Jan Christoph Wehrstedt, Siemens AG
Jennifer Brings, University of Duisburg-Essen
Birte Caesar, Helmut Schmidt University Hamburg
Marian Daun, University of Duisburg-Essen
Linda Feeken, OFFIS e.V
Constantin Hildebrandt, Helmut Schmidt University Hamburg
Wolfram Klein, Siemens AG
Vincent Malik, Siemens AG
Boris Wirtz, OFFIS e.V.
Stefanie Wolf, Siemens AG

6

Modeling and Analyzing Context-Sensitive Changes during Runtime

For collaborative embedded systems, it is essential to consider not only the behavior of each system and the interaction between systems, but also the interaction of systems with their often dynamic and unknown context.

In this chapter, we present a solution approach based on process building blocks—describing both the modelling approach as well as the model execution approach—for engineering and operation to achieve the goal of developing systems that deal with dynamics in their open context at runtime by re-using the models from the engineering phase.

W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_6

6.1 Introduction and Motivation

CESs operate in open dynamic contexts

Software-intensive embedded systems differ from classic systems in that they interact with their operational context through sensors and actuators [Daun et al. 2016]. The same holds for collaborative embedded systems (CESs) and collaborative (embedded) system groups (CSGs), as their behavior and functions are strongly influenced by changes in the systems' contexts. For example, dynamic traffic conditions—such as pedestrian behavior (which is very difficult to predict), construction work, or other unexpected traffic participants—are major challenges to be met by a platoon of autonomous cars. Similar challenges arise for adaptable and flexible factories in the case of producing individualized products or new variants of the product mix. If the system acts autonomously, a decision has to be taken very quickly on how to react to changes during runtime. This also concerns the operator who has to decide how to deal with the system due to changes.

CESs must be able to cope with context changes during operation

Therefore, methods for coping with these changes during operation must be developed, and this requires the development of suitable methods and models that can be used during operation. These methods must be developed and validated during the engineering phase, which requires the reuse or rather the migration of models from the engineering phase towards runtime.

6.2 Solution Concept

To achieve the goal of developing systems that can handle the dynamics of their open context and to reuse the models from engineering in order to deal with these changes during operation, we present a novel solution approach as described in Figure 6-1; the notation will be explained later in this section.

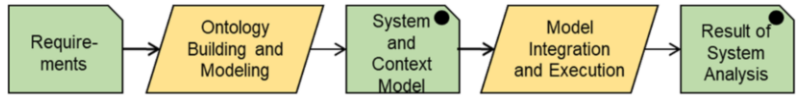

***Fig. 6-1**: Process steps for developing models for CESs interacting with their context and their execution during operation*

Modeling the system and the context

Initially, we develop a modeling approach for both the system and the context: as ontologies are a suitable technology for enabling semantic

interoperability, they allow the creation of information models that link machine-readable meaning to information, thus enabling CESs to mutually understand the shared information. This allows the system and context models to be reused for different applications and also guarantees a certain completeness of the information.

Contributions

Based on that, we provide insights into a scenario analysis approach that identifies, depicts, and analyzes certain contextual situations based on a graphical modeling notation. In this approach, spatial/context constraints are captured as invariants, which change over time. We then present different approaches for model creation, using the example of modeling the capabilities of collaborative embedded systems. Subsequently, in Section 6-4, we describe how to develop decision methods based on these models and runtime information. This leads to the models generated being integrated into executable models that can be used for system analysis. We explain this general approach with different examples: a capabilities check of a system group to fulfill requirements given in a context situation and the seamless integration of simulation for validation of dynamic system behavior.

CrESt process building block notation

Some of the methods presented can be broken down into a set of sub-methods with certain interdependencies and specific types of artifacts (see Figure 6-2). To show the relationships between such sub-methods (e.g., how to combine them), to clearly assign them to the design phase or to the runtime of a CES/CSG, and to classify their degree of formalization and automation, we choose the following notation:

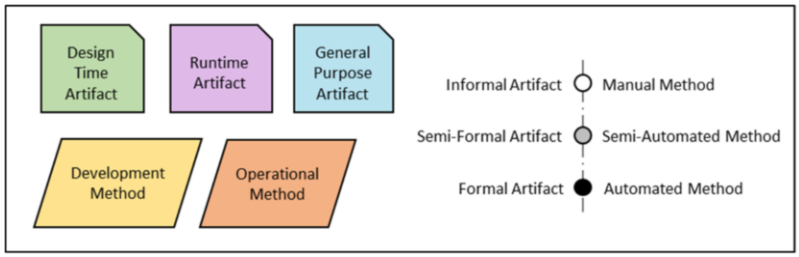

Fig. 6-2: *CrESt process building block (PBB) notation*

6.3 Ontology and Modeling

CESs operate in open contexts

The development and operation of CESs that can cope with a frequently changing environment has become a major field of research. Of particular interest is a CES's ability to operate in open contexts, that is, situations where context objects (e.g., other CESs)

enter or leave the CES's context at runtime [Schlingloff et al. 2016]. As CESs are usually embedded in a network of CESs (i.e., a CSG), an individual CES must be able to process and communicate complex information from/to changing communication partners during its life cycle in order to provide its functionality.

Ontologies as a solution concept

This addresses the development of the models as well as the modeling approach itself. Ontologies are becoming the appropriate means for these systematic approaches. In the following section, we therefore address a modeling approach for the system and context to develop models that can cope with online decisions.

6.3.1 Ontology Building

Ontologies must be derived systematically

Ontologies provide a suitable technology for formalizing different aspects of a CES and could potentially be the provider of data, information, and knowledge for different software functionalities [Sabou et al. 2019]. In the case of a CES in the manufacturing domain, an ontology can be used to formalize manufacturing-related capabilities (see Sections 6.3.2 and 6.4.2), features relevant for reconfiguration of a manufacturing system (see Section 6.3.3), or serve as input information for a simulation of the manufacturing system (see Section 6.4.1). The development of an ontology is a non-trivial task that requires high efforts from different stakeholders. Therefore, an efficient ontology building procedure is crucial in order to have a positive cost-benefit trade-off. The latter is the goal of the method described below, which is summarized by Figure 6-3.

Ontologies define terminologies

An ontology may take a variety of forms, but it will of necessity include a vocabulary of concepts and some specification of their meaning, including definitions and an indication of how concepts are interrelated. In general, the above-mentioned concepts and relationships (including attributes) form what is referred to as the terminology box (TBox), while the axioms (e.g., individuals of the ontology) form the assertional box (ABox) that follows the definitions of the TBox [Hildebrandt et al. 2018].

A three-step approach to ontology building

The method consists of three PBBs. The first PBB focuses on the elicitation of requirements for the ontology under development and the documentation of these requirements. The elicitation of ontological requirements, as presented in [Hildebrandt et al. 2018a], begins with a set of project requirements. During the documentation of the requirements of a CES, requirement models (e.g., sequence charts) of the CES that uses the ontology are annotated with ontological requirements. Ontological requirements are stated as

competency questions. These are an informal notion of a query that should be answered by the ontology in a certain way, for example, “Which processes can be performed by machine X and what sensor measures which data?”.

The second PBB is concerned with building the TBox of the ontology under development, which is described in detail in

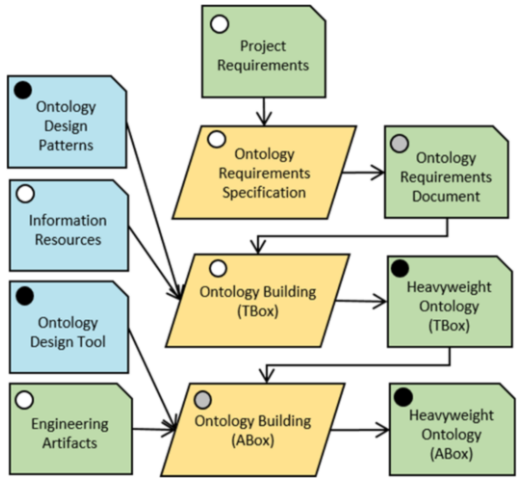

Fig. 6-3: *Process building block for the ontology building method*

[Hildebrandt et al.2020]. This PBB begins with a search for information resources that provide the relevant terms and relationships in order to build the ontology under development. Potentially suitable information resources are industry standards (e.g., ISO or IEEE standards [VDI 2005] for the process description or [IEC 2016] for properties and features), scientific publications, or project reports. Standards are preferred due to the high maturity of the concepts and relationships used as well as their potentially wide dissemination. When the proposed method is applied, heavyweight ontologies are created which formalize the knowledge of a domain (e.g., process description) in a reusable manner and can therefore be used as ontology design patterns. As presented in [Hildebrandt et al. 2020] several ontology design patterns have already been published. In an ontology building project, these ontology design patterns are combined to obtain the ontology under development.

Building the ABox

After having built the TBox of the ontology, we can begin with building the ABox of the ontology, which is sometimes referred to as the knowledge graph. The ABox contains all the relevant real-world facts (e.g., features or properties of a manufacturing process “Milling”

that was or will be performed) that should be contained in the answers to the competency questions stated at the beginning. In order to create the ABox, we use the ontology design tool we developed, which automatically transforms engineering artifacts (e.g., 3D CAD, AutomationML [Lüder et al. 2017]) into the ABox via an extract, transform, load pipeline. If there are no engineering artifacts available, we use a semi-automatic approach that relies on ontology design patterns again, see [Hildebrandt 2020a].

The final result is an ontology that describes facts about a CES that can be shared with other CESs. An application of an ontology in the domain of CESs is shown in Section 6.3.3, 6.3.3, 6.4.1, and 6.4.2 respectively.

6.3.2 Capability Modeling

Capabilities of technical systems

The capability of a technical system describes whether and with what quality the system is able to perform a specific task or fulfill a specific goal, while the task or the goal may vary with the context the system is operating in. The quality of the activity that is performed by the system depends on the context (i.e., on current requirements or boundary conditions), as well as on the immanent properties of the system (cf. [Reschka 2016], [Weiß et al. 2019]).

Capability models formalize CES and CSG capabilities

Capability models are used to formalize and document the capabilities of a CES or CSG. They are usually defined at design time of a CES or CSG or even prior to that, but they are predominantly used for runtime evaluations (e.g., a manufacturability check of a product based on the capabilities of available production systems).

Approach for creating capability models

Figure 6-4 gives an overview of the creation of capability models, distinguishing between the metamodel level, the domain capability model, the potential extension of the domain model based on project-specific boundary conditions, and the integration of the project-specific capabilities into the system model.

In the following, we explain the four sub-methods of the capability modeling approach, shown in Figure 6-4, in more detail.

Capability metamodel creation: The capability metamodel defines the structure of a capability model and its abstract syntax (cf. [Sprinkle et al. 2010]). It is neutral to any domain or application case. The metamodel is created based on a specification that may be derived from the requirements that various domains have with

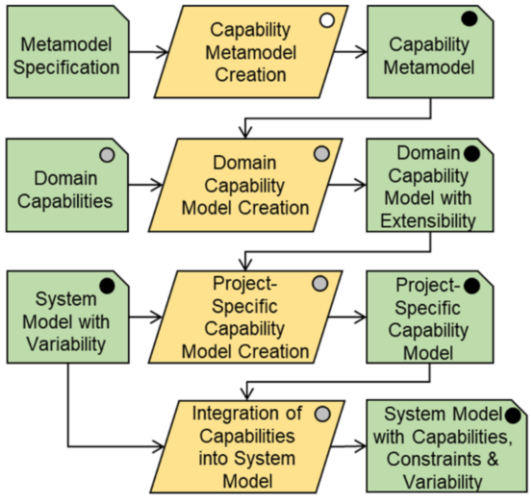

Fig. 6-4: *Method for creating capability models*

respect to capability descriptions. We present a basic version of such a metamodel for capabilities in Figure 6-5.

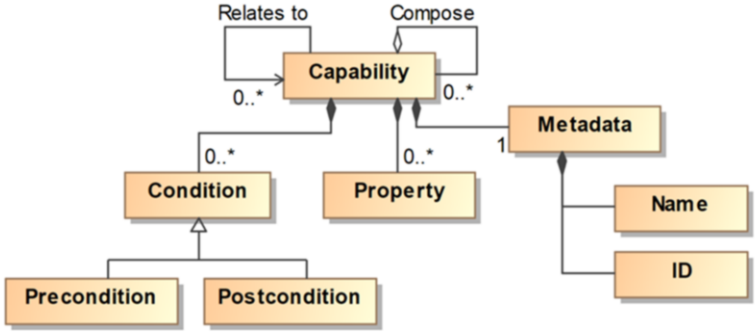

Fig. 6-5: *Metamodel for capabilities*

In this metamodel, a capability is characterized by metadata such as a name and an ID, as well as by properties and pre- or postconditions (e.g., a required condition of a workpiece before applying a production capability in the discrete manufacturing domain). In addition, capabilities may relate to each other or may in turn be composed of capabilities. Such characteristics of a capability can be standardized for a certain domain by developing domain capability models.

Domain capability model

Domain capability model creation: The capability model for a certain domain is created by identifying the capabilities of a domain based on its defined scope and by considering existing ontologies as well as expert knowledge and industry standards. These domain capabilities are the input for a domain capability model that is independent from any specific system manufacturer. In addition, the domain capability model should be extendable in order to enable project-specific modifications (e.g., for niche or special applications or in case of technical innovations). To give an example, a model for manufacturing capabilities that applies to adaptable and flexible factories in the discrete manufacturing domain can be created. A domain capability model for discrete manufacturing must cover a defined structure and uniform nomenclature, as well as a means of describing manufacturable product features and typical restrictions and boundary conditions of the production systems. Moreover, potential interdependencies between capabilities—for example, with regard to production process chains and assembly sequences—must be taken into account (cf. [Wolf et al. 2020]). Example 1-1 shows an exemplary capability description in the discrete manufacturing domain:

Example 6-6: Capability "drilling"

Metadata:

Name: Drilling

ID: 331-01

Properties (*excerpts only*):

Related manufacturing features: blind hole, through-hole

Diameter (mm)

Depth (mm)

Project-specific capability model

Project-specific capability model creation: For niche or special applications, or in the case of technical innovations, extensions to the existing domain capability model may be necessary. When a project-specific capability model is created, the domain capability model is used as a basis. The system under development, for which the capabilities need to be modeled, provides the input for the extension of the domain capability model. The result of this method is a project-specific capability model applicable to CESs and CSGs.

Integration into system model

Integration of capabilities into the system model: Finally, within a specific project, the capabilities of each CES and/or CSG must be described and interlinked with the system model. In this step, capabilities are assigned to the system and its sub-systems while

taking the system-specific constraints on these capabilities into account. Example 6-7 shows the capability "drilling" of an exemplary machine with specific values. Please note that the generic capabilities modeled in the domain capability model can be instantiated not only for a specific production setup (i.e., distinct machines), but also for specific products to be produced to allow manufacturability checks during the operation phase of a factory (see Section 6.4.2 "Capability Matching" for further details). With regard to a manufacturing system, the properties of a capability may have a range of possible values (cf. Example 6-7), whereas in the case of a product to be produced, properties may have just a single value (e.g., diameter d = 10 mm).

Example 6-7: Capability "drilling" of Machine A

Metadata:
- Name: Drilling
- ID: 331-01

Properties (*excerpts only*):
- Related manufacturing features: blind hole, through-hole
- Diameter (mm): 3 - 30
- Depth (mm): <= 100

Capabilities are subject to variability

As the capabilities of a system may be subject to variability, care must be taken to ensure that a differentiation between current and theoretical capabilities is possible. This is especially relevant if a system performs a reconfiguration or re-parameterization, as this usually implies changes in the capabilities of the system (see Section 6.3.3. The artifact that is finally generated is the system model, with capabilities and variability, which forms the basis for runtime evaluations.

Capability models can be implemented as ontologies (e.g., using the methodology in Section 6.3.1) or as feature models (cf. Section 6.3.3 and [Wolf et al. 2020]).

6.3.3 Variability Modeling for Context-Sensitive Reconfiguration

Reconfiguration during runtime extends the available capability

As stated in Section 6.3.2, certain CESs can be reconfigured during runtime, which means that the capabilities provided in a certain state can differ from the capabilities in a future state. This ability to change between different states is called reconfiguration. As CESs interact in an open and dynamic environment, a context-sensitive reconfiguration is desirable. Therefore, the goal of the engineering

method variability modeling for context-sensitive reconfiguration is to support the creation of context-sensitive variability models (CSVM) for runtime usage. Context-sensitive variability models are dedicated models that represent different configurations a system can take. According to [Mauro et al. 2016], the problem space of a context-sensitive variability model consists of three parts: a feature model, a context model, and cross-tree constraints. Accordingly, the engineering method must include a separate creation phase for each part.

Figure 6-8 shows the overview of the engineering method, as well as the runtime usage. The first step is the creation of a feature model

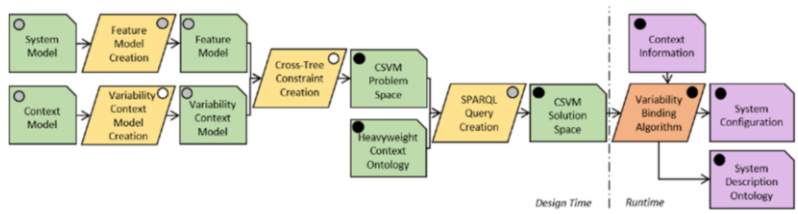

Fig. 6-8: *Procedural overview variability modeling for context-sensitive reconfiguration*

in which all common and variable parts of a system are captured [Kang et al. 1990]. To identify and extract this information, the system model, which is composed of different engineering artifacts, must be analyzed. For a manufacturing system, this system model could be a 3D-CAD drawing in the form of a step file or control code according to IEC 61131. In the second step, a variability context model is generated that contains all relevant context information for triggering the reconfiguration of a system. For this purpose, the system's context must be analyzed. Accordingly, a concept is developed that is illustrated for the manufacturing domain and helps to identify the relevant information for reconfiguration — for example, other CESs providing certain capability such as handling or a certain product requirement such as a drilling hole diameter. For this purpose, different approaches, for example, [Marks et al. 2018], are analyzed and combined. Subsequently, to conclude the third step of the engineering procedure, the context model must be integrated with the feature model by formulating cross-tree constraints.

Cross-tree constraints are used to describe the dependencies between components that are required, for example, to provide a certain capability or to specify that a certain capability can only be provided in a certain configuration. These logical formulas are phrased as

described in [Kang et al. 1990]. Once the third step of the engineering procedure has been completed, the problem space of the context-sensitive variability model is defined. The solution space is then used to enable the system to provide a self-description of its current configuration, including a description of the capabilities available. Therefore, the fourth step of the engineering method is the SPARQL query creation. SPARQL [SPARQL 2020] is a query language that builds upon the W3C standard Web Ontology Language [OWL 2020] and can be used to create, update, or query ontologies.

To create the SPARQL statements, the terminology box (TBox) that comprises the terms and relationships for describing a real-world phenomenon in an abstract manner must be considered [Asunción et al. 2004]. In the case of reconfiguration of modular manufacturing systems, the TBox describes how each system has to be characterized to be able to collaborate with other modules of the manufacturing system — for example, the module type package in the process industry [Ladiges et al. 2018], referred to in Figure 6-8 as "Heavyweight System Ontology," which is created following the ontology building method of Section 6.3.1. Thus, each reconfiguration requires an update of the system description such that it is always aligned with the current configuration of the manufacturing system. The SPARQL statements created are used to create and alter the assertional box (ABox), which contains the axioms (i.e., individuals of the ontology) [Baader et al. 2003] that represent the system description. The SPARQL statements are separated into snippets and related to features of the feature model. Only those features are represented in the system description that are selected in the current configuration. A detailed description can be found in [Caesar et al. 2019]. Once the fourth step is complete, the problem and solution space of the context-sensitive variability model is defined and can be used for reconfiguration during runtime, see Figure 6-8. Details of the variability binding algorithm can be found in Chapter 18.

6.3.4 Scenario-Based Modeling

Scenarios can be described by use case models

During requirement elicitation, use case descriptions are a well-established means of gaining insight into the system to be designed. However, use case descriptions and requirement models (as needed in Section 6.3.1 for ontology building) are often informal and lack a concise semantics, meaning that it is difficult to reuse them later on in the development process. If scenarios are expressed using a specification language that is intuitive and lightweight but still concise

and with a formal background, integration in the overall process (and even scenario-based development) becomes much easier.

Traffic sequence charts visualize traffic scenarios

In traffic-related applications (such as the implementation of maneuvers of a platoon of vehicles on a highway), it is especially important to express spatial properties and constraints. Traffic Sequence Charts (TSCs) are a visual formalism that allows intuitive and concise specification of traffic scenarios based on a formal semantics [Damm et al. 2018]. TSCs are based on acyclic graphs of chart nodes. Chart nodes capture constraints over a time interval and can be combined into a chart using sequence, choice, or parallel composition. In a chart, the constrained time intervals are seamless — that is, there is no time gap within a sequence of two nodes. Chart nodes include simple invariants (constraints that hold throughout the complete constrained interval), conditions (constraints holding at least once), and complex nodes specifying communication/event patterns or containing complete charts.

Spatial views describe spatial constraints between objects, represented by symbols in a topological view. Spatial views can be used as base constraints for invariant and condition nodes. TSCs are interpreted with respect to a modular world model that defines object classes, interfaces, and (if necessary) behaviors. World model modules are exchangeable, as long as appropriate interfaces are provided.

Spatial views are used as base constraints

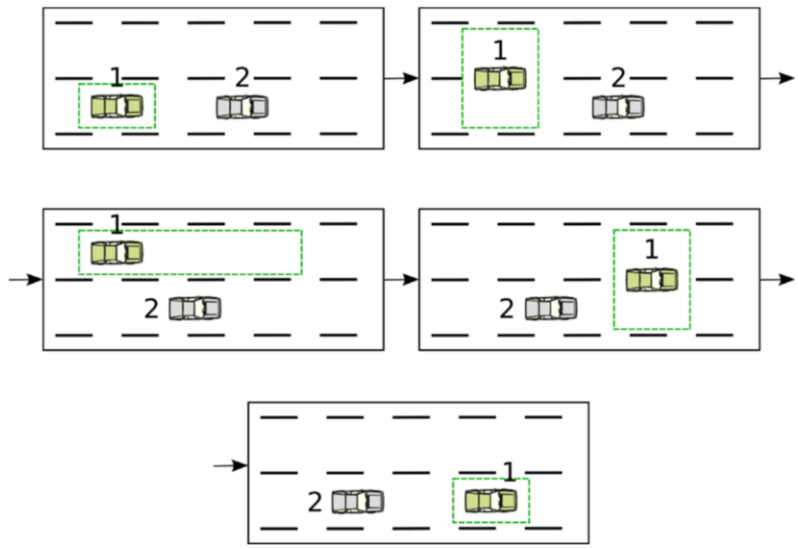

Fig. 6-9: *TSC excerpt of an overtaking maneuver*

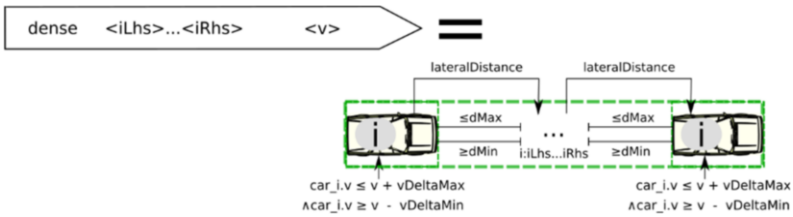

Fig. 6-10: *Vehicle group symbol abbreviation*

The combination of spatial, time, and communication constraints allows traffic scenarios to be defined intuitively. For example, the TSC excerpt in Figure 6-9 describes an overtaking maneuver: during the first invariant, car 1 is somewhere (within the box with dotted lines) behind car 2. During the second invariant, car 1 is still behind car 2, but somewhere in both lanes. In the third invariant, car 1 is somewhere next to car 2, etc.

Spatial patterns

An important aspect for a specification language for CSGs is to represent spatial patterns regardless of the concrete number of systems in the group. For example, a platoon driving on a highway lane can be considered as a sequence of vehicles on that lane connected by spatial and non-spatial relationships. In a TSC, this is expressed using the ellipsis notation; an example can be seen in the lower part of Figure 6-10. The individual vehicles are represented by car symbols parameterized with the position of the vehicle in the group, ranging from iRhs (head of the group) to iLhs (tail of the group). All vehicles drive in the same velocity range, and each adjacent pair of vehicles travels in bounded longitudinal and lateral distance.

Abstractions help manage complexity

Showing detailed information for many objects can quickly become overwhelming and leads to important information being overlooked and lost. Therefore, TSCs offer the possibility to define on-the-fly visual abstractions by introducing abbreviation symbols. These symbols abbreviate spatial patterns in full depth but can be customized to display the most important information only, therefore (in some sense) highlighting the relevant essence of the group in the current situation. For example, Figure 6-10 introduces an abbreviation symbol that is parameterized in a start and end vehicle index (thus allowing selection of subgroups) as well as traveling speed. The individual vehicles are represented by car symbols.

Figure 6-11 shows two invariants being valid for the same time interval. The upper invariant describes the inner structure of a platoon, the lower one its relationship to the context. As the picture describes normal operation, there must be no other vehicle in the

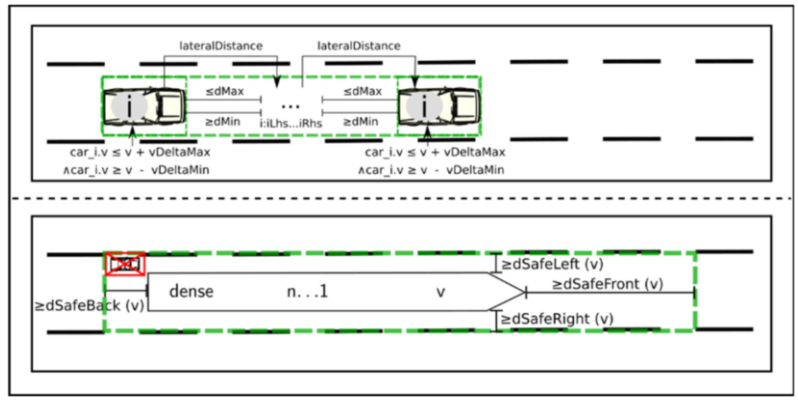

Fig. 6-11: *Separation of concerns: platoon inner structure and context*

immediate neighborhood of the platoon (depicted by the small crossed out car in the upper left corner of the box with dotted lines).

Patterns allow specification of complex situations

The patterns introduced allow the specification of complex collaborative maneuvers, such as how a platoon can circumvent an obstacle. The integrated formalism of TSCs provides the possibility to analyze the specified maneuvers, for example, by means of consistency checks as presented in [Becker 2020].

6.4 Model Integration and Execution

Operational behavior must be validated

During operation, the behavior of the system or the CSG must be validated taking the context into consideration. This can be done using simulation models, for example. Because generating these models requires high effort, it is unavoidable to automatically generate and calibrate them based on data from the real system and the knowledge of the system and context behavior. This leads to an integration of simulation into the design and runtime phase.

6.4.1 Model Generation for Simulation Models

Model Generation via Knowledge Graph

Coping with heterogeneous input data

With heterogeneous input data for the system and context, the automated generation of an executable model requires knowledge of the given system and context structure as well as an efficient connection of input data. As the context and system information are usually stored in heterogeneous data sources and formats, a common

data model, including context as well as system information, must be extracted and combined.

Ontologies help with managing the variety of data

This variety of data can be managed using ontologies as described above, represented through the application of a knowledge graph (see Section 6.3.1). In order to gain a single source of data description, the data model of the system, as well as the context used in the knowledge graph, is linked to the original data sources and formats. Queries can then be used to filter the relevant data and their linkage for the generation of simulation models. The models are matched using defined interfaces. As output of the knowledge graph, both models are extracted by export functions using requests leading to an executable simulation model.

The following properties of the knowledge graph make this approach applicable as an interface for heterogenous input data in dynamic context: in order to avoid the high effort of the classical and often manual generation of models as described above, the knowledge graph approach requires no predefined data model, offers fast access to complex hierarchical structures, as well as semantic search and analysis.

Application to a Real Production System

During operation of a production system, there are several ways to operate the system depending on requirements regarding available resources, production orders, or production time and costs. Therefore, the future system behavior must be predicted and the best operation strategy identified based on the current state.

The generalized system and context data models in the knowledge graph are concretely filled with current context information such as production plans, resource availability, and product mix provided by the *context information* artifact [Rosen et al.2020]. In addition, the system model is represented by different engineering artifacts, such as operation strategies, the production machines and their capabilities (see Section 6.3.2), as well as the plant layout (see Figure 6-12).

Simulation can be used for different tasks

Depending on the simulation task—for example, material flow analysis or 3D-kinematic analysis—the adapted data from the knowledge graph can be used to generate different types of simulation models for discrete or continuous production processes. For both aspects, the independent generation of the context and system model represented either as a process model or as operation strategies, including their coupling, is possible. Moreover, different *simulation*

models can be generated automatically by varying the parameters

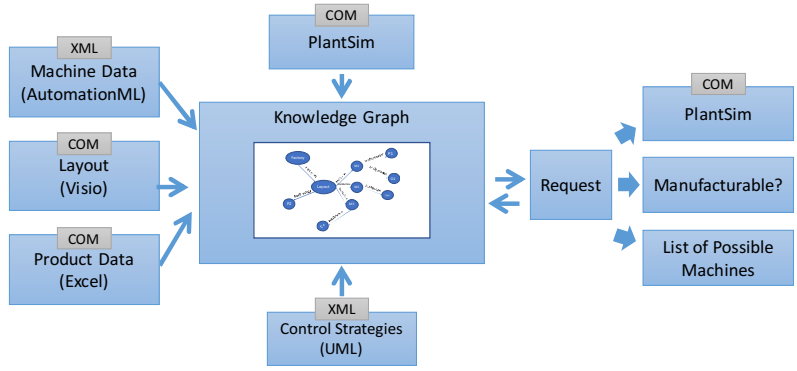

***Fig. 6-12**: Knowledge graph for factory simulation*

dependent on the operation strategy. A final assessment of the different simulation results is performed using selected KPIs (e.g., throughput, buffer utilization, productivity, costs, energy) in order to identify the "best" operation strategy.

For example, as described in Section 6.3.3 for different configurations, simulation models can be generated to select the best suitable reconfiguration or rather to provide the evaluation of different strategies to allow the operator to make a decision (see Chapter 3).

6.4.2 Capability Matching

Required capabilities may change due to context dynamicity

Context dynamicity leads to rapid changes in the operational environment of a CES (cf. [Tenbergen et al. 2018]). To cope with these changes, CESs may dynamically recombine at runtime to form a CSG that aims to fulfill a certain, usually context-dependent, goal. Note that CESs and CSGs do not always share the same goals or aim to fulfill complementing goals [Daun et al. 2019]. Due to the dynamic formation, configurations can occur where individual participants aim to achieve conflicting goals [Brings 2020]. As the dynamic formation of CSGs at runtime can hardly be foreseen at design time, a method is needed to examine whether a system group configuration actually provides the capabilities that are required under certain contextual boundary conditions. Therefore, we developed a method, based on model matching techniques, that enables such examinations by applying the following step-by-step approach (cf. Figure 6-13):

Capability matching procedure

(1) Derivation of required capabilities: The first step is to determine which generic capabilities and especially which combinations of

capabilities meet the requirements imposed by the goal or task of the CSG (e.g., the fulfillment of a customer's production request in an adaptable and flexible factory). The domain capability model introduced in Section 6.3.2 can be used to describe these capabilities needed for a certain goal attainment.

(2) *Matching of available and required capabilities and determination of suitable CSG configurations:* The next step is to answer the question of whether, based on the available CESs, a CSG can be formed that is able to provide these required capabilities. At this point, the system models of the individual CESs, with their capabilities and corresponding variability, are mapped to the required capabilities in order to identify suitable capability combinations and determine appropriate CES and CSG configurations (cf. Section 6.3.3 for reconfigurations).

(3) *Evaluation of alternatives*: Finally, if there is more than one possibility to form the CSG, the most appropriate option must be identified by using optimization criteria or considering timing or strategic aspects. The results are a certain combination of capabilities with allocated systems and a defined CSG configuration.

Figure 6-13 shows the process building blocks for this method.

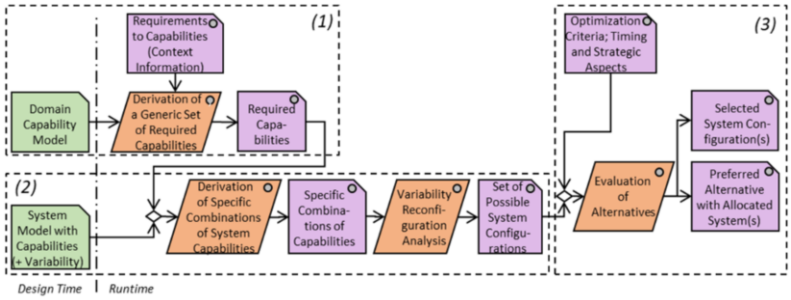

Fig. 6-13: *Process building blocks for capability matching*

Capability matching for the adaptable and flexible factory

In the following, we illustrate the application of this method for the adaptable and flexible factory use case. In this use case, the requirements for the capabilities of CESs arise from a production request for an individualized product. The factory is equipped with production systems represented by various CESs and we must examine whether they can form a suitable CSG for the fulfillment of the production request.

Instantiation of generic capabilities from the domain capability model

Figure 6-14 illustrates schematically how the generic capabilities of the domain (or project-specific) capability model (see Section 6.3.2) are instantiated for the product and the production systems, thus generating the required and provided capabilities that form the basis for the matching.

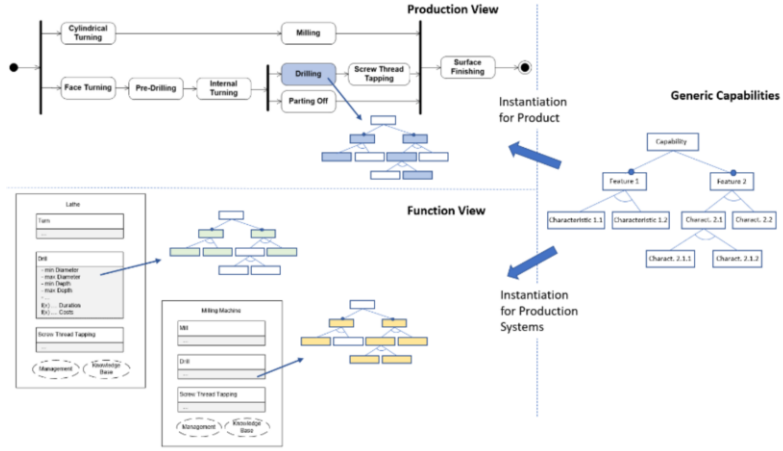

Fig. 6-14: *Schematic sketch showing the instantiation of generic capabilities*

For capability matching, a check determines whether the required capabilities, represented by the production view in Figure 6-14, match the capabilities provided by the resources of the factory as shown in the function view in Figure 6-14.

Integration of production and function view

Consequently, the integration of the production view and the function view allows us to examine whether a certain product can be produced by the adaptable and flexible factory. Figure 6-15 illustrates the combination of the production view and the function view.

The figure shows three machines—lathe, milling machine, and polishing machine—as well as their production functions (i.e., instantiated capabilities from the domain or project-specific capability model). In addition, two different production process sequence variants for manufacturing the product with the given production systems are shown. There are some common steps between these two production processes (e.g., at the beginning, the raw material is first turned with the lathe, then drilled and turned again). Other steps differ: for example, screw thread tapping is conducted either on the lathe (Example I) or on the milling machine (Example II). In both cases, different intermediate products are exchanged between the lathe and the milling machine. Depending on the choice made, the

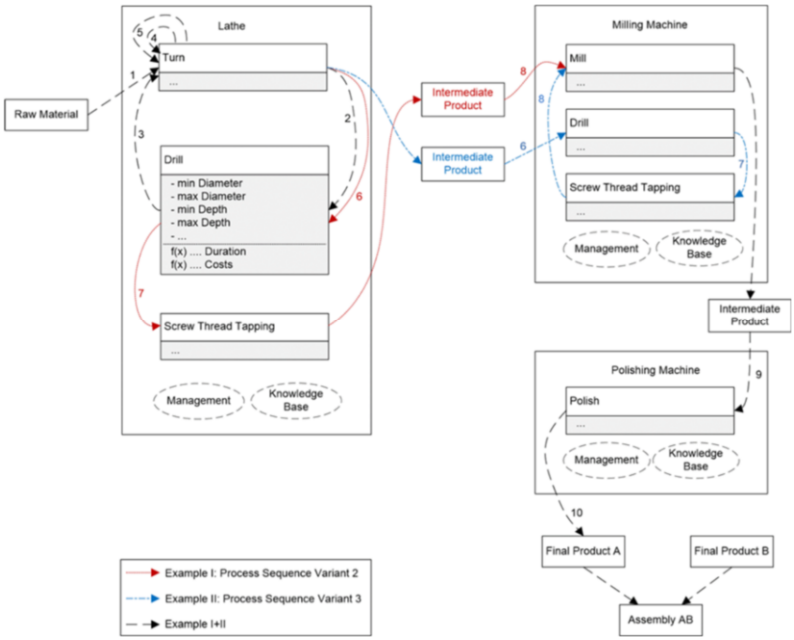

Fig. 6-15: *Integration of production view and function view to check manufacturability*

process can differ in time and costs. Therefore, the time and costs for each step must be calculated so that the optimal solution can be found.

For further information on the views presented and the principles of the matching method, please refer to [Daun et al. 2019a].

6.5 Conclusion

This chapter illustrated a modeling approach for analyzing the behavior of CESs during operation by re-using models from the engineering phase. We illustrated this approach for selected examples, addressing the main line of this developing approach. To improve the quality and reduce the effort for each step, additional improvements are necessary that lead to reusable ontologies, standardization of concepts, and interfacing to allow integration of tools. This leads to possible extensions not described in this chapter.

Even if the model generation process can then be executed automatically, a lot of effort is still required to develop the underlying ontologies in advance. Therefore, the ABox and TBox necessary for building the ontologies based on existing and established engineering artifacts must also be developed. Using databases also reduces this

effort as the manual mapping between the ABox, TBox, and the industrial application can be reused more easily [Hildebrandt 2020a].

In addition to further reduction of efforts for the modeling, automatic model validation is also a big benefit. This can be done using review models [Daun et al. 2020]. All these approaches are part of a vision to introduce model-based development approaches to non-experts in engineering and operation and efficient model generation and execution during runtime.

6.6 Literature

[Asunción et al. 2004] G.-P. Asunción, F.-L. Mariano, O. Corcho: Ontological Engineering: With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web. 1st Edition, Springer-Verlag, London, 2004.

[Baader et al. 2003] F. Baader, D. Calvanese, D. McGuinness, P. Patel-Schneider, D. Nardi: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge university press, 2003.

[Becker 2020] J. S. Becker: Partial Consistency for Requirement Engineering with Traffic Sequence Charts. Software Engineering, 2020.

[Brings 2020] J. Brings, M. Daun, T. Weyer, K. Pohl: Goal-Based Configuration Analysis for Networks of Collaborative Cyber-Physical Systems. In: 35th ACM Symp. Applied Computing, 2020, pp. 1387–1396.

[Caesar et al. 2019] B. Caesar, M. Nieke, A. Köcher, C. Hildebrandt, C. Seidl, A. Fay, I. Schaefer: Context-Sensitive Reconfiguration of Collaborative Manufacturing Systems. In: 9th IFAC Conf. Manufacturing Modelling, Management and Control (MIM), Germany, Berlin, 2019, pp. 307–312.

[Damm et al. 2018] W. Damm, E. Möhlmann, T. Peikenkamp, A. Rakow: A: Formal Semantics for Traffic Sequence Charts. In: Principles of Modeling – Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday, 2018, pp. 182-205.

[Daun et al. 2016] M. Daun, B. Tenbergen, J. Brings, T. Weyer: SPES XT Context Modeling Framework. In: Advanced Model-Based Engineering of Embedded Systems, Springer, 2016, pp. 43-57.

[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. In: 34th ACM Symp. Applied Computing, 2019, pp. 1600–1609.

[Daun et al. 2019a] M. Daun, J. Brings, P.A. Obe, et al.: Using View-Based Architecture Descriptions to Aid in Automated Runtime Planning for a Smart Factory. In: 2019 IEEE Int. Conf. Software Architecture (ICSA-C). pp. 202–209.

[Daun et al. 2020] M. Daun, J. Brings, T. Weyer: Do Instance-Level Review Diagrams Support Validation Processes of Cyber-Physical System Specifications: Results from a Controlled Experiment. In: Int. Conf. Software and System Processes, 2020.

[Hildebrandt 2020a] C. Hildebrandt: Lightweight Industrial Ontology Design Support: A Tool for ABox Creation Based on Ontology-Design Patterns. Online: https://github.com/ConstantinHildebrandt/lion; accessed on: 02/28/2020.

[Hildebrandt et al. 2018] C. Hildebrandt, S. Törsleff, B. Caesar, A. Fay: Ontology Building for Cyber-Physical Systems: A Domain Expert Centric Approach. In: 14th IEEE Conf. Automation Science and Engineering, CASE, 2018, pp. 1079-1086.

[Hildebrandt et al. 2018a] C. Hildebrandt, S. Törsleff, T. Bandyszak, B. Caesar, A. Ludewig, A. Fay: Ontology Engineering for Collaborative Embedded Systems – Requirements and Initial Approach. In: Modellierung in der Entwicklung von kollaborativen eingebetteten Systemen (MEKES), 2018, pp. 57-66.

[Hildebrandt et al. 2020] C. Hildebrandt, A. Köcher, C. Küstner, C.-M. Lopez-Enriquez, A.W. Müller, B. Caesar, C.S. Gundlach, A. Fay: Ontology Building for Cyber-Physical Systems: Application in the Manufacturing Domain. *IEEE T. Autom. Sci. Eng.*, 2020.

[IEC 2016] International Electrotechnical Commission (IEC). (2016) IEC 61360 - Common Data Dictionary (CDD - V2.0014.0014). Online: http://cdd.iec.ch/cdd/iec61360/iec61360.nsf, accessed on: 02/28/2020.

[Kang et al. 1990] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Carnegie-Mellon University, USA, Pennsylvania, Pittsburgh, 1990. – Technical Report.

[Ladiges et al. 2018] J. Ladiges, A. Fay, T. Holm, U. Hempen, L. Urbas, M. Obst, T. Albers: Integration of Modular Process Units Into Process Control Systems. In: IEEE Transactions on Industry Applications, Vol. 54, No. 2, 2018, pp. 1870–1880.

[Lüder et al. 2017] A. Lüder, N. Schmidt: AutomationML in a Nutshell. In: Handbuch Industrie 4.0 Bd. 2. Springer Vieweg, Berlin, Heidelberg, 2017, pp. 213-258 (available in German only).

[Marks et al. 2018] P. Marks, X. L. Hoang, M. Weyrich, A. Fay: A Systematic Approach for Supporting the Adaptation Process of Discrete Manufacturing Machines. In: Research in Engineering Design Vol. 29, No. 4, 2018, pp. 621–641.

[Mauro et al. 2016] J. Mauro, M. Nieke, C. Seidl, I. C. Yu: Context-Aware Reconfiguration in Evolving Software Product Lines. In: Tenth Int. Workshop on Variability Modelling of Software Intensive Systems, USA, New York, 2016, pp. 41–48

[OWL 2020] OWL 2 Web Ontology Language – World Wide Web Consortium (W3C), https://www.w3.org/TR/owl2-overview/; accessed on 04/27/2020.

[Reschka 2016] A. C. Reschka: Fertigkeiten- und Fähigkeitengraphen als Grundlage des sicheren Betriebs von automatisierten Fahrzeugen im öffentlichen Straßenverkehr in städtischer Umgebung. Dissertation, Technische Universität Braunschweig, 2017 (available in German only).

[Rosen et al. 2020] R. Rosen, D. Beyer, J. Ficher, W. Klein, V. Malik, J.C. Wehrstedt: Flexible Produktion durch digitale Zwillinge in der Produktion. In: Kongress Automation 2020, VDI, 2020 (available in German only).

[Sabou et al. 2019] M. Sabou, S. Biffl, A. Einfalt, L. Krammer, W. Kastner, F.J. Ekaputra: Semantics for Cyber-Physical Systems: A Cross-Domain Perspective. In: Semantic Web – Interoperability, Usability, Applicability, 2019.

[Schlingloff et al. 2016] B.-H. Schlingloff, H. Stubert, W. Jamroga: Collaborative Embedded Systems — A Case Study. In: 3rd Int. Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC), 2016, pp. 17–22.

[SPARQL 2020] SPARQL Query Language for RDF – World Wide Web Consortium (W3C https://www.w3.org/TR/rdf-sparql-query/ accessed on 07/23/2020.

[Sprinkle et al. 2010] J. Sprinkle, B. Rumpe, H. Vangheluwe, G. Karsai: Metamodelling: State of the Art and Research Challenges. In: Model-Based Engineering of Embedded Real-Time Systems. Int. Dagstuhl Workshop, 2007, Springer Berlin Heidelberg 2010, pp. 57-76.

[Tenbergen et al. 2018] B. Tenbergen, M. Daun, P. A. Obe, J. Brings: View-Centric Context Modeling to Foster the Engineering of Cyber-Physical System Networks. In: *IEEE Int. Conf. Software Architecture, ICSA,* 2018, pp. 206–216.

[VDI 2005] VDI/VDE Richtlinie 3682: Formalised process descriptions - Concept and graphic representation, Beuth Verlag Berlin 2005-9.

[Weiß et al. 2019] S. Weiß, B. Böhm, J. Vollmar, B. Caesar, A. Fay: Modellierung von Fähigkeiten industrieller Anlagen für die auftragsgesteuerte Produktion. In: Kongress Automation 2019, VDI, 2019 (available in German only).

[Wolf et al. 2020] S. Wolf, B. Caesar, A. Fay, B. Böhm: Erstellung eines Domänenmodells zur Beschreibung von Fähigkeiten fertigungstechnischer Anlagen für die auftragsgesteuerte Produktion. In: Kongress Automation 2020, VDI, 2020 (available in German only).