, the CSG functions that are required to fulfill the CSG goals can be executed. An exemplary CSG function in the factory is the manufacturing of the product. The CSG function for manufacturing the product can only be executed, and thus the goal of fulfilling the customer order achieved, through the individual CES functions of the modules. Further exemplary CSG functions are the definition of the production sequence and the calculation of the production time. These CSG functions can also only be realized by aggregating the CES functions of the modules.

*Separation between
system function and
collaboration function*

The metamodel also shows a separation between system function and collaboration function. The system function represents the individual contribution of a CES to a CSG. A system function of a factory module can be drilling, milling, transport, or assembly, for example.

The collaboration functions enable the modules to communicate with each other, to exchange information about production orders, and to coordinate their contributions in the CSG. To coordinate the contribution of a module, the requirements of the products to be manufactured resulting from the orders must be compared with the available functions of the modules. In other words, a check is required to determine whether the functions of the modules are suitable to contribute to the production of the order. Such matching is also part of the collaboration functions.

In a centrally organized factory, the task of matching the requirements of the order with available functions is the responsibility of the coordinator. In this scenario, each module that should be involved in possible production orders must inform the coordinator of its available system functions and provide appropriate descriptions of the scope of these functions.

The coordinator matches requirements and functions

In a decentralized factory, where the modules coordinate with each other without a central coordinator, each module must be able to check whether it can contribute to the production and must be able to communicate the result of this check to the other modules. The other modules must be able to understand this contribution and compare it with their contributions. This is the prerequisite for determining whether the contributions delivered in total (the resulting CSG functions) are sufficient to produce the product.

Self-check contribution possibilities

Receiving a new order within the factory can be seen as a trigger event from the context, which means that the modules have to adapt their behavior. The execution of this adaptation is enabled by the adaptation logic in the metamodel. A single module can adapt its behavior by using the adaptation logic. Such an adaptation can, for example, be that a module changes its current configuration and thus its executable functions. Depending on the specific module, this reconfiguration can be done automatically by the module itself or partially automated with the support of a worker in the factory.

4.6.2 Modeling of Goals for Transport Robots

Another example which helps to improve the understanding of the elements of the metamodel is based on the use case of transport robots. This example looks at several transport robots (i.e., CESs) within a factory, with each robot being responsible for transporting different materials (i.e., an overall CSG). In order to receive different materials as input for various products, the individual transport robots connect to modules and conveyor belts, which allows the transport robots to take part in multiple production processes at once. The main purpose (i.e., the system goals) of the transport robots consists of executing the production logistics and ensuring punctual delivery and pick-up of materials between production process sites, for which the transport robots provide several functions.

Autonomous transport robots operate in a factory

In order to optimize the transport of goods within the factory from a logistical point of view, individual transport robots must negotiate possible orders and distribute them jointly. To enable this negotiation and to coordinate further behavior, the transport robots must

Negotiate orders and coordinate behavior

Figure 4-11 shows an excerpt of the goal model for the CTRF. When applying GRL, these tasks can be divided into further tasks to allow a more detailed specification. In terms of the metamodel, these tasks can be considered as functions. The individual tasks of the CTR presented here correspond to system functions in the metamodel. The functions for communication between the CTRs within the CTRF that are not shown here correspond to the collaboration functions.

GRL goal modeling

The CTR pursues the goal to *optimize their current goods transportation*. The goal can be fulfilled when the CTR performs the different tasks shown. The CTRF pursues the goal to *optimize the goods transportation* of all participating CTRs. As these goals are interdependent, they are linked in the goal model by a bidirectional dependency (shown by the two *Ds* on the connecting line). The task *optimal order acceptance decision* has some positive influence on the goal *optimal current resource usage* and is therefore displayed as a contribution arrow marked with the plus icon. While this refers mainly to the goal part of the metamodel, the relation to functions is made clear as the tasks define what functions need to be implemented to fulfill which goals.

Differentiate between goals and tasks

4.7 Related Work

A comparative literature review was conducted by [Erden et al. 2008] to investigate different function modeling approaches and their similarities and differences. For example, functional model ontologies [Chandrasekaran and Josephson 2000], [Umeda et al. 1996], [Umeda et al. 1995], and [Yoshioka et al. 2004] aim at developing frameworks and languages for modeling the functionality of a system from the different viewpoints [Erden et al. 2008]. None of the proposed functional model ontologies consider the modeling of functions of complex CESs and CSGs, including the ramifications of the contexts in which these systems operate.

In [Chandrasekaran and Josephson 2000], the authors define two function viewpoints: “environment-centric viewpoint” and “device-centric viewpoint.” These viewpoints correspond to the collaboration and system functions proposed in our work respectively. In the first viewpoint, the function is related to the external effects that an object or a system has on its environment. In contrast, in the second viewpoint, functions are related to the internal features and parameters of the system. In our metamodel, to a certain extent we subsume both the viewpoints proposed in [Chandrasekaran and

Josephson 2000]: 1) considering systems' functions that have effects on their environments, specifically in our case the context as the relevant part of the environment; 2) as well as the other systems involved in a collaboration, including their internal system parameters, states, and behaviors. Furthermore, [Gero 1990] has developed a function-behavior-structure model. In his model, he considered a function as an intermediate step between the behavior of the system and the user's goal.

A few frameworks have been proposed in literature to define a well-formed functional behavior of the system systematically. FOCUS (cf. [Broy and Stølen 2012], and [Broy 2014]), previously explained in Section 4.3, is an instance of such a formal framework that provides models and formalisms for specifications and development of distributed interactive and dynamic systems. In our contribution, according to FOCUS, we define the behavior of a function as a stream of messages across its input and output channels, which through its interfaces, take up information, material, or energy, and transform it before outputting it.

To the best of our knowledge, there has been no previous work on modeling functions for CESs from multi-dimensional aspects as proposed in our metamodel, including the dynamicity of the contexts of the system, role and goal modeling, and complex properties of these systems such as collaboration and adaptivity. The domain-independent metamodel proposed in this paper closes this gap.

4.8 Conclusion

The new challenges in the model-based development of embedded systems arising from collaboration make it necessary to adapt and extend existing modeling languages. In this chapter, we showed the aspects to be considered in the modeling of functions for CESs and CSGs in a metamodel. We then evaluated this metamodel and illustrated it using two examples from the use cases of the adaptable and flexible factory and autonomous transport robots. Based on the metamodel, specific extensions of modeling languages can be executed. Depending on domain-specific requirements, methods for the application of these extended modeling languages can be developed. The use case examples presented in this chapter will be used as a basis for further research.

4.9 Literature

- [Brings et al. 2019] J. Brings, M. Daun, T. Bandyszak, V. Stricker, T. Weyer, E. Mirzaei, M. Neumann, J. S. Zernickel: Model-Based Documentation of Dynamicity Constraints for Collaborative Cyber-Physical System Architectures: Findings from an Industrial Case Study. In: *Journal of Systems Architecture - Embedded Systems Design*, vol. 97, 2019, pp. 153–167, DOI: 10.1016/j.sysarc.2019.02.012.
- [Brings et al. 2020] J. Brings, M. Daun, T. Weyer, K. Pohl: Goal-Based Configuration Analysis for Networks of Collaborative Cyber-Physical Systems. In: *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, Brno, Czech Republic, Mar. 2020, pp. 1387–1396, DOI: 10.1145/3341105.3374011.
- [Broy and Stølen 2012] M. Broy, K. Stølen: *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer Science & Business Media, 2012.
- [Broy 2014] M. Broy: A Model of Dynamic Systems. In: *From Programs to Systems. The Systems Perspective in Computing*, pp. 39-53. Springer, Berlin, Heidelberg, 2014.
- [Broy 2017] M. Broy: *Formalizing Adaptivity, Dynamics, Context-Awareness, Autonomy*, 2017, white paper.
- [Chandrasekaran and Josephson 2000] B. Chandrasekaran, John R. Josephson: Function in Device Representation. In: *Engineering with Computers* 16.3-4: pp.162-177, 2000.
- [Daun et al. 2019a] M. Daun, T. Weyer, K. Pohl: Improving Manual Reviews in Function-Centered Engineering of Embedded Systems Using a Dedicated Review Model. In: *Software and Systems Modeling*, vol. 18, no. 6, 2019, pp. 3421–3459, DOI: 10.1007/s10270-019-00723-2.
- [Daun et al. 2019b] M. Daun, V. Stenkova, L. Krajinski, J. Brings, T. Bandyszak, T. Weyer: Goal Modeling for Collaborative Groups of Cyber-Physical Systems with GRL: Reflections on Applicability and Limitations Based on Two Studies Conducted in Industry. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC 2019, Limassol, Cyprus, April 8-12, 2019, 2019, pp. 1600–1609, DOI: 10.1145/3297280.3297436.
- [Erden et al. 2008] M.S. Erden, H. Komoto, T. J. van Beek, V. D'Amelio, E. Echavarria, T. Tomiyama: A Review of Function Modeling: Approaches and Applications. *AI Edam* 22, no. 2, 2008, pp. 147-169.
- [Eisenbart et al. 2012] B. Eisenbart, L. Blessing, K. Gericke: *Functional Modelling Perspectives Across Disciplines: A Literature Review*. International Design Conference - Design 2012, Dubrovnik, Croatia, 2012.
- [Gero 1990] J. S. Gero: Design Prototypes: A Knowledge Representation Schema for Design. *AI magazine* 11.4, 1990, pp. 26-26.

- [Giese et al. 2013] H. Giese, H. A. Müller, M. Shaw, R. De Lemos: Software Engineering for Self-Adaptive Systems II. Springer, Berlin, 2013.
- [Hayward et al. 2020] A. Hayward, M. Daun, W. Böhm, A. Petrovska, L. Krajinski, A. Fay: Modellierung von Funktionen in der modellbasierten Entwicklung von Systemverbünden kollaborierender cyber-physischer Systeme, Tagung: Entwurf komplexer Automatisierungssysteme (EKA), Magdeburg, 2020, (available in German only).
- [Krupitzer et al. 2015] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, C. Becker: A Survey on Engineering Approaches for Self-Adaptive Systems. *Pervasive and Mobile Computing* 17, 2015, pp. 184-206.
- [Lalanda et al. 2013] P. Lalanda, J. A. McCann, A. Diaconescu: *Autonomic Computing: Principles, Design and Implementation*. Springer Science & Business Media, 2013.
- [Ludewig et al. 2018] A. Ludewig, M. Daun, A. Petrovska, W. Böhm, and A. Fay: Requirements for Modeling Dynamic Function Networks for Collaborative Embedded Systems. In: *Joint Proceedings of the Workshops at Modellierung 2018 co-located with Modellierung 2018*, Braunschweig, Germany, 2018, vol. 2060, pp. 79–89.
- [Meissner et al. 2014] H. Meissner, M. Cadet, N. Stephan, C. Bohr: Model-Based Development Process of Cybertronic Products and Production Systems. In: *Advanced Materials Research*, Vol. 1018, 2014, pp. 539–546.
- [Pahl et al. 2013] G. Pahl, W. Beitz, J. Feldhusen, K.H. Grote: *Methoden und Anwendung erfolgreicher Produktentwicklung*. 8. Aufl., Heidelberg: Springer. 2013 (available in German only).
- [Petrovska and Pretschner 2019] A. Petrovska, A. Pretschner: Learning Approach for Smart Self-Adaptive Cyber-Physical Systems. *IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pp. 234-236. IEEE, 2019.
- [Pohl 2010] K. Pohl: *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer Publishing Company, Incorporated, 2010.
- [Pohl et al. 2012] K. Pohl, H. Hönniger, R. Achatz, M. Broy, Eds., *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*. Berlin Heidelberg: Springer-Verlag, 2012.
- [Pohl et al. 2016] K. Pohl, M. Broy, H. Daembkes, H. Hönniger: *Advanced Model-Based Engineering of Embedded Systems*, Springer, Cham, 2016.
- [Regnat et al. 2019] Regnat, N. et. al.: Seamless Model-Based Approach - MQ1.AP2.D4. Published by Subproject MQ1 in CrESt. Internal Project Deliverable of CrESt, 2019.
- [Umeda et al. 1995] Y. Umeda, T. Tomiyama: FBS Modeling: Modeling Scheme of Function for Conceptual Design. *Proc. of the 9th Int. Workshop on Qualitative Reasoning*, 1995.

- [Umeda et al. 1996] Y. Umeda, I. Masaki, Y. Masaharu, Y. Shimomura, T. Tomiyama: Supporting Conceptual Design Based on the Function-Behavior-State Modeler. *Ai Edam* 10, no. 4: 275-288, 1996.
- [VDI 2221] VDI Guideline 2221: Systematic Approach to the Development and Design of Technical Systems and Products, 1993.
- [VDI 2222] VDI Guideline 2222 Blatt 1:1997-06: Methodic Development of Solution Principles, 1997.
- [Vogelsang 2015] A. Vogelsang: Model-Based Requirements Engineering for Multifunctional Systems. Dissertation. Technische Universität München, Institut für Informatik, 2015.
- [Weiß et al. 2019] S. Weiß et. al.: Modeling of Dynamics in the Open Context of Collaborative Embedded Systems – EC4.AP2.D3. Published by Subproject EC4 in CrEST. Internal Project Deliverable of CrEST, 2019.
- [Yoshioka et al. 2004] M. Yoshioka, Y. Umeda, H. Takeda, Y. Shimomura, Y. Nomaguchi, T. Tomiyama: Physical Concept Ontology for the Knowledge Intensive Engineering Framework. *Advanced engineering informatics*, 18(2), 2004, pp.95-113.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.