Design-time Specification of Dynamic Modular Safety Cases in Support of Run-Time Safety Assessment

Elham Mirzaei [1], Carmen Cârlan [2], Carsten Thomas [1], Barbara Gallina [3]

[1] HTW Berlin, University of Applied Sciences, Berlin, Germany

[2] fortiss GmbH, Munich, Germany

[3] Mälardalen University, Västerås, Sweden

Abstract *Open Adaptive Complex Systems – such as road vehicle platoons or fleets of cooperative robots – may use dynamic reconfiguration to adapt to system or environment changes. One approach enabling this feature is Service-oriented Reconfiguration, where new configurations are created by composing the available services in an unconstrained manner. Due to the high number of possible service compositions, not all configurations can be pre-assured at design-time. Despite recent progress, there is no satisfactory approach for specifying safety cases in support of their re-evaluation at run-time, after system reconfiguration. To this end, in previous work, we introduced Dynamic Modular Safety Cases (DMSC). A DMSC is a modular safety case, which can be dynamically re-constructed and re-assessed given service reconfiguration. In continuation of the previous work, in this paper we provide guidelines for specifying safety cases at design-time, whose modular structure mirrors the system service decomposition, to enable their re-construction and re-evaluation at run-time in the event of a system reconfiguration. Aiming to support the specification of DMSC, we extend FASTEN, an engineering tool for the design and verification of safety-critical systems. We exemplify the specification of DMSCs in FASTEN for an illustrative example from the smart factory domain.*

1 Introduction

System safety, i.e., the fact that the system deployment does not pose an unacceptable risk of harm, needs to be assured. Assurance here is interpreted as "*grounds for justified confidence that a claim has been or will be achieved*" (ISO/IEC/IEEE 2019). In most safety-critical domains, such as automotive, or nuclear, or rail system safety assurance also assumes the creation of a system safety case. A Safety Case (SC) is defined as "*a reasoned and compelling argument, supported by a body* *of evidence, that a system, service or organisation will operate as intended for a defined application in a defined environment*" (Ministry of Defence 2007). Despite some criticism against SCs (Leveson 2020), they are widely applied in various domains, and their provision may be even required (e.g., air traffic control (Eurocontrol 2006), road and rail transportation (CENELEC 2007)). Typically, SCs are created at design-time and maintained throughout the life of the system.

Open adaptive complex systems (often encompassing Systems of Systems), such as road vehicle platoons within the automotive domain and fleets of cooperative robots within the robotics domain, are characterized by dynamic evolution/reconfiguration and emergent behaviour (Boardman and Sauser 2006). Very often, these systems are safety-critical. Hence, maintaining the safety assurance of such system after dynamic changes that are imposed by reconfiguration is necessary (Kelly 2003).

In the literature, reconfiguration is often classified into three types, namely *predefined*, *constrained,* and *unconstrained selection* (Bradbury, et al. 2004). Unconstrained selection provides more flexibility in terms of adaptation at run-time amongst all possible variations for creating a new configuration. Open adaptive complex systems use reconfiguration to adapt their structure and behaviour to changes in constituent systems or in their environment. In earlier work, we have introduced the Service-oriented Reconfiguration (SoR) approach (Wudka, et al. 2020) (Thomas, et al. 2021), which supports unconstrained reconfiguration at run-time based on the Service-oriented Architecture (SoA) concept and blueprints.

The SCSC[1] Service Assurance Working Group (SAWG) provides guidance on challenges related to the safety assurance of services, e.g., inter-service interference, mapping from service decomposition to modules within modular assurance, and deviation to the reference architecture due to the change in configuration during deployment (SAWG 2020). They highlight the necessity of extending beyond the traditional approaches in system safety engineering to address those challenges, considering that the future developments in business and technology are likely to adopt this service paradigm in the next generation of safety-critical complex systems.

In the context of reconfiguration classifications, unconstrained reconfiguration is the most challenging approach from the safety assurance perspective. This is mainly because, in the current practice, an SC is developed manually, by a safety engineerat design-time, compiling evidence produced during the execution of safety assurance activities. As the argumentation structure comprised by the SC largely depends on the system configuration, the current manual approach for SC development at design-time is inappropriate for SoR-based systems, for which knowing all possible configurations prior to operation is difficult. There is a need to enable automated development and assessment of the system safety case at run-time, considering the run-time system reconfiguration.

[1] Safety Critical Systems Club

To this end, in our previous work, we introduced the DMSC approach (Mirzaei, Thomas and Conrad 2020), which combines two state-of-the-art approaches for developing safety cases: Modular Safety Case (MSC) and Dynamic Safety Case (DSC). MSC address challenges such as system complexity (Kelly 2003) and frequent system evolution by breaking down a safety case into several connected modules, and DSC aim at re-evaluating the validity of design-time safety assumptions at run-time (Denney, Pai and Habli 2015). The DMSC approach unites the modular structure concept and the concept of dynamic update and re-evaluation at run-time.

In this paper, we provide guidelines for constructing DMSC. The core idea behind these guidelines is to construct the SC at design-time in a manner that enables automatic SC re-construction and re-evaluation at run-time. Further, we also show how FASTEN – an engineering platform for the creation and maintenance of safety cases – has been extended to support the specification of DMSC and the management of their relations with the structural elements of open adaptive complex systems. Finally, we show how to develop DMSC following our proposed guidelines and using the FASTEN tool for an illustrative example from the smart factory domain.

The rest of this paper is organized as follows. In Section 2, we discuss the language we use in this paper for the specification of SC, and we briefly describe the reconfiguration approach we consider, highlighting the requirements it imposes with respect to the system safety assurance. Section 3 provides a brief overview of available methods for the creation and management of the SC. Section 4 outlines our proposed solution for the specification of DMSC. In Section 5, we present tool-support for the proposed solution, and we exemplify its usage for a small use case involving two robots in a factory. Finally, in Section 6, we summarize our contributions and outline the next steps.

2 Fundamentals

2.1 The GSN/SACM Metamodel

To better structure the SC, graphical notations have emerged over the past years, one of the most frequently used notations being the Goal Structuring Notation (GSN) (ACWG-GSN 2021). In GSN, structured SC arguments are constructed by goals, strategies, solutions, assumptions, and context definitions. Usually, a top-level safety goal is first defined, which is later decomposed to sub-goals and the step of goal decomposition is re-iterated until the evidence for the satisfaction of the sub-goals is referenced.

GSN supports the specification of MSC (Industrial Avionics Working Group 2012), a concept on which our DMSC approach relies, by introducing away goals, which are repeating a claim presented in other modules, thereby creating a reference from one argument to another. Fig.1 presents an MSC using GSN elements.

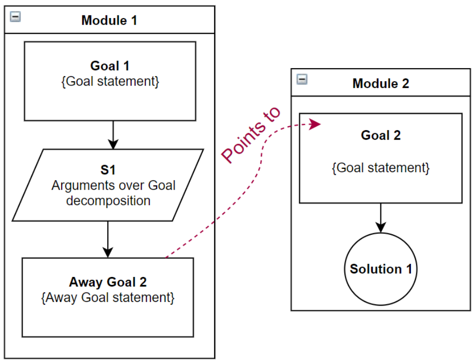


Fig. 1. A Modular Safety Case modelled in GSN using an Away Goal

MSC have been proposed to address challenges such as system complexity (Kelly 2003) and frequent system evolution by breaking down a safety case into several connected modules. If the safety case modularization is done appropriately, the modular structure of the safety case limits the change impact propagation only to a certain part of the safety case. Consequently, to update the safety case in the event of system changes, it may be only necessary to change certain modules, rather than the entire SC. This leads to the reduction in the cost and efforts on SC changes (Kelly and Bates 2005).

To improve standardization and interoperability between tools which support the modelling of GSN-based safety cases, the GSN/SACM metamodel was introduced (ACWG-GSN-MM 2021). GSN/SACM maps the GSN constructs to the SC elements described by the Structured Assurance Case Metamodel (SACM) (Object Management Group 2020) proposed by the Object Management Group (OMG). Among others, the GSN/SACM metamodel allows the specification of traces from safety case elements to other artefacts. As an example for the relation between the GSN notation and SACM, see the mapping of a GSN SC fragment (Goal G1 in Fig.2) to its SACM representation (Fig.3), taken from (ACWG-GSN-EX 2021).

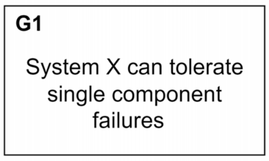


Fig. 2. An example SC fragment using GSN notation (ACWG-GSN-EX 2021)

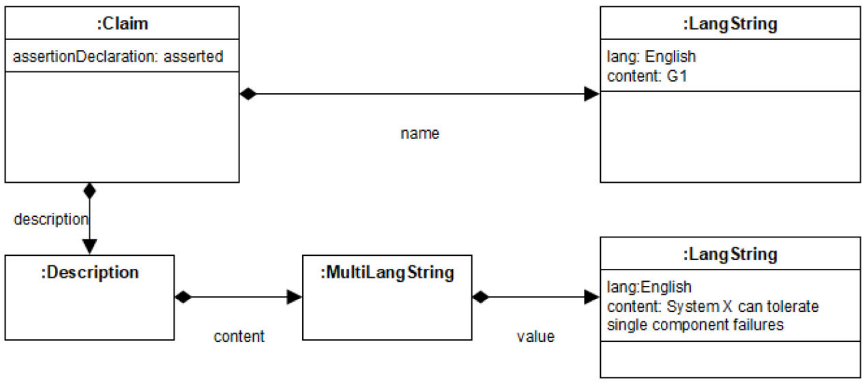


Fig.3. SACM representation equivalent to the SC fragment shown in Fig. 2 (ACWG-GSN-EX 2021)

2.2 Service-oriented Reconfiguration (SoR)

In this subsection, we recall basic elements of our Service-oriented Reconfiguration (SoR) approach to provide the context for the DMSC specifications, which will be introduced in Section 4.

Service-oriented Architecture (SoA) is a software design pattern that supports system modularization and interaction between system components (Richardson 2018). Applying this pattern to open adaptive complex systems (Siefke, et al. 2020), we perceive these as to be composed of constituent systems (e.g., a fleet of robots consisting of individual autonomous robots), where the constituent systems realize their intended functions by sets of interconnected services for sensing, computation and actuation. Open complex adaptive systems may react to internal changes (e.g., malfunction behaviour) and changes in their environment by flexibly adapting the interconnections between services at run-time, within individual constituent systems or across several constituent systems. The SoR approach (Thomas, et al. 2021) builds on the SoA pattern to define a reconfiguration mechanism that – at run-time – creates new configurations for open adaptive systems using service blueprints to specify potential service network configurations.

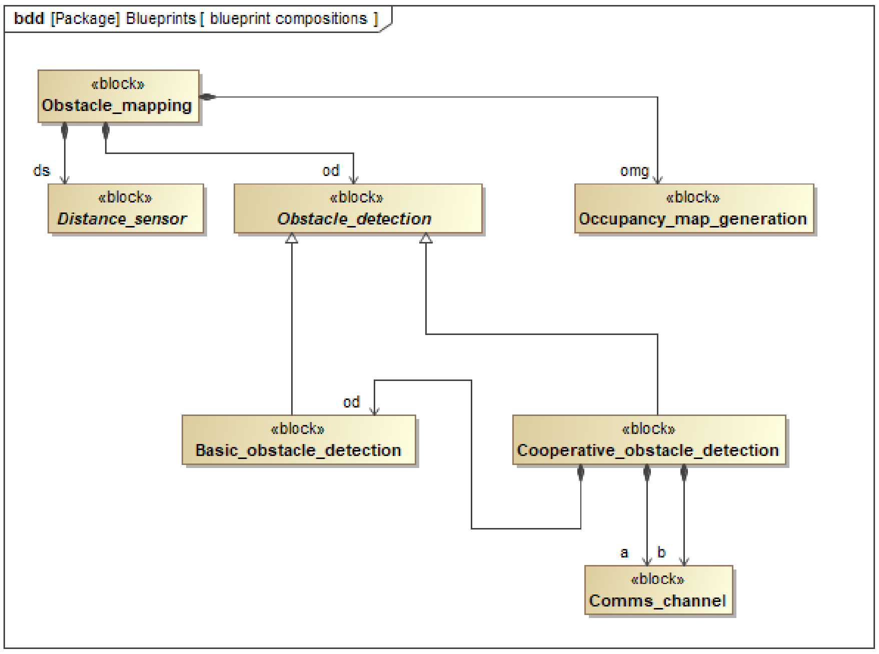

bdd [Package] Blueprints [blueprint compositions]
«block» Obstacle_mapping
ds
od
omg
«block» Distance_sensor
«block» Obstacle_detection
«block» Occupancy_map_generation
od
«block» Basic_obstacle_detection
«block» Cooperative_obstacle_detection
a
b
«block» Comms_channel

Fig. 4. An example service blueprint inheritance and decomposition hierarchy

A service blueprint is a template defining potential services, either as basic services or as service compositions. Depending on the type of blueprints, the specifications may include interface, parameters, internal structure, and other elements. Like classes in object-oriented programming, blueprints build a specialization hierarchy. This concept enables polymorphic instantiation of services (specialized services can be used as substitute of their more generic ancestors) and is the key concept supporting unconstrained reconfiguration at run-time.

Fig.4 illustrates an example of blueprint inheritance and decomposition hierarchy for a specific service named “Obstacle mapping”, which is decomposed into three constituent service blueprints, “Distance sensor”, Obstacle detection”, and “Occupancy map generation”. While in this example “Distance sensor” and “Obstacle map generation” are always basic service blueprint, “Obstacle detection” may be either a basic service blueprint or a service composition blueprint.

SoR is implemented by means of two basic components: *System Discovery* and *Reconfiguration* as illustrated in Fig.5.The availability of service instances is supervised via the *System Discovery* component, which allows all the available services to register themselves as available services at run-time, managing related information such as the corresponding service blueprint. The *Reconfiguration* component uses this information to create new configurations by traversing top-down through the service blueprints and instantiating service compositions and invoking available instances of basic services.

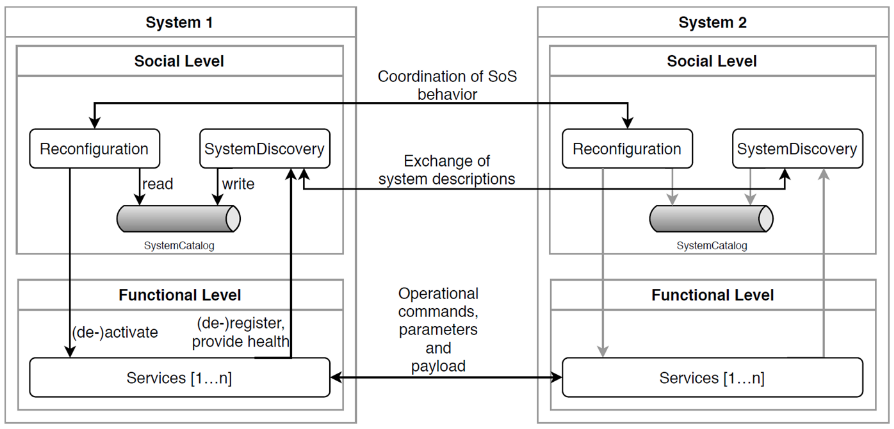


Fig. 5. The SoA-based concept of Service-oriented Reconfiguration (SoR)

In the basic SoR approach, this results in creating a set of possible configurations, from which the most suitable configuration is chosen based on evaluation of performance functions. For safety-related applications, one needs to ensure that only configurations are chosen for which their safety is assured. To achieve this, the DMSC concept proposed in this paper enables run-time creation and evaluation of SCs for each of the possible configurations.

3 Related Work

Several state-of-the-art approaches propose the automated development of SC based on the automated instantiation of arguments patterns. An SC pattern specifies an abstract, reusable structure of a successful argumentation structure, containing placeholders for system-specific information, which can be filled in later, during pattern instantiation, i.e., during the usage of the pattern in the argument of a certain system (Kelly and McDermid 1998).

Denney and Pai provide formal semantics for creating SC patterns within GSN models, clarifying their restrictions and specifying a generic data model and pattern instantiation algorithm (Denney and Pai 2013). Following their work, they extended patterns with pattern metadata (Denney and Pai 2015), to capture the notion of tracing between pattern elements, e.g., informal claims. They further proposed formal foundations for composing different GSN arguments developed by patterns instantiation (Denney and Pai 2016). They defined arbitrary patterns composition, by taking the union of all links in the respective patterns, using shared identifiers as the points at which to join.

Finally, they implement their developed concepts in AdvoCATE (Denney and Pai 2018), a modelling tool which provides tool support for GSN-based SC pattern instantiation. However, the pattern instantiation is not a fully automated activity due to the usage of instantiated data table, which is manually specified by the safety engineer at design-time. Despite the provided formalization, they do not define the algorithmic checks for the evaluation of composed patterns at run-time.

Šljivo et al. (Šljivo, et al. 2020) propose extended design pattern templates with contractual specifications, providing clear understanding of designed patterns compatibility with a given system environment, checking whether they fulfil the guaranteed safety claims. In particular, they define *PatternAssumptions*, representing conditions that shall be met for the correct usage of the design pattern, while the *PatternGuarantees* represent the conditions that the correct application of the pattern yields. The approach is implemented within the AMASS platform (De La Vara, et al. 2020), where contract-based reasoning via OCRA is enabled. Nevertheless, their approach has not been exploited for the dynamic re-construction of SC.

Vierhauser et al. (Vierhauser, et al. 2021) introduce a mechanism for unmanned aerial vehicles (UAV) based on composable Safety Assurance Case (SAC). Their system assurance case is composed of: 1) an Infrastructure Safety Assurance Case (ISAC), which argues about the satisfaction of infrastructure-level safety goals, and 2) Pluggable Safety Assurance Cases (pSACs), which specify the safe operation of individual systems within a Complex open and adaptive system. They extend GSN with interlock points to dynamically plug at run-time sub-trees of pSAC to ISAC's interlock points. By combining their approach to monitoring methods at run-time, they check the validity of the entire SC. Their method similarly supports the idea of module assembly in SC for complex open and adaptive systems considering operational data. Nonetheless, they do not address the re-evaluation of composed modules patterns dynamically at run-time.

As explained earlier in this paper, DSC have been proposed to support the re-evaluation of the validity of design-time safety assumptions at run-time (Denney, Pai and Habli 2015), (Asaadi, et al. 2020). To this end, Denney and Pai propose that the SC is machine-comprehensible and hence, formalized.

Calinescu et al. introduced ENTRUST (Calinescu, et al. 2018), an end-to-end methodology for the engineering of trustworthy self-adaptive software systems, also implementing the DSC approach. They propose the development of a system safety assurance using SC patterns, which are partially instantiated with placeholders for the assurance evidence that cannot be obtained until the uncertainties associated with the system are resolved at run-time. One proposed pattern argues about the satisfaction of a set of safety requirements by the current system configuration. Given a system reconfiguration, ENTRUST proposes the re-verification of all safety requirements and, based on the obtained verification evidence, the re-instantiation of the respective SC pattern. Still, the re-verification of all requirements is time-consuming and not in line with the need for the rapid assurance of the new configuration required by open adaptive complex systems.

Cheng et al. (Cheng, et al. 2020) introduced the AC-ROS approach, which enables Robot Operating System (ROS) based platforms to conform to GSN models at run-time, thereby assuring that the system continues to satisfy its safety requirements after system reconfiguration. Despite this approach provides first steps towards DSC, its scope is limited to ROS-based systems.

Although state-of-the-art approaches enable initial support for dynamic SC management, they do not provide specific guidelines to develop SC elements such as modules and patterns to enable their composition in line with structural changes of open complex and adaptive systems.

4 DMSC Specifications

In this section, we propose an automated approach for constructing the SC of open complex and adaptive systems in a modular manner, reflecting the current system configuration.

4.1 General concepts

First, to allow the automated construction of SC, there is a need for a formal specification of SC, following a certain structure, with certain semantics. On the one hand, GSN is one of the most frequently used notations for structuring SCs. On the other hand, Yan et al. reviewed the current Model Based Engineering (MBE) techniques for generating SCs and they suggested that the SACM metamodel can support automatic SC generation, which they claim reduces the workload and the potential for errors, and supports SC evolution along with the system development (Wei, et al. 2019) , (Yan, Foster and Habli 2021). Consequently, for the specification of DMSC, we use the GSN/SACM metamodel, presented in Section 2.1.

The DMSC approach differentiates between the SC construction at design-time and at run-time. It proposes module-based assembly of SC, in correspondence to the open complex and adaptive systems reconfiguration. In other words, for each new configuration, the system SC is re-constructed based on automated composition of SC modules.

To facilitate this, the composition of SC modules must mirror the architecture of the open complex adaptive system, i.e., the composition of the network of services. For each service and each safety property that the service needs to satisfy, one module is constructed. The goal decomposition within such an SC module corresponds to the service composition specified by the blueprint implemented by the respective service. Such module has a direct trace link to the related blueprint.

We categorize SC modules based on the service blueprint it addresses as follows:

- **Composed module**: specifies the safety argument corresponding to a service composition. The argumentation within such module can be further developed in other basic and/or combined modules, arguing about the composed services.

- **Basic module**: specifies the safety argument associated to a basic service.

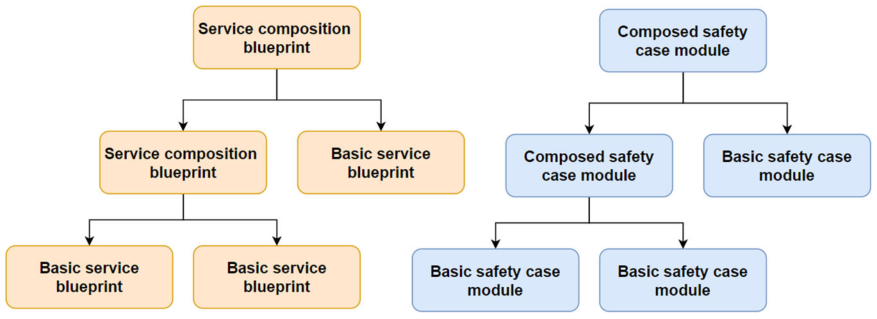


Fig. 6. Mapping between safety case and service blueprint architecture

Fig. 6 describes the mapping between SoR and SC architecture and Fig. 7 illustrates an overview of the proposed approach for DMSC development.

In the next subsections, we describe how to develop the SC of a SoR-able system, by differentiating the steps that need to be taken at design-time and the one to be taken at run-time.

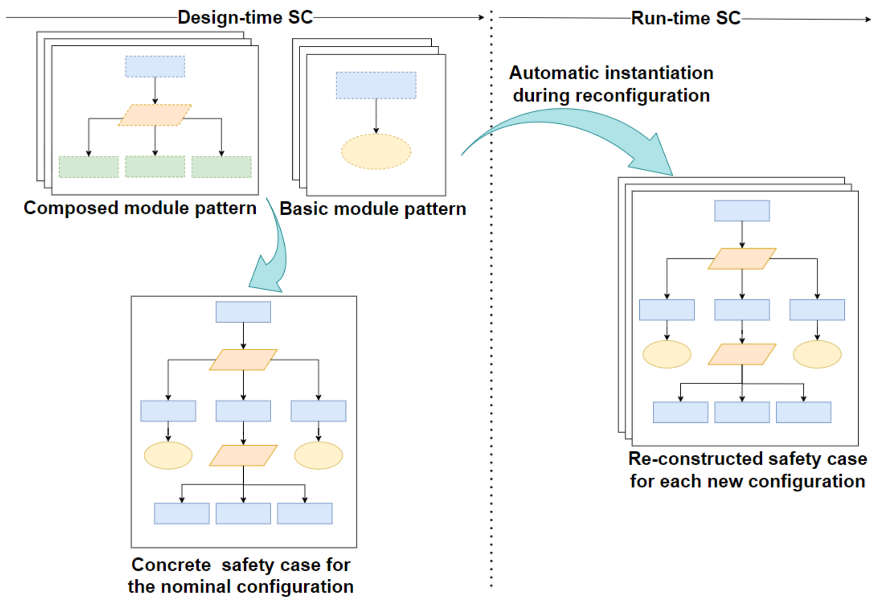

Design-time SC
Run-time SC
Automatic instantiation during reconfiguration
Composed module pattern
Basic module pattern
Concrete safety case for the nominal configuration
Re-constructed safety case for each new configuration

Fig. 7. General overview of the DMSC approach application at design-time and run-time

4.2 Design-time SC specifications

In order to support the generation of SC modules, we use SC patterns. Each pattern and the modules instantiating that pattern provide a trace link to a service blueprint, and, respectively, to the available services. Such a mapping enables automatic module composition respective to the service composition within each configuration. Consequently, a concrete SC can be re-constructed and instantiated at run-time for each possible system configuration.

First, to ease the specification of SC modules arguing about the assurance of basic services, for each property type, we propose an SC pattern, which can be instantiated for each available basic service. Such module entails a top-level *Goal* about the satisfaction of one safety property (e.g., the failure rate of the addressed service). This module provides a fully developed argumentation, namely they reference the evidence on which the argumentation is based. We assume that the argumentation about safety properties of basic services is not subject to change during reconfiguration (since the reconfiguration, as realized in SoR, affects structure only), so that the design-time argumentation related to basic services remains untouched and valid during run-time reconfiguration.

Similarly, we propose an SC pattern for each composed module, which addresses the service composition blueprint and each safety property that needs to be demonstrated. In comparison to the basic modules, the argumentation within composed modules is not completely developed, but it is based on the argumentation about the safety assurance of the composed services. To achieve this, these patterns entail *Away Goals*, each pointing at instantiation to an SC module arguing about the safety assurance of a constituent service of the service composition. Further, these patterns entail a strategy explicitly indicating how different *Away Goals* support the top-level *Goal.* The top-level *Goal* guarantees the satisfaction of a certain safety property only if the assumptions correspondent to these properties, which are supported by *Away Goals*, are valid. Such a strategy may specify a safety property as a function that takes the qualitative or quantitative guarantees of the safety properties corresponding to the constituent services as inputs and combines them to compute the valid result for the service composition blueprint. For instance, for the failure rate as a safety property, assuming the service composition blueprints are decomposed to a series of basic service blueprints, the top-level *Goal* guarantees will be computed as the sum of all failure rates provided by the pointed *Away Goals* (see Fig.8).

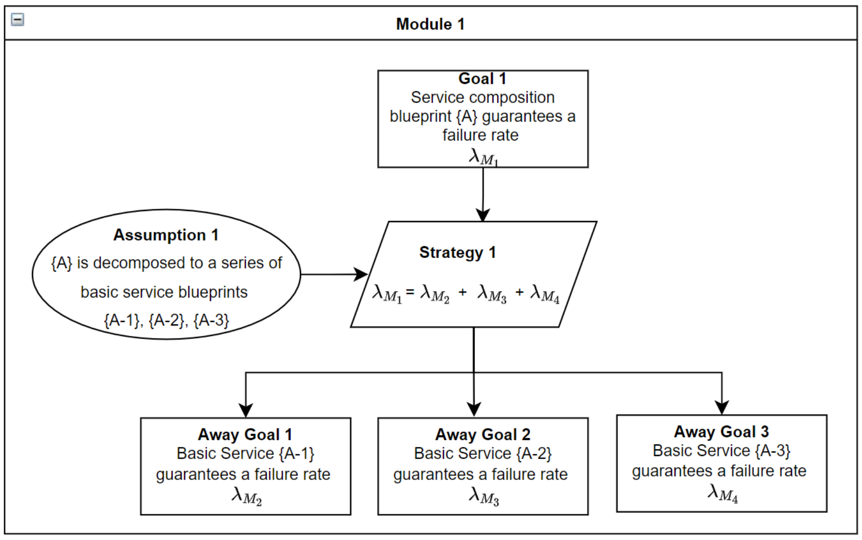

Module 1
Goal 1
Service composition blueprint {A} guarantees a failure rate λ_{M_1}
Assumption 1
{A} is decomposed to a series of basic service blueprints {A-1}, {A-2}, {A-3}
Strategy 1
$\lambda_{M_1} = \lambda_{M_2} + \lambda_{M_3} + \lambda_{M_4}$
Away Goal 1
Basic Service {A-1} guarantees a failure rate λ_{M_2}
Away Goal 2
Basic Service {A-2} guarantees a failure rate λ_{M_3}
Away Goal 3
Basic Service {A-3} guarantees a failure rate λ_{M_4}

Fig. 8. Safety property function embedded into strategy to calculate the provided guarantees from basic service blueprints

For open complex adaptive systems, constructing SC modules following the previous specification leads to creation of the SC structure. The key information to determine which basic and/or composed modules can be assembled later at run-time are the instantiation mapping data linking the SC to the service blueprint architecture.

At design-time, the SC of the initial nominal configuration is developed by connecting the SC module scoping a nominal service composition with basic and/or composed SC modules via *Away Goals*.

We extend the GSN/SACM metamodel having a claim associated to a certain type of service composition blueprint. This Away *Goal* is supposed to point to a *Goal,* which supports a claim associated to a certain type of basic and/or service composition blueprint. This leads to automatic instantiation of different *Away Goal*s along with service composition blueprint instantiation. Fig. 9 presents an overview of our proposed extension for GSN/SACM metamodel.

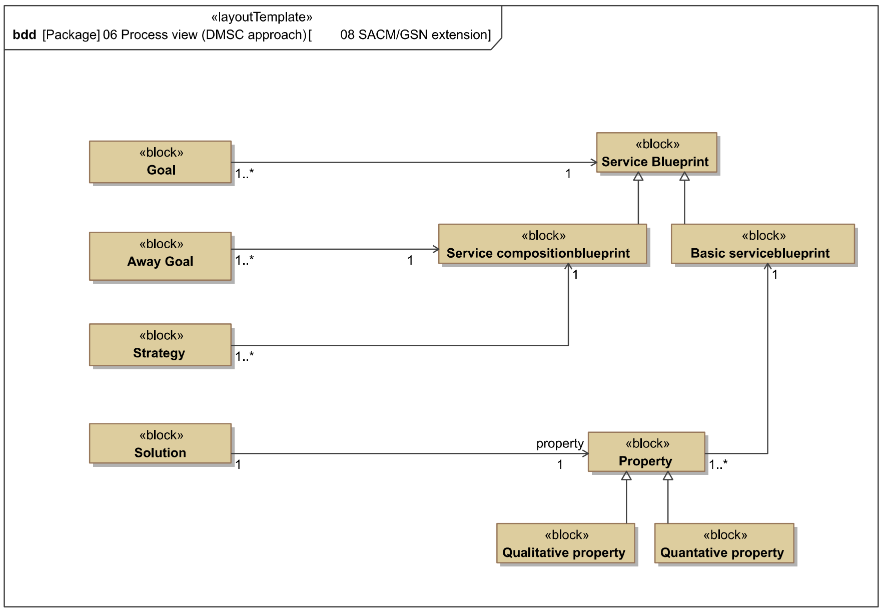

«layoutTemplate»
bdd [Package] 06 Process view (DMSC approach) [08 SACM/GSN extension]
«block» Goal
«block» Away Goal
«block» Strategy
«block» Solution
«block» Service Blueprint
«block» Service compositionblueprint
«block» Basic serviceblueprint
«block» Property
«block» Qualitative property
«block» Quantative property
property
1..*
1

Fig. 9. The extended GSN/SACM metamodel supporting DMSC

To this end, we need to be able to specify a direct trace link between a *Goal* and one of the available service blueprints. To enable the specification of such direct trace links, we make use of the modelling capabilities offered by SACM (Selviandro, Hawkins und Habli 2020). In the following, we explain how SACM supports the specification of direct trace links, leaving out any other modelling concepts that are not relevant for this specification.

According to the GSN/SACM, all GSN constructs extend the `ArtifactElement` SACM class, which extends the `ModelElement` SACM class. Consequently, inheriting from the `ModelElement` SACM class, any GSN construct has a `description`. The GSN/SACM relationship is illustrated in (ACWG-GSN-MM 2021).

The `description` specifies the claim of a GSN construct in `MultiLangString`, i.e., different languages (e.g., various natural languages such as English or German, or more formal languages such as Linear Temporal Logic). Further, a `description` may entail one or more `Terms`. A `Term` actually specifies a direct trace link or a placeholder for a direct trace link to a certain type of artefact. Each `Term` has an `externalReference` to a referenced artefact (i.e., models and model elements) of the type specified by the type attribute. Terms with empty `externalReference` can be used as to-be-instantiated parameters of abstract claims in parametrized safety case patterns (Matsuno and Taguchi 2011) i.e., placeholders for concrete trace links. Consequently, in our approach we use `Terms` in the claims of SC elements to establish mappings between the SC model and the service blueprint models.

4.1 Run-time SC construction and evaluation

For the purpose of run-time re-evaluation of each composed SC module, we formalise the design-time SC to be machine readable using the `SysML v2.0` textual notation[2], with some minor extensions. Once the reconfiguration is triggered, the SC modules will be re-assembled following the provided mapping between SC and blueprint architecture at design-time, which facilitates the instantiation of blueprints alongside with their SC modules per each new created configuration.

Further, for verifying the validity of new constructed SC, we check assume/guarantee relations, where the assumptions specify safety-related properties assumed in the current module that are expected to be demonstrated as valid by the modules pointed to by the *Away Goals*, and the guarantees specify safety-related properties that are demonstrated by the current module, given the satisfaction of its assumptions. The SC evaluation algorithm re-assesses each possible new configuration by verifying the assume/guarantee relation through traversing in the new composed SC modules. Eventually, if no violation is identified in the guarantees, the configuration is assessed as valid. Consequently, the configuration becomes part of the set of valid configurations, from which the SoR *Reconfiguration* component selects the most suitable configuration as the target configuration to be implemented.

5 Tool Support and Example

In our proposed approach, part of the development and assurance artefacts is done at design-time, such as the specification of service blueprints, safety case patterns, and SC modules. Hence, tool support for the specification of these artefacts would be beneficial.

5.1 Tool implementation

In this section, we discuss how we extended the FormAl SpecificaTion Environment (FASTEN)[3] in order to offer the needed tool support. FASTEN is an open-source environment for the specification, verification and assurance of safety critical systems. One characteristic of FASTEN is that it allows the deep integration between models of different aspects of the system (Ratiu, et al. 2021), e.g., a safety

[2] https://github.com/Systems-Modeling/SysML-v2-Release

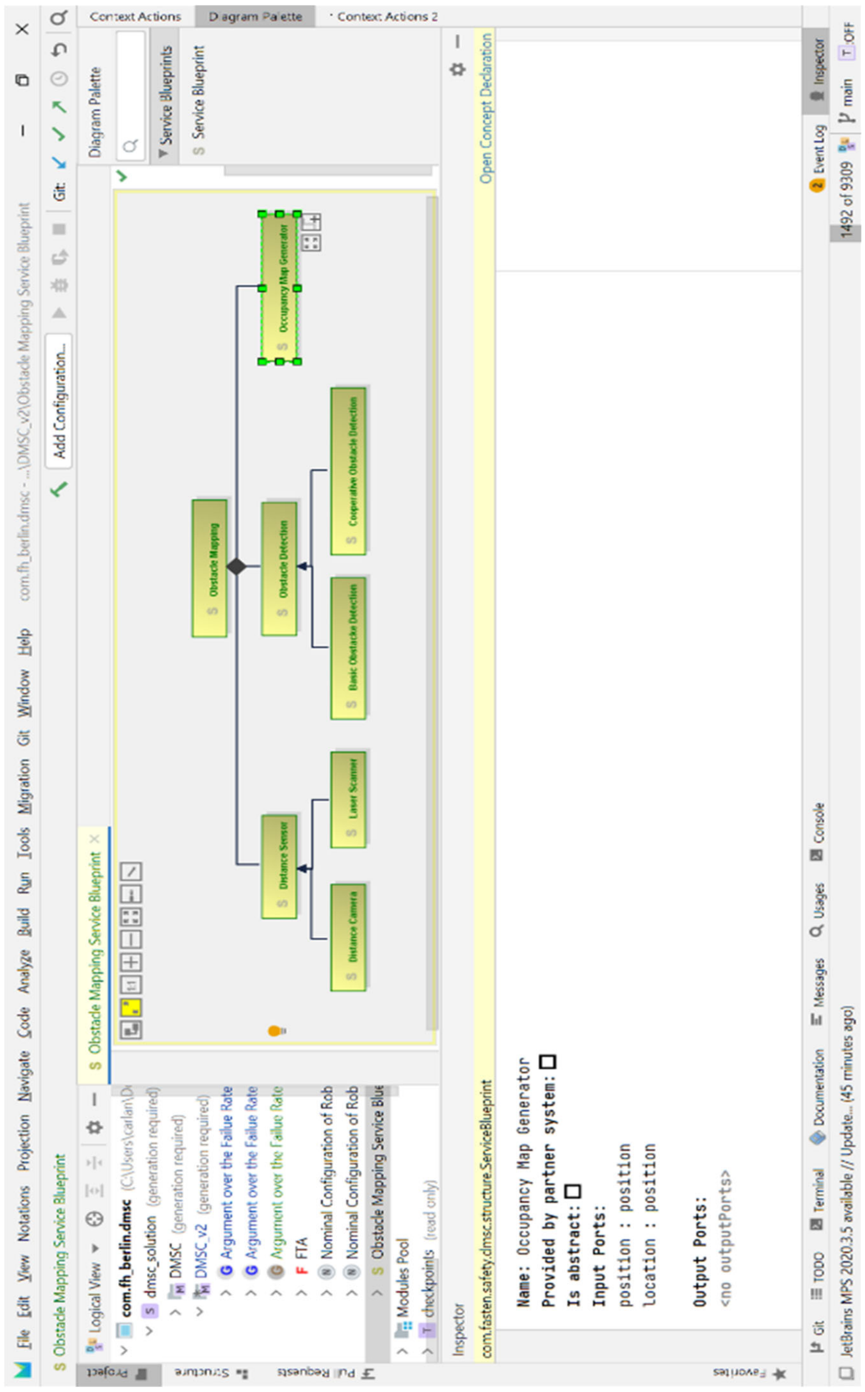
[3] https://sites.google.com/site/fastenroot/home

case model and the system architecture. FASTEN is built on JetBrains Meta Programming System (MPS), which is an open-source language workbench that targets Domain-specific Languages (DSLs). In contrast to general-purpose languages (GPLs), DSLs support the specification of systems in languages that directly use the concepts and logic from a specific application domain. Basically, FASTEN is a stack of DSLs, easily extensible via the specification of new DSLs. Among others, FASTEN has DSLs for the specification of system architecture, GSN-based safety cases, and of GSN-based SC patterns.

To enable the modelling of service blueprints presented in Subsection 2.2, and the mapping between SC and service blueprint models, we extend the FASTEN platform with a new stack of DSLs – FASTEN.DMSC. To enable the specification of direct trace links from safety case elements to the specified service blueprints, we extend the `IWord` interface from the FASTEN platform, which enables the specification of direct trace links from one model element to another. Examples of such trace links can be seen in Fig.6.

5.2 Example

We next illustrate how we apply the DMSC development guidelines, which are proposed in Section 4, to a simple, but clear example, while using FASTEN for the modelling activities. The example considers a scenario in a factory layout, where a group of two transport robots, robot R1 and robot R2, collaborate with each other.
We focus on the service composition blueprint of obstacle mapping – a service, which is provided by both robots from our example. In Fig.10 we show a screenshot from FASTEN tool, with the editor where we modelled the obstacle mapping service blueprint. According to the blueprint, an obstacle mapping service is composed of three other services, namely the distance sensor service, the obstacle detection service, and the occupancy map generator service.


File
Edit
View
Notations
Projection
Navigate
Code
Analyze
Build
Run
Tools
Migration
Git
Window
Help
com.fh_berlin.dmsc - ...\DMSC_v2\Obstacle Mapping Service Blueprint
Obstacle Mapping Service Blueprint
Add Configuration...
Git:
Logical View
com.fh_berlin.dmsc (C:\Users\carlan\D
dmsc_solution (generation required)
DMSC (generation required)
DMSC_v2 (generation required)
Argument over the Failure Rate
Argument over the Failure Rate
Argument over the Failure Rate
FTA
Nominal Configuration of Rob
Nominal Configuration of Rob
Obstacle Mapping Service Blue
Modules Pool
checkpoints (read only)
Obstacle Mapping
Distance Sensor
Obstacle Detection
Occupancy Map Generator
Distance Camera
Laser Scanner
Basic Obstacle Detection
Cooperative Obstacle Detection
Diagram Palette
Service Blueprints
Service Blueprint
Context Actions
Diagram Palette
Context Actions 2
Project
Structure
Pull Requests
Favorites
Inspector
com.fasten.safety.dmsc.structure.ServiceBlueprint
Open Concept Declaration
Git
TODO
Terminal
Documentation
Messages
Usages
Console
Event Log
Inspector
JetBrains MPS 2020.3.5 available // Update... (45 minutes ago)
1492 of 9309
main
OFF

```
Name: Occupancy Map Generator
Provided by partner system: ☐
Is abstract: ☐
Input Ports:
position : position
location : position

Output Ports:
<no outputPorts>
```

Fig. 10. The obstacle mapping service composition blueprint modelled in FASTEN

Next, we present the steps taken for the modelling of a modular SC arguing about the safety assurance of the obstacle mapping service provided by robot R1. The SC justifies the safety of associated failure rate to this service being sufficiently low, where the failure rate of the composed service is computed from the failure rates of the composing services. For simplification purposes, we assume that the failure rates of each service are independent.

As a first step, we model in FASTEN an SC pattern arguing about the failure rate met by a given basic service (see Fig.11.a) The top-level goal (G1) of this pattern has a placeholder for a direct trace link to the addressed blueprint, using an extension of the `IWord` interface. The argument is supported by the results of a Fault Tree Analysis (FTA). Based on this pattern, at design-time, we create a set of SC basic modules, each corresponding to an available basic service. For our example, we create the SC modules corresponding to the available basic services of both robot R1 and robot R2: the basic distance sensor, the basic obstacle detection, and the basic occupancy map generation. Further, we also create a pattern arguing about the fact that the failure rate met by a service composition is sufficiently low (see Fig.11.b).

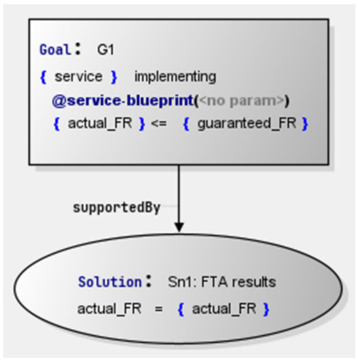

Goal: G1
{ service } implementing
@service-blueprint(<no param>)
{ actual_FR } <= { guaranteed_FR }
supportedBy
Solution: Sn1: FTA results
actual_FR = { actual_FR }

Fig. 11.a Safety case pattern for arguing about the sufficiency of the failure rate of a basic service

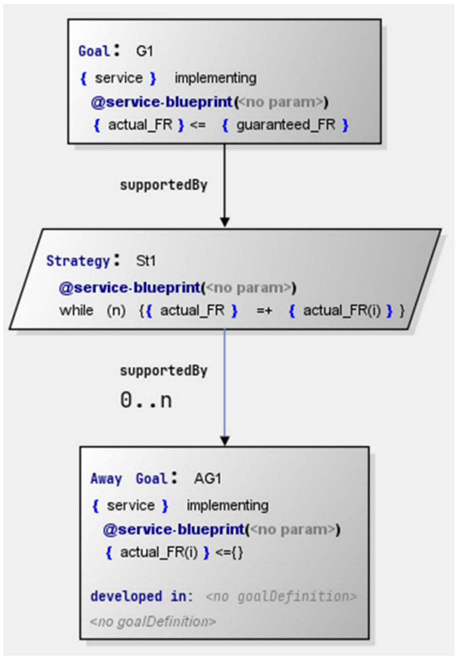

Goal: G1
{ service } implementing
@service-blueprint(<no param>)
{ actual_FR } <= { guaranteed_FR }
supportedBy
Strategy: St1
@service-blueprint(<no param>)
while (n) {{ actual_FR } =+ { actual_FR(i) } }
supportedBy
0..n
Away Goal: AG1
{ service } implementing
@service-blueprint(<no param>)
{ actual_FR(i) } <={}
developed in: <no goalDefinition>
<no goalDefinition>

Fig. 11.b Safety case pattern for arguing about the sufficiency of the failure rate of a service composition

Next, we model an SC module developed by partially instantiated pattern for arguing that a system implementing the composed obstacle mapping service has a failure rate that is sufficiently low. The structure of this SC module mirrors the composition of services presented by the service blueprint model (see Fig.12).

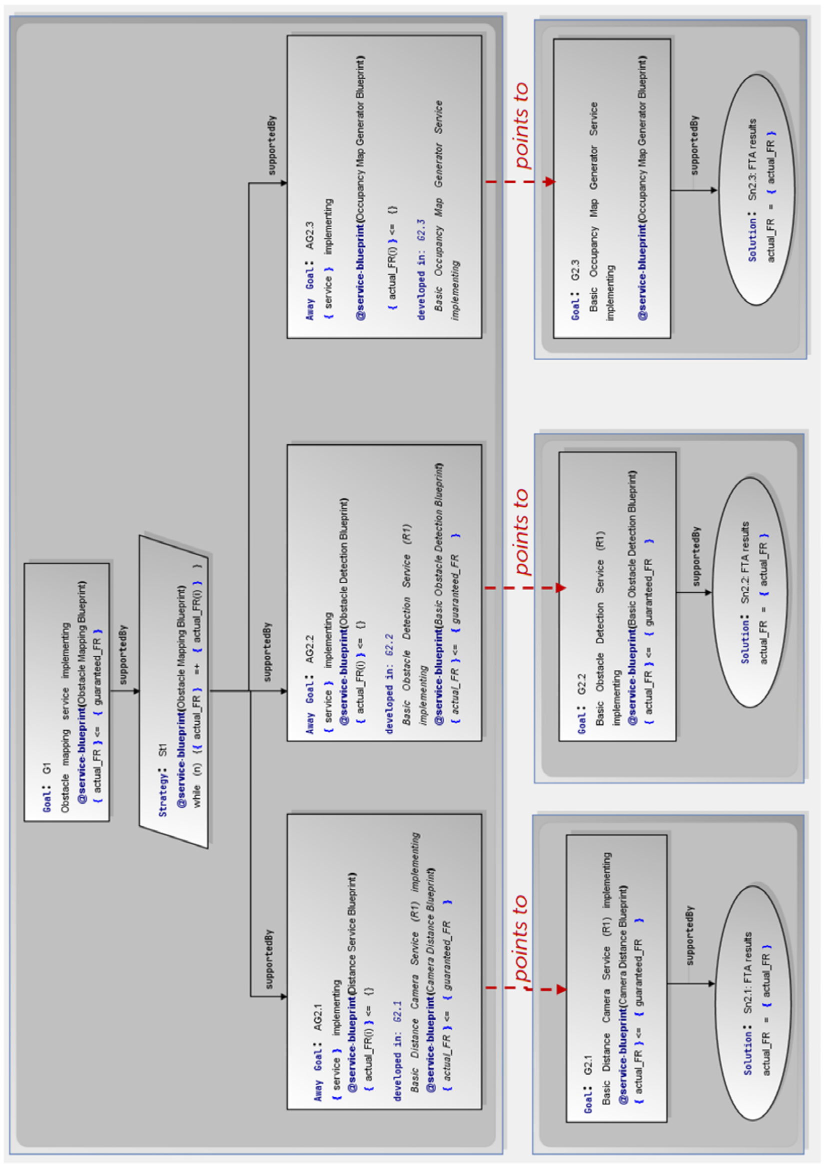

Goal: G1
Obstacle mapping service implementing
@service-blueprint(Obstacle Mapping Blueprint)
{ actual_FR } <= { guaranteed_FR }
supportedBy
Strategy: St1
@service-blueprint(Obstacle Mapping Blueprint)
while (n) {{ actual_FR } =+ { actual_FR() } }
supportedBy
Away Goal: AG2.1
{ service } implementing
@service-blueprint(Distance Service Blueprint)
{ actual_FR() } <= {}
developed in: G2.1
Basic Distance Camera Service (R1) implementing
@service-blueprint(Camera Distance Blueprint)
{ actual_FR } <= { guaranteed_FR }
Away Goal: AG2.2
{ service } implementing
@service-blueprint(Obstacle Detection Blueprint)
{ actual_FR() } <= {}
developed in: G2.2
Basic Obstacle Detection Service (R1) implementing
@service-blueprint(Basic Obstacle Detection Blueprint)
{ actual_FR } <= { guaranteed_FR }
Away Goal: AG2.3
{ service } implementing
@service-blueprint(Occupancy Map Generator Blueprint)
{ actual_FR() } <= {}
developed in: G2.3
Basic Occupancy Map Generator Service implementing
points to
points to
points to
Goal: G2.1
Basic Distance Camera Service (R1) implementing
@service-blueprint(Camera Distance Blueprint)
{ actual_FR } <= { guaranteed_FR }
Goal: G2.2
Basic Obstacle Detection Service (R1) implementing
@service-blueprint(Basic Obstacle Detection Blueprint)
{ actual_FR } <= { guaranteed_FR }
Goal: G2.3
Basic Occupancy Map Generator Service implementing
@service-blueprint(Occupancy Map Generator Blueprint)
Solution: Sn2.1: FTA results
actual_FR = { actual_FR }
Solution: Sn2.2: FTA results
actual_FR = { actual_FR }
Solution: Sn2.3: FTA results
actual_FR = { actual_FR }

Fig. 12. The concrete safety case for the nominal configuration instantiated from the obstacle mapping service composition blueprint at design-time

As discussed in Section 4, considering the nominal configuration, the top-level safety *Goal* is instantiated at design-time with a reference to the composed service blueprint. Since the obstacle mapping service is composed by basic and/or service compositions blueprints, the satisfaction of the top-level *Goal* is demonstrated by arguing that the failure rates of the components implementing the service blueprints are sufficiently low. Therefore, the argumentation within this SC pattern is supported by *Away Goals*, which, after instantiation, point to SC modules arguing about the failure rates met by the services composing the obstacle mapping service. At design time, the *Away Goals* are instantiated considering the nominal configuration. At run-time, whenever a reconfiguration occurs, whereas the instantiated top-level *Goal* does not undergo any other changes, the *Away Goals* are to be re-instantiated, i.e., they will point to different SC modules, depending on the chosen composing services in the new configuration.

In Fig.12, we show how the SC for the nominal configuration is modelled in FASTEN, based on the instantiated patterns. The nominal configuration of robot R1 implements the obstacle mapping service by composing three basic services for distance sensor, occupancy map generator and obstacle detection service. Therefore, the *Away Goals AG2.1, AG2.2, AG2.3* in the safety case arguing about the failure rate of the obstacle mapping service point to *G2.1, G2.2* and *G2.3*, which argue about the fact that the failure rates associated to these basic services are sufficiently low.

Once we create the design-time patterns and modules, we formalized the modules using `SysML v2.0` textual notation for the SC instantiation and evaluation at run-time. As a reconfiguration is triggered, a new SC fragment composing the modules scoping the composing services is created. For all the new possible configurations, a re-evaluation will be done verifying the assume/guarantee relations between the obstacle mapping module and the distance sensor basic and/or composed modules.

In our example, for the obstacle mapping modules pattern, we assume the option to instantiate either basic or cooperative services for the distance sensor, obstacle detection, and occupancy map generation. According to the combination formula (Cameron 1994), the three options out of six available services result in 20 possible configurations – which is a considerable number for such simple example. Whilst concrete safety cases for these 20 configurations could be created and evaluated at design-time, this is impossible for any complex open adaptive system of meaningful size and complexity. Here, the number of possible configurations easily reaches thousands.

6 Conclusion and future work

In this paper, we continue our ongoing line of work on developing DMSC to support SoR within the context of complex open and adaptive systems. In particular, we describe how to develop a design-time SC in a structured manner to facilitate safety assessment during reconfiguration at run-time. To this end, we outline the guidelines for the design-time specification of DMSC, while also using SC patterns. Further, we enable the co-evolution between system and safety architectures by defining trace links between the service blueprint architecture in SoR and the SC hierarchy established using the DMSC method and patterns. This also facilitates more purposeful and systematic SC maintenance by restricting the propagation of change impact only to certain SC modules.

Together with the proposed design-time SC, we additionally propose the formalization of SC modules with the purpose of automated pattern instantiation and reconstruction of a concrete SC at run-time for each new created configuration. The SC automatically created at run-time can be evaluated by validating the assume/guarantee relations between modules and assuring the module composition. Further, we provide tool-support for our proposed safety case development guidelines by extending FASTEN – a system and safety engineering platform with capabilities for modelling service blueprints, service-oriented architectures based on those blueprints and DMSCs. Finally, we applied the proposed DMSC specification guidelines to an example from the smart factory domain.

In this paper, we propose the development of a DMSC at run-time, via the composition of SC modules specified at design-time. As a next step, we will elaborate on how to define formalised assume/guarantee contracts for each SC module and we will propose an automatic analysis of the compatibility of these contracts.

Acknowledgments Barbara Gallina is partially supported by the by Sweden's Knowledge Foundation via the SACSys (Safe and Secure Adaptive Collaborative Systems) project. Elham Mirzaei and Carsten Thomas are supported by the German Ministry for Education and Research in frame of the ITEA3 research project CyberFactory#1[4] under funding ID 01IS18061D.

References

ACWG-GSN. 2021. "Goal Structuring Notation Community Standard." Vers. 3. Safety-Critical Systems Club (SCSC) Assurance Case Working Group (ACWG). https://scsc.uk/r141C:1?t=1.

ACWG-GSN-EX. 2021. "GSN-SACM Argumentation Example." Vers. 2.1. Safety-Critical Systems Club (SCSC) Assurance Case Working Group (ACWG). https://scsc.uk/file/gc/GSN2SACM_examples-1084.pdf.

ACWG-GSN-MM. 2021. "GSN Metamodel Specification." Vers. 2.2. Safety-Critical Systems Club (SCSC) Assurance Case Working Group (ACWG). https://scsc.uk/file/gc/GSN_metamodelV2-2-1210.pdf.

[4] https://www.cyberfactory-1.org/home/

Asaadi, Erfan, Ewen Denney, Jonathan Menzies, Ganesh J Pai, and Dimo Petroff. 2020. "Dynamic Assurance Cases: A Pathway to Trusted Autonomy." *Computer* 53 (12): 35 - 46.

Boardman, John, and Brian J Sauser. 2006. "The Meaning of System of Systems." Edited by IEEE. *IEEE/SMC International Conference on System of Systems Engineering.* Los Angeles, CA, USA: IEEE. doi:10.1109/SYSOSE.2006.1652284.

Bradbury, Jeremy S, James R Cordy, Juergen Dingel, and Michel Wermelinger. 2004. "A survey of self-management in dynamic software architecture specifications." *WOSS '04: Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems.* New York, NY, USA: Association for Computing Machinery. 28–33.

Calinescu, Radu, Danny Weyns, Simos Gerasimou, Muhammad Usman Iftikhar, Ibrahim Habli, and Tim Kelly. 2018. "Engineering Trustworthy Self-Adaptive Software with Dynamic Assurance Cases." *IEEE Transactions on Software Engineering* 44 (11): 1039 - 1069. doi:10.1109/TSE.2017.2738640.

Cameron, Peter J. 1994. *Combinatorics: Topics, Techniques, Algorithms.* Cambridge University Press.

CENELEC. 2007. *EN 50129: Railway applications - Communication, signalling and processing systems - Safety-related electronic systems for signalling.* Standard, International Electrotechnical Commission.

Cheng, Betty H C, Robert Jared Clark, Jonathon Emil Fleck, Michael Austin Langford, and Philip K McKinley. 2020. "AC-ROS: assurance case driven adaptation for the robot operating system." *MODELS '20: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems.* New York, NY, USA: Association for Computing Machinery. 102–113.

De La Vara, Jose Luis, Eugenio Parra, Alejandra Ruiz, and Barbara Gallina. 2020. "The AMASS Tool Platform: An innovative solution for assurance and certification of cyber-physical systems." *Joint Proceedings of REFSQ-2020 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track co-located with the 26th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2020),.* Pisa, Italy: CEUR Workshop Proceedings, CEUR-WS.

Denney, Ewen, and Ganesh Pai. 2013. "A Formal Basis for Safety Case Patterns." In *Computer Safety, Reliability, and Security*, by Guiochet J., Kaâniche M. Bitsch F., 21-32. Berlin & Heidelberg, Germany: Springer.

Denney, Ewen, and Ganesh Pai. 2016. "Composition of Safety Argument Patterns." In *Computer Safety, Reliability, and Security. SAFECOMP*, by Guiochet J., Bitsch F. Skavhaug A., 51-63. Springer.

Denney, Ewen, and Ganesh Pai. 2015. *Safety Case Patterns: Theory and Applications.* Technical, NASA.

Denney, Ewen, and Ganesh Pai. 2018. "Tool support for assurance case development." *Automated Software Engineering* (Springer) 25 (3): 435-499.

Denney, Ewen, Ganesh Pai, and Ibrahim Habli. 2015. "Dynamic Safety Cases for Through-Life Safety Assurance." *IEEE/ACM 37th IEEE International Conference on Software Engineering.* Florence, Italy: IEEE. 587-590.

Eurocontrol. 2006. "Safety Case Development Manual, ed. 2.2." Eurocontrol (European Organisation for the Safety of Air Navigation).

Industrial Avionics Working Group. 2012. "Modular Software Safety Case Process Description." Accessed November 2021. https://www.amsderisc.com/wp-content/uploads/2013/01/MSSC_201_Issue_01_PD_2012_11_17.pdf.

ISO/IEC/IEEE. 2019. "ISO/IEC/IEEE 15026-1: Systems and software engineering - Systems and software assurance - Part 1: Concepts and vocabulary." Standard.

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

Kelly, Tim. 2003. “Managing Complex Safety Cases.” In *Current Issues in Safety-Critical Systems*, by Anderson T. Redmill F., 99-115. London: Springer. doi:10.1007/978-1-4471-0653-1_6.

Kelly, Tim, and Simon Bates. 2005. “The Costs, Benefits, and Risks Associated With Pattern-Based and Modular Safety Case Development.” *UK MoD Equipment Safety Assurance Symposium.*

Leveson, Nancy. 2020. “White Paper on Limitations of Safety Assurance and Goal Structuring Notation (GSN).” http://sunnyday.mit.edu/safety-assurance.pdf.

Matsuno, Yutaka, and Kenji Taguchi. 2011. “Parameterised Argument Structure for GSN Patterns.” *11th International Conference on Quality Software.* Madrid, Spain: IEEE. 96-101. doi:10.1109/QSIC.2011.35.

Ministry of Defence. 2007. *Defence Standard 00-56: Safety Management Requirements for Defence Systems.* Standard, UK: Ministry of Defence.

Mirzaei, Elham, Carsten Thomas, and Mirko Conrad. 2020. “Safety Cases for Adaptive Systems of Systems: State of the Art and Current Challenges.” *Workshops. EDCC 2020. Communications in Computer and Information Science.* Munich, Germany: Springer, Cham. 127-138. doi:10.1007/978-3-030-58462-7_11.

Object Management Group. 2020. “Structured Assurance Case Metamodel (SACM), Version 2.1.” https://www.omg.org/spec/SACM/2.1/PDF.

Ratiu, Daniel, Arne Nordmann, Peter Munk, Carmen Carlan, and Markus Voelter. 2021. “FASTEN: An Extensible Platform to Experiment with Rigorous Modeling of Safety-Critical Systems.” In *Domain-Specific Languages in Practice*, by Cicchetti A., Ciccozzi F., Pierantonio A. Bucchiarone A., 131-164. Springer, Cham. doi:doi.org/10.1007/978-3-030-73758-0_5.

Richardson, Chris. 2018. *Microservices Patterns: With Examples in Java.* New York: Manning Publications.

SAWG, SCSC. 2020. “SCSC Publications.” Vers. V1.0. *SCSC.* Edited by Mike Parsons. Service Assurance Working Group (SAWG). February. Accessed November 2021. https://scsc.uk/scsc-156.

Selviandro, Nungki, Richard Hawkins, and Ibrahim Habli. 2020. “A Visual Notation for the Representation of Assurance Cases Using SACM.” In *IMBSA 2020: Model-Based Safety and Assessment*, by Marc Zeller and Kai Höfig, 3-18. Springer. doi:10.1007/978-3-030-58920-2_1.

Siefke, Lennart, Volker Sommer, Björn Wudka, and Carsten Thomas. 2020. “Robotic Systems of Systems Based on a Decentralized Service-Oriented Architecture.” *Robotics* 9 (4): 78. doi:10.3390/robotics9040078.

Šljivo, Irfan, Garazi Juez Uriagereka, Stefano Puri, and Barbara Gallina. 2020. “Guiding Assurance of Architectural Design Patterns for Critical Applications.” *Journal of Systems Architecture* 110: 101765. doi:10.1016/j.sysarc.2020.101765.

Thomas, Carsten, Elham Mirzaei, Björn Wudka, Lennart Siefke, and Volker Sommer. 2021. *Service-Oriented Reconfiguration in Systems of Systems Assured by Dynamic Modular Safety Cases.* Vol. 1462, in *Dependable Computing - EDCC 2021 Workshops. EDCC 2021. Communications in Computer and Information Science*, by Rasmus Adler, Amel Bennaceur, Simon Burton, Amleto Di Salle, Nicola Nostro, Rasmus Løvenstein Olsen, Selma Saidi, Philipp Schleiss, Daniel Schneider and Hans-Peter Schwefel, 12-29. Munich, Germany: Springer. doi:10.1007/978-3-030-86507-8_2.

Vierhauser, Michael, Sean Bayley, Jane Wyngaard, Wandi Xiong, Jinghui Cheng, Joshua Huseman, Robyn Lutz, and Jane Cleland-Huang. 2021. “Interlocking Safety Cases for Unmanned Autonomous Systems in Shared Airspaces.” Edited by IEEE. *IEEE Transactions on Software Engineering* 47 (5): 899-918. doi:10.1109/TSE.2019.2907595.

Wei, Ran, Tim P Kelly, Xiaotian Dai, Shuai Zhao, and Richard Hawkins. 2019. “Model based system assurance using the structured assurance case metamodel.” *Journal of Systems and Software* 154: 211-233. doi:10.1016/j.jss.2019.05.013.

Wudka, Björn, Carsten Thomas, Lennart Siefke, and Volker Sommer. 2020. “A Reconfiguration Approach for Open Adaptive Systems-of-Systems.” *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW).* Coimbra, Portugal: IEEE. 219-222. doi:10.1109/ISSREW51248.2020.00076.

Yan, Fang, Simon Foster, and Ibrahim Habli. 2021. “Safety Case Generation by Model-based Engineering: State of the Art and a Proposal.” *The Eleventh International Conference on Performance, Safety and Robustness in Complex Systems and Applications.* International Academy, Research, and Industry Association. https://eprints.whiterose.ac.uk/172352/.

Title of Article: Design-time Specification of Dynamic Modular Safety Cases In Support of Run-Time Safety Assessment

Name of Author(s): Elham Mirzaei, Carmen Cârlan, Carsten Thomas, Barbara Gallina

Name of Copyright Owner (if not author):

Address of Copyright Owner:

Signature [illegible]

…………………………………..