Birthe Böhm, Siemens AG
Carmen Cârlan, fortiss GmbH
Annelie Sohr, Siemens AG
Stephan Unverdorben, Siemens AG
Jan Vollmar, Siemens AG

3

Architectures for Flexible Collaborative Systems

Collaborative systems are characterized by their interaction with other systems in collaborative system groups in order to reach a common goal. These systems interact based on fixed rules and have the ability to change structurally, if necessary. Changes in the collaboration are usually triggered from outside and are time-discrete with a rather wide time scale. The architectures of these systems and system groups must support flexibility and adaptability at runtime while also ensuring specific qualities, although these changes and their consequences cannot be fully foreseen in all combinations at design time.

In order to enable knowledge preservation and reuse for the design of system architectures for flexible collaborative systems and system groups, we present a method for designing reference architectures for systems and system groups. For this approach, we present an example of a reference architecture for an operator assistance system. To adequately consider safety requirements during the design, we further introduce a method which adapts safety argumentation for flexible collaborative systems to changes in their specification or operating context.

W. Böhm et al. (eds.), *Model-Based Engineering of Collaborative Embedded Systems*,
https://doi.org/10.1007/978-3-030-62136-0_3

3.1 Introduction

Designing architectures for flexible collaborative systems and their system groups is still a challenge due to the novelty of these systems and a lack of proven methods that address their specific requirements [Böhm et al. 2018]. This applies in particular to the design of system groups and the systems collaborating within these groups.

Characteristics of flexible collaborative systems

Flexible collaborative systems assume a fixed collaboration that adheres to a fixed set of rules. Changes are usually triggered not from the system itself but, for example, by an operator of this system. These changes are not as frequent as in dynamically coupled or adaptive systems. Typical examples of flexible collaborative systems are adaptable and flexible factories.

In Section 3.2, we provide a method for designing reference architectures for collaborative embedded systems (CESs) and collaborative system groups (CSGs). Such reference architectures can then be used as blueprints for deriving system architectures for specific systems. In addition, they can be used to design specific CSGs and collaborating CESs at an interface level to allow for independent design and development of the CESs and CSGs but enable their collaboration. We then apply this approach to adaptable and flexible factories, and briefly present the resulting high-level logical reference architecture. This overview is detailed in Section 3.3 by applying the approach to one of the CESs identified, a simulation-based operator assistance system.

For numerous CESs and their CSGs, safety requirements are crucial and must be guaranteed. Our proposed safety case modeling approach in Section 3.4 supports the execution of automatic consistency checks between the safety case model and the system architecture. This approach can be used to prove that the architecture of a system satisfies the required safety properties. It ensures that, in the event of changes to the system specification or the operating context, the logical architecture still fulfills the safety requirements.

Finally, in Section 3.5, we provide conclusions and give an outlook on future work.

3.2 Designing Reference Architectures

A typical approach for designing architectures for systems starts with eliciting specific requirements. This step is followed by identifying

functions needed. Based on these functions, we create a logical architecture and, finally, a problem-specific technical architecture. This procedure must be repeated from scratch for each specific system. Therefore, in particular for organizations that frequently design similar systems, reuse of existing solutions promises a reduction in effort and the possibility to make experiences and knowledge available to future projects or even across organizational borders. Various reuse approaches can be classified. For example, VDI/VDE 3695 defines, among other things, reference models or architectures as one possible way of enabling reuse of artifacts within the engineering of systems [VDI/VDE 3695 2010].

Reference architectures are a reuse approach for organizations that expect to build similar systems in the future and already have good knowledge of these systems. They are used as blueprints for future systems and may be adapted for specific systems. In addition, reference architectures may be also applied when designing specific CSGs (e.g., for the adaptable and flexible factory) to define the necessary roles, system, and collaboration functions of CESs but also protocols, data structures, etc. to enable collaboration within this CSG. Different organizations may subsequently use this reference architecture to design CESs which may collaborate in these CSGs.

In this section, we present a method for designing reference architectures for CESs and CSGs. In addition, we give a short insight into a reference architecture for adaptable and flexible factories. This reference architecture is based on a general reference architecture for CESs and CSGs.

Definition of reference architecture and architecture framework

A reference architecture is defined as "the outcome of applying the architectural framework to a class of systems to provide guidance and to identify, analyze and resolve common, important architectural concerns. A reference architecture can be used as a template for concrete architecture of systems of the class" [Lin et al. 2017]. Complementing this, an architecture framework is defined as "conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders" [ISO/IEC/IEEE 42010 2011]. The SPES_XT modeling framework (see Chapter 2) is an example of such an architecture framework and is used in the following for designing reference architectures as well as system architectures.

3.2.1 Method for Designing Reference Architectures

The general procedure for designing reference architectures and deriving system architectures from reference architectures is shown in Figure 3-1. While a reference architecture is created only once, numerous system architectures can be derived from a single reference architecture. The transitions between the viewpoints in Figure 3-1 show the general procedure for designing reference and system architectures.

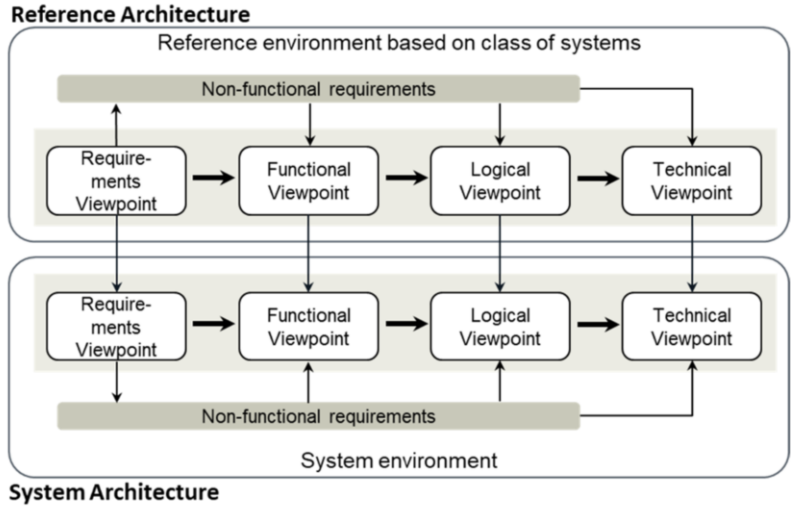

Fig. 3-1: *General approach for designing reference and system architectures*

Non-functional requirements

In addition, the role of non-functional requirements (e.g., requirements related to safety), which are elicited in the requirements viewpoint, is highlighted. In some cases, these requirements cannot be assigned to single functions or to logical or technical components and should therefore be revised regularly during the design of reference as well as system architectures — this is indicated by the arrows related to the non-functional requirements in the figure above. In Section 3.4, we provide a method for integrating safety cases into reference or system architectures to provide an approach for safety-related requirements.

Finally, in Figure 3-1, the arrows from the reference architecture viewpoints to the system architecture viewpoints indicate the reuse of design results for designing system architectures. However, it may be necessary to adapt or complement the reference architecture content.

Scope definition

As a first and critical step within the requirements viewpoint for defining reference architectures, we define the scope of systems for which the reference architecture will be defined. This means that we need to forecast the future systems for which we want to use the reference architecture as a blueprint.

Key design decisions for reference architectures

Next, we determine which kind of reference architecture we want to design. There are several key design decisions that have to be taken, for example:

- *Coverage*: Reference architectures can, for example, cover a common core of all considered systems, offer combinable and reusable building blocks, or provide a solution that will cover all requirements of all considered systems and is then tailored to fit to one specific system.
- *Extensibility*: Reference architectures may, for example, allow white box extensibility, which means that its components can be fully adapted. On the other hand, only black box reuse that does not allow any internal modifications may be allowed. Other forms include grey box reuse, which is a mixture of both.
- *Granularity*: The level of granularity of the reference architecture must also be decided. The goal is to be as detailed as possible while still covering the future system architectures for the intended set of systems. A reference architecture may, for example, define only interfaces of systems or components or provide a full detailing of all systems.
- *Viewpoints*: Consequently, the reference architecture may define views of the requirements viewpoint only, or also comprise views of the functional, logical, technical, and other viewpoints. While a reference architecture that covers all viewpoints would appear to be the best option, it also allows less changeability or requires more effort if there are frequent changes.

These key design decisions mainly depend on the similarity of the set of systems and their requirements.

Further requirements elicitation considers the scope of systems

Subsequently, further requirements are elicited for the reference architecture based on the decisions made above. In addition, even requirements of the set of selected systems that are not implemented by the reference architecture may have to be considered to prepare their later implementation. For collaborative systems in particular, the CSG must be considered as well as the CESs — for both CSG and CES design. This results from the general concept described in Chapter 2. If a CES is to contribute to different CSGs, all relevant CSGs have to be involved.

Functional architecture

On this basis, we then extract the necessary functions of our reference architecture while also considering the collaboration and system functions for both CSGs and CESs. It is important to keep the relations between requirements and functions, and further on, to logical and technical components, as traces. These traces allow us to check, for example, whether all requirements are implemented by functions or logical and technical components. Vice versa, in the case of changes to the technical solution, the traces also enable us to check whether all requirements are still fulfilled.

Logical architecture

Based on the functional architecture, we create a logical architecture for the set of selected systems. Within this logical architecture, the CSGs are usually logical components and are composed by the CESs.

Technical architecture

Finally, a technical reference architecture may be created. Since CSGs are virtual, the collaboration and system functions have to be implemented by the CESs. For all architecture viewpoints, it is crucial to document design decisions and trace the relationships between the different viewpoints and between the elements in the viewpoints. We then refine any viewpoints as far as possible.

Deriving system architectures from reference architectures

Once the reference architecture is created, we can use it to derive system architectures for future systems. Again, we need to elicit requirements but now for a specific system we want to build. We then compare these requirements with the requirements for our reference architecture and identify similarities as well as differences. Subsequently, we assess these similarities and differences while keeping in mind the parameters for our reference architecture. By using the traces between all architectural components, we can then customize the reference architecture by following the traces and adjusting the elements with divergent or refined requirements — if our extensibility concept permits these adaptions. In addition, we have to integrate new requirements which have not been considered in the reference architecture but are needed for the specific system [Unverdorben et al. 2019].

Example 3-2: Using a Reference Architecture as a Template

Imagine a reference architecture for a factory which includes a requirement to display all alarm data to operators to allow them to recognize critical situations and ensure smooth production. However, for one specific factory, the data will be analyzed first to identify critical situations and only decision-relevant data will be displayed to the operators.

Since just one requirement has changed, we still want to use the reference architecture for this factory. Therefore, we identify the changed requirement in the reference architecture and follow the traces to related requirements (e.g., alarms will be displayed in a flat list), functions, and logical and technical components. In our example, we find that all requirements dedicated to the data collection are still applicable and the related functions and logical and technical components can remain unchanged. However, the requirements that address data preparation for the operator must be replaced by, firstly, data analysis and, secondly, an adapted user interface for the operator. This affects the related functions but also the logical and technical components. For example, an additional data analysis function is introduced which is assigned to a logical data analysis component. In the technical solution, this logical component is realized by an additional software component.

Note that any changes to the original reference architecture during derivation of a system architecture must be reflected on carefully since they may indicate improvements for the reference architecture. Thus, continuous feedback from system architecture design to reference architecture design is important for keeping the reference architecture up to date. In the case of changes to the reference architecture, there must also be an update concept for existing systems based on a prior version of the reference architecture.

To use the method described above successfully, tool support for modeling reference and system architectures is useful. [Böhm et al. 2020] introduces a modeling tool which implements this method.

3.2.2 Application Example: Reference Architecture for Adaptable and Flexible Factories

For adaptable and flexible factories, we created a reference architecture using the method described above. The focus is on core requirements and the reference architecture must cover the requirements, functional, and logical viewpoints. Since we want to be independent from any specific technical solution, the objective is not a technical reference architecture.

Requirements for the adaptable and flexible factory

The adaptable and flexible factory was already introduced in Chapter 1. In order to extend the requirements for such a factory, we used the application scenarios described in [BMWi 2017a] and [BMWi 2017b] as a basis: the main goal of the factory is to produce products. Incoming product orders must be analyzed in terms of required capabilities and compared with available capabilities within and, optionally, across factories (see also Section 6.4.2). The factory might need to reconfigure its production and, eventually, produces the

product. Besides this basic production process, we assumed that a need for high capacity utilization and guaranteed delivery dates requires production planning. Other goals of the factory are optimization of production, integrated maintenance, collaboration in marketplaces, and continuous development of its product portfolio.

In addition to the application scenarios, requirements arose from the use cases described in this book and the concepts presented in Chapter 2 as guiding principles. On this basis, we designed a general reference architecture for CESs and their CSGs, which not only considers the general concepts but also refines, for example, collaboration and system functions and, subsequently, the logical architecture.

We then created our reference architecture for adaptable and flexible factories. Figure 3-3 shows a basic diagram of the logical reference architecture which presents the CSGs identified, which are derived from the base CSG at the top.

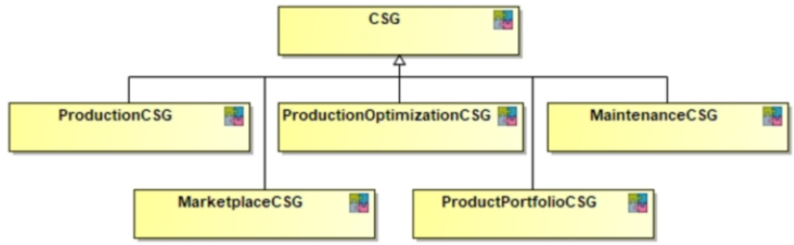

Fig. 3-3: *Refinement of CSG for adaptable and flexible factories*

The CSGs within the reference architecture for adaptable and flexible factories have the following goals and define, accordingly, the following functions:

- *ProductionCSG*: The goal of this CSG is the manufacture of a product specified within a production order. For this purpose, it realizes functions for analyzing incoming product orders with respect to producibility and additional constraints such as delivery dates, price, etc. It also contains functions, for example, for maintaining a production plan for this product, tracking the production, and collecting data for operation control.
- *ProductionOptimizationCSG*: The main goal of this CSG is to optimize the production of the factory. Therefore, it realizes operator support functions—for example, detecting bottlenecks, failures, or unused capacities in production—and deduces measures based on these observations. A close interaction between this CSG and the operator is crucial and may be realized

by an operator assistance system as part of this CSG. This CES is described in more detail in Section 3.3.

- *MaintenanceCSG*: In order to keep the factory productive and in a good state, this CSG defines functions related to preventive and reactive maintenance, as well as maintenance planning and implementation.
- *MarketplaceCSG*: This CSG ensures collaboration between adaptable and flexible factories by offering production capabilities available in the factory and requesting external capabilities via marketplaces.
- *ProductPortfolioCSG*: The goal of this CSG is the continuous development of the factory in order to, for example, reach a high capacity utilization. For this purpose, it combines functions for analyzing missing production functions according to recent product orders, detecting possible improvements (e.g., based on current bottlenecks), and suggesting corresponding measures, etc.

For these CSGs as well as for CESs within the adaptable and flexible factory, the logical architecture was detailed further.

Application to demonstrator

We also used the reference architecture for a factory model demonstrator to derive a specific logical system architecture and to define a technical architecture on top. This pilot showed that the reference architecture is a good basis for deriving system architectures, provided that the underlying general concept is applicable.

3.3 Reference Architecture for Operator Assistance Systems

In Subsection 3.2.2, we identified a CSG for production optimization for adaptable and flexible factories. A central CES contributing to this goal is an operator assistance system. It manages the collaboration of the various CESs in the CSG and offers an interface to the human operator. The CESs being handled by the operator assistance system comprise production machines providing data and they are controlled by the operator, planning and management tools, and additionally model- and data-based evaluation services such as simulation and optimization. These CESs must be combined dynamically in a context- and situation-specific manner. In this section, we now want to take a deep dive into a technical reference architecture for an operator assistance CES.

3.3.1 Simulation-Based Operator Assistance

Simulation can help to optimize production

It is a challenging task to operate adaptable and flexible systems, such as production plants in discrete manufacturing and process industries or connected infrastructure systems such as energy and water grids. The need for more flexibility in operation grows with a higher variety of products, smaller lot sizes, and fluctuating markets. Despite an increasing degree of automation, there are still many decisions to be made by human operators in a short time that target various aspects such as cost, time, and quality. Specific data- and simulation-driven operator support applications can help to handle the task [Boschert et al. 2018], [Rosen et al. 2018]. A digital twin, that is, a virtual replica of the physical system, connects data from different sources and models from different hierarchies. It can form the core of intelligent operator assistance systems [Rosen et al. 2019].

Reference architecture as enabler for low-code assistance system development

Today, integrating simulation and digital twin approaches into operation support for complex systems is still a time-consuming and resource-intensive, typically customer- and project-specific task. You need automation, software, simulation, and domain experts to do this. Therefore, we want to present a technical reference architecture that can support the development of such assistance systems. By using the reference architecture, operator assistance systems can be easily realized on a low-code and low-modeling base and development time can be reduced significantly.

Assistance systems need to be very flexible

One of the main challenges for the development of an operator assistance CES is that it requires a high degree of flexibility: the CES provides different applications such as virtual monitoring and short-term prediction and optimization on different levels such as machine, line, and factory level, and can run in different situations such as normal operation and failure situations. This imposes the need for flexible, situation-specific collaboration of calculation modules and multiple use of data and models.

The concept of a reference architecture for an operator assistance CES will be outlined in the following. For more details, the reader is referred to [Zhou et al. 2019].

3.3.2 Design Decisions

Reference architecture contains execution core and collections of basic elements

We make the following key design decisions for the operator assistance reference architecture:

- *Scope*: We consider simulation-based assistance systems for the operation of adaptable and flexible factories.

- *Coverage*: We cover a common core with generic metamodels and an execution engine to run configurable workflows of evaluations and an extendible collection of re-usable data interfaces, evaluations, and user interface (UI) elements.
- *Extensibility*: The common core is limited to black box reuse in order to guarantee interoperability of services in arbitrary workflows, for different assistance functions, across different plants, and over time. Full white box extensibility is provided for the collections of data interfaces, evaluations, and UI elements.
- *Granularity and viewpoints*: A detailed technical architecture is set up since we aim to implement the architecture as a software framework for the future development of operator assistance systems.

3.3.3 Technical Reference Architecture

Modular, service-oriented architecture and configurable workflows

The technical reference architecture which is finally derived from the design decisions described in Subsection 3.3.2 and additional requirements implements a concept of a service-oriented architecture, model-based data structures and flows, and generic but customizable UI components.

System functions are divided into encapsulated, exchangeable, and configurable sub-functions. These sub-functions or services can be recombined in many ways to create various workflows which offer different assistance functions. For seamless data exchange between all services, a common component-based metamodel is introduced which is most notably suited for model-based services such as simulation and optimization.

The architecture for operator assistance systems can be divided into three horizontal layers: the data layer, the service layer, and the UI layer, see Figure 3-4. The technical reference architecture provides generic implementations of the core elements in this architecture: the execution engine calling services as specified in workflows, a UI backend, and a data management based on metamodels for component libraries, plants, and workflows.

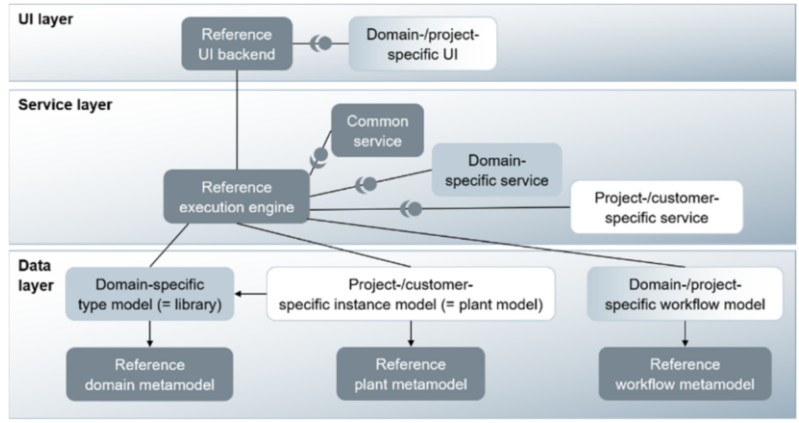

Fig. 3-4: *General technical architecture for an operator assistance system*

Reuse of reference architecture reduces development effort

When implementing a specific operator assistance system, these reference architecture elements form the base. Starting from there, firstly, unspecific or domain-specific frontloading and, finally, project- or customer-specific engineering is performed, see Figure 3-5. Implementing new services or new adapters for existing computational modules such as simulation tools is part of the frontloading. With an increasing number of domains and projects addressed, the reference architecture becomes more elaborate and the collection of reusable services grows. The effort is shifted away from software implementation towards model engineering and system configuration: specifying domain libraries, setting up workflows and data contracts, generating plant models, and configuring UIs. The complete development process is further facilitated by defined process steps, toolkits, and many templates.

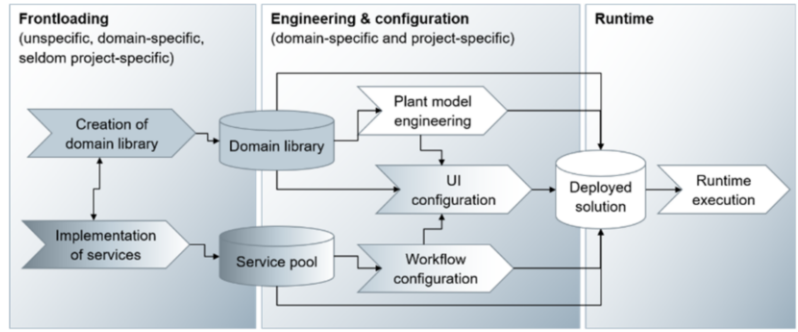

Fig. 3-5: *General steps of the reference-based development process*

3.3.4 Workflow of Services and Data Flow

Collaboration of services via common runtime model

The reference architecture strictly separates the logical and sequential workflow of services and the data flow during runtime execution, as shown for a generic workflow in Figure 3-6. There is no bilateral data exchange between the services. Each service communicates only with the current runtime model and does not know about the source and the destination of any specific variable value. This ensures consistency of data during the whole workflow, simplifies configuration of workflow sequences and data contracts, and guarantees flexibility to replace individual services.

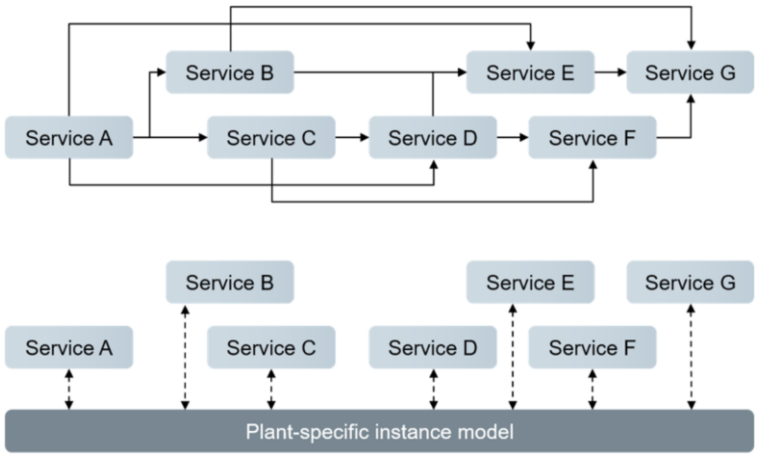

Fig. 3-6: *Workflow (upper part) and data flow (lower part)*

3.3.5 Application Example for an Adaptable and Flexible Factory

Reduced development cost for operator assistance in adaptable and flexible factories

The technical reference architecture presented was implemented as a software framework which was successfully applied in the development of a prototypical assistance system for the operation of an adaptable and flexible factory. The prototype system integrates data from an enterprise resource planning (ERP) system, from a manufacturing execution system (MES), and machine data via the standard communication protocol OPC UA. It contains functions for virtual monitoring of the production, online calibration of the models, detection of any failures and deviations, prediction of critical situations such as bottlenecks, and job shop and flow shop schedule optimization. Figure 3-7 shows the workflows of three of these functions and illustrates how services are reused and re-combined to

offer various functions. Development time was significantly reduced compared to a project- and task-specific development by using the reference architecture as the starting point and core of the system.

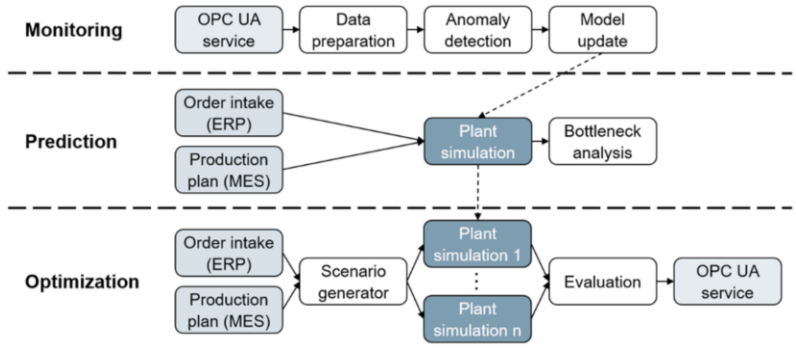

Fig. 3-7: *Workflows for three different assistance functions*

3.4 Checkable Safety Cases for Architecture Design

In this section, we introduce a method for safety argumentation in the design of system and reference architectures. Safety requirements are crucial for CESs and CSGs that may harm people, equipment, or the environment. Adaptable and flexible factories are a typical example of safety-critical systems. Our goal is to support the construction and maintenance of the argumentation that the system architecture of a flexible system satisfies the system safety properties. To this end, we introduce checkable safety cases.

Systems implementing safety functionality that will operate safely in a given operational context must be proven. To this end, more and more safety standards nowadays, such as ISO 26262 [ISO 2018] in the automotive industry, recommend the creation of a safety case. A safety case is a collection of documents entailing an implicit, well-reasoned argument that the system is acceptably safe to operate in a given context, based on certain evidence [Bloomfield and Bishop 2010]. To enable the automated manipulation of safety cases, several approaches for modeling safety cases have been proposed in literature, the most prominent approaches being based either on the Structured Assurance Case Metamodel (SACM) [SACM 2019] or the Goal Structuring Notation (GSN) [GSN 2018].

The validity of the safety case models must be revised every time there is a change in the system specification. However, currently, such validity revision is done manually, implying a considerable amount of effort and costs. Given the frequent changes to architectural

structures of flexible systems, there is a need to automate validity checks for safety cases. To this end, we introduce checkable safety case models with the scope of supporting safety engineers in maintaining valid safety case models given changes in other system models. Checkable safety case models are a special type of safety case model that is integrated with system models, which are amenable to automated checks.

Extension of the SPES_XT modeling framework

To this end, we extend the SPES_XT modeling framework with a new system view, that is, the safety case view. The safety case models are to be integrated with the other system models corresponding to different viewpoints (e.g., requirements viewpoint, logical viewpoint). The safety case model is to be modeled alongside the system development and will be maintained to ensure consistency with other system models during the entire system lifecycle.

To support safety engineers in modeling checkable safety cases, we propose a set of checkable safety case patterns. Similar to design patterns, safety case patterns are templates for re-occurring safety fragments that can be reused in different safety cases [Kelly and McDermid 2010]. These templates entail placeholders for system-specific information which are to be filled when the pattern is used in a certain safety case. We extend the concept of safety case patterns with checkable safety case patterns. Checkable safety case patterns come with a set of automated checks that may be performed on the safety case fragment obtained after the instantiation of the pattern.

Modeling checkable safety case fragments for reference architectures

Among other things, the safety case of a system must entail an argument about the satisfaction of safety properties by the system architecture. As reference architectures are blueprints to be used for modeling system architectures, for each such reference architecture we provide a pattern for arguing about the fact that the reference architecture satisfies certain safety properties. When the architecture of a certain system uses a certain reference architecture as a blueprint, the corresponding safety case checkable pattern can be used to model the safety argumentation for the constructed system architecture.

3.4.1 Checkable Safety Case Models – A Definition

Safety case models on which automated checks can be executed

To support safety engineers in the cumbersome, time-consuming process of keeping safety case models consistent with system models (e.g., system architecture models), we propose checkable safety cases.

The validity of checkable safety case models is checked by the automatic execution of sanity checks, based on explicit specification of semantics of safety case elements, and the integration of the safety case model with system models and automated verification approaches [Cârlan et al. 2019], see Figure 3-8.

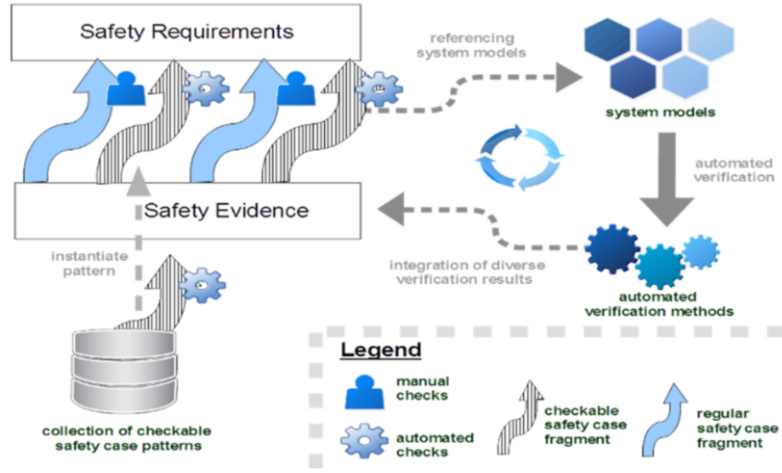

Fig. 3-8: *Safety argumentation based on contract-based verification*

Given a change in a system model that is traced from the safety case model, consistency checks between the safety case model and the system models are automatically executed. These consistency checks assess whether the argumentation is still valid considering the changes in the system model that the argumentation applies to. Then, the safety engineer must update the safety argument in accordance with the changes, while also generating the evidence required. Given that system models are amenable to automated checks, the results of such checks can be used as evidence in safety cases. Therefore, we integrate safety case models with such automated verification approaches, thus enabling 1) automatic detection of stale evidence, and 2) automatic integration of new verification results as evidence, while assessing the impact of the new evidence on the confidence in the overall argumentation.

Checkable safety cases entail checkable and non-checkable argumentation fragments

Checkable safety case models entail both checkable and non-checkable argumentation fragments that are connected with each other. On the one hand, non-checkable argumentation fragments entail regular safety case elements, as defined by the Goal Structuring Notation (GSN) — a standardized graphical notation for describing safety cases and currently the most frequently used language for

modeling safety cases [GSN 2018]. On the other hand, checkable safety case fragments entail a set of interconnected specialized safety case elements. Specialized safety case elements extend GSN, with each specialized element representing a reoccurring claim in safety cases, thus having certain semantics. Specialized safety case elements reference certain types of system model elements or entail metadata regarding certain verification approaches. They may be connected to each other only via specialized connections, which extend the connections specified in GSN. In contrast to GSN-based connection types that ensure the correct construction of arguments from a semantic point of view, specialized connections enable intrinsic checks on safety case models, which ensure the construction of semantically correct arguments.

3.4.2 Checkable Safety Case Patterns

To support safety engineers in modeling checkable safety cases, we propose an exemplary set of checkable safety case patterns.

Checkable safety case patterns enhance state-of-the-art patterns to enable automated checks

While the argumentation structure of checkable safety case patterns is based on state-of-the-art patterns, the connected elements the structure contains are specializations of regular safety case elements. The specialized safety case elements have variable declarations, which are placeholders for a reference to a certain type of system element or verification information. The variables are to be instantiated with specific references when the pattern is used to model the safety case of a certain system. The relationships among specialized safety case elements are described via dedicated connections, thus enabling intrinsic consistency checks, which prohibit pattern misuse — a specialized safety case element may be connected only to certain types of other specialized safety case elements.

A checkable safety case pattern is specified as presented in the following [Kelly and McDermid 2010]. We extend the specification of regular safety case patterns with information specific to checkable safety case patterns:

- *Name*: the identifying label of the pattern giving the key principle of its argument
- *Intent*: the goal the pattern is trying to achieve
- *Motivation*: the reasons that gave rise to the pattern and the associated checks
- *Structure*: the structure of the argument in GSN

- *Participants*: each element in the pattern and its description; here we differentiate between plain SACM-based elements and specialized elements — for the specialized elements, the corresponding metadata is explained
- *Collaborations*: how the interactions of the pattern elements achieve the desired effect of the pattern; here we explain the specialized connections among the specialized elements and how the specialized safety case elements will be connected with the regular elements
- *Applicability*: the circumstances under which the pattern could be applied, that is, the necessary context
- *Consequences*: what remains to be completed after pattern application
- *Implementation*: how the pattern should be applied; here we discuss how the safety case elements are to be instantiated

The following documentation information is specific to checkable safety case patterns:

- *Prerequisites*: regarding the existence of certain system models or of certain verification tools
- *Automated checks*: the checks that can be executed on the safety case fragments produced after the instantiation of the pattern

3.4.3 An Example of Checkable Safety Case Patterns

Arguing about the satisfaction of a certain safety property by an architecture

In Figure 3-9, we present part of the checkable safety case fragment concerning the satisfaction of a certain safety property by a system architecture built in a contract-based manner. The system architecture entails assume-guarantee (A/G) contracts that formalize safety properties. The properties are satisfied if: 1) the contracts of the architecture model are correctly refined by the contracts of the components within the architecture model (claim expressed as *Refinement Check* specialized goals); 2) the contracts of the architecture components are satisfied (claim expressed as *Compatibility Check* specialized goals); and 3) each architecture component correctly implements its contracts (claim expressed as *Implementation Check* specialized goals). Each claim in the argument is a specialized safety case element, with a certain meaning and with certain references to system model elements. Given specialized connections between specialized elements, intrinsic consistency checks are enabled. For example, elements of the type *CBD Strategy* may be supported only by goals of the type *Compatibility Check*,

Refinement Check, and *Implementation Check*, ensuring the validity of the argument structure. The *CBD Strategy* references a certain component in the system architecture that will implement the safety contract. Consequently, to ensure the validity of the argumentation, we check whether the sub-goals of the type *Implementation Check* supporting *CBD Strategy* reference only children of the component referenced by the strategy. The validity of claims of the type *Implementation Check* is checked via an automated verification tool able to check architecture models annotated with contracts — a model checker. In the example presented in Figure 3-9 the model checker used is NuSMV [Cimatti et al. 2002].

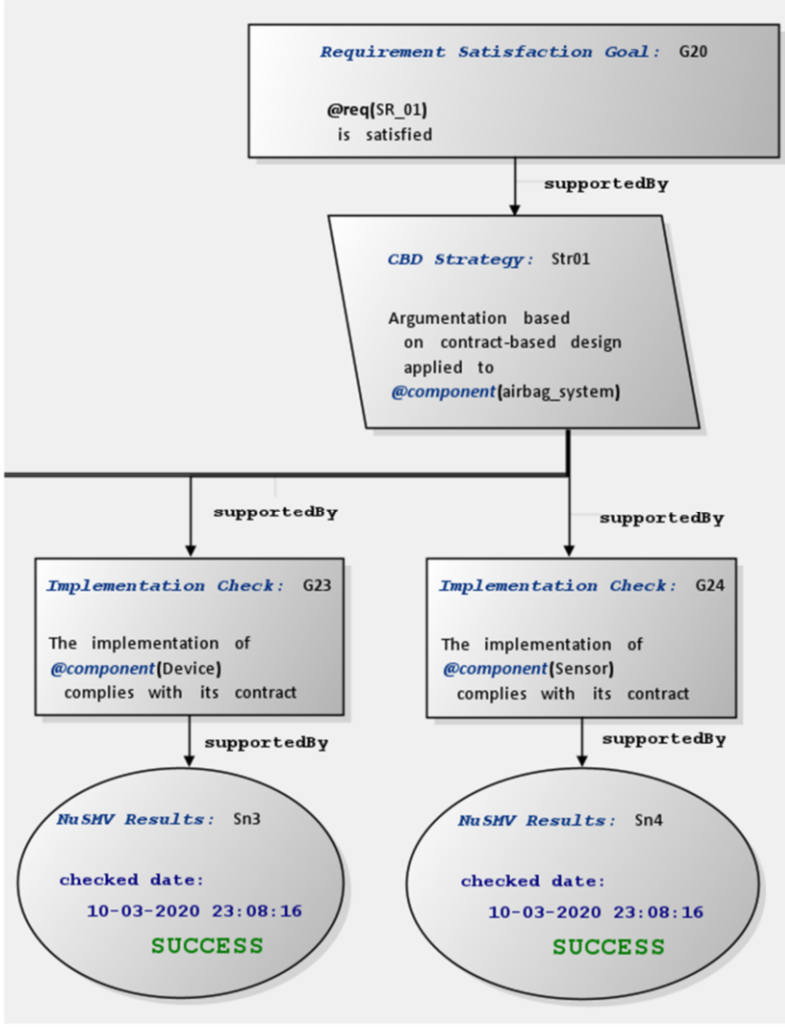

Fig. 3-9: *GSN-based safety case fragment*

In Figure 3-9 a GSN-based safety case fragment is shown arguing about the verification via NuSMV model checker of the system architecture model against system safety properties specified as contracts. Due to space constraints, the figure displays only part of the argumentation, namely the argumentation legs regarding the correct implementation of the subcomponents of the architecture.

In conclusion, we propose the creation of checkable safety case patterns that argue about the implementation of safety properties by a system architecture which may also be based on a certain reference architecture. Given the specialized safety case elements contained in the pattern and their integration with system models and verification tools, the validity of the argumentation fragment resulting from the pattern instantiation is automatically checked if there is a change in the corresponding system architecture model. These automated checks are especially needed if there are frequent changes.

3.5 Conclusion

In this chapter, we presented a general method for designing reference architectures and deriving system architectures for CESs and their CSGs in order to support reuse of system architectures. In addition, the method can be used to design a CSG and the interfaces of collaborating CESs within this CSG. In a next step, the architectures of the CES can be refined based on the reference architecture. This enables the integration of CESs of different organizations within one CSG. As an application example, we provided a short overview of the reference architecture for adaptable and flexible factories, detailed by a CES implementing an operator assistance system. The technical reference architecture for this CES shows the reuse potential for various operator assistance systems and provides a promising basis for future systems.

In order to consider non-functional requirements in the system architecture, we also introduced checkable safety case models. These checkable safety cases support maintenance of the validity of safety case models and keep them consistent with system architecture. This method may be used for the construction of the safety argumentation system architectures based on reference architectures.

In addition to the methods presented, we also developed prototypical tools which support and facilitate the application of the methods. The methods and reference architectures presented in this chapter have been applied successfully but should nevertheless be

applied to other CESs and their CSGs to prove their benefits. Future research may extend them beyond their current scope, for example, by involving artificial intelligence as design support as well as considering artificially intelligent CESs and CSGs in particular.

3.6 Literature

[Bloomfield and Bishop 2010] R. Bloomfield, P. Bishop: Safety and Assurance Cases: Past, Present and Possible Future – an Adelard Perspective. In: *Making Systems Safer,* Springer, London, 2010, pp. 51-67.

[BMWi 2017a] BMWi: Platform Industrie 4.0 – Aspects of the Research Roadmap. In Application Scenarios. https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/aspects-of-the-research-roadmap.pdf; accessed on 07/07/2020.

[BMWi 2017b] BMWi: Platform Industrie 4.0 – Fortschreibung der Anwendungsszenarien der Plattform Industrie 4.0. https://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/fortschreibung-anwendungsszenarien.html; accessed on 07/07/2020 (available in German only).

[Böhm et al. 2018] B. Böhm, M. Zeller, J. Vollmar, S. Weiß, K. Höfig, V. Malik, S. Unverdorben, C. Hildebrandt: Challenges in the Engineering of Adaptable and Flexible Industrial Factories. In: I. Schaefer, L. Cleophas, M. Felderer (eds.): Workshops at Modellierung 2018, Modellierung 2018, Braunschweig, Germany, February 21–23, 2018, pp. 101–110.

[Böhm et al. 2020] B. Böhm, J. Vollmar, S. Unverdorben, A. Calà, S. Wolf: Holistic Model-Based Design of System Architectures for Industrial Plants. In: VDI – Verein Deutscher Ingenieure e.V. (eds.): Automation 2020, Baden-Baden, 2020.

[Boschert et al. 2018] S. Boschert, R. Rosen, C. Heinrich: Next Generation Digital Twin. In: Proceedings of the 12th International Symposium on Tools and Methods of Competitive Engineering — TMCE 2018, Delft, 2018, pp. 209-218.

[Cârlan et al. 2019] C. Cârlan, V. Nigam, S. Voss, A. Tsalidis: ExplicitCase: Tool-Support for Creating and Maintaining Assurance Arguments Integrated with System Models. In: Proceedings of *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW),* 2019, pp. 330-337.

[Cimatti et al. 2002] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: *Computer Aided Verification. CAV 2002.* Lecture Notes in Computer Science, vol 2404. Springer, Berlin, Heidelberg.

[de La Vara et al. 2016] J. L. de La Vara, M. Borg, K. Wnuk, L. Moonen: An Industrial Survey of Safety Evidence Change Impact Analysis Practice. In: *IEEE Transactions on Software Engineering*, *42*(12), 2016, pp: 1095-1117.

[GSN 2018] Assurance Case Working Group. Goal Structuring Notation Community Standard (Version 2). https://www.goalstructuringnotation.info/; accessed on 01/11/2020.

[ISO/IEC/IEEE 42010 2011] International Organization for Standardization, International Electrotechnical Commission, Institute of Electrical and Electronics Engineers (eds.): Systems and Software Engineering - Architecture Description.

Geneva, s.l., New York, http://ieeexplore.ieee.org/servlet/opac?punumber=6129465; accessed on 04/04/2020.

[ISO 2018] International Organization for Standardization (ISO): 26262: Road Vehicles - Functional Safety.

[Kelly and McDermid 2010] T. Kelly and J. McDermid, "Safety case patterns-reusing successful arguments," IEE Colloquium on Understanding Patterns and Their Application to Systems Engineering (Digest No. 1998/308), London, UK, 1998, pp. 3/1-3/9.

[Lin et al. 2017] S.-W. Lin, M. Crawford, S. Mellor (Eds.): The Industrial Internet of Things Volume G1: Reference Architecture. Industrial Internet Consortium (IIC) Technology Working Group, 2017. http://www.iiconsortium.org/IIC_PUB_G1_V1.80_2017-01-31.pdf; accessed on 07/07/2020.

[Rosen et al. 2018] R. Rosen, S. Boschert, A. Sohr: Next Generation Digital Twin. In: atp magazin, Atp-Mag. 60, 2018, pp. 86–96.

[Rosen et al. 2019] R. Rosen, J. Jaekel, M. Barth, O. Stern, R. Schmidt-Vollus, T. Heinzerling, P. Hoffmann, C. Richter, P. Puntel Schmidt, C. Scheifele: Simulation und Digitaler Zwilling im Engineering und Betrieb automatisierter Anlagen - Standpunkte und Thesen des GMA FA 6.11. In: VDI – Verein Deutscher Ingenieure e.V. (eds.): Automation 2019, Baden-Baden, 2019 (available in German only).

[SACM 2019] Structured Assurance Case Metamodel. URL https://www.omg.org/spec/SACM/About-SACM/; accessed on 04/11/2020.

[Unverdorben et al. 2019] S. Unverdorben, B. Böhm, A. Lüder: Concept for Deriving System Architectures from Reference Architectures. In: 2019 IEEE International Conference on Industrial Engineering and Engineering Management: IEEM2019: Dec. 15-18, Macau/IEEE International Conference on Industrial Engineering and Engineering Management - [Piscataway, NJ]: IEEE, 2019, pp. 19-23.

[VDI/VDE 3695 2010] Association of German Engineers (VDI), Association for Electrical, Electronic & Information Technologies (VDE): VDI 3695 Blatt 3 - Engineering of Industrial Plants - Evaluation and Optimization - Subject Methods. 2010.

[Zhou et al. 2019] Y. Zhou, T. Schenk, M. Allmaras, A. Massalimova, A. Sohr, J. C. Wehrstedt: Flexible Architecture to Integrate Simulation in Run-Time Environment. Presented at the Automation Congress 2019, VDI, Baden-Baden, 2019.