Improving Consistency Checks between Safety Concepts and View Based Architecture Design

Pablo Oliveira Antonino[a*], **Mario Trapp**[a]
[a] Fraunhofer IESE, Kaiserslautern, Germany

Abstract: Despite the early adoption of ISO 26262 by the automotive industry, managing functional safety in the early phases of system development remains a challenge. One key problem is how to efficiently keep safety assurance artifacts up-to-date considering the recurrent requirements changes during the system's lifecycle. Here, there is a real demand for means to support the creation, modification, and reuse of safety assurance documents, like the Safety Concepts described in ISO 26262. One major aspect of this challenge is inconsistency between safety concepts and system architecture. Usually created by different teams at different times and in different contexts of the development environment, these artifacts are often completely disassociated. This becomes even more evident when system maintenance is necessary; in this case, the inconsistencies result in intensive efforts to update the safety concepts impacted by the changes, and, consequently, significantly decrease the efficiency and efficacy of safety assurance. To overcome this challenge, we propose a model-based formalization approach for specifying safety concepts that allows creating precise traces to architectural elements while specifying safety concepts using natural language. We observed that our approach minimize the inconsistencies between safety models and architecture models, and offers basis to perform automated completeness and consistency checks.

Keywords: Safety Concepts, Safety Requirements, Architecture Design, Traceability.

1. INTRODUCTION

In 2011, ISO 26262 [1] was published as a safety standard in the automotive industry, emphasizing functional safety management in early phases of system development. Despite its early adoption, there are still open issues that limit the efficiency of assuring the safety of complex systems. Our experience has shown that two core contributors to these issues are (i) recurrent requirements changes during development time and over the system's lifetime, and (ii) the multitude of different artifacts to be considered. This intermittent dynamics leads to challenges regarding how to efficiently keep safety assurance artifacts up to date. One central safety assurance artifact defined in ISO 26262 is the Safety Concept. Safety concepts are requirements with a strong emphasis on the architectural elements that compose the measures to be used to prevent safety-critical failures [1].

In practice, safety concepts have been defined by means of natural text in documents, spreadsheets, or requirements databases. Sometimes, graphical notations like the Goal Structuring Notation (GSN) [2] or UML [3] are used to provide a more structured overview. Nevertheless, the lack of an underlying formalism of these approaches is a key factor contributing to the incompleteness and inconsistency of safety concept specifications.

One major aspect of this challenge is the relationship between safety concepts and the system architecture. As these two artifacts are usually created by different teams at different moments and in different contexts of the system development environment, they are, often, completely disassociated. However, by definition, the requirements defined in safety concepts, often result from a safety analysis of the preliminary architecture. Therefore, this lack of traces between safety concept and architecture

[*] pablo.antonino@iese.fraunhofer.de

is a key factor contributing to inconsistencies between safety concepts and the actual architecture design, and, consequently, to the incompleteness of safety concept specifications.

To overcome this challenge, we propose a model-based formalization technique for specifying safety concepts that supports safety engineers in creating precise traces to architectural elements while specifying safety requirements using natural language. Our approach consists of two items: (i) the Safety Concept Decomposition Pattern, which is a structural decomposition of elements that we understand to be mandatory in any safety concept specification, and (ii) the Parameterized Safety Concept Specification templates, which are generic parameterized textual templates that should be instantiated for the different elements of the Safety Concepts Decomposition pattern, and has the purpose of guide engineers during the specification of the safety concepts using natural language. With that, we ensure seamless integration of safety concepts and architectural design without the need to use formal specification languages like Lambda Calculus or Z, rather preserving intuitiveness during the specification of safety concepts.

The remainder of this paper is organized as follows: In Section 2, we provide a general overview of ISO 26262, particularly of safety concepts; in Section 3, we discuss the main challenges in specifying safety concepts in practice; in Section 4 we discuss the related works; in Section 5, we present our approach; in Section 6, we show how part of the safety concept of a Power Sliding Door Module is specified with our approach; and in Section 7, we conclude and present perspectives for future works.

2. ISO 26262 AND THE NOTION OF SAFETY CONCEPTS

ISO 26262 is an adaptation of IEC 61508 [4], addressing functional safety of electrical and/electronic (E/E) systems in the automotive industry. It defines a safety lifecycle that addresses safety-related aspects during the concept, development, and production phases (cf. Figure 1). It is important to mention that ISO 26262 deals only with possible hazards caused by malfunctions of E/E safety-related systems; it does not address hazards like electric shocks, radiation, or corrosion (a complete list can be found in [1]). Safety Concepts are particularly addressed in the Concept and Product Development phases (cf. Figure 1). Therefore, in this section we will focus only in these two phases of the safety lifecycle.

The Concept phase is where the following items are considered: (i) item definition, (ii) initiation of the safety lifecycle, (iii) hazard analysis and risk assessment, and (iv) the functional safety concept. The main item of interest for us in this phase is the Functional Safety Concept (FSC), which is the "*specification of the functional safety requirements, with associated information, their allocation to architectural elements, and their interaction necessary to achieve safety goals.*" [1].

The Product Development at the system level phase is where technical safety requirements (refinement of the functional safety concept) are specified, taking into account not only the functional concept, but also technical aspects of the preliminary architecture. The specification of the technical safety requirements and their allocation to system elements (software and hardware) is called Technical Safety Concept (TSC) [1].

3. SAFETY CONCEPT SPECIFICATION IN PRACTICE

To better understand these challenges, let us consider the example of a Power Sliding Door Module (PSDM) system adapted from [5]. We adapted the function network (cf. Figure 2) and the deployment view (cf. Figure 3) from [5], and created a data view (cf. Figure 4) on our own, assuming, then, these three views as the PSDM preliminary architecture. To get a general understand on the PSDM, let us start with the central component *Open Door Computation* (cf. Figure 2), whose mission is to trigger the door opening based on two events:

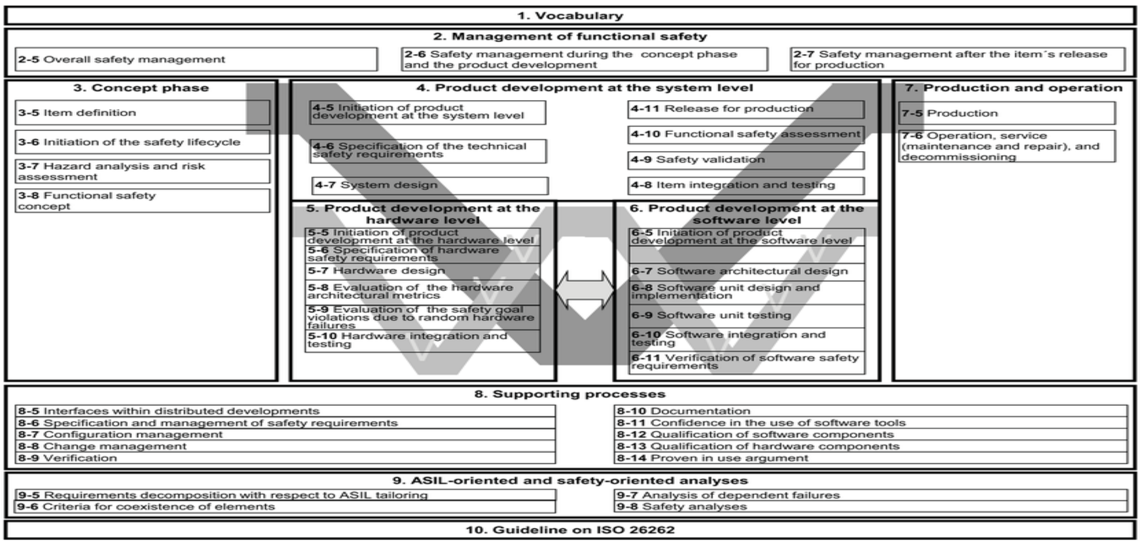

1. Vocabulary
2. Management of functional safety
2-5 Overall safety management
2-6 Safety management during the concept phase and the product development
2-7 Safety management after the item´s release for production
3. Concept phase
3-5 Item definition
3-6 Initiation of the safety lifecycle
3-7 Hazard analysis and risk assessment
3-8 Functional safety concept
4. Product development at the system level
4-5 Initiation of product development at the system level
4-6 Specification of the technical safety requirements
4-7 System design
4-11 Release for production
4-10 Functional safety assessment
4-9 Safety validation
4-8 Item integration and testing
5. Product development at the hardware level
5-5 Initiation of product development at the hardware level
5-6 Specification of hardware safety requirements
5-7 Hardware design
5-8 Evaluation of the hardware architectural metrics
5-9 Evaluation of the safety goal violations due to random hardware failures
5-10 Hardware integration and testing
6. Product development at the software level
6-5 Initiation of product development at the software level
6-6 Specification of software safety requirements
6-7 Software architectural design
6-8 Software unit design and implementation
6-9 Software unit testing
6-10 Software integration and testing
6-11 Verification of software safety requirements
7. Production and operation
7-5 Production
7-6 Operation, service (maintenance and repair), and decommissioning
8. Supporting processes
8-5 Interfaces within distributed developments
8-6 Specification and management of safety requirements
8-7 Configuration management
8-8 Change management
8-9 Verification
8-10 Documentation
8-11 Confidence in the use of software tools
8-12 Qualification of software components
8-13 Qualification of hardware components
8-14 Proven in use argument
9. ASIL-oriented and safety-oriented analyses
9-5 Requirements decomposition with respect to ASIL tailoring
9-6 Criteria for coexistence of elements
9-7 Analysis of dependent failures
9-8 Safety analyses
10. Guideline on ISO 26262

Figure 1 - ISO 26262 overview (extracted from [1]).

i Passenger/driver request: The request to open the sliding door is triggered when the driver or passenger presses the *Door Opener button* (cf. Figure 2). This sends a signal to the *Door Opener Request Processor*, which is responsible for converting the door opener request signal to a format that can be used by the *Open Door Computation* component.

ii Vehicle Speed: To determine the *vehicle speed* (cf. **Figure 4**), first the *wheel rotation speed* is read by the *Wheel Rotation Speed Sensor* (cf. Figure 2), which then sends this information to the *Wheel Rotation Speed Processor*, which is responsible for converting the sensed wheel rotation speed into a format that can be used by the other components. Next, the processed *wheel rotation speed* is sent to the *Computation Vehicle Speed* component, which computes the vehicle speed based on this information. From this point, the computed vehicle speed is sent to the *Open Door Computation* component and also to the *Vehicle Speed Integrity Checker*, which checks if that the vehicle speed is not corrupted or outside a predefined acceptable value range. If the speed value is not as expected, it will notify the *Open Door Computation* component that the value sent by the *Computation Vehicle Speed* component should not be accepted.

Once the *Open Door Computation* component receives the open door request and the vehicle speed, it evaluates if the vehicle is at a speed that allows the door to be opened, which, according to the specification available in [5], is 15km/h. If the vehicle is at 15km/h or less, the *Open Door computation* component notifies the *Open Door Signal Generator* component, which, then, sends a signal that triggers the *Sliding Door Actuator*, which is responsible for opening the sliding door. There is also a component called *Sliding Door Actuator Monitor*, which is responsible for verifying whether the door was indeed opened. If it detects that the actuator did not work properly, it notifies the *Open Door Computation* component to re-send the open door command.

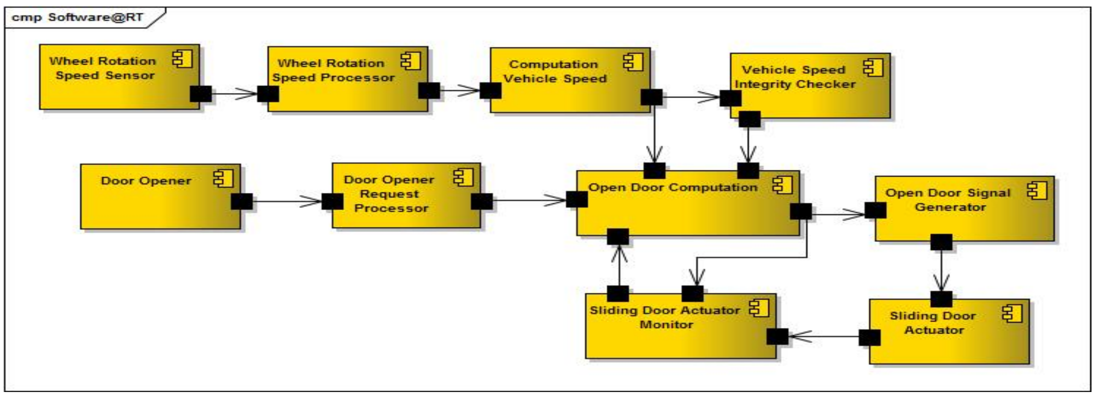

cmp Software@RT
Wheel Rotation Speed Sensor
Wheel Rotation Speed Processor
Computation Vehicle Speed
Vehicle Speed Integrity Checker
Door Opener
Door Opener Request Processor
Open Door Computation
Open Door Signal Generator
Sliding Door Actuator Monitor
Sliding Door Actuator

Figure 2 – Power Sliding Door Module functional network (adapted from [5]).

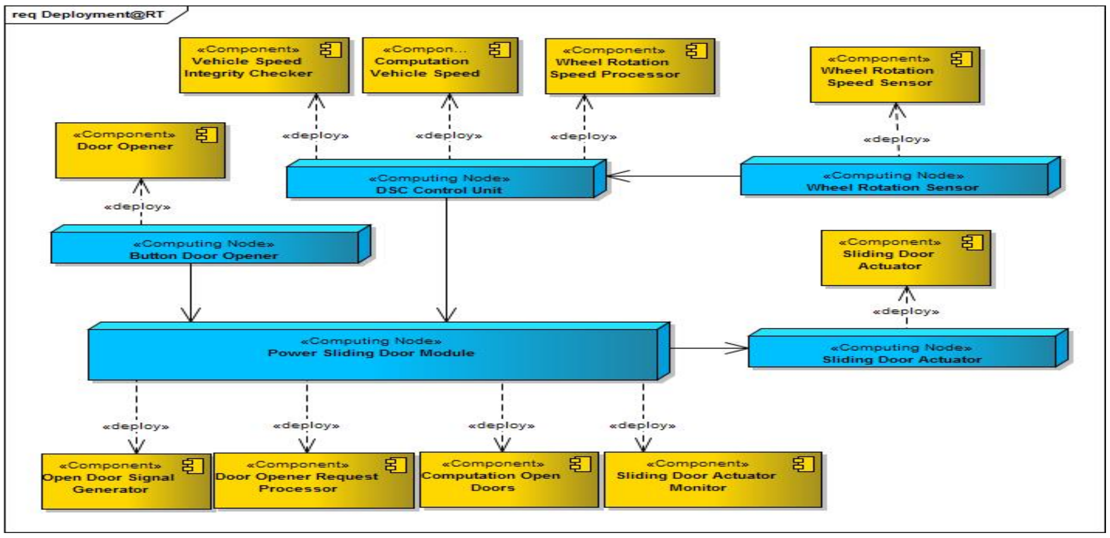

req Deployment@RT
«Component» Vehicle Speed Integrity Checker
«Compon... Computation Vehicle Speed
«Component» Wheel Rotation Speed Processor
«Component» Wheel Rotation Speed Sensor
«Component» Door Opener
«deploy»
«Computing Node» DSC Control Unit
«Computing Node» Wheel Rotation Sensor
«Computing Node» Button Door Opener
«Component» Sliding Door Actuator
«Computing Node» Power Sliding Door Module
«Computing Node» Sliding Door Actuator
«Component» Open Door Signal Generator
«Component» Door Opener Request Processor
«Component» Computation Open Doors
«Component» Sliding Door Actuator Monitor

Figure 3 - Power Sliding Door Module deployment view (adapted from [5]).

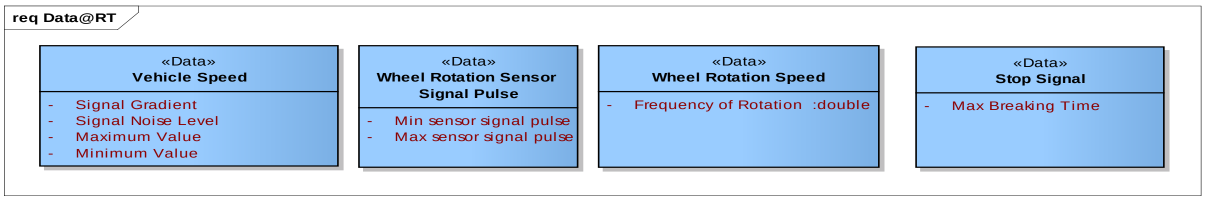
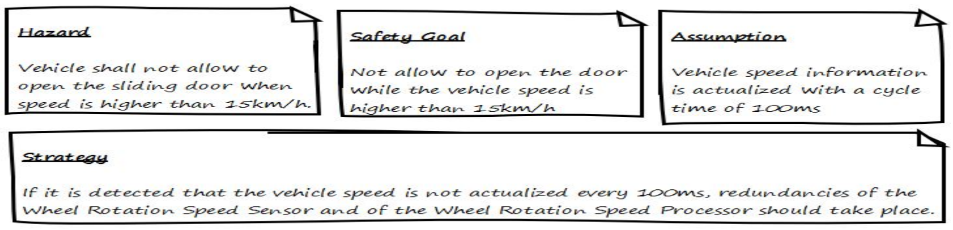

req Data@RT
«Data» Vehicle Speed
- Signal Gradient
- Signal Noise Level
- Maximum Value
- Minimum Value
«Data» Wheel Rotation Sensor Signal Pulse
- Min sensor signal pulse
- Max sensor signal pulse
«Data» Wheel Rotation Speed
- Frequency of Rotation :double
«Data» Stop Signal
- Max Breaking Time

Figure 4 - Power Sliding Door Module data view.

Let us assume that the safety engineers identified that if the vehicle speed is not updated to the PSDM system within 100ms, the safety of the passengers might be compromised, because they might be able to open the door while the vehicle is at a high speed. To better structure this potential situation, let us assume that they have explicitly specified the hazard, safety goal, and a strategy (cf. Figure 5) that should be addressed by the system architecture to avoid the occurrence of this situation. The specification presented in Figure 5 is already decomposed to a much greater extent than in actual practice. In general, one will only find a statement like: *"The information about the actual vehicle speed should be updated within a cycle time of 100ms"* and a simple link from this textual statement to a component in the architecture model that addresses this safety requirement, and this link will be considered the safety concept (cf. [5]). This kind of traceability might be useful when the traces are simple, for instance, when the architectural elements that address a safety concept are located in only one architecture diagram. Now let us consider that to address this safety concept, more than one component is necessary; let's assume that it is also necessary to modify deployment items, data formats, and types, change the structure of a component in terms of adding and/or removing ports, change data flows, etc. Even appropriate automated support, such as the one provided by PREEvision[†] and MEDINI Analyze[‡], do not provide adequate support for specifying safety concepts and, at the same time, supporting the creation of proper trace links to architectural elements from different architecture views along the textual specification.


Hazard

Vehicle shall not allow to open the sliding door when speed is higher than 15km/h.

Safety Goal

Not allow to open the door while the vehicle speed is higher than 15km/h

Assumption

Vehicle speed information is actualized with a cycle time of 100ms

Strategy

If it is detected that the vehicle speed is not actualized every 100ms, redundancies of the Wheel Rotation Speed Sensor and of the Wheel Rotation Speed Processor should take place.

Figure 5 - PSDM Safety Concept specification (adapted from [5]).

[†] http://vector.com/vi_preevision_en.html

[‡] http://www.ikv.de/index.php/en/products/functional-safety

4. RELATED WORK

As already mentioned, in practice, the most common way to specify safety concepts is using natural text in documents, spreadsheets, or requirements databases. There are, nevertheless, some initiatives aimed at providing more structured specifications, such as the one proposed by Habli et al. [6], where it is investigated how traceability between safety cases modeled with the Goal Structuring Notation (GSN) and architectural models represented in SysML can be improved. Birch et al. [2] analyzed the implicit safety argument structure of ISO 26262 and conducted a case study showing how these arguments can be logically decomposed and structured using GSN. Denney and Pai [7] proposed a breakdown pattern specifically for safety case specifications aimed at automated instantiation of arguments. Domis et al. [8] introduced Safety Concepts Trees (SCTs) as a means for modeling how safety goals are broken down into safety requirements, which, in turn, are continuously refined by other safety requirements using typical logical gates. However, SCTs are more concerned with taking safety concepts to the modelling domain, and not with the content of the elements that compose safety concepts specifications as we do.

In a nutshell, we indicate how to structure and relate the elements that we consider important to be in safety concepts specification (cf. Section 5.1), and how to elaborate the content of each element (cf. Section 5.2). It means that our approach is not strictly tight to any modelling approach (e.g. SCTs and GSN); actually, engineers are free to use the approach and notation they are used to, and just have to consider the elements described in the Safety Concepts Decomposition Pattern, and use the guidelines to elaborate their contents.

With respect to the support to structure and relate the elements of safety concepts specification, the closest to our approach are those proposed by Birch et al. [2] and Denney and Pai [7]. Nevertheless, Denney and Pai focus on a structural decomposition of safety cases, and do not offer appropriate means for supporting safety concept decomposition. Birch et al. propose rational arguments strategies that are useful to justify *"why"* a safety requirement is, indeed, a Safety Requirement, using GSN to structure the justification argument. Our approach does care with the "*why*" as well, but in a more structured way (cf. Section 5), as we explicitly indicate elements that, when precisely described, provides enough basis to justify the existence of a safety concept, such as the potential causes of failures, their nature, and the associated failure mode. Additionally, we provide means to come up with precise descriptions of (i) these elements that justify the existence of safety concepts, and (ii) safety requirements specifications that indicate, already along their definition, *"how"* the architecture should be modified to address failure causes, and, consequently, satisfy the safety goal.

With respect to support in elaborating the content of the elements that compose safety concepts specifications, Firesmith [9] presents parameterized requirements for different types of safety requirements. He argues that, because safety requirements usually have the form of system specific quality criteria associated with different levels of quality measures, they can be written as instances of parameterized safety requirements templates. However, the parameterized templates proposed by Firesmith are far from appropriate to specify safety concepts elements. Therefore, we have created parameterized templates that are appropriate to the elements of our Safety Concepts Decomposition Pattern, strongly focusing on parameterization of architecture design elements.

Summarizing, to the best of our knowledge, there is no other model-based approach that allows semi-formal hierarchical decomposition of functional and technical safety concepts, and that also guides the creation of traces to architectural elements from multiple views while specifying safety concepts using natural language.

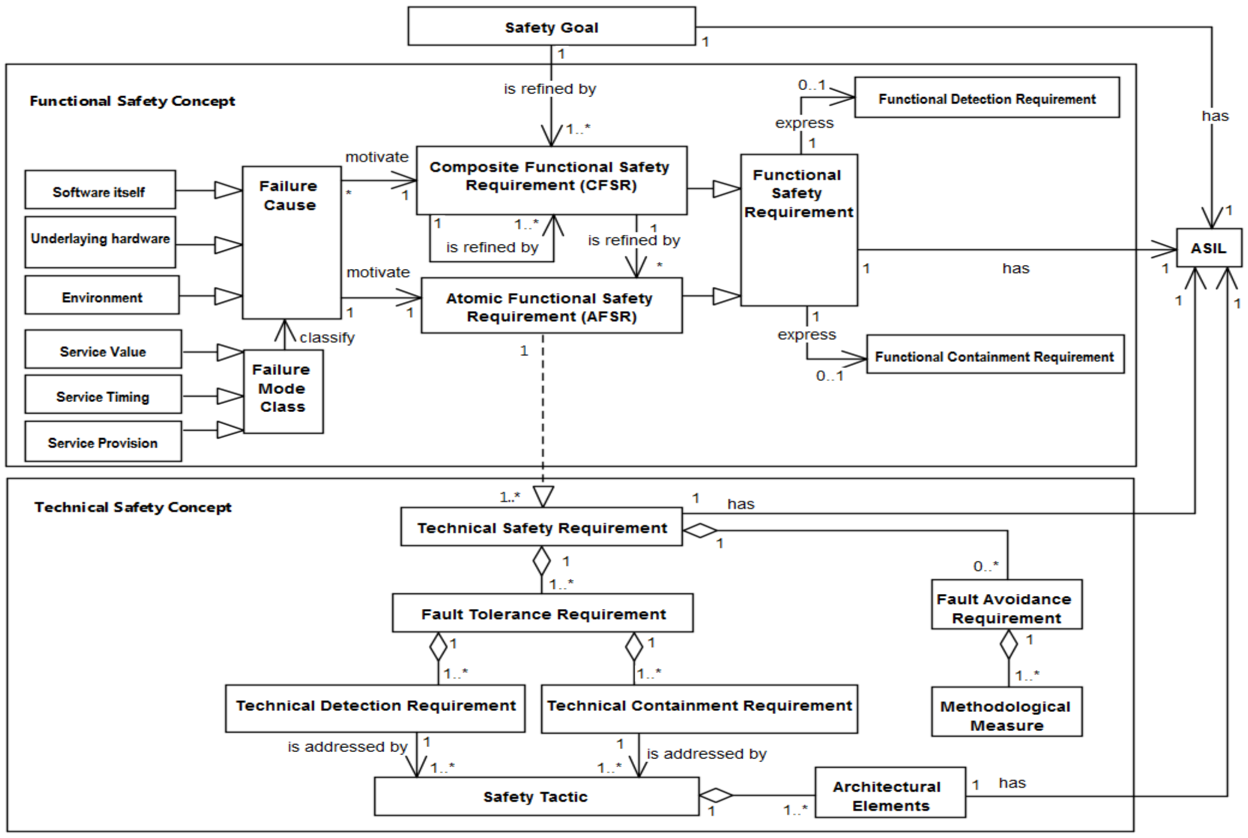
5. MODEL-BASED FORMALIZATION OF SAFETY CONCEPTS SUPPORTING THE USE OF NATURAL LANGUAGE

In this section we present our approach to specify safety concepts, which consists of two core items: (i) the Safety Concept Decomposition Pattern, which is a structural decomposition of elements that we consider important to be in any safety concept specification, and (ii) the Parameterized Safety Concept Specification templates, which are generic parameterized textual templates that should be instantiated for the different elements of the Safety Concepts Decomposition pattern

To properly specify safety concepts with our approach, we assume that the following three sets of artifacts should be already in place: (i) the result of the Hazard and Risk analysis, (ii) the preliminary architecture, and (iii) the results of Failure Mode and Effect Analysis – FMEA, where problems in the preliminary architecture are described.

5.1. Safety Concept Decomposition Pattern

ISO 26262 explicitly indicates the content of safety concept specifications, but it doesn't specify a defined structure for them. What happens in practice is that each safety concept specification has a different structure, most likely based on the understanding and experience of the engineers involved in the specification process. Furthermore, because of this lack of a structured definition, following the argumentation of safety concept specifications is not a trivial task, especially if the reader is not the author. The consequence is that it becomes difficult to ensure that all the aspects demanded by ISO 26262 are present in the specification, and it is also hard to ensure that the architecture elements referenced by the safety concepts are consistent with those specified in the architecture design. To overcome this challenge, we have specified the Safety Concept Decomposition Pattern, which comprises elements that we understand as being fundamental for ensuring the completeness of functional and technical safety concepts. The metamodel of the Safety Concept Decomposition Pattern is shown in Figure 6, which is followed by the descriptions of its elements.



Safety Goal
Functional Safety Concept
is refined by
Functional Detection Requirement
express
has
Software itself
Underlaying hardware
Environment
Failure Cause
motivate
Composite Functional Safety Requirement (CFSR)
Functional Safety Requirement
ASIL
is refined by
Atomic Functional Safety Requirement (AFSR)
classify
Service Value
Service Timing
Service Provision
Failure Mode Class
Functional Containment Requirement
Technical Safety Concept
Technical Safety Requirement
Fault Tolerance Requirement
Fault Avoidance Requirement
Technical Detection Requirement
Technical Containment Requirement
Methodological Measure
is addressed by
Safety Tactic
Architectural Elements

Figure 6 - Safety Concepts Decomposition pattern metamodel.

- **Safety Goal:** Top-level requirement resulted from the hazard and risk analysis assessment. For each hazardous event with an ASIL evaluated in the hazard analysis, a safety goal should be determined [1].

- **ASIL:** Levels to specify the necessary requirements and safety measures to avoid unreasonable residual risks, with D representing the most stringent and A the least stringent level [1].

- **Failure Cause:** Condition of a system (or parts of it) that motivates the existence of a safety requirement. It is usually identified when conducting a safety analysis to identify paths that can lead to a critical system failure. We adopted the convention proposed by Wu and Kelly [10], thus assuming that a cause can be categorized according to one of the three following abstract types:

  - **The software itself:** Incomplete or inaccurate specification, or incorrect design and implementation can cause unexpected behavior of the software.

  - **Underlying hardware:** Correct software can still misbehave because of unexpected behavior of the underlying hardware.

  - **Environment:** Also known as environment disturbances, causes can be originated outside the software and can be fed into the system in the form of inputs.

- **Failure mode class:** Manner in which an element or an item fails [1]. We understand that a failure mode class is a higher level of abstraction that classifies failure causes, in order to enrich them semantically. Following the example of Wu and Kelly [10], we also adopted the failure mode classification proposed by Fenelon et al. [11]:

  - **Service provision:** Omission (expected event does not occur), Commission (spurious occurrence of event).

  - **Service timing:** Early (event occurs before time required), Late (event occurs after the time required).

  - **Service value:** Coarse incorrect (detectable incorrect value delivered), Subtle incorrect (undetectable incorrect value delivered).

- **Functional Safety Requirement (FSR):** Specification of implementation independent safety measure, including its safety-related attributes [1]. We understand that it can express a Functional Detection Requirement (high level description of measures to detect failures) or a Functional Containment Requirement (high level description of measures to handle failures).

- **Composite Functional Safety Requirement (CFSR):** Functional Safety Requirement that has more than one failure cause that motivates its existence. i.e., $cfsr \in CFSR: \{FSR || FSR.cause \geq 1\}$. CFSRs can be refined by other CFSRs, or by Atomic Functional Safety Requirement (AFSR).

- **Atomic Functional Safety Requirement (AFSR):** functional safety requirement that has only one failure cause that motivates its existence. i.e., $afsr \in AFSR: \{FSR || FSR.cause = 1\}$. AFSRs refine CFSRs that cannot be decomposed into finer grains anymore. AFSR are realized by a set of Technical Safety Requirements.

- **Technical Safety Requirement (TSR):** Description of strategies to realize an Atomic Functional Safety Requirement [1].

- **Fault Avoidance Requirement:** We adopted the definition of Laprie, and consider Fault Avoidance Requirements as a group of means that aim for systems free of faults, and comprises fault prevention and fault removal mechanisms [12].

- **Fault Tolerance Requirement:** Description of means that allows *"living"* with systems that are susceptible to faults. [12].

- **Technical Detection Requirement:** Description of how Functional Detection Requirements will be realized by elements of the architecture design.

- **Technical Containment Requirement:** Description of how Functional Containment Requirements will be realized by elements of the technical architecture design. It describes means to take a system from a state containing errors and faults to a state without detected errors and without faults [12].

- **Safety tactics**: Architectural design decisions made to avoid or handle failures to which safety-critical systems are subject [10]. They become more concrete when they are realized by safety patterns indicating architectural elements (mainly components, connector, deployment units, and communication channels) and a set of constraints on how instances of these types should be combined into a system to detect or contain a failure [13].

5.2. Parameterized Safety Concept Specification Templates

Even though we do understand that the user should be free to write textual the safety concepts specifications, we believe that some guidelines can be useful to indicate items that must not be absent from the specification. Therefore, we have created Parameterized Safety Concepts Specification Templates that should be used to guide the engineers during the specification of some elements of the Safety Concepts Decomposition Pattern. The template elements delimited by square brackets are textual descriptions that should contain elements to be linked to the architecture design. We understand that these elements should be selected during the safety concept specification in order to ensure early traceability in the specification process. Moreover, these elements are mandatory in the safety concept specification because they are about the very core notion of safety concepts defined in ISO 26262, which is to assign architectural elements to the safety requirements. Therefore, if architectural elements are not included in the safety concept specification, its completeness is compromised. The textual descriptions between parentheses have no such strict constraints; however, they still address some constraints in the sense that they indicate important constraints that should be considered, as, for instance, an indication that a signal should not be sent later than within 2ms. The parameterized templates are as follows:

- **Safety Goal:** [System || Component Group || Component || Computing Node] shall (avoid || not cause || not allow || not be || not || no) (harm).

- **Functional Detection Safety Requirement:** The System shall detect (accidental harm | Safety incident | Hazard | Safety Risk). This template was reused from [9], and there it describes safety requirements of type Detection of Violation of Prevention.

- **Functional Containment Safety Requirement:** When the System shall detects (accidental harm | Safety incident | Hazard | Safety Risk), then the system shall (List of Actions). This template was also reused from [9], and there it describes safety requirements of type Reaction to Violation of Prevention.

- **Technical Safety Requirement:** The template for this element depends on the failure mode classification of the failure cause that motivates the existence of the Atomic Safety Requirement associated to the Technical Safety Requirement. The two possible templates are:

- **Service Value Failure Mode:** [System || Component Group || Component || Computing Node] shall (perform action) [artifact affected by action] (Values threshold of measurement: within || exactly with || not exceed || not less than) [Data constraint].

- **Service Timing Failure Mode:** [System || Component Group || Component || Computing Node] shall [perform action] [artifact affected by action] (timing threshold of measurement: within || before || after || exactly || no later than) [timing constraint].

➢ **Fault Tolerance Requirement:** Detect and Handle (type of violation) violation of [artifact affected by action].

➢ **Technical Detection Requirement:** It should be detected if [artifact affected by action] is not (action performed - past tense) (threshold of measurement) [Value Constraint||Timing Constraint].

➢ **Technical Containment Requirement:** This element should be described as free text so the engineer can describe his strategy in detail. However, it is important to highlight that this description must reference architectural elements.

➢ **Safety Tactics:** We have created specification templates for almost all the safety patterns described by Douglass [13], and examples are shown in Section 6.

It is important to highlight that the elements of the Safety Concepts Decomposition Pattern doesn't have associated parameterized templates are those that not necessarily have to reference architectural elements. However, the references can be created, whenever the engineers understand that such references will contribute to a clearer understand of the safety requirement. Another aspect to be considered is that we do not impose that the textual constructions have to be strictly formulated as indicated in the templates. However, it is strongly recommended that the specifications contain references to the elements indicated between the square brackets and the parentheses, since they are the key to ensuring the completeness and consistency of the safety concept specifications.

To illustrate the instantiation of these parameterized templates, consider the example below, where a Technical Safety Requirement and a Fault Tolerance Requirement are specified for an Airbag System. The color coding is intended to make it easier to understand the mapping between the template and the example:

➢ **Technical Safety Requirement:**

- **Template:** [System || Component Group || Component || Computing Node] shall (perform action) [artifact affected by action] (Values threshold of measurement: within || exactly with || not exceed || not less than) [Data constraint]

- **Example:** [Front Acceleration sensor] shall (send) [Front sensed acceleration signal amplitude] (with at least) [0,56dB]

➢ **Fault Tolerance Requirement:**

- **Template:** Detect and Handle (type of violation) violation of [artifact affected by action]
- **Example:** Detect and Handle (value range) violation of [front sensed acceleration signal amplitude]

6. SPECIFYING SAFETY CONCEPTS FOR A POWER SLIDING DOOR MODULE WITH OUR APPROACH

In this section we show how the Power Sliding Door Module example described in [5] look like when specified with our approach. However, due to space constraints we will focus in only two safety requirements: (i) "*The information about the actual vehicle speed should be actualized with a cycle time of 100ms.*", and (ii) "*The wheel rotation speed should be measured with an accuracy of at least 30 rad/min*". Also due to space constraints, along this section we describe in detail only the first requirement; the model with the two requirements is show in Figure 7. It also important to mention that the the preliminary architecture considered was the one shown in Figure 2, Figure 3, and Figure 4. Please, also note that the items from the specification retain the square brackets and the parentheses to facilitate the understanding of the example. However, in practice, when the trace links are created, these signs should be removed to guarantee the natural flow of reading.

- **Safety Goal:** [Vehicle] shall (not allow) (door to be opened while the vehicle speed is in motion).

- **CFSR:** Control Unit shall send accurate vehicle speed information to power sliding door module.

- **Failure Cause 1:** Vehicle speed is not updated in time.

- **Failure Cause 2:** Wheel vehicle speed is not measured with the proper accuracy.

Note: Due to the lack of space, we will show the decomposition used to address only Failure Cause 1. The description of the items related to Failure Cause 2 can be seen in Figure 7.

- **AFSR (referring to Cause 1):** The information about the actual Vehicle Speed should be updated with an updating cycle of vehicle speed. ***Violation:*** *Timing Violation – Time to execute operation****; Failure Mode:*** *Service Timing****.***

- **Technical Safety Concept (Service Timing Failure Mode):** [Computation Vehicle Speed Component] shall (update) [vehicle speed] (not later than) [a cycle time of 100 ms].

- **Fault Tolerance Requirement:** Detect and handle (timing accuracy) violation of [vehicle speed] updating.

- **Detection Requirement:** It should be detected if [Vehicle Speed] is not (updated) (not later than) [a cycle time of 100 ms] at the **[Computation Vehicle Speed component]**.

- **Containment Requirement:** Redundancy - there should be a [redundant Wheel Rotation Speed Sensor] and a [redundant Rotation Speed Processor] that should substitute the [Wheel Rotation Speed Sensor] and [Rotation Speed Processor] if it is detected that the [vehicle speed] is not updated every [100ms]**.**

- **Detection Safety Tactic:** Let's assume that the engineers decided to monitor the vehicle speed using a Watchdog. As previously mentioned, we have specified a grammar for most of the safety patterns described by Douglass [13].

  - **Template:** [Watchdog component] monitors [monitored architectural element] to check if [monitored aspect] is (action) (threshold of measurement) [Timing Constraint].

  - **Example:** [Watchdog component] monitors [computation Vehicle Speed component] to check if [vehicle speed] is (updated) (not later than) [a cycle time of 100ms].

- **Containment Safety Tactic:** Let's assume that the engineers decided to use homogeneous redundancies of Wheel Rotation Speed Sensor and Rotation Speed Processor.

  - **Template:** [Component], which is deployed to [Computing Node||Thread], have (n) homogeneous redundancy(ies), which is(are) deployed to: [Computing nodes||Threads] [n .. n-1].

  - **Example:** [Rotation Speed Processor], which is deployed to [DSC Control Unit], have (1) homogeneous redundancy, which is deployed to: [DSC Control Unit].

We observed that safety concepts bases on the Safety Concepts Decomposition Pattern offer great basis for safety engineers in identifying if all failure causes were properly safe-guarded. Another positive aspect is about the compliance created between functional and safety concepts, which is a valuable step towards safety concepts correctness. Another observed benefit is with respect to the consistency improvement between safety concepts and architecture design, because the preliminary architecture can be considered while the safety engineer are writing down the safety concept using the parameterized templates as basis, and not, as usual, first defining the safety concepts, and only later indicate which architecture element addresses them. In a sense, we observed improvements the overall system specification process, because engineers can make early detection if any rearrangement in the design is necessary, or if new architecture elements are required. Another positive aspect is that safety engineers are comfortable to use it because they can keep writing the specifications textually, and also can keep using the mechanisms they are used to for specifying safety concepts, because, in a nutshell, we only indicate the elements to be considered in the specification, how they should be structured, and suggests how the textual content should look like. Therefore, they can do it using well know modeling mechanisms such as GSN and SCTs, and tools like MEDINI Analyze and PREEvision.

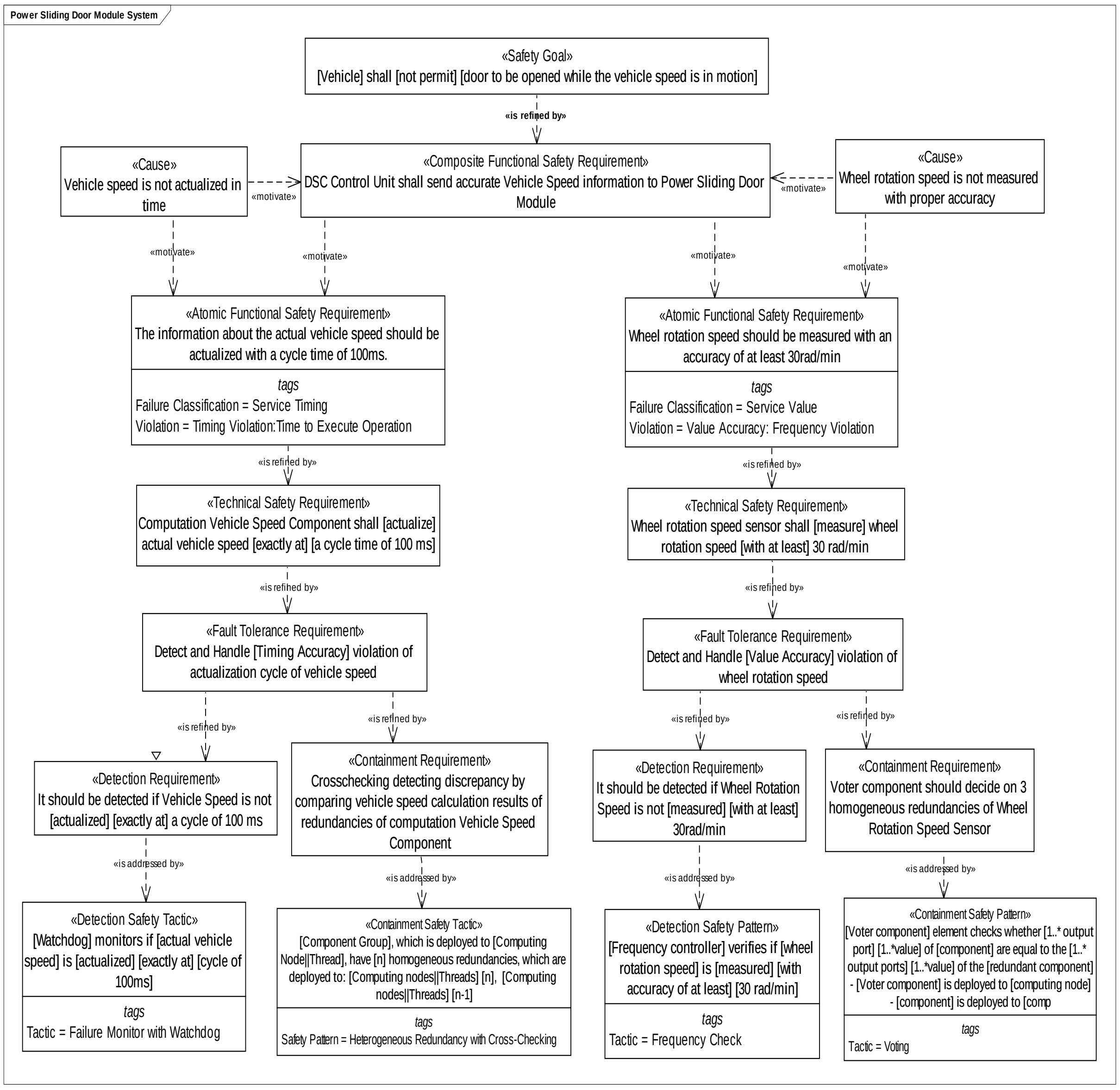

Power Sliding Door Module System
«Safety Goal» [Vehicle] shall [not permit] [door to be opened while the vehicle speed is in motion]
«is refined by»
«Cause» Vehicle speed is not actualized in time
«motivate»
«Composite Functional Safety Requirement» DSC Control Unit shall send accurate Vehicle Speed information to Power Sliding Door Module
«motivate»
«Cause» Wheel rotation speed is not measured with proper accuracy
«motivate»
«motivate»
«motivate»
«motivate»
«Atomic Functional Safety Requirement» The information about the actual vehicle speed should be actualized with a cycle time of 100ms.
tags
Failure Classification = Service Timing
Violation = Timing Violation:Time to Execute Operation
«Atomic Functional Safety Requirement» Wheel rotation speed should be measured with an accuracy of at least 30rad/min
tags
Failure Classification = Service Value
Violation = Value Accuracy: Frequency Violation
«is refined by»
«is refined by»
«Technical Safety Requirement» Computation Vehicle Speed Component shall [actualize] actual vehicle speed [exactly at] [a cycle time of 100 ms]
«Technical Safety Requirement» Wheel rotation speed sensor shall [measure] wheel rotation speed [with at least] 30 rad/min
«is refined by»
«is refined by»
«Fault Tolerance Requirement» Detect and Handle [Timing Accuracy] violation of actualization cycle of vehicle speed
«Fault Tolerance Requirement» Detect and Handle [Value Accuracy] violation of wheel rotation speed
«is refined by»
«is refined by»
«is refined by»
«is refined by»
«Detection Requirement» It should be detected if Vehicle Speed is not [actualized] [exactly at] a cycle of 100 ms
«Containment Requirement» Crosschecking detecting discrepancy by comparing vehicle speed calculation results of redundancies of computation Vehicle Speed Component
«Detection Requirement» It should be detected if Wheel Rotation Speed is not [measured] [with at least] 30rad/min
«Containment Requirement» Voter component should decide on 3 homogeneous redundancies of Wheel Rotation Speed Sensor
«is addressed by»
«is addressed by»
«is addressed by»
«is addressed by»
«Detection Safety Tactic» [Watchdog] monitors if [actual vehicle speed] is [actualized] [exactly at] [cycle of 100ms]
tags
Tactic = Failure Monitor with Watchdog
«Containment Safety Tactic» [Component Group], which is deployed to [Computing Node||Thread], have [n] homogeneous redundancies, which are deployed to: [Computing nodes||Threads] [n], [Computing nodes||Threads] [n-1]
tags
Safety Pattern = Heterogeneous Redundancy with Cross-Checking
«Detection Safety Pattern» [Frequency controller] verifies if [wheel rotation speed] is [measured] [with accuracy of at least] [30 rad/min]
tags
Tactic = Frequency Check
«Containment Safety Pattern» [Voter component] element checks whether [1..* output port] [1..*value] of [component] are equal to the [1..* output ports] [1..*value] of the [redundant component] - [Voter component] is deployed to [computing node] - [component] is deployed to [comp
tags
Tactic = Voting

Figure 7 - PSDM Safety Concepts specified with our approach.

7. FUTURE WORK AND CONCLUSIONS

This work was motivated by the lack of approaches for semi-formal hierarchical decomposition of safety concepts and for the creation of traces to architectural elements while specifying safety requirements using natural language. To fill this gap, we proposed a model-based formalization technique for specifying safety concepts that supports safety engineers in creating precise traces to architectural elements while specifying safety requirements using natural language. The approach consists of a Safety Concept Decomposition Pattern and a Parameterized Safety Concepts Specification Templates, with the former specifying the elements to be considered in a safety concept specification and the latter specifying a grammar containing elements whose presence in textual specifications of safety concepts is strongly recommended to ensure completeness and consistency. This is our first step towards the realization of automated consistency and completeness checks of safety concepts.

Acknowledgments

This work is supported by the Fraunhofer-Innovation Cluster Digitale Nutzfahrzeugtechnologie (Digital Commercial Vehicle Technology), and by the Software Platform Embedded Systems "XT" - SPES XT project. We would also like to thank Sonnhild Namingha for proofreading.

References

[1] International Organization for Standardization. *“ISO/DIS 26262 - Road Vehicles – Functional Safety”*, Technical Committee 22 (ISO/TC 22), Geneva, Switzerland, 2011.

[2] J. Birch, R. Rivett, I. Habli, B. Bradshaw, J. Botham, D. Higham, P. Jesty, H. Monkhouse, and R. Palin. *“Safety Cases and Their Role in ISO 26262 Functional Safety Assessment”*. Springer, In Proceedings of 32nd SAFECOMP, 2013.

[3] Object Management Group. *“UML profile for modeling QoS and FT characteristics and mechanisms.* Technical report, April 2008.

[4] International Organization for Standardization. *“IEC 61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems”*, The International Electrotechnical Commission, Geneva, Switzerland, 1998.

[5] M. Hillenbrand, M. Heinz, K. D. Müller-Glaser, N. Adler, J. Matheis, and C. Reichmann. *“An approach for rapidly adapting the demands of ISO/DIS 26262 to electric/electronic architecture modelling”*. International Symposium on Rapid System Prototyping, 2010.

[6] I. Habli, I. Ibarra, R. Rivett, and T. Kelly. *"Model-Based Assurance for Justifying Automotive Functional Safety”* SAE Technical Paper, 2010.

[7] E. Denney, G. Pai. *“A Formal Basis for Safety Case Patterns”*. Springer, In Proceedings of 32nd SAFECOMP, 2013.

[8] D. Domis, M. Forster, S. Kemmann, and M. Trapp. *“Safety concept trees”*. In Reliability and Maintainability Symposium, 2009. RAMS 2009. Annual, pages 212 - 217, jan. 2009.

[9] D. Firesmith. *“A Taxonomy of Safety-Related Requirements”*, Software Engineering Institute White Paper, 2004.

[10] W. Wu and T. Kelly, *“Safety Tactics for Software Architecture Design*”, in COMPSAC, 2004.

[11] P. Fenelon, J. A. McDermid, M. Nicolson, and D. J. Pumfrey. 1994. “*Towards integrated safety analysis and design.”* SIGAPP Appl. Comput. Rev. 2, 1 (March 1994), 21-32. DOI=10.1145/381766.381770 http://doi.acm.org/10.1145/381766.381770

[12] A. Avizienis, J.-C. Laprie, B. Rendell and C. Landwehr, *“Basic Concepts and Taxonomy of Dependable and Secure Computing,”* IEEE Trans. Dependable Secur. Comput., vol. 1, pp. 11--33, Jan 2004.

[13] B. P. Douglass. “*Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*”. Addison-Wesley Longman Publishing Co., Inc., 2005, Boston, MA, USA.