Contract number: Eurostars 6095 Safe-E

Safe Automotive soFtware architEcture – Extension (SAFE-E)

WP3

Deliverable D3.4.b: Updated proposal for extension of Meta-model for safety case modeling and documentation

Please note: This document is a direct input to the SAFE Project (Contract number: ITEA2 – 10039) as Deliverable D3.1.3b

Due date of deliverable: 30/11/2014

Actual submission date: 28/11/2014

Start date of the project: 01/09/2011 **Duration:** 40 months

Project coordinator name: Andreas Eckel

Organization name of lead contractor for this deliverable: fortiss GmbH

Editor: Maged Khalil – fortiss GmbH

Contributors: Maged Khalil – fortiss GmbH
Eduard Metzker – Vector Informatik GmbH (SAFE Partner)

Reviewers: Vladimir Rupanov – fortiss GmbH
Eduard Metzker – Vector Informatik GmbH (SAFE Partner)

Revision chart and history log

Version	Date	Reason
0.1	2012-09-19	Initialization of document
0.2	2012-10-18	Input to Introduction Section 4
0.3	2012-11-07	Input to Overview on ISO Section 5
0.4	2012-11-16	Input to EAST-ADL Overview Section 7
0.5	2012-11-16	Input to Overview on ISO Section 5
0.6	2012-11-23	Input to Methodology Section 6
0.7	2012-11-29	Further Input to Methodology Section 6
0.8	2012-12-20	Editing and Input in various sections
0.9	2013-02-13	Input to SAFE Meta-model Contribution Section 8
0.10	2013-02-15	Introduction of Patterns Section 6.3
0.11	2013-03-08	Editing and Input in various sections
0.12	2013-03-24	First version ready for review
0.13	2013-03-26	Incorporation of first review comments
0.14	2013-03-27	Updated version reviewed. Input to Section 8.3.
1.0	2013-03-28	Incorporation of review comments, finalization of deliverable
1.1	2014-07-15	Document structure update for new extended deliverable version
1.2	2014-10-29	Section 9 introduced with new fortiss content
1.3	2014-11-20	Updated fortiss content
1.4	2014-11-27	Last changes, document ready for review
2.0	2014-11-28	Incorporation of review comments, finalization of deliverable

1 Table of contents

2 List of figures

3 Executive Summary

This work task targets the topics of safety case modeling and documentation according to ISO 26262.

An overview of the relevant sections of ISO 26262 based on the requirements allocated to this work task is presented.

Since it is the objective to develop a Meta-model extension for safety cases the current version of EAST-ADL is analyzed and gaps identified. Proposals for the extension of EAST-ADL are given.

The proposed methodology for safety case documentation in accordance with ISO 26262 is presented. Methods of how to use the provided safety case capability in a generic as well as in a pattern-based approach are explained.

Finally, the contribution of this work task to the collective SAFE Meta-model, which is developed and used cooperatively with the affiliated ITEA2 SAFE Project and which is based on EAST-ADL, is presented.

4 Introduction and overview of document

The document at hand provides information about a methodology for safety case modeling and documentation and a proposal for the extension of the collective SAFE Meta-model, which is developed and used cooperatively with the affiliated ITEA2 SAFE Project, which enables its use in a model-driven development environment. In the following subsection the scope of the work task as well as the structure of the document is presented.

4.1 Scope of WT 3.4

Embedded in work package 3, work task 3.4 deals with the safety case modeling including the ability to describe artifacts of the SAFE Meta-model in a safety case relevant context, as well as an expression of relations between artifacts in that context. The basis for this work task is the structured argumentation notation known as the Goal Structuring Notation (GSN) which is presented in section 6. WT 3.4 aims to provide a methodology for argumentation about safety cases and a Meta-model extension suitable for the collective SAFE Meta-model. Furthermore WT 3.4 explores how the provided methodology can be used in a pattern-based approach. In order to be able to do so, mainly the following artifacts and their interrelations are considered:

Hazard

Hazards represent the potential source of harm and form a key aspect of the hazard analysis and risk assessment and a focal point for safety activities. A concept to express hazards in formal as well as informal formulation is provided by a SAFE work task.

Hazardous Event

Hazardous events are relevant combinations of hazards and operational situations in a given operating mode. A SAFE work task develops a suitable representation including a concept for hazardous events shall enable the classification according to the parameters severity, probability of exposure, and controllability. Based on these parameters the ASIL classification is performed which is be supported by the Meta-model concept. Hazardous events can be categorized into general problem patterns that, in their turn, result in general solution patterns which can be collected into libraries that include the corresponding safety case template.

Safety Goal

Safety goals are derived from hazardous events and enables the expressions and documentation of safety goals with their respective parameters and association with a safe state (“operating mode of an item without an unreasonable level of risk” [1]). The safety goal is the starting point for any safety case.

Functional Safety Concept

The functional safety concept details the approach that will be used to counteract or mitigate the effects of the hazardous event and satisfy the corresponding safety goal(s). Functional safety concept can be categorized into known solution patterns, e.g., use redundancy to overcome the low reliability of a single channel or a source of common cause failures.

Technical Safety Concept

The technical safety concept describes the actual implementation details of the corresponding functional safety concept. E.g., redundancy can be achieved by using a homogenous hardware redundancy pattern, or a software logical redundancy pattern, according to the requirements of the situation.

4.2 Structure of document

The document is structured as follows:

Subsequent to this introduction an overview on the parts of ISO 26262, which are relevant for safety case modeling and documentation, is given in section 5.

Within chapter 6, the methodology for safety case description is explained. To do this, in a first step some background information and motivation is given, followed by a general introduction to the methodology used. Safety case description elements and their relations are discussed in section 6.2. An overview of testing of safety cases is given in section 6.3.

Section 7 discusses safety argumentation within EAST-ADL. EAST-ADL is an architecture description language (ADL), which has been developed in various projects in which both automotive vendors and users cooperated, where the objective was to define an ADL tailored to the needs of the automotive industry. On the one hand, the current version of EAST-ADL and in particular the dependability part is described und studied from a safety case perspective. On the other hand, some proposed extensions from parallel research and industrial projects to this current version are explained which introduce or enhance the possibility to perform safety case documentation in compliance with ISO 26262. The version referenced in this document, published at the time of the first version of the deliverable at (www.east-adl.info), is EAST-ADL V2.1 [44].

The contribution of this work task to the collective SAFE Meta-model is described in section 8. Within this section an overview on the part of the Meta-model as well as a detailed description of the classes and links used to construct the Meta-model is presented. Moreover, an example for the application of the Meta-model for safety case documentation is presented.

Section 9 gives an overview of pattern-based approaches and details the use of solution patterns for the generation of safety case skeletons or templates. It proposes a holistic approach to information encapsulation for design pattern documentation and the reuse of safety mechanisms. The approach is based on a modular library encompassing argumentation elements as well as development artifacts. The approach is supported by examples, generalizing the description of safety mechanisms, i.e. so-called *tactics*, within the scope of the SAFE Meta-Model.

Finally, in section 10 a summary and conclusion are given.

5 Overview on ISO 26262

Addressing the development process of electric / electronic components for passenger cars, the ISO 26262 “Road vehicles – Functional safety” came into effect in November 2011. This standard introduces a safety lifecycle which “encompasses the principal safety activities during the concept phase, product development, production, operation, service and decommissioning” ([1], part 2, p.3) and which can be seen as a guideline that demands a risk-assessment based development approach with seamless traceability.

Within this section, an overview on the relevant parts of ISO 26262 with regard to safety case modeling is given. The selection of the presented parts is based on the requirements allocated to this work task from the ISO26262 analysis activities in WP2.

However, as the purpose of a safety case is, simply put, to link safety goals to the solution fulfilling them using clear lines of argumentation, it is clear that safety case modeling capability is relevant for virtually all aspects of the ISO26262 activities covered by the SAFE Meta-model, in order to provide the link between generated artifacts and the requirements driving them.

In Figure 1 an overview on the different parts of ISO 26262 is given, with relevant parts directly derived from requirements allocation colored red and other safety case modeling relevant parts colored blue.

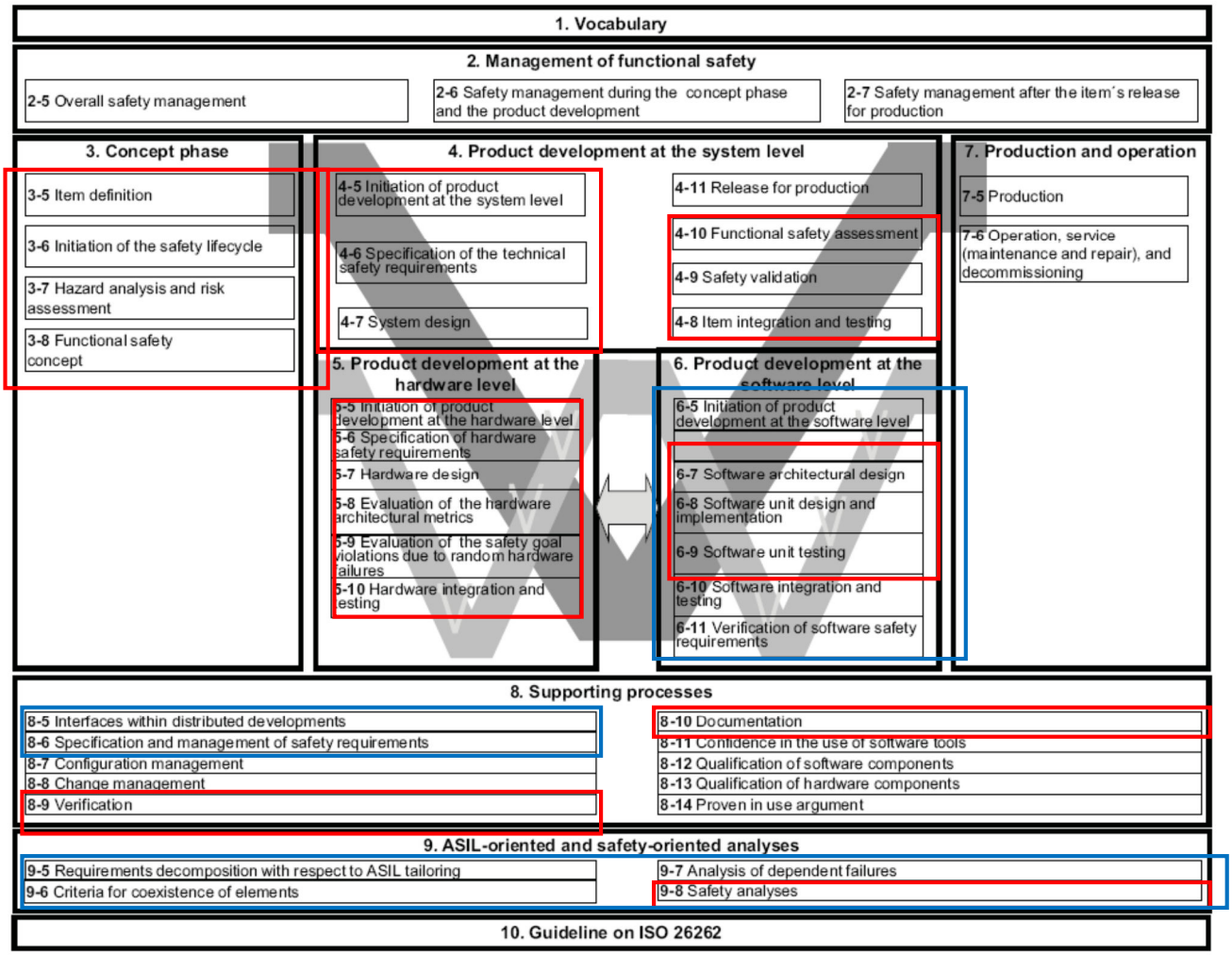

Figure 1: Overview on ISO 26262 (Relevant parts highlighted)

In the following, an overview on the relevant aspects from the respective ISO26262 parts is given.

Part 3: Concept Phase

The concept phase comprises mainly four different parts, namely the item definition, the initiation of the safety lifecycle, the hazard analysis and risk assessment, and the functional safety concept. These parts are explained in the following subsections.

Item Definition

The objective of the definition is to provide an overview on the item, the implemented functionalities and the dependencies as well as interactions of the item with the environment or other items of the vehicle. This information shall be provided in form of functional and non-functional requirements of the item. Moreover, the item definition includes a boundary description of the item as well as of elements of the item, i.e. a description of the interfaces and the expected as well as provided functionalities and interactions.

Initiation of the Safety Lifecycle

During the sub-phase of the initiation of the safety lifecycle it is distinguished between new developments and modifications of existing items. Depending on this the entire safety lifecycle or a tailored version needs to be applied.

Hazard Analysis and Risk Assessment

In general, the hazard analysis and risk assessment takes place based on the item definition and evaluates present risks without taking into account internal safety mechanisms of the item. This serves as the basis for defining safety goals and deriving the functional safety concept and its requirements, as shown in Figure 2.

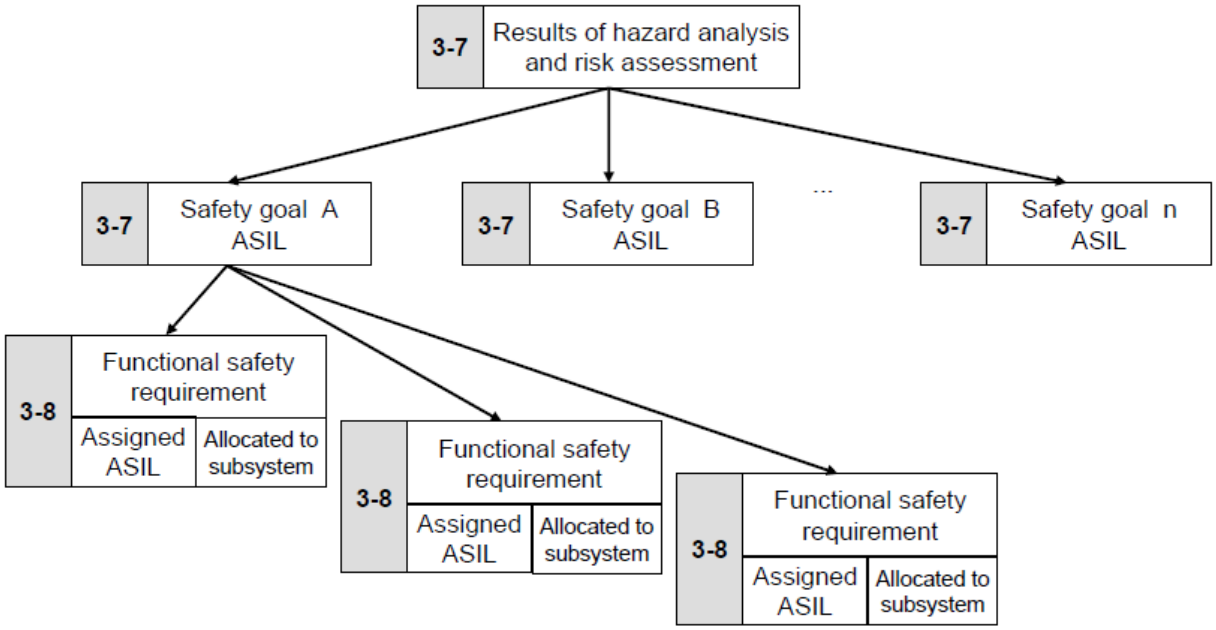

Figure 2: Concept phase activities after HA/RA carried out [1]

In a first step of the analysis, possible operational situations that are scenarios which might occur during the vehicles lifetime are collected. In this step it is important also to cover situations that arise

through foreseeable misuse of the vehicle. Subsequent to the definition of operational situations hazards which are related to the item need to be determined. Although the hazards need to be related to the item and are associated with a malfunction of the item, the description takes place on vehicle level, i.e. the resulting behavior at vehicle level needs to be determined. After identifying the hazards, relevant combinations of both, hazards and operational situations, are captured as hazardous events. These hazardous events are subject to classification according to the three parameters controllability, probability of exposure and severity. Based on the parameters the ASIL (Automotive Safety Integrity Level) is determined and assigned to the hazardous event. In case the determination of the ASIL leads to ASIL A, B, C or D, a safety goal has to be derived from the particular hazardous event. These safety goals are the top-level safety requirements for the item and serve as a basis for the later development of the functional safety concept.

Functional Safety Concept

Subsequent to the hazard analysis and risk assessment the functional safety concept is developed. The functional safety concept consists of functional safety requirements and preliminary architectural assumptions. The functional safety requirements which are derived from the safety goals are allocated to the elements of the item.

Part 4: Product Development – System Level

During this phase the development of the item from the system level perspective takes place. The process is based on the concept of a V-model. Starting point (on the upper left side) is the specification of the technical safety requirements which is followed by the development of the system architecture and the system design. The way up to the upper right point of the V-model is built by the integration, verification, validation and functional safety assessment activities.

Part 5: Product Development – Hardware Level

During this phase the development of the item from the hardware perspective is performed. The process is again based on a V-model, going down with the specification of hardware safety requirements as well as hardware design and implementation and back upwards with hardware integration and testing.

Part 6: Product Development – Software Level

During this phase the development of the item from the Software perspective is performed. The process is again based on a V-model, going down with the specification of Software safety requirements as well as Software design and implementation and back upwards with Software integration, testing and validation.

Part 8: Supporting Processes

The relevant requirements for this work task arise from two sections of part 8 (supporting processes), namely “Verification” and “Documentation”, with the following overview limited to these sections.

Verification

Within the section “Verification” requirements are given which need to be fulfilled in order to ensure that the work products comply with their requirements.

Documentation

Within the section “Documentation” requirements are given which need to be fulfilled in order to ensure that the work products and processes and all relevant links are properly documented.

Part 9: Automotive Safety Integrity Level (ASIL)-oriented and Safety-oriented Analyses

The assigned and accepted requirements for this work task arise from one section of part 9 (automotive safety integrity level (ASIL)-oriented and safety-oriented analyses), namely “Safety analyses”. An overview of only this section of part 9 is given.

Safety Analyses

With the help of the safety analyses consequences of faults and failures on functions, behavior and design of items and elements shall be examined. Moreover, the analyses provide information on causes and conditions that could lead to the violations of a safety goal or safety requirement. Additionally, the analyses contribute to the identification of new hazards not discovered during the hazard analysis and risk assessment.

6 Methodology for Safety Case Description

After presenting the relevant parts of ISO 26262 covered by the requirements from WP 2 allocated to this work task, we now present the methodology we propose to express safety cases.

6.1 Introduction

Although ISO 26262 requires looking at the risk emanating from the item without considering other elements of the vehicle architecture and without considering internal safety measures (cp. ISO 26262:3-2011, requirement 7.4.1.2), this risk itself is determined by the role of the item in the vehicle architecture. An example for this is that an EPS (electric power steering) system can be realized in a way that it can be overruled in any case by the driver. This would lead to a totally different classification compared to the realization of an EPS which cannot be overruled by the driver due to a too strong impact [45].

Therefore the model-based development process foreseen by SAFE-E has to take into account not only the item features but also all other elements / attributes that potentially contribute to the risk on vehicle level. The architecture suitable for the consideration of these needs has to fulfill the following aspects:

- there is a hierarchical architecture
- environmental aspects have to be distinguished
- functional / technical aspects have to be distinguished
- within technical aspects the hardware and software aspects need to be distinguished

The resulting architecture which is used in the SAFE Metamodel [48] is presented in the following figure.

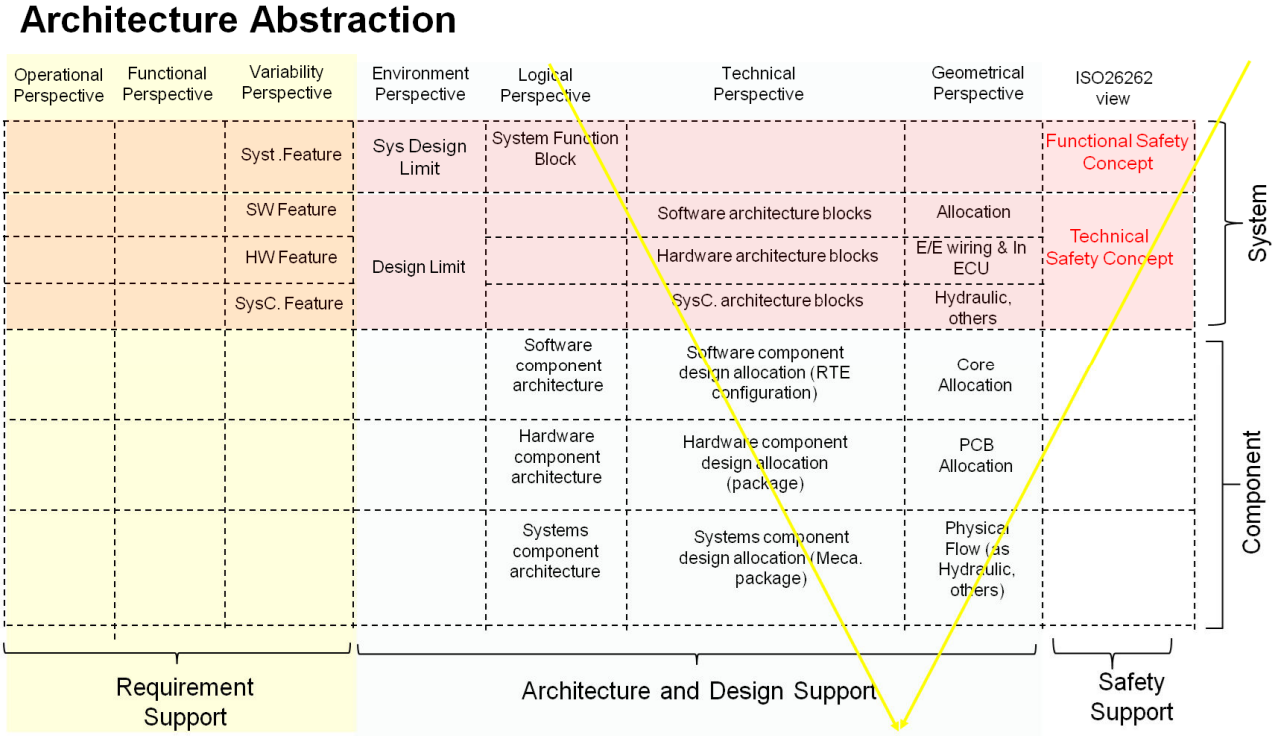

Figure 3: Overview on Structure of Architecture

Due to the structure of the architecture matrix shown the ASIL allocation could be different. Moreover, ASIL decomposition could be applied on any horizontal level, which has influences to the lower horizontal levels. Analogous to this, safety requirements could be allocated to different elements

within the horizontal level; this implies that a safety mechanism could be implemented into a sensor or alternatively into the controller or the actuator. By applying graceful degradation also the technical behavior in case of failure could be different and again this would lead to different inheriting of safety requirements to lower horizontal level.

As already depicted in the previous section 5, ISO26262 requires a detailed traceability linking requirements and their fulfillment to the underlying architecture in a comprehensible manner. This link, between safety goals and the solutions fulfilling them, using clear lines of argumentation is, simply put, the purpose of a safety case.

A common definition of a safety case is “a documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment”, whereby an argument is defined as “a connected series of claims intended to establish an overall claim.” In attempting to persuade others of the truth of an overall claim, we make supporting claims. These claims may themselves need further support. This gives rise to a hierarchy of claims (representing a logical chain of reasoning) by which an argument is established.

Given the multitude of generated safety-critical artifacts, let alone non safety-critical ones, it is clear that establishing this link through tracing alone is not possible, especially since simple traceability does not provide the expressiveness required for adequately describing the relations of various artifacts within a safety case context. This gives rise to the need for a dedicated safety case specific view on the artifacts.

6.1.1 Background and Motivation

“A safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context.” [13]

The concept of the ‘safety case’ has already been adopted across many industries (including defense, aerospace, nuclear and railways), see [1], [31], [34], [35], [36] and [37] and has been emphasized in numerous research works such as [33], [38], [39] and [40]. Studying the safety standards and guidance relating to these sectors, it is possible to identify a number of definitions of the safety case – some clearer than others. The definition given above attempts to cleanly define the core concept that is in agreement with the majority of the definitions we have discovered.

According to [13], a commonly observed failing of safety assessments is that the role of the safety argument is often neglected. In such safety cases, many pages of supporting evidence are often presented (e.g. hundreds of pages of fault trees or Failure Modes and Effects Analysis tables), but little is done to explain how this evidence relates to the safety objectives. The reader is often left to guess at an unwritten and implicit argument.

Both argument and evidence are crucial safety case elements that must go hand-in-hand. Argument without supporting evidence is unfounded, and therefore unconvincing. Evidence without argument is unexplained; it can be unclear how (or even if) safety objectives have been satisfied.

Safety cases thus have to be supported by the SAFE Meta-model, with the following rationale:

- The structured information management can be used as part of a safety argument in a safety case, and gives support to systematic safety/reliability analysis.
- The ability to support a safety case at a software architecture description level is important since it addresses an expanding area of functionality where the complexity is high.
- Traceability between the safety case and the design information is made possible, facilitating the work of the safety engineer, i.e. identifying the right information.
- Facilitate the system development of safety critical systems, by providing a link to where in a safety argument a certain “Entity” or “Item” under change is used. (Impact analysis of safety related systems)

6.2 Safety Case Description Elements and Relations

Within this section the description of the safety case description elements we use from GSN and their relations is given.

In order that safety cases can be developed, discussed, challenged, presented and reviewed amongst stakeholders, and maintained throughout the product lifecycle, it is necessary for them to be documented clearly. The documented argument of the safety case should be structured to be comprehensible to all its stakeholders. It should also be clear how the evidence is being asserted to support this argument.

GSN has been standardized into a first version since November 2011 and the specification document is readily available for free download. It is the primary source of the following two subsections.

The following subsection 6.2.1 gives a description of the standard elements used to describe artifacts while the subsequent section 6.2.2 details the types of relationships and the possible combinations used. Figure 4 depicts an example of the use of GSN [12].

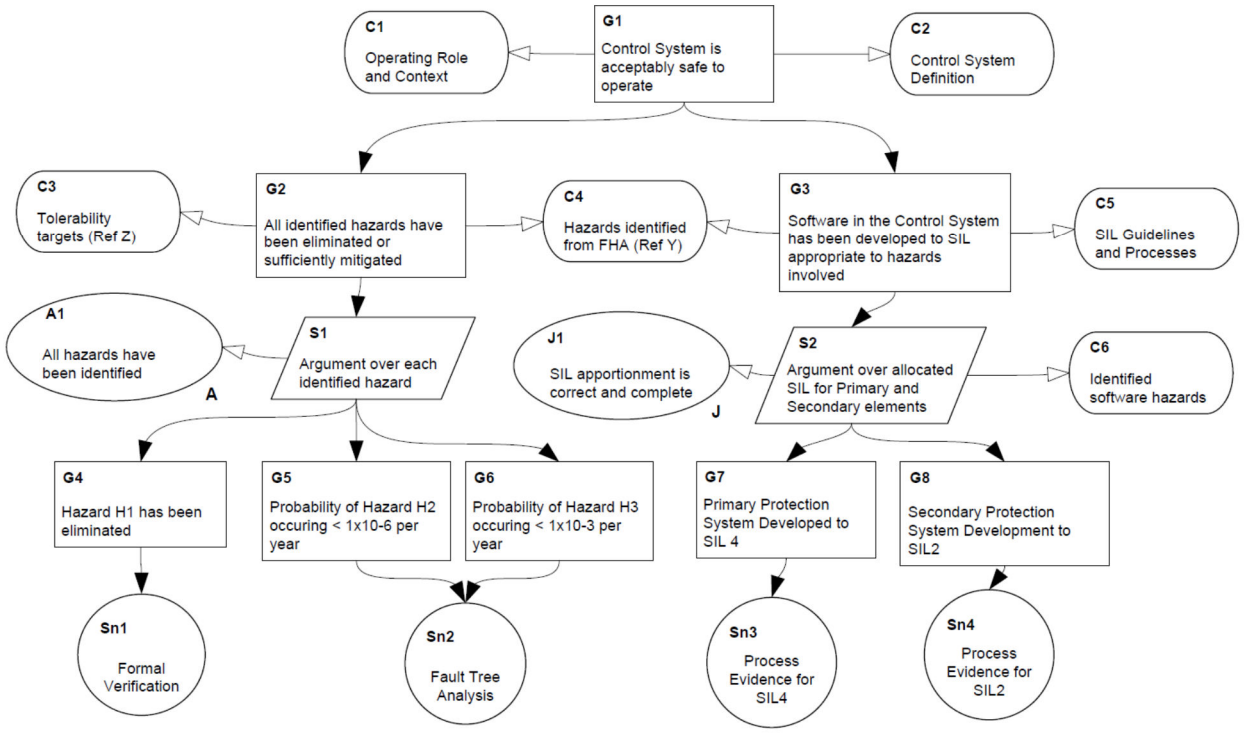

Figure 4: Example Goal Structure

6.2.1 Specification Elements

GSN [12] uses the following basic elements to describe the role of artifacts within a safety case:

Goal

- rendered as a rectangle,
- presents a claim forming part of the argument.

- One or more sub-goals may be declared for a given goal. This structure then asserts that if the claims presented in the sub-goals are true, this is sufficient to establish that the claim in the main goal is true.

Strategies

- rendered as a parallelogram,
- describe the nature of the inference that exists between a goal and its supporting goal(s)
- and are used to describe the nature of the inference which is asserted as existing between sub-goals and the parent goal.

Solutions

- A ***solution***, rendered as a circle, presents a reference to an evidence item or items.
- Multiple solutions may satisfy a goal.
- Multiple goals may be satisfied by one solution.

Contexts

- Claims can only be asserted to be true in a specified context. Context elements are used to make this relationship clear.
- A ***context***, rendered as an oblong rectangle with rounded out sides, presents a contextual artifact. This can be a reference to contextual information, or a statement.
- Where used, contexts define or constrain the scope over which the claim is made.

Assumptions

- An assumption is an intentionally unsubstantiated statement. The scope of an assumption is the entire argument. Having connected an assumption to a goal, the assumption is taken to be connected to the entirety of the argument supporting this goal.
- Therefore, it is not necessary to restate the assumption in the supporting argument.
- rendered as an oval with the letter ‘A’ at the bottom-right.

Justifications

- rendered as an oval with the letter ‘J’ at the bottom-right,
- presents a statement of rationale,
- and does not alter the meaning of the claim made in the goal, but provides rationale for its inclusion or its phrasing.
- Should an equivalent justification be required elsewhere in the argument, it will need to be re-stated or re-linked.

Furthermore it is possible to represent ***undeveloped entities***, rendered as a hollow diamond applied to the center of an element, indicating that a line of argument has not been developed. It can apply to goals (as below) and strategies.

A specific instance of ***undeveloped entities*** is an ***undeveloped goal***, rendered as a rectangle with the hollow-diamond ‘undeveloped entity’ symbol at the center-bottom, presenting a claim which is intentionally left undeveloped in the argument.

6.2.2 Relations and possible expressions

The core elements described in the previous section are linked using the following types of relationships [12]:

SupportedBy

- rendered as a line with a solid arrowhead, allows inferential or evidential relationships to be documented.
- Inferential relationships declare that there is an inference between goals in the argument. Evidential relationships declare the link between a goal and the evidence used to substantiate it.
- Permitted connections are:
    - goal-to-goal,
    - goal-to-strategy,
    - goal-to-solution,
    - strategy to goal.

InContextOf

- rendered as a line with a hollow arrowhead, declares a contextual relationship.
- Permitted connections are:
    - goal-to-context,
    - goal-to-assumption,
    - goal-to-justification,
    - strategy-to-context,
    - strategy-to-assumption
    - and strategy-to-justification.

6.2.3 Steps for building a safety case goal structure

Works of Kelly and McDermid, culminating in the GSN standard, base on well-revised argumentation works and suggest a multi-step approach to formulating goal structures and hence building safety cases, as seen in the following subsections.

Top Down: When classically working from Safety Goals

For a classic staged approach to safety arguments, Kelly [5] defines six steps in the top-down development of a goal structure:

1. Identify the goals to be supported;
2. Define the basis on which the goals are stated;
3. Identify the strategy used to support the goals;
4. Define the basis on which the strategy is stated;
5. Elaborate the strategy (and proceed to identify new goals – back to step 1), or step 6;
6. Identify the basic solution.

Bottom-Up: When working from existing evidence

When analyses evidence already exists or when an existing safety case needs updating, it is possible to adapt Kelly's six steps for top-down GSN development, into a process which can be used to develop a goal structure from the bottom up [12]:

1. Identify evidence to present as GSN solutions;

2. Infer 'evidence assertion' claims to be directly supported by these solutions, and present these as GSN goals;

3. Derive higher-level sub-goals that are supported by the evidence assertions;

4. Describe how each layer of sub-goals satisfies the parent goal (i.e. strategy);

5. Check that any necessary contextual information is included;

6. Check back down the structure for completeness;

7. Join the resulting goal structure to a known top goal or a set of sub-goals.

Using these methods in conjunction with the steps explained in section [8.2], it is possible to generate and document safety cases from Model elements directly. An implementation for safety case documentation and assessment as part of the SAFE project was carried out by Vector inside its commercial tool PreeVISION [59].

6.3 Testing of safety cases

Safety cases can have more than one role, depending on when and how they are generated and used. Each of these roles defines what types of tests can be performed and to which means.

Descriptive Safety Cases

Use:

When goal structures are generated to argue about existing products or during products development a safety case evolves which describes the safety of the system under development and the fulfillment of its safety goals. This safety case describes the status quo and aims at providing a clear and justifiable argument that the safety goals are met.

Testing:

Using the set of relation rules identified in Section 6.2.1 and 6.2.2, e.g., each **Goal** must resolve to a **Solution**, in a model-based development environment it is possible to perform completeness and consistency checks on this type of safety case.

Prescriptive Safety Cases

Use:

In conjunction with the use of patterns, once a decision is made to solve a known problem or use a trusted solution pattern or purchase a COTS component, the developer can directly create an instance of the (partial) safety case template or skeleton for the known pattern/component. It thus becomes possible to know what the safety case *should* look like and to use this as development guideline.

Testing:

Using the generated safety case skeleton as a reference, the developer can test existing solutions against it to see whether their arguments hold.

The extent of the testing depth and automation rely on how high the integration of the safety case elements into the development artifact landscape is.

7 Safety Cases with EAST-ADL

Within this section the current status of the architecture description language EAST-ADL V2.1 with regard to safety cases is described. Furthermore, proposals for an extension of the EAST-ADL concept are described which could lead to an enhancement of the possibility to model and document safety cases according to ISO 26262.

EAST-ADL introduces different levels of abstraction within a hierarchical modeling concept which facilitates controlling the complexity of systems. These levels are:

- Vehicle level,
- Analysis level,
- Design level,
- Implementation level, and
- Operational level.

7.1 Current status of EAST-ADL and other suggested extensions

Besides the different abstraction levels EAST-ADL includes several packages like, for instance, the variability package, the timing package, and the dependability package which is of special interest for this work task. An overview on the dependability package [44] is given in Figure 5.

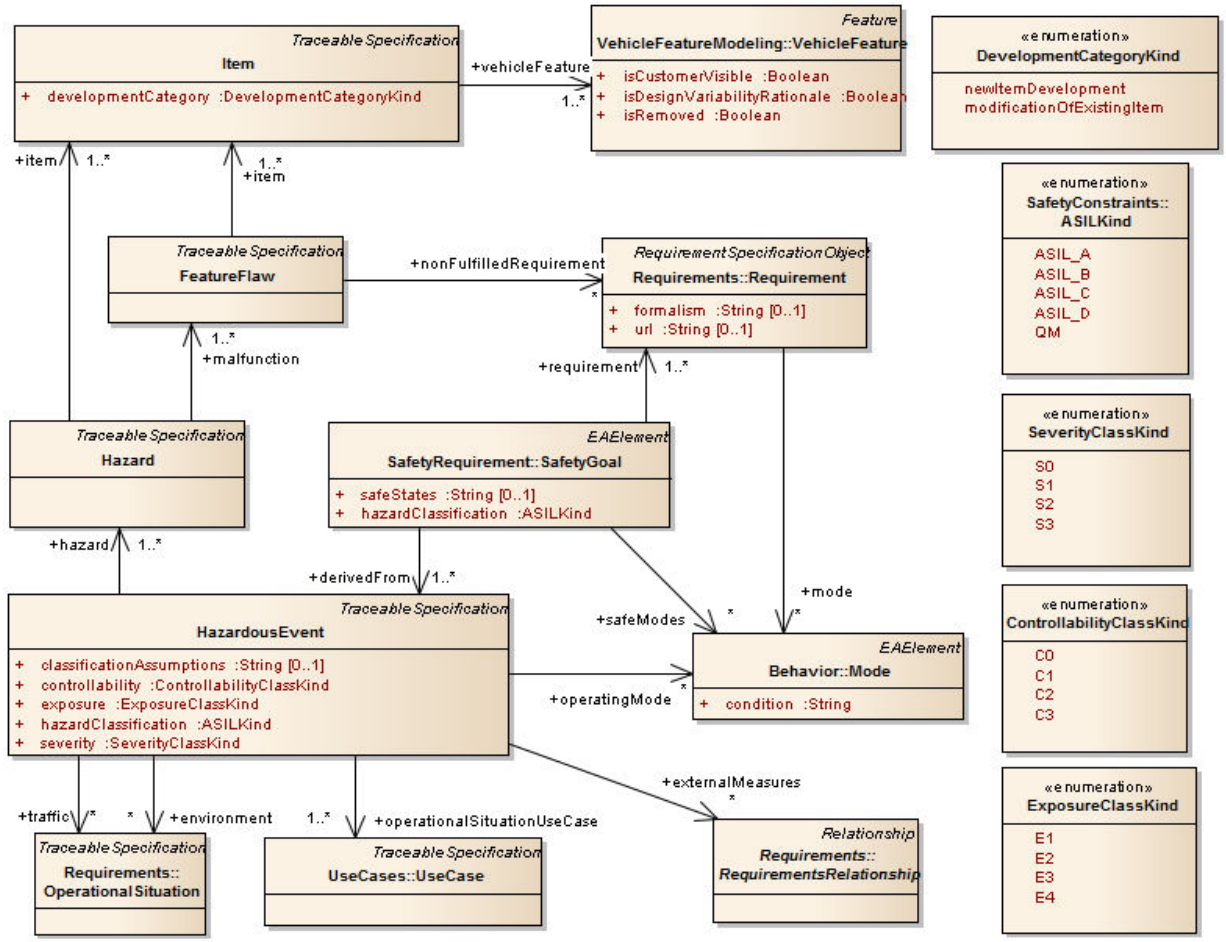

Figure 5: EAST-ADL Dependability Package [44]

As it can be seen in the figure, the basic artifacts needed for expressing safety activities, like for instance hazards, hazardous events and safety goals, are already included. For this work task it

should be the objective to reuse as much as possible the already existing content provided in EAST-ADL.

Furthermore, EAST-ADL (version 2.1.9.1) includes Safety Case description capability in the SafetyCase Sub-Package [44], shown in Figure 6.

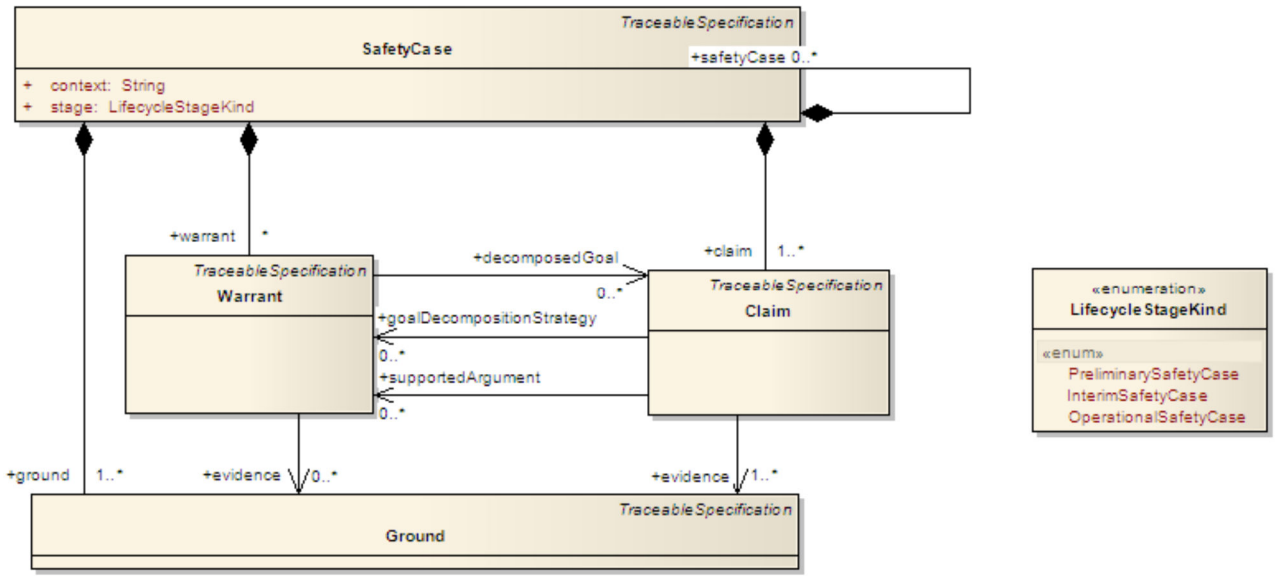

Figure 6: EAST-ADL Safety Case [44]

These various elements are described in the following subsection [EAST-ADL Domain Model Specification 2.1.9.1)] and their class relations are shown in detail in Figure 7:

Claim:

A Claim represents a statement, the truth of which needs to be confirmed and which has associations to the strategy for goal decomposition and to supported arguments. It also holds associations to the evidences for the SafetyCase.

Ground:

Claim is based on Grounds (evidences) - specific facts about a precise situation that clarify and make good the Claim.

Ground represents statements that explain how the SafetyCase Ground clarifies and make good the Claim.

Ground has associations to the entities that are the evidences in the SafetyCase.

LifecycleStageKind:

The SafetyCase should be initiated at the earliest possible stage in the safety program so that hazards are identified and dealt with while the opportunities for their exclusion exist.

The LifecycleStageKind is an enumeration meta-class with enumeration literals indicating safety case life cycle stage.

SafetyCase:

SafetyCase represents a safety case that communicates a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a given context.

Safety Cases are used in safety related systems, where failures can lead to catastrophic or at least dangerous consequences.

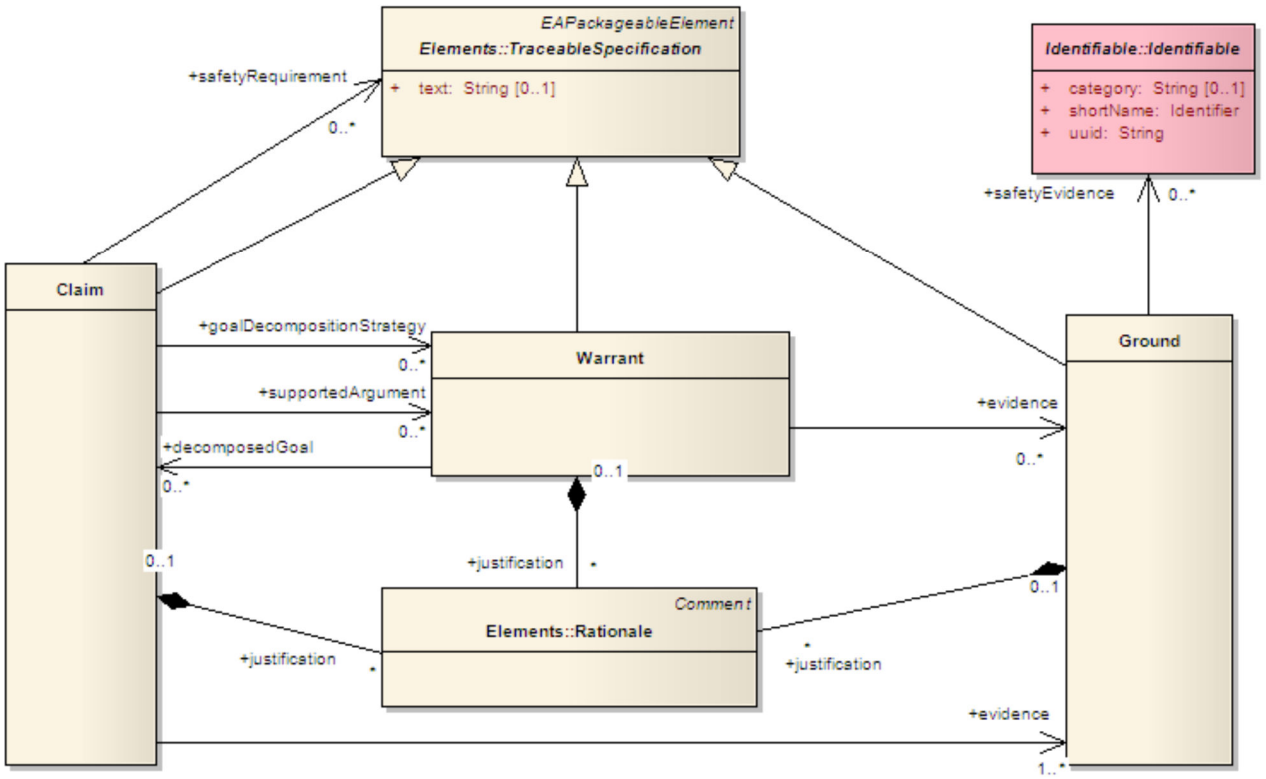

Figure 7: EAST-ADL Ground, Warrant and Claim Diagram [44]

The structures shown in Figure 7 are in turn based on the Toulmin Model of Argumentation defined in Stephen Toulmin's 1958 work "Uses of Argumentation" [2], the main concept of which is shown graphically in Figure 8.

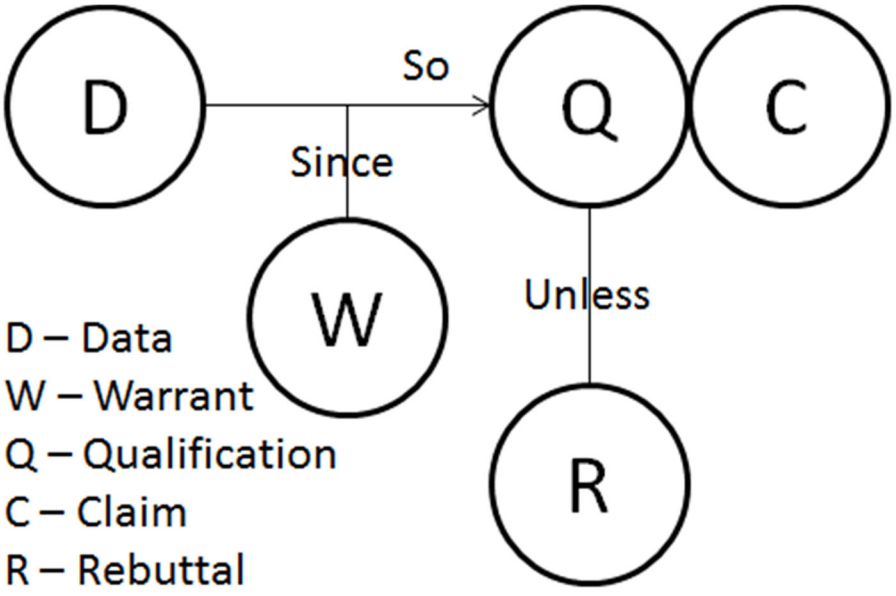

Figure 8: Toulmin Argumentation Concept Elements

Expressing context, which is a central concept of safety argumentation, and differentiating assumption from strategies or justifications are difficult or unclear using this scheme. Furthermore, Toulmin argumentations as described here follow a forward facing line of argumentation, i.e., the claim is

valid unless refuted, which may be useful when speaking of innocence in a court of law, but not conducive to good engineering in the safety critical engineering domain. We have chosen to expand this capability using GSN, to specifically offer a few extra elements such as context, assumption and justification elements, as will be seen in the following section.

7.2 Proposed extensions to EAST-ADL

Investigating the current capability to express safety cases in EAST-ADL, as shown in the previous section 7.1, showed that there is a potential need for extensions, according to the description given in Section 6. Our research into safety case modeling has favored the Goal Structuring Notation (GSN), which was introduced in Section 6, and has already been successfully used in the nuclear, aerospace and railway domains. Extension of EAST-ADL to include Elements of GSN has already been suggested in several ongoing research projects, such as the proposed safety case extensions for EAST-ADL2 in the ATTEST project. The approach followed in this project is most similar to our exploration and findings and will be used here where suitable to avoid repetition. The potential extensions together with their rationale are described in the following.

Introduction of Safety Case Class with GSN Notations

Instead of using the current SafetyCase we support the extension of EAST-ADL with a **Safety Case** class based on GSN as suggested in [3] and shown in Figure 9.

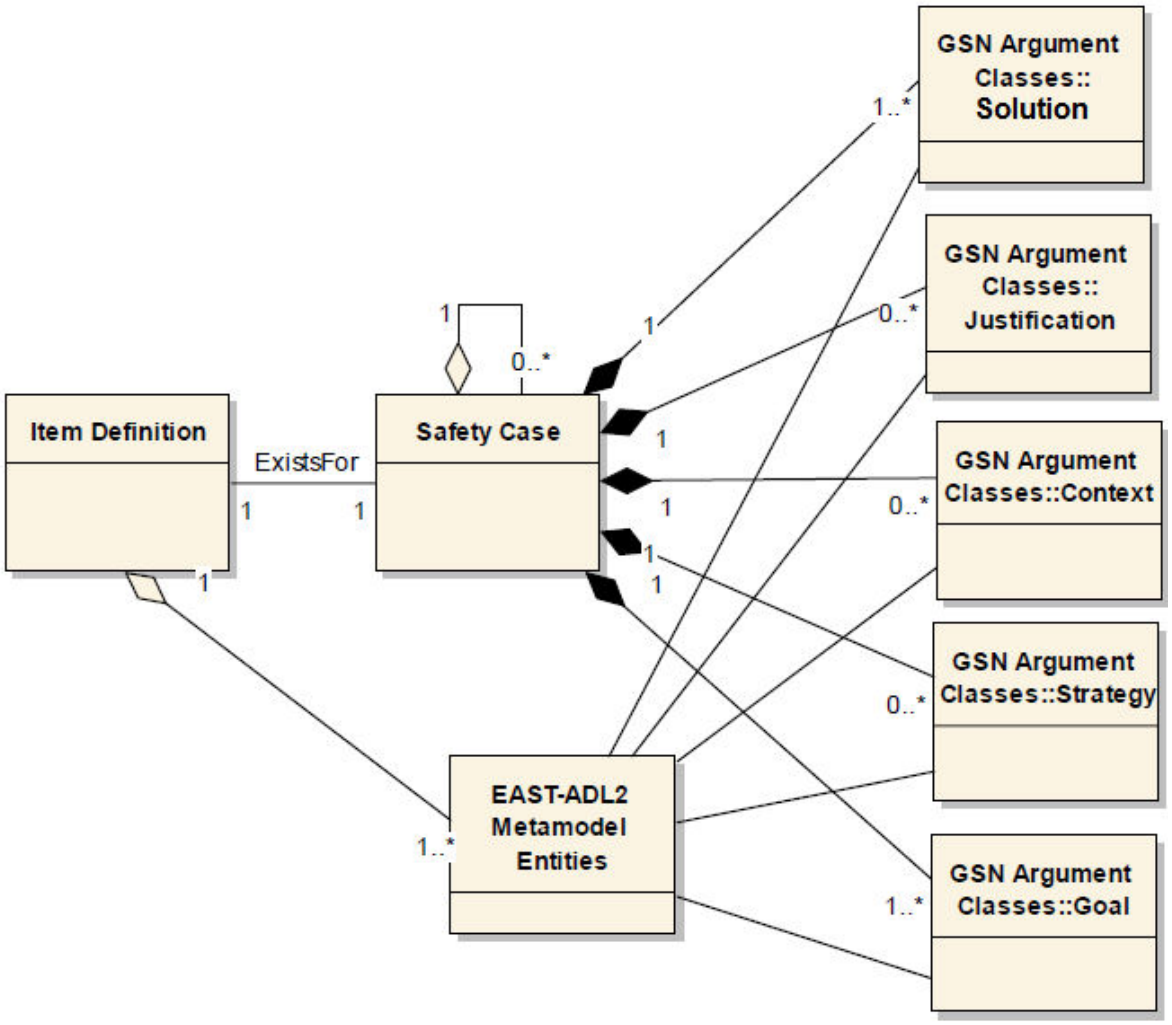

Figure 9: Proposed EAST-ADL2 extension for modeling safety cases in ATESST Project[3]

The Safety Case class is centered on the Safety Goal, which can be decomposed directly into 2 or more goals or indrectly via the use of a strategy. Each goal shall resolve to at least one solution. This rule can be used in section 6.3 to test safety cases for completeness and consitency. Contexts and justification are presented and assumptions can be included as well, as seen in the internal class diagram, shown in Figure 10 with the safety goal forming the center of the safety case structure. The elements and relations shown in Figure 10 were previously explained in section 6.2.

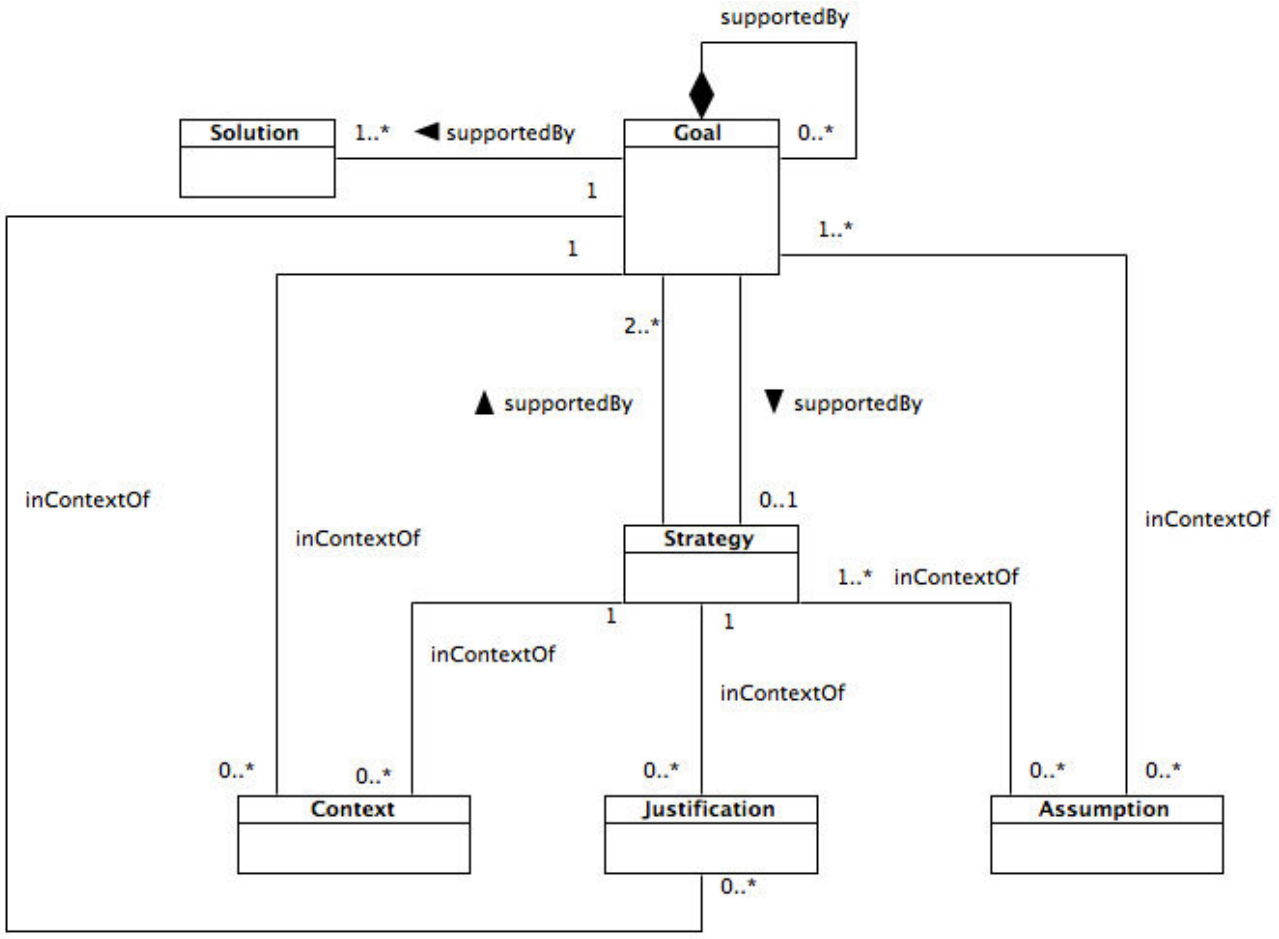

Figure 10: Internal Structure of proposed Class Safety Case

Linking Safety Case Class to EAST-ADL Models

An especially important element of the class Safety Case is the **Solution** entity, which represents any information that supports or, in its ultimate form, proves that the Goal it is connected to is achieved. As such, the information can be of many types. In Figure 11 a class diagram of **Solution** is presented which shows how the **Solution** entity can be specialized to hold the wide array of information that can support a claim. The second level in the hierarchy can consist of general EAST-ADL classes that could supply this information, as shown.

The safety case argument can be seen as consisting of two general branches; the product safety argument and the SAFE process argument. The later part of a safety case argument is considered to be out of scope of the EAST-ADL metamodel since it is supposed to be independent of methodology. However, such process parts as are covered in activities of WP6 can be used in this argumentation as well, i.e. support of assessment activities and application rules etc.

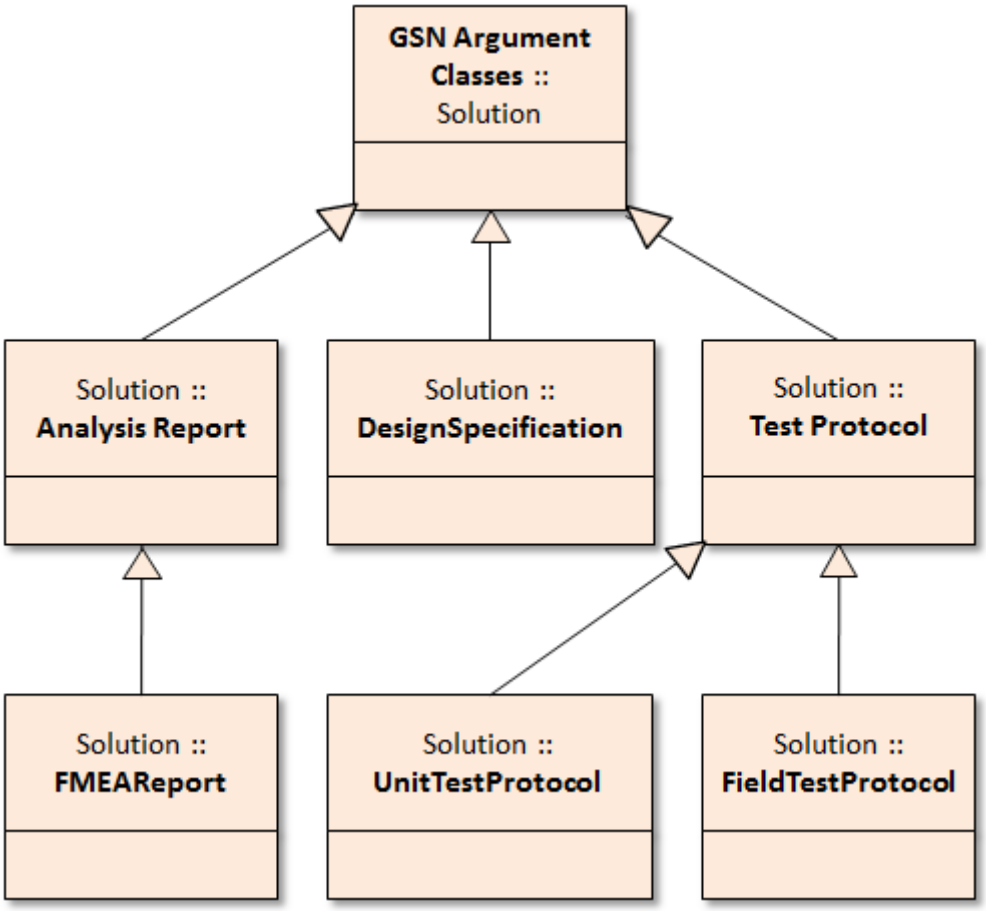

Figure 11: Possible links from EAST-ADL to Solution Element

There are several alternatives to integrate the safety case package to the EAST-ADL Meta-model:

1- A collection of EAST-ADL entities are associated with the safety case entity.

2- EAST-ADL2 entities are directly associated with the safety case entity.

3- GSN entities are directly associated with EAST-ADL entities.

While the first two alternatives seem easier the traceability they allow is also much limited. Traceability is a central as well as mandatory aspect of safety-critical development and as such we suggest following alternative 3 were possible.

A potential first instance of this alternative would be to link many of the output results of the EAST-ADL V&V (verification and validation) package to the Safety Case Solution element, as shown in Figure 9 and Figure 11 and discussed in section .

8 Contribution to the SAFE Meta-model

Within this section the contribution of WT 3.4 to the collective SAFE/SAFE-E Meta-model is described. At the beginning an overview about the model is given which is followed by the detailed description of the classes and interconnections for the simple class option.

8.1 Overview

The contribution of this work task is mainly captured in two class diagrams of the SAFE Meta-model created in Enterprise Architect. In the first diagram, which is shown in Figure 12, the artifacts needed for the hazard analysis and risk assessment and their interconnections are modeled [48].

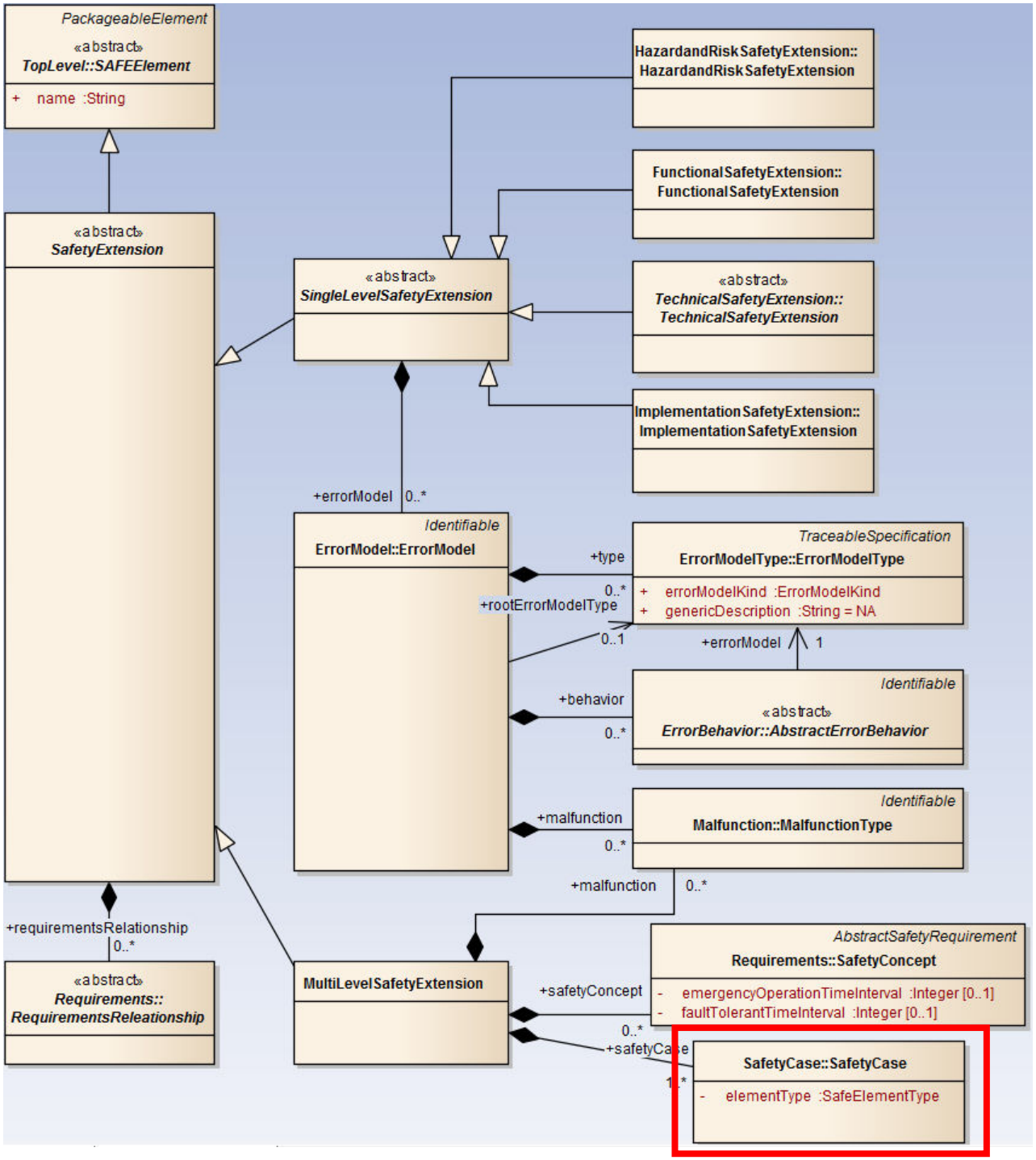

Figure 12: SAFE Meta-model Extensions Overview with SafetyCase highlighted

The internal structure of the SafetyCaseExpression Package [46] is shown in the following figure.

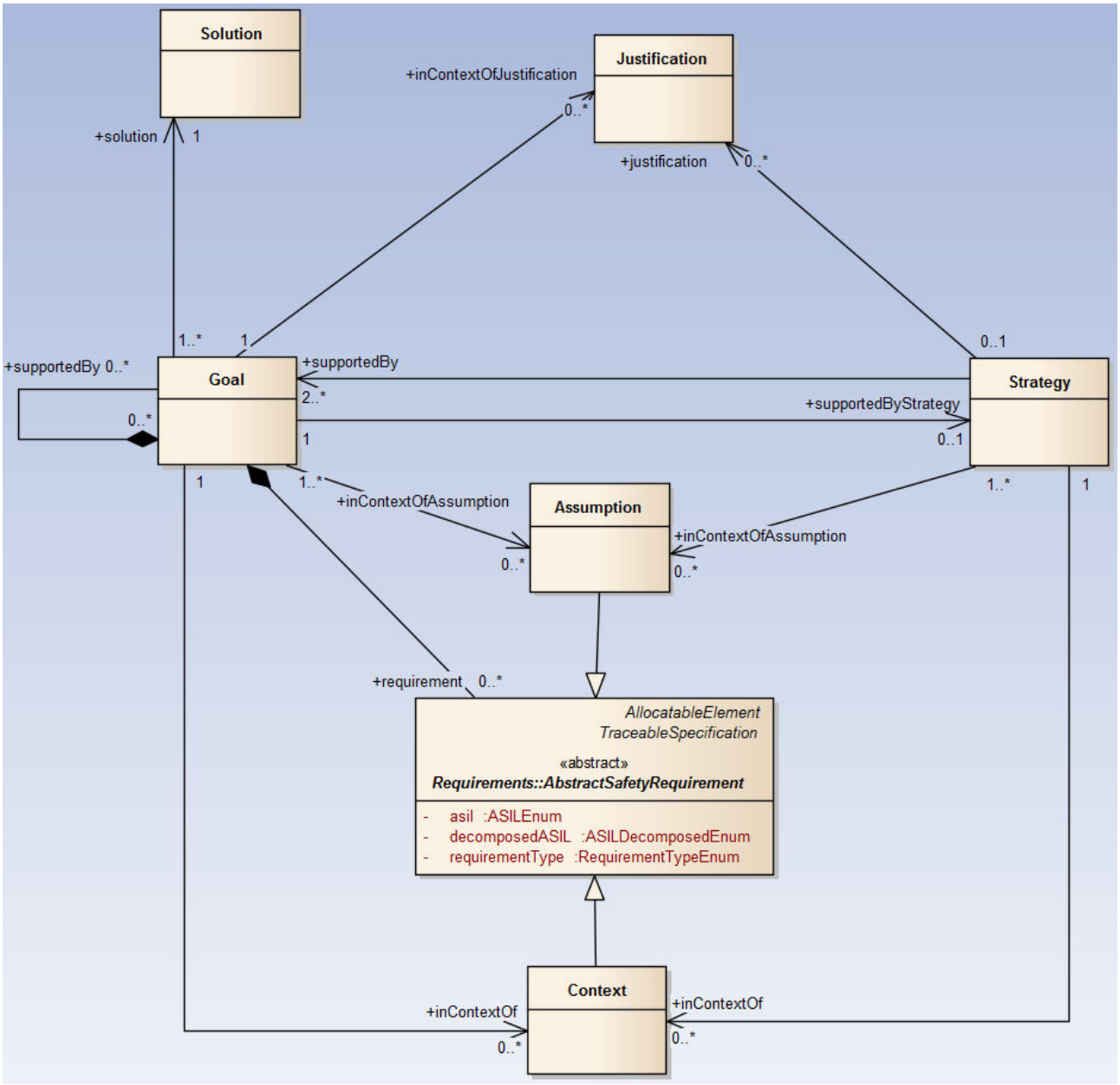

Figure 13: Internal structure of SafetyCaseExpression SAFE Meta-model

As it can be seen there are various elements originating in the current EAST-ADL version that can be reused for the SAFE Meta-model. In case of referencing an element it is assumed that the attributes defined for the class in EAST-ADL are inherited.

8.2 Proposed Interaction with SAFE Meta-model Elements for Documentation

Aside from being useful for developing safe systems, a safety case's original and ongoing purpose is to document the correct development of the safety-critical product and the fulfillment of all safety goals.

By assigning the correct safety case element to each used artifact and joining the artifacts in the safety case context through the appropriate relations it is possible to generate safety case reports

encapsulating the information required for proving fulfillment of the safety goals in a comprehensible and defensible manner. The depth of the reports and the degree of automation depends on the level of integration of the safety case elements into the generated Meta-model artifacts as shown in section 7.2. In the following it is explicitly not recommended using GSN on an "atomic level" on individual SAFE Meta-model artifacts, but rather to use GSN to explain the structure of a safety case report, based on the concepts of the SAFE Meta-model.

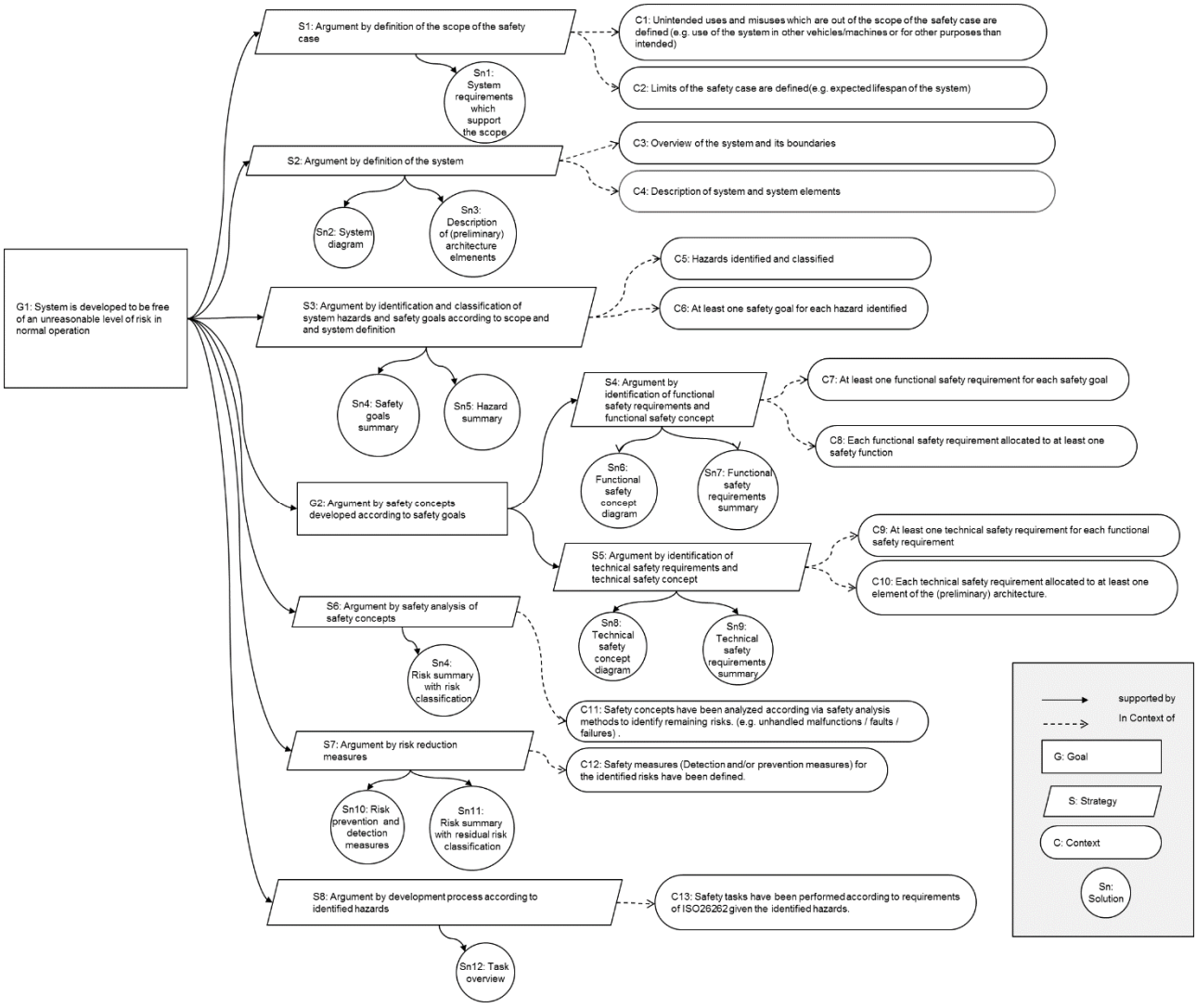

Figure 14: Strategies of a safety case report modeled in GSN exploiting the SAFE MM

Figure 14 illustrates the strategies of a safety case report exploiting the concepts defined in the SAFE Meta-model. The report would cover the following areas, as proposed in [5]:

- Scope
- System Description
- System Hazards
- Safety Requirements
- Risk Assessment
- Hazard Control / Risk Reduction Measures
- Safety Analysis / Test

- Safety Management System
- Development Process Justification

The last two areas are out of the scope of this concept because the safety management system and the development process specification are not in the scope of the SAFE-E project..

In the following sections we propose how these areas can be generated based on concepts of the SAFE Meta-model.

8.2.1 Area: Scope

The scope can be generated based on the items associated to the hazard and risk analysis. The following picture shows the relevant part of the SAFE Meta-model [46]:

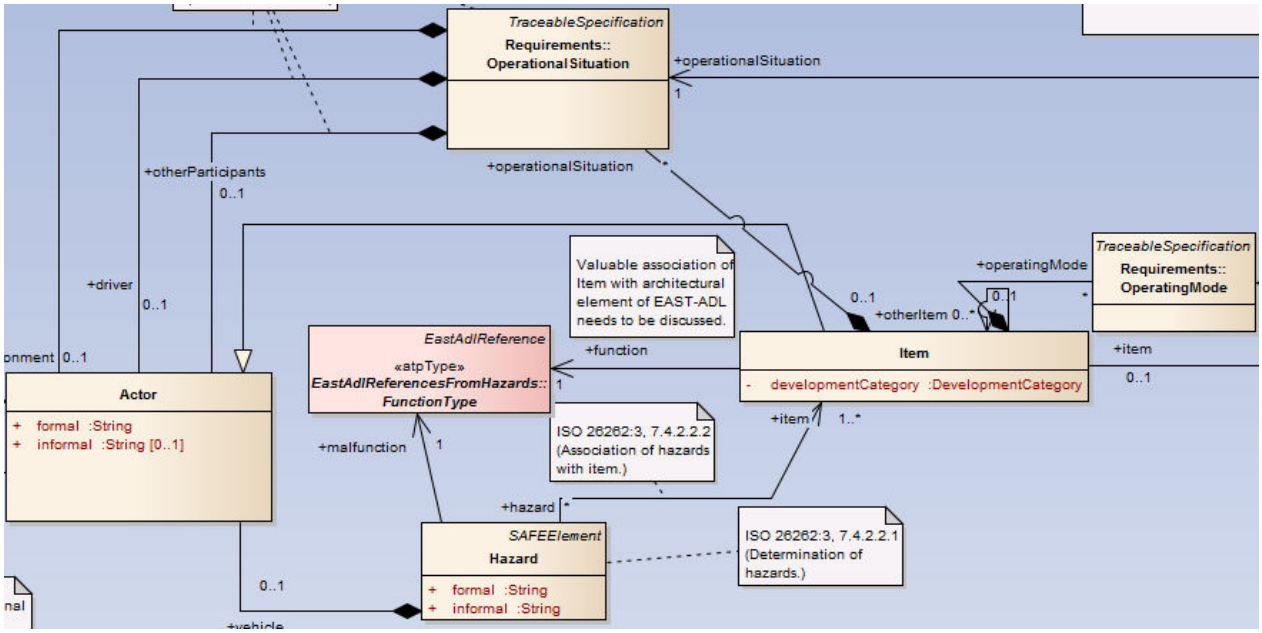

Figure 15: SAFE Meta Model Concepts for Scope

The following SAFE Meta-model concepts should be provided in the scope area of the safety case report:

- Requirements which support the scope of the safety case, e.g. in terms of unintended uses, misuses, limits or expected system life span
- Operating Modes
- Operational Situations

8.2.2 Area: System description

The system description should not provide full design detail but rather support the reader of the safety case report to make sense of the system hazards and requirements which are later described in the report.

The following SAFE Meta-model concepts [46] should be provided in the system description area of the safety case report:

- System descriptions
- Component descriptions
- System diagrams

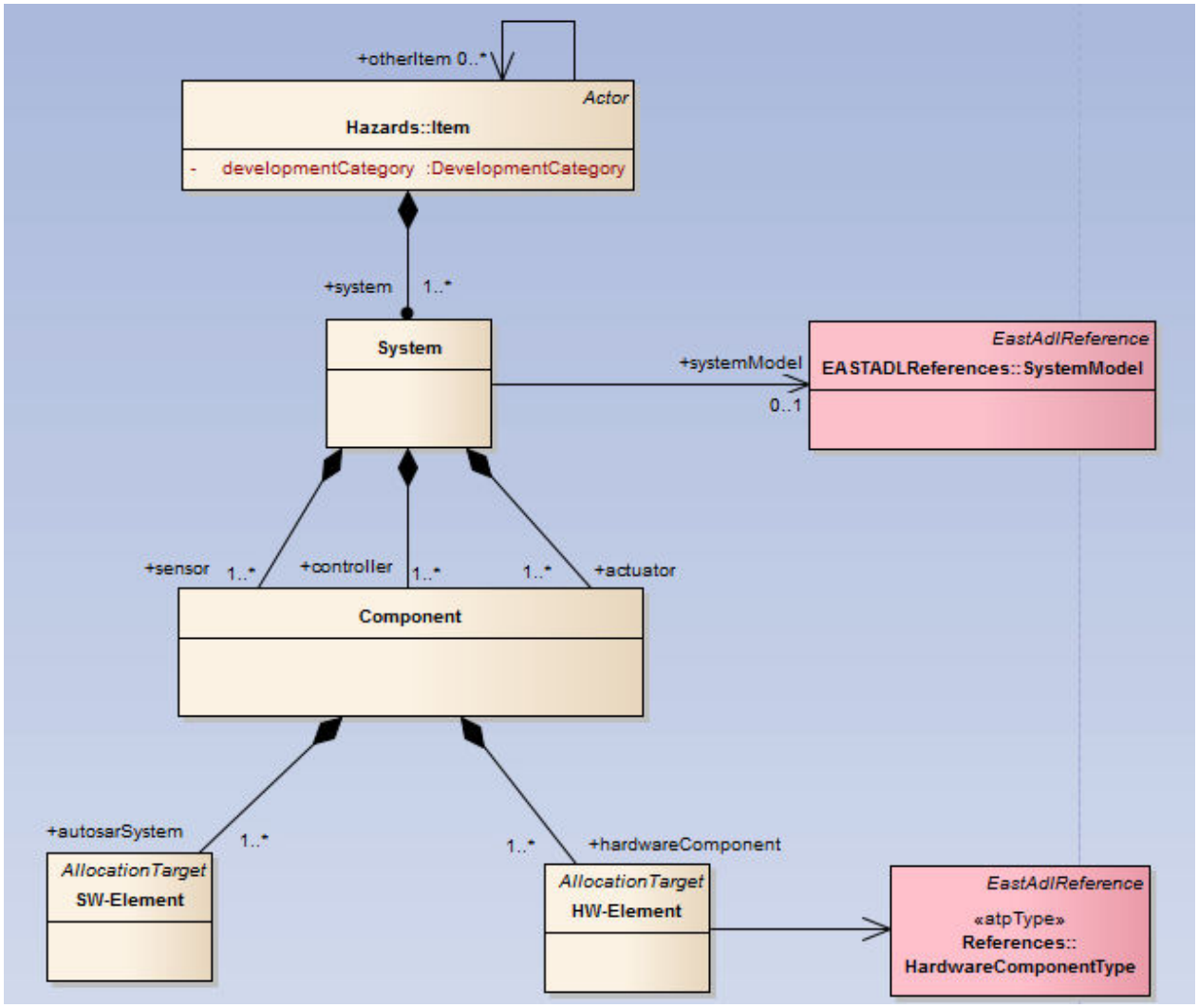

Figure 16: Item Architecture of the SAFE Meta-model

8.2.3 Area: System Hazards

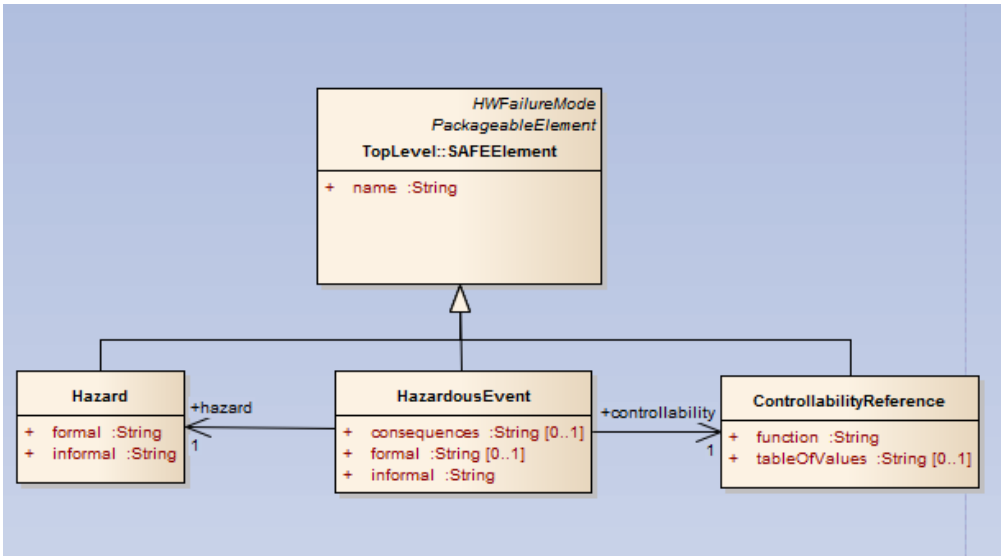

Figure 17: Hazards as defined in the SAFE Meta-model (Excerpt)

The system hazards area should list the key hazards posed by the system in order to summarize the identified hazards [5]. Hazards are an explicit part of the SAFE Meta-model as depicted in the previous figure.

The following SAFE Meta-model concepts [46] should be provided in the system description area of the safety case report:

- Hazard

8.2.4 Area: Safety Requirements

In the area safety requirements a number of sources for safety requirements must be taken into account [5] and are interpreted in the context of the SAFE Meta-model [46] as outlined in the following points:

1. Safety requirements derived from hazard analysis (including safety goals as top level safety requirements)
2. Safety requirements which are the results of refinements from higher level safety requirements
3. Safety requirements which have been given directly by the customer or safety standards

The following picture shows the relevant excerpt of the SAFE Meta-model [46].

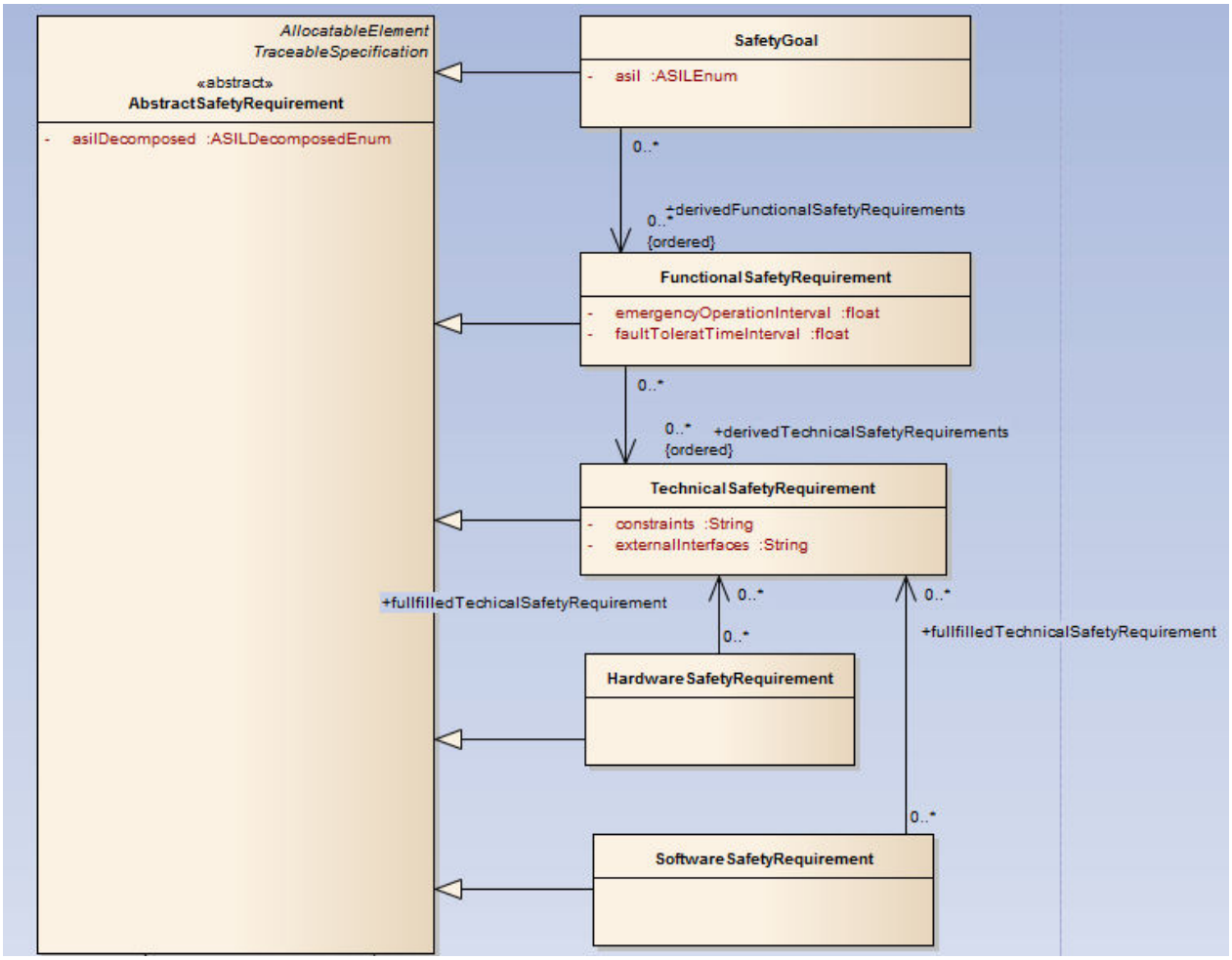

Figure 18: Safety Requirement Expression as defined in the SAFE Meta-model (Excerpt)

The following SAFE Meta-model concepts should be provided in the system description area of the safety case report:

- Safety Goals
- Functional Safety Requirements
- Technical Safety Requirements
- Hardware Safety Requirements
- Software Safety Requirements

8.2.5 Area: Risk Assessment

The area risk assessment of the safety case report aims to describe the level of residual risk which is left after risk reduction measures have been applied [5].

The initial risk associated with a hazard is captured in the SAFE Meta-model in the hazardous event. However the residual risk after the implementation of risk reduction measures is not yet part of the SAFE Meta-model [46] and can therefore not provided in the safety case report.

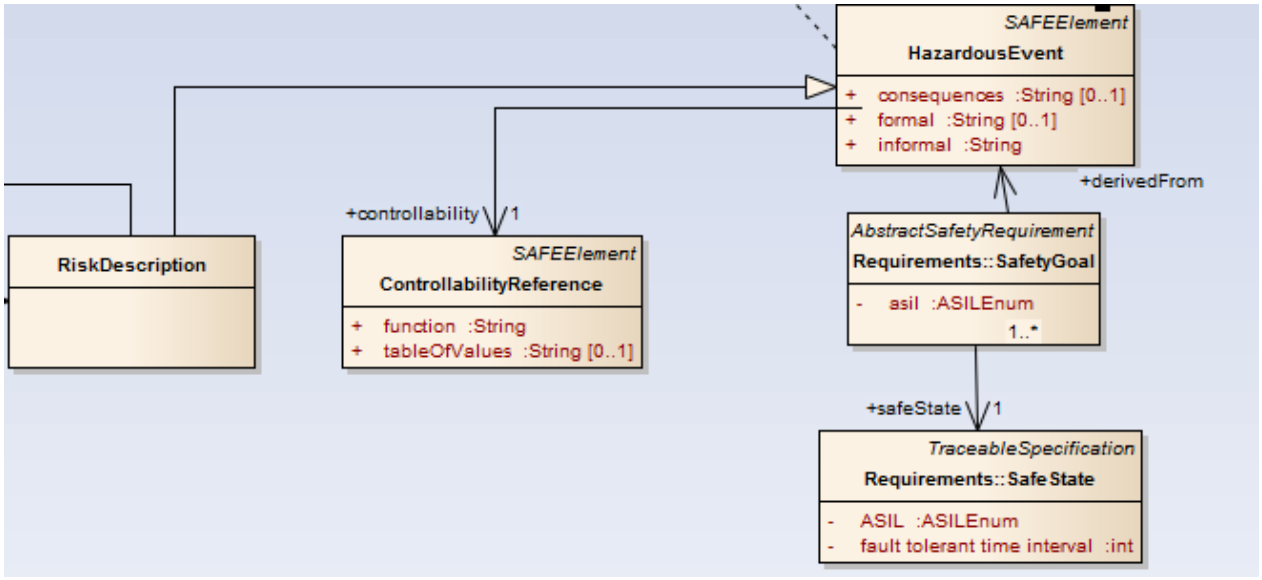

Figure 19: Initial risk description in the SAFE Meta-model

The following SAFE Meta-model concepts should be provided in the system description area of the safety case report:

- Hazardous Event

8.2.6 Area: Risk Reduction Measures

The area risk reduction measures describes means for reducing the probability of hazard occurrence or mitigation of hazard occurrence which have been integrated in the system design [5].

The following figures show excerpts of the SAFE Meta-model [46] which could be exploited for this information.

The AnalysisLevelElement contributes to reducing or mitigating hazards by addressing the safety requirements which have been allocated to them.

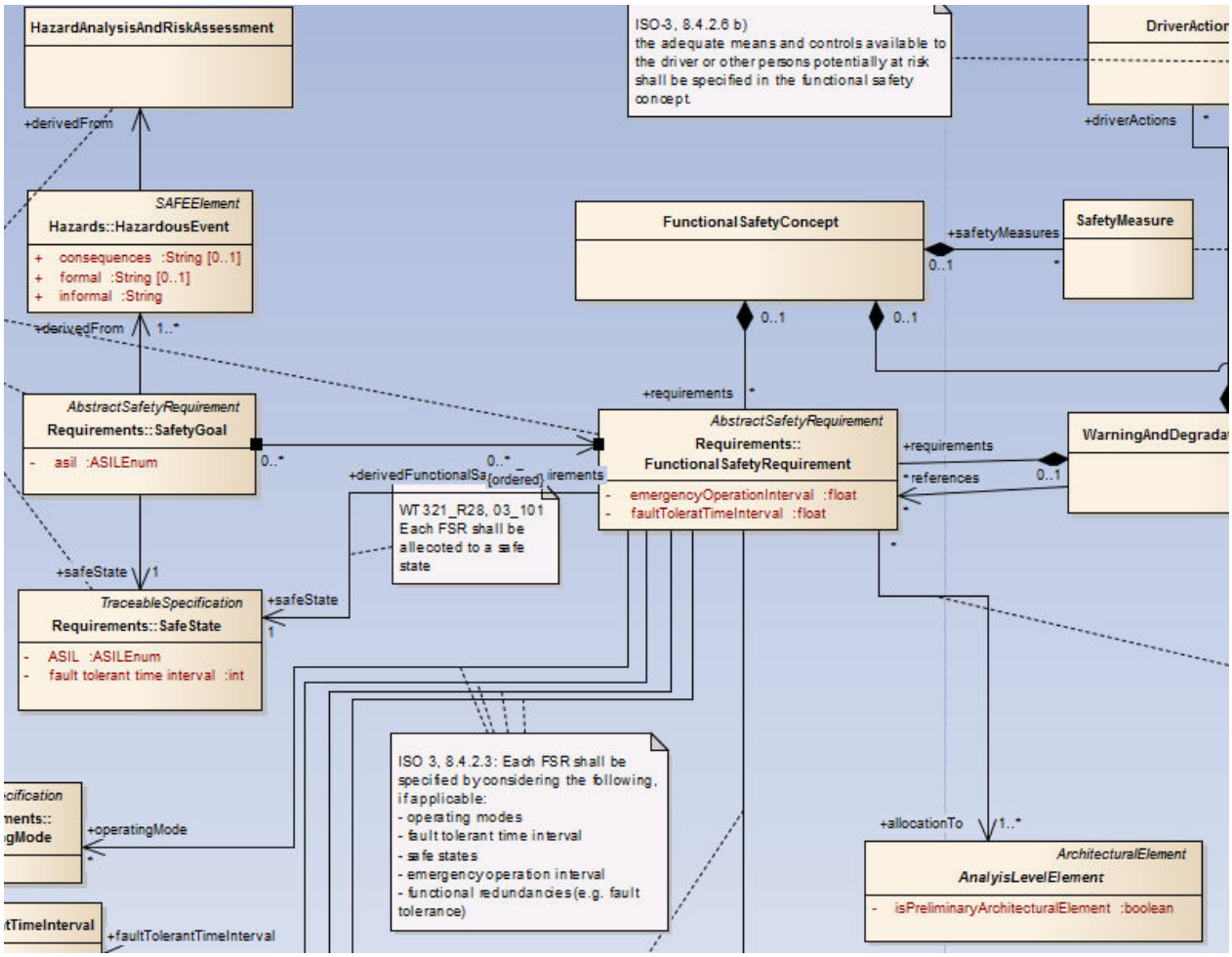

Figure 20: Functional Safety Concept of the SAFE Meta Model (Excerpt)

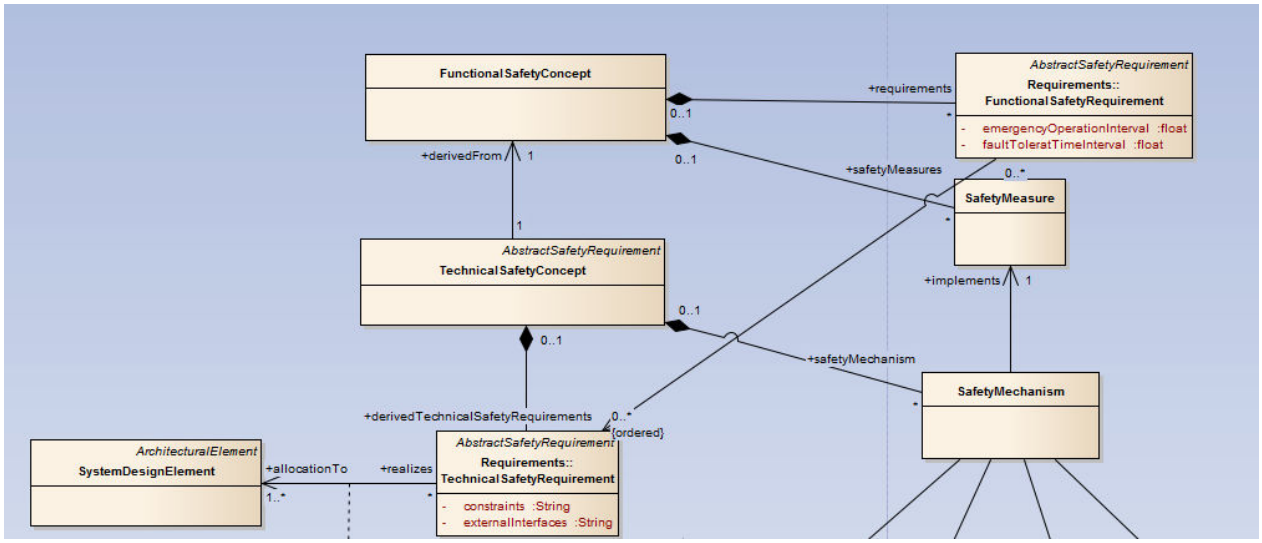

Figure 21: Technical Safety Concept of the SAFE Meta Model (Excerpt)

The following SAFE Meta-model concepts [46] should be provided in the risk reduction measures area of the safety case report:

- AnalysisLevelElements, which address one or more FunctionalSafetyRequirement, which have been created to reduce or mitigate risks from one or more Hazards

- SystemDesignElements, which address one or more TechnicalSafetyRequirements, which have been created to reduce or mitigate risks from one or more hazards

8.2.7 Area: Safety Analysis

The area safety analysis provides evidence that the risk reduction measures which have been reported in the previous area are sufficient [5]. Examples for means to achieve this are safety analysis methods (e.g. FMEA), inspections or in-service evidence. As in the other areas of the safety case report only a summary is required while details can be maintained in other documents.

The following figure shows an extract of the SAFE Meta-model [46] that support analysis such as fault tree analysis and FMEA.

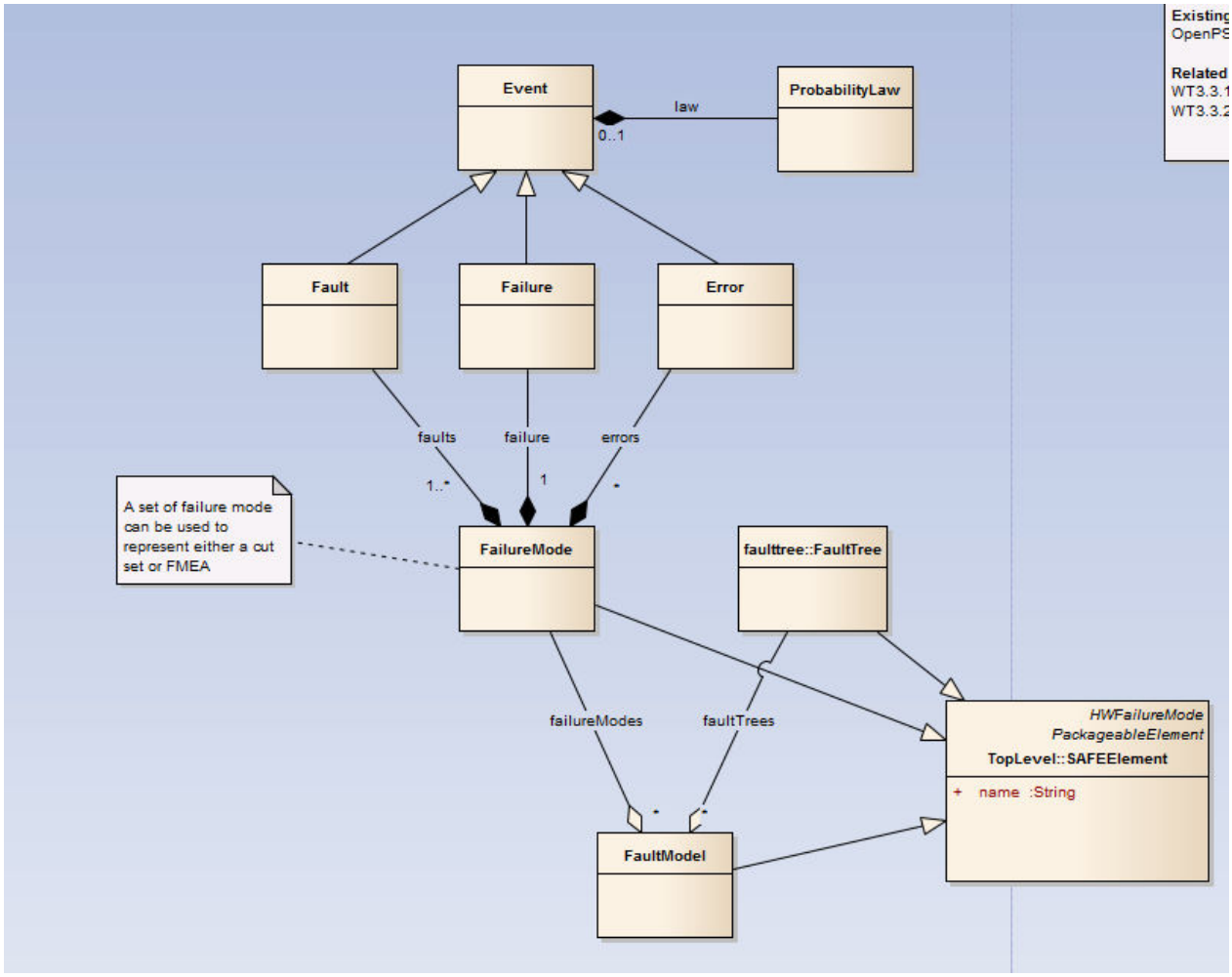

Figure 22: Concept for FTA and Failure Modeling in the SAFE Meta Model (Excerpt)

A potential approach to document that the risk reduction measures defined previously are sufficient is to check if every failure which is identified in an FMEA can be associated with a risk reduction measure.

The following SAFE Meta-model concepts [46] should be provided in the safety analysis area of the safety case report:

- Failures identified in an FMEA together with the associated risk reduction measures (e.g. architecture elements or requirements as prevention measures or test cases as detection measures)

However linking risk reduction measures with failures is not yet supported by the SAFE Meta-model. This is an obvious point for improvement of the SAFE Meta-model.

9 Safety Case Patterns

The idea of design patterns was original proposed by Christopher Alexander [15], an architect who wrote several books on the field of urban planning and building construction, and who pioneered the notion that users know more about their needs and rationales than architects working abstractly. The concept of design patterns is a universal approach to describe common solutions to widely recurring design problems. A pattern is an abstract representation for how to solve a general problem which occurs over and over in many applications. Describing proven solutions as patterns provides a good documentation for these solutions and makes them more accessible for future use in new systems. Ever since, this concept has been applied to many different domains including hardware and software design.

Patterns support and help designers and system architects choose suitable solutions for design problems, and thus also found resonance in the development of safety-critical embedded systems, where a high level of confidence in the implemented solutions (in this case through repeated testing and use in field) is highly desirable. Reusing proven solutions also makes it possible to reuse the arguments originally used to prove the safety of these solutions and thus gives rise to the concept of safety case patterns, as discussed in [14] and [10]. A holistic approach to combine development artifacts and argumentation elements into a pattern library was introduced in [52] and further discussed in [58]. The latter work, by M.Khalil et al., forms the basis for the work presented here.

Section 9.1 gives the necessary background and motivation. Section 9.2 explains our approach and gives an overview of the data model of the pattern library element as well as the usage process and most important attributes, supported by an example. In section 9.6 we summarize the approach and discuss the impact of this contribution.

9.1 Introduction

In practice, the reuse of architectural designs, development artifacts and entire code sequences is widely-spread**,** especially in well-understood domains. This also holds true for the development of safety-critical products, and extends to reusing the corresponding safety-cases aiming to document and prove the fulfillment of the underlying safety goals. This, however, is marred by several problems [52]:

- Most analyses (FMEA, FTA, etc.) have to be performed at system level, yet the components/ measures / safety mechanisms themselves need to be reused independently,
- and are not tied in any structured manner to other elements needed to provide the relevant context.
- Safety-cases in the automotive domain are not well integrated into architectural models and as such
- they do not provide comprehensible and reproducible argumentation
- nor any evidence for the correctness of the used arguments.

Reuse in a safety-critical context, and particularly the reuse of safety cases, is mostly ad-hoc, with loss of knowledge and traceability and lack of consistency or process maturity being the most widely spread and cited drawbacks [14], [52].

The use of patterns in the development of safety-critical products is already in wide spread use. Catalogues exist [43], [32] that discuss highly organized and well-known safety mechanisms, such as comparators or “1outof2” voters. Safety case templates can be generated, stored and reused for many categories of patterns. Some of these patterns are truly abstract and tackle higher system description levels, such as the “High Level Software Safety Argument Pattern Structure” presented in [8], while some can target a certain context, such as the use of Commercial-Off-The-Shelf (COTS) Components in a safety-critical context [7]. Patterns of safety cases for well-known problems have been suggested in academic literature [14], [41], [16], [6], [8], [7] and [43]. Further discussion is provided in [49] and [52].

The reuse of safety mechanisms can be made both simpler and more robust through the encapsulation of all information into a consistent structured package, which can then be stored in a library element, along with the corresponding safety case to support it [52]. Dealing with non-functional requirements, especially safety, at later development stages is difficult and highly costly [47]. Front-loading these aspects into an integrated solution environment and properly leveraging the added value of a model-based approach to solving this problem requires automation using adequate tool support. This does not yet exist, despite these activities and analyses having mostly become part of the day-to-day business of safety-critical development [16]. The structure and usage of this reusable comprehensive library element concept in a seamless model-based development tool is the focus of this chapter.

9.1.1 Background and related work

While the Gang of Four book [17] has been the most popular work on design patterns over the last two decades, there have been several attempts in the literature to adapt this concept in many fields of system design. In 1987 Cunningham and Beck presented five patterns for designing windows-based user interfaces with Smalltalk [18]. Around the same time, Jim Coplien began to document C++ Idioms that represent specific constructs like patterns for C++, which were first published in a book in 1991 [19], with newer additions as a paper in 1997 [20].

From 1990 to 1993, several papers addressing the use of design patterns in object oriented programming were published at the OOPSLA (Object-Oriented Programming Systems, Languages, and Applications) conference. In 1994, the Hillside group (see http://hillside.net) organized the first PLoP (Pattern Languages of Programs) annual conference. The revised papers from PLoP are normally published in the book series “Pattern Languages of Program Design” (see e.g. [21]). Meanwhile, Buschmann et al. published the book “Pattern-oriented software architecture: a system of patterns” [22] which includes a collection of relatively independent solutions to common design problems represented as a catalog of design patterns.

Another well-known work is the catalog presented by Bruce Douglass in [23]. This catalog includes a set of patterns for real-time embedded systems. The presented patterns deal with real-time design issues like concurrency, resource sharing, distribution, and SAFE and reliable architecture.

The concept of design pattern has become an important area of research in many fields like: fault tolerance [24], telecommunications [25], embedded systems [26], [27], security [28], [29], and many other fields. Each of these fields has its own patterns and sometime its own representation, but all follow the basic principle of design patterns. Works seeking to structure and facilitate the selection of architectural patterns and measures adequate for safety-critical requirements have already been presented [32], but the selection process is still highly manual and case dependent, and focuses on individual patterns as a guideline with no link to a holistic view.

On the other hand, there is extensive work on formalized reasoning about safety cases and safety concept argumentation [38], [42], [13], as well as the use of patterns for safety cases [14], which culminated in a standardization of the structured notation known as GSN [12]. An analysis of industrial safety cases yielded a finite set of patterns [41] from which it is possible to construct a pattern library of problem types and their respective solutions and use it to automate the generation of and argumentation about safety concepts as well as safety cases, for which a proof-of-concept was implemented in the research CASE tool AutoFOCUS3 [30, as discussed in section 9.4. Safety case pattern catalogues and the capability for compositional argumentation is suggested in several research works such as [6], [7], [8], [9], [10], [11] and [16].

9.1.2 Safety case pattern templates

Safety case templates can be generated, stored and reused for many categories of patterns, such as:

- Strategy (Problem solving) patterns

- Design patterns
- Constraint patterns and Safety Mechanisms
- COTS – Commercial Off The Shelf Components

9.1.2.1 Problem solving strategy patterns

As discussed in [41] the range of problems developers in a domain face and subsequently the algorithms they favor in solving them are not unlimited. By identifying recurring paradigms and analyzing them it is possible to generate templates of their safety cases. The strategy node is the core element of the actual argumentation in a problem pattern safety case template, supporting the justification of goals being fulfilled by sub-goals. By applying strategy patterns, it is possible to build argumentations using only accepted justifications. This way confidence in the correctness of the argument can be increased. Still the appropriateness and correct use of a pattern has to be evaluated before trusting a safety case, but the question of the fundamental validity of each single argumentation step itself need not be argued. On the long run, it is desirable to establish general and domain-specific patterns as a pattern library for a faster and easier creation of safety arguments [41].

Example Pattern: Logical transformation

The goal is a logical combination leading to desirable or undesirable situations. It is possible to generate a safety case skeleton that represents the pattern of avoiding a situation in which the constraint is violated, e.g. to keep the gap to the forward car larger than or equal to a minimum safe distance in an adaptive cruise control. This is simplified by transforming the goal into avoiding a gap which is smaller than the safe distance [41], as shown in Figure 23.

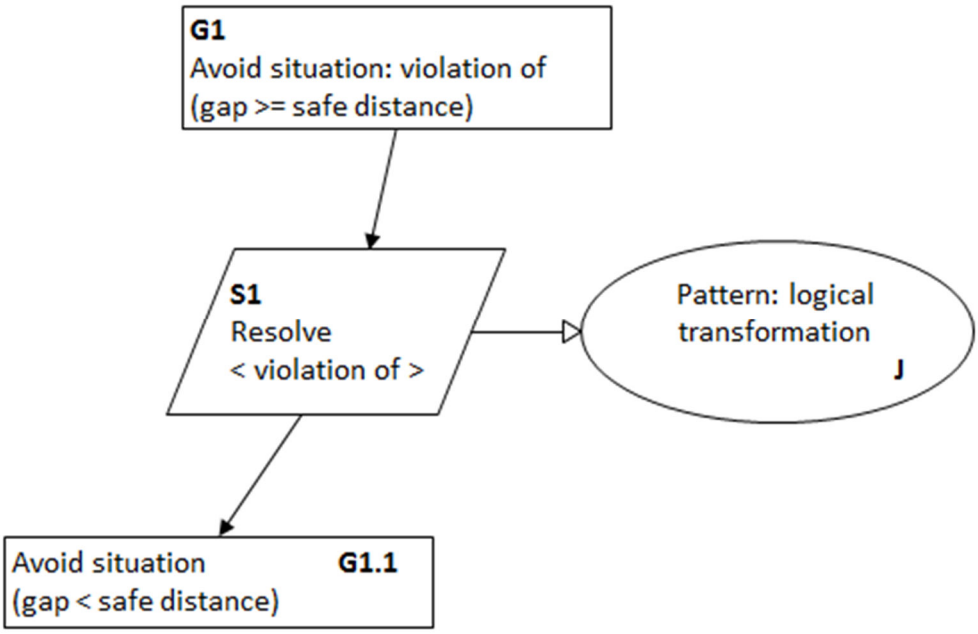

Figure 23: An example usage of the logical transformation pattern [41]

9.1.2.2 Design Patterns

The ISO 26262 safety standard [1] requires that component architectures, independently of their functionalities, display certain characteristics or adhere to constraints. Some of these such as modularity, simplicity, and an adequate level of granularity and encapsulation are simply good engineering practices aimed at avoiding failures arising from unnecessary complexity.

Other requirements, such as freedom from interference, which is the absence of dependent failures (cascading and common cause failures) in safety-critical components, are aimed at guaranteeing correct operation of safety-critical functions.

Furthermore, the ISO26262 [1] dictates that "if the embedded software has to implement software components of different ASILs, or safety-related and non-safety-related software components, then all of the embedded software shall be treated in accordance with the highest ASIL, unless the software components meet the criteria for coexistence in accordance with ISO 26262-9:2011, Clause 6." This latter clause mentions freedom from interference and component independence as being the central requirements for such co-existence.

According to ISO26262, there are several ways for components to have freedom from interference, such as:

- be functionally diverse (the use of totally different approaches to achieve the same results);
- be based on diverse technologies (the use of different type of equipment to perform the same result)
- not share common parts or services whose failure could result in a dangerous mode of failure of all systems
- be designed so that the predominant failure mode for common support systems (e.g. power supply) is in a safe direction (i.e. fail-safe)
- not share common operational or maintenance or test procedures
- be physically separated such that foreseeable failures do not affect redundant safety-related systems.

Example Pattern: Redundancy

The guidelines above have in turn matured into various design patterns, many of which revolve around redundancy and partitioning, which is the separation of functions or component elements to achieve a design, which can be used for fault containment to avoid cascading failures. The design patterns vary in their addressed context, structure, and presented solution and can be categorized in many ways. For example for redundancy there exist [32]:

- **Hardware Patterns**: Includes the patterns that contain explicit hardware redundancy. This group contains the following patterns:
    - Homogeneous Duplex Pattern
    - Heterogeneous Duplex Pattern.
    - Triple Modular Redundancy Pattern
    - M-Out-Of-N Pattern
    - Monitor-Actuator Pattern
    - Sanity Check Pattern
    - Watchdog Pattern
    - Safety Kernel Pattern

- **Software Patterns**: Includes the patterns that use software diversity (redundancy) to tolerate software faults. This group contains the following patterns:
    - N-Version Programming Pattern
    - Recovery Block Pattern.

- Acceptance Voting Pattern
- N-Self Checking Programming Pattern
- Recovery Block with Backup Voting Pattern

- **Combination of Hardware and Software Patterns**: Include the following patterns that do not contain explicit hardware redundancy or software diversity:
  - Protected Single Channel Pattern
  - 3-Level Safety Monitoring Pattern

All the above patterns fall under one general paradigm, i.e. redundancy, and as such can be covered by a general safety case pattern template as shown in Figure 24, namely that of solving the lack of confidence in the safety-critical channel using a redundancy strategy. This general template can in turn be instantiated and subsequently further specified and extended to suit the specific case and used solution, for example using context elements.

Decision trees to provide assistance in selecting the suitable pattern have been developed and discussed in [32].

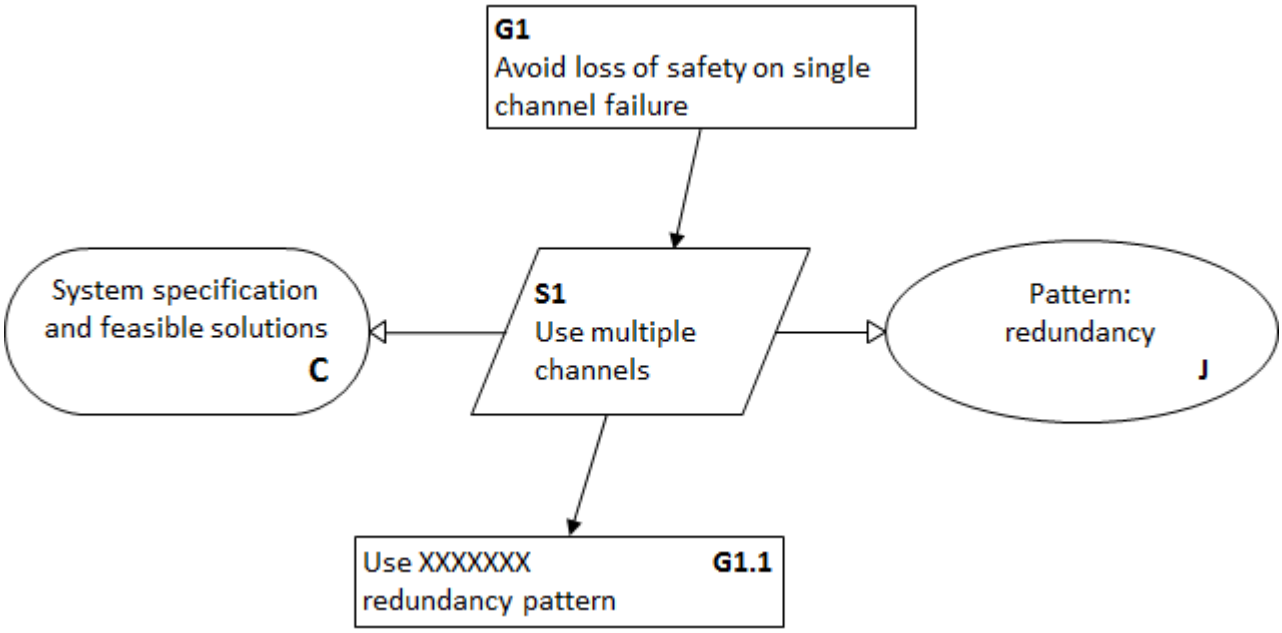

Figure 24: Safety case template for channel redundancy patterns

9.1.2.3 COTS Components

These pattern category targets the re-use of previously developed hardware and software components, including COTS products. The safety cases for these may be built around proofs of validity or the proven in use argumentation in case of sufficient field data. Similarly to safety mechanisms, COTS components can also be accompanied by safety case snippets or templates that can be instantiated as needed and integrated into the safety case at the insertion point of the COTS product.

9.1.2.4 Safety Mechanisms

Safety mechanisms are trusted solutions to repeated engineering problems facing designers and developers of safety-critical systems which have developed to patterns commonly used in the realization of technical safety concepts. A collection of safety mechanisms has been discussed in [43]. Safety mechanisms, their integration into the SAFE Meta-model and the generation of their code have been covered in WT3.6. Similarly to Strategy patterns and design patterns, safety mechanisms can also be accompanied by safety case snippets or templates that can be instantiated as needed

and integrated into the safety case at the generation point. The reuse of safety mechanisms is the focus of the approach discussed in the following sections.

9.1.3 Using safety case patterns

Combining the parts explained in the previous subsections a 5-step approach could be suggested using which it would be possible to quickly construct safety cases from a pattern library.

Step 1: Definition of Safety Goals

Step 2: Identification of problem type

Step 3: Selection of suitable paradigm or design pattern / safety mechanism / COTS Component

Step 4: Creation of safety goal instances

Step 5: Generation of safety case skeleton(s)

Step 6: Link Elements to Safety Case and do a Top-Down Check

Step 7: Testing for safety case / concept completeness and consistency

Step 8: Integration into existing safety cases (with compositional argumentation)

9.1.4 Problem

In our context, a typical reuse scenario would involve broadly reusing an architectural measure or design pattern, e.g., homogenous (hardware) duplex redundancy, in which a vulnerable single channel carrying a critical signal is duplicated to increase reliability, or more precisely one or more of the logical development artifacts employed, such as a comparator function. If the previous safety case is at all reused, it serves as a detached guide of what needs to be provided to close the case, i.e. serving a prescriptive role, which is arguably better than nothing [52]. Yet be it a design pattern or development artifact, a single item does not tell the entire story. For example, to correctly deploy homogenous redundancy, many other aspects have to be covered:

- one has to define the requirements the pattern fulfills,
- refine the requirements and draw up a specification,
- detail a (logical) component architecture,
- and optimize a deployment strategy that guarantees the duplicate components will not run on the same hardware resource.

These steps have to be preceded by feasibility checks or making explicit assumption about the system, e.g., that a second hardware channel is allowed, which is a contextual bit of information. This is not all; to justify reusing this pattern, one would also have to include any tests or information proving that this particular pattern is suitable for the goal it targets, as well as perhaps why this method and not another was chosen, e.g., why just use 2 channels and not a 2-out-of-3 pattern in our example.

Finally, all parts comprising this information, as well as their relations, which are more complex than simple traces, must be captured in a comprehensive and comprehensible manner which should also provide a suitable interface to the environment the reused element will be deployed in. Thus, the reuse of trusted design patterns or safety mechanisms cannot be confined to reusing the central artifact alone. Much of the information, in this case highly critical information, remains trapped in the heads of the developer and if mentioned at all most details remain implicit [52].

This gives rise to the need of some kind of encapsulation of the reusable safety mechanism (with requirements, specification, components, etc.), along with a minimum set of guarantees that can be achieved at that level and support via a solid argumentation. The encapsulation has to be structured, defining a minimum set of necessary information for correct reuse.

9.2 Approach

Using a simplified description of safety mechanisms according to the most common error management subtypes (avoidance/ detection/ handling) we define a pattern library covering known solution algorithms and architectural measures/constraints in a seamless holistic model-based approach with corresponding tool support. The pattern library comprises the minimum set of elements needed for correct reuse, i.e. the requirement the pattern covers, the specification of how one plans to implement it and the architecture elements/ measures / constraints required as well as the supporting safety case template and may include deployment or scheduling strategies, which would then be integrated into existing development environments. This enables an early analysis of hazards and risks, as well as the early adoption of multiple measures at the architectural design level during the concept phase, which is recommended (if not dictated) by many safety standards, e.g.,[1].

Subsequently, fault types can be matched both to probable hazards but more importantly to the problem categories they fall into or are most similar to, from a system architecture design viewpoint. Combining this with known architectural constraints and patterns for solving them, we can thus reason about which types of architectural patterns are relevant for the system under analysis. The fault types, along with their requirements, are bundled with solution arguments, comprising components, their (sub-) architectures, deployment plans and schedules, into pattern libraries, which are rounded up by the corresponding safety-case templates or skeletons to provide argumentation for achieving the goals. Because our scope is safety mechanisms, the artifacts used here for safety argumentation all fall into the product information category of safety evidence [56].

Underlying the approach is the consistent use of patterns, as described in [49], [16], [50], from the categorization of hazard types, over the abstract modeling of the respective safety concepts, and down to their implementation in the system architecture description, with a focus on providing argument chains in a seamless model-based environment. A simple overview is shown in Figure 25. More details are given in [52]. Our proof-of-concept implementation leverages preexisting capabilities in our research CASE tool AUTOFOCUS3 (AF3) [30] (such as logical architecture description, model-based requirements engineering, model-checking, optimized deployments, safety case expression, among many others) [50], [51] to generate the artifacts for the safety mechanism libraries.

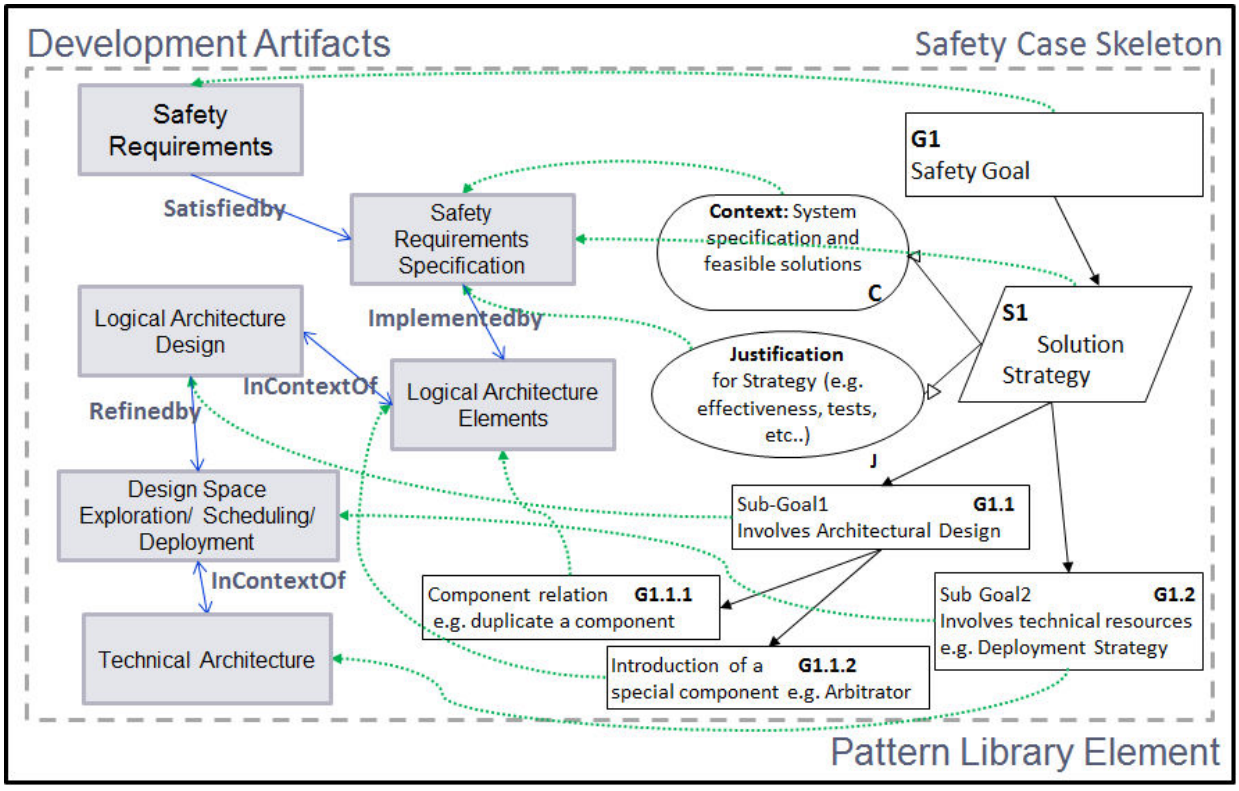

Figure 25: Pattern Library Approach: Overview of artifact relations [52]

In this section, we mainly focus on the structure and usage of elements from a library of reusable patterns, including the data models and workflow required to support seamless integration into a model-based development environment. Automation via tool-support is essential to achieve the intended advantages.

9.3 Structure Model of Library Element

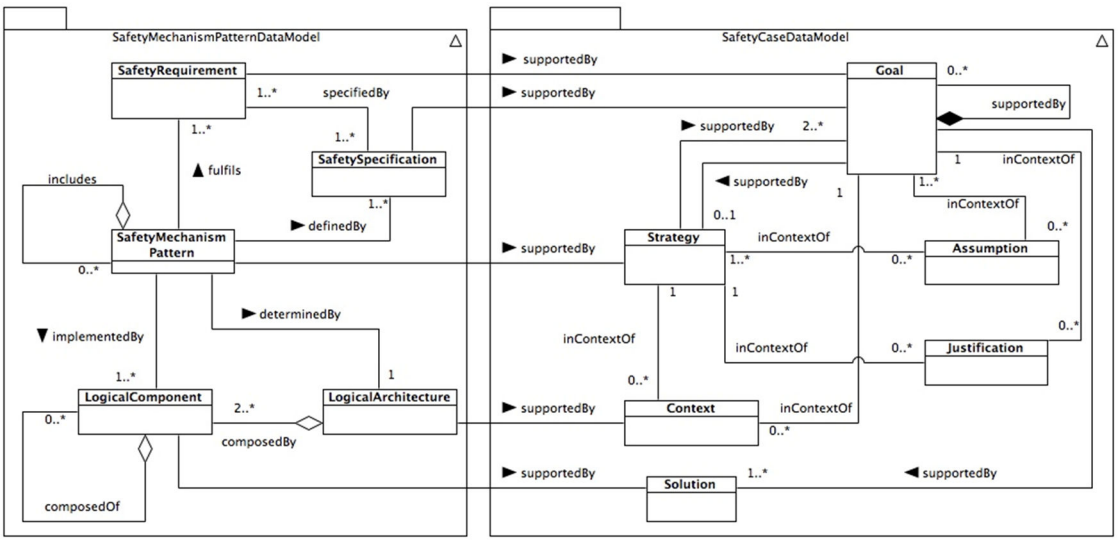

Figure 26: Structure Model for Safety Mechanism Library Element [58]

Figure 26 shows the basic structure model of a pattern library element. The right hand side is a safety case structure linked to the corresponding development artifacts on the left hand side, which in turn support each argumentation element. We use the Goal Structuring Notation (GSN) - in its Community Standard Version 1.0 [12] – for the graphical expression of safety cases as it was the most mature option suited to our needs. GSN is a structured (yet not formal) notation, which provides graphically differentiated basic safety case element types, such as goals, evidence, strategy, justification and so on, as well as a clear description of necessary and allowed connection types between these elements. Our decision to employ GSN is detailed in section 7.

The library elements were chosen to cover the most basic building blocks of any systems-engineering approach, such as [53], as well as the minimum requirements for information expressed in safety standards, such as [1]. Extending the model to capture more information, as well as testing the existing model for soundness using numerous varied mechanism examples, was started in this document and is the focus of our ongoing work.

While the left part of the model, which contains development artifacts, can entirely originate in one seamless tool (as will be shown in our implementation example in Section 9.4.1), this is not necessary. The development artifacts may reside in multiple repositories and be in varying formats; the binding element is the safety case shown on the right hand side, which gives the story and rationale behind the reusable pattern, organized by its relation to the *StructureModel* entities. This aspect is particularly important for a real world development context, which almost always entails a heterogeneous tool-chain.

9.3.1 Structure Model Elements

Figure 27 provides a closer look at the left hand side of Figure 26, detailing the development artifacts library structure part. The definition and scope of these fundamental building blocks are in line with the system engineering view provided in the SPES2020 Project [53].

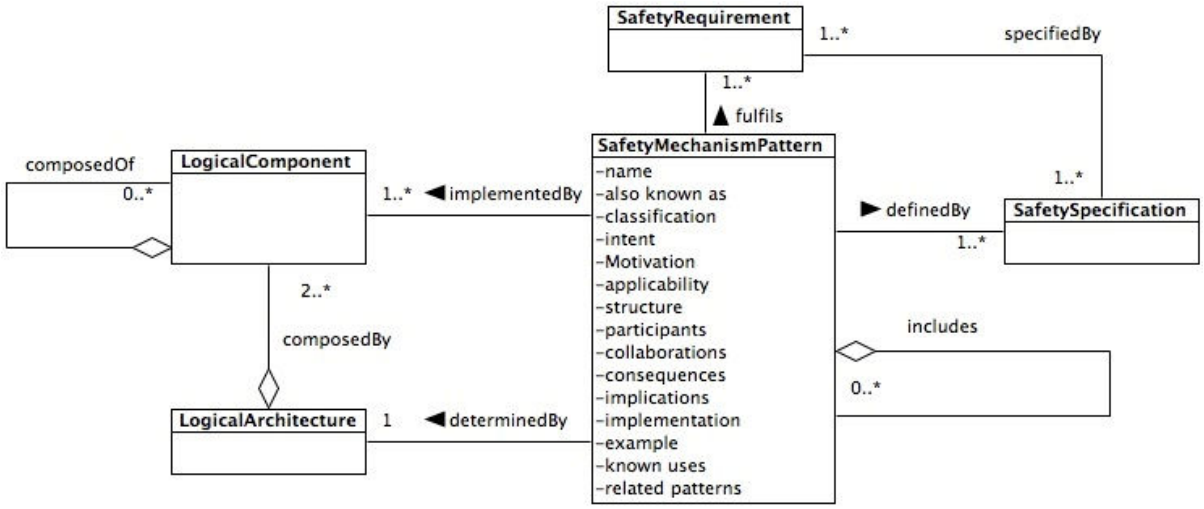

Figure 27: Pattern Library Element: Structure Model for Development Artifacts [58]

The entities of the safety mechanism pattern data model seen in Figure 27 are:

- **SafetyMechanismPattern**: is a container object collecting information necessary for reuse, which is listed as a catalogue of attributes and adapted to the context of safety mechanism patterns. The majority of these attributes originated in the works of the Gang of Four [17] and were further consolidated for use in the safety field in Tim Kelly's works [5], [43]. Two clear exceptions are the attribute ***Implications***, taken from [32], which describes how using a safety mechanism pattern affects other non-functional requirements such as reliability, modifiability, cost & execution times, and the attribute ***Related Patterns***, which typically contains a purely nominal list of similar or affiliated patterns, but in our case is used to link related safety mechanism *classes*. This provides direct traceability and allows for a quick propagation and adaptation of changes across related patterns, increasing confidence in reuse as well as the maintainability of the library.
- **SafetyRequirement**: details the requirement the pattern fulfils, as well as any safety related attributes, e.g., Safety Integrity Level.
- **SafetySpecification**: a concrete description of the safety mechanism's functional behavior and its interaction with the environment, and serves as a basis for the implementation in a logical component. It can be formal, e.g., in AF3 the behavior of safety mechanisms can be captured in state automata.
- **LogicalComponent**: is the implementation or concrete representation of a safety mechanism at software level, whose definition includes transfer behavior, e.g., code specification or a state automaton. It can be a composition of logical components which collectively determine the safety mechanism's behavior.
- **LogicalArchitecture**: defines the context of the safety mechanism pattern, by describing the boundaries / interface of the *LogicalComponent* implementing the mechanism as well as giving a description of any other components involved.

9.3.2 Structure Model Relations

The relationships between the model's entities are as follows:

- ***fulfils***: The use of the pattern fulfils the stated safety requirement(s).
- ***specifiedBy***: this relation shows that a safety requirement is *specified* or brought to a more concrete representation by one or more specifications of the implementation, providing the "how" to the "what". This relation is also bidirectional, because it could be that a solution specification *specifies* one or more safety requirements.
- ***definedBy***: a safety mechanism's behavior is defined by its specification. Can also give a description of the architectural context of the safety mechanism.
- ***determinedBy:*** the safety mechanism pattern is determined by a unique structure or logical architecture.

- ***implementedBy***: the safety mechanism belonging to the pattern is implemented by a logical component at software level.
- ***composedOf***: a logical component *can be* composed of other logical components or simply direct descriptions, e.g., a code specification or a state automaton.
- ***composedBy***: this relationship shows that a logical architecture or context of a safety mechanism is *composed by* at least two logical components, which interact with the safety mechanism's logical component. In its simplest form, one is an input and the other is an output component.
- ***includes***: this relationship indicates that a safety mechanism pattern can include other patterns within its structure. This entails inheritance of the respective safety requirements, safety specification, logical component and the logical architecture of the related or reused safety mechanism patterns. The form and structure of inclusion has to be defined in a case-by-case manner.

9.3.3 Pattern Catalogue Attributes

Contrary to the typically purely verbose nature of capturing pattern attributes, our use of a model-based approach means that we are able to integrate this information and capture most of it into the elements of the structure model. This is a particularly important feature, as verbose lists have a tendency to be overlooked, misused or misinterpreted and we consider them to constitute an obstacle to pattern reuse. The captured attributes are:

- ***Name***: captured in the name of the safety mechanism pattern itself.
- ***Classification***: states to which category of safety mechanisms (*Failure Avoidance, Failure Detection and Failure Containment/ Handling)* [43] the pattern belongs, and is captured in the structure of the library itself.
- ***Intent***: this attribute is captured in the *SafetyRequirement* entity.
- ***Motivation***: this attribute provides the rationale behind using the pattern and is captured in the safety requirement entity, with any additional, implementation-specific information being captured in the *SafetySpecification* entity.
- ***Applicability***: describes the architectural context of the safety mechanism as well as instantiation information and is captured by the *LogicalArchitecture* entity and augmented by the workflow described in section 9.4 of this document.
- ***Structure***: this attribute is captured by the entities *LogicalComponent* and *LogicalArchitecture* and their relation.
- ***Participants***: are captured at an interaction level by the *LogicalArchitecture* entity and at a structural level by the structure of the safety mechanism library.
- ***Collaborations***: this attributed is captured by the structure of the safety mechanism library and the safety case supporting it.
- ***Consequences***: the parts of this attribute describing which components in the logical architecture or inside the safety mechanism's component have to be instantiated or further developed upon usage are captured in the structure model as well as in the workflow described in section 9.4.
- ***Implementation***: is partially captured in the *SafetySpecification* entity as well as the entities *LogicalComponent* and *LogicalArchitecture* and their relation.
- ***Related patterns***: this attribute is captured in the data model for inclusion relations by the recursive aggregation relation of the *SafetyMechanismPattern* entity.

9.3.4 Mapping to SafetyCaseModel

Having defined the development artefacts part of the structure model, a quick explanation of their mapping to the corresponding elements in the *SafetyCaseModel* seen on the right hand side of Figure 26 is given next.

- **SafetyRequirement – *supportedby* – Goal**: the entry point to any reuse problem is a goal or requirement that has to be fulfilled.
- **SafetyMechanismPattern – *supportedby* – Strategy**: the information captured in the safety mechanism pattern and the information it contains lays out a plan for solving the problem, i.e. a

strategy, and helps explains the relation between the elements of the pattern, most importantly their refinement and decomposition.
- **SafetySpecification – *supportedby* – Goal**: a safety specification is mapped to the (sub-)goal entity in the safety case data model, as it describes a refinement and decomposition of the higher level goals towards implementing a solution.
- **LogicalComponent – *supportedby* – Solution**: the logical component is the solution for the stated goal.
- **LogicalArchitecture – *supportedby* – Context**: the logical architecture provides context to the safety pattern and the logical component and its implementation, without which the pattern is either meaningless or cannot be described concisely.

9.4 Tool implementation and usage workflow

Our approach and the implementation support in AF3 envision two roles: a safety mechanism *Developer* and a safety mechanism *User*. To support the *Developer* role a comprehensive analysis of many safety mechanisms was carried out, which identified three archetypes of structure models for capturing safety mechanisms in AF3. To save a new safety mechanism, the *Developer* has to decide which archetype best fits his need. The developer is aided in this task by a wizard, which also assures that library elements cannot be created without the mandatory information. These AF3 categories and the *Developer* role are not the focus of this specification.

For the *User* role, we must differentiate between using the library element in an already existing project or in a green field one, i.e. I need a redundant design for my function/application/system and wish to start with a comprehensive structure model that includes all the necessary artefacts and then fill in details later.

The analysis of the safety mechanisms and the usage roles, led to the identification of three categories for the usage of the artefacts captured in the library element. Taking the usage scenario into account, these categories define how each artefact in the library element will be added into the development environment. These are:

- **Copy**: Artefacts are copied into the environment as is. *SafetyRequirements*, and *SafetySpecification* elements, along with all the *SafetyCaseModel* artefacts, are treated in this manner, as they are clearly defined static elements that do need to be changed upon usage.
- **Instance**: artifacts are to be instantiated upon usage. This may include the input of some parameters by the users. *LogicalComponent* entities are always used in this manner. *LogicalArchitecture* elements are treated in this manner if and only if the usage mode is “green-field”.
- **Replace**: Artefacts are replaced by ones existing in the model. This mode applies to *LogicalArchitecture* elements and is obviously only usable when deploying the library element into a preexisting model. E.g., logical components stored in the library element to describe the interaction of the central safety mechanism component with its environment are replaced by existing components from the model, to which the library pattern / safety mechanism is to be applied.

The deployment of the library elements is guided by a wizard, whose function it is, among others, to ensure that library elements are deployed seamlessly. To that end, the wizard prompts the user at each step to link / map the components of the inserted library to their counterparts in the existing model. E.g., a comparator deployment prompts the user to select the two components whose outputs are to be compared, as well as the original requirement, which prompted the deployment of the comparator safety mechanism from the library in the first place. The development process still has to be monitored and approved at key decision points by a human expert.

The implementation carried out in AF3 currently covers four safety mechanisms, *1oo2Voting*, *Comparison*, *RangeCheck*, and *HomogeneousDuplexRedundancy* [43]. Continuing with the redundancy example introduced in section 9.1.4, the following section illustrates the implementation using screenshots.

9.4.1 Implementation example in AF3

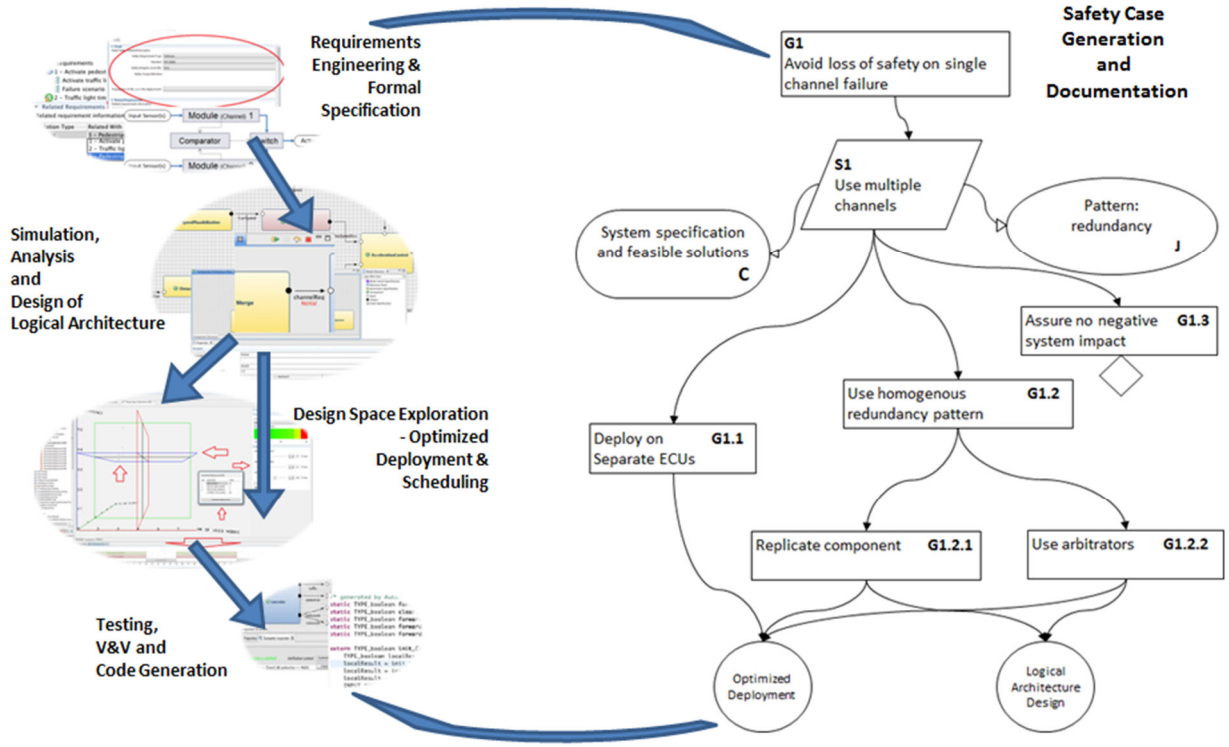

Figure 28: Proof-of-concept implementation for Duplex Redundancy Pattern [52]

Figure 28 shows an illustrative overview of the pattern library implementation. On the left are snapshots of screen grabs of functionalities for the generation of library artifacts in AF3 and on the right is an example for a simplified corresponding safety case, thus encapsulating all the information required to reuse the pattern.

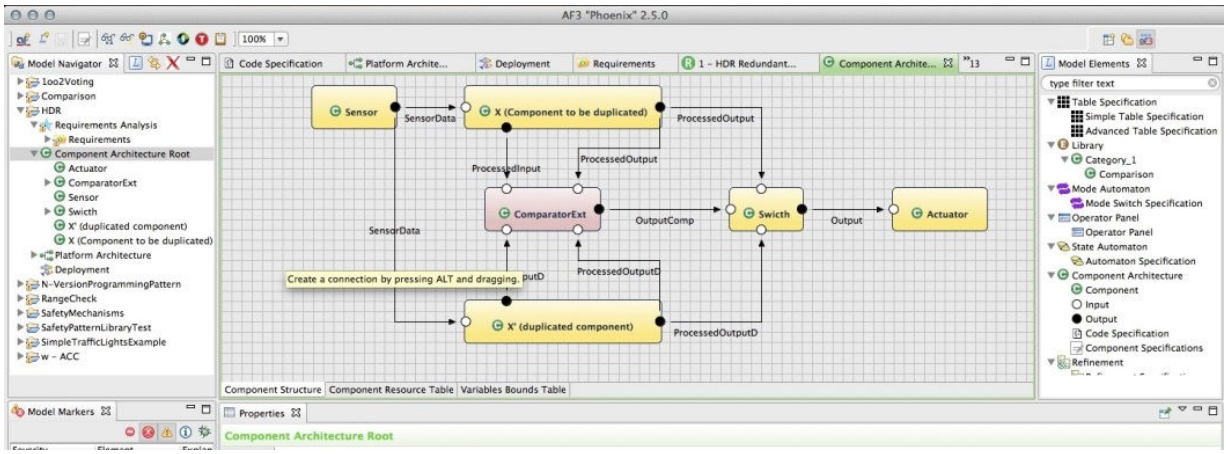

Figure 29: Implementation Example HDR: LogicalArchitecture [58]

The interaction of the *HomogeneousDuplexRedundancy (HDR)* implementation example safety mechanism with its environment is captured in the *LogicalArchitecture* element shown in Figure 29, with the corresponding safety case shown collapsed in Figure 30. Using the relations described in the GSN standard [12] and implemented in AF3, it is possible to perform completeness and consistency checks on the safety case, such as "*solutions must stem from goals and no other elements*" or "*no open goals remain*". Not shown but implemented with AF3, are the corresponding requirements or specification of any of the central *LogicalComponent* elements, e.g., the *Comparator*. Also

possible but not shown, is a formal specification of mechanism behavior, allowing automatic test suite generation, as discussed in WP4.3 [57].

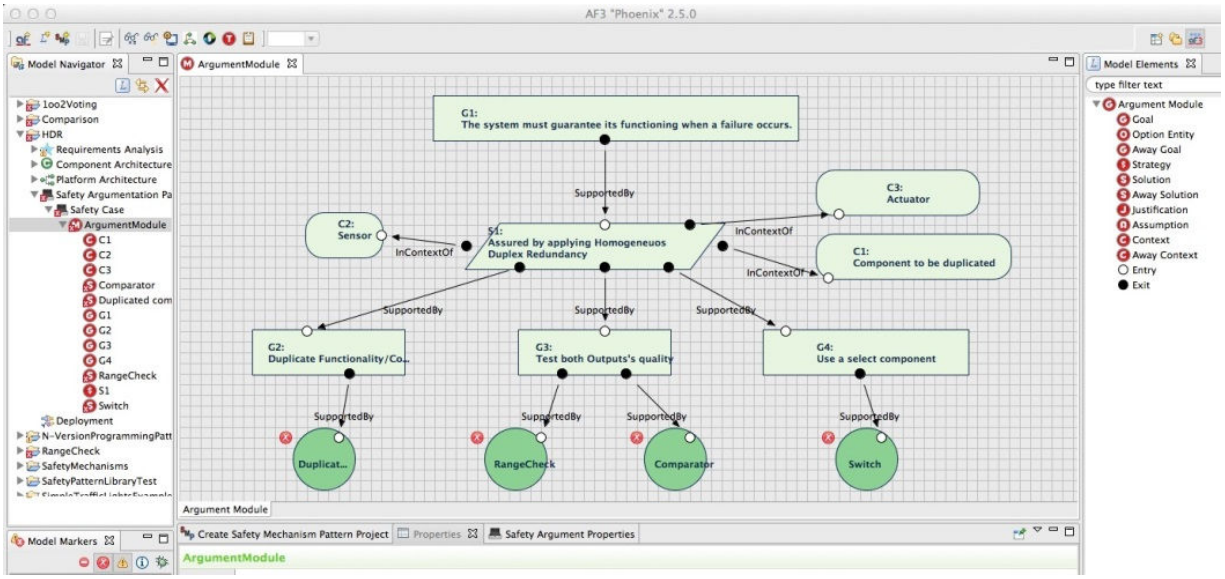

Figure 30: Implementation Example HDR: SafetyCaseModel [58]

9.4.2 Extended Structure Model

The library element is, however, incomplete for the HDR example without capturing the deployment rules, which determine how the software components are assigned to hardware units. This is a new piece of information and gives rise to the need to extend the structure model with optional elements beyond the basic ones previously shown in Figure 26. To encapsulate more information, such as the deployment rule, or facilitate a refinement of the specification, such as the categories provided by ISO26262 in its functional and technical safety concepts, we extended the structure model as shown in Figure 31. Other elements could also be introduced to capture any other information pertaining to the suitability of the safety mechanism, e.g., tests or analyses.

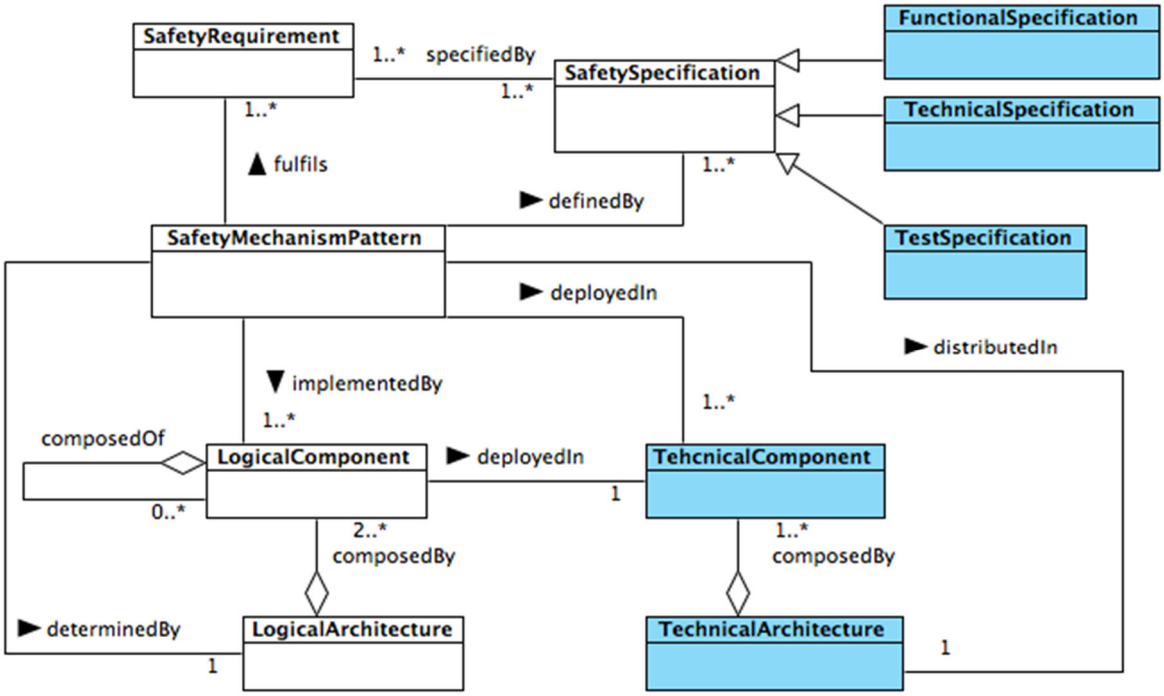

Figure 31: Extended Structure Model for Development Artifacts [58]

9.4.3 Modular and compositional argumentation

The use of patterns as a way of documenting and reusing successful safety argument structures was pioneered by Kelly [5], who developed an example safety case pattern catalogue providing a number of generic solutions identified from existing safety cases. This was further developed by Weaver [6], who specifically developed a safety pattern catalogue for software. A software safety pattern catalogue was also developed by Ye [7], specifically to consider arguments about the safety of systems including COTS software products. Further discussion of these approaches' strengths and drawbacks is given in [8], [52].

These approaches all focused on generating safety case templates for known patterns, whereas the safety case templates themselves are the focus. Our approach encourages the encapsulation of the reusable development artifacts along with the supporting safety cases to enable correct and speedy reuse in a safety-critical context. The ARRL approach presented in the OPENCOSS project [55] offers a similar but more generalized workflow, yet our approach differs in that it catalogues existing information, targets a specific use case, and does not force a new framework or language on to the user.

Using our approach does give rise to at least three interesting questions:

- What is an adequate boundary interface for modular compositional argumentation?
- How do we guarantee that all goals (and sub-goals) have been identified?
  and more importantly,
- How do we guarantee that the introduced pattern or component has no negative impact on the system?

As previously discussed in [52] and supported by our implementation results, the safety goal presents a very adequate boundary interface to the system, whose problems the pattern solves. This is alignment with common practice in engineering projects, where "requirements catalogues" are the standard mode of exchanging information about needs and evaluating results. To illustrate this, imagine that the comparator we choose to employ in the example shown in Figure 28 is a COTS HW/SW unit we buy from a trusted supplier who also provides the corresponding detailed safety case for the arbitrator. That (sub-) safety case snaps on to the collapsed safety case, shown in Figure 30, at the safety sub-goal (Comparator) it satisfies. The entire safety case in Figure 30 would then itself snap into the parent (system) safety case at the top-level goal boundary defining its purpose, and this process would repeat itself hierarchically and cumulatively. As such, satisfying all the system's goals at the interface would enable not only modular but also a cumulative argumentation.

The answer to questions 2 and 3 is partially given in the usage description in section 9.4. We do not think that a human expert can be totally taken out of the loop, especially if a deterministic behavioral description is not included into the library, as in our case. Thus key approval is delegated to the expert at all integration nodes and supported by automated tool checks, such as the one shown in Figure 28, where the undeveloped safety goal *G1.3*, which leads to no solution and will as such be detected by the tool checks, serves as a reminder to carry out an impact analysis after integrating the safety mechanism (and its safety case) into the existing model. Similarly, we propose that assumptions made in library safety cases require the approval of the system integrator before acceptance. This check is easily implemented.

To achieve true compositionality, the underlying framework used to describe the behavior of each safety mechanism should support temporal logic, and the descriptions of both the components and the safety cases supporting them would have to be deterministic. Such descriptions lie outside the scope of this specification, which focuses on structural as opposed to behavioral models, and could be the subject of future work.

9.5 Mapping onto SAFE Meta-Model

As was seen in WT3.6 [60], safety mechanisms or *tactics*, play an important part safety-critical development and have to be correctly and precisely specified. Mapping the approach presented in the previous section and tailoring it onto the SAFE Meta-model, a detail rich and context specific environment, presents a high advantage potential for information encapsulation for this class of solutions.

9.5.1 Safety Tactics in SAFE

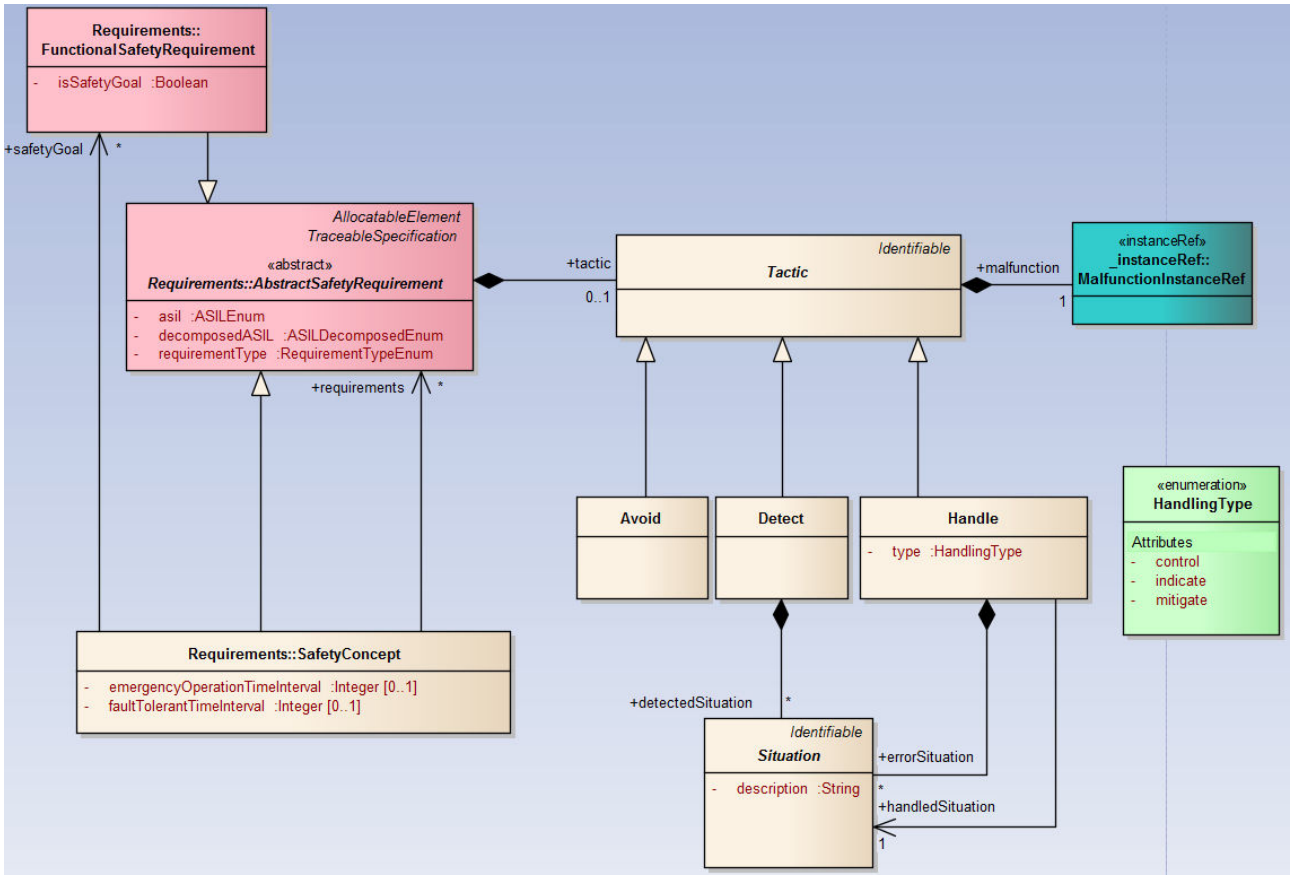

Figure 32: Tactics classification in SAFE Meta-model [60]

Figure 32 gives a high level class description of the derivation and traceability as well as classification of safety tactics, according to the type of error management they offer (avoid-detect-handle) [43]. The *Tactic* provides a container for the classification, through which a dysfunctional view encompassing malfunction modes for each tactic are captured, as well as the connection to malfunction instances which are meant to be covered by the tactic.

Traceability is guaranteed by the SAFE Meta-model satisfy mechanism linking hazard description and subsequent analysis to the corresponding top level safety goal and onto the concept to which the tactic and the requirements it fulfills belong.

On the other hand, the requirements are then refined through the multiple abstraction layers supported in the SAFE Meta-model (through a mapping analogy from EAST-ADL to ISO26262, for more details see the SAFE Meta-model description D3.5d [48]) to the specification of Software Safety Requirements (SSR), which holds the tactic fine description and carries the same name. This then links to implementation descriptions, configurations and actual instances. As specified in [60], and demonstrated in [61], it is possible to automatically generate code for the tactics, based on this structured description. The lower half of Figure 33 shows the tactic descriptor and the implementation artifacts for both an AUTOSAR conforming generator as well as CHROMOSOME, our research middleware [62], for a Voter.

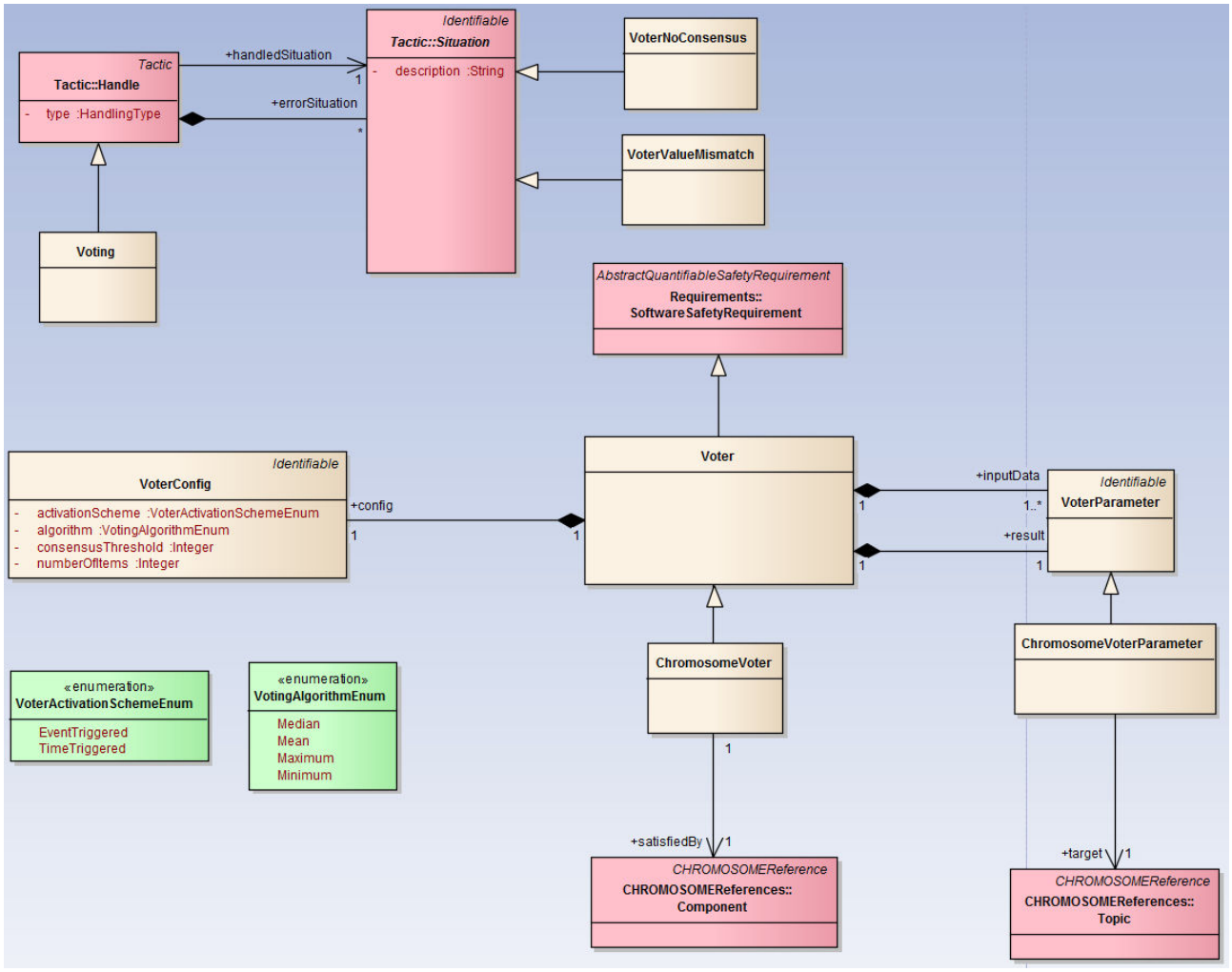

Figure 33: Voter Tactic in SAFE Meta-model [60]

The instantiation of the tactic classification mechanism from Figure 32, is seen in the upper half of Figure 33, as well as in the right hand side of Figure 34, which presents a simpler tactic, HealthMonitor, with CHROMOSOME implementation only, as shown on the left hand-side of the figure.

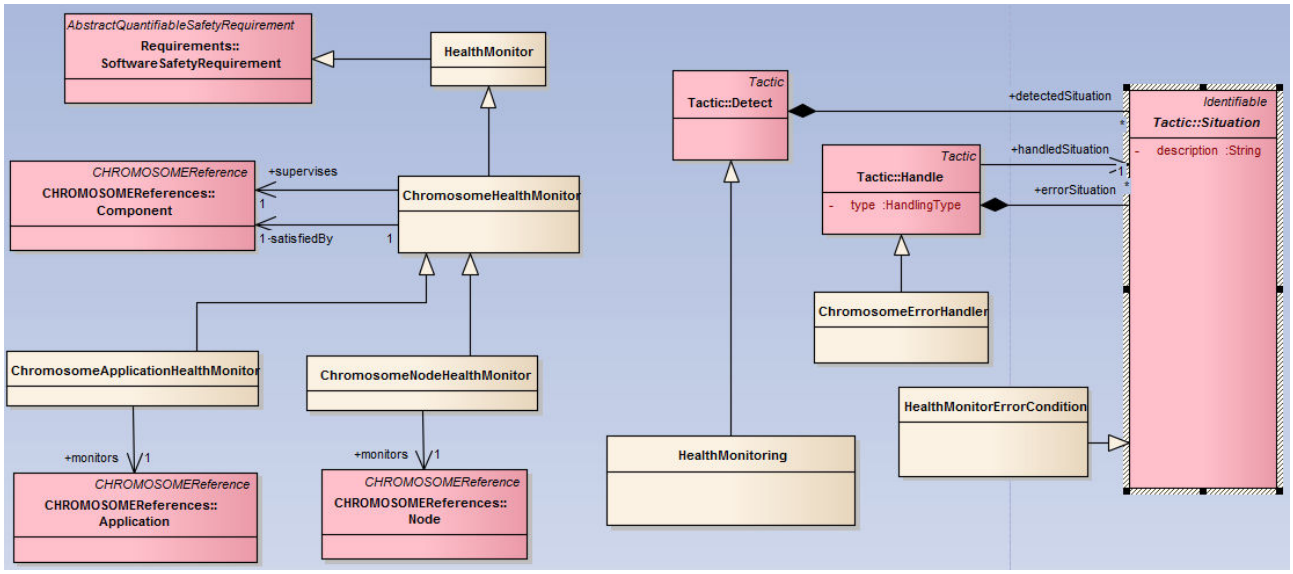

Figure 34: HealthMonitor Tactic in SAFE Meta-model [60]

The previous figure can be rearranged and, the link between the classification aspects and the requirements specification and implementation description be made explicit, through the inclusion of the SAFE Meta-Model requirements structure representation. It is noteworthy that no direct link exists between the classification branch and the implementation specification branch, which are connected only at the apex through specialization of the requirements artifacts. To prevent model ambiguity through loss of information and assure conformity between the two branches, a further categorization of the functionality of the tactic was added to the tactic classification mechanism. This is then matched to the tactic description SSR via OCL.

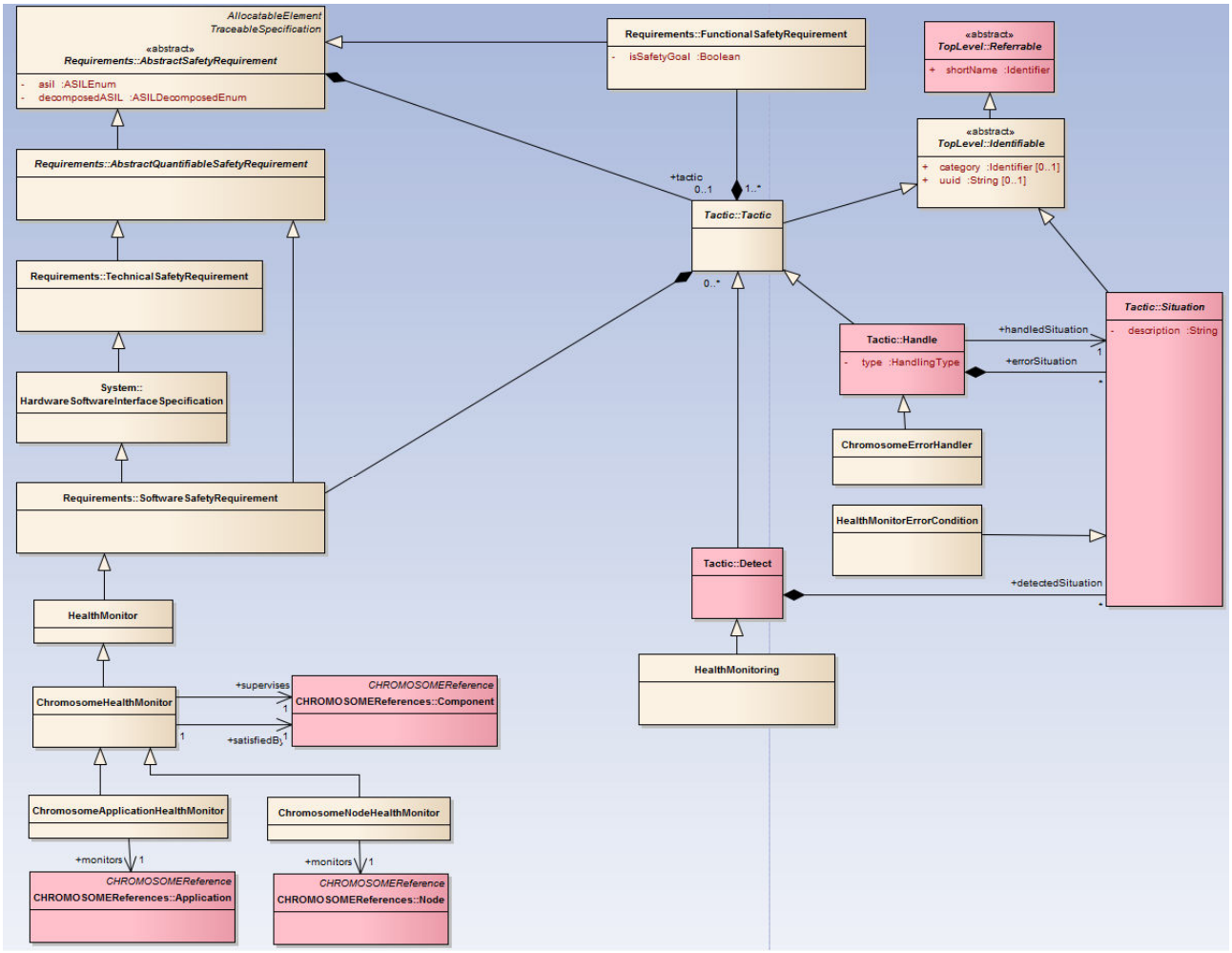

Figure 35: Alternate View on HealthMonitor Tactic in SAFE Meta-model

9.5.2 Structured pattern attribute encapsulation in SAFE Meta-model elements

If we regroup the artifacts from Figure 35 into abstract categories in analogy to the approach presented in section 9.2, we get the following main high level categories.

- Requirements
- Tactic Description
- Tactic Classification
- Tactic Implementation

If these categories are then supported by safety case arguments (supported in the SAFE Meta-model as seen in section) in a structured approach as shown in Figure 36, it would be possible to capture most pattern attributes within the model elements, as discussed in section 9.3.3.

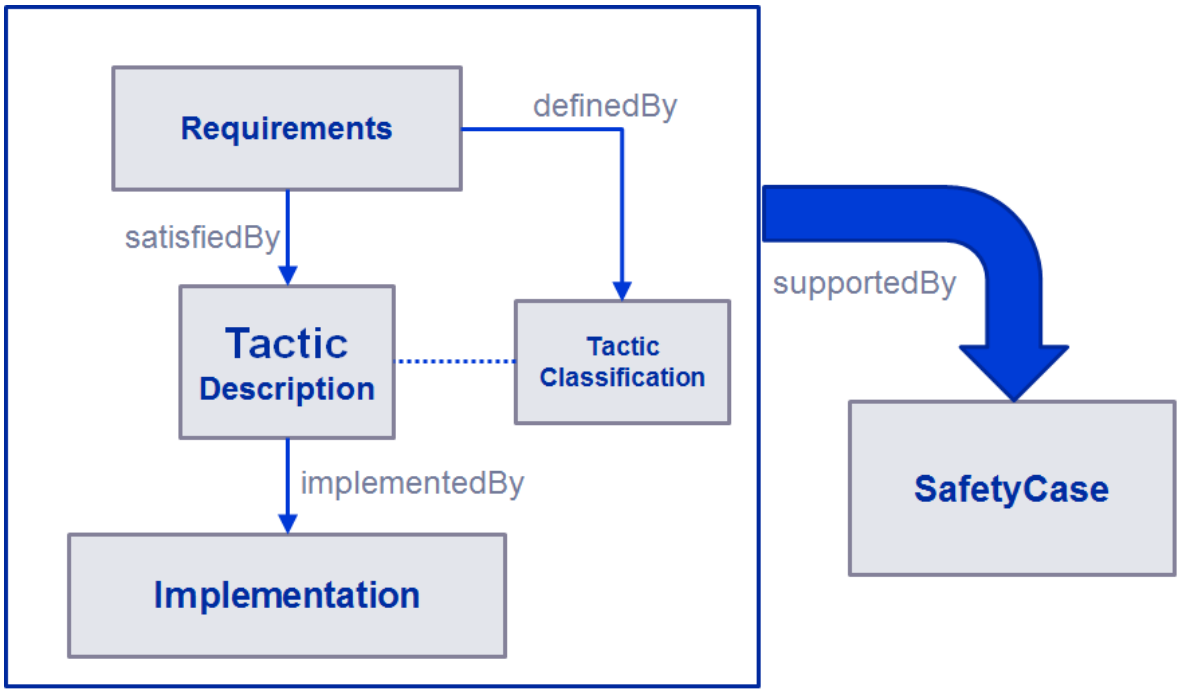

Figure 36: Abstract artefact structure for tactic library in SAFE Meta-model

Contrary to the typically purely verbose nature of capturing pattern attributes, our use of a model-based approach means that we are able to integrate this information and capture most of it into the elements of the structure model. This is a particularly important feature, as verbose lists have a tendency to be overlooked, misused or misinterpreted and we consider them to constitute an obstacle to pattern reuse. The captured attributes are:

- ***Name***: captured within the *Tactic Description* category.
- ***Classification***: captured in great detail in the *Tactic Classification*.
- ***Intent***: captured in the *Requirements* category.
- ***Motivation***: captured in the *Requirements* category and supported by the S*afetyCase* argumentation.
- ***Applicability***: captured in the *Tactic Classification* category, as well as in the implementation layers of the *Requirements* category and the *Implementation* category.
- ***Structure***: captured in the *Implementation* category.
- ***Participants***: captured at an interaction level by the *Implementation* category and at a structural level by the structure of the tactic library.
- ***Collaborations***: this attributed is captured by the structure of the tactic library and the *SafetyCase* supporting it.
- ***Consequences***: captured in the tactic library structure as well as by the *Implementation* category.
- ***Implementation***: captured by the *Implementation* category.
- ***Related patterns***: captured in the *Tactic Classification* category.

9.6 Conclusion

The design of functional safety systems is largely driven by best practices – like the use of fault monitors, safe states, or redundant paths. These best practices can often be presented in the form of patterns – both to describe a possible solution, but also to document an argumentation about their contribution to a safety case. The provision of a library of such patterns allows the identification and comprehensive (re-)use of suitable architectural measures and of corresponding components along with their safety cases, facilitating a cumulative if not compositional argumentation about the achieved safety goals.

The approach defines the structure of such a library, encapsulating all necessary information for a correct reuse of safety mechanisms. The approach not only adds new attributes to the ones known in pattern catalogue literature but also captures most of the information from these attributes into the model itself. This decreases reliance on the hitherto typically verbose attribute catalogue and we consider it to be a major bonus for the approach, as textual lists have a tendency to be overlooked, misused or misinterpreted and we consider them to constitute an obstacle to pattern reuse.

Furthermore, the approach features an explicit model-based representation of safety mechanisms within the context of their usage and the problems they solve. This has several advantages, including 1) a better characterization of the problem space addressed by the pattern – better than the textual description otherwise used in pattern templates, 2) a more natural representation of the transformations embodied in the application of the pattern, and 3) a better handle on the selection, rationale and application of the patterns [54].

Thus, the approach contributes to the optimization of development with respect to system safety in general, and specifically to safety-critical component reuse. We also believe the paradigm to be extensible to COTS (Commercial Of-The Shelf) components, as well as providing support for the "Safety Element out of Context" clause cited in safety standards, such as the ISO26262 [1].

This section presents the structure and workflow of an approach which facilitates the reuse of safety mechanisms by encapsulating relevant information in a pattern library with tool support. We furthermore explored the possibilities and showed the opportunities supported by this approach. A holistic pattern-based approach to the construction of safety-cases in a seamless model-based development of safety-critical systems requires several elements, the main constituents of which are:

- A library of reusable argumentation patterns – both in form of problem patterns (e.g., faults like early, late, or wrong sensor information; temporal or spatial interference between functions) and solution patterns (e.g., error avoidance, detection, mitigation; sensor fault detection and correction mechanisms; partitioning mechanisms) – built from elements of a model-based description of the system under development (e.g., requirements, functions, SW-/HW-components, channels, busses, deployment strategies) as well as GSN safety cases (e.g., goals, solutions, justifications, contexts)
- A mechanism for the instantiation (e.g., stuck at-/noise-like faults; different filter for fault detection) and application (e.g., linking to the corresponding HW- and SW-elements) of those patterns in a compositional fashion.
- A mechanism to check the soundness of the constructed argumentation (e.g., no open sub-goals; all context are covered by corresponding system elements) w.r.t. to its structure.

Using the approach presented here and a proof-of-concept implementation in our research CASE tool AF3, we have shown the feasibility of handling all three of the above points to varying degrees. Most importantly, the quality of the captured attributes increases greatly with the domain adequacy and content richness of the modeling framework used, as seen in section 9.5.

10 Conclusions and Discussion

This document provides information about the safety case concept as adopted by many safety critical industries describes the proposed methodology for safety case modeling and documentation as well as for an extension of the collective SAFE/SAFE-E Meta-model for hazard and environment modeling.

Besides giving an overview on the relevant parts of ISO 26262 the requirements arising from WT 2.1 (ISO 26262 Analysis), a focal part of this deliverable lies in the presentation of the methodology for safety case modeling and documentation. This methodology is compliant to the requirements given in ISO 26262 and in addition comprises aspects arising from experiences in the development of automotive systems as well as other safety critical industries (such as defense, railways and aerospace). However, the ability to describe and link development artifacts in a safety case relevant context is only half the story. How this capability is employed is the other half of this document. The concepts of solution and design patterns are introduced, along with the concept of compositional argumentation. These concepts are far from fully matured and therefore, the initial concepts presented in this document can be seen as a basis for further development.

The initial contribution to the SAFE Meta-model presented in this deliverable provides the possibility to link safety-critical artifacts together to form an ISO 26262 compliant safety case. At the same time the requirements coming from the methodology are considered. In case the methodology is extended there might also arise the need to adapt the corresponding part of the SAFE Meta-model, including those developed in other work tasks.

Since it is an objective to reuse EAST-ADL as much as possible the status of the current version of EAST-ADL is presented and initial proposals for extensions are formulated. However, these proposals need to be further elaborated in future. For the proposed extension of the SAFE Meta-model EAST-ADL references are used whenever possible.

The specification described in this work is further supported by proof-of-concept implementations. Most promising among these is the support for pattern libraries and the capability of modular and compositional argumentation introduced in section 9. This work could be expanded to include new concepts as well, such as the concept of assured safety arguments, a new structure proposed by Hawkins and Kelly in [4] for introduces a confidence argument that documents the confidence in the structure and evidence of the safety argument.

11 List of Abbreviations

AF3	AUTOFOCUS 3 – A research CASE tool developed by fortiss GmbH
ASIL	Automotive Safety Integrity Level
ATESST	Advancing Traffic Efficiency and Safety through Software Technology
COTS	Commercial Off The Shelf
EPS	Electric Power Steering
GSN	Goal Structuring Notation
EAST-ADL	Electronic Architecture and Software Tools- Architecture Description Language
HA	Hazard Analysis
RA	Risk Analysis
FMEA	Failure Mode and Effect Analysis
FTA	Fault Tree Analysis
HW	Hardware
OCL	Object Constraint Language
SAFE	Safe Automotive soFtware architecture – the affiliated ITEA2 Project
SSR	Software Safety Requirement
SW	Software
WT	Work Task

12 Acknowledgments

This document is based on the SAFE and SAFE-E projects. SAFE is in the framework of the ITEA2, EUREKA cluster program Σ! 3674. The work has been funded by the German Ministry for Education and Research (BMBF) under the funding ID 01IS11019, and by the French Ministry of Economy and Finance (DGCIS). SAFE-E is part of the Eurostars program, which is powered by EUREKA and the European Community. The work has been funded by the German Ministry of Education and Research (BMBF) and the Austrian research association (FFG) under the funding ID E!6095. The responsibility for the content rests with the authors.

The AF3 project is a long term ongoing tools development effort led by the fortiss department of “Software and Systems Engineering". The functionality provided currently by the tool is mainly the result of the joint work of the following persons:

- Florian Hölzl (mailto:hoelzl@fortiss.org): long-term project lead, tooling infrastructure for developing views and modular languages
- Dr. Vincent Aravantinos (mailto:aravantinos@fortiss.org): current AF3 coordinator, formal methods for model checking
- Sabine Teufl (mailto:teufl@fortiss.org): MIRA - model-based requirements engineering
- Mou Dongyue (mailto:mou@fortiss.org): MIRA - model-based requirements engineering (partly under the IMES project), model based testing, refinement modelling
- Dr. Sebastian Voss (mailto:voss@fortiss.org): design space exploration, GSN-based modelling of safety cases, ASIL allocation, and optimized (safety-oriented) deployment and schedule generation under the SPES XT, RECOMP and ARAMiS projects
- Dr. Daniel Ratiu (mailto:ratiu@fortiss.org): user friendly formal specification and verification

Many thanks also to the other members of the AF3 team.

13 References

[1] International Organization for Standardization: ISO 26262 “Road vehicles - Functional safety”. www.iso.org. 2011.

[2] S. Toulmin: “Uses of Argumentation”. Cambridge University Press, 1958; 2nd edn., 2003

[3] The ATESST Consortium: “ATESST Project Deliverable D2.2.2”. 2007.

[4] R. Hawkins, T. Kelly, J. Knight and P. Graydon: “A New Approach to Creating Clear Safety Arguments”. Advances in Systems Safety - Proceedings of the Nineteenth Safety-Critical Systems Symposium, Southampton, UK, February 8-10, 2011. Springer. 2011.

[5] T. Kelly. “Arguing Safety – A Systematic Approach to Managing Safety Cases”. PhD Thesis. Department of Computer Science, The University of York. 1998.

[6] R. Weaver. “The Safety of Software – Constructing and Assuring Arguments”. PhD Thesis. Department of Computer Science, The University of York. 2003.

[7] F. Ye. “Justifying the Use of COTS Components within Safety Critical Applications”. PhD Thesis. Department of Computer Science, The University of York. 2005.

[8] R. Hawkins, K. Clegg, R. Alexander, T. Kelly. “Using a Software Safety Argument Pattern Catalogue: Two Case Studies”. In Proceedings of SAFECOMP 2011, LNCS 6894, pp. 185 – 198, F. Flammini, S. Bologna, and V. Vittorini (Eds.). Springer, Heidelberg. 2011.

[9] M. Jaffe, R. Busser, D. Daniels, H. Delseny, G. Romanski. „Progress Report on Some Proposed Upgrades to the Conceptual Underpinnings of DO178B/ED-12B”. In Proceedings of the 3rd IET International Conference on System Safety. 2008.

[10] T. Kelly. “Concepts and principles of compositional safety case construction.” Technical Report COMSA/2001/1/1. The University of York. 2001.

[11] R. Hawkins and T. Kelly. “A Software Safety Argument Pattern Catalogue”. Department of Computer Science, The University of York. 2008.

[12] Origin Consulting (York) Limited, on behalf of the Contributors. “Goal Structuring Notation (GSN)”. GSN COMMUNITY STANDARD VERSION 1. November 2011.

[13] T. Kelly and R. Weaver: “The Goal Structuring Notation . A Safety Argument Notation”. Proc. DSN 2004 Workshop on Assurance Cases, 2004

[14] Kelly, Tim P., and John A. McDermid. "Safety case construction and reuse using patterns." 16th International Conference on Computer Safety, Reliability and Security (SAFECOMP 1997). 1997.

[15] C. Alexander. “A Pattern Language: Towns, Buildings, Construction”. Oxford University Press, New York. 1977.

[16] M. Khalil. “Pattern-based methods for model-based safety-critical software architecture design”. ZeMoSS 2013 Workshop at the SE 2013 conference in Aachen. February 2013.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. “Design Patterns: Elements of Reusable Object-Oriented Software.” Addison-Wesley, Boston, MA, USA. 1997.

[18] K. Beck and W. Cunningham. “Using pattern languages for object-oriented programs.” In Position Paper for the Specification and Design for Object-Oriented Programming Workshop, The 3rd Conference on Object- Oriented Programming Systems, Languages and Applications, Orlando, USA. 1987.

[19] J. O. Coplien. “Advanced C++ programming styles and idioms”. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 1992.

[20] J. O. Coplien. “Idioms and patterns as architectural literature”. IEEE Softw., 14(1):36–42. 1997.

[21] D. Manolescu, M. Voelter, and J. Noble. “Pattern Languages of Program Design 5.” Addison-Wesley Professional. 2005.

[22] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. “Pattern-oriented software architecture: a system of patterns.” John Wiley & Sons, Inc. New York. 1996.

[23] B. P. Douglass. “Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems.” Addison-Wesley, New York, Boston, MA, USA. 2002.

[24] R. Hanmer. “Patterns for Fault Tolerant Software.” Wiley Publishing. 2007.

[25] L. Rising. “Design patterns in communications software.” Cambridge University Press, New York, NY, USA. 2001.

[26] M. Pont. “Patterns for Time-Triggered Embedded Systems: Building reliable applications with the 8051 family of microcontrollers.” ACM Press, New York. 2001.

[27] M. J. Pont and M. P. Banner. “Designing embedded systems using patterns: a case study.” Journal of Systems and Software, 71(3):201–213. 2004.

[28] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann,and P. Sommerlad. „Security Patterns: Integrating Security and Systems Engineering.” John Wiley & Sons. March 2006.

[29] N. Yoshioka, H. Washizaki, and K. Maruyama. “A survey on security patterns.” Progress in Informatics:, 5:35–47. 2009.

[30] AutoFOCUS 3 (AF3), research CASE tool, af3.fortiss.org, fortiss GmbH. 2014.

[31] European Organisation for Civil Aviation Equipment “EUROCAE ED-79/ARP-4754: Certification considerations for highly integrated aircraft”. Safety Standard. 2011.

[32] A. Armoush. “Design Patterns for Safety-Critical Embedded Systems”. Ph.D. Thesis, RWTH-Aachen. 2010.

[33] J. Börcsök. „Funktionale Sicherheit: Grundzüge sicherheitstechnischer Systeme“. VDE. 2011.

[34] European Organization for Civil Aviation Equipment (EUROCAE) and United States Department of Defense (DOD). “EUROCAE ED-12B/DO-178B: Software considerations in airborne systems and equipment certification”. Aviation Safety Standard. 1992.

[35] United States Department of Defense (DOD). “DO-178C: Software considerations in airborne systems and equipment certification”. 2012.

[36] European Committee for ElectroTechnical Standardization. “EN-50126 Railway Standard”. www.cenelec.eu. 2008.

[37] International Electrotechnical Commission (IEC). “IEC 61508. Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems (E/E/PE, or E/E/PES)”. Safety Standard. 2nd Edition. 2010.

[38] Y. Papadopoulos, J.A. McDermid. “The Potential for a Generic Approach to Certification of Safety-Critical Systems in the Transportation Sector”. Reliability Engineering and System Safety, 63(1):47-66, Elsevier Science,1999.

[39] Nancy Leveson. “Engineering a safer world”. MIT Press. 2011.

[40] M. Martinus. „Funktionale Sicherheit von mechatronischen Systemen bei mobilen Arbeitsmaschinen“. Technische Universität München. 2004.

[41] S. Wagner, B. Schätz, S. Puchner, P. Kock. “A Case Study on Safety Cases in the Automotive Domain: Modules, Patterns, and Models”. In Proc. of International Symposium on Software Reliability Engineering (ISSRE '10), IEEE Computer Society. 2010.

[42] Tim Kelly. “A Systematic Approach to Safety Case Management.” SAE 2004 World Congress & Exhibition. Safety-Critical Systems Session. Detroit, MI, USA. March 2004.

[43] W. Wu and T. Kelly. “Safety Tactics for Software Architecture Design”. In *Proceedings of the 28th Annual International Computer Software and Applications Conference - Volume 01* (COMPSAC '04), Vol. 1. IEEE Computer Society, Washington, DC, USA, 368-375. 2004.

[44] The EAST-ADL Consortium. EAST-ADL V2.1. http://www.east-adl.info/. 2012.

[45] The SAFE Consortium. Deliverable D3.1.1a “Final proposal for extension of Meta-model for hazard and environment modeling”. ITEA2. 2013.

[46] The SAFE Consortium. Deliverable D3.5a “Initial proposal for Meta-model definition”. ITEA2. 2012

[47] D.R. Lindstrom. “Five Ways to Destroy a Development Project”. IEEE Software, pp. 55-58. Sept. 1993.

[48] The ITEA2 SAFE Project / The EUROSTARS SAFE-E Project; www.safe-project.eu

[49] The SAFE / SAFE-E Consortium. Deliverable D3.1.3 / D3.4 “Proposal for extension of Meta-model for safety-case modeling and documentation”. www.safe-project.eu. 2013.

[50] S.Voss, B. Schätz, M. Khalil, C. Carlan “A step towards Modular Certification using integrated model-based Safety Cases” VeriSure 2013.

[51] S. Voss, B. Schätz, “Deployment and Scheduling Synthesis for Mixed-Critical Shared-Memory Applications”. Proceedings of the 20th Annual IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS) 2013.

[52] M.Khalil, B. Schätz, S. Voss. “A Pattern-based Approach towards Modular Safety Analysis and Argumentation”. Embedded Real Time Software and Systems Conference (ERTS2014) – Toulouse, France. February, 2014.

[53] SPES2020 Consortium. K. Pohl, H. Hönninger, R. Achatz, M. Broy. “Model-Based Engineering of Embedded Systems – The SPES 2020 Methodology”. Springer-Verlag. 2012.

[54] H. Mili, G. El-Boussaidi. “Representing and applying design patterns: what is the problem?” Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science. Volume 3713, pp 186-200, 2005.

[55] E. Verhulst. OPENCOSS Project Presentation. “Cross-domain systems and safety engineering: is it feasible?”. Flanders Drive Seminar. Brussels. 2013.

[56] S. Nair, J. L. de la Vara, M. Sabetzadeh, L. Briand. “Classification, Structuring, and Assessment of Evidence for Safety: a Systematic Literature Review”. 6th IEEE International Conference on Software Testing, Verification and Validation (ICST 2013).

[57] The EUROSTARS SAFE-E Consortium. Deliverable D4.3 “Prototypical Plug-In Implementation for Test Suite Generation”. Fortiss GmbH. 2014

[58] M.Khalil, A. Prieto, F. Hölzl. “A pattern-based approach towards the guided reuse of safety mechanisms in the automotive domain”. International Symposium on Model-Based Safety and Assessment IMBSA2014 – Munich, Germany. Springer LNCS 8822. October, 2014.

[59] The SAFE Consortium. Deliverable D4.2.6.b “Final version of plug-in for safety and multi criteria architecture modeling and benchmarking”. ITEA2. 2014

[60] The SAFE / SAFE-E Consortium. Deliverable D3.6.b “Safety Code Generator Specification”. www.safe-project.eu. 2013.

[61] The SAFE / SAFE-E Consortium. Deliverable D4.2.7.b / D4.5b “Plug-In for Safety Code Generation”. www.safe-project.eu. 2014.

[62] The CHROMOSOME Middleware project. http://www.fortiss.org/en/research/projects/chromosome/. Fortiss GmbH. 2014.

