[BBD+21] B. Böhm, W. Böhm, M. Daun, A. Hayward, S. Kranz, N. Regnat, S. Schröck, I. Stierand, A. Vogelsang, J. Vollmar, S. Voss, T. Weyer, A. Wortmann: Engineering of Collaborative Embedded Systems.
In: Model-Based Engineering of Collaborative Embedded Systems, pp. 15–48, Springer, Jan. 2021.
www.se-rwth.de/publications/

Birthe Böhm, Siemens AG
Wolfgang Böhm, Technical University of Munich
Marian Daun, University of Duisburg-Essen
Alexander Hayward, Helmut-Schmidt-Universität
Sieglinde Kranz, Siemens AG
Nikolaus Regnat, Siemens AG
Sebastian Schröck, Robert Bosch GmbH
Ingo Stierand, OFFIS e.V.
Andreas Vogelsang, Technische Universität Berlin
Jan Vollmar, Siemens AG
Sebastian Voss, fortiss GmbH
Thorsten Weyer, University of Duisburg-Essen
Andreas Wortmann, RWTH Aachen University

2

Engineering of Collaborative Embedded Systems

Embedded systems are being increasingly used in changing environments where they no longer fulfill their associated stakeholder goals on their own, but rather in interaction with other embedded systems. This transition to networked, collaborative embedded systems is creating new application opportunities that impose numerous challenges for developers of these systems. In this introductory chapter of the book, we present the complexity of these systems and the challenges associated with them in a coherent manner. We illustrate the challenges using two use cases, "Vehicle Platooning" and "Adaptable and Flexible Factory." Finally, we reference the challenges of developing collaborative embedded systems to the individual chapters of this book, which describe various methods of mastering the complexity in more detail.

W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_2

2.1 Introduction

New class of systems

With the transition from classical embedded systems to networked, collaborative embedded systems (CESs), new applications for industry are emerging. The ability of a company to efficiently develop CESs of the highest quality will therefore become a decisive competitive factor. At the same time, this transition will lead to a leap in the complexity of the systems under consideration. Not only single embedded systems, but also dynamically changing networks of CESs at runtime have to be considered. Since the success of products in the area of embedded systems is strongly determined by their quality, it must be possible to guarantee a high system quality despite the increasing complexity. Therefore, it is essential to be able to control the complexity of CESs with efficient methods. This includes suitable methods for specification, implementation, and validation of these systems. The development of CESs goes hand in hand with important safety and security issues, which have to be addressed comprehensively for a broad industrial application by relevant development approaches.

This chapter gives an informal introduction to the challenges of developing CESs. We start with the definition of important terms and then describe the challenges that have to be overcome in the development of such systems. These challenges are explained in more detail by means of two use cases. Finally, at a high level of abstraction, we provide an overview of selected results achieved in the CrESt project[1]. Details of the results can be found in the corresponding contributions of this book (see Section 2.6 and the Appendix).

2.2 Background

Model-based systems engineering

Model-based systems engineering (MBSE) [Selic 2003] aims to reduce the conceptual gap between problem domains (mechanical engineering, automation, biology, law, etc.) and the solution in software [France and Rumpe 2007], and to integrate contributions from the participating domains. For this purpose, models—often in the terminology of problem domains—are used as documentation. Furthermore, development artifacts that reduce this gap with an explicit description of problem domain concepts can be accessed with

[1] Website of the CrESt project: https://crest.in.tum.de/ (available in German only)

sufficient formalization of efficient automation. These artifacts also simplify the integration of contributions from different domain experts by abstracting solution domain details.

In the German Federal Ministry of Education and Research (BMBF) project SPES2020 [Pohl et al. 2012] and the follow-up project SPES_XT [Pohl et al. 2016], significant results for MBSE have been achieved that further advance the development of highly automated embedded systems and have already established themselves as a methodological approach in industry.

The two SPES projects provide a methodological toolkit for MBSE that allows an efficient model-based development of embedded systems. At the same time, the toolkit is based on a solid scientific foundation with a special focus on consistency and semantic coherence (see [Broy and Stolen 2001], [Broy 2010]). The SPES methodology building set is based on three principles of outstanding importance:

Principles of the SPES methodology

- ❑ Consistent consideration of interfaces along the design process
- ❑ Decomposition of the interface behavior and the description of systems via subsystems and components at different levels of granularity
- ❑ Definition of models based on the previous points for a variety of cross-sectional topics (variability, safety, etc.) and analysis options

In SPES, a system model is a conceptual ("generic") model for describing systems and their properties. It describes what constitutes a system as the result of a conceptualization. System models define the components of the system and its structure, the essential properties, and other aspects that have to be considered during development. Among other things, system models define what requirements refer to (subject of discourse). In SPES, the system model consists of (see Figure 2-1):

System model

- ❑ An *operational context*[2] that influences or is influenced by the system at runtime
- ❑ An *interface* that clearly separates the system from its operational context
- ❑ A *behavior* of the system that can be observed at the interface (indicated by arrows at the interface)

[2] SPES distinguishes between the operational context, which in turn consists of the structural, functional, and behavioral context, and the knowledge context (see e.g., [Pohl et al. 2016]). In this chapter, however, only the operational context is relevant.

- An *inner structure* of interrelated and communicating elements (architecture), which are themselves systems

The system model used in SPES is static in the sense that model elements do not change at runtime. This applies in particular to the appearance and disappearance of elements in the operational context as well as the adaptation of the system interface at runtime. With the transition from classical embedded systems as considered in SPES2020 and SPES_XT to networked CESs, new applications for the MBSE approach arise. At the same time, this transition leads to a leap in the complexity of the systems under consideration. Not only single embedded systems but also networks of CESs have to be considered. Such system networks can be constituted dynamically at runtime by a multitude of different embedded systems (homogeneous or heterogeneous type) [Grosz 1996]. In their interaction, these system networks enable the users to achieve a comprehensive added value that goes beyond the benefits of the individual systems. In such systems, both the exact system configuration (i.e., the system boundary) and the system context at design time can only be anticipated with considerable uncertainty. In the context of the development of CESs, this raises new and important questions

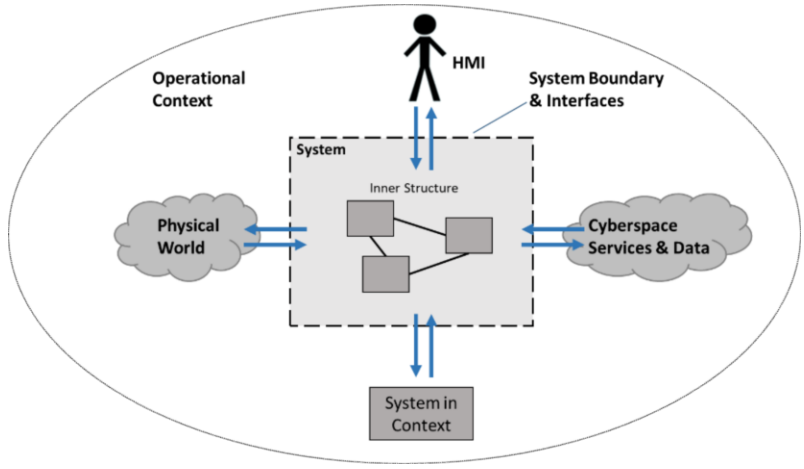

***Fig. 2-1**: SPES system model*

regarding the functional safety of the systems and the dynamically formed system networks [Damm and Vincentelli 2015], [SafeTRANS 2019].

The BMBF project CrESt[3], which was conceived as a continuation of the work of SPES2020 and SPES_XT, addressed these new challenges and the increasing complexity in the development of highly automated embedded systems and developed the SPES framework for MBSE with regard to CESs.

2.3 Collaborating Embedded Systems

2.3.1 Collaborative and Collaborating Systems

With the term collaboration, we denote the (active) interaction of several embedded systems in one system network. The purpose of collaboration is to achieve a common goal through the mutual provision of functions that individual systems alone cannot achieve [Broy and Schmidt 2001], [Sha et al. 2008]. Collaboration therefore serves to achieve the goals defined in a single system or a system group and can take various forms with regard to possible binding times, the type of coupling, the process of forming the group, or the collaboration management. From the point of view of our system model, it is not so easy to distinguish collaboration from "simple" interaction. In fact, collaboration must of course manifest itself at the interfaces of the collaborating systems in the form of interaction.

A collaborative system can therefore be distinguished from a non-collaborative system not so much by the system model as by its origin, its use, and its purpose. Maier has defined two properties that must apply to collaborative systems (as opposed to non-collaborative systems) [Maier 1998]:

- Operational independence of elements: The systems involved in a collaboration provide added value even if they are operated independently of the collaboration.
- Managerial independence of elements: The systems involved in a collaboration are actually developed and operated independently.

Collaborative embedded system (CES)

Taking these properties into account, we define a *collaborative embedded system* (CES): CESs are embedded systems that can collaborate with other CESs to achieve goals that are difficult or impossible for a single CES to achieve alone.

[3] Funded by the German Federal Ministry of Education and Research under the funding code 01IS16043

Collaborative system group (CSG)

A *collaborative system group* (CSG) is formed dynamically at runtime by a set of CESs that collaborate with each other. The CESs involved can take on different roles in the group. It is important to note that a CSG can also be seen as a system in the sense of Figure 2-1, where the internal structure is formed by the collaborating CESs.

Collaborating systems

We call a CES a collaborating CES if it is actively involved in a CSG at a certain point in time. Note that a system can be collaborative for a certain CSG type (e.g., platoon), while it is not collaborative for another CSG type (e.g., adaptable and flexible factory).

Note that a CSG and the CESs are at different levels of granularity

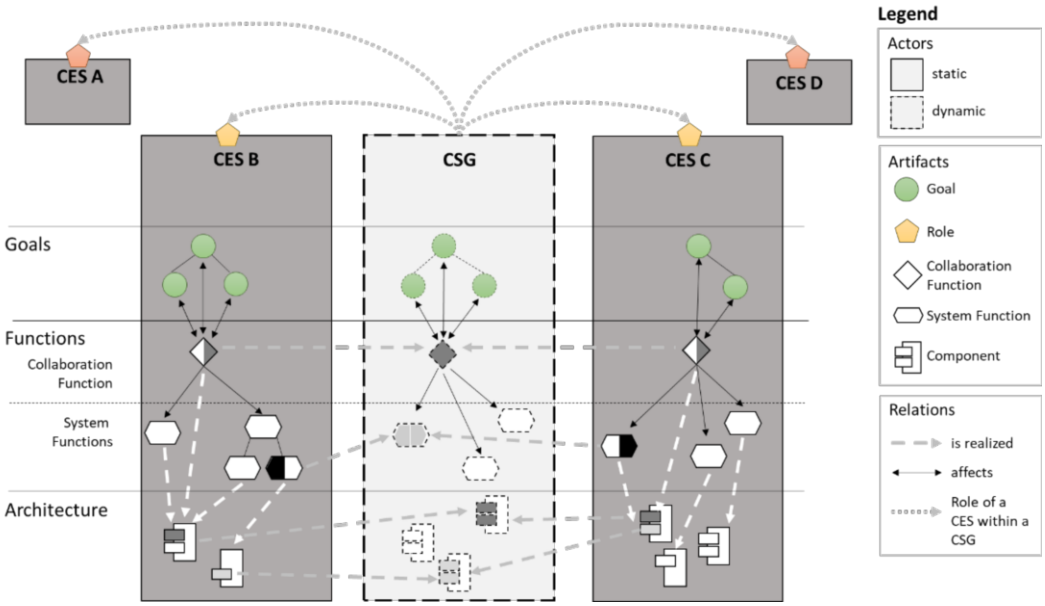

Fig. 2-2: *Goals, functions and architectures in collaborative system groups*

in the SPES modeling framework (see [Pohl et al. 2016]): while the CSG models describe the overall system and are thus located at the highest level of granularity, the CES models are located at the next level of granularity of the framework, and thus represent architectural components (subsystems) of the CSG.

CESs can be developed and realized with the help of methods defined in CrESt[4]. The most important concepts for the collaboration of CESs are illustrated in Figure 2-2.

2.3.2 Goals of System Networks

CES goals

In addition to the CESs, the CSGs also have goals that are negotiated when the CSG is formed. This involves checking whether there is sufficient agreement with regard to the achievement of the goals of the participating CESs. We differentiate between negotiable goals ("soft goals"), which can be adjusted if necessary to allow the CES to participate in a CSG, and non-negotiable goals ("hard goals"), which, if they conflict with the goals pursued by a CSG, may result in a CES being unable to join a CSG. Goals can also be refined hierarchically. Furthermore, relationships can be used to define dependencies between goals. The set of goals pursued by a CES, as well as the relationships between the individual goals, form the goal system of a CES, which is already fundamentally (generically) defined during development. This goal system is then individually instantiated at runtime in the respective CES instances, thus concretizing the goals.

CSG goals

During the conceptual development of a CSG, a basic goal system consisting of soft and hard goals is also defined. This goal system contains overarching goals that can only be achieved within the CSG through cooperation between the CESs involved. At runtime, the CSG goal model is instantiated by goal negotiations between the participating CESs: the overarching goals are specified and compared with the individual goals of the participating CESs. Within the collaboration, the participating CESs make their system functions available to each other in order to achieve common goals that they cannot achieve on their own. If conflicts arise—for example, between the overarching goals of the CSG and the individual goals of the participating CESs—these must be resolved.

2.3.3 Coordination in System Networks

Control mechanisms

Where different CESs contribute collaboratively to a CSG goal, it must be ensured that the individual contributions are coordinated and aligned. Different control mechanisms are conceivable here. For

[4] For a better distinction between CES and CSG, we assume in the following that CESs, unlike CSGs, are "static" in the sense that their functional scope and physical architecture are already fully known at the time of design. In particular, this excludes the possibility of nesting system networks of CSGs.

example, the collaboration of several systems within a CSG can be centrally controlled by the role of a coordinator. The CSG coordinator can also decide when and under which conditions other CESs join or leave the CSG. In contrast, collaboration within a CSG can also be organized decentrally. Depending on how critical the contribution of individual CESs is to the common goal, their commitment to the CSG will also be more or less firm: for example, a CSG can forbid its participants to leave the group before certain common goals have been achieved.

Whether a CSG is to be managed and organized centrally or decentrally depends on the circumstances of the respective domain on the one hand, and on the other hand, on the roles the CESs can take within the collaboration. In special cases, it may be necessary to prepare CESs for collaboration through structural design. Should these CESs wish to enter a new CSG in order to collaboratively contribute in another way to a new collaborative goal, a reconfiguration might be necessary that can only be performed by an external actor and for which the CES or even the CSG has to be temporarily taken out of service.

2.3.4 Dynamics in System Networks

In the following, dynamics is understood to mean both the dynamics within CSGs and the dynamics within their operational contexts. Dynamics refers to the change over time in structure and behavior over time.

Dynamicity of the CSG

The concrete inner CSG structure is dynamic, since new CESs can join and leave the CSG at runtime. For example, at design time, there is no definition of how many CESs are currently involved in the CSG. At this point in time, only the types and roles of the participating systems and the basic structure of the CSG are defined.

Dynamicity of the operational context

The operational context is also dynamic, since systems can enter and leave the context. The special consideration of dynamics here is that there is transition between system components of CSGs (CESs) and their context. This includes, for example, the fact that individual systems in the context of a system join together to form a CSG. The question as to which objects belong to the context of a system and which do not (i.e., which CESs are part of the CSG) depends on their relevance in connection with the fulfillment of the CSG goals.

Open world assumption

For dynamic systems, depending on the application domain, openness in the sense of the *open world assumption*[5] plays an important role [Keet et al. 2013]. In contrast to the *closed world assumption*, there is no assumption here that the possible states of the operational context are completely known from the outset, but rather that states may exist that are unknown to the system. This view has important consequences. For example, an object recognition algorithm would not be able to reject an unknown object as a possible malfunction but would have to recognize "unknown" objects. In other words, openness is understood to mean the property that the environment in which a CES or CSG is to operate is not fully known at design time. If, from the perspective of the CES or CSG, the context in which it is expected to operate is not fully known, this is called openness of context. If the structure of the CSG itself is not fully known at the time of development, this is called openness of CSG. Openness can refer to object instances and object types. The former allows the occurrence of additional objects of previously known types, while the latter also allows the occurrence of objects of new types.

Uncertainty

As a further consequence, CESs and CSGs formed at runtime that operate in open contexts must be able to deal with imprecise, contradictory, uninterpretable, and even missing context information [Bandyszak et al. 2020]. The phenomenon of such "fuzzy" information about properties of the CES or CSG context is characterized by the term "uncertainty" of context information. CESs and CSGs must be able, whenever necessary and possible, to mitigate the existing uncertainty individually or within a CSG — that is, to dissolve the uncertainty completely, reduce it, or take appropriate measures to continue to operate reliably and robustly in the context of the given uncertainty.

Conceptual definition

At the design stage, CSGs are developed conceptually — for example, in the form of standardization of interfaces and the definition of basic architectural decisions and concepts for the formation of a CSG. This defines the type of the CSG and its abstract properties and goals. However, the overall system behavior and the complete architecture of a CSG can only be specified after instantiation and only taking into account the CESs. This concrete realization by CESs only takes place dynamically at runtime through the interaction of the collaborating CESs involved. All prerequisites

[5] The "closed world assumption" describes the principle that only events that were considered at design time occur in a context and that other events should be treated as failures.

that a CES must fulfill in order to participate in a CSG must be described conceptually at development time. If a CES assumes one or more roles in a CSG, it provides the system group with the necessary functions.

The formation of a CSG must be designed and specified conceptually during the development phase — both at the level of the CESs and at the level of the CSG at various levels of detail by describing the necessary interfaces and protocols. This ensures that the CESs have a common definition of the communication (suitable protocols and interfaces), of roles to be assumed and their interaction in the CSG, system functions to be provided, and other quality requirements of the CSG during runtime. Here, too, the respective domains specify the level of detail to which a CSG is planned during this concept development and the extent to which knowledge about the nature of the CSG differs between the CESs (potentially involved).

2.3.5 Functions

Function interface

In order to fulfil the goals defined in the CESs and CSGs, different functions that must be implemented in the CESs are required. A function can be described at its interfaces by inducing a certain behavior on the basis of predefined, possible inputs and thereby generating different outputs [Broy and Stolen 2001]. The current implementation is encapsulated by the interface and the input/output behavior. For functions to actually be executed, they must be implemented in an architecture.

System functions

We distinguish (logically) between two classes of functions: one subset is formed by the system functions, which can be represented at very different levels of detail and represent the concrete end-to-end added value compared to the system context and to the achievement of the CSG or CES goals that a CSG or CES is capable of providing.

Collaboration functions

The second class consists of the collaboration functions: prior to collaboration, the CESs must communicate with each other, exchange information about their possible contributions in the form of system functions, communicate and, if necessary, adapt, negotiate, and prioritize their goals, and define the concrete course of the collaboration. This requires a comparison between the requirements for goal fulfillment and available system functions. The role that each CES will take on within the CSG must also be determined before the actual collaboration takes place. This depends, among other things, on which role a CES can generally take on due to its functional nature. All these basic functions for the realization of a collaboration, and

especially the alignment between goals and system functions, are summarized in the model as collaboration functions.

A collaboration function differs from the system functions in that it does not so much represent the individual contribution of the CES, but rather provides the functional basis for enabling the collaboration. Every CES must have collaboration functions in order to be able to collaborate in principle, regardless of which concrete system functions it contributes to a collaboration. Which CESs within a CSG communicate with each other and which hierarchies exist to make decisions depends on whether the CSG is organized centrally, with fixed hierarchies, or decentrally.

Emergent behavior

In either case, the CSG is a construct that is pre-designed at design time, implemented in the CES, and dynamically assembled at runtime. Both the goals of a CSG and its functions are aggregated components that are implemented only in the CES. Thus, a CSG function for achieving a CSG goal consists of a combination of system functions of several CESs involved in the collaboration or, if applicable, of one or more system functions of a CES. The coordinated execution of the system functions of the CESs generates the behavior that realizes the CSG function. This behavior can also be described as emergent, since the CSG functions may create new properties of the CSG as a result of the interaction of its collaborating elements. The emergent properties of the CSG cannot always be directly—or at least not always obviously—traced back to properties of the CES, which they have in isolation.

CSG architecture

Just as the functions of a CSG are aggregated from the functions of at least one CES, the CSG architecture is formed just as dynamically from the static architectures of the CESs. In contrast to the static architecture of a CES, which can be developed and planned explicitly, the dynamic architecture of a CSG is again planned conceptually and the necessary elements are provided in the architectures of the CESs. This allows a comparison as to whether a CSG includes the corresponding architectural elements that are necessary to achieve the overall goal of the CSG. Only in this way is it possible to dynamically form different CSGs at runtime.

2.4 Problem Dimensions of Collaborative Embedded Systems

With the consideration of collaborative embedded systems in CrESt, two further problem dimensions are added to the systems considered

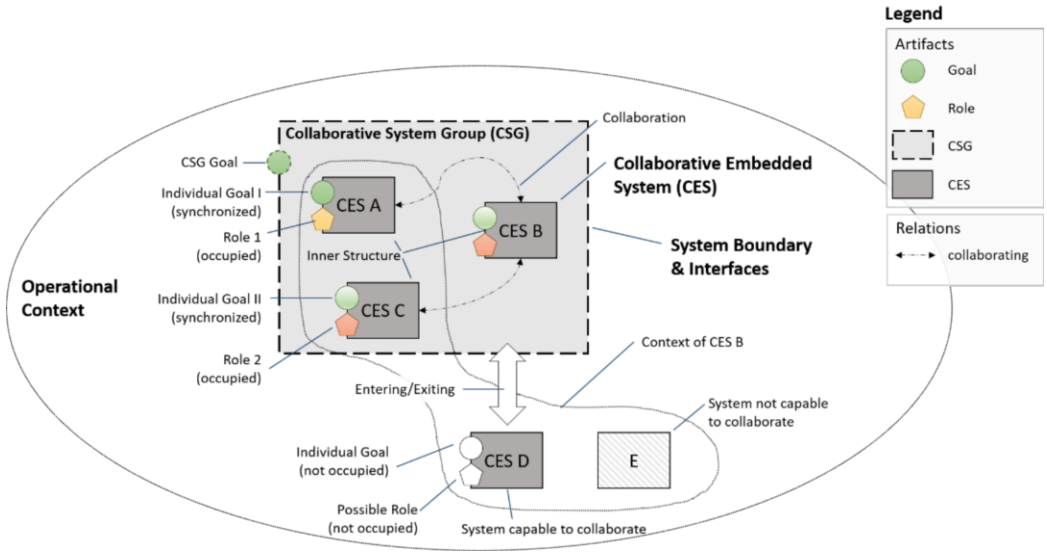

***Fig. 2-3**: Collaborative embedded systems at a glance*

in SPES: collaboration and dynamics at runtime. A distinction is also made here between CESs and CSGs, which form these embedded systems at runtime. In the following sections, we explain these additional dimensions in detail, especially with regard to the system concept. Figure 2-3 provides an overview of the most important terms and relationships.

In addition to the classic characteristics of the system model, such as interface, observable behavior at the interface, operational context, and internal structure (see Figure 2-1), communication between CESs, system goals, and the role of the CES in the system network must also be considered in the case of CESs. Furthermore, the characteristics are no longer statically and completely known at design time but can change at runtime. For example, when a CES enters the CSG, its internal structure changes. At the same time, the operational context of both the entering CES and the CSG changes. For systems in context, we also distinguish between collaborative systems—that is, systems that are able to enter a CSG due to their architecture and functions—

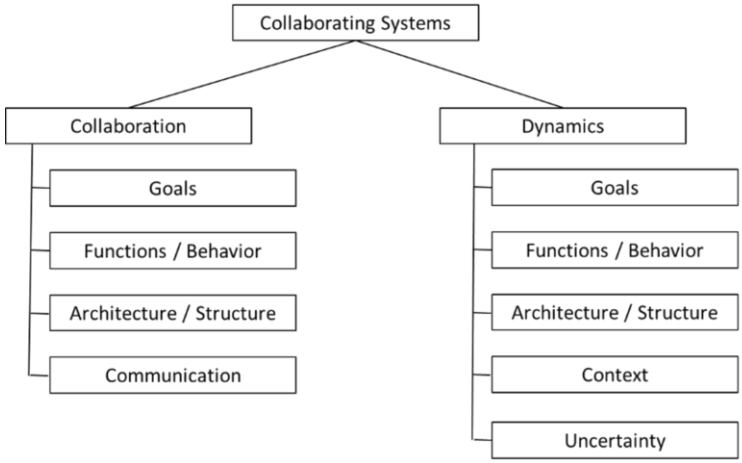

***Fig. 2-4**: Taxonomy of CrESt challenges*

and non-collaborative systems that cannot take an active role in the CSG at any time.

Based on these specific challenges for collaborative embedded systems, a taxonomy of challenges can be defined, as shown in Figure 2-4. For the two superordinate categories "*Collaboration*" and "*Dynamics*," a number of characteristics were defined. The challenges are described in detail in the following sections.

2.4.1 Challenges Related to Collaboration

CESs must be designed in such a way that they can operate in conceptually conceived types of CSG. This requires both the communication of objectives and the ability to take on different collaborative roles and act accordingly. To this end, these systems must be able to make their system functions available to CSG and - also in terms of quality - to communicate them to other CESs at runtime. For this purpose, collaboration is considered under the following aspects:

Goals

- *Goals*: A CES must be able to align its individual goals with the goals of CSG. In doing so, the CES must decide what contribution it can make to the common goals and which individual goals, if any, must be adjusted (see "Hard Goals" and "Soft Goals" in Chapter 2).

Functions / behavior

- *Functions / behavior*: A CES must be in a position to provide CSG with its own system functions. In addition, options must be

provided for how it can adapt its own functions and qualities within the framework of the negotiated CSG objectives.

Architecture / Structure

- *Architecture / Structure*: A CSG is an initially virtual entity that is thought of at design time and then forms (and can dissolve) dynamically at runtime. At design time only a conceptualization takes place. It is realized through the interaction of the participating CES and their architecture components.

Communication

- *Communication:* The basic ability of the CES to communicate with other CESs is realized by means of the collaboration functions. Among other things, these functions also form the basis for negotiating objectives, assigning roles and communicating available system functions to CSG.

2.4.2 Challenges Related to Dynamics

The developers of dynamic CESs need concepts and methods that support their design so that they can operate in a highly dynamic and possibly open operational context in dynamically formed CSGs at runtime and, if necessary, with "fuzzy" information in a targeted manner. As shown in Figure 2-3, the context of a CES differs significantly from that of a CSG according to its hierarchical structure. The context of a CSG consists exclusively of surrounding systems of the CSG, while the context of a CES is formed by the other CESs of a CSG and, if applicable, by parts of the context of the CSG.

As shown in Figure 2-2, the system and collaboration functions of CSG are formed by the functions of the CESs. A challenge to a CSG must therefore always be solved by the CESs involved, possibly also by the interaction of several CESs. Dynamics, openness and uncertainty give rise to the following challenges for the development of these systems, among others:

Goals

- CSGs must be able to adapt their goals to the changes in the operational context that they perceive via their sensor technology. This is particularly the case when CESs become part of CSG or leave CSG. This requires the possibility to dynamically adjust the goals of the CESs (see challenges on collaboration goals).

Functions / Behavior

- CSGs must be able to cope with changes in available functions. This concerns on the one hand the system functions of the CESs in an operational context and on the other hand the collaboration functions of the CESs within the CSG. For this purpose, CESs must be able to describe their available system or collaboration

functions and to adapt their system functions according to the negotiated goals. For example, a CES must be able to communicate with new objects that have been added to its CSG or operational context. In open systems (in the sense of the open world assumption), this requires the ability to communicate with systems of previously unknown types and, where this is not possible, to handle them safely in other ways. At the CSG level, it must be possible to describe the functions required to achieve certain goals in the form of sought-after capabilities and to search for these in their CSG and operational context. They must be able to recognize when new system functions are available and, if a system function is no longer available, search for alternatives. Finally, a CSG must be able to select available alternative system functions according to defined criteria.

- ❑ The CESs of a CSG must be able to deal with structural changes of the CSG. In particular, they must be able to detect changes in their operational context and CSG and adapt their interfaces accordingly. The dynamic entry or exit of a CES from the group or the change of roles of the CES within a CSG also affects the internal structure of the CSG. *Architecture / Structure*
- ❑ Both CESs and CSGs must be able to deal with changes in the operational context of the CSG or of CESs within the CSG. To do so, they must be able to consider the behavior of context objects (of the CESs and CSGs) when planning their own goal fulfillment and implementing desired functions. To analyze the current context behavior and the potential changes resulting from it, it is necessary to define the desired or expected behavior of context objects. Finally, CESs must be able to evaluate behavioral changes in terms of their impact on the CES or CSG under consideration and to draw conclusions from this. This includes, for example, the adjustment of goals. *Context*
- ❑ Dealing with uncertainty and fuzziness in data collection is also relevant for classical systems. For dynamic systems, which should be able to operate in open contexts, it must be possible to resolve or deal with uncertainties in terms of the open world assumption. *Uncertainty*

2.5 Application in the Domains “Cooperative Vehicle Automation” and “Industry 4.0”

In the following, we consider and concretize the challenges on the basis of exemplary application domains.

2.5.1 Challenges in the Application Domain "Cooperative Vehicle Automation"

The use case "Cooperative Vehicle Automation" investigates system networks that are formed between vehicles in order to achieve common goals. An obvious scenario in this context is "vehicle platooning" (computer-controlled convoy driving). In the sense of CrESt, this is a system group (i.e., a CSG) of individual vehicles (CESs) that drive in close succession in close proximity to each other with the aid of automated control systems. Often, the common goal of such a network is to reduce the fuel consumption of all participants and to relieve the individual drivers. During their participation in a platoon, individual vehicles coordinate their own goals with the common goals. For example, individual vehicles with individual destinations for a certain route can join a platoon that has a different final destination but is travelling in the same direction.

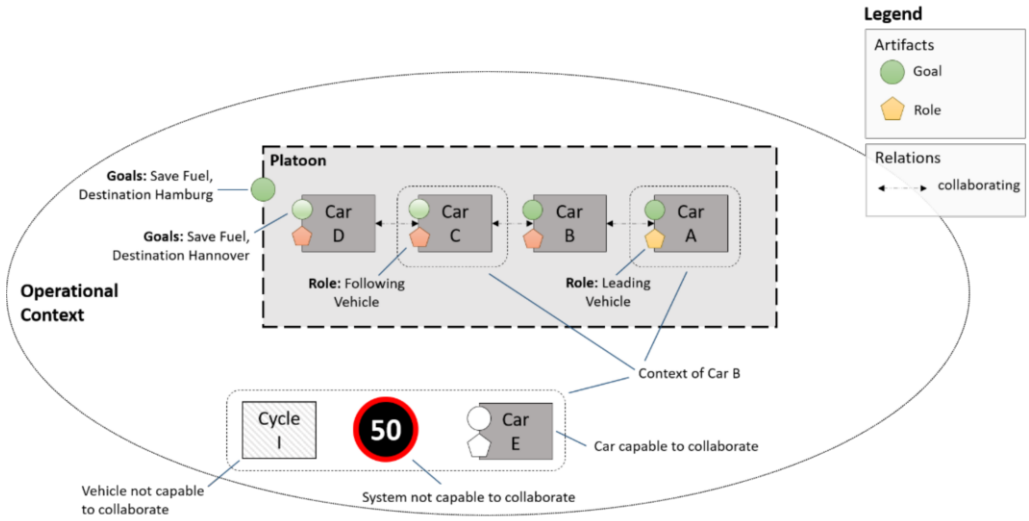

***Fig. 2-5**: Overview of collaboration in computer-controlled convoy driving*

Figure 2-5 shows an example of the structure of such a platoon, consisting of vehicles A to D. Car A, at the head of the convoy, takes on the central role of coordination, referred to here as the "leading vehicle." In this role, the vehicle coordinates basic tasks such as the creation and dissolution of the platoon, or processes such as the execution of a lane change for the entire platoon. The other vehicles take on the role of "following vehicle" and thereby transfer part of their control to the lead vehicle. In addition, individual vehicles of the

platoon can also contribute further system functions. This allows new sub-functions of the platoon to be formed and the overall functionality of the platoon to be expanded. For example, a vehicle could bring special sensors into the platoon for better environmental monitoring, which are then available to the platoon as a whole.

In order for a platoon to be formed, certain requirements must be met by the participating vehicles. The preliminary design phase for platoons must therefore define which requirements, such as wireless communication connections, standardized communication protocol, suitable distance sensors, must be met by the vehicles of a platoon.

Collaboration

In the following, we look at the specific challenges in the area of collaboration using the example of a vehicle entering a platoon. Car E wants to enter a platoon consisting of four vehicles (see Figure 2-5).

Goals

Car E must coordinate its individual goals, such as destination and cruising speed, with the platoon's common goals before entering. The cruising speed is a soft goal, so Car E is allowed to adjust its speed to the cruising speed of the platoon.

Functions and behavior

Upon entry, Car E is assigned its future role (usually as a following vehicle) in the platoon. It must adapt its behavior to this role. For example, decisions on initiating acceleration, braking, and lane changing processes are transferred from Car E to the lead vehicle.

Architecture and structure

When entering the platoon, Car E will give an entry position. This specification can influence and optimize the structure of a platoon — for example, for an imminent exit of another vehicle.

Communication

For the entry of Car E into the platoon, extensive communication between Car E and the platoon's lead vehicle is necessary. Car E has to express its wish to enter the platoon. The platoon has to communicate its common goals, as well as the entry requirements, such as role and entry position. In addition, communication is also necessary within the platoon. Before entry, the lead car must ask the members of the platoon, for example, to create a gap at the entry position. After Car E has pulled in, the lead vehicle must ask the other members of the platoon to close this gap again.

Dynamics

Let us now look at the special challenges in the area of dynamics using the example of the entry of a vehicle (Car E) into the platoon.

Goals

The entry of Car E into the platoon may lead to adjustments to the community goals (soft goals) of the platoon. For example, Car E could

bring special sensors that can detect the environment more precisely into the platoon, and thus enable a higher cruising speed for the entire platoon.

Functions and behavior

The entry of Car E can also lead to a change of roles within the platoon. For example, for Car A in its role as leader, the size of the platoon could be limited to four vehicles. For the inclusion of Car E as the fifth vehicle, the leading role must therefore be transferred to one of the other vehicles that supports the corresponding platoon size. However, it could also be that Car A hands over its role as lead vehicle to Car E after entry because Car A will leave the platoon in a short time. Functions such as the coordination of acceleration, braking, and lane change of the platoon then move from Car A to Car E.

Architecture and structure

The entry of Car E changes the internal structure of the platoon, such as the number and order of the participating cars. Depending on the sensor types contained in Car E (e.g., for distance measurement), the interfaces of the other members may have to be adapted. Interface adaptations may also be necessary if sensors are missing or unknown sensor types are used.

Context

The context of the platoon is constantly changing. New vehicles, traffic signs, but also unpredictable obstacles on the road can appear at any time. In addition, new functionalities can appear in context, such as the sensor data of a traffic control system that provide information about the road surface. The platoon must be able to detect these changes fast enough and adapt its behavior accordingly. With the entry of Car E into the platoon, the context changes for the platoon as well as for Car E and the previous vehicles of the platoon. For Car E, the context no longer contains the platoon as a whole, but rather the individual vehicles inside the platoon. For the vehicles of the platoon, Car E now becomes a member of their own association.

Uncertainty

Platoons operate in an open environment and must therefore deal with a high degree of uncertainty and fuzziness. A platoon must be able to deal with road users not yet known at the time of the platoon's design. Road safety must be guaranteed even then. Future vehicles with new features (such as extended information about the environment) should be included in the platoon and their capabilities should be able to be used.

2.5.2 Challenges in the Application Domain “Industry 4.0”

The visions of an adaptable and flexible factory are complex and are described by different scenarios in connection with the Industry 4.0

vision [Plattform Industry 4.0 2017a], [Plattform Industry 4.0 2017b], [Plattform Industry 4.0 2017c].

One scenario frequently described in this context is order-driven production, where the CESs involved in the production of a product (also called modules in the factory) form a CSG (also called production network in the factory) based on the requirements of the product to be manufactured and with the goal of manufacturing the product. The application of the concept described in 2.3.1 results in a production network that is formed to produce a specific production order and is dissolved again after its completion.

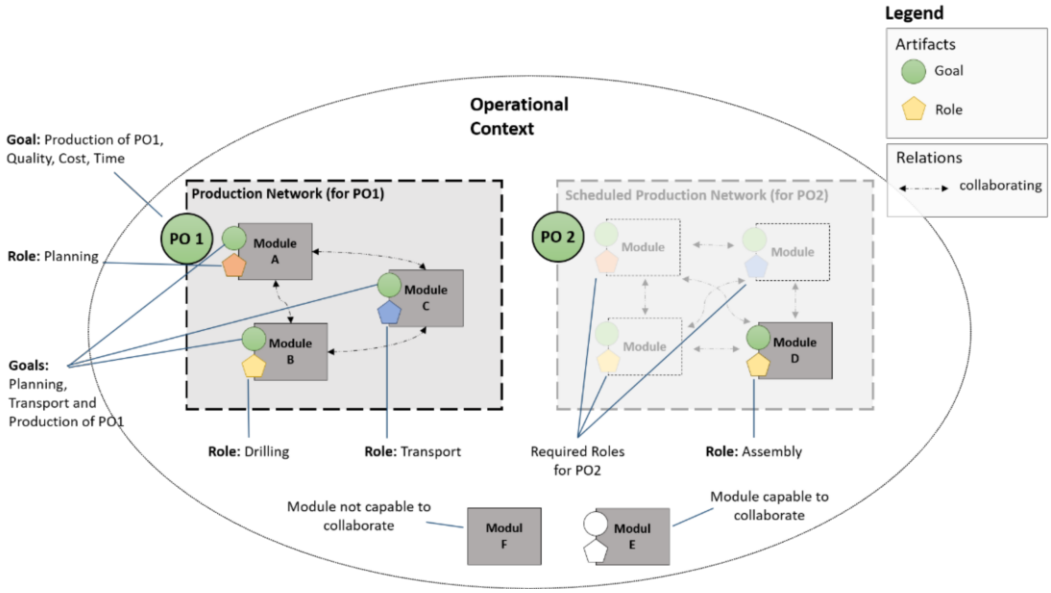

***Fig. 2-6**: Overview of collaboration in order-driven production*

In the example of the adaptable and flexible factory, the interaction of the CESs in the CSG must also be considered and described by means of suitable models in order to serve as a basis for the development of the CESs and to enable collaboration during operation. Figure 2-6 shows such an existing and a planned production network for the processing of two production orders.

To process manufacturing order PO1, a different composition of production modules is required than for manufacturing order PO2 (see Figure 2-6). When the order is received, the modules agree, based on the order information, whether and under which conditions (costs, quality, and time) they can contribute to the production. For the

production of PO1, for example, a collaboration of modules A, B, and C with the roles production planning unit, production station (in the form of a drilling station), and transport device is required, while for the production of PO2, a further function of module D in the role of an assembly device must be added. Even in the adaptable and flexible factory, there are modules that, due to lack of suitable functionality, are not capable of collaborating with other modules (e.g., module F in the figure).

Individual goals of the modules, such as achieving the highest possible throughput, energy-efficient production, or adherence to certain maintenance intervals, must be taken into account when forming production networks and compared with the higher-level and possibly conflicting goals of the production network. The fulfilment of the production order represents the overriding overall goal, which can only be achieved through the individual contributions of the modules involved in production.

In this scenario, it is assumed that in different factories, different modules with heterogeneous system functions or production functions (such as drilling, milling, transport, and assembly) are available for the production of individual customer orders, initially detached from each other. Depending on the shape of the product to be manufactured, different manufacturing functions are required for production. In contrast to platooning, this scenario is mainly characterized by the multitude of very different functions of individual CESs.

The contributing modules must both align their functions with the requirements resulting from the order and communicate their respective contribution to the production to each other in order to jointly determine the feasibility and the sequence of processing. Depending on the functions required, it may be necessary to reconfigure modules before production can start because they cannot perform a required function in the current configuration. Depending on the scope, the reconfiguration can be performed either automatically by the module itself, or by an external actor. The frequency with which individual modules are reconfigured depends on the requirements of the respective production network.

By providing their respective, very heterogeneous functions, the modules assume roles required for the production (such as production planning unit, production cell, assembly station, transport device) and contribute to production within a CSG.

Collaboration

In order to realize the collaboration of the modules for the joint production of a product, numerous challenges have to be met. By combining the very heterogeneous functions of individual modules, it should be possible to manufacture a product that a single module could not manufacture on its own due to its limited possibilities.

Goals

Since each individual module can make only a limited contribution to the overall production, and since these individual contributions must be coordinated for an aggregated overall contribution, achievable intermediate goals (such as progress in the production process that can be achieved by the individual module) must be defined. This requires that the modules have machine-interpretable descriptions for their respective functions and that they exchange these descriptions, as well as metadata (e.g., units used, qualities provided) in the context of communication with other modules through their collaboration functions. From these descriptions, we can derive whether and to what extent a contribution can be made to the production of a product.

Function and behavior

During production, consideration must be given to the fact that the sequence of functions to be performed by the modules varies depending on the product to be manufactured. The information about the sequence of the functions of the modules to be executed is determined based on the production order. While, for example, in the case of a platoon, the functions to be executed for integrating or leaving the platoon are very similar, even with varying vehicles and destinations, in a factory, even when manufacturing very similar products, a geometrically determined, very different sequence of functions may be required in production.

The production sequence as the goal of collaboration and the resulting involvement of the modules must therefore be redefined for each production order. For example, for the production of PO1 and PO2, both functions of module A and module B are required. For PO1, however, it may be necessary for module A to execute its production functions first and module B afterwards, whereas PO2 requires the functions of module B first and then those of module A. PO2 may even require manual reconfiguration by an employee in the factory of module B because a specific tool is required. This reconfiguration must also be considered and provided for in the collaboration.

Architecture and structure

In addition to providing individual system functions, the architecture of factory modules also implements communication with other CSG modules. While communication about platooning targets is done dynamically at system runtime, targets of CSGs and CESs of adaptable

and flexible factories are aligned at configuration time. Due to the heterogeneity of goals, roles, collaboration, and system functions, the ways in which modules are combined to form CSGs are much more complex than in platooning and require intelligent procedures for coordination.

Communication

In the adaptable and flexible factory in particular, and in similar heterogeneous collaborative contexts, it may therefore be necessary to involve people as actors (experts) in the CSG formation process. In addition to the inter-CES communication, opportunities to involve experts in collaboration planning must also be provided.

Dynamics

Goals

The goal pursued by a production network is to fulfil a single production order. The production network comprises all modules that contribute to the production of the corresponding product with their production functions and, if necessary, additional functions such as production monitoring. If, for example, module B fails in its role as a production cell (drilling station) due to an error during the processing of PO1, compensation strategies are required to ensure the fulfilment of the order. In this case, the production network could request the required function from module E, since this module does not belong to any network at this time and has the basic possibility of assuming the required role.

Functions and behavior

The failure of a system function of a module as well as the search for and integration of alternative modules with comparable system functions shows the dynamics with regard to function and behavior. Such a change can result in further adaptations because new transport routes may have to be considered.

Architecture and structure

Every time a module fails or a new module is added to the production network, the architecture and structure of the network also change.

Context

Furthermore, production networks must be able to deal with changes in their context. For example, the delivery of required materials by external suppliers in the context of PO1 could change.

Uncertainty

While in platooning, a vehicle with the role of “following vehicle” leaving the platoon can be assumed to be an everyday occurrence and usually does not prevent the joint achievement of goals, the failure of modules in the production network can mean that the order cannot be fulfilled. These and other forms of uncertainty must also be taken into account during the design of individual modules of the adaptable and flexible factory.

2.6 Concepts and Methods for the Development of Collaborative Embedded Systems

2.6.1 Enhancements Regarding SPES2020 and SPES_XT

To meet the requirements for the development of collaborative embedded systems, new methods have been developed in CrESt. These methods were classified according to their contribution to the taxonomy of challenges (see Figure 2-4). Some methods can be classified into the taxonomy several times because they offer solutions for different challenges.

Process building blocks

The "Process Building Block Framework" was used to document the methods (see [Pohl et al. 2016]). This framework allows systematic documentation of how artifacts are created and processed in the development process. Each "process building block" has a well-defined input (e.g., models, etc.) and output (models, analysis results, etc.). Input and output can be further restricted by pre- and post-conditions and are assigned to the SPES viewpoints. Process building blocks can be connected to each other via relationships and thus provide a mapping to the desired development process.

SPES viewpoints and cross-cutting topics

In SPES2020 and SPES_XT, a framework for the creation of a system model was developed. The models are organized in four viewpoints: requirements viewpoint, functional viewpoint, logical viewpoint, and technical viewpoint. In addition, the framework includes special models for cross-cutting topics such as safety, variability, and validation. For the description of CESs in a system model, this framework has been extended in CrESt. The existing viewpoint structure was retained. Existing models were extended and a number of new model types were defined. These extensions are used, among other things, to describe a CSG and its relationships to CESs — for example, with respect to goals and functions (see Figure 2-7). The two main classes of the taxonomy (Figure 2-4) "Collaboration" and "Dynamics" impact all four viewpoints. Therefore, they have to be considered in the models of all viewpoints. In addition to the extensions of existing viewpoints, specific model types have been defined that consider collaboration and dynamics and cannot be

assigned directly to existing viewpoints. They are orthogonal to the

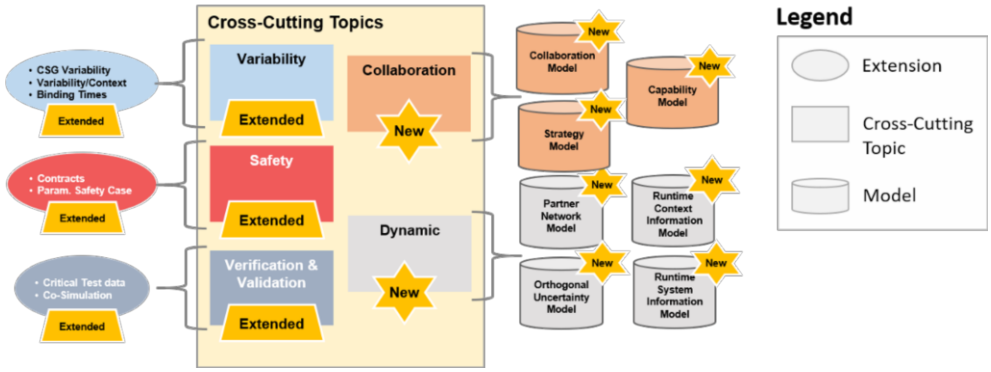

***Fig. 2-7**: CrESt framework (part1)*

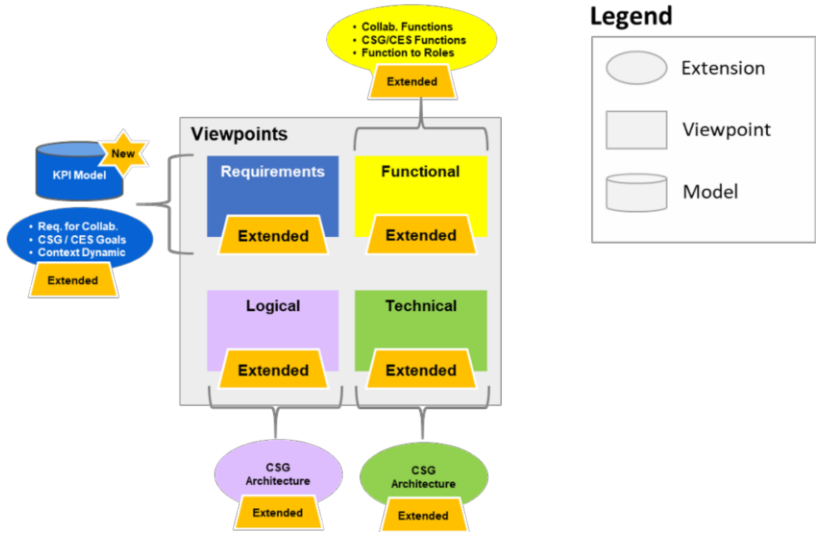

***Fig. 2-8**: CrESt framework (part2)*

viewpoints and have cross-connections to several viewpoints. Examples are models that describe collaboration strategies in a CSG or information about dynamics at runtime (see Figure 2-8).

2.6.2 Collaboration

In order to support the development of CESs and CSGs in terms of collaboration, the methodological toolbox of SPES, including the modeling approach contained therein, was extended in CrESt. A list of

methods developed to support collaboration can be found in the appendix of this chapter[6].

Goals

Goal-Oriented Requirements Language

The Goal-Oriented Requirements Language (GRL) [Daun et al. 2019] can be used to model the common goals of a CSG and the relationships to the individual goals of the CES members. With the help of this formal description, the necessary skills and key performance indicators (KPIs) of the CSG members, whose interaction in the context of a collaboration makes the achievement of the common goals possible, can be derived.

Language for partner network models

In order to analyze the individual goals of potential CSG members or their development organizations, CrESt defined a suitable language for partner network models.

Combination of different variability models

In order to illustrate the variability or configurability of a CSG based on the configuration possibilities of its members, CrESt results allow for the combination of different variability models.

Based on these extensions of the modeling framework, specific methods were developed to achieve the goals of a collaboration. Thus, it is possible to determine, at runtime, whether or not a collaborative goal can be achieved in the current CES constellation with the CES capabilities currently available. The possibility to achieve a common goal by making possible adjustments to the participating CESs is also taken into account. For example, this approach can be used to determine, for an adaptable and flexible factory, whether it is possible to produce a product with the required quality. If not, the approach allows a check to determine whether a suitable (re-) configuration of the modules can make that production possible. Further details can be found in Chapter 6.

In order to achieve a common goal of a CSG, it may be necessary for individual members to adapt the individual goals they pursue in order to subordinate them to those of the group. For example, in order to reduce their own fuel consumption by participating in a platoon, all participating vehicles must adapt their speed to the speed of the platoon. With the help of CrESt methods, suitable strategies can be derived and verified at runtime to optimally achieve both the common goals of a CSG and the goals pursued by the members of the collaborating CESs (for details, see Chapter 9 and Chapter 10).

[6] The CrESt results are available on request. See: https://crest.in.tum.de/ (website available in German only),

Functions and Behavior

The modeling of functions has been extended to support the modeling of CESs in the framework. Two types of functions can now be distinguished.

Modeling of collaboration functions

The first type of function is the collaboration functions necessary for the collaboration of CESs in a CSG and the system functions of a CES that serve to achieve the system goals. To enable the dedicated consideration of collaboration functions in particular, appropriate modeling approaches were provided in order to ultimately enable collaboration at runtime and the associated systematic coordination of different functionalities in a CSG. When designing a CSG, it is now possible to specify which collaboration functions are necessary for the participation of a CES in a CSG.

Goals and roles in a collaboration

In addition, the conceptual relationships between system and CSG functions, as well as, for example, goals and roles in a collaboration, were worked out. For a CSG, we can specify which roles the individual CES members can take on and which functions they have to provide to the CSG for this purpose. In a platoon, for example, the lead vehicle must be able to plan and execute lane changes for the entire platoon. By means of collaboration functions, system functions such as acceleration and braking must be orchestrated in a suitable way, so that, for example, the entry and exit of a vehicle into and from a platoon is made possible. A detailed description of function modeling can be found in Chapter 4 and Chapter 5.

Describing CSG behavior

Additionally, the framework has been extended to formally describe the behavior of a CSG through contracts and scenarios at design time (see Section 8.4). Furthermore, approaches including suitable tools were developed to analyze the behavior of a CSG by co-simulation. Details can be found in Chapter 12 and Chapter 13. The confidence of the CES members in the behavior of the other members plays an important role in the creation of a CSG. In CrESt, an approach was therefore developed to build up mutual trust in the behavior of CES members, for instance within a platoon, with the help of digital twins (see Chapter 14).

Architecture and Structure

Support of virtual CSG characteristics

The goal-oriented requirements models are used in CrESt to derive supporting architectures of CESs and CSGs. In addition, the architecture modeling in the framework has been extended to support the virtual characteristics of a CSG. This means that all components of a CSG architecture are realized by components of the participating

CESs. The design of a CSG is therefore described at two levels. At the development stage, the architecture is defined at an abstract level with the help of reference architectures developed in CrESt (for example, in the context of standardization). A detailed description of this approach can be found in Chapter 3 and Chapter 5. At runtime, the abstract architecture is instantiated into a concrete architecture. For example, only the framework conditions for a platoon are specified at design time. At runtime, a platoon then consists of a defined number of concrete vehicles.

Communication

CESs must be able to communicate with the different partners both within a CSG and in their environment. For example, in a platoon, the following vehicles must be able to be instructed by the lead vehicle to form a gap for the entering vehicle at the given position.

Semantic interoperability

In CrESt, an approach has been developed to achieve semantic interoperability between different and changing communication partners regarding the exchanged (possibly complex) information by means of ontologies. For example, it is important to exchange information about the specific capabilities of the individual CES members. The CrESt framework was therefore extended by a formal description of the capabilities of a CES (for details, see Section 6.3 and Chapter 7).

Safety contracts

Safety contracts must also be communicated at runtime for safety-critical systems. In CrESt, a corresponding method has been developed for this purpose. (see Section 8.4). Furthermore, suitable communication patterns were defined for the communication of CESs in a CSG and made available on the basis of the Coaty middleware framework [Coaty]. A detailed description can be found in Section 10.6.

2.6.3 Dynamics

With regard to the problem dimension dynamics, both the modeling of CESs and CSGs and the methodological toolkit for developing these systems have been expanded. The appendix of this document contains a list of the methods developed in CrESt for this dimension[7].

[7] The CrESt results are available on request. See: https://crest.in.tum.de/ (website available in German only).

Goals

Strategies for individual and community goals

With the approaches developed in CrESt, community goals can now be negotiated dynamically at runtime. The decisions made at runtime are based on strategies for ensuring that individual and community goals are achieved as well as possible. Such strategies also serve to plan and enable adaptation at runtime based on the achievement of goals. CrESt provides methods for deriving appropriate strategies and operationalizing the adaptation (see also Chapter 9).

Commitments

In order to achieve the common goals of a CSG, all CES members must fulfill their commitments. Therefore, CrESt has developed methods for assessing the risk and impact of erroneous behavior of a CES and for ensuring that the goals are met in this case as well. In the use case of the adaptable and flexible factory, for example, the failure of one module must not lead to serious damage to the factory workers and the other modules. A prerequisite for this is a method for a goal-based review of CESs at runtime. These methods are described in Section 10.2 and Section 8.3.4.

Functions and Behavior

Safety-critical analyses

Both CESs and CSGs change their behavior dynamically at runtime. In the case of a CSG, for example, this can be done by CES members joining and leaving the CSG. This dynamic behavior makes it difficult to perform safety-critical analyses completely at design time. Therefore, methods are made available in the framework where parts of the security analysis can be shifted to runtime. In order to execute these parts with acceptable effort at runtime, corresponding preparatory work at design time is necessary. Analyses with regard to risks, errors, and uncertainties can thus be analyzed in a model-based manner at design time and combined into modular safety checks that are evaluated at runtime (see Chapter 8 and Chapter 3). In addition, argument-based and contract-based approaches based on behavioral models that allow a semi-automated or fully automated safety demonstration at runtime have been developed (Section 8.4). A model-based approach to risk analysis supports safety engineers in assessing the safety of newly configured systems at runtime. Problems arising from adjustments to systems at runtime can be identified by predictive simulation under certain circumstances. A detailed description can be found in Section 10.3.

Monitoring methods

In addition, CrESt also provides suitable monitoring methods that monitor both functional behavior and time behavior.

Architecture and Structure

Dynamic architectures

In CrESt, an approach was developed for deriving a dynamic architecture by considering the corresponding architectures for different context situations. Dynamic architectures of CSGs can also be designed using reference architectures from the building set. At the development stage, the architecture is defined only at an abstract level (for instance, in the context of standardization). The concrete CESs or the number of CESs that make up the architecture at runtime are not known at this time. Only at runtime is this abstract architecture instantiated into a concrete architecture (for details, see Chapter 3 and Chapter 5. During runtime, this concrete architecture can change again and again as the members of the CSG change — that is, when they enter or leave the system.

Dynamic binding

In CrESt, approaches from software product line development were used to enable the dynamic binding of components at runtime and to analyze possible architectures at development time with regard to their variable—that is, potentially dynamic—components. This approach is detailed in Chapter 5. A CES is often integrated in different CSGs, which poses special challenges for variability modeling at design time, since short-term change requests to a CES are often implemented only for a specific configuration in a single CSG. In CrESt, methods have been developed to merge these changes into a single CES configuration with the current version of the CES product lines fully automatically (see Chapter 18 for more details).

Co-simulation methods

Testing of a designed CSG is made more difficult by the fact that a large number of different CES combinations are possible. In order to test a large number of scenarios during development, methods for co-simulating the real world with the virtual world in CrESt were developed. Using evolutionary test methods, the critical situations of a system can be identified and the quantity of test cases can be reduced to these situations.

Context

Context-sensitive variability models

CESs operate in a constantly changing environment to which they have to adapt their behavior. In CrESt, approaches have been developed to support the systems in adapting and using context information. The creation of context-sensitive variability models facilitates the search for a valid CSG configuration for a changed context.

Digital twins

Another approach combines the use of digital twins with predictive simulation using the perceived context to find the optimal

configuration for each situation (Section 3.2, Chapter 15, and Section 10.3).

Runtime-specific context models

In addition, methods have also been developed to observe and evaluate the effects of context changes on the system and its behavior at runtime (see Section 8.3.1). These are based on the modeling of runtime-specific context models. The CrESt framework now also supports sufficient testing of adapting systems in a dynamic environment. Further details can be found in Chapter 6.

Uncertainty

Identification of uncertainty types

CESs operate in an open and dynamic environment. They are developed independently of each other and can combine to form different constellations at runtime. This significantly increases the complexity regarding potential uncertainties that can occur at runtime. In CrESt, methods have been developed to systematically identify the different types of uncertainties (e.g., regarding collaboration, data quality, sensor perception, information exchange) at design time (see Chapter 7). For the systematic documentation of the uncertainties identified, a model-based approach was developed in which the uncertainties are described orthogonally—that is, in separate models with uncertainty-specific model elements—and are related to various system or context models.

For an adaptable and flexible factory, for example, this allows uncertainties that could disrupt the production process or lead to a production stop to be analyzed and documented. The uncertainties identified can then be linked to the models of the individual machines and the model used to describe the production process. For a platoon, this method can be used to identify and model uncertainties such as incompleteness and ambiguity with regard to the information exchanged between vehicles on the driving environment (for details see Section 7.3.1).

AI-based techniques for data-driven components

Another method developed in CrESt aims at identifying and handling uncertainties that may arise from the use of data-driven components—that is, AI-based techniques—for the evaluation of environmental data (see Section 7.3.2). For this purpose, the quantification of the uncertainty regarding the output of data-driven components (e.g., the recognition of a traffic sign) at runtime is enabled. This serves to ensure that data-driven components whose behavior cannot be completely predicted at development time meet safety-critical requirements during operation.

2.7 Conclusion

Within the CrESt project, important concepts of collaborative embedded systems were identified. From the resulting specific challenges, a number of key features (such as goals, communication, uncertainty) were developed. The methodological building blocks developed, as well as the extensions of existing building blocks, focus on addressing these challenges and were assigned to the main features.

Restrictive assumptions

A specific, somewhat more restrictive system concept was deliberately chosen as the basis for the work. On the one hand, the assumption was made that a CES collaborates in at most one CSG at any given time. On the other hand, hierarchical CSGs (i.e., system networks of system networks) were excluded from the analysis. For many use cases, including those considered in the project, these assumptions are quite practical. Future work in this topic area should look more closely at these limitations.

Integration of AI technologies

Increasingly, methods of artificial intelligence (AI) are being used in embedded systems. The AI methods (for example, machine learning, deep learning, data analytics, semantic technologies) are as diverse as their applications. These range from the analysis and classification of existing situations to the interpretation and evaluation, diagnosis and prognosis, and the creation of proposals for action and independent action in the sense of autonomous systems. The use of AI technologies makes it possible to process incoming information appropriately and to adapt to changing conditions at runtime.

A central challenge for the integration of AI technologies in CESs and CSGs is to guarantee the essential functionality and quality characteristics of the systems. In general, the behavior of AI methods cannot be completely determined at development time. Therefore, it is unclear which adaptations the systems make at runtime and in what way this influences the collaboration and dynamics of the CESs and CSGs. An interesting question here is whether and how the necessary conceptual development of the CSG level can be replaced by the use of AI methods at runtime.

Novel challenges

Furthermore, the integration of AI components in the context of uncertainties leads to novel effects and challenges that have to be considered as early as development time. These include, for example, data that is not 100% trustworthy (i.e., data with undetected systematic deviations or fuzziness), non-deterministic behavior, runtime variances, malicious misinformation, and commands from

outside the system boundaries. These uncertainties affect the knowledge gained from AI components. This and the dynamic adaptability create completely new challenges for the development and quality assurance of embedded systems.

The secured integration of powerful AI technologies in CESs and CSGs marks a decisive development step for future collaborative systems. The necessary extensions of the design methodology would have to be investigated in future projects.

2.8 Literature

[Bandyszak et al. 2020] T. Bandyszak, M. Daun, B. Tenbergen, P. Kuhs, S. Wolf, T. Weyer: Orthogonal Uncertainty Modeling in the Engineering of Cyber-Physical Systems. In: IEEE Transactions on Automation Science and Engineering, IEEE 2020.

[Bresciani et al. 2004] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos: Tropos: An Agent-Oriented Software Development Methodology. In: Autonomous Agents and Multi-Agent Systems 8, 2004, pp. 203–236.

[Broy and Stolen 2001] M. Broy, K. Stolen: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement, Springer 2001.

[Broy 2010] M. Broy: A Logical Basis for Component-Oriented Software and Systems Engineering. In: The Computer Journal, vol. 53, no. 10, 2010, pp. 1758-1782.

[Broy and Schmidt 2014] M. Broy, A. Schmidt: Challenges in Engineering Cyber-Physical Systems. IEEE Computer, 47(2), IEEE 2014, pp. 70-72.

[Broy et al. 2020] M. Broy, W. Böhm, M. Junker, A. Vogelsang, S. Voss: Praxisnahe Einführung von MBSE – Vorgehen und Lessons Learnt, White Paper, fortiss GmbH, 2020 (available in German only).

[Coaty] Coaty.io: Coaty Developer Guide. https://coatyio.github.io/coaty-js/man/developer-guide/, accessed on: 07/14/2020.

[Damm and Vincentelli 2015] W. Damm, A. S. Vincentelli: A Conceptual Model of System of Systems. Second International Workshop on the Swarm at the Edge of the Cloud, ACM 2015.

[Daun et al. 2019] 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, 2019.

[France and Rumpe 2007] R. France, B. Rumpe: Model-Driven Development of Complex Software: A Research Roadmap. In: Future of Software Engineering (FOSE'07), IEEE 2007, pp. 37-54.

[Grosz 1996] B. J. Grosz: Collaborative Systems. In: AI Magazine, vol 17, no 2, 1996.

[Horkoff et al. 2019] J. Horkoff, F. Başak Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, L. Piras, J. Mylopoulos, P. Giorgini: Goal-Oriented Requirements Engineering: An Extended Systematic Mapping Study. In: Requirements Engineering 24, 2 (June 2019), 2019, pp. 133–160.

[Keet et al. 2013] C. M. Keet, W. Dubitzky, O. Wolkenhauer, K.-H. Cho, H. Yokota: Open World Assumption. In: Encyclopedia of Systems Biology, Springer, New York, 2013.

[Lamsweerde 2000] A. v. Lamsweerde: Requirements Engineering in the Year 00: A Research Perspective. In: Proceedings of the 22nd International Conference on Software Engineering, Invited Paper, ACM Press, June 2000.

[Maier 1998] Mark W. Maier: Architecting Principles for Systems-of-Systems. In: Systems Engineering 1(4), 1998, pp. 267-284.

[Plattform Industrie 4.0 2017a] Plattform Industrie 4.0: Application scenario in practice: order-controlled production of a customised bicycle handlebar. BMWi, Berlin, 2017.

[Plattform Industrie 4.0 2017b] Platform Industrie 4.0: Aspects of the Research Roadmap in Application Scenarios. BMWi. https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/aspects-of-the-research-roadmap.pdf; accessed on 04/04/2020.

[Plattform Industrie 4.0 2017c] Plattform Industrie 4.0: Fortschreibung Anwendungsszenarien der Plattform Industrie 4.0. BMWi: https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/fortschreibung-anwendungsszenarien.html; accessed on 04/08/2020.

[Pohl et al. 2012] K. Pohl, H. Hönninger, R. Achatz, M. Broy (Eds.): Model-Based Engineering of Embedded Systems: The SPES2020 Methodology, Springer, Heidelberg/New York, 2012.

[Pohl et al. 2016] K. Pohl, M. Broy, H. Daembkes, H. Hönninger (Eds.): Advanced Model-Based Engineering of Embedded Systems: Extensions of the SPES2020 Methodology, Springer, Heidelberg/New York, 2016.

[SafeTRANS 2019] SafeTRANS e.V.: Safety, Security, and Certifiability of Future Man-Machine Systems, 2019.

[Selic 2003] B. Selic: The Pragmatics of Model-Driven Development. In: IEEE Software, 20(5), IEEE 2003, pp. 19-25.

[Sha et al. 2008] L. Sha, S. Gopalakrishnan, X. Liu, Q. Wang: Cyber-Physical Systems: A New Frontier. In: 2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008), 2008, pp. 1–9.

2.9 Appendix

In the CrESt project, methods and building blocks for modeling collaborative systems and system networks were developed. The documents containing a detailed description of the project results can be requested via the project website (https://crest.in.tum.de/, website available in German only).