Contract number: ITEA2 – 10039

Safe Automotive soFtware architEcture (SAFE)

ITEA Roadmap application domains:

Major: Services, Systems & Software Creation

Minor: Society

ITEA Roadmap technology categories:

Major: Systems Engineering & Software Engineering

Minor 1: Engineering Process Support

WP3

Deliverable D3.3.3b: Final specification for comparison of architecture

Due date of deliverable: 27/12/2013

Actual submission date: 20/12/2013

Start date of the project: 01/07/2011 **Duration:** 36 months

Start date of the work task: 30/04/2012 **Duration:** 20 months

Project coordinator name: Stefan Voget (Continental)

Organization name of lead contractor for this deliverable: FZI Forschungszentrum Informatik

Editors: Nico Adler (FZI), Stefan Otten (FZI)

Contributors: Nico Adler (FZI), Stefan Otten (FZI), Eduard Metzker (Vector Informatik), Thomas Peikenkamp (OFFIS), Markus Oertel (OFFIS), Andreas Baumgart (OFFIS), Sebastian Voss (Fortiss), Maged Khalil (Fortiss)

Reviewers: Eduard Metzker (Vector Informatik), Martin Hillenbrand (FZI)

Revision chart and history log

Version	Date	Reason
0.1	04.12.2012	Initialization of document
0.2	05.02.2013	Sections: executive summary, scope, structure, overview of ISO relevant parts
0.3	06.02.2013	Hardware architecture assessment
0.4	06.02.2013	Description of SAFE meta model extension
0.5	12.02.2013	Consistency checks and metrics for the system safety concept
0.6	15.02.2013	Changes in general parts of document; review of existing parts
0.7	18.02.2013	Update: assessment of hardware
0.8	19.02.2013	Update: pattern-based seamless development
0.9	27.02.2013	Update based on review comments
1.0	27.02.2013	Finalization of document D3.3.3a (protected)
1.1	09.07.2013	Update: Feedback of OEM Advisory Board considered
1.2	21.10.2013	Initialization of document D3.3.3b
1.3	04.11.2013	Update of document structure
1.4	26.11.2013	Added Vector contribution: Consistency of safety case (Chapter 9)
1.5	28.11.2013	Added FZI contribution: Assessment of Hardware (Chapter 8)
1.6	13.12.2013	Review Chapter 1 to 9
1.7	16.12.2013	Update: Review comments chapter 1 to 9
1.8	20.12.2013	Added Offis contribution (Chapter 10) Added Fortiss contribution (Chapter 11)
2.0	20.12.2013	Finalization of public document D3.3.3b (published 20.12.2013)
2.0.1	03.02.2014	Fortiss: Changes in Chapter 11 and 6.1, additional contributor

1 Table of contents

2 List of figures

3 List of tables

4 List of abbreviations

ASIC	Application-Specific Integrated Circuit
ASIL	Automotive Safety Integrity Level
AUTOSAR	AUTomotive Open System ARchitecture
E/E	Electric/Electronic
EAST-ADL	Electronics Architecture and Software Technology – Architecture Description Language
ECU	Electrical Control Unit
EDA	Electronic Design Automation
EEA	Electric/Electronic Architecture
FM	Failure Mode
FMEA	Failure Mode and Effects Analysis
FMECA	Failure Mode, Effects and Criticality Analysis
FMEDA	Failure Mode, Effects and Diagnostic Analysis
FR	Failure Rate
FRC	Failure Rate Class
FRD	Failure Rate Distribution
FTA	Fault Tree Analysis
HARA	Hazard and Risk Analysis
HW	Hardware
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
LF	Latent Fault
LFM	Latent-Fault Metric
MPF	Multiple-Point Fault
MPFL	Multiple-Point Fault Latent
OEM	Original Equipment Manufacturer
PFD	Probability of Dangerous Failure on Demand
PFH	Probability of Dangerous Failure per Hour
PMHF	Probabilistic Metric for Random Hardware Failures
RBD	Reliability Block Diagram
RF	Residual Fault
SF	Safe Fault
SFF	Safe Failure Fraction
SG	Safety Goal
SIL	Safety Integrity Level
SM	Safety Mechanism
SPF	Single-Point Fault
SPFM	Single-Point Fault Metric
SR	Safety-Related
XML	Extensible Markup Language
XSD	XML Schema Definition

5 Executive Summary

The methodologies presented in this deliverable D3.3.3 describe qualitative and quantitative assessment of architectures regarding functional safety. Therefore, the methodologies for safety evaluation are in accordance with the needs of ISO 26262, based on requirement analysis performed in the SAFE project.

The deliverable targets architecture evaluation and assessment in terms of model-based development with the focus on functional safety. Thus, the context of architecture modeling with enrichment of failure information and instrumentation for model-based evaluation is presented.

Four specific topics are described into detail: “Assessment of hardware on different level of abstraction”, “Consistency checks for the safety case”, “Common-cause analysis in the geometric perspective using physical properties and environmental conditions” and “Multi-criteria deployment optimization and schedule generation”. The methodologies are classified regarding the relevant parts of ISO 26262 [1].

Moreover, the relation of the methodologies to corresponding work task in the WP4 “Technology Platform” is described in order to provide concepts for tool implementation. This enhances the level of automatism in terms of model-based architecture safety evaluation.

6 Introduction and overview of document

This document for safety evaluation in terms of assessment of electric and electronic architectures provides different methodologies to perform qualitative and quantitative assessment on different abstraction layers and different level of granularity.

6.1 Scope of work task WT3.3.3

Deliverable D3.3.3 deals with the use of SAFE meta model constructs and provides methodologies to assess architectures regarding functional safety. The defined methodologies are shortly presented in the following:

Assessment of hardware on different level of abstraction

To provide a model-based safety evaluation of hardware designs, a minimal set of constructs were extracted out of the SAFE meta model. A methodology and a process description for hardware safety evaluation is provided. Different phases of hardware concept and development as well as qualitative and quantitative assessments are addressed. This contribution facilitates the basis for a research prototype implementation, covered in WT4.2.6.

Consistency checks for the safety case

The SAFE meta model allows to capture the system safety concept in semi-formal way. In WT3.3.3, the SAFE meta model concepts are used to define a simple template for documenting consistency checks and metrics to evaluate and improve the formal quality of the system safety case. This contribution shall provide the basis for tool supported automatic consistency checks and metrics which shall be implemented in WT4.2.6.

Multi-criteria Deployment Optimization and Schedule Generation

The method comprises an efficient approach for generating suitable system architectures for embedded systems efficiently, which allows multi-criteria deployment optimization and schedule generation and can be used as part of a larger design space exploration approach. It is possible to use multiple criteria for the approach, and the implementation counterpart to this document features a multi-criteria capable plug-in that can target ASIL-based allocation, worst-case-execution-time and number of hardware nodes for deployment in the draft version. A concept for the integration of safety metrics identified in the SAFE/SAFE-E project, such as the hardware metrics or geometrical aspects identified in methods 1-3 and presented in sections 9, 10 and 11, will be featured in its final version.

6.2 Structure of the document

The following Section 7 gives an overview of the ten parts of ISO 26262 and classifies the methodologies of this work task to the corresponding parts and clauses. The first methodology for safety evaluation of hardware designs is presented in Section 8. Section 9 provides the second methodology for consistency checks in context of the safety case. Section 10 describes the third methodology regarding common-cause analysis in the geometric perspective. The fourth methodology in context of multi-criteria deployment optimization and schedule generation is presented in Section 11. Finally, in Section 12, a brief conclusion is given.

7 Overview of ISO 26262

This section briefly describes which parts of the ISO 26262 are involved for the two presented methodologies of safety evaluation and assessment. Relevant clauses within the parts of the ISO 26262 are framed in different colors, as shown in Figure 1. A brief explanation for each of the relevant parts is presented as well as the specific relation to the methodologies.

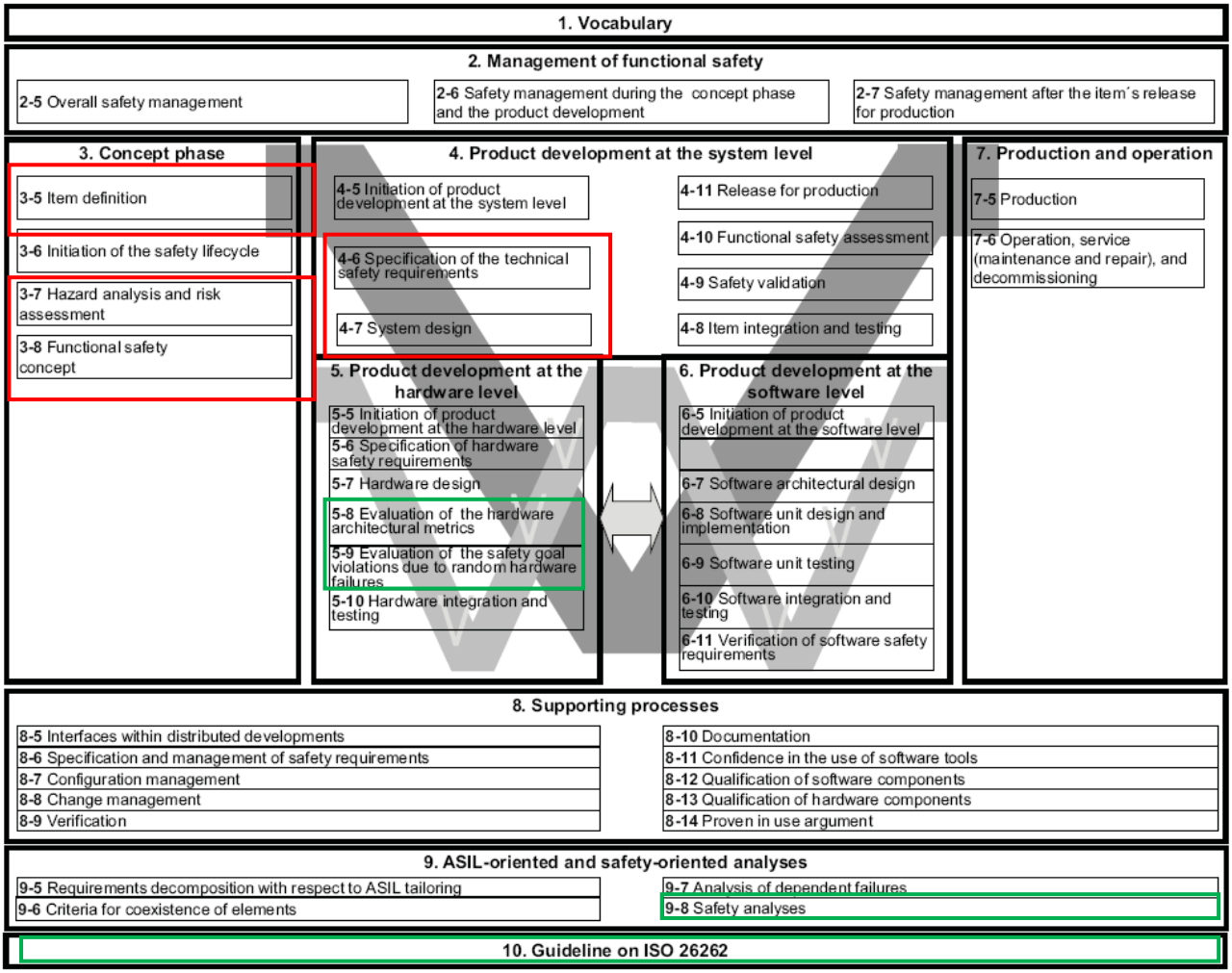


Figure 1: Overview of ISO 26262 [1] with highlighted relevant parts for work task WT3.3.3

7.1 Explanation of relevant ISO 26262 Parts

Part 3 "Concept phase":

During the concept phase for automotive systems, ISO 26262 Part 3 describes the definition of the item for development. Afterwards, requirements regarding the initiation of the safety lifecycle are presented. For the item, a hazard analysis and risk assessment has to be performed. This leads to the definition of the corresponding functional safety concept.

Part 4 "Product development at the system level":

ISO 26262 Part 4 specifies requirements and recommendations for the product development at the system level. Therefore, in a first step requirements for the initiation of the system level

product development are defined and technical safety requirements specified. The technical safety concept in context of the product has to be captured to start the design of the system. After system design, hardware and software as described in Parts 5 and 6 is developed. For the overall item integration and testing as well as safety validation and the assessment regarding functional safety for the complete system, Part 4 provides the corresponding requirements. Concluding, product release is mentioned.

Part 5 "Product development at the hardware level":

Requirements for the product development at the hardware level are specified in ISO 26262 Part 5. This part focuses on the definition of requirements for the initiation of product development at the hardware level, coming from system specification of Part 4. Afterwards, the specific hardware safety requirements are mentioned to start the initiation of the hardware design. For verification, requirements focusing on the evaluation of the hardware architectural metrics and the evaluation of violation of the safety goal due to random hardware failures and hardware integration and testing are specified.

Part 9 "ASIL-oriented and safety-oriented analyses":

ASIL-oriented and safety-oriented analysis is specified in ISO 26262 Part 9. This includes the definition of requirements decomposition with respect to ASIL tailoring as well as the definition of criteria for the coexistence of elements. Additionally, requirements regarding general safety analyses and the analysis of dependent failures for the system are described.

Part 10 "Guideline on ISO 26262":

Part 10 of ISO 26262 is intended to provide a guideline for the application of ISO 26262. Therefore it contains additional explanations and examples in order to facilitate comprehension of the concepts of ISO 26262. It has informative characteristic, therefore in case of inconvenience, the requirements, recommendations and information specified in the other parts of ISO 26262 apply. Part 10 was published in August 2012 after official release of the ISO 26262.

7.2 Relation of presented methodologies to relevant ISO 26262 Parts

Methodology 1 (green): Assessment of hardware on different level of abstraction

For the assessment of hardware, especially ISO 26262 Part 5 is in focus as it describes the product development at the hardware level. For different safety analyses and their interconnections, Part 9 Clause 8 contains additional information. The guideline of Part 10 provides application examples regarding different hardware safety evaluations.

Methodology 2 (red): Consistency checks for the safety case

The consistency checks and metrics defined in WT3.3.3 can be applied during item definition, hazard and risk analysis and the functional safety concept, which are related to ISO 26262 Part 3, and specification of the technical safety requirements and the system design, included in ISO 262626 Part 4. They are especially useful when iterating between these tasks.

8 Methodology 1: Assessment of hardware on different level of abstraction

8.1 Introduction

Novel functionalities and innovations, such as driver assistance systems, lead to growing technological complexity of electric and electronic systems for road vehicles. Achieving reliability as a key factor to ensure hazardless operation is therefore constantly becoming more and more challenging. To provide a common understanding and state-of-the-art for the automotive domain, the safety standard ISO26262 [1] was published in 2011 as an adaption of the IEC 61508 [2] for functional safety of road vehicles. It captures requirements and activities during the entire lifecycle of automotive safety-related electric or electronic systems.

Regarding the hardware architecture of automotive systems, a high reliability has to be achieved especially in context of random hardware failures. These failures occur unpredictably during the lifetime of electric systems due to exemplarily aging effects and can never be avoided. Therefore, during concept and development of the hardware architecture, evaluations have to be performed iteratively to ensure and document sufficient rationale.

8.2 Motivation

The mentioned complexity of automotive hardware architectures requires consideration of functional safety aspects already in early concept and development phase. In order to cope with the requirements for safety analysis and evaluation regarding hardware, ISO 26262 [1] Part 9 “Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analysis” Clause 8 “Safety analyses” claims the evaluation considering qualitative and quantitative aspects to determine faults and failures in the system and determine their impact. These analyses have to be supported at appropriate level of abstraction iteratively during concept and development. Different analysis methods according to ISO 26262 Part 9 are shown in Figure 2.

Quantitative Analysis	Qualitative Analysis
Qualitative FMEA	Quantitative FMEA
Qualitative FTA	Quantitative FTA
HAZOP	Quantitative ETA
Qualitative ETA	Markov Models
	Reliability Block Diagrams

Inductive analysis method
Deductive analysis method

Figure 2: Qualitative and quantitative analysis methods mentioned in [1]

Regarding hardware architectures, quantitative analysis complements qualitative analysis. Quantitative evaluation supports the verification of the hardware design against target values. The target values are described based on the ASIL-classification of the safety goal, which is determined based on hazard analysis and risk assessment (HARA). Being able to evaluate the robustness of the hardware architectures allows identification of potential hazardous designs. The quantitative evaluation is required for final verification of the hardware design especially in late development phases. Benefit from early introduction of safety evaluation and including target value verification would serve as an input for definition of safety measures to ensure reliable operation of the hardware architecture throughout system lifetime. Early determination of dangerous failures of the hardware could additionally reduce development costs and time.

8.2.1 Safety evaluation of hardware designs

Focusing on the different safety analyses and evaluations during product development of the hardware, ISO 26262 [1] Part 5 “Product development at the hardware level” describes quantitative evaluations which have to be performed for hardware designs to ensure robustness and facilitate evidence. The evaluations claimed are shown in Figure 3.

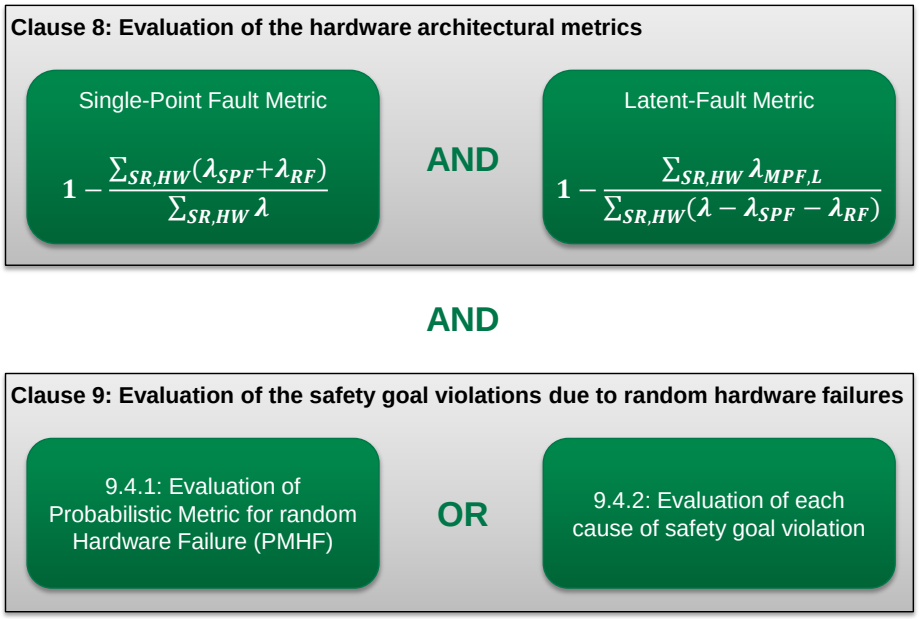


Figure 3: Hardware safety evaluation according to [1] Part 5

Clause 8 “Evaluation of the hardware architectural metrics” describes the assessment of hardware architectures by applying hardware architectural metrics. These metrics provide quantitative information on the robustness of the hardware architecture against critical random hardware failures. Critical in this context means, that these failures have direct or indirect influence on the behavior of the system regarding the violation of the specific safety goal. Therefore, two metrics are described: the single-point fault metric and the latent-fault metric where both have to be applied including verification against target values.

In Clause 9 “Evaluation of the safety goal violations due to random hardware failures” the residual risk of safety goal violation is evaluated. This assessment has to be done complementary to the hardware architectural metrics. ISO 26262 proposes two alternative methods: the Probabilistic Metric for random Hardware Failures (PMHF) or the evaluation of each cause of safety goal violation by using failure rate classes (FRC). Whereas PMHF represents a global approach describing the maximum probability for the violation of the safety goal, the failure rate class method considers an individual evaluation of each cause of safety goal violation.

8.2.2 Hardware design levels

For hardware designs, different levels of abstractions are present during different phases of concept of development. These are orthogonal to the described evaluations. The iterative application of these evaluations during different phases of hardware design has to be ensured in terms of functional safety.

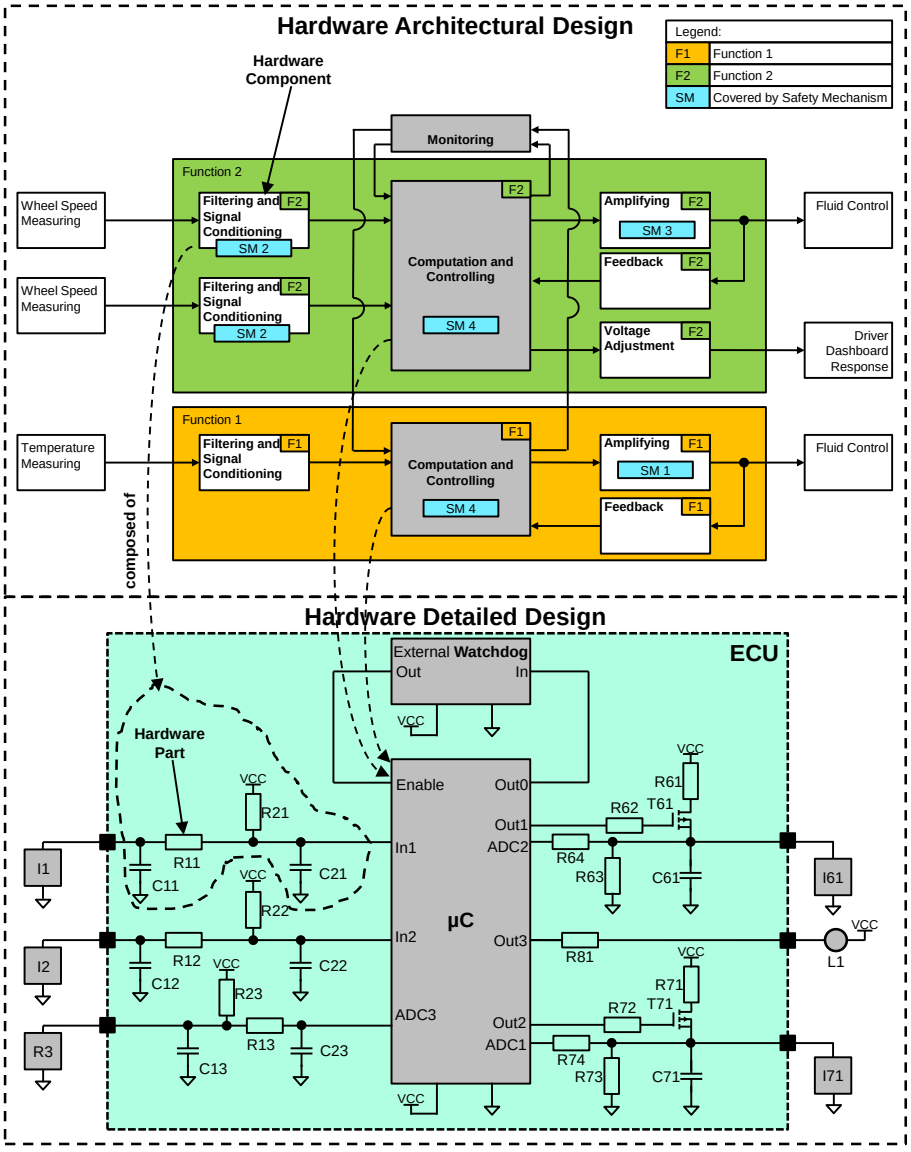


Figure 4: Exemplarily differentiation between hardware architectural design and hardware detailed design based on electronic schematic example for a valve control of ISO 26262 Part 5 Annex E [3]

For the description of hardware designs, two different development phases are established in [1] Part 5: the *hardware architectural design* and the *hardware detailed design*. The hardware architectural design focuses on the description of hardware in early concept and development phase according to [1] Part 5 7.4.1. It represents an initial view on the hardware capturing functionalities in corresponding hardware components. The hardware architectural design consists of the hardware components as black boxes and their interconnections. For a new development, hardware components exemplarily the filtering and signal conditioning could be re-used and taken from company specific libraries. At this level of development, no concrete realization in at the level of detailed schematic is established. An early safety evaluation on this level including verification

against target values would facilitate the introduction of additional or the improvement of existing safety mechanisms.

During development phase, these hardware components are refined at the level of hardware schematics in terms of interconnected hardware parts. This is specified by the hardware detailed design. Therefore, a hardware component can be composed of one or more hardware parts. For complex hardware parts such as a microcontroller, the hardware part can compose several hardware components. At the detailed level, the safety evaluation serves as verification for the final hardware design. Figure 1 shows our exemplary differentiation for hardware architectural and hardware detailed design based on the example schematic for a valve control provided by ISO 26262 [1] Part 5 Annex E.

Further information for Figure 4: The example schematic from ISO 26262 Part 5 Annex E is described at the level of electronic schematics consisting of different hardware parts. To explain the presented concept of safety evaluation at different level of abstraction for hardware design, the corresponding hardware architectural design was rebuild [3]. The example consists of two different functions, which are implemented on a single ECU. Function one has the temperature as input, measured via R3, and controls the valve I71. The corresponding safety goal is “valve 2 shall not be closed for longer than *x* ms when the temperature is higher than 100°C”. Function two has the wheel speed measured redundantly via I1 and I2 as input and controls the valve I61. The safety goal defined for this function is “valve 1 shall not be closed for longer than *y* ms when the speed is higher than 100 km/h”.

8.3 Basics and related work

The next subsections describe basic information for hardware element failure and reliability especially in context of random hardware failures to facilitate a common understanding. Additionally, related work and research activities in the context of functional safety regarding hardware are briefly presented.

8.3.1 Basics for hardware safety evaluation

Quantitative evaluation of hardware architectures requires knowledge about statistical data of hardware elements such as failure rates and failure modes. Constructs for hardware safety evaluation are presented in the IEC 61508 “Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems” [2]. The IEC 61508 was published in 1998, the current revised second version in 2010. IEC 61508 defines two different types of hardware elements. Type A describes an element which entire failure description and behavior is well-known, exemplarily resistors. Type B describes a hardware element, for which the entire failure behavior and description is not well-known, exemplarily microcontrollers or abstract hardware entities [2]. Due to the fact that ISO 26262 was adapted from the IEC 61508 for the automotive domain, a lot of subjects regarding hardware evaluation are related. In the following, main topics of functional safety and reliability regarding hardware in context of ISO 26262 are described.

Failure Rate:

The reliability of a hardware element is described by its failure rate λ. It represents the amount of failures of the element in a certain time. The failure rate is dependent on time, λ(t), and its characteristics can be described using a bathtub-curve according to Figure 5.

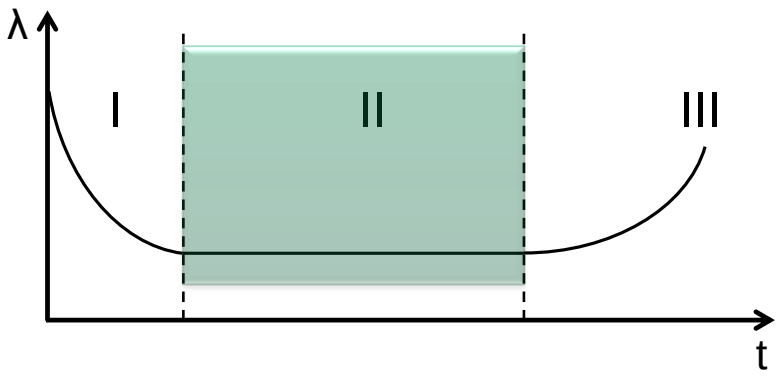


Figure 5: Failure Rate over time [4]

The first phase (I) describes a decreasing failure rate due the fact that the amount of early failures is higher. The second phase (II), seen as the lifetime of the element, describes a constant failure rate, whereas in phase (III), the failure rate increases because of the element's aging [4]. In context of the presented concept for hardware safety evaluation, the failure rate is assumed to be constant. When a constant failure rate is applied, it can be expressed using the unit Failure-In-Time (FIT). 1 FIT is one failure in 10^9 h. In order to avoid bias during quantitative evaluation, a scaling factor should be applied, when considering failure rates from different sources.

Failure Mode:

The behavior of a hardware element in case of a failure is represented by different failure modes. A single hardware element can have several failure modes. For instance, a resistor typically has the failure modes "Open Circuit" and "Short Circuit". A failure rate distribution indicates which part of the overall failure rate of the hardware element is distributed to the specific failure mode. The failure rate distribution is a percental value between 0 and 100%.

According to ISO 26262 [1], hardware failure rates, failure modes and their failure rate distributions can be determined using commonly recognized industry sources, statistic data or expert judgment. ISO 26262 exemplarily lists IEC Standard TR 62380 [5], MIL-HDBK-338B [6] or UTE C 80 811 [7] as sources to determine the corresponding values. Additionally, documents like Siemens Handbook SN 29500 can be used.

Failure Mode Classification:

To describe the impact of a specific failure mode of a hardware element to the system behavior, the failure mode of the hardware element has to be classified using the specific fault types. ISO 26262 proposes a classification as shown in Figure 6.

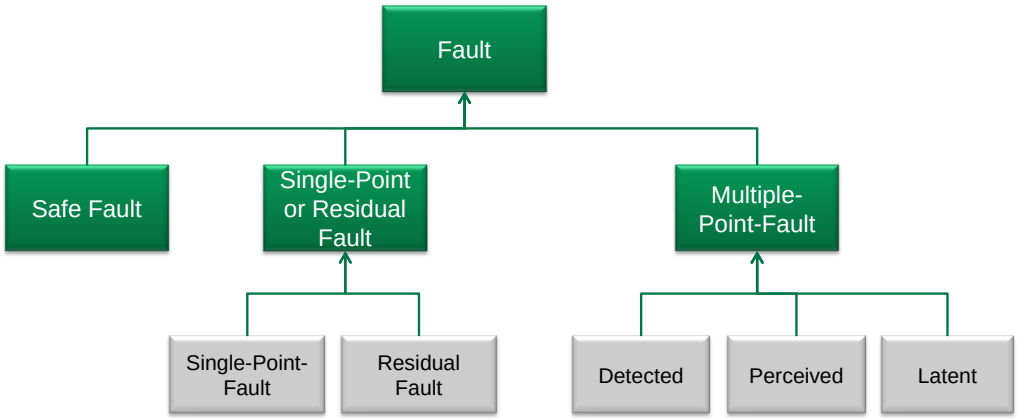


Figure 6: Classification of faults according to ISO 26262 [1]

In a first step, a fault of a hardware element can be classified into safe fault, single-point or residual fault or multiple-point fault. A safe fault has potential to violate the safety goal and therefore represents a fault of order zero. Single-point or residual fault indicates a direct violation of the safety goal as a fault of first order. The classification as multiple-point fault indicates a potential violation of the safety goal in combination with another independent fault of order n, while n is equal or larger than 2.

Going into detail, a direct violation can either be classified as single-point fault, which is not covered by a safety mechanism. A residual fault represents a direct violation that describes the part of the failure, which is not covered by a safety mechanism. For multiple-point faults, the fault can be *detected* by a safety mechanism, *perceived* by the driver through system behavior or *latent*, which is the part of the fault which is not *detected* or *perceived*. Latent multiple-point faults are also abbreviated to latent faults [1] Part 1, 1.71. ISO 26262 Part 1, 1.101 suggests consideration of multiple-point faults up to an order of 2, named dual-point faults.

In comparison, IEC 61508 classifies faults only into safe fault and dangerous faults. Afterwards they can be differentiated into detected or undetected, depending on the coverage of a safety mechanism. Recapitulatory, ISO 26262 provides a finer classification of faults.

Safety Mechanism:

For hardware elements, a safety mechanism describes a ”technical implementation by E/E function” [1], which prevents single-point faults, reduces residual faults and prevents multiple-point faults from being latent. The effectiveness of a safety mechanism is presented with two diagnostic coverages K_{DC}. The diagnostic coverage $K_{DC,RF}$ with respect to residual faults describes the effectiveness of the safety mechanism regarding direct violations of the safety goal. The diagnostic coverage $K_{DC,MPF,L}$ addresses the effectiveness regarding multiple-point faults.

Safety Goal:

A top level safety requirement which is present for a system can be described as a safety goal. Similar to the Safety Integrity Level (SIL) classification provided by the IEC 61508, the ISO 26262 defines Automotive Safety Integrity Levels (ASIL) to describe the safety-relation of a specific safety goal. The ASIL is determined by the parameters “controllability”, “severity” and “probability of exposure” of the event. ISO 26262 proposes a classification from ASIL-A to ASIL-D with ASIL-D being the most stringent. QM as an additional value requires no specific activities related to safety requirements of ISO 26262 [1] Part 3 7.4.4.1.

Safety Analysis:

The most common safety analysis methods, described in ISO 26262 as shown in Figure 2, are briefly described. The fault tree analysis (FTA) represents a graphical failure analysis technique [8] [9]. Quantitative FTA can be achieved by adding probability values to the events. FTA is a well-established practice for safety analysis in a lot of industry domains, exemplarily avionic [10] or nuclear and represents a deductive methodology.

The failure mode effects analysis (FMEA) focuses on a structured qualitative analysis of the elements of a system to identify potential failure modes and their effects on system behavior. It was initially formalized in [11]. The failure mode, effects and criticality analysis (FMECA) in addition to classical FMEAs addresses quantitative issues by adding information about severity and probability of failures using a criticality metric [12]. The failure mode, effects and diagnostic analysis (FMEDA) considers additional quantitative failure data and diagnostic coverage, exemplarily of safety mechanisms. FMEDA was introduced based on the paper of [13], the name was first used in [14]. FMEDA methodology has been refined especially in IEC 61508 to determine additional quantitative results such as the safe failure fraction. FMEDA is a common practice for safety analysis of electric/electronic systems [12].

Other analyses mentioned in ISO 26262 are reliability block diagrams (RBD) and markov models.

8.3.2 Related work

In the technical report [15] of Leitner-Fischer et al., the different parts of the ISO 26262 are presented in order to analyses them regarding compatibility with their QuantUM method and tool for the support of requirements demanded by the standard. For Part 5 “Product development at the hardware level” it is mentioned that hardware evaluation could be supported, but their “UML-based approach is not really suitable to describe hardware architectures” [15].

In the paper of Jeon et al. [16], they focus on hardware especially with classification of ASIL C or D. The process from specification of safety requirements, design, up to the integration and testing in context of hardware is described. This process is in accordance with ISO 26262. Additionally, the demanded metrics of Part 5 are explained high level.

In the article [17] of Bellotti and Mariani, hardware evaluation is mentioned in context of microcontroller design. This concerns functional safety and ISO 26262, as functional safety requirements have impact on the design of microcontrollers and microprocessors. In the article of Sinha [18] the functional analysis of a proposed brake-by-wire system is described in context of safety.

Svancara [19] [20] gives a detailed description of ISO 26262 [1] Part 5 Clause 9 regarding evaluation of residual risk of safety goal violation using the second method with failure rate classes shown with an electrical powered steering use case. The methodology is demonstrated into detail, but no implementation or integration into model-based environments is considered.

Regarding the modeling and propagation of failures, the approach of HiP-HOPS has to be mentioned [21] [22]. A graphical approach for a description of failure propagation in an integrated architecture description language is presented in [23].

8.4 Preparation for model-based hardware evaluation

8.4.1 Interface to hardware description

Focus of work task WT3.2.2 "*Hardware description"* is a proposal for meta model adaption based on EAST-ADL regarding structural description of hardware and an extension regarding hardware failure data [26]. The meta model constructs serve as an input for the SAFE meta model provided by WT3.5. The meta model constructs form the basis for the methodologies for evaluation presented in this deliverable.

A brief excerpt of the meta model extension for hardware failure description is shown in Figure 7.

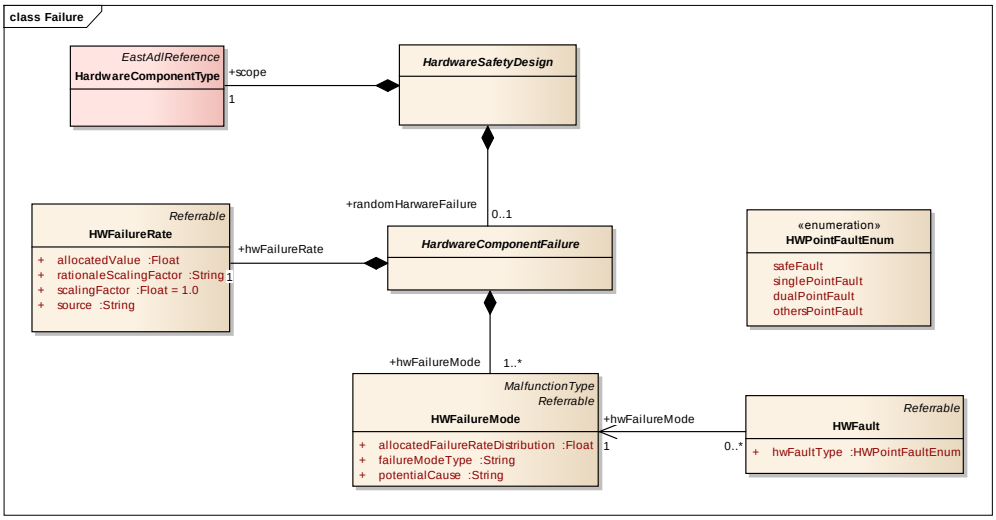


Figure 7: Excerpt of SAFE meta model regarding hardware failure [26]

The extension for hardware failure description is based on the EAST-ADL references to *"HardwareComponentType"* and *"HardwareComponentPrototype"*. This construct allows the composition of several hardware component prototypes in a new hardware component type. For the concrete hardware component type, a hardware component failure extension is defined, which captures all relevant failure information. The hardware failure extension covers the following classes:

- Class *"HWFailureMode"* for the description of hardware failure modes. These failure modes cover a failure rate distribution, the failure mode type (exemplarily *"ShortCircuit"*) and a potential cause (e.g. overheating).
- Class *"HWFailureRate"* is used for the annotation of a failure rate values. Additionally, scaling factors according to [1] Part 5 including a rationale and the source of the failure rate can be described.
- Class "*HWFault"* is linked to a hardware failure mode and defines the classification of the fault in context of a specific safety goal. For the classification, the enumeration *"HWFaultEnum"* is provides the corresponding literals.
- Class *"HWSafetyMechanism"* is an abstract representation of a safety mechanism which is covering a specific hardware failure mode. No concrete realization of safety mechanism is presented for the meta model approach.

8.4.2 Continuous hardware modeling and evaluation model concept

For the assessment of hardware designs in context of random hardware failures, a model-based description is required as a basic prerequisite. This includes a structural description of the hardware design as well as annotated failure information, links to the related safety goals including target values and safety mechanisms with their diagnostic coverage. The SAFE meta model provides all the different perspectives on a safety-related system in terms of functional safety. This includes topics such as COTS, safety case, software modeling etc. The SAFE meta model contributions are currently in synchronization and alignment in terms of integration issues. Additionally, EAST-ADL and AUTOSAR are connected providing meta model constructs for specific modeling purpose as reference points.

Focusing on model-based hardware safety evaluation, a meta model with a minimal set of constructs allows a clear structuring and description of the methodology within this deliverable. Therefore, specific parts of the SAFE meta model were extracted in terms of model-based hardware safety evaluation. This leads to a proposed minimal combination of meta model classes for model-based hardware quantitative and qualitative assessment, as shown in Figure 8.

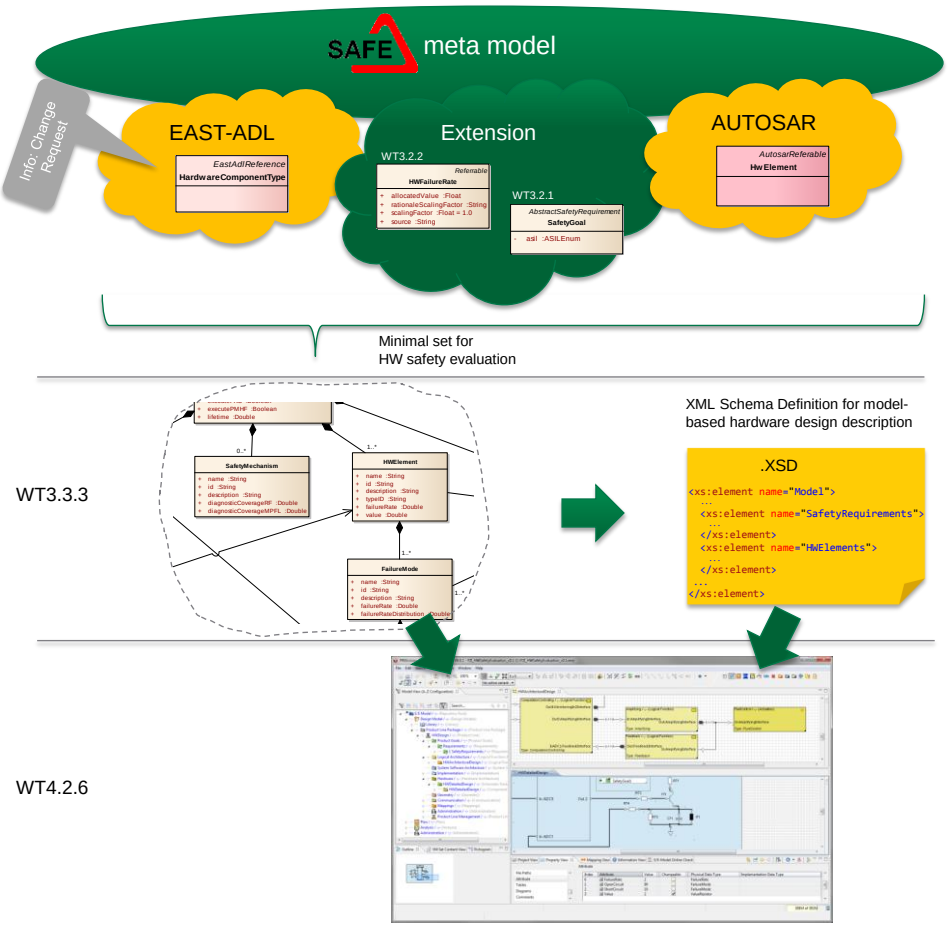


Figure 8: Model-based hardware safety evaluation

8.4.3 Minimal meta model for hardware safety evaluation

The complete minimal meta model for integrated hardware safety evaluation to support assessment is provided in Figure 9. The meta model classes including their attributes contained are explained briefly in the following.

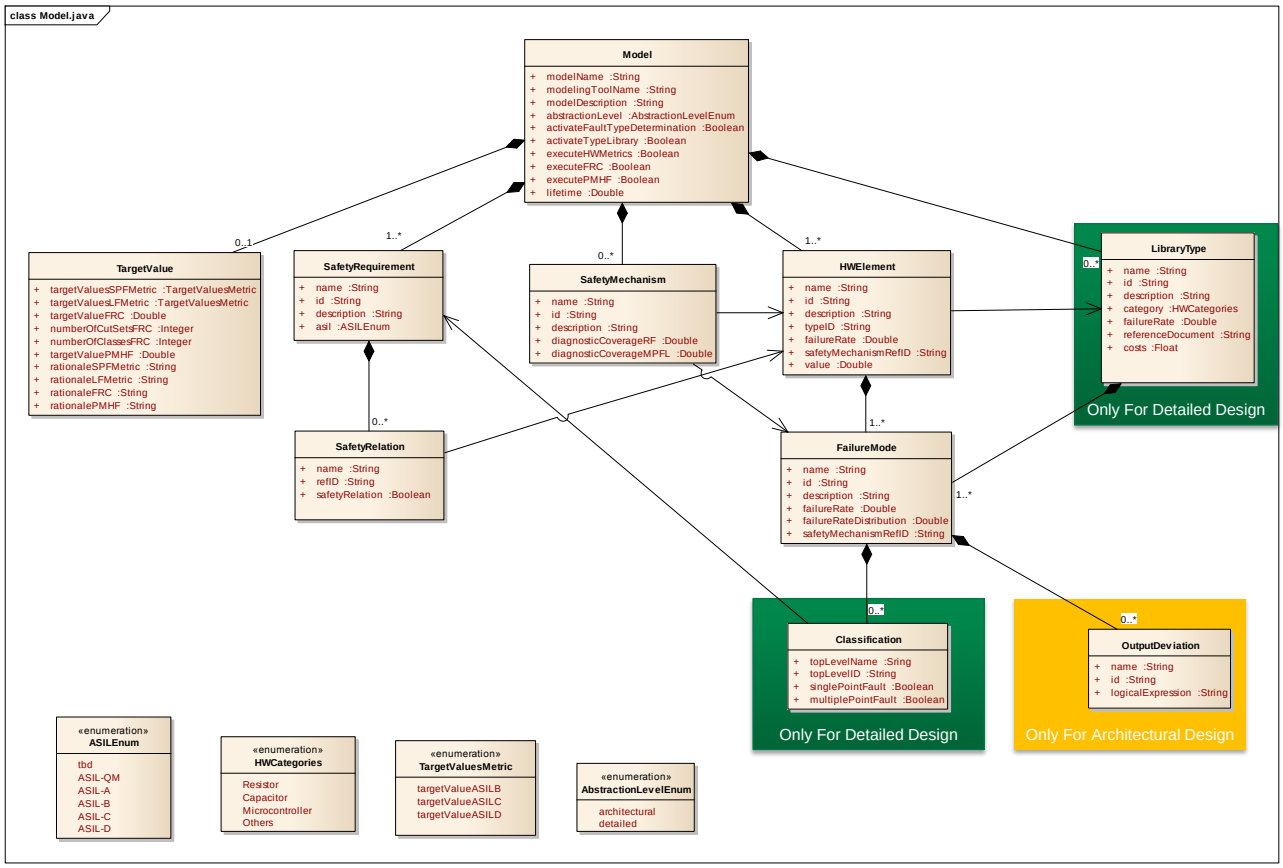


Figure 9: Minimal proposed meta model for model-based hardware safety evaluation

8.5 Concept for model-based hardware safety evaluation on different abstraction levels

As introduced, ISO 26262 [1] Part 9 Clause 8 claims the evaluation of hardware designs on appropriate level of abstraction whereas ISO 26262 [1] Part 5 Clause 7 describes two different levels of abstraction during hardware concept and development phase. For these two orthogonal levels, we provide a concept for model-based hardware safety evaluation as shown in Figure 10.

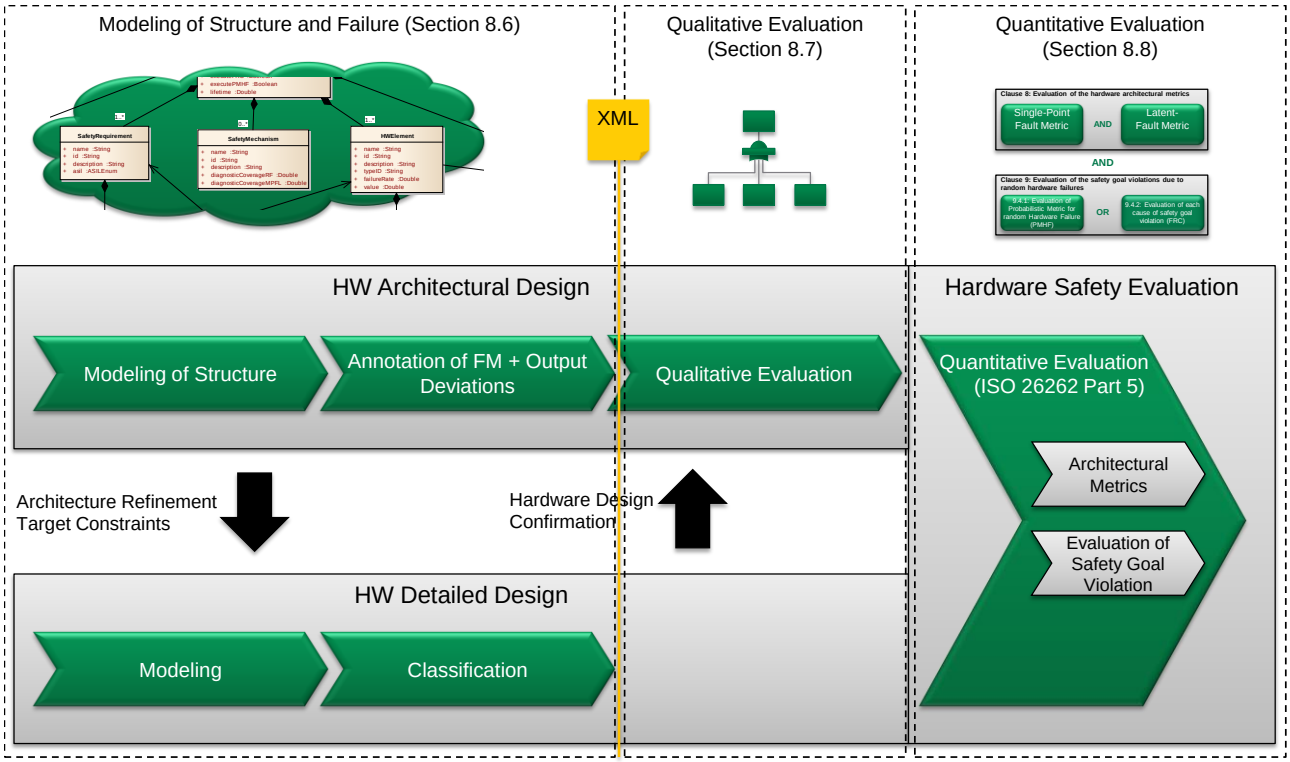


Figure 10: Overview: Concept for hardware safety evaluation and abstraction levels

On higher level of abstraction, the hardware architectural design is established representing an initial view on the hardware design. For this level of abstraction, a deductive methodology for model-based hardware safety evaluation is proposed. Annotation of failure modes and their propagation through the system can be evaluated using a fault tree analysis, see also the HiP-HOPS approach [21]. This qualitative analysis supports the quantitative evaluation with the determination of the impact of failure modes to the top-level event. The top-level event in this context represents the violation of a specific safety requirement. The quantitative evaluations represent an initial safety evaluation, based on assumptions for failure rates and failure modes. The initial evaluation facilitates early definition of safety mechanisms or identification of safety-critical sections of the architecture. The initial hardware design can be iteratively reworked and improved. This is supported by the verification of results against target values.

During subsequent phase of development, the hardware architectural components are specified by the hardware detailed design at the level of electronic schematics. The hardware detailed design is described by concrete hardware parts. The hardware detailed design can also be evaluated regarding functional safety [27]. This evaluation describes the final verification of the hardware design and therefore confirms the initial hardware safety evaluation.

8.6 Structural and failure modeling of hardware designs

This chapter explains the concept for structural modeling and failure information of hardware designs. The two different abstraction layers are presented and links to the meta model are provided.

8.6.1 Modeling of hardware structure

For structural modeling of the hardware, the class *HWElement* is in focus. This class captures all relevant information for a hardware component regarding the hardware architectural design or a hardware part in context of hardware detailed design.

8.6.1.1 Hardware architectural design

Due to the complexity, the concept and development phase regarding hardware is not exclusively done at the level of detailed schematic. Therefore, in a first step, the hardware design is modeled at functional block level. Hardware components which capture certain functionalities are defined as abstract blocks with in- and outputs. These blocks are defined in context of the technical safety concept derived from the system design specification, according to ISO 26262 Part 4 Clause 7. The hardware components can be taken from previous designs or described in a company-specific library. Therefore, a re-use of certain parts is facilitated. Regarding the meta model for hardware safety evaluation, the hardware component represents a specialization of the class *HWElement*, which is not separately introduced in order to provide generic information. An example for a hardware architectural design which is continuously refined in this deliverable is shown in Figure 11.

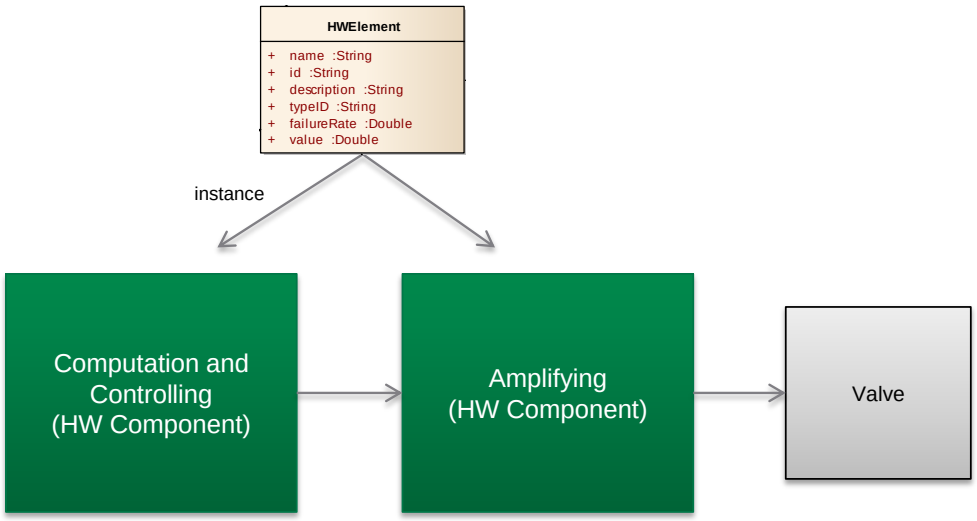


Figure 11: Hardware architectural design structural modeling example

Here, an example containing two hardware components *ComputationAndControlling* and *Amplifying* is given based on the output stage of function one of the example from ISO 26262 Part 5 Annex E, as shown in Figure 4. For the hardware components, input and output ports are provided including the direction of the connection. These two hardware components serve as an example for the complete process of safety evaluation regarding the hardware architectural design.

8.6.1.2 Hardware detailed design

The hardware detailed design describes the hardware architecture on a detailed level of abstraction with hardware parts such as resistors or capacitors. Before modeling the hardware detailed design, a type library containing all relevant hardware part types that are used in the model can be defined. As the hardware detailed design contains a high amount of hardware parts, the type library concept fits the corresponding needs. The meta model class *LibraryType* forms the basis.

The type library contains all hardware parts which can be instantiated in the schematic. Additionally, all failure data regarding failure modes and failure rates are deposited in the library. The type library can also be used to express different technologies which are used for the same type of hardware part such as metal film resistors or surface mount device resistor. A consistent example for this deliverable for modeling of hardware detailed designs using the type library is shown in Figure 12.

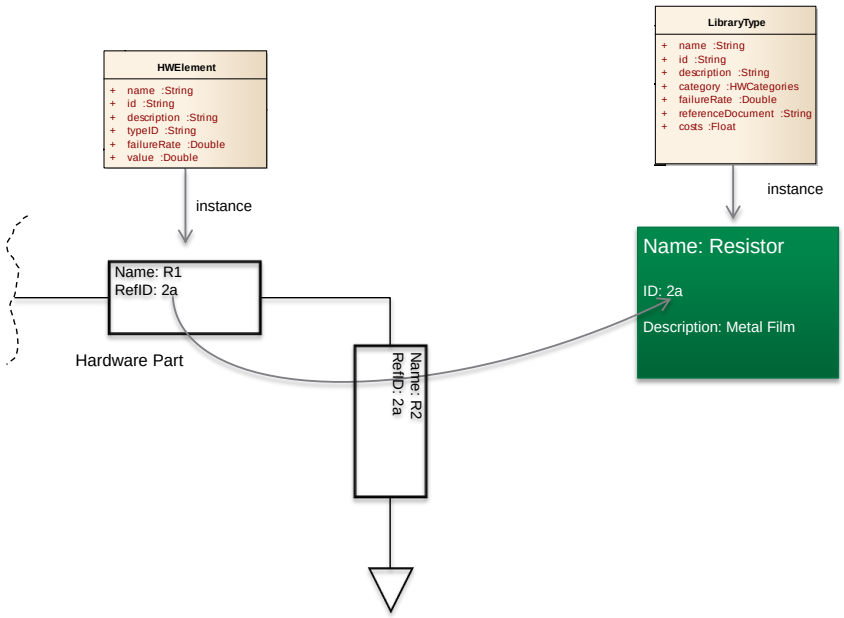


Figure 12: Hardware detailed design structural modeling example

Here, the hardware parts *R1* and *R2* are instantiated as hardware elements for the hardware detailed design. Both of these hardware parts are referencing to the same hardware library type *Resistor* which exemplarily represents a metal-film resistor type. The reference is achieved using the attribute *refID* of the hardware element, which contains the *id* of a hardware library type.

Using the type library concept, the hardware parts and their interconnections have to be modeled. The corresponding schematic contains all hardware parts with their interconnections. Additional attributes such as the value of exemplarily resistors or capacitors have to be captured for each instantiated part, due to the fact that they are not equal for all instantiated hardware parts of the same hardware part type.

8.6.2 Modeling of hardware failure

In order to provide hardware safety evaluation according to ISO 26262 Part 5, the structural description of hardware was extended to provide information about failure data.

8.6.2.1 Modeling of safety requirements

The initiation of the product development at the hardware level requires consideration of hardware safety requirements and the overall system safety concept. Safety requirements for the hardware design have to be defined and annotated to the model. Regarding this model-based approach, the model-based definition of safety-related hardware elements using the class *SafetyRelation* is supported. An example is shown in Figure 13.

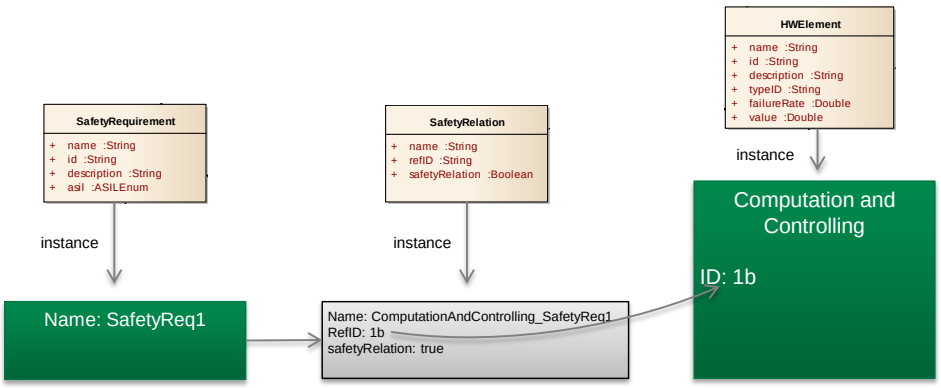


Figure 13: Safety-relations of hardware elements to safety requirements

Here, the safety requirement *SafetyReq1* contains a safety-relation named *ComputationAndControlling_SafetyReq1*. In this safety-relation, the *refID* references the specific hardware element. The attribute *safetyRelation* specifies whether the hardware element is safety-related or not for the safety requirement using a Boolean attribute type.

8.6.2.2 Modeling of safety mechanism

For the quantitative evaluation of the hardware design, failure mode coverage due to safety mechanism has to be considered. Therefore, safety mechanisms are modeled using the corresponding class of the meta model. The safety mechanism refers to a specific hardware element or failure mode it covers. This reference is achieved due to the attribute *safetyMechanismRefID* which is contained in the classes *HWElement* and *FailureMode*. Therefore, either the coverage of safety mechanisms for a specific failure mode or the coverage for the overall hardware element can be expressed.

8.6.2.3 Hardware architectural design

For each hardware component, certain failure modes have to be defined. The failure modes represent assumptions or expert knowledge exemplarily from previous designs. According to the multiplicity in the meta model, for each hardware element one or more failure modes can be annotated. A failure rate for the each failure mode can be defined as an attribute. An exemplary annotation of failure modes for hardware components is shown in Figure 14.

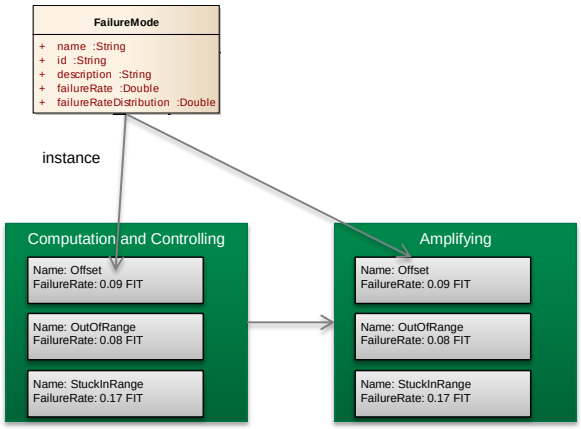


Figure 14: Annotation of failure information for hardware architectural design

Here, the two hardware components *ComputationAndControlling* and *Amplifying* both have the three failure modes exemplarily *Offset*, *OutOfRange* and *StuckInRange* annotated. For each of the failure modes, a specific failure rate in FIT is defined, which is an assumption as mentioned above.

For the hardware architectural design, the consequences of random failures of the hardware components for the system behavior can be determined with a deductive analysis exemplarily a fault tree analysis. A model-based approach for the propagation of failure modes to a system top-level failure is provided to facilitate quantitative evaluation according to ISO 26262. This is facilitated based on output deviations for each hardware component which describe a logical failure expression, related to the approach of HiP-HOPS [21].

One logical expression for the system could describe the overall fault tree. Due to complexity issues, a separation of single output deviations and the annotation to the boundaries of the hardware component is performed. An exemplarily output deviation regarding the hardware architectural design is shown in Figure 15.

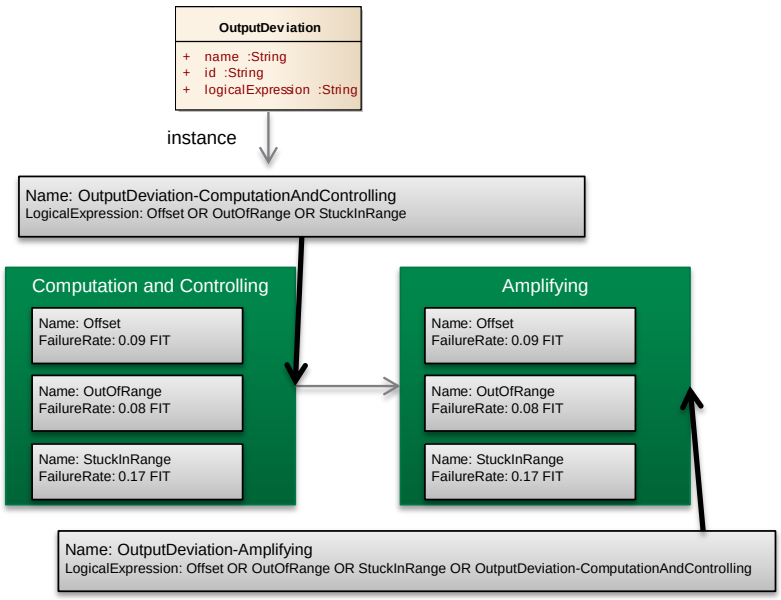


Figure 15: Annotation of output deviations for hardware architectural design

In this example, each of the hardware components has one output deviation, which represents an OR combination of all failure modes contained. Additionally, for the hardware component *Amplifying*, the preceding output deviation of *ComputationAndControlling* is included in the logical failure expression. The logical expressions of the output deviations are represented as a *String* value.

8.6.2.4 Hardware detailed design

Subsequent to structural modeling of the hardware detailed design, the failure data in terms of failure modes and failure rates has to be annotated to the model. This failure data of standard hardware parts is well-established. Therefore, they are not representing assumptions at this phase of development. The estimated failure data in terms of failure rates and failure modes can be taken from recognized industry source, statistics based on field tests or expert judgment, as described in Section 8.3.1. An annotation of failure information for hardware detailed design is shown in Figure 16.

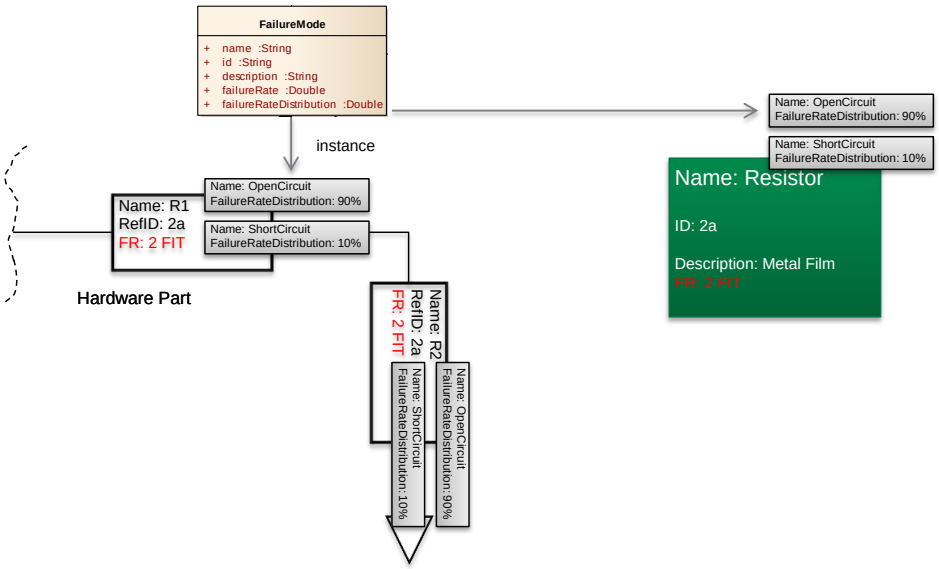


Figure 16: Annotation of failure information for hardware detailed design

For the hardware detailed design, the failure rate is annotated to the hardware element itself (shown in red letters). For each failure mode, a failure rate distribution is described in percentage, which specifies the portion of the overall failure rate of the hardware element which is distributed to the failure mode. The failure rate of the failure mode (which is recommended to be defined for the hardware architectural design) can be calculated according to Equation 1.

$$\lambda_{FM} = \lambda_{HWElement} \cdot FRD_{FM} \qquad \textbf{Equation 1}$$

The failure rate of the hardware element represents the sum of all failure modes specific failure rates.

$$\lambda_{HWElement} = \sum \lambda_{FM} \qquad \textbf{Equation 2}$$

Using these two equations, the failure rates specified for the failure modes at architectural level can be converted to the failure rate of the overall hardware element and a failure rate distribution for the failure mode, as used for the hardware detailed design. Therefore, in the model, either a failure rate for the failure mode (recommended for architectural design) or a failure rate for the hardware element and a failure rate distribution for the failure mode (recommended for detailed design) must be present. For the hardware detailed design, it is also possible to provide the failure information by annotation to the hardware library type. If specific failure information for a hardware element instance is provided, the failure information is not taken from the hardware library type.

In contrast to the hardware architectural design, for the hardware detailed design no failure propagation approach is facilitated due to the large number of parts and growing complexity. As

output deviations for hardware parts are not defined, a qualitative analysis to determine the classification of failure modes is not facilitated. This classification has to be manually specified for each failure mode. The flow diagram for such a failure mode classification to derive the safety goal violation of hardware part failure modes is described in Figure 17.

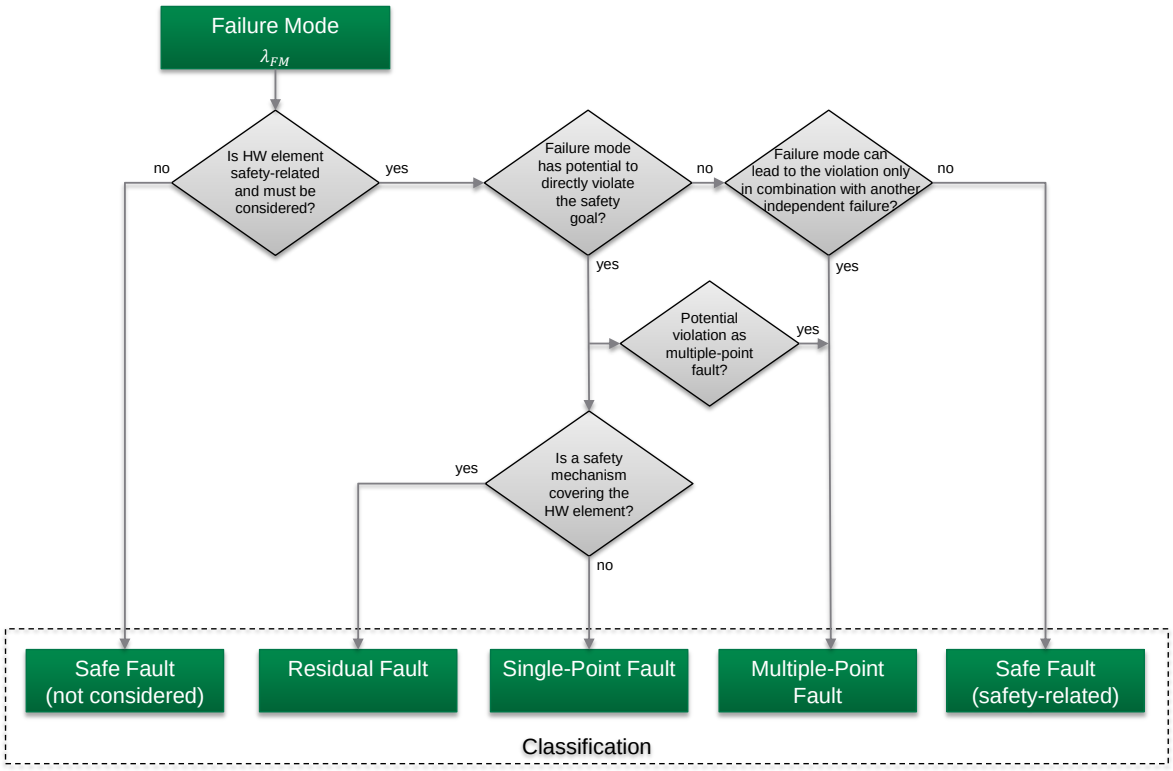


Figure 17: Simplified flow diagram of [1] for manual determination of classification

For each failure mode, the corresponding hardware element is either safety-related or not. If not, the failure mode is classified as a safe fault which has not to be considered in the analysis. For safety-related hardware elements, if the failure mode directly violates the safety goal it can be covered by a safety mechanism and classified as a residual fault or classified as a single-point fault which is not covered. If the failure mode has the potential for violation in combination with another independent fault, it is classified as a multiple-point fault. A classification as multiple-point fault and a direct violation is possible. If the failure mode does not violate the safety goal in any case, it is classified as a safe fault, which is safety-related and considered in the quantitative evaluation.

The classification of a failure mode can be different for different related safety requirements. The annotation is supported by using the class *Classification* which has the multiplicity 0..* for each failure mode and is used in context of a safety requirement. Therefore, it contains the attributes *topLevelName* and *topLevelID*, which refer to the specific safety requirement for which the classification is present. The classification additionally has the attributes *singlePointFault* and *multiplePointFault*, which are both of type Boolean.

Failure Mode Classification	Value Attribute *singlePointFault*	Value Attribute *multiplePointFault*
Safe Fault	False	False
Single-Point Fault	True	False
Residual Fault (SM present)	True	False
Multiple-Point Fault	False	True
Single-Point Fault and Multiple-Point Fault	True	True

Table 1: Derivation of *Classification* attributes values

The specific failure mode is classified as a Safe Fault, Single-Point Fault, Residual Fault or Multiple-Point Fault, according to Section 8.3.1. The value for the model attributes can now be derived as shown in Table 1. For the case that a failure mode is directly violating the safety goal and additionally in combination with another independent fault, both values are true. Default value for both attributes is false which relates to the classification as safe fault. The annotated classification for the hardware detailed design is exemplarily shown in Figure 18.

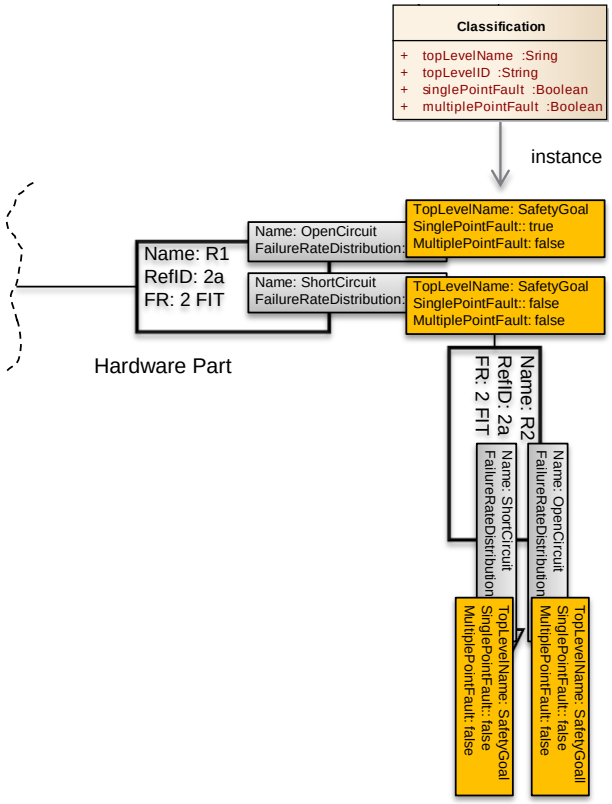


Figure 18: Annotation of Classifications for hardware detailed design

All annotated classifications are done in context of the top-level event *SafetyGoal*. For this example, both failure modes *OpenCircuit* directly lead to the violation of the specific safety goal, whereas the failure modes *ShortCirtuit* are represented as safe faults. The annotated classifications as well as the failure information and hardware modeling directly support a quantitative analysis for the hardware detailed design.

8.7 Qualitative evaluation

After output deviations are defined for the hardware architectural design, a qualitative analysis follows as an intermediate phase to identify component failure modes which lead to the system failure as violation of the safety goal. This qualitative analysis supports a quantitative evaluation by the automated determination of classifications.

8.7.1 Fault tree generation

In a first step, each output deviation of a hardware component is translated into a single fault tree. For the output deviations provided in the example, the corresponding single fault trees are shown in Figure 19.

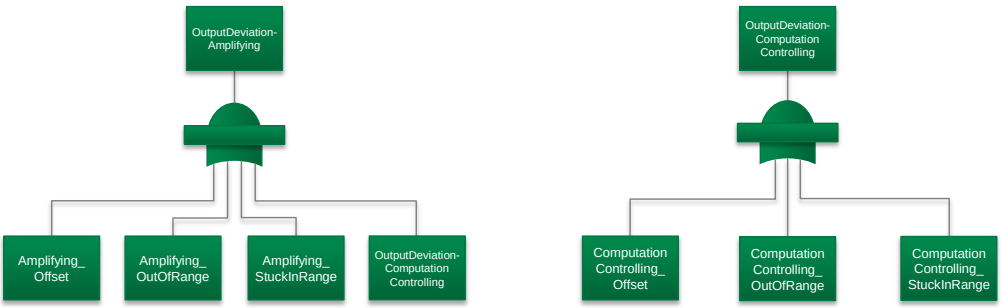


Figure 19: Exemplary single fault tree of output deviations

After the generation of all relevant single fault trees, starting from the top-level event as violation of a safety goal, the overall fault tree can be composed. The output deviations related to the top-level event contain their preceding output deviations which facilitates the composition of the overall fault tree. Regarding the example, *OutputDeviation-Amplifying* contains the output deviation *OutputDeviation_ComputationControlling*. These two fault trees can now be combined which is shown in Figure 20.

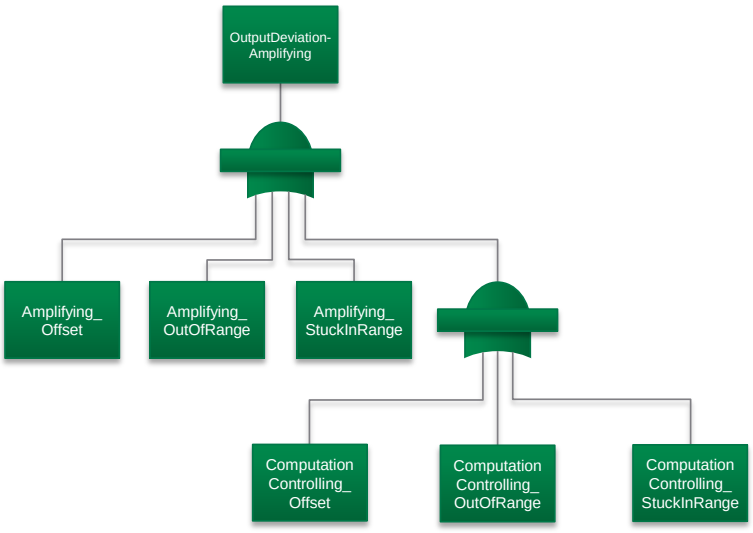


Figure 20: Complete fault tree for top-level OutputDeviation-Amplifying

8.7.2 Analysis and failure mode classification

The fault trees for the tagged top-level events can afterwards be analyzed in order to determine the contribution of the contained component failure modes to the top-level event. Therefore, a minimal cut-set analysis is facilitated. The minimal cut-sets with contained failure modes as basic events are determined. The order of the corresponding cut-set provides a classification of the failure mode into the fault classes provided by ISO 26262 Part 5 7.4.3.2. The determination is related to ISO 26262 Part 10 and captured in the following Figure 21.

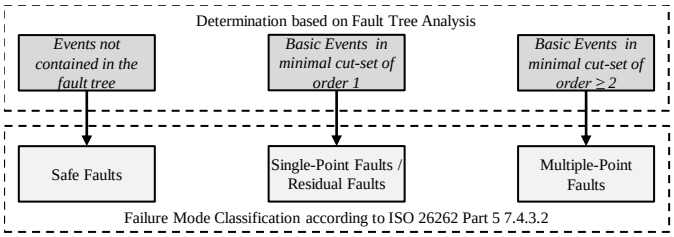


Figure 21: Derivation of failure mode classifications based on qualitative fault tree analysis [3]

If the failure mode is not contained in the fault tree, it is classified as a safe fault due to the fact that it has on impact on the safety goal violation. If the failure mode is contained in a minimal cut-set of order one in the fault tree, it is classified as a single-point or residual fault. These faults directly lead to the violation of the safety goal. If the failure mode is contained in a minimal cut-set of order greater than one, it is classified as a multiple-point fault. These faults lead to the violation of the safety goal only in combination with another independent failure mode.

Regarding the classification using fault tree analysis, ISO 26262 Part 10 Annex B.3.1 provides modeling examples. For classification as a single-point fault, the event is affiliated with the top-level event as the violation of the safety goal only via OR-gates and therefore in a cut-set of order one. The representation in a fault tree is exemplarily shown in Figure 22.

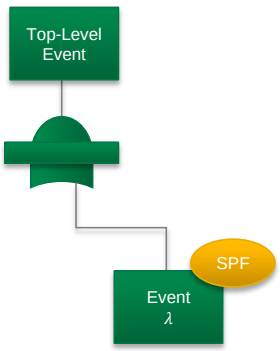


Figure 22: Classification: Single-point fault in fault tree

For residual faults, several ways of modeling within a fault tree are possible. Either the residual fault is represented as a single event which contains the coverage of a related safety mechanism by the specific failure rate calculation. This is shown on the left hand side of Figure 23. In this case, the residual fault in a fault tree is modeled in the same way as a single-point fault, only the probability of the event is different due to additional safety mechanism coverage. The second possibility would be, to model the safety mechanism as an extra event. This is shown on the right hand side of Figure 23. Here, the residual fault in the fault tree is represented as an AND-combination of the overall fault and the percentage not covered by the safety mechanism and would be represented in a cut-set of order two.

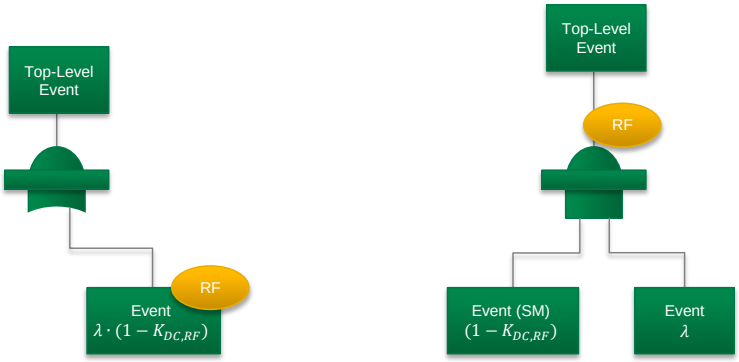


Figure 23: Classification: Residual fault in fault tree

In this model-based methodology, the approach without modeling of the safety mechanism as an extra event is preferred. This facilitates the analysis of residual faults which can be derived from a cut-set order of one – same as a single-point fault – due to their direct violation of the safety goal. The safety mechanism is independently tagged for the failure mode for specific failure rate determination. This is also considered for the representation of multiple-point faults.

The safety-relation is determined at hardware element level, not at the level of failure modes according to ISO 26262 Part 5 Annex E. In context of the approach for hardware architectural design, using the fault tree qualitative analysis, the approach shown in Figure 24 is used.

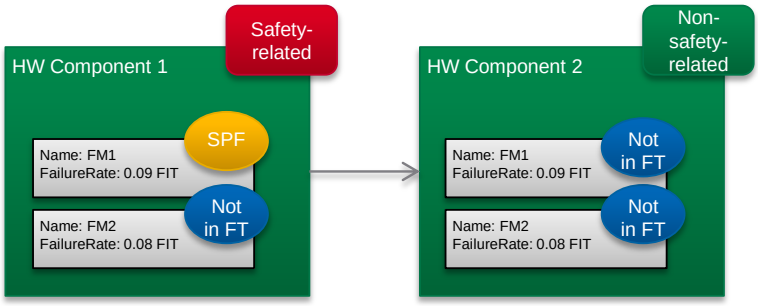


Figure 24: Determination of safety-relation for hardware components

If the hardware component under consideration has at least one failure mode contained in the corresponding fault tree for the violation of the safety goal, it is classified as safety-related. Exemplarily, FM1 of HW Component 1 is classified as a single-point fault, the HW Component 1 is safety-related. FM2 in this case is a safe fault which contributes to the overall safety-related failure rate. If the hardware component has no failure mode which is contained in the fault tree, it is classified as non-safety-related. Therefore, FM1 and FM2 of HW Component 2 do not contribute to the overall safety-related failure rate. However, the determination of safety-relation could also be performed at the level of failure modes.

8.8 Quantitative evaluation

The quantitative evaluation according to ISO 26262 Part 5 claims the hardware architectural metrics and the evaluation of residual risk of safety goal violation. The concept for the quantitative evaluation of hardware designs is described. An overall view of the hardware quantitative evaluation is shown in Figure 25.

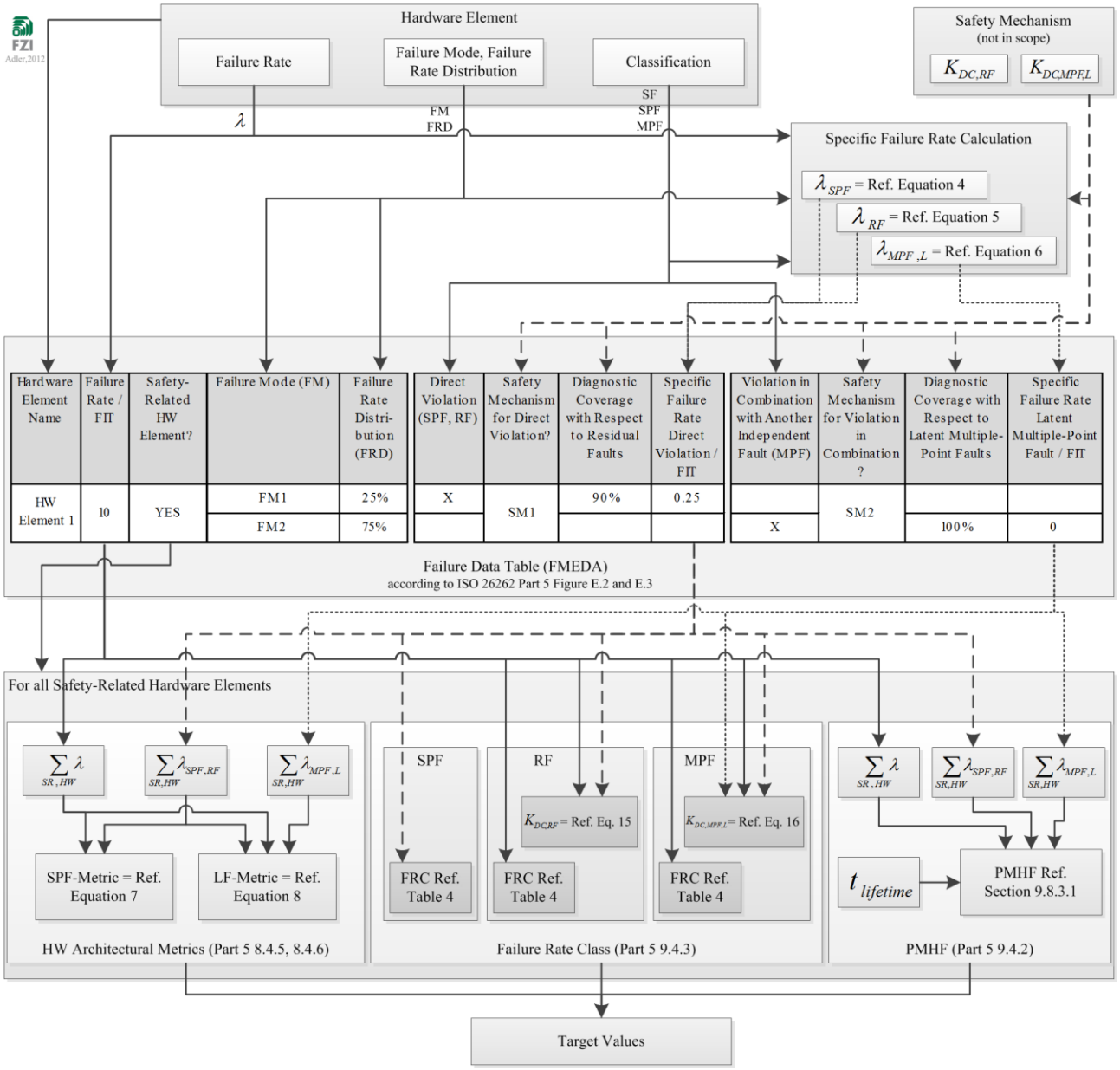


Figure 25: Overview: Flow diagram for hardware safety evaluation

Based on the model information regarding failure modes and classifications, the specific failure rates can be calculated in context of safety mechanisms. The classification process is shown in Figure 17. The determination of specific failure rates and the classification represents a FMEDA which leads to the construction of a failure data table capturing all relevant information for the safety evaluation. The failure data table contains all hardware elements failure mode, their effects on the system behavior via the classifications as well as quantitative values for failure modes and diagnostic coverage. Based on the failure data table, the quantitative evaluations according to ISO 26262 Part 5 can be performed in terms of hardware architectural metrics and evaluation of safety goal violations.

8.8.1 Hardware FMEDA

The classification of the failure modes, safety-relevance and the failure information of hardware elements captured in the model and together with the calculation of specific failure modes represents a FMEDA as a first step for quantitative hardware safety evaluation. The calculation of the specific failure rates in context of the classification and safety mechanism coverage is described in the following.

For a safe fault, the specific failure rate is calculated according to Equation 3. The failure rate distribution of the specific failure mode describes the ratio of the overall failure rate which is distributed to the failure mode in percent.

$$\lambda_{SF} = \lambda_{HWElement} \cdot FRD_{FM} \quad \textbf{Equation 3}$$

In the same way, the specific failure rate for a single-point fault is calculated according to Equation 4.

$$\lambda_{SPF} = \lambda_{HWElement} \cdot FRD_{FM} \quad \textbf{Equation 4}$$

The calculation for single-point and safe fault follow the same equation, but a safe fault is considered as safe (fault of order zero) whereas a single-point fault violates the safety goal (fault of order one). The equation for the specific failure rate of a residual fault, which is a special case of a single-point fault when a safety mechanism is present, is calculated according to Equation 5.

$$\lambda_{RF} = \lambda_{HWElement} \cdot FRD_{FM} \cdot \left(1 - K_{DC,RF}\right) \quad \textbf{Equation 5}$$

For this calculation, the diagnostic coverage with respect to residual faults of the safety mechanism present is considered. In case of a multiple-point fault, the failure rate of the latent multiple-point fault is calculated according to Equation 6.

$$\lambda_{MPF,L} = \left[\left(\lambda_{HWElement} \cdot FRD_{FM}\right) - \lambda_{SPF} - \lambda_{RF}\right] \cdot \left(1 - K_{DC,MPF,L}\right) \quad \textbf{Equation 6}$$

In this context the diagnostic coverage with respect to latent multiple-point fault of the safety mechanism is taken. As only the latent multiple-point fault is potentially violating the safety goal, the detected and perceived multiple-point specific failure rate is not in scope. If the failure mode is also directly violating the safety goal as a single-point or residual fault, the specific failure rates of the direct violation are not considered for the multiple-point failure rate calculation.

The specific failure rates of all single failure mode contributions as well as common information of the hardware element such as name, the failure rate, failure modes are stored in a failure data table. A possible predefined structure of this table is presented in ISO 26262 [1] Part 5 Table E.2 and E.3. The formal way of storage serves to identify the main failure modes which contribute to the single-point fault and latent-fault metric and evaluation of residual risk of safety goal violations. This is especially in focus if the target values are not met and additional measures have to be initiated.

The failure data table represents the basis for all safety assessments of the hardware and represents the primary result of the FMEDA for hardware elements.

8.8.2 Hardware architectural metrics

The hardware architectural metrics describe the overall robustness of the system against specific faults. The evaluation is claimed for ASIL-C and ASIL-D classification of the safety goal and recommended for ASIL-B. The hardware architectural metrics are related to the safe-failure fraction (SFF) of IEC 61508. Due to the fact, that ISO 26262 provides a more detailed classification of faults, as described in Section 8.3.1, two different architectural metrics are present: the single-point fault metric and latent-fault metric. The single-point fault metric evaluates the impact of single-point faults on the system whereas the latent-fault metric analyzes the impact of latent multiple-point faults.

For the calculation of the hardware architectural metrics, the following values have to be determined:

- Total Failure Rate: $\sum \lambda$
- Total Safety-Related Failure Rate: $\sum_{SR} \lambda$
- Sum of Single-Point and Residual Faults: $\sum (\lambda_{SPF} + \lambda_{RF})$
- Sum of Latent Multiple-Point Faults: $\sum \lambda_{MPF,L}$

The single-point fault metric considers the amount of direct violations of the safety goal (single-point or residual faults) in the hardware design. This amount is divided by the total safety-related failure rate. For visualization, the value is afterwards subtracted from 100%. Therefore, a high amount of single-point or residual faults which is critical for system reliability leads to a low amount of the single-point fault metric, high robustness is achieved by a value near to 100%. The single-point fault metric can be regarded as a kind of diagnostic coverage regarding direct violations in context of the overall hardware design. The single-point fault metric can be calculated according to Equation 7.

$$SPF - Metric = 1 - \frac{\sum_{SR,HW} (\lambda_{SPF} + \lambda_{RF})}{\sum_{SR,HW} \lambda_{HWElement}} \qquad \textbf{Equation 7}$$

For the latent-fault metric, the sum of all latent multiple-point fault specific failure rates of all safety-related hardware elements is considered. All direct violations (single-point and residual faults) are already considered in the single-point fault metric and therefore subtracted in the denominator. Equal to the single-point fault metric, a high value of the latent-fault metric near to 100% indicates high robustness of the hardware design against latent multiple-point faults.

$$LF - Metric = 1 - \frac{\sum_{SR,HW} \lambda_{MPF,L}}{\sum_{SR,HW} (\lambda_{HWElement} - \lambda_{SPF} - \lambda_{RF})} \qquad \textbf{Equation 8}$$

For comparison to IEC 61508, the formula for the SFF is given in Equation 9. Basically the same concept is used as it can be seen from the structure of the equations. For the SFF, the numerator contains the safety-critical failures which are dangerous and undetected, whereas the denominator contains the sum of all safety-related failures in the hardware design. Due to the finer differentiation regarding the order of the faults, the architectural metrics provide two separate evaluations.

$$SFF = 1 - \frac{\sum_{SR,HW} \lambda_{DU}}{\sum_{SR,HW} \lambda_{HWElement}}$$ **Equation 9**

The calculated values for the hardware architectural metrics can afterwards be verified against target values. These target values depend on the ASIL-classification of the corresponding safety requirement and can be user-defined or taken from recommendation of ISO 26262 as shown in Table 2.

	ASIL-B	ASIL-C	ASIL-D
Single-Point Fault Metric	≥ 90%	≥ 97%	≥ 99%
Latent-Fault Metric	≥ 60%	≥ 80%	≥ 90%

Table 2: Recommended target values for the hardware architectural metrics [1] Part 5 Table 4, 5

The target values shall provide evidence that the hardware design complies with the safety requirements. Other target values can be derived from similar well-trusted design principles. Well-trusted in this context means, that they were used without any condition that deviates from expectation regarding functional safety.

8.8.3 Evaluation of safety goal violations due to random hardware failures

The evaluation of residual risk of safety goal violation can be performed using one of the alternative methods, the probabilistic metric for random hardware failure (PMHF) or the second method using failure rate classes (FRC). The evaluation is claimed for ASIL-C and ASIL-D classification of the safety goal and recommended for ASIL-B.

8.8.3.1 PMHF

The probabilistic metric for random hardware failure describes an overall probabilistic value for a top-level system failure. It is related to the probability of dangerous failure per hour (PFH) and probability of dangerous failure on demand (PFD) of IEC61508. PFD is according to [2] only claimed for E/E systems with a low demand mode of operation, exemplarily described as maximum one demand per year [2] Part 4, 3.5.16. For systems with high or continuous demand mode, PFH has to be evaluated. The concept from PFH forms the basis for PMHF of ISO 26262.

The interpretation of PFH was widely discussed, exemplarily in the article of [29]. The proposed right notion is that the PFH refers to the average value of the unconditional failure intensity $w(t)$. The calculation of the PFH can then be expressed according to Equation 10.

$$PFH = w_{avg} = \frac{1}{T} \cdot \int_0^T w(t)dt$$ **Equation 10**

The unconditional failure intensity $w(t)$ in the case of non-repairable entities can be identified as the probability density $f(t)$ which refers to the probability of failure $F(t)$ versus time as $f(t) = \frac{dF(t)}{dt}$. For repairable items, the probability of failure is assumed to be zero after the repair which leads to very complex probabilities. ISO 26262 Part 5 9.4.2.2 describes the PMHF as the “average probability per hour over the operational lifetime of the item”. This relates to the definition of the PFH as the average value of the unconditional failure intensity of the item. In ISO

26262 Part 5 9.4.2.1, it is stated that “target values for the maximum probability of the violation of each safety goal shall be defined”.

Due to the fact that fault tree analysis is used for the model-based approach in context of the hardware architectural design, the use of PMHF for this level of detail would be reasonable. In ISO 26262 Part 10 Annex B.4, different scenarios regarding probability evaluation using fault tree analysis are discussed. For an event, considering constant failure rates as described in Section 8.3.1 and an exponential distribution, following Equation 11 describes the probability of failure over time $F(t)$.

$$F(t) = 1 - e^{-\lambda \cdot t} \qquad \textbf{Equation 11}$$

This probability of failure over time in hours is shown in Figure 26 for an exemplarily failure rate of $\lambda = 2\,FIT$.

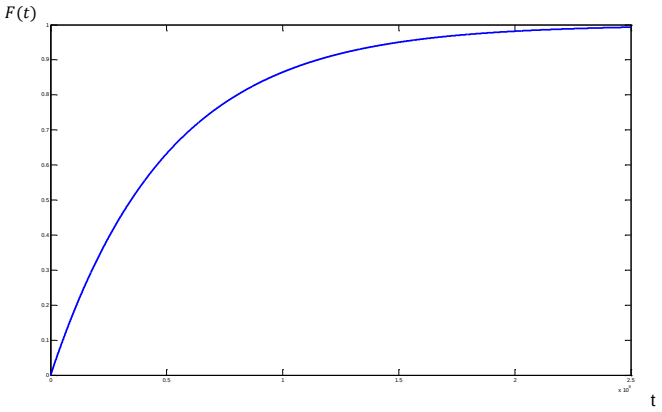


Figure 26: Probability of failure $F(t)$ over time

Scaling of the x-axis in hours of operation for this example is very high ($10^9 h$), due to the low failure rate value. For common lifetime of vehicles, which is assumed as 5000h according to ISO 26262 Part 10 Annex B.4, and low failure rates ($\lambda \cdot t << 0.01$), the exponential distribution can be simplified to a linear function, with the gradient of λ.

For comparison, in Figure 27 the exponential distribution for $\lambda = 2\,FIT$ is plotted in blue, while the simplified function $F(t) = \lambda \cdot t$ is plotted in red in the same diagram. Here the x-Axis is shown for the assumed system lifetime of 5000h.

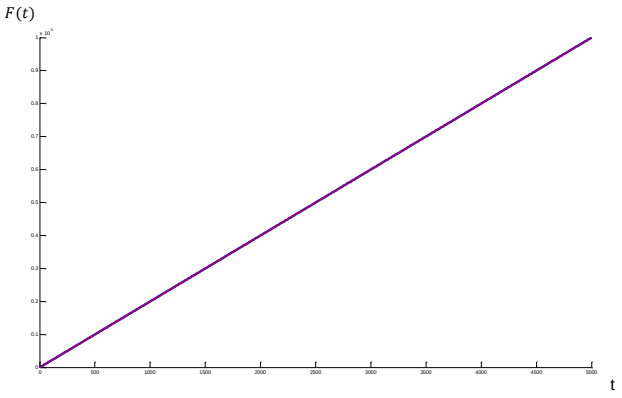


Figure 27: Simplified linearization of probability of failure $F(t)$ over system lifetime

The corresponding unconditional failure intensity $w(t)$ for this example would be constant over time and equal to the failure rate λ of the event. This is shown in Figure 28. The average unconditional failure intensity w_{avg} in this case would be of same value $\lambda = 2\,FIT$.

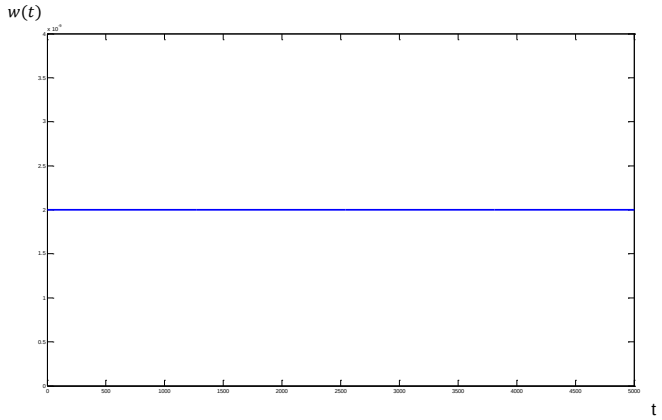


Figure 28: Unconditional failure intensity $w(t)$ over system lifetime

If considering two events with failure rates λ_1 and λ_2 of a fault tree following the simplified distribution as described, when having an OR-combination between the two events, the simplified linearization would lead to the probability of failure over time as described in Equation 12.

$$F_1(t) = \lambda_1 \cdot t + \lambda_2 \cdot t = (\lambda_1 + \lambda_2) \cdot t \qquad \textbf{Equation 12}$$

Therefore, the unconditional failure intensity and the average value would be the addition of the two failure rates λ_1 and λ_2. If considering AND-combination, the probability of failure over time for the linearized approach would be described as shown in Equation 13.

$$F_2(t) = \lambda_1 \cdot t \cdot \lambda_2 \cdot t = (\lambda_1 \cdot \lambda_2) \cdot t^2 \qquad \textbf{Equation 13}$$

This leads to a polynomial probability of failure $F_2(t)$ over time which is shown in Figure 29. This polynomial probability of failure over time results in a linear unconditional failure intensity $w(t)$ with the gradient of $2 \cdot \lambda_1 \cdot \lambda_2$. Therefore, the average value w_{avg} in this case is represented as $\lambda_1 \cdot \lambda_2 \cdot t$.

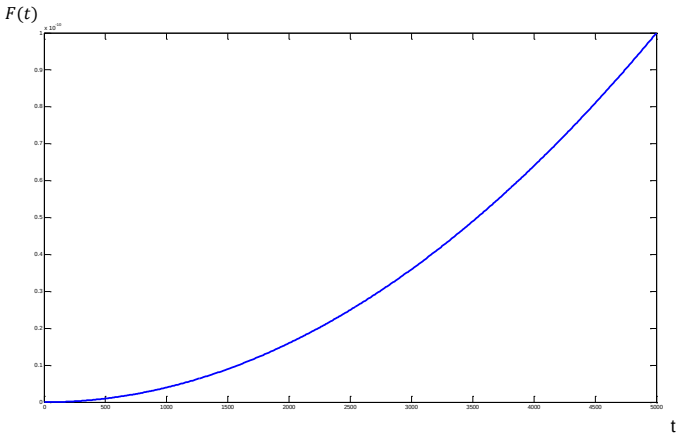


Figure 29: Probability of failure $F_2(t)$ over system lifetime for polynomial approach

ISO 26262 Part 10 Annex B.4 describes, that in this case of AND-combination, the corresponding target value must be met for $\lambda_1 \cdot \lambda_2 \cdot t$ while using the system lifetime.

A simplified estimation of PMHF could be described as the sum of all residual and single-point faults due to their direct influence (OR-combination). The AND-combination of latent-fault failure rates lead to a low influence on the PMHF, even if higher failure rates are present. Therefore, a simplified PMHF value could be calculated according to Equation 14.

$$PMHF \approx \sum_{SR,HW} (\lambda_{SPF} + \lambda_{RF}) \qquad \textbf{Equation 14}$$

Target values for PMHF depend on the ASIL classification of the safety requirement. Recommended target values are given in Table 3. In comparison to the PMHF, target values for PHF from [2] are also presented according to the corresponding SIL classification.

ASIL	Random hardware failure target values (PMHF)	PFH target values	SIL
ASIL-D	$< 10^{-8}h^{-1}$ $(equal\ to\ 10\ FIT)$	$\geq 10^{-9}h^{-1}$ $(1\ FIT)$ to $< 10^{-8}h^{-1}$ $(10\ FIT)$	SIL-4
ASIL-C	$< 10^{-7}h^{-1}$ $(equal\ to\ 100\ FIT)$	$\geq 10^{-8}h^{-1}$ $(10\ FIT)$ to $< 10^{-7}h^{-1}$ $(100\ FIT)$	SIL-3
ASIL-B	$< 10^{-7}h^{-1}$ $(equal\ to\ 100\ FIT)$	$\geq 10^{-7}h^{-1}$ $(100\ FIT)$ to $< 10^{-6}h^{-1}$ $(1000\ FIT)$	SIL-2
		$\geq 10^{-6}h^{-1}$ $(1000\ FIT)$ to $< 10^{-5}h^{-1}$ $(10000\ FIT)$	SIL-1

Table 3: Recommended target values for PMHF [1] Part 5 Table 6 and PFH [2] Part 1 Table 2

In addition to the recommended target values of [1], they can be derived from field data or quantitative analysis techniques applied to similar well-trusted design principles.

8.8.3.2 Evaluation of each cause of safety goal violation (FRC)

ISO 26262 [1] Part 5 9.4.3.2 recommends the application of the second method as failure rate class (FRC) for the hardware detailed design. This individual evaluation of each cause of safety goal violation is recommended to be applied for the detailed abstraction level. The process for the evaluation using the failure rate class approach is shown in Figure 30.

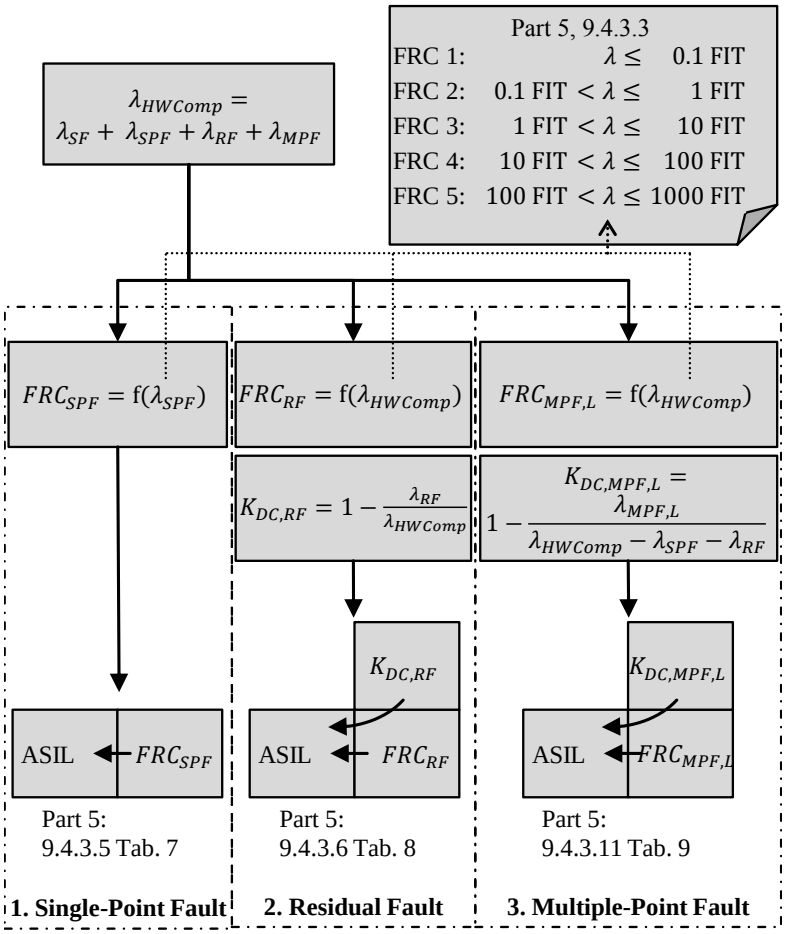


Figure 30: Overview: Failure rate class method [27]

For the failure rate class method, each individual violation of a safety goal by a hardware element has to be evaluated. The evaluation is not done at the level of failure modes, thus at the level of hardware elements. All failure modes of a hardware element with the same classification (single-point, residual or multiple-point) are clustered and evaluated together. In contrast to PMHF, the FRC method provides a more stringent evaluation as each violation has to meet a specified target value. For PMHF, only the overall target value has to be met, allowing single failure modes to maintain a high failure rate.

Failure Rate Class	Lower bound	Upper bound
Failure Rate Class 1	–	$\leq 0.1\ FIT$
Failure Rate Class 2	$0.1\ FIT <$	$\leq 1\ FIT$
Failure Rate Class 3	$1\ FIT <$	$\leq 10\ FIT$
Failure Rate Class 4	$10\ FIT <$	$\leq 100\ FIT$
Failure Rate Class 5	$100\ FIT <$	$\leq 1000\ FIT$

Table 4: Failure rate classes example for number of cut-sets = 100

The evaluation includes the classification of the failure rate into proposed failure rate classes according to Table 4. This is achieved in accordance with ISO 26262 Part 5 9.4.3.3. For residual and latent multiple-point faults, a diagnostic coverage on hardware element level has to be determined which is verified against target values from ASIL of the safety goal. The target values are taken from the overall target values from PMHF, according to Table 3. This ensures

consistency between the two alternative methods for evaluation of residual risk of safety goal violation. This global target value is distributed to each violation dividing it by a number which can be based on the number of minimal cut-sets in the system. ISO 26262 proposes the recommended value of 100. The recommended value can be altered if it is ensured that a correct failure rate classing is maintained when considering different cut-set order together.

The maximum value for the failure rate class 1 represents the target value of PMHF ASIL-D divided by the recommended value of 100 taken from [1] Part 5 9.4.3.4. For every higher number of failure rate class, the failure rate maximum value shall be less than or equal to 10 times the failure rate corresponding to the next lower failure rate class. The number of failure rate classes can be defined according to the failure rate values existent in the system, the failure rate classes 1, 2 and 3 are introduced analogous to the occurrence levels 1, 2 and 3 which are used in a FMEA.

For the evaluation of residual risk of safety goal evaluation, dedicated measures have to be applied, if prescribed by the target verification. These dedicated measures according to ISO 26262 Part 5 [1] 9.4.2.4 can include:

- Design features
- Special sample tests
- Burn-in test
- Dedicated control set
- Assignment of safety-related special characteristics

Single-Point Fault Verification:

In case of single-point faults, all single-point fault specific failure rates of a single hardware element have to be summed up. The sum of these failure rates then has to be classified according the failure rate classes, see Table 4. The corresponding failure rate class is verified against target values according to the ASIL of the safety goal as given in Table 5.

ASIL	Failure Rate Class
ASIL-D	Failure rate class 1 + dedicated measures
ASIL-C	Failure rate class 2 + dedicated measures or Failure rate class 1
ASIL-B	Failure rate class 2 or Failure rate class 1

Table 5: Failure rate class target values for single-point faults, [1] Part 5 Table 7

For ASIL-D classification of the safety requirement, the failure rate sum regarding single-point faults of each hardware element has to be classified in failure rate class 1 and additionally dedicated measures have to be ensured. For ASIL-C, either failure rate class 1 or failure rate class 2 with additional measures have to be met. For ASIL-C, failure rate class 1 or failure rate class 2 are required.

Residual Fault Verification:

In case of residual faults, the failure rate has to be classified according to Table 4 for the overall failure rate of the hardware element. Additionally, the diagnostic coverage with respect to residual faults has to be calculated on hardware element level. This diagnostic coverage is not the same as the diagnostic coverage of a safety mechanism on system level. The diagnostic coverage with respect to residual faults on hardware element level is calculated according to Equation 15.

$$K_{DC,RF} = 1 - \frac{\lambda_{RF}}{\lambda_{HWComp}}$$ **Equation 15**

The failure rate classification and the diagnostic coverage have to be verified together against target values according to the ASIL of the safety goal as given in Table 6.

ASIL	$K_{DC,RF} \geq 99.9\ \%$	$K_{DC,RF} \geq 99\ \%$	$K_{DC,RF} \geq 90\ \%$	$K_{DC,RF} < 90\ \%$
ASIL-D	Failure rate class 4	Failure rate class 3	Failure rate class 2	Failure rate class 1 + dedicated measures
ASIL-C	Failure rate class 5	Failure rate class 4	Failure rate class 3	Failure rate class 2 + dedicated measures
ASIL-B	Failure rate class 5	Failure rate class 4	Failure rate class 3	Failure rate class 2

Table 6: Failure rate class target values for residual faults, [1] Part 5 Table 8

Where the evaluation for single-point faults describes a direct verification of the failure rate classification against the target ASIL, for the residual fault a third criteria as the diagnostic coverage of the hardware element regarding residual faults is taken into account. Dedicated measures have to be taken for a diagnostic coverage lower than 90% while targeting ASIL-D or ASIL-C. This table can be extended to the left by adding additional columns which describe a higher diagnostic coverage than the maximum 99.9%. This can be achieved in accordance with ISO 26262 Part 5 9.4.3.7.

Latent Multiple-Point Fault Verification:

In case of multiple-point faults, ISO 26262 Part 5 7.4.3.2 suggests to limit the analysis to dual-point faults in the most cases. When applying the second method for the evaluation as FRC for multiple-point faults, plausibility of dual-point failures has to be considered according to ISO 26262 Part 5 Clause 9.4.3.8 and 9.4.3.9. If the dual-point failure is not plausible, it shall be accepted with the safety goal target.

Plausibility is on the one hand given if one of the dual-point faults remains latent for a time longer than the multiple-point detection interval. For ASIL-D and ASIL-D, if there is no value prescribed, the multiple-point fault detection interval can be specified as equal or lower than the item's "power-up to power-down" cycle, according to ISO 26262 Part 5 Clause 6.4.8. Additionally, a dual-point fault is plausible, if one of the hardware parts diagnostic coverage regarding multiple-point latent faults is lower than target values given in Table 7.

ASIL	Hardware part diagnostic coverage
ASIL-D	$< 90\%$
ASIL-C	$< 80\%$

Table 7: Plausibility of dual-point faults

If the multiple-point fault has to be evaluated, the failure rate classification is provided for the overall failure rate of the hardware element in the same way as for residual faults. The diagnostic coverage with respect to latent multiple-point faults has to be calculated on hardware element level according to Equation 16. Same as for residual faults, this diagnostic coverage is not equal to the diagnostic coverage of a safety mechanism on system level.

$$K_{DC,MPF,L} = 1 - \frac{\lambda_{MPF,L}}{\lambda_{HWComp} - \lambda_{SPF} - \lambda_{RF}}$$ **Equation 16**

The failure rate classification and the diagnostic coverage both have to be verified against target values provided by the ASIL of the safety goal. Target values are shown in Table 8.

ASIL	$K_{DC,MPF,L} \geq 99\ \%$	$K_{DC,MPF,L} \geq 90\ \%$	$K_{DC,MPF,L} < 90\ \%$
ASIL-D	Failure rate class 4	Failure rate class 3	Failure rate class 2
ASIL-C	Failure rate class 5	Failure rate class 4	Failure rate class 3

Table 8: Failure rate class target values for dual-point faults [1] Part 5 Table 9

For the dual-point fault individual evaluation, a combination of the occurrence of the fault represented by the failure rate classification and the diagnostic coverage has to be evaluated against target values provided by the ASIL classification of the safety requirement.

8.9 Outlook

The presented methodology for hardware safety evaluation will be further refined and applied to use cases in terms of concept validation.

For complex parts, such as a microcontroller or an ASIC, further analysis of the internal structure, as shown exemplarily in Figure 31, is necessary. Therefore, an approach according to the safety evaluation presented in this deliverable is promising. This will be in focus of our future research in terms of the presented model-based methodology.

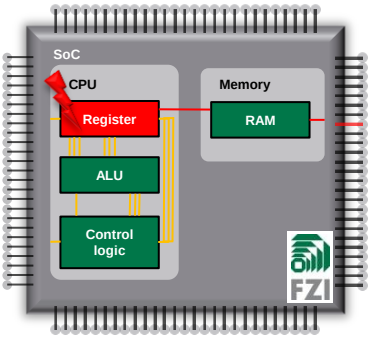


Figure 31: System-on-chip safety analysis

9 Methodology 2: Consistency checks for the safety case

The ISO 26262 points out the importance of checking the work products throughout the whole safety lifecycle for consistency e.g. ISO 26262-2, C.2.2, ISO 26262-3, 7.4.5.1, 8.4.5.1, ISO 26262-4, 6.4.1.2, 6.4.6.1, 7.4.1.1 [1]. As the essential results of the work products are summarized in the safety case, the safety case offers a good focal point to assure the consistency of central work products.

The formalized SAFE meta model structures can be exploited to produce tangible automatic consistency checks which can improve the quality of the macro structure and ensure the consistency of traceability of the elements of the safety case.

Chapter 9.1 describes the macro structure of a safety case report which is based on the initial structure defined in WT3.1.3. A template is used to describe how the sections of a safety case report can be automatically generated and which meta model elements are involved.

Chapter 0 first describes the general idea of performing consistency checks on the SAFE meta model. Furthermore a template is given which supports the documentation of individual consistency rules, including the rationale behind the rule and patterns for checking the consistency. The template is applied to describe consistency rules for checking the macro structure of a safety case.

9.1 Macro structure of the safety case report

In the following tables the macro structure of the safety case report is described. The “Report section” contains the name of the section of the safety case. The “Section summary” gives a rationale why the information is provided in the safety case. The “Artifacts” gives information on the artifacts which are provided in the safety case report section. The “Meta model artifact pattern” gives a graphical representation of the relevant meta model artifacts and their relations. The artifacts which are marked with a check box are included in the actual report section. Sometimes there is more than one artifact pattern required. In this case only one pattern is given as an example.

<table>
<tr><th colspan="2">Report section:

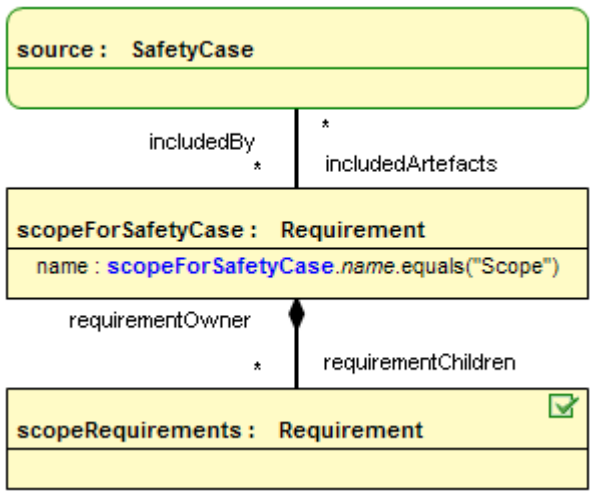
Scope</th><th>Meta model artifact pattern</th></tr>
<tr><td>Section summary</td><td>The following requirements describe the scope of this safety case. The scope requirements limit the applicability of the arguments presented in this safety case in the intended way. The scope section of the safety case report defines the context within which the remainder of the safety case arguments is valid.</td><td rowspan="2"> 
</td></tr>
<tr><td>Artifacts</td><td>List of requirement artifacts which describe the scope for the safety.</td></tr>
</table>

Table 9: Scope section of safety case report

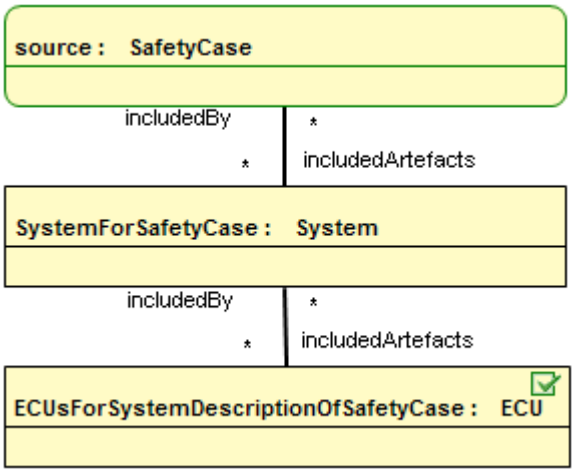
Report Section: System Description		**Meta model artifact pattern (example)**
Section Summary	The system description presents an overview of the system. It is not the purpose of this section to provide full design detail. Full design detail is available in the system specification documents. The descriptions in this section are intended to help the reader of the safety case to understand the following sections.	
Artifacts	Artifacts of the system description such as ECUs, sensors, actuators, processors, HW Modules	

Table 10: System description section of safety case report

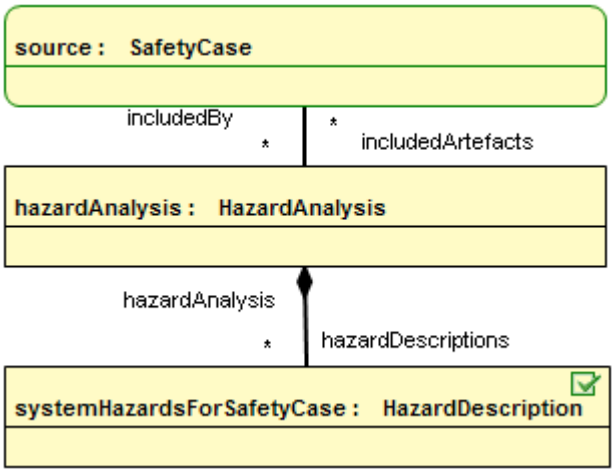
Report Section: System Hazards		**Meta model artifact pattern**
Section Summary	The system hazard section presents an overview of the system hazards. It is not the purpose of this section to provide full detail. Full detail including operational scenarios and operating modes is available in the hazard and risk analysis report.	
Artifacts	List of system hazards.	

Table 11: System hazards section of safety case report

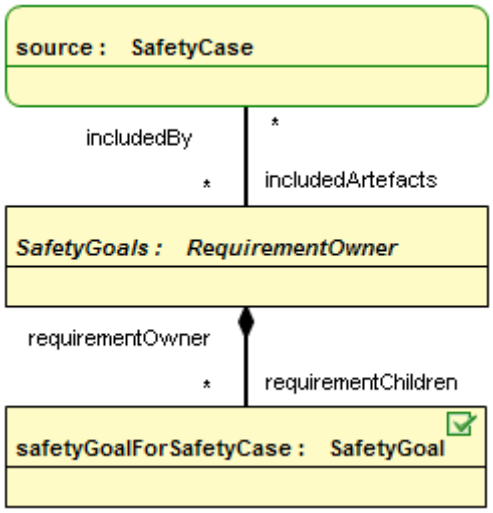
Report Section: System Requirements		**Meta model artifact pattern (example)**
Section Summary	The safety requirements section presents an overview and summary of the safety requirements for the system. It lists the safety goals, functional safety requirements and technical safety requirements. It is not the purpose of this section to provide full detail. Full detail is provided in the specification of the system.	
Artifacts	List of safety goals, functional safety requirements and technical safety requirements.	

Table 12: Safety requirements section of safety case report

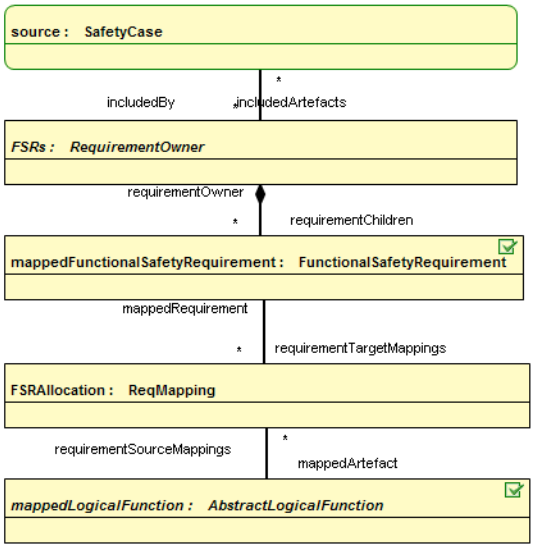
Report Section: Risk reduction measures: Functional safety concept		**Meta model artifact pattern (example)**
Section Summary	The section summarizes the functional safety concept. For each functional safety requirement, the function which implements the safety requirement is given.	
Artifacts	List of functional safety requirements together with the elements of the preliminary architecture.	

Table 13: Risk reduction section (functional safety concept)

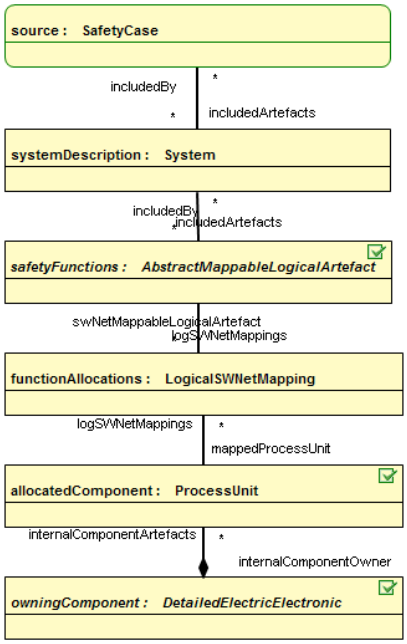
Report Section: Risk reduction measures: Functional safety concept		Meta model artifact pattern (example)
Section Summary	The section summarizes the technical safety concept. For each function the hardware or software element which implements the function is given.	
Artifacts	List of functions together with the hardware or software elements to which the function is allocated.	

Table 14: Risk reduction section (technical safety concept)

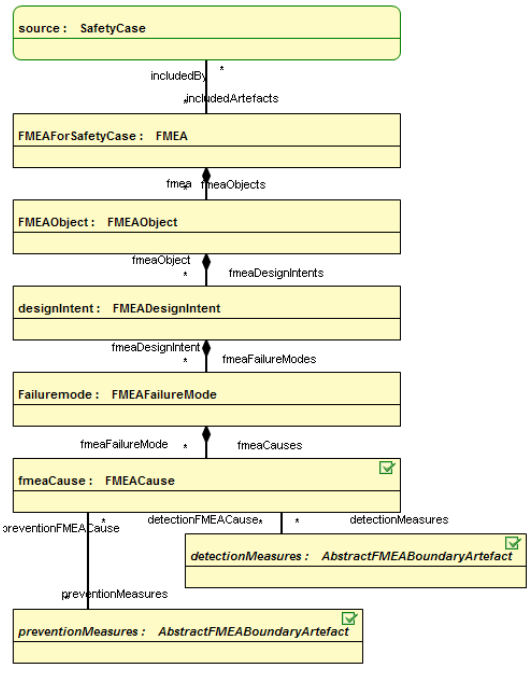
Report Section: Safety analysis: Overview of malfunctions / faults / failures		Meta model artifact pattern (example)
Section Summary	The safety analysis presents an overview of the malfunctions / faults / failures which habe been identified by analyzing the safety concepts presented in the previous section. Full detail of the analysis is available in the reports of the respective methods (e.g. FMEA report).	
Artifacts	List of malfunctions / faults / failures detected by safety analysis together with the identified detection and prevention measures.	

Table 15: Safety analysis section (malfunction / faults / failures)

9.2 General concept for application of consistency checks and metrics

Performing coverage and consistency checks during the development of the system safety case helps to ensure the formal quality of each work product as well as the formal quality of the overall safety case. This is illustrated in the following picture for the work products safety goals and functional safety requirements: A set of consistency checks is applied to a section of the SAFE meta model. In the example the section includes the safety goals which are derived from the hazard and risk analysis. Consistency checks could now examine the meta model artifacts for consistency with related artifacts, e.g. hazards or the derived functional safety requirements.

As indicated in Figure 32 results of the consistency checks could be fed in to quality metrics which compute a formal quality of the work product, e.g. the coverage of safety goals by functional safety requirements.

Both consistency checks and quality metrics can be used in quality reports which help the safety engineer to

- Instantly identify formal quality issues and correct them
- Get a quantitative summary on the formal quality of the work product

In this deliverable we focus on the definition of the consistency checks.

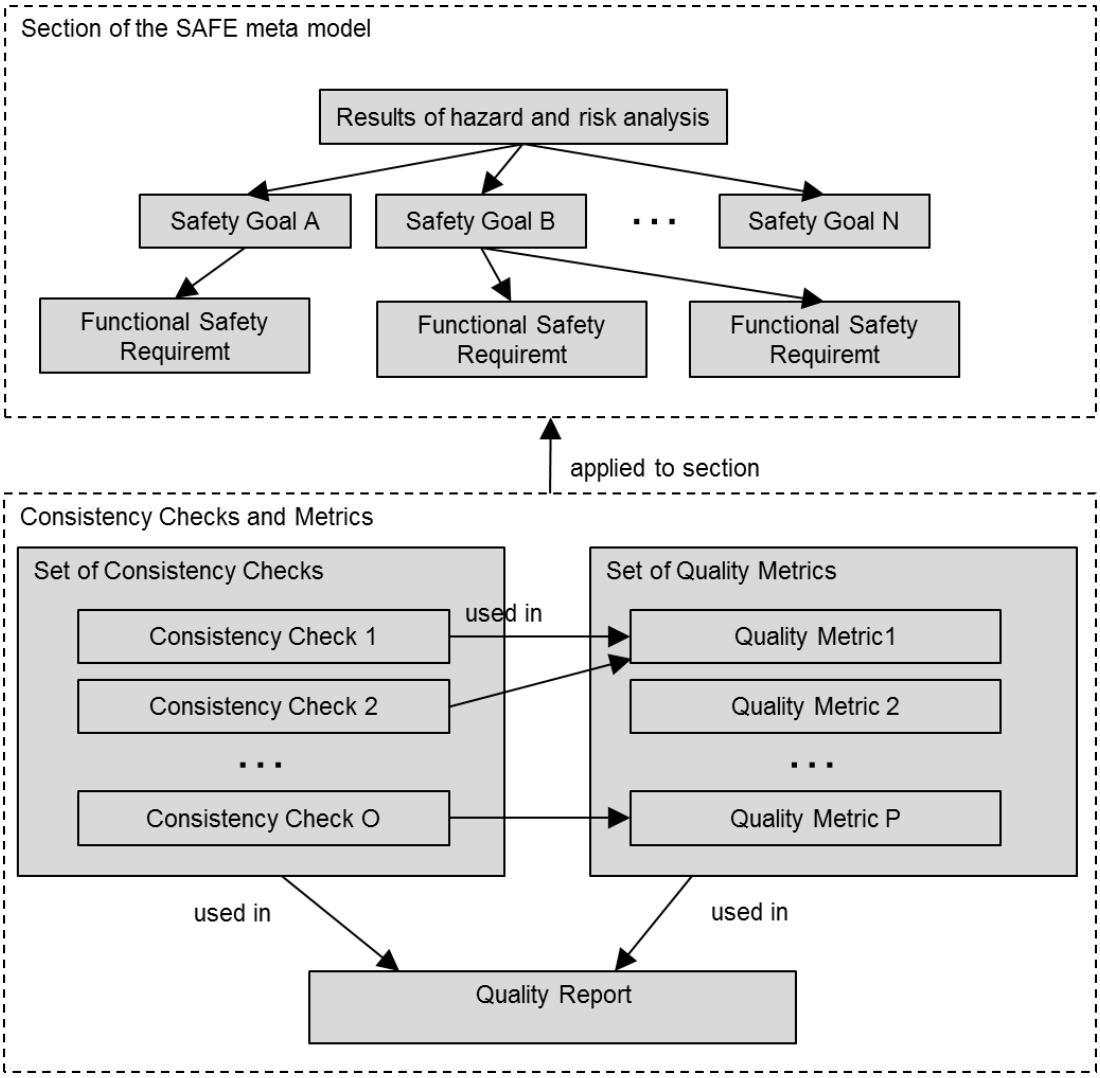


Figure 32: Interrelation of consistency checks and metrics and the SAFE meta model

9.3 Consistency checks for the safety case

The overall structure of the safety case has been defined in WT3.3.3. and further formalized in the previous sections. The preliminary work helps to define the meta model artifact patterns which need to be checked to ensure the consistency of the safety case.

In the following sections the consistency checks are described in detail. The “Short description” gives a short hint why the artifact of the safety case is inconsistent. The “Severity” classifies how important the inconsistency is. The “Explanation” gives more details on the reason of the inconsistency often with a reference to ISO 26262. The “Task” gives advice how to resolve the inconsistency. The “Meta model artifact pattern” gives a graphical representation of the relevant meta model artifacts and their relations which are relevant for checking the consistency. The artifacts which are marked with a grey color indicate negations. This means that the source artifact is inconsistent if the grey artifact is not present in the pattern. Sometimes there is more than one artifact pattern required to formulate a complete check. In this case only one pattern is given as an example.

9.4 Consistency checks for the macro structure of the safety case

The consistency checks described in this section ensure that all major sections of a safety case are available.

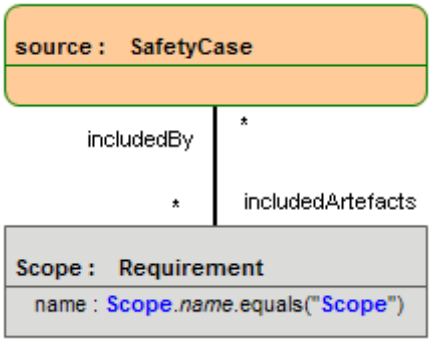
Check: SCMS001_SafetyCaseHasNoScope		**Meta model artifact pattern**
Short description	A scope is not defined for the safety case.	
Severity	Error	
Explanation	Requirements which describe the scope for the safety case are missing.	
Task	Add scope requirements to the safety case.	

Table 16: Checking the safety case for scope definition

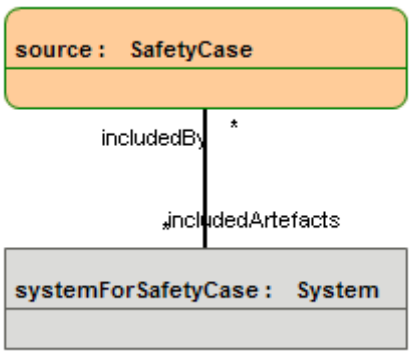
Check: SCMS010_SafetyCaseHasNoSystemDescription		**Meta model artifact pattern**
Short description	A system description is not defined for the safety case.	
Severity	Error	
Explanation	The system description shall provide an overview of the system for the safety case.	
Task	Add a system artifact to the safety case.	

Table 17: Checking the safety case for system description

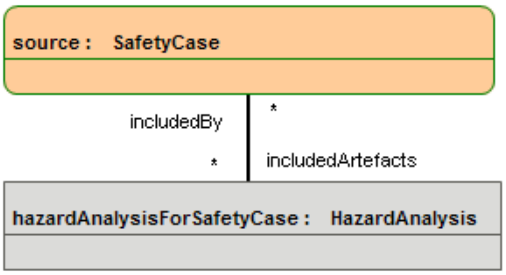
Check: SCMS020_SafetyCaseHasNoHazardAnalysis		Meta model artifact pattern
Short description	A hazard and risk analysis is not defined for the safety case.	
Severity	Error	
Explanation	The system hazards for the safety case are identified by applying the hazard analysis and risk assessment (see ISO 26262-3, 7.1)	
Task	Add a hazard analysis artifact to the safety case.	

Table 18: Checking the safety case for hazard and risk analysis

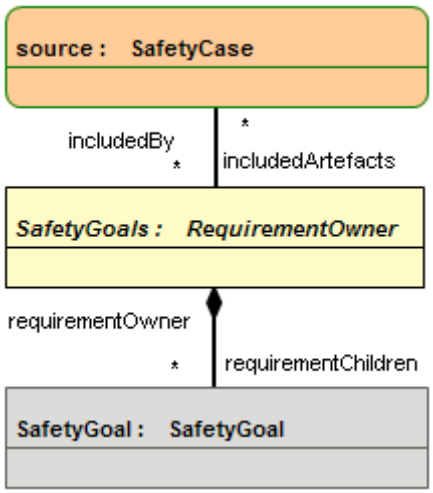
Check: SCMS032_SafetyCaseHasNoSafetyGoals		Meta model artifact pattern
Short description	Safety goals are not defined for the safety case.	
Severity	Error	
Explanation	Safety goals shall be identified based on the hazard and risk analysis (see ISO 26262-3, 7.4.4.3).	
Task	Add a requirement artifact to the safety case which contains the identified safety goals.	

Table 19: Checking the safety case for safety goals

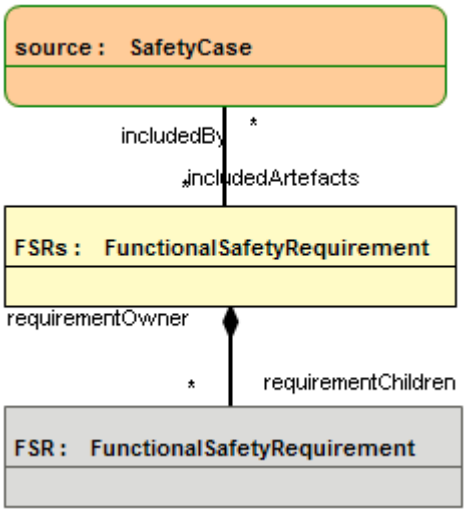
Rule Check SCMS034_SafetyCaseHasNoFSR		**Meta model artifact pattern**
Short description	Functional safety requirements are not defined for the safety case.	
Severity	Error	
Explanation	Functional safety requirements shall be identified based on the safety goals (see ISO 26262-3, 8.1).	
Task	Add a requirement artifact to the safety case which contains the identified functional safety requirements.	

Table 20: Checking the safety case for functional safety requirements

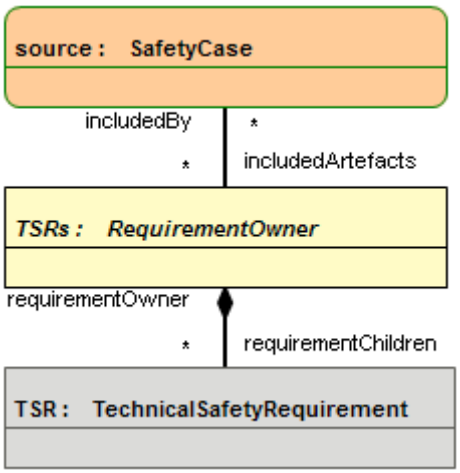
Check: SCMS036_SafetyCaseHasNoTSR		**Meta model artifact pattern**
Short description	Technical safety requirements are not defined for the safety case.	
Severity	Error	
Explanation	Technical safety requirements shall be identified based on the functional safety concept (see ISO 26262-4, 6).	
Task	Add a requirement artifact to the safety case which contains the identified technical safety requirements.	

Table 21: Checking the safety case for technical safety requirements

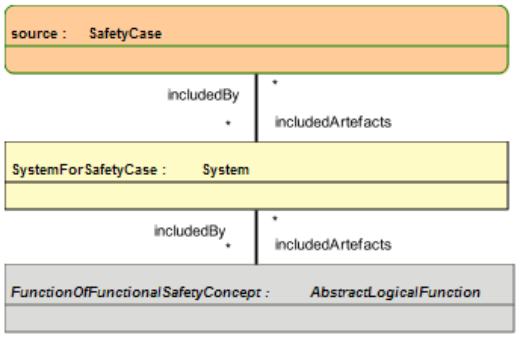
Check: SCMS040_SafetyCaseHasNoFunctionsForFunctionalSafetyConcept		Meta model artifact pattern
Short description	The functions for the functional safety concept are not defined for the safety case.	
Severity	Error	
Explanation	During the development of the functional safety concept, the functional safety requirements shall be allocated to preliminary architectural elements (see ISO 26262-3, 8). In PREEvision these preliminary artifacts are the functions of the logical architecture.	
Task	Add the identified logical functions to the system of the safety case.	

Table 22: Checking the safety case for functions of the functional safety concept

10 Methodology 3: Common-cause analysis in the geometric perspective using physical properties and environmental conditions

An important issue described in the automotive standard for functional safety ISO 26262 is the assurance of freedom from interference and the analysis of dependent failures. Physical factors like temperature, electromagnetic interference (EMI) or collisions can be an unintended cause for failures. They are potentials to cause dependent failures which do not occur independently as assumed and possibly lead to violations of safety requirements. Safety-related systems are typically developed in a distributed way and on different granularity levels such that common causes that result from physical interferences can easily be overseen. An example is the possible overheating that can result from multiple elements mounted in the same location of the vehicle. Each component itself does not produce more heat than the cooling system could handle, but caused through the positioning of the component the local heat limits can be exceeded and the components, which initially were assumed to be independent of each other, are likely to fail in a common cause. It is not obvious to detect potential dependent failures and interference between components in a system's architecture. Thus, it is important to have a methodology that guides the realization of safety-related systems and helps identifying situations with dependent failures which lead to violations of safety requirements. Furthermore the analysis should give that much insight of the relation of the affected elements that strategies for resolving the common cause failures can be developed.

To be able to identify these common causes additional information needs to be available in the development models:

- Physical properties of the components (e.g. production of heat or EMC properties)
- Physical constraints of the components (e.g. range of temperature where nominal operation can be guaranteed, maximum EM radiation that can be shielded)
- Geometric installation spaces (Components need to be map-able to positions inside the vehicle)
- Environmental conditions (e.g. the defined environmental temperature range is an important input to evaluate the maximum temperature in various regions of the vehicle)

It is intended to provide an interface from the models used in the safety oriented part of the development process to models used for physical simulations.

The methodology that is described in the following section helps installing a hardware architecture to a geometric architecture. It allows identifying potential violations of safety requirements due to dependent failures which are caused by coupling effects related to physical component characteristics and to the geometric system design. Based on an analysis of the safety requirement violations adequate mans of compensation can be introduced that ensure fulfilling the safety requirements. The methodology consists of several activities which can be used in an ISO 26262 oriented development process. The activities and their order are depicted in Figure 33. The methodology can be applied on system level (ISO 26262 Part 4) or on hardware level (ISO 26262 Part 5). ISO 26262 compliant safety goals as well as functional and technical safety requirements must be defined before. The installation methodology shall ensure the definition of a geometric topology for a hardware architecture that complies with safety requirements. It helps identifying potential dependent failures related to geometric design decisions for a hardware architecture.

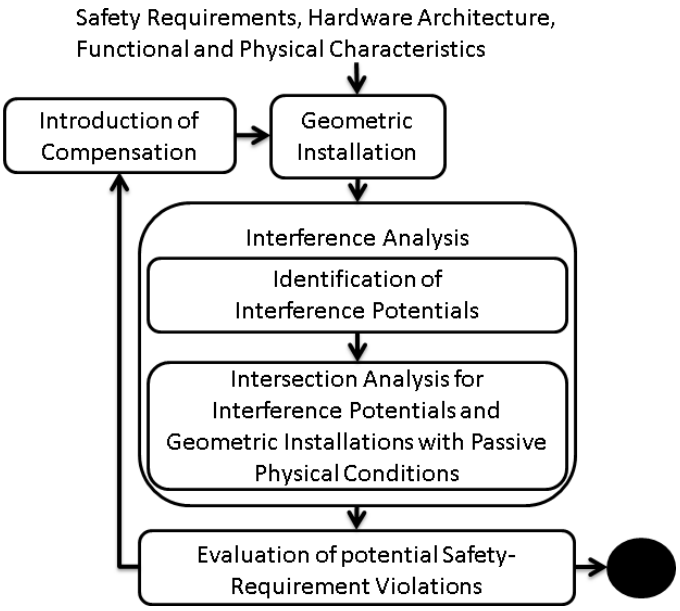


Figure 33: Steps for assessment of installations with environmental factors

10.1 Describing physical constraints and geometric installation

The description of physical constraints and geometric installations is the first step in the assessment of installations with environment factors after the safety requirements, a hardware architecture and physical environment conditions are defined. In this step first the physical environment conditions are assessed for the hardware components of the hardware architecture and for other components in the geometric architecture. In a second activity hardware component prototypes of the hardware architecture are installed to a geometric architecture.

10.1.1 Physical condition

Each hardware component typically has physical constraints and properties (e.g. maximum operation temperature). Physical conditions can also be assigned to non E/E elements to e.g. part of the chassis (like the coefficient of heat conductivity). The physical properties can be defined based on requirements, as described before, or based on the specification of single hardware or geometric components. As described in [30] one can distinguish between active and passive physical conditions.

- **Active Physical Properties**

  Active physical conditions are physical characteristics of a component which can actively influence an environment factor. Other component conditions like functional behavior can again influence the active condition. Examples are the characterization of the surface temperature of a hardware component, the heat flow related to a solid structure as well as cooling characteristics with regard to specific cooling modes.

- **Passive Physical Properties**

  Passive physical conditions are assumptions on how a component's behavior or other characteristics relate to an environment factor in the component's physical environment.

The condition therefore defines how a physical environment factor can influence a component. An example is the temperature region for which proper function of a component is promised. Other temperature regions are either too hot or too cold for the usage of the component.

The description of passive conditions can relate to component states including errors as part of the functional states which may lead to component failures. The violation of passive conditions can therefore be considered as conditions leading to faults, which are “abnormal condition(s) that can cause an element or an item to fail”, according the ISO 26262 vocabulary (ISO 26262 part 1). The description of active physical conditions can address conditions, events and states for a component including faults, errors and failures. Addressing them in the physical conditions will help detecting potential violations of safety requirements during the interference analysis. Hence, it is important to distinguish in which operating modes the active properties are valid. E.g. the emitted heat might be total different in nominal operation mode than in the case of a cooling fault.

To specify the physical properties, we propose to use the concept of contracts [2] to separate between assumptions and promises. Such contracts define physical component characteristics for an assumed environment. These characteristics refer to specific environment factors like temperature or EMC as defined by the ISO 26262 part 9, clause 7.4.4. We assume that physical conditions either define how a component (actively) influences an environment factor or that they define how a component (passively) reacts to an environment factor.

10.1.2 Geometric Installation

A geometric hardware topology is defined for the hardware architecture. Positions of a geometric architecture are assigned to hardware component prototypes with their safety requirements, physical conditions as well as behavioral and other characteristics. The definition of a geometric hardware topology will allow analyzing whether a chosen hardware architecture is compliant to safety requirements with regard to its installation to a geometric architecture. A geometric architecture is either designed or modified respecting the hardware architecture to be installed. It is assumed that the installation of a hardware architecture for an item to the geometric architecture of a vehicle is defined on system level (ISO 26262 part 4) as part of the system design (ISO 26262 part 4, clause 7). The geometric topology of the hardware components themselves is defined on hardware level (ISO 26262 part 5).

10.2 Idea of Analysis

In this section we propose the idea of a method to analyze possible interference for hardware component prototypes in a geometric hardware topology allowing the detection of potential dependent failures. “Single events or single causes that could bypass or invalidate a required independence or freedom from interference between given elements and violate a safety requirement or a safety goal” will be detected as required in the ISO 26262 part 9, clause 7. For this part of the ISO 26262, the method can be used to consider “similar and dissimilar redundant elements”, “functions and their respective safety mechanisms”, “physical distance between hardware elements, with or without barrier” and “common external resources”. The method is applied on system level to identify coupling effects on environmental factors as systematic failures, as described in ISO 26262 part 4, clause 7.4.3. It can be applied in accordance to ISO 26262 part 4, clause 8 “Item integration and testing” in order to verify physical conditions of an item implementation in the actual environment and in accordance to ISO 26262 part 5, clause 7 “Hardware design”.

The Analysis consists of two steps, the identification of interference potentials and the identification of affected safety goals.

The identification of interface potentials needs to be performed for each type of passive physical properties of a component. It need to be identified for each of these types which active physical properties directly or indirectly affect the passive constraint. For these components a physical simulation needs to be started, which as a result needs to provide the situation which could violate the physical constraints (passive conditions). For example, in the case of maximum operation temperature the result of the simulation can be a heat map representing the maximum temperature reached during the simulation run at various positions in the vehicle.

In the second step it needs to be identified if one of these temperatures exceeds the constraints existing for a particular place in the geometric installation model. If the temperature is bound to failure modes of the system, it can be possible that multiple faults occur at the same time based on the temperature, there are dependent. Existing fault trees can be modified and the result visualized.

10.3 Further Work

In the upcoming revision of this deliverable we will detail the modeling and analysis steps. In particular the interface of the physical simulation, how the meta-model needs to be designed to exchange the needed information and how the different properties need to be formalized using contracts.

11 Methodology 4: Multi-criteria deployment optimization and schedule generation

In the following section we present an approach that allows multi-criteria deployment optimization and schedule generation and can be used as part of a larger design space exploration approach as mentioned in [34]. It is possible to use multiple criteria for the approach, and the implementation counterpart to this document will feature a multi-criteria capable plug-in that can target ASIL-based allocation, worst-case-execution-time and number of hardware nodes for deployment. A theoretical concept for the inclusion of selected other safety metrics identified is going to be developed in the final version of the document, as identified by methodologies 1-3 and presented in sections 9, 10 and 11. The approach and descriptions thereof presented in this section are based entirely on the work of Sebastian Voss, presented in collaboration with Bernhard Schätz in "*Deployment and Scheduling Synthesis for Mixed-Critical Shared-Memory Applications*" at the ECBS 2013 [36].

11.1 Approach

The presented methodology represents an efficient approach for generating suitable system architectures for embedded systems efficiently. The focus is on a joint generation of schedules and deployment for mixed-criticality multicore architectures using shared memory. The presented approach computes task and message schedules that are optimized with respect to a global discrete time base. As part of the solution, the approach generates an optimized assignment of tasks to computation resources (cores) concerning local memory constraints of cores and criticality constraints of tasks. This approach is integrated into the AUTOFOCUS 3 tool-chain [32], using a formally defined model of computation with explicit data-flow and discrete-time semantics to develop multi-criticality embedded systems.

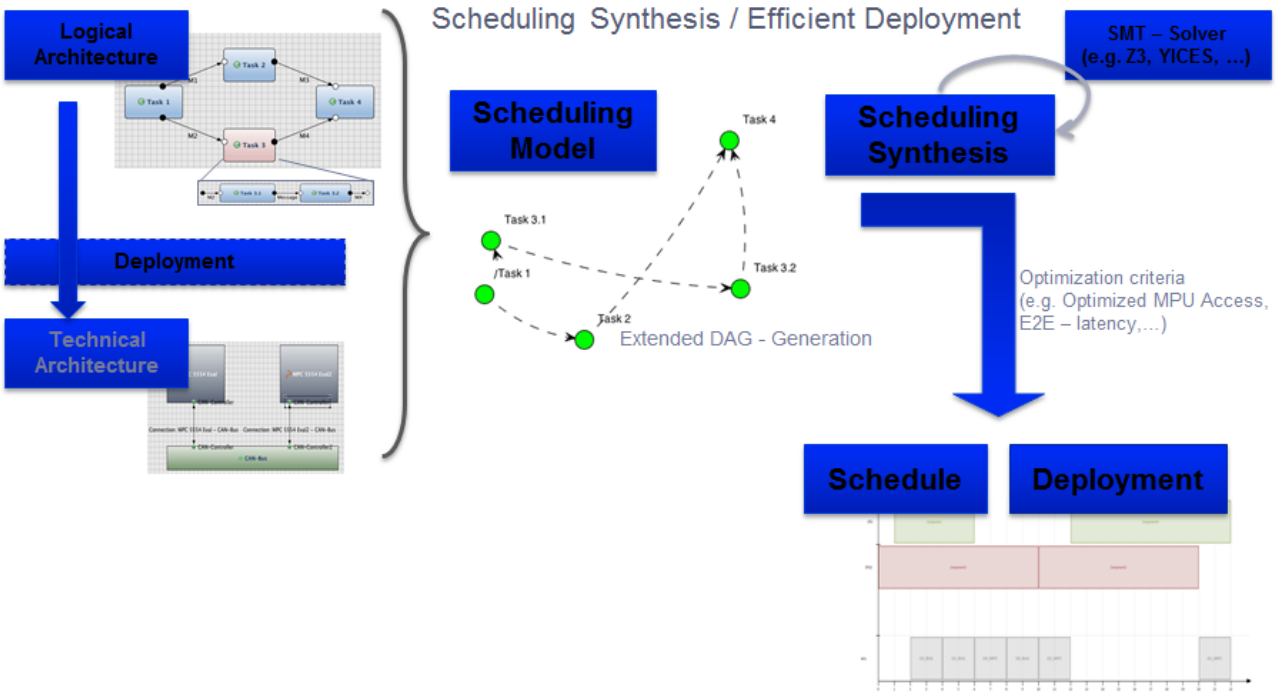


Figure 34: Deployment Synthesis in AF3 [37]

The approach relies on a symbolic encoding scheme, based on a system model that is derived from the system architecture. A formalization describing the scheduling problem as a satisfiability problem using boolean formulas and linear arithmetic constraints, which are the tackled by a state-of-the-art satisfiability modulo theory (SMT) solver in order to compute the joint schedule and deployment for such architectures, is presented in [36].

Implementations are being carried out in the research CASE tool AutoFOCUS3 (AF3)[9], the basic sequence of which is shown in Figure 4, part of the tutorial referenced in [37], and are presented in more detail in Section 9 of [34] as well as in [36] and [41].

Using this approach we provide an efficient deployment for multi-criteria problems (e.g. timing, scheduling) as well as calculate (optimized) partitioning and mapping of systems according to ASIL levels in a mixed-criticality environment, and has been developed in the context of the SPES_XT Core project [38].

As shown in Figure 26, the ASIL levels, which are propagated through the component links on the logical architecture, provide one criterion and we can freely select other criteria such as execution time, energy consumption or any other resource optimization.

The deployment synthesis is based on rules definition carried out inside the solver in AF3, as further explained in [36].

11.1.1 Scheduling Model

A mixed-critical application may consist of several components providing various functions. These functionalities can be described as computational activities, called tasks. We define $\boldsymbol{T} = \{\boldsymbol{t_0}, \boldsymbol{t_1}, \ldots, \boldsymbol{t_n}\}$ as a set of tasks. These tasks generally communicate by messages – in the following represented by $\boldsymbol{M} = \{\boldsymbol{m_0}, \boldsymbol{m_1}, \ldots, \boldsymbol{m_o}\}$ – and therefore cannot be executed in arbitrary order. The dependency of tasks is described by a precedence relation defining the execution ordering and is represented as a directed labeled graph, called a precedence graph $\boldsymbol{G} = \{\boldsymbol{T},\boldsymbol{E}\}$, where $\boldsymbol{E} \subseteq \boldsymbol{T} \times \boldsymbol{M} \times \boldsymbol{T}$ represents the dependencies between these tasks via the exchanged messages, as shown in figure 5. For these dependencies, we define two different functions: $\boldsymbol{\tau} : \boldsymbol{T} \rightarrow \boldsymbol{2^M}$ such that $\boldsymbol{\tau}(\boldsymbol{t}) = \{\boldsymbol{m} \mid \exists \boldsymbol{t'}.(\boldsymbol{t}, \boldsymbol{m}, \boldsymbol{t'}) \in \boldsymbol{E}\}$, and $\boldsymbol{\rho} : \boldsymbol{M} \rightarrow \boldsymbol{T}$ such that $\boldsymbol{\rho}(\mathrm{m}) = \boldsymbol{t'}$ for $(\boldsymbol{t}, \boldsymbol{m}, \boldsymbol{t'}) \in \boldsymbol{E}$, where $\boldsymbol{\tau}$ describes the set of messages $\boldsymbol{m} \in \boldsymbol{M}$ triggered by a task $\boldsymbol{t}$, and $\boldsymbol{\rho}$ describes for each message $\boldsymbol{m} \in \boldsymbol{M}$ the corresponding receiving task $\boldsymbol{t'} \in \boldsymbol{T}$. Furthermore, each task may have an annotated criticality, w.r.t. different safety integrity levels (SIL) as used, e.g., in [32].

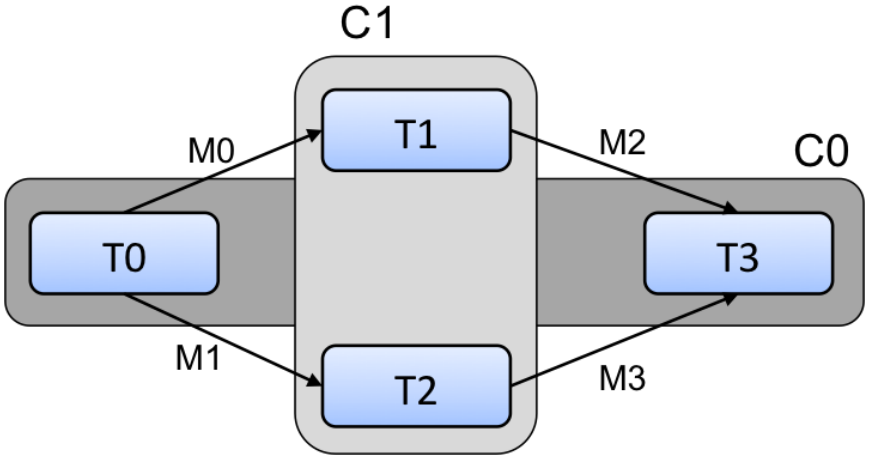


Figure 35: Graphical Visualization of a precedence graph ***G*** [36]

A computation resource may execute a set of concurrent tasks, that is, tasks that can overlap in time. These computation resources are called cores. Let $\boldsymbol{C} = \{\boldsymbol{c_0}, \boldsymbol{c_1}, \ldots, \boldsymbol{c_m}\}$ be a set of cores. Furthermore, let $\boldsymbol{\eta} : \boldsymbol{C} \rightarrow \boldsymbol{2^T}$ be a function that assigns to every core a set of tasks running on it. A set of buses $\boldsymbol{B}$ is used to transport messages between cores. For reasons of simplicity, in the following we focus on a single communication resource – i.e., $\boldsymbol{B} = \{\boldsymbol{b}\}$ – that can be used by all computing resources.

11.1.2 Satisfiability Modulo Theory – SMT

Satisfiability Modulo Theories (SMT) enable checking the satisfiability of logical formulas over one or more theories. SMT combines the boolean satisfiability with other background theories, such as, linear arithmetic, arrays, uninterpreted functions, etc. [32]. Thus, the well-known constraint satisfaction problem of propositional satisfiability SAT, where the goal is to decide whether a formula over boolean variables can be made `true` by choosing `true/false` values for its variables, is extended by more expressive logics such as first-order logic. First-order logic formulas consist of logical connectivities, variables, quantifiers, functions and predicate symbols. In SMT, interpretations of some symbols are constrained by a background theory (e.g. linear arithmetics, etc.). SMT provides a *model* as a solution. This model consists of interpretations for the variable, function and predicate symbols that make the formula true. Finding optimized solutions requires either some meta-search techniques or the usage of retractable assertions, enabling for a simple re-execution. We will demonstrate a binary search on top of the provided solution. Further information on satisfiability modulo theories can be found in [32].

11.2 SMT Based Deployment and Scheduling Synthesis

As mentioned in previous sections, we present an approach for jointly generating (safety-related) deployments for multicore architectures using a shared memory (based on different SIL levels) and their corresponding schedules. The presented approach uses a formalization for this joint generation of deployments and schedules, consequently leading to a much higher class of complexity than single scheduling synthesis.
We formalize this problem as a satisfiability problem using boolean formulas and linear arithmetical constraints. We demonstrate that efficient SMT solvers can be used for finding deployments of functions to cores w.r.t. schedulability rules, allocated SIL-levels and soft- and hardware memory constraints.

11.2.1 SMT Solver YICES

YICES [40], [43] is an efficient SMT solver developed at SRI International. It supports a combination of first-order theories, such as arithmetics, uninterpreted functions with equality, bit vectors, arrays, recursive data-types, and more. YICES is able to solve classical SMT problems, namely it decides the satisfiability of propositionally complex formulas in such theories. Further information concerning YICES architecture and algorithms can be found in [44].

11.2.2 Translation to YICES

We propose to solve the defined problem, as previously formulated in the introductory section by using an SMT solver. Therefore, we need to encode the joint scheduling and deployment problem

as a decision problem using boolean formulas with linear arithmetic constraints in order to check the validity. The scheduling problem comprises to find a valid deployment of task to cores, w.r.t. the given constraints. By finding a valid solution, we are then able to generate a deployment and its corresponding schedule that comply to the requirements of an optimized global discrete time base ℓ, as previously defined.

With respect to this goal of finding a deployment including an optimized task and message schedule based on AF3 semantics [35], we implement a binary search finding the shortest latency possible. Using this binary search as a meta search strategy, latency can bounded, making quantifier instantiation terminating, and thus the approach applicable, with respect to boundedness.

11.2.2.1 Assumptions

Our target is to demonstrate that suitable system architectures can be generated efficiently. A suitable system architecture (in our case) includes a safety-oriented deployment and its corresponding time schedule. In order to meet this objective, we need to generate a task and message schedule, meaning to calculate starting times for all tasks $t \in T$ and messages $m \in M$, including valid allocation of tasks to cores (e.g. $t_{i,core} = \{core_{ij}\}$, where $t_i \in T$ and $core_i \in C$).

The following assumptions are used in this approach:

- The precedence graph is defined a priori. The assignments ($T : T \rightarrow 2^M$ and $\rho : M \rightarrow T$) are defined as well. Preemption of tasks is not considered.
- As messages are input respectively outputs of a certain task (corresponding to the given precedence graph G), the precedence relations have to be guaranteed, according the their causality.
- As each message $m \in M$ is transferred via a write and read operation in and out of the shared memory MEM, we distinguish between a write and read part for each message $m \in M$.
- The time which is estimated as communication duration for each write and read message $m \in M$ corresponds to the time for transmitting it over the bus and writing it into the shared memory. The read and write operations are accumulated as task computation time.

Furthermore, while the developed approach does not rely on the following restrictions, in the following for simplification purposes, we expect the computing resources (cores) in the system to be identical concerning computation speed, and only use a single communication bus B and a single shared memory MEM.

11.2.2.2 Definitions

The given precedence graph G comprises several elements: a set of cores, a set of tasks and a set of messages. Thus, we begin by defining type declarations for these precedence graph elements in YICES.

Definition 1 (Tasks):

A task type specification is done using a set of properties in a dedicated data structure that is defined as a record type in YICES input language, as follows:

```
(define-type TASKS (record
start_time :: nat
computation_time :: nat
complete_time :: nat
core :: cores
sil::ASIL
ram::nat))
```

The defined task record stores parameters of a single executable task t_i. The variable ***start_time*** defines the starting, ***computation_time*** the given computation duration and ***complete_time*** the finishing time of a certain task. The variable ***core*** represents the core on the multi-core chip a task is allocated to. Furthermore, each task has a dedicated safety integrity level (***sil***) and needs a certain amount of memory (***ram***).

Definition 2 (Messages):

We specify a message type by using a message record (comparable to Definition 2) that stores the parameter information of a single message m_i. The variable ***start_time*** stores the starting time of a message. The ***communication_duration*** stores the given transmission duration and the ***complete_time*** stores the finishing time of a message.

```
(define-type MESSAGES (record
start_time :: nat
communication_duration :: nat
complete_time :: nat))
```

Definition 3 (Cores):

The set of cores is defined as $\boldsymbol{C} = \{\boldsymbol{c_1}, \ldots, \boldsymbol{c_s}\}$. All cores $\boldsymbol{c} \in \boldsymbol{C}$. This set of cores can be specified using a `scalar`:

```
define-type CORES (scalar 1 2 ... s))
```

where 1, 2, . . . , ***s*** complies to the size of the set ***C***. A record definition, comparable to tasks and messages defines further properties of a core, e.g. a dedicated safety integrity level, w.r.t. a standard (`sil::ASIL`). We use scalar coding instead of a subrange types, because this leads to a reduction of decisions by a factor of 100. The set of tasks ***T*** and messages ***M*** is specified comparatively.

11.2.2.3 Assertions

In the following, we describe the assertions used to solve the given problem.

Assertion 1 (Scheduling Attributes):

As tasks and messages have certain computation or communication times, we need to define these durations as parameters. The goal is to effectively generate optimized and safety related deployments, w.r.t. design rules given by the safety standards. Therefore, a set of additional attributes is needed, w.r.t. safety and memory consumption. These attributes are defined a priori and can be described as follows:

$\models t_i$.computation_t = s,

$\models t_i$.sil = t,

$\models t_i$.ram = u,

$\models m_i$.communication_duration = v

where $t_{i,sil}$ comprises the safety integrity level, $t_{i,ram}$ the necessary memory used by this task and s, t, u, v are the concrete use case-related values.

Assertion 2 (Task Allocation):

Task computation times need to be disjoint, if tasks are allocated to the same computing resource (core), meaning there is only one task at most that is currently using the resource at a time.

$\not\models \exists$ t (t ∈ Time) (

t_i.start_time ≤ t < t_i.complete_time ∧

t_j.start_time < t ≤ t_j.complete_time ∧

t_i.core = t_j.core ∧ $t_i \neq t_j$)

We make use of YICES quantifiers for specifying this constraint:

```
(assert (forall (t1::TASKID t2::TASKID) (or
(or (or ((= t1 t2)
(/ = (select (tasks t1) node) (select (tasks
t2) node)))
(>= (select (tasks t1) start_time) (select
(tasks t2) complete_time)))
(>= (select (tasks t2) start_time) (select
(tasks t1) complete_time)))))
```

Assertion 3 (Precedence Graph):

The goal is to calculate a schedule, where all precedence relations defined in $\boldsymbol{T}$ ($\boldsymbol{t_{send}}$) = {$\boldsymbol{m_{i\_write}}$} and $\boldsymbol{\rho}$($\boldsymbol{m_{i\_read}}$) = {$\boldsymbol{t_{rec}}$} are met. The causality of a task is important in that context. Therefore, a task ($\boldsymbol{t_{send}}$) derived from a weak-causal AF3 component should meet the following timing constraints: The complete time of this task ($\boldsymbol{t_{send}}$) and the start time of the *write* - part of a message ($\boldsymbol{m_i}$) (indicated as $\boldsymbol{m_{i\_write}}$) should be equal and the complete time of the *read* - part of this message should be less or equal to the start time of the receiver task ($\boldsymbol{t_{rec}}$). Note that *write* and *read* – parts of a message can be timely separated, as this is one of the characteristics using shared memory systems:

$$\vDash (m_{i\_write}.\text{start\_time} = t_{send}.\text{complete\_time}) \wedge$$
$$(m_{i\_read}.\text{complete\_time} \leq t_{rec}.\text{start\_time}),$$

where message $\boldsymbol{m_{i\_write}}$, $\boldsymbol{m_{i\_read}} \in \boldsymbol{M}$ and $\boldsymbol{t_{send}}$, $\boldsymbol{t_{rec}} \in \boldsymbol{T}$. We make use of lambda expressions, to denote unnamed functions like this:

```
(define LINK :: (-> TASKID MESSAGEID MESSAGEID
TASKID) (lambda
( source:: TASKID messageSEND:: MESSAGEID
messageREC:: MESSAGEID target:: TASKID)
(... expression)
```

where [*expression*] realizes the arithmetic constraints specified above.

In case a sender task ($\boldsymbol{t_{send}}$) is derived from a strong-causal component this semantic intends a different behavior: The complete time of the sender task ($\boldsymbol{t_{send}}$) should be greater or equal to the start time of the message ($\boldsymbol{m_{i\_write}}$).

$$\vDash (m_{i\_write}.\text{start\_time} >= t_{send}.\text{complete\_time})$$

where message $\boldsymbol{m_{i\_write}} \in \boldsymbol{M}$ and $\boldsymbol{t_{send}} \in \boldsymbol{T}$. The implementation in YICES is comparable to the previous one. Using these different semantics (strong and weak - causal) implies that the design space varies with respect to the possibilities given by the message allocation and causality.

Assertion 4 (Message Allocations):

A disjoint access of messages to a shared communication resource needs to be guaranteed, meaning there is only one message at a time that can be transmitted and written into the shared memory:

$$\nvDash \exists\ t\ (t \in \text{Time})\ (\ m_i.\text{start\_time} \leq t < m_i.\text{complete\_time} \wedge\ m_j.\text{start\_time} < t \leq m_j.\text{complete\_time} \wedge m_i \neq m_j)$$

In case task t_{send} and task t_{rec} are allocated to the same computing resource, the complete time of each task is calculated as follows:

m_i.complete_time = m_i.start_time
iff: t_{send}.core = t_{rec}.core

In case sender task t_{send} and receiver task t_{rec} are allocated to different computing resources, the complete time is calculated as follows:

m_i.complete_time = m_i.start_time + m_i.communication_duration
iff: t_{send}.core ≠ t_{rec}.core

We use a function of YICES input language to specify this calculation:

```
((define DURATION :: (-> TASKID MESSAGEid
TASKID) (lambda
( source:: TASKID messageSEND:: MESSAGEID
messageREC:: MESSAGEID target:: TASKID)
(if (= (select (tasks source) node) (select
(tasks target) node))
(select (messges message) start_time)
(+ (select (messages message)
start_time) (select (messages message)
computation_time)))))
```

Furthermore, local memory is needed by the set of tasks T on their allocated cores C. Thus, a core $core_i$ provides enough memory for all tasks t_i that are allocated to this core. We specify this constraint using a comparable functions as the link function specified previously.

Assertion 5 (Safety Integrity Level)

In order to provide a formalization of the joint generation of schedules and safety-critical deployments, we use SIL-annotations of components, to specify deployment constraints:

⊨ ($t_{i,sil}$ <= $core_{i,sil}$)

where task $t_{i,sil}$ comprises to all task $t_i \in T$ with $\eta(core_i) = \{ t_i \}$, meaning all tasks that are allocated to the core $core_i \in C$:

```
(assert (forall (t:: TASKID) (<= (select
(tasks t) sil)
(select (nodes (select (tasks t) node))
sil))))
```

11.2.2.4 Correctness Properties

Finally, given system requirements are specified as a correctness property for the given SMT - based joint scheduling and deployment generation approach. We specify the global discrete time base as a length | ℓ |. The calculated deployment should contain a schedule using this length | ℓ | (e.g. 100 time units).

```
(assert (<= (- endLatency startLatency) 100))
```

The check command is used to check whether the current logical context is satisfiable or not. If the joint problem of deployment and scheduling is satisfiable using the constraints imposed by the given end-to-end system requirement, a solution model is given.

11.3 Description based on an Example

In the following we present an automotive use case to demonstrate the usability of the presented approach using a real world example.

11.3.1 Adaptive Cruise Control (ACC) – System

The Adaptive Cruise Control (ACC) is an automotive use case. The ACC automatically adjusts the traveling speed of an automotive vehicle by controlling the acceleration and breaking momentum, based on a driver-defined reference speed, the current speed as well as the distance to a (possibly present) leading vehicle. Figure 6 shows how the ACC system is modeled using our research CASE tool AF3, with an architecture consisting of 5 main components (speed and distance plausibilization, speed- and distance-based control, (de-)acceleration computation).

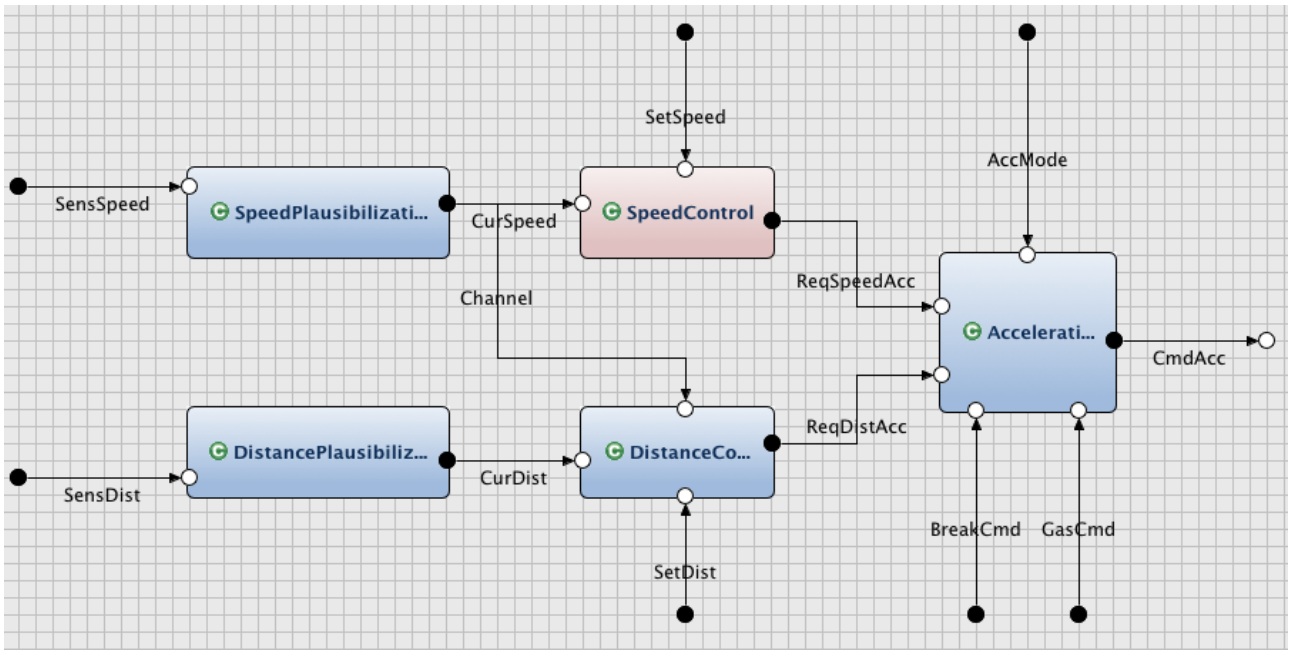


Figure 36: Automotive Use Case: Adaptive Cruise Control (ACC) model in AF3 [36]

AF3 provides different component semantic with respect to timing: the notion of strong and weak causality. This approach supports both of them. Therefore, we specify all components as weak-causal, except of one component, named ”*DistancePlausibilization*”. This component is specified as strong-causal. Furthermore, there are different levels of criticality. Component ”*DistancePlausibilization*” and component ”*SpeedPlausibilization*” have an annotated safety integrity level (SIL) of 2, while all other components are of SIL 1, with higher SIL numbers denoting a higher criticality level.

11.3.2 ACC Schedule Synthesis

We demonstrate the proposed approach by using the AF3 system model of the ACC from the previous section. This model is transformed into a scheduling model represented by an extended precedence graph ***G***. We use the scheduling model as a basis for the presented SMT-based scheduling approach.

For the given ACC use case, we generate a set of 5 tasks $\boldsymbol{T} = \{\boldsymbol{t}_{SpeedPlausibilization}, \boldsymbol{t}_{DistancePlausibilization}, \ldots, \boldsymbol{t}_{Accerelation}\}$ according to the given components and a set of 13 messages $M = \{\boldsymbol{m}_{SensSpeed}, \ldots, \boldsymbol{m}_{CmdAcc}\}$. Furthermore, based on the technical architecture of the ACC, we generate to 2 cores $\boldsymbol{C} = \{\boldsymbol{core}_1, \boldsymbol{core}_2\}$, a shared memory ***MEM*** and an avalon bus ***B***.

In the next step, assertions are generated w.r.t. the defined system attributes, e.g. the computation time (`computation_time`) or the different safety integrity levels (`SIL`) for each task. This is done for all elements in the scheduling model.

The YICES `assert` command is used for specifying the system attributes:

```
(assert (= (select (task
SpeedPlausibilization) computation_time) 10))
(assert (= (select (task
SpeedPlausibilization) sil) 1))
...
(assert (= (select (m CurSpeed)
communication_duration) 2))
...
```

In a next step, we define constraints that are imposed by precedence relations defined in the generated precedence graph ***G***. As tasks and messages cannot be scheduled in an arbitrary order, precedence relations are defined by the functions ***T*** and ***ρ*** are used to guarantee all precedence relations in ***G*** (e.g. $\boldsymbol{T}(\boldsymbol{t}_{DistancePlausibilization}) = \{\boldsymbol{m}_{CurSpeed}\}$ and $\boldsymbol{\rho}(\boldsymbol{m}_{CurSpeed}) = \{\boldsymbol{t}_{DistanceControl}\}$).

As a system requirement, the goal is to minimize the logical tick duration $|\ell|$. Therefore, we demand for a schedule with a latency of less than 100 time units, as discussed in section 11.2.2.4.

11.3.3 Satisfied Solution Model

The function of a SMT solver is to check the satisfiability of logical formulas over one or more theories. The solution model provided by the SMT solver is a valid deployment for the given deployment problem under consideration. However, the SMT solver outputs one solution that

fulfills the defined constraints. A valid solution, a model, consists of interpretations for the variables, functions and predicate symbols that makes the formula true. For analysis and demonstration of operation, we invoke YICES on the given Adaptive Cruise Control (ACC) - System.

Solution Model given by YICES SMT - Solver:

```
(= duration 34)
(= (task DistancePlausibilization)
(mk-record
start_time :: 0
computation_time :: 10
core :: 0))
ram :: 250
sil :: 1
(= (task SpeedPlausibilization)
(mk-record
start_time :: 12
computation_time :: 10
core :: 0))
ram :: 100
sil :: 1
...
(= (message CurSpeed_write
(mk-record
start_time :: 22
communication_duration :: 2
(= (message CurDist
...
```

The solution model comprises all defined elements. This includes a solution for the defined task and message scheduling problem. Furthermore, the following allocation of tasks to cores has been generated, w.r.t. a priori defined safety integrity level and memory constraints: $\eta(core_0) = \{t_{DistancePlausibilization}, t_{SpeedPlausibilization}\}$ and $\eta(core_1) = \{t_{SpeedControl}, t_{DistanceControl}, t_{Accerelation}\}$.

For instance, task $t_{DistancePlausibilization}$ has a calculated ***start_time*** of 0 time units. Task $t_{SpeedPlausibilization}$ starts at 12 time units and is allocated to core $core_0$. Message $m_{CurSpeed\_write}$, for instance, has an allocated starting time of 22 time units.

Thus, based on the solution model provided, we are able to extract an integrated task and message schedule $\gamma = \{t_i \mapsto \gamma_i, \ \forall\ t_i \in T\}$ that is integrated into the AF3 system model.

$$\gamma = \{t_{SpeedPlausibilization} \mapsto \langle 12; \{22, 0\} \rangle,$$
$$\{t_{DistancePlausibilization} \mapsto \langle 0, \{10, 12\} \rangle,$$
$$\{t_{SpeedControl} \mapsto \langle 2, \{\} \rangle,$$
$$\{t_{DistanceControl} \mapsto \langle 14, \{\} \rangle$$
$$\{t_{Accerelation} \mapsto \langle 24, \{\} \rangle\}$$

The calculated schedule is illustrated in figure 7. As previously explained, tasks $\boldsymbol{t_{SpeedPlausibilization}}$ and $\boldsymbol{t_{DistancePlausibilization}}$ are allocated to the same core $\boldsymbol{core_0}$. The other tasks are allocated to the core $\boldsymbol{core_1}$. Hence, the execution ordering needs to be disjoint for that node.

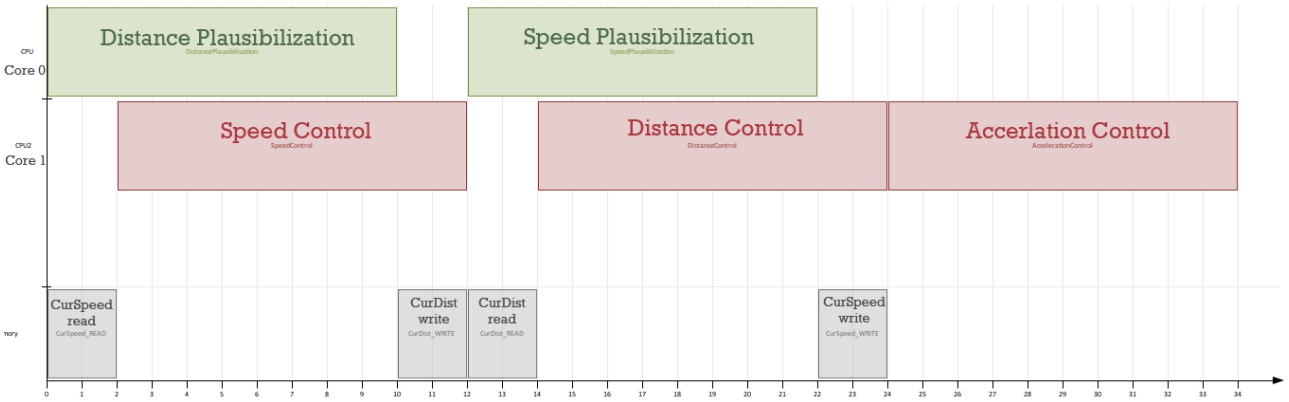


Figure 37: Optimized Schedule of Active Cruise Control in AF3 [36]

Thus, the optimized global discrete time base $\boldsymbol{\ell}$ for the given AF3 system model under consideration is calculated to be 34 time units. YICES SMT Solver needs less than 100 msec to calculate a valid solution. Using a binary search for optimizing the duration $|\boldsymbol{\ell}|$ of the global discrete time base, YICES uses less than 1 sec.

12 Conclusions

This second version of deliverable D3.3.3 describes concepts and methodologies for the assessment of architectures in context of functional safety. Four different methodologies are presented: First methodology addresses model-based hardware safety assessment on different level of abstraction. The second methodology is in focus of coverage and consistency checks for the safety case. Third methodology describes performing common-cause analysis in the geometric perspective using physical properties and environmental conditions. Methodology 4 provides multi-criteria deployment optimization and schedule generation. Additionally, in the work package 4 “Technology platform”, the methodologies are in focus of implementation work.

13 References

[1] International Standards Organization, ISO 26262 Standard, “Road Vehicles - Functional Safety,” http://www.iso.org/, 2011.

[2] International Electrotechnical Commission, IEC 61508 Standard, “Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems,” http://www.iec.ch/functionalsafety, 2009.

[3] Adler, N., Otten, S., Mohrhard, M., and Müller-Glaser, K.-D., “Rapid Safety Evaluation of Hardware Architectural Designs Compliant with ISO 26262,” in Rapid System Prototyping (RSP), 24rd IEEE International Symposium on, 2013, pp. 66–72.

[4] Börcsök, J., “Functional Safety: Basic Principles of Safety-Related Systems,” Hüthig Verlag, 1. Edition 2007.

[5] International Electrotechnical Commission, "Technical Report: Reliability data handbook - Universal model for reliability prediction of electronics components, PCBs and equipment," IEC Standard TR 62380, Rev. Aug. 2004.

[6] Departement of Defense, "Military handbook: electronic reliability design handbook," MIL-HDBK-338B, Rev. Oct. 1998.

[7] FIDES Group, "Reliability Methodology for Electronic Systems," FIDES guide 2009 edition A, Rev. Sept. 2010.

[8] International Electrotechnical Commission, IEC 61025 Standard, “Fault tree analysis (FTA),” 2006.

[9] Veseley, W. E. et al., NUREG-0492 Standard, “Fault Tree Handbook,” 1981.

[10] Veseley, W. E. et al., “Fault Tree Handbook with Aerospace Applications,” NASA Office of Safety and Mission Assurance, 2002.

[11] Departement of Defense, “Military Standard: Procedures for Performing a Failure Mode, Effects and Criticality Analysis,” MIL-STD1629, 1974.

[12] Grebe, J. C., Goble, W. M., “FMEDA – Accurate Product Failure Metrics,” FMEDA Development Paper, Rev. 1.1, 2007.

[13] Collett, R. E. and Bachant, P. W., “Integration of BIT Effectiveness with FMECA,” Proceedings of the Annual Reliability and Maintainability IEEE Symposium, New York, 1984.

[14] Moore Products Co., “Safety Manual for QUADLOG,” 1997.

[15] Leitner-Fischer, F. and Leue, S., "The QuantUM Approach in the Context of the ISO Standard 26262 for Automotive Systems," Technical Report soft-11-01, University of Konstanz, 2011.

[16] Jeon, S.-H., et al., "Automotive hardware development according to iso 26262," presented at 13th International Conference on Advanced Communication Technology (ICACT), Korea, Feb. 13-16, 2011.

[17] Bellotti, M., and Mariani, R., "How future automotive functional safety requirements will impact microprocessors design," Microelectronics Reliability, 2010, doi:10.1016/j.microrel.2010.07.041

[18] Sinha, P., "Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives," Reliability Engineering and System Safety, 2011, doi:10.1016/j.ress.2011.03.013.

[19] Svancara, K., Forbes, W., Priddy, J., Kudanowski, M. et al., "Experience with the second method for eps hardware analysis: "evaluation of each cause of safety goal violation due to random hardware failures", " presented at VDA Automotive SYS Conference on Quality and Functional Safety Management for Automotive software-based Systems, Germany, May 14-16, 2012.

[20] Svancara, K., Priddy, J., Lovric, T., Miller, J. et al., "Advantages of the Alternative Method for Random Hardware Failures Quantitative Evaluation - a Practical Survey for EPS," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 6(2):377-388, 2013, doi:10.4271/2013-01-0190.

[21] Papadopoulos, Y. et al., "Engineering failure analysis and design optimisation with hip-hops," Engineering Failure Analysis, vol. 18, no. 2, pp. 590 – 608, 2011, the Fourth International Conference on Engineering Failure Analysis Part 1.

[22] M. Walker, Y. Papadopoulos, D. Parker, H. Lönn, M. Törngren, D. Chen, R. Johansson, and A. Sandberg, "Semi-automatic fmea supporting complex systems with combinations and sequences of failures," SAE Int. J. Passeng. Cars - Mech. Syst. 2(1), pp. 791–802, 2009.

[23] Adler, N., Hillenbrand, M., Müller-Glaser, K.-D., Metzker, E., and Reichmann, C., "Graphically notated fault modeling and safety analysis in the context of electric and electronic architecture development and functional safety," in Rapid System Prototyping (RSP), 23rd IEEE International Symposium on, 2012, pp. 36–42.

[24] AUTOSAR, "AUTOSAR Project Objectives V3.0.0, R4.0 Rev. 3," Rev. Dec. 2011.

[25] ATESST2 Consortium, "EAST-ADL Domain Model Specification - Deliverable D4.1.1," Rev. June 2010.

[26] Cuenot, P., Adler, N., and Otten, S., SAFE Project, "Deliverable D3.2.2b: Proposal for extension of meta model for hardware modeling," 2013.

[27] Adler, N., Otten, S., Cuenot, P., and Müller-Glaser, K., "Performing Safety Evaluation on Detailed Hardware Level according to ISO 26262," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 6(1):102-113, 2013, doi:10.4271/2013-01-0182.

[28] Vector Informatik GmbH, "PREEvision User Manual Version 6.0.1", 2013.

[29] Innal, F., Dutuit, Y., Rauzy, A., and Signoret, J.-P., "New insight into the average probability of failure on demand and the probability of dangerous failure per hour of safety instrumented systems", Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability June 1, 2010 vol. 224 no. 2 75-86, 2009, doi:10.1243/1748006XJRR278.

[30] Baumgart, A., „A Contract-Based Installation Methodology for Safety-Related Automotive Systems", SAE Technical Paper 2013-01-0192, April 2013

[31] Damm, W., Baumgart, A., Böde, E., Büker, M., Ehmen, G., Gezgin, T., Henkler, S., Hungar, H., Josko, B., Oertel, M., Peikenkamp, T., Reinkemeier, P., Stierand, I. and Weber, R., "Architecture Modeling", OFFIS, March 2011.

[32] L. de Moura and N.Bjoerner, "Satisfiability modulo theories: An appetizer," in SBMF, 2009, pp. 23–36.

[33] H. Gall, "Functional safety IEC 61508 / IEC 61511 the impact to certification and the user," in Proceedings of the 2008 IEEE/ACS International Conference on Computer Systems and Applications, ser. AICCSA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 1027–1031. [Online]. Available: http://dx.doi.org/10.1109/AICCSA.2008.4493673

[34] The SAFE-E Consortium. Deliverable D4.4.a "First version of plug-in for safety and multi criteria architecture modeling and benchmarking". EUROSTARS. 2013.

[35] AutoFOCUS3. Fortiss GmbH. 2013, af3.fortiss.org.

[36] Sebastian Voss, Bernhard 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.

[37] Sebastian Voss, Antoaneta Kondeva, Daniel Ratiu, Bernhard Schätz, “Seamless Model-based Development of Embedded Systems with AF3 Phoenix”. Tutorial at the 20th Annual IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS) 2013.

[38] SPES-XT Project Consortium. spes2020.informatik.tu-muenchen.de/spes_xt-home. 2013.

[39] Z3 SMT Solver, Microsoft Research, Microsoft Corportation. research.microsoft.com/en-us/um/redmond/projects/z3. 2013.

[40] YICES SMT Solver, SRI Tools, Stanford Research Institute. yices.csl.sri.com. 2013.

[41] Sebastian Voss, Johannes Eder, Florian Hölzl, Bernhard Schätz. “An integrated Design Space Exploration Approach”. Submitted.

[42] Automotive, Railway and Avionics Multicore Systems – ARAMiS Project Consortium.. www.projekt-aramis.de. 2013.

[43] B. Dutertre and L. de Moura, “The yices smt solver,” Computer Science Laboratory, SRI International, Tech. Rep.

[44] B. Dutertre and L. de Moura, “Fast linear-arithmetic solver for dpll(t),” in Proc. 18th Computer-Aided Verification conference, ser. LNCS, vol. 4144. Springer-Verlag, 2006, pp. 81–94.

14 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 the 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.