Project acronym:	*SafeAdapt*
Project title:	*Safe Adaptive Software for Fully Electric Vehicles*
Grant Agreement number:	608945
Coordinator:	*Dr.-Ing. Dirk Eilers*
Funding Scheme:	*FP7-2013-ICT-GC*

Deliverable 4.2

Specification of the Design Process for Safe Adaptive Embedded Systems and Tool Support for V&V Adaptive System Behaviour

Due date of deliverable:	30th June 2015
Actual submission Date:	1st July 2015
Lead beneficiary for this deliverable:	CEA

Dissemination level		
PU	**Public**	**X**
PP	**Restricted to other programme participants (including the Commission Services)**	
RE	**Restricted to a group specified by the consortium (including the Commission Services)**	
CO	**Confidential, only for members of the consortium (including the Commission Services)**	

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013)

Document Information	
Title	Specification of the design process for safe adaptive embedded systems and tool support for V&V adaptive system behaviour
Creator	TECNALIA: Mª Carmen Palacios, Alejandra Ruiz, Maite Álvarez, Adrian Martin, Garazi Juez Uriagereka
Description	This document describes the design process for safe adaptive embedded systems as well as the tool support for V&V of adaptive system behaviour for the SafeAdapt project.
Publisher	CEA
Contributors	CEA: Ansgar Rademacher, Önder Gürcan, Reda Nouacer, Gilles Mouchard Fraunhofer: Dulcineia Oliveira da Penha, Philipp Schleiss, Gereon Weiss Siemens: Kai Hoefig, Cornel Klein TTTech: Andreas Eckel Revision – Pininfarina: Sandro Morero, Duracar: Ken Lam, Ficosa: Andrea Saccagno
Language	en-GB
Creation date	05/01/2015
Version number	1.0
Version date	30/06/2015
Audience	☐ internal ☒ public ☐ restricted

Table of Contents

List of Figures

List of Tables

Executive Summary

This document defines the development, verification and validation processes for safe adaptive embedded systems, enabling the certifiability of adaptive embedded systems in the automotive domain with special focus on fully electric vehicles regarding ISO 26262 (International Organization for Standardization (ISO), 2011).

One of the main SafeAdapt project objectives is to empower developers to create safety-critical software more efficiently by reducing costs and coping with certification requirements. As said above, the proposed SafeAdapt Methodology fully addresses the ISO 26262 standard and it is supported by model-based tools used for different steps from requirements to detailed software and hardware implementation and validation. On the one hand, system and components design is based on well-known standards such as UML (UML), EAST-ADL (EAST-ADL), SysML (SysML), MARTE (MARTE), ARText (ARText) and AUTOSAR (AUTOSAR). Different tools are included since everyone has different strengths with respect to modelling, algorithmic capabilities, etc. On the other hand, early verification and validation is applied during the design process of the safe adaptive system; especially important in safety-critical systems where dynamic reconfiguration might impair safety, verification and validation throughout the design process.

This document includes:

- An introduction for the Task 4.2 “Specification of the design process for safe adaptive embedded systems and tool support for V&V adaptive system behaviour”
- Terms and definitions needed to understand the concepts described in this document.
- Detailed description of the tools included in the SafeAdapt Tool Chain.
- Work-flow guidelines, according to the ISO 26262 standard, for the modelling, design, verification and validation of adaptive critical systems.
- The most common ways of using the SafeAdapt Methodology by industry.

1 Introduction

The promising advent of fully electric vehicles also means a shift towards fully electrical control of existing and new vehicle functions. In particular, critical X-by-wire functions require sophisticated redundancy solutions. As a result, the overall Electric/Electronic (E/E) architecture of a vehicle is becoming even more complex and costly.

The main idea of SafeAdapt (Safe Adaptive Software for Fully Electric Vehicles) is to develop novel architecture concepts based on adaptation to address the needs of a new E/E architecture for Fully Electric Vehicles (FEVs) regarding safety, reliability and cost-efficiency. This will reduce the complexity of the system and the interactions between its functions by generic, system-wide fault and change handling. It also enables extended reliability despite failures, improvements of active safety, and optimized resources. This is especially important for increasing reliability and efficiency regarding energy consumption, costs and design simplicity.

SafeAdapt follows a holistic approach for building adaptive systems in safety-critical environments that comprises methods, tools and building blocks for safe adaptation. The technical approach builds on a SafeAdapt Platform Core, encapsulating the basic adaptation mechanisms for re-allocating and updating functionalities in the networked, automotive control systems. This will be the basis for an interoperable and standardized solution for adaptation and fault handling in AUTOSAR (AUTOSAR). Although it is not initially covered by the project, the SafeAdapt approach also considers functional safety with respect to the ISO 26262 standard (International Organization for Standardization (ISO), 2011) to support the certification of safety-critical systems in the e-vehicle domain.

SafeAdapt provides an integrated approach for engineering such adaptive, complex and safe systems, ranging from tool chain support, reference architectures, modelling of system design and networking, up to early validation and verification. For realistic validation of the adaptation and redundancy concepts, an actual vehicle prototype with different and partly redundant applications is developed.

1.1 Document scope

The purpose of this document is to define the development, verification and validation processes for safe adaptive embedded systems, that is, which tool should be used for which purpose and how the different tools interact within a common tool chain. Thus, enabling the certifiability of adaptive embedded systems in the automotive domain with special focus on fully electric vehicles regarding ISO 26262.

One of the main SafeAdapt project objectives is to empower developers to create safety-critical software more efficiently by reducing costs and coping with certification requirements. As said above, the proposed SafeAdapt Methodology fully addresses the ISO 26262 standard and it is supported by tools used for different steps from requirements to detailed software and hardware implementation and validation. On the one hand, system and components design is based on well-known standards such as UML (UML), EAST-ADL (EAST-ADL), SysML (SysML), MARTE (MARTE), ARText (ARText) and AUTOSAR. Different tools are included since everyone has different strengths with respect to modelling, algorithmic capabilities, etc. On the other hand, early

verification and validation is applied during the design process of the safe adaptive system; especially important in safety-critical systems where dynamic reconfiguration might impair safety, verification and validation throughout the design process. It should be considered that functional, non-functional, and trustworthiness properties need to be ensured even during reconfiguration transitions.

1.2 Document outline

The remainder of the report is structured as follows:

- Section 2 includes terms and definitions that provide the necessary background required to properly understand the concepts described in this document.
- In section 3, the tools included in the SafeAdapt Tool Chain are presented. These tools support the modelling, design, verification and validation of adaptive critical systems. In addition, the tools are depicted along the ISO 26262 lifecycle.
- Section 4 provides detailed insight into the tools. For every tool it is provided a general description, its position along the V-lifecycle and the managed inputs and outputs artefacts.
- Section 5 provides the work-flow guidelines for the design process and support of verification and validation phases. The aim is to help users to start working with the SafeAdapt Tool Chain, guiding their actions according to the ISO 26262 standard.
- In section 6 the most common ways of using the SafeAdapt Methodology by industry are presented.
- Section 7 drafts final conclusions.

2 Terms and Definitions

Term	Definition
Architecture	Representation of the structure of the item or functions or systems or elements that allows identification of building blocks, their boundaries and interfaces, and includes the allocation of functions to hardware and software elements (ISO 26262).
Automotive Safety Integrity Level	An Automotive Safety Integrity Level (ASIL) represents an automotive-specific risk-based classification of a safety goal as well as the validation and confirmation measures required by the standard to ensure accomplishment of that goal.
Component-ISO	ISO 26262 defines a Component-ISO as a non-system level element that is logically and technically separable and is comprised of more than one hardware part or more software units. A component is part of a system.
Element	System or part of a system, including components, hardware, software, hardware parts, and software units -- effectively, anything in a system that can be distinctly identified and manipulated.
Error	Discrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition.
Failure	Termination of the ability of an element to perform a function as required. Note: Since an element's specification defines its required function, the standard recognizes incorrect specification as a potential a source of failure.
Fault	Abnormal condition that can cause an element or an item to fail.
Functional Safety	Absence of unreasonable risk due to hazards caused by malfunctioning behaviour of Electrical/Electronic systems.
Hazardous Event	A hazardous event is a relevant combination of a vehicle-level hazard and an operational situation of the vehicle with potential to lead to an accident if not controlled by timely driver action.
Item	Item is used to refer to a specific system or array of systems that implements a function at vehicle level to which the ISO 26262 Safety Life Cycle is applied. That is, the item is the highest identified object in the process and is thereby the starting point for product-specific safety development under this standard.
Malfunctioning Behaviour	Failure or unintended behaviour of an item with respect to its design intent. Hazard Potential source of harm caused by malfunctioning behaviour of the item.

Safety Goal	A safety goal is a top-level safety requirement that is assigned to a system, with the purpose of reducing the risk of one or more hazardous events to a tolerable level.
Safety Requirement	Safety requirements include all safety goals and all levels of requirements decomposed from the safety goals down to and including the lowest level of functional and technical safety requirements allocated to hardware and software components.
Validation	Process of evaluating the system impact e.g., on safety. That is, validation checks and tests whether the system "does what it was designed for“, quoted by the performance indicators (based on user needs).
Vehicle	Reference to a passenger car that can be either simulated or a real vehicle.
Verification	Describes the test of a system/function against its requirements. Determination of completeness and correct specification or implementations of requirements from a phase or subphase.

3 Tool Chain

3.1 Description

SafeAdapt provides tool support and a methodology to ensure that innovative architecture solutions are equally assisted in the design process. The SafeAdapt Tool Chain includes modelling, design and validation support. This tool uses a model-based design flow, which is complemented by pre-existing AUTOSAR tool-chains, to design adaptivity. Moreover, the SafeAdapt approach enables early verification and validation of the systems non-functional requirements such as adaptability.

In brief, the SafeAdapt Tool Chain is composed of the following tools presented in alphabetical order in Table 1.

Tool	Purpose
Arctic Studio (ARCCORE)	AUTOSAR modelling and code generation.
ARTOP	AUTOSAR reference implementation. Based on Eclipse
AUTOSAR Gateway	Export AUTOSAR from UML/EAST-ADL models (Papyrus add-on)
composeR (SIE)	Safety analysis tool compliant to FTA analysis as defined by various standards such as IEC 61508 or ISO 26262.
Dynacar RT (TEC)	Help during SW and HW testing phase. Configurable vehicle model running in a real-time system. Models from third parties (Simulink, Dymola) can be integrated on the same platform.
ERNEST (ESK)	Verification and validation of non-functional properties of networked embedded systems at early design stages.
FMEDAexpress (SIE)	Safety analysis tool for FME(D)A analysis according to IEC 61508 or ISO 26262.
Papyrus (CEA)	General purpose UML modelling tool supporting SysML (including SysML specific diagrams), MARTE and EAST-ADL profiles. Moreover, it offers several possibilities to customize the user interface.
Prossurance (TEC)	Safety assurance management system. Prossurance supports compliance assessment and certification of safety-critical products. Construction of safety cases.
Qompass (CEA)	Design tool for model transformation and code generation. Qompass helps to deploy component-based systems taking into account SW and HW architecture. The tool supports realizing arbitrary interactions between software components. Qompass also supports a separation of concerns by enabling containers that embed the original component and

	intercept its communication with the environment as well as offering additional service.
Sophia	Safety analysis, supports FTA analysis (Papyrus add-on)
TTE-Tools (TTT)	Tool for generation of a valid network configuration for end systems and switches for time-triggered-, rate-constrained and best-effort Ethernet based networks.
UNISIM-VP (CEA)	Cross-platform open source simulation environment. Its purpose is to be used during co-design, integration and validation of hardware/software systems. The simulation environment comprises a set of tools and services such as program loaders, OS ABI translators, instrumentation and graphical debugger.
XMT (SIE)	Model oriented system design.

Table 1. SafeAdapt tools

3.2 Tools along the ISO 26262 lifecycle

Figure 1 presents an overview of the SafeAdapt tools regarding the design, implementation and V&V flow.

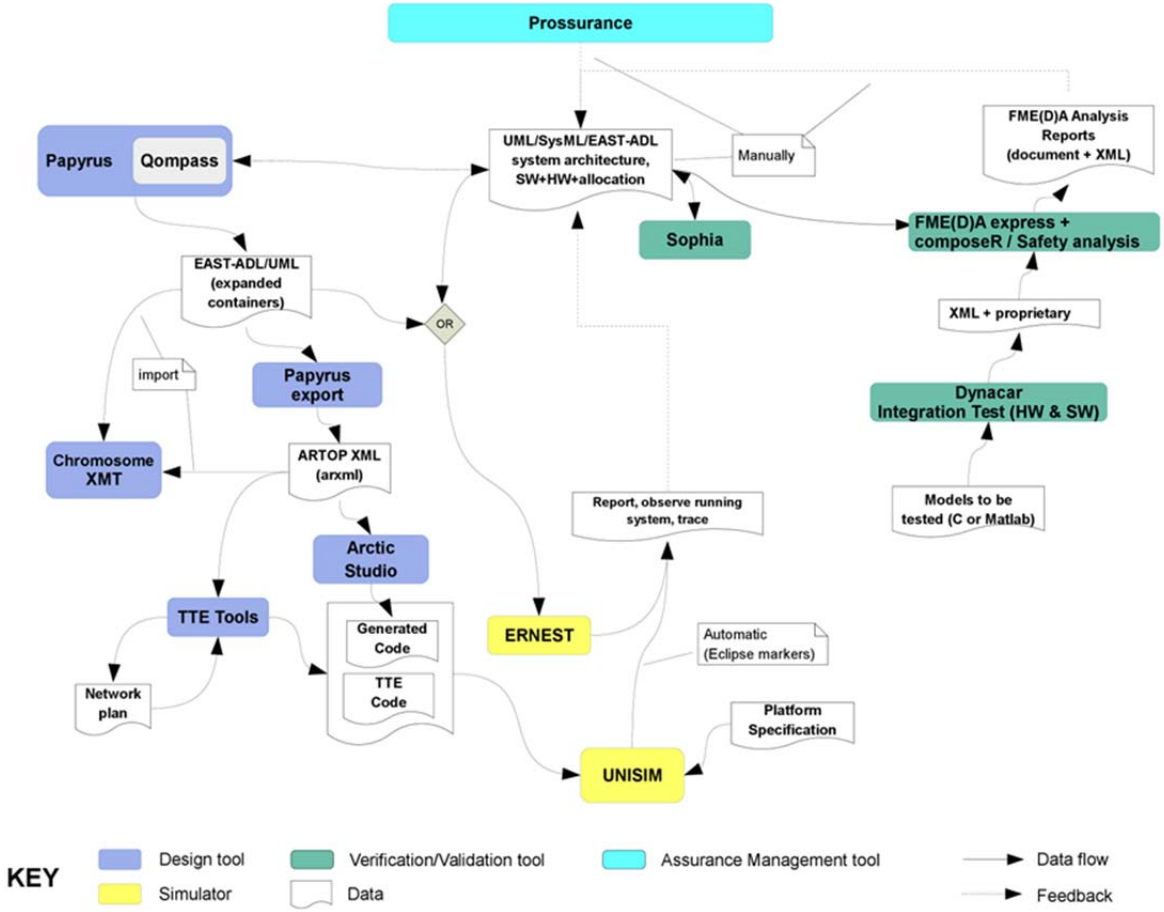


Figure 1: SafeAdapt tools overview

4 Detailed Description of Tools

4.1 Arctic Studio

4.1.1 General description

The Arctic Studio tool chain provides a complete software development environment for automotive embedded software solutions based on the open industry-leading standard AUTOSAR. The tool chain supports all stages of an automotive Information and Communication Technology (ICT) project and provides tools for different types of tasks, such as application development, embedded platform development, and system integration.

4.1.2 Tool along the lifecycle.

Arctic Studio is applied during the development phase of the lifecycle. This tool addresses ISO 26262, part 6 Product development at software level, phase 6-7 Software architectural design.

4.1.3 Artefacts metamodel

- Full access to AUTOSAR arxml files through the Artop open source project
- Wizards for creating AUTOSAR projects and AUTOSAR files
- Full support for handling configurations split into multiple files
- AUTOSAR viewer with possibility to walk through the AUTOSAR configuration in a tree view
- Support of AUTOSAR standard version 4.0.2, 4.0.3 and 4.1.1

4.1.4 Inputs

As an input Arctic Studio requires AUTOSAR configuration files that where either imported using arxml files or created inside of ArcticStudio.

Furthermore, ArcticStudio supports the import of "Software Component Description" files (ARText) and provides importers for communication matrices in form of AUTOSAR ECU extract and CANdb files.

4.1.5 Outputs

The end result of the Arctic Studio tool chain is a configuration dependent RTE in form of C-code and a compiled, linked, and executable binary image (ELF) for the target platform.

4.2 composeR

4.2.1 General description

composeR is a safety analysis tool compliant to FTA analyses as defined by various standards such as IEC 61508 and ISO 26262.

composeR works standalone with no other information provided, but it is also able to do a compositional safety analysis based on development artefacts such as SysML IBD or UML Composite Structure Diagrams (CSD).

composeR is based on ESSaRel (The ESSarel research project & tool) and it has been developed with Eclipse/Magic Draw Plugin technologies.

4.2.2 Tool along the lifecycle.

This tool addresses the following parts of ISO 26262:

- Part 3. Concept phase, sub-phase 3-8 Functional safety concept (supporting the derivation of functional safety requirements).
- Part 4. Product development at system level phase, sub-phases 4-7 System design (system design verification), and 4-9 Safety validation.
- Part 5. Product development at hardware level phase, sub-phases 5-9 Evaluation of safety goal violations due to random hardware failures.
- Part 6. Product development at software level phase, sub-phase 6-7 Software architectural design (efficiency verification of the safety mechanisms).
- Part 9. ASIL-oriented and safety-oriented analyses, sub-phases 9-7 Analysis of dependent failures and 9-8 Safety analyses.

4.2.3 Artefacts metamodel

The information is stored as XML file. In addition, import and export utilities are available (information format can be both EMF models and XML file).

4.2.4 Inputs

composeR requires as input (automated) XML-based information on design artefacts (IBD/CSD) on .mdzip files and (manually) information on safety behaviour using FTA/FMEA.

4.2.5 Outputs

composerR delivers as output (regarding the analysis) top event probabilities, minimal cut sets or FIT rates for part count. To sum up, composeR enriches XML file with failure information to be used in different tools.

4.3 Dynacar RT

4.3.1 General description

The tool has been developed to help during the SW and HW testing phases. It is a configurable vehicle dynamics model running in a real time (1 millisecond execution) system (PXI from National Instruments). Models from third party software (Simulink, Dymola) can be integrated on the same platform. The system has several outputs which can be used as virtual sensors. These sensors can be connected to other software (SIL) or Hardware (HIL).

The model runs on a real time platform from National Instruments (PXI) and has been created using Labview RT. It runs over the Veristand platform which helps in model integration and connectivity.

Tecnalia develops the SW and uses it for prototyping.

4.3.2 Tool along the lifecycle

Dynacar RT can be used for SIL and HIL testing:

- SIL: The model can be connected with controller and vehicle subsystem virtual models for software verification.
- HIL: Due to the PXI connection capabilities it can be easily connected to real components and ECU controllers in combination with virtual components for HIL testing.

This tool addresses the following parts of ISO 26262:

- Part 4. Product development at system level phase, sub-phase 4-9 Safety validation.
- Part 6. Product development at software level phase, sub-phases 6-9 Software unit testing (test environment for software unit testing – SIL & HIL), 6-10 Software integration and testing (test environment for software integration testing – SIL & HIL) and 6-11 Verification of software safety requirements (test environment).

4.3.3 Artefacts metamodel

The model inputs/outputs can be linked internally using Veristand (from National Instruments) in case of being provided by another external model installed on the same HW or can be linked using any kind of communication, CAN, PROFIBUS, FLEXRAY, ETHERNET, Analog/Digital signals, etc. All the outputs can be selected and stored internally on the real time controller (PXI) in TDMS format (proprietary format from National Instruments for logging data) and can be easily exported into Matlab, Excel, etc. Then the tests can be analysed through the logged data.

4.3.4 Inputs

Dynacar RT is a virtual rolling chassis with component models (engine, transmission, brakes, steering, suspension and tires). Users can substitute the default models with their own models.

Several inputs are available for model integration and vehicle control that are sorted according to the following components:

- Aerodynamics
- Auto
- Brakes
- Controls
- Engine
- Friction
- Tires
- Gearbox

- Powertrain
- Torque Transfer Device

4.3.5 Outputs

Several outputs are available for virtual sensors and models which are sorted according to the following components:

- Aerodynamics
- Brakes
- Chassis motion
- Controls
- Logitech steering wheel (G27)
- Ground and Road
- Powertrain
- Simulation
- Steering
- Tires
- Wheels

4.4 ERNEST

4.4.1 General description

ERNEST is a simulation-based analysis tool for the verification and validation of non-functional properties of networked embedded systems. ERNEST checks, for instance, if given timing constraints are fulfilled.

The framework is compatible with EAST-ADL and an AUTOSAR-Connection is in prototype state.

The simulation framework is written in C++ and uses SystemC.

The whole tool is delivered as Eclipse plug-ins and uses standard Eclipse technologies like EMF, Ecore, Xtend and CDT.

4.4.2 Tool along the lifecycle.

ERNEST is used at early design stages in order to evaluate the system architecture and verify the system's non-functional properties.

This tool addresses the following parts of ISO 26262:

- Part 4. Product development at system level phase, sub-phases 4-7 System design (system design verification) and 4-9 Safety validation

4.4.3 Artefacts metamodel

The ERNEST framework is based on a specific meta-model which describes the system architecture and constraints and it is developed using the Ecore standard. Architectures description modelled in EAST-ADL or AUTOSAR can be transformed into the ERNEST input model format via model-to-model-transformations.

4.4.4 Inputs

ERNEST uses architecture descriptions and constraints are described via the ERNEST format (metamodel), as input for realizing its analysis. This information can be generated from e.g. EAST-ADL or AUTOSAR models via model-to-model transformations.

4.4.5 Outputs

As a result of the analysis after the simulation, ERNEST provides a visualization mechanism with an overview of the given constraints. The visualization shows at which point during the simulation timing constraints have not been fulfilled. Furthermore, ERNEST is able to back propagate this information into the input model, to mark the connections and functions related to the failed timing constraints.

4.5 FMEDAexpress

4.5.1 General description

FMEDAexpress is a safety analysis tool for FMEDA analysis according to IEC 61508 or ISO 26262.

FMEDAexpress has been developed with C#.

4.5.2 Tool along the lifecycle.

This tool addresses the following parts of ISO 26262:

- Part 3. Concept phase, sub-phase 3-8 Functional safety concept (supporting the derivation of functional safety requirements and the choice of the best concept alternative).
- Part 4. Product development at system level phase, sub-phases 4-7 System design (system design verification) and 4-9 Safety validation.
- Part 5. Product development at hardware level phase, sub-phases 5-8 Evaluation of the hardware architectural metrics and 5-9 Evaluation of the safety goals violations due to random hardware failures.
- Part 6. Product development at software level phase, sub-phase 6-7 Software architectural design (efficiency verification of the safety mechanisms).
- Part 9. ASIL-oriented and safety-oriented analyses, sub-phase 9-8 Safety analyses

4.5.3 Artefacts metamodel

FMEDAexpress manages information on XML files and it can be accessed using SQL.

4.5.4 Inputs

FMEDAexpress requires as input involved components, failure modes, effects and countermeasures. All these pieces of information are provided manually. In addition, FMEDAexpress can import parts or component lists with failure modes given in XML format.

4.5.5 Outputs

FMEDAexpress delivers as output: dangerous detected/dangerous undetected failure rates, Single-point fault metric (SPFM), Latent-fault metric (LFM), Probabilistic Metric for random Hardware Failures (PMHF), Diagnostic Coverage (DC), FMEDA report. Together with this information, the tool also provides some extra reliability information: Mean Time Between Failures (MTBF), Mean Time To Failure (MTTF) and so on. In addition, entire data is available in XML format to be further processed in other tools.

4.6 Papyrus

4.6.1 General description

Papyrus is a general purpose UML modelling tool. It consists of a set of Eclipse plug-ins. It supports the UML extension mechanisms in the form of profiles and offers several possibilities to customize its user interface. In particular, Papyrus supports the profiles SysML (including SysML specific diagrams), MARTE and EAST-ADL.

Papyrus is an official Eclipse project and it is available within the Eclipse modelling bundle. Within the scope of SafeAdapt project, Papyrus 1.0.x coming with Eclipse Luna will be used. More information about Papyrus can be found on Eclipse.org/papyrus. It is the base for the model transformation and code generation tool Qompass, which is presented in Section 4.8.

4.6.2 Tool along the lifecycle.

The tool is used to manage the system model. It exports models towards AUTOSAR and receives results from the simulation tools ERNEST and UNISIM-VP.

This tool addresses the following parts of ISO 26262:

- Part 3. Concept phase, sub-phases 3-5 Item definition, 3-7 HARA and 3-8 Functional safety concept.
- Part 4. Product development at system level phase, sub-phases 4-6 Specification of the technical safety requirements, 4-7 System design and 4-9 Safety validation.
- Part 6. Product development at software level phase, sub-phase 6-6 Specification of the software safety requirements.

4.6.3 Artefacts metamodel

Papyrus 1.0.x uses the UML 2.5 meta-model, SysML 1.2, MARTE 2.1.12 and EAST-ADL 2.

4.6.4 Inputs

System/application model in form of UML 2.5 model. Simulation results (format needs to be defined).

4.6.5 Outputs

The output of the Papyrus base tool is an updated UML or SysML model. However, Papyrus has several add-ons, notably Qompass described below and an AUTOSAR gateway. The latter allows exporting AUTOSAR models in form of UML models applying an AUTOSAR profile or in form of an XML file compatible with ARTOP, the standard AUTOSAR implementation within Eclipse.

4.7 Prossurance

4.7.1 General description

Prossurance is a safety assurance management system to support a cost-effective compliance assessment and certification of safety-critical products in sectors such as aerospace, railway, maritime and automotive.

Prossurance can work in two different ways: as a file-based or database-base (Postgress) tool. This tool has been developed with the following technologies: Eclipse with GMF and EMF, XText, Subversion (SVN) Team Provider for artefact versioning if desired, a subversion client such as TortoiseSVN, Java Environment 1.7 and Windows Operating System.

Prossurance is a set of solutions built on top of the OPENCOSS project (OPENCOSS project).

4.7.2 Tool along the lifecycle

Since Prossurance addresses the construction of safety cases, it is used along all phases of the system concept and development processes. Due to the complexity of this, in SafeAdapt the focus will be on the Safety Concept (sub-phases 3-5 Item definition, 3-7 HARA and 3-8 Functional safety concept).

4.7.3 Artefacts metamodel

Proprietary metamodel including concepts of: regulations, general standards, company procedures, etc.

Mainly files in common formats (Word, Excel, PDF, txt…) are used to register pieces of evidence.

Prossurance manages the following own formats:

- EMF for managing and persisting semantic data at a higher level of abstraction
- GMF for notation with diagramming data that represents shapes and connections displayed in graphical editors
- XText for the use of restricted language and structured property descriptions to support compositional certification.

4.7.4 Inputs

- Standards, Regulations and Company Procedures are manually transformed into Prossurance concepts.
- Argumentation Patterns are used as a way for reusing successful safety strategies.
- Safety Argument Contracts are used in case of modular/compositional certification.

- Work Products are stored and managed as artefacts/pieces of evidence.

4.7.5 Outputs

Users can save diagramming data as images that can be inserted/copied to any report. In addition, data are available in EMF (models), GMF (notations) and XText formats.

4.8 Sophia for Papyrus

4.8.1 General description

Sophia is a framework devoted to Model Based Safety Assessment Support. Sophia is fully integrated with the Papyrus modelling tool (see section 4.4) and provides dedicated modules to support the various safety assessment techniques (FMEA, FTA, qualitative and quantitative reliability analysis) used to satisfy certification requirements and covering different stages of safety assessment life-cycle according to different standards. These modules are provided as various Eclipse features that can be embedded within an Eclipse environment. Safety modelling is based on dedicated UML profiles applied on the existing system design model. These profiles can be applied either on a SysML model or on DSLs implemented using UML profiles extension mechanisms. Various analyses are achieved either directly or using external formal tools in an integrated way through dedicated gateways (languages supported are AltaRica and SMV); results are presented within the modelling environment.

It is currently not clear, whether we will use of Sophia within SafeAdapt. Therefore we will focus on the other safety analysis tools in chapter 5.

4.8.2 Tool along the lifecycle.

The tool is used to perform different kinds of safety analysis required at the various stages of the safety assessment life-cycle (Preliminary Hazard analysis, FMEA, FTA, minimal cut-sets, critical sequences).

Sophia promotes an innovative cooperative approach of safety assessment and system design processes. The two processes share a common model of architecture and safety assessment starts at early stages to issue recommendations or refined safety requirements to design process, while designers evaluate the impact of possible solutions taking into account other evaluation criteria such as performance or real-time issues.

Technically, Sophia provides seamless means to build a safety model from the design architecture model using dedicated annotations and provides tools or gateways to formal tools (ARC, NuSMV, xFTA) to produce artefacts for safety justification. Moreover Sophia provides automatic documentation generation.

Main features are:

- Safety Requirements modelling and traceability
- PHA (Preliminary Hazard Analysis)
- FTA (Fault-tree Analysis – Automatic fault-tree generation from annotated model) and minimal cut-sets computation

- FME(C)A (Failure Modes, Effects and Criticality Analysis)
- Safety properties Analysis using model-checking tool

Sophia is compatible with ISO 61508 standard and provides support for ISO 26262 and EN 50128. The current version of Sophia provides support for the following parts of ISO 26262:

- Part 3-7 HARA and 3-8 Functional safety concept.
- Part 4. Product development at system level phase, sub-phases 4-6 Specification of the technical safety requirements, 4-7 System design and 4-9 Safety validation.
- Part 6. Product development at software level phase, sub-phase 6-6 Specification of the software safety requirements.

4.8.3 Artefacts metamodel

Papyrus 1.1.x uses the UML 2.5 meta-model, SysML 1.2, MARTE 2.1.12 and EAST-ADL 2. Sophia handles SysML models and dedicated DSLs based on UML/SysML. The approach could also be applied to EAST-ADL but it would be wiser to study a way of adapting Sophia to its error-model. The general annotation for error propagation is quite similar to that provided in EAST-ADL, but Sophia is richer regarding FMEA and quantitative analysis artefacts. Sophia annotations for propagation analysis can also be applied to hardware models. The framework is fully integrated with Papyrus and is extensible to support added functionalities.

4.8.4 Inputs

System/application model in form of UML 2.5 model and safety annotations provided using dedicated profile. For FMEAS inputs can be provided from excel sheets.

4.8.5 Outputs

The output of the Sophia tool depends on the analysis performed. Dedicated documentation is generated according to the type of analysis. Results are displayed within the Papyrus tool and can be consulted either by safety experts or by designers. Fault-trees are generated in OpenPSA format which is an exchange format used by several RAMS tools such as xFTA or GRIF. Exports of FMEAs can also be achieved in excel format.

Safety Requirements can be consulted from the design view of the system and allocated to components.

4.9 Qompass

4.9.1 General description

Qompass is a design tool for model transformation and code generation. The Qompass tool helps designers to deploy component-based systems. This means that designers take into account not only the SW architecture but also the HW architecture and allocation of SW to HW. The tool has a support for realizing arbitrary interactions between software components. These interactions are defined in a model library. Thus, it is possible to target multiple middleware technologies, e.g. interaction styles used in automotive domain, e.g. communication via the AUTOSAR virtual function bus (though not realized yet). Qompass also supports a separation of concerns by

enabling containers that embed the original component and intercept its communication with the environment as well as offering additional service. Containers in the context of SafeAdapt could serve two purposes:

- Provide additional information about a component (“self-X”), in particular non-functional information such as the resource usage and timing constraints. This information can be used to support reconfiguration decisions at runtime, such as the eligibility of components to execute on a certain node. The sum of this additional information provide a kind of model@runtime
- Support the implementation of safe reconfiguration algorithms that assure consistency during the migration of components with a state from one node to another. For instance, Quiescence developed by Kramer blocks new components requests, waits for ongoing to finish and then copies the component state and retargets connections. The interception facilities of a container shall use the runtime mechanisms provided by the safe adaptation core.

Qompass also supports the generation of C/C++ code. However, such code is probably not used in the context of SafeAdapt (except for some experimentation), as code is primarily generated from AUTOSAR tools.

4.9.2 Tool along the lifecycle.

This tool is used in combination with Papyrus tool. So, for further details consult Papyrus tool.

4.9.3 Artefacts metamodel

Qompass takes component, platform and allocation descriptions defined in UML 2.4 or UML 2.5 format enriched with FCM and MARTE profile.

4.9.4 Inputs

Qompass takes system model in form of component, platform and allocation descriptions defined in UML 2.5 format enriched with EAST-ADL, FCM and MARTE.

4.9.5 Outputs

Results of the model transformations, e.g. added adaptation-aware interaction components and reflective information (model@runtime). The models are UML 2.4/2.5 models conforming to the standard Eclipse UML2 plugin.

4.10 TTE-Tools

4.10.1 General description

Classification of the TTEthernet tool suite: Design tools

Compatible with standard: TTEthernet is an SAE standard (SAE AS6802)

The TTEthernet development tool suite is a product offered by TTTech Computertechnik AG. The SafeAdapt project will use the TTEthernet tool suite for the configuration of the data communication conducted via TTEthernet traffic and across TTEthernet based electronic switches,

end systems and control units. The detailed user manuals for the TTEthernet tool suite are delivered in combination with any TTEthernet tool delivery (license).

Basically the TTEthernet tool suite consists of:

- TTEPlan: TTEPlan is the TTEthernet network planning tool. Based on input provided to the tool, TTEPlan creates the whole network configuration databases.
- TTEBuild: TTEBuild allows converting XML-based device configuration database files into binary configuration images required by the TTE Switches and the TTE End Systems.
- TTELoad: TTELoad is an application suitable to configure a TTE Switch based on TTEthernet switch IP that also supports bootstrap configurations of TTE Switches.

A general overview is provided in SafeAdapt deliverable D2.3 “Requirements for the Design Process and Tools for Safe Adaptation”.

4.10.2 Tool along the lifecycle.

The TTEthernet tools enable the developer / design engineer to conduct seamless design, create configuration and provide data loading for TTEthernet based networks. The tools are built around open XML data bases.

These tools fully address the “6-8 Software unit design and implementation” part of ISO 26262 in relation with communications topic.

4.10.3 Artefacts metamodel

The TTEthernet Tools use XML file format as a representation of the binary code needed for the network and device configuration for easy and direct human readability. HEX or BIN file formats resulting from the TTEBuild Device Configuration tool can be used for direct download to the switches.

Third party tools can be used as well in order to add or modify configuration of the parameters.

4.10.4 Inputs

TTEPlan: configuration data by human, interactively lead input routines for the network configuration.

TTEBuild Network Configurator: TTEPlan XML output file.

TTEBuild Device Configurator: TTEBuild Network Configurator output XML file.

4.10.5 Outputs

TTEPlan: XML representation of the schedule file including requirements.

TTEBuild Network Configurator: XML representation of the network configuration file set.

TTEBuild Device Configurator: XML representation of the target device switches and end-systems as well as one binary device configuration image per device in the network in BIN or HEX format. They can be used for direct download to the TTEthernet switches and can be used in combination with the End-system drivers.

4.11 UNISIM-VP

4.11.1 General description

UNISIM-VP is a cross-platform open source simulation environment based on SystemC industry standard. Its purpose is to be used during co-design, integration and validation of hardware/software systems.

The simulation environment comprises a set of tools and services such as program loaders, OS ABI translators, instrumentation and graphical debugger. Supported hosts are Windows, Linux and Mac OS X.

The available platforms are MPC755, MPC7447A, PPC440, Virtex-5 FXT, ARM7, ARM9, Star12X, TMS320C3X.

4.11.2 Tool along the lifecycle.

The UNISIM-VP simulation environment will be used to emulate the targeted hardware platform and hence to execute the embedded software.

UNISIM-VP can serve as a foundation of a virtual validation environment, because it can be interfaced to test cases generators for a better fault coverage and to runtime verification tools for diagnosis of defects. Additional information about the use within SafeAdapt can be found in section 6.2.

The main use cases are:

- Non-intrusive debugging and testing of software: Unmodified software can be debugged and tested using UNISIM-VP simulators without affecting either its functional or temporal behavior. Thanks to services, the user can drive simulation, profile the software, inspect the system status, instrument system under study (hardware pins, program variables, registers, etc.), and then analyze the result (trace analysis)
- Hardware/software integration: The software stack can be debugged and tested within a representative hardware environment before the availability of the physical target hardware
- Development of SystemC IPs (intellectual property) and new virtual platforms: UNISIM-VP is an open source (BSD license) simulation environment that comprises a SystemC module library, and a set of services (debugging, loaders…). It can be a foundation for the development of new SystemC IPs and new virtual platforms
- Support the simulation of hardware faults using component attributes (parameters). These attributes are used to notify the simulator engine of the occurrence of a hardware fault. At each fault occurrence, respective architecture specific flags are set.

4.11.3 Artefacts metamodel

There are difference levels at which UNISIM-VP architecture is described:

- The hardware structure: description of hardware components (processor, memory …) and their interconnections. UNISIM-VP virtual platforms are interfacing third party tools using SystemC/TLM2 standards. An engineer has to write specific SystemC module to interact through hardware interface.

Today there is no generic meta-model, works are planned to generate the top-level simulator file from UML/MARTE models.

- Configuration of hardware, i.e. configure clock frequencies or memory size: Described by XML files corresponding to a small schema definition.
- Ad hoc simulation module is used to supporting hardware fault strategy.

4.11.4 Inputs

Embedded software binary with debugging information.

1. Full stack application software
2. Simulator configuration XML file corresponding to an XML-Schema (see second bullet in preceding section)

4.11.5 Outputs

Embedded software execution trace (instructions, software symbols, hardware registers, hardware interface).

4.12 XMT - The CHROMOSOME Modelling Tool

4.12.1 General description

CHROMOSOME stands for Cross-domain Modular Operating System or Middleware. XMT is the Eclipse-based model-driven design tool of CHROMOSOME with automatic code generation capabilities for static configuration of the target system. CHROMOSOME has a large set of designated features and is designed to evolve over time. It is completely open source and hence transparent to developers and end users.

4.12.2 Tool along the lifecycle.

CHROMOSOME (often abbreviated by XME) is a domain-independent, data-centric middleware for cyber-physical systems. From the point of view of an application component, CHROMOSOME abstracts from basic functionality that is traditionally found in operating systems and middleware, like scheduling and communication. Apart from that, it offers model-driven design tools with code generation capabilities that allow a user to design the distributed system in an abstract way. So, it is related to the phase of “product development at software level”.

4.12.3 Artefacts metamodel

Currently there is no data exchange with other tools in the toolchain available. Software components are modelled manually and the output is target code.

4.12.4 Inputs

The tool requires information about software components and how functions interact with each other. There is no automated import for information from other tools.

4.12.5 Outputs

The output is target code.

5 Guidelines and Methodology

This section provides our work-flow guideline for the design process and support of verification and validation phases. The aim is to help users to start working with the SafeAdapt Tool Chain, guiding their actions according to ISO 26262 "Road vehicles – Functional safety".

This standard is intended to be applied to safety-related systems that include one or more electrical and/or electronic (E/E) systems and that are installed in series production passenger cars with a maximum gross vehicle mass up to 3 500 kg. ISO 26262 does not address unique E/E systems in special purpose vehicles such as vehicles designed for drivers with disabilities.

ISO 26262 defines functional safety for automotive equipment applicable throughout the lifecycle (management, development, production, operation, service, decommissioning) of all automotive electronic and electrical safety-related systems. It aims to address possible hazards caused by malfunctioning behaviour of E/E safety-related systems, including interaction of these systems.

More precisely, it is a risk-based safety standard, where the risk of hazardous operational situations are qualitatively assessed and safety measures are defined to avoid or control systematic failures and to detect or control random hardware failures, or mitigate their effects.

To this end, it comprises around 750 clauses on approximately 450 pages, 9 normative parts and a guideline as the 10th part. ISO 26262 is based upon an industry-standard V-model as a reference process model as it is shown in Figure 2.

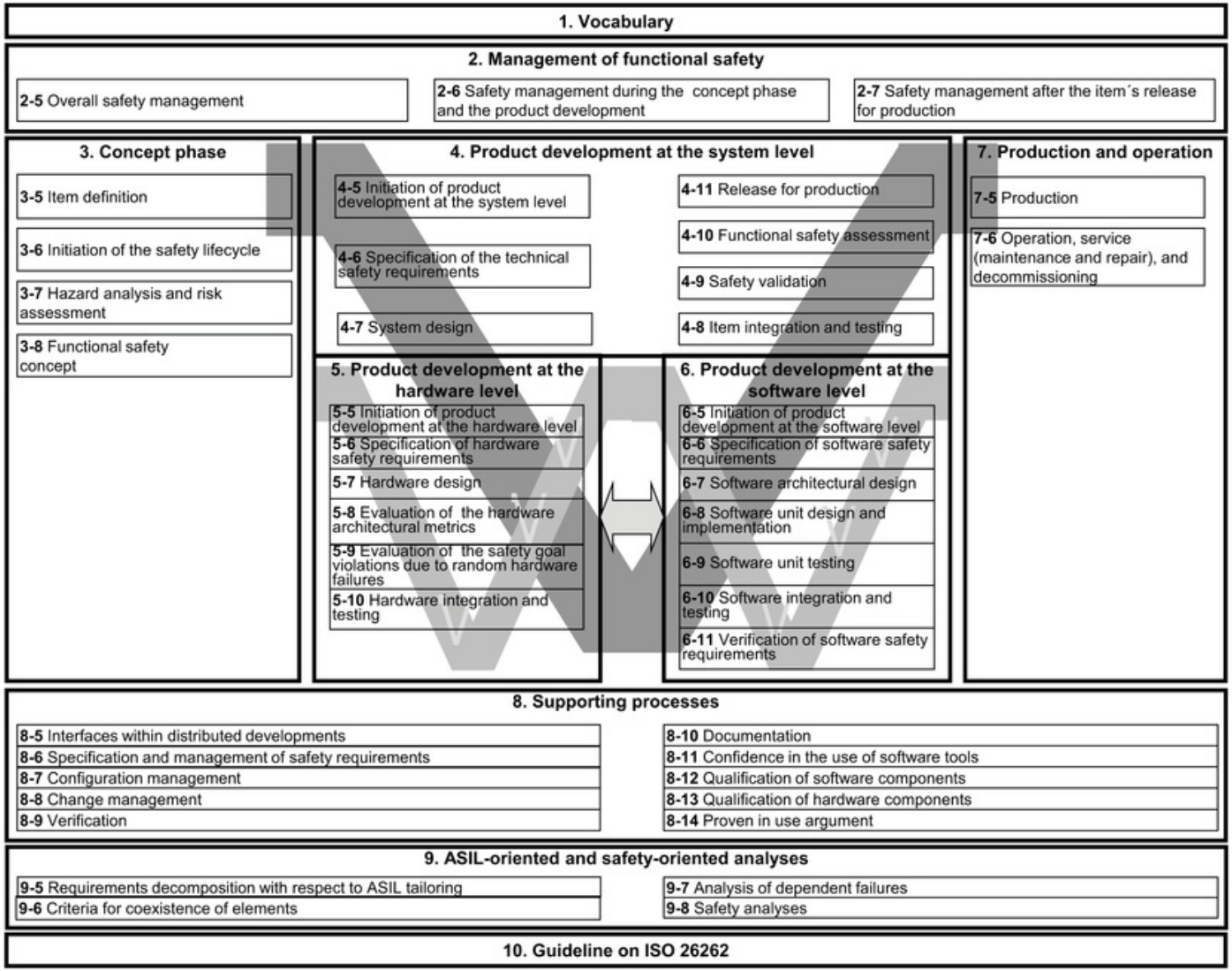


Figure 2: Overview of ISO 26262

One of the main SafeAdapt project objectives is to empower developers to create safety-critical software more efficiently by reducing costs and coping with certification requirements. As said above, the proposed SafeAdapt Methodology fully addresses the ISO 26262 standard and it is supported by an integrated tool chain that follows a model-based design flow, which is complemented by pre-existing AUTOSAR tool chains. Specifically, SafeAdapt Methodology focuses on the following phases of the ISO 26262 safety life-cycle:

- Concept phase, (Part 3)
- System level development – specification, (Part 4)
- Software level development, (Part 6)
- System level development – integration and validation (Part 4)

As shown in next figure, these parts of the standard are digitalised with the Prossurance tool using its prescriptive knowledge management functionality. Such functionality allows the management of standards information as well as any other information derived from them, such as interpretations about intents, mapping between standards, etc.

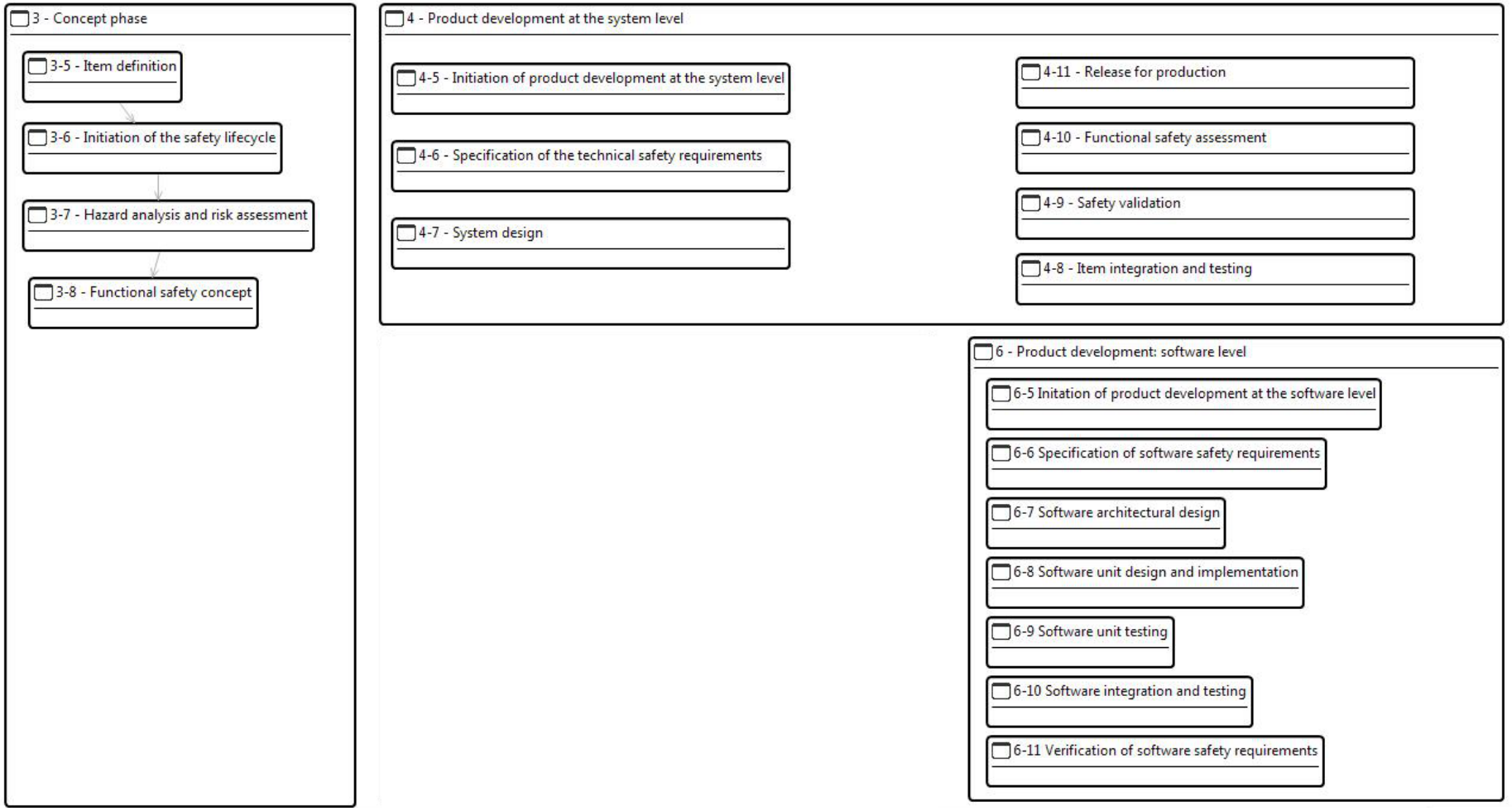


Figure 3: Overview of ISO 26262 in Prossurance tool

5.1 Concept phase

ISO 26262-3:2011 specifies the requirements for the Concept phase for automotive applications, including the following:

- item definition,
- initiation of the safety lifecycle,
- hazard analysis and risk assessment, and
- functional safety concept.

As shown in next figure, the Concept phase is digitalised with the Prossurance tool using its prescriptive knowledge management functionality. This functionality manages standards information as well as any other information derived from them, such as interpretations about intents, mapping between standards, etc. The process and activities which are required to be carried out in order to address the Concept phase are defined with Prossurance tool. In addition, required and produced assets and assurance artefacts are identified. In latter case this identification is shown through green relationships while red dotted relationships are used in case of required workproducts.

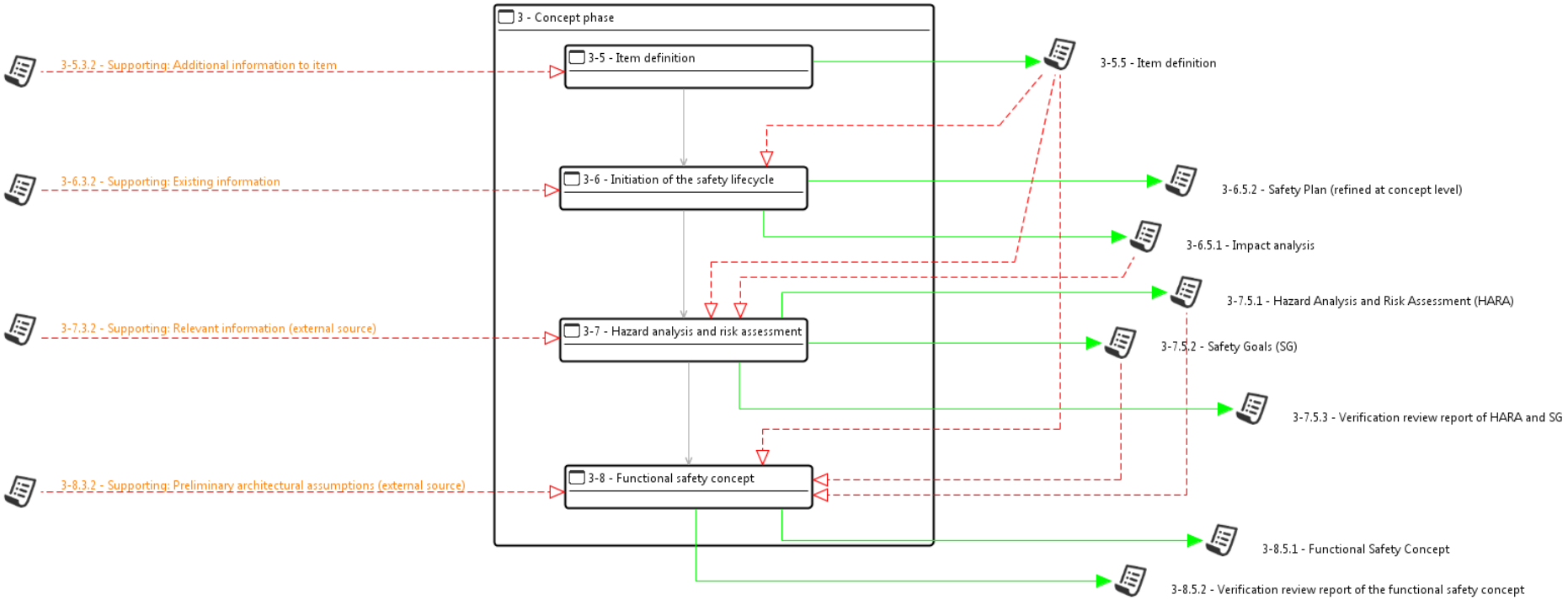


Figure 4: ISO 26262 Concept phase in Prossurance tool

In the following chapters detailed guidelines are presented in order to address Concept phase in accordance with the standard.

➢ 3-5 Item definition

As defined in the ISO 26262 standard, the Item Definition phase has two objectives. The first objective is to define and describe the item, its dependencies on, and interaction with, the environment and other items. While the second objective is to support an adequate understanding of the item so that the activities in subsequent phases can be performed.

So, the first step of the safety design flow consists of identifying and describing the “item” under development. It represents the functions, components, or (sub)systems of particular concern in regards to functional safety. To perform the item safety analysis, it is essential to properly understand the item itself in terms of input(s)/output(s), functionality, interfaces, environmental conditions and to define the item target function, which is the function description in terms of outputs behaviour. At the beginning of the safety analysis activities, the boundary of the item and the item’s interfaces with other elements are determined.

Following a top-down approach, Papyrus and Qompass tools allow accomplishing these objectives since they are based on SysML and EAST-ADL standards. EAST-ADL2 provides an ontology and a concrete language for system definition and information management. The EAST-ADL2 language contains multiple levels of abstraction: VehicleLevel, AnalysisLevel, DesignLevel, and ImplementationLevel. Each abstraction level corresponds to one specific view of the system architecture at a particular development stage.

In addition, in SafeAdapt, Papyrus and Qompass tools are complemented with Excel templates in order to make easier the process to non-expert users.

According to ISO 26262, the Item Definition work product should include the following information:

- The functional and non-functional requirements of the item as well as the dependencies between the item and its environment shall be made available.
- The boundary of the item, its interfaces, and the assumptions concerning its interaction with other items and elements, shall be defined considering.

The first step begins with the item’s target feature definition (the feature description in terms of the vehicle’s output(s) behaviour) as it is shown in Figure 5.

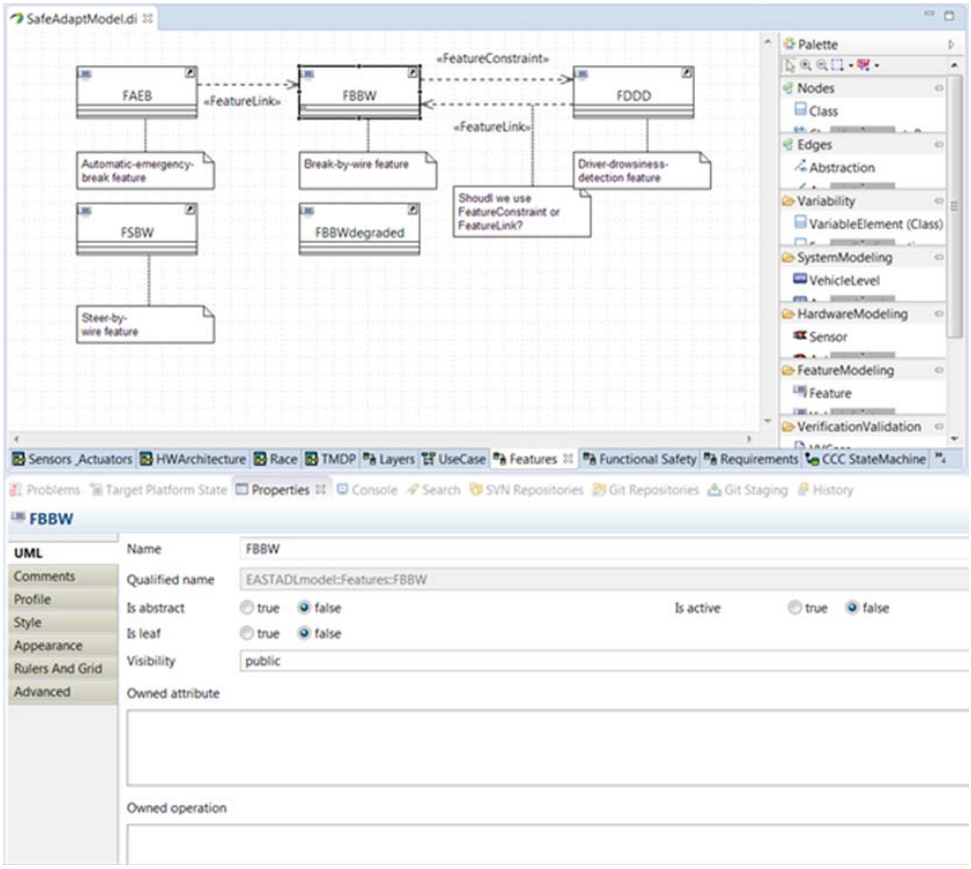


Figure 5: Target features definition in Papyrus tool

To carry out these activities, the following process has been set up in order to capture the requirements:

- One partner has been chosen as responsible for the integration of all contributions. This action should be done in a regular base and exceptionally when a requirements review meeting was arranged.
- There is a Requirements mother file template (in Excel format) as a basis where changes/enhancements should be specified.
- The current version of the requirements will be stored in the file named as [date]_SafeAdapt_Requirements_V[xy].xlsx (i.e. "2014-03-28_SafeAdapt_Requirements _V01.xlsx"). In this way versions shall be recognizable by its date and its version number.
- The specific partners contributions will be stored in a file following name syntax: [date]_SafeAdapt_Requirements_V[xy]_[Companyacronym].xlsx (i.e.: "2014-03-28_SafeAdapt_Requirements _V01_TTT.xlsx"). As a first action prior to make modifications, the mother file will be always copied under the partner specific file name and then it is possible to start working. Everything new will be marked with red colour.
- The partner responsible for the integration will integrate the changes into the requirements mother file. And then, partner specific files will be stored in an auxiliary directory.

As a result of this process, all functional and non-functional requirements will be specified and reviewed. For instance, some requirements are shown in Figure 6.

	A	B	C	D	E	F	G	H
1	Requirement ID:	Category < Function	Sub Categor	Short Description: <Req Name>	Description: <Req Description>	Verification Method	Rationale: <The rationale behind this req.>	Depend <Dep. to oth
2	DEL-001	Non-Functional	System	Network Topology - Double star architecture - Sensors/Actuators [1]	All sensors and actuators that are not related to a fail operational application (x-by-wire) shall be connected to at least on of the gateways that enable the interfacing with the TTEthernet network unless otherwise specified. The distribution (which sensor/actuator is connected to which			
3	DEL-002	Non-Functional	System	Network Topology - Double star architecture - Sensors/Actuators [2]	All sensors and actuators that are related to a fail operational application (x-by-wire) shall be connected to different gateways that enable the interfacing with the TTEthernet network. The distribution (which sensor/actuator is			
4	DEL-003	Non-Functional	System	Network Topology - Double star architecture - Gateways [1]	The network shall have at least two gateways for the sensors/actuators. The gateways shall enable the interfacing			
5	DEL-004	Non-Functional	System	Network Topology - Double star architecture - Gateways [2]	Each gateway in the network shall be connected to each available TTEthernet switch in the network to support			
6	DEL-005	Non-Functional	System	Network Topology - Double star architecture - Switches	The network shall have at least two TTEthernet switches which shall be connected to each ECU.			
7	DEL-006	Non-Functional	System	Network Topology - Double star architecture - ECUs	The network shall at least contain two ECUs (RACE from Siemens and TMDP from Delphi) that are hosting the			
8	DEL-007	Non-Functional	System	Network Topology - Double star architecture - Common requirements	The network structure shall be flexible enough to support additional ECUs, switches and gateways to be scalable for			
9	DEL-008	Non-Functional	Hardware	RaCam Interface - Gatewaying to TTEthernet	The interface of Delphi's RaCam is CAN. The messages will be sent out in a CAN-burst (see the following requirements for further information). At least one of the gateways shall be		RaCam module specification.	
10	DEL-009	Non-Functional	Software	RaCam Interface - Specification of Radar CAN messages	The radar completes the processing of the target data with a cycle time of 50 msec +/- 5 msec. At the completion of this processing, the radar shall transmit all its CAN messages in one group. The instrumentation buffers shall not overflow as a result of these sequential CAN messages. The spacing between the messages in the group should be minimized.		RaCam module specification.	
11	DEL-010	Non-Functional	Software	RaCam Interface - Specification of CAN burst messages [1]	1 x Status Message 1 (4E0h) 1 x Status Message 2 (4E1h) 1 x Status Message 3 (4E2h) 1 x Status Message 4 (4E3h)		RaCam module specification.	
12	DEL-011	Non-Functional	Software	RaCam Interface - Specification of CAN burst messages [2]	1 x Track Message 1 (500h) ... 1 x Track Message 64 (53Fh)		RaCam module specification.	

Figure 6: Requirements in Excel format

Then, the specified requirements are introduced/updated into the Papyrus tool. Papyrus differentiates between functional and quality requirements (which typically focus on some non-functional property of the system). So, requirements can be formalized using the constraints of EAST-ADL, including, timing, safety and behaviour. A requirement element is linked to any other element using a Satisfy relation. A Derive relation between requirements supports tracing between an original and derived requirement. The Refine relation links the requirement and the constraint, or other element used to specify the textual requirement in more detail. Finally, requirements can be grouped and structured using the RequirementContainer construct.

In this way, Figure 7 shows an example of the requirements specification.

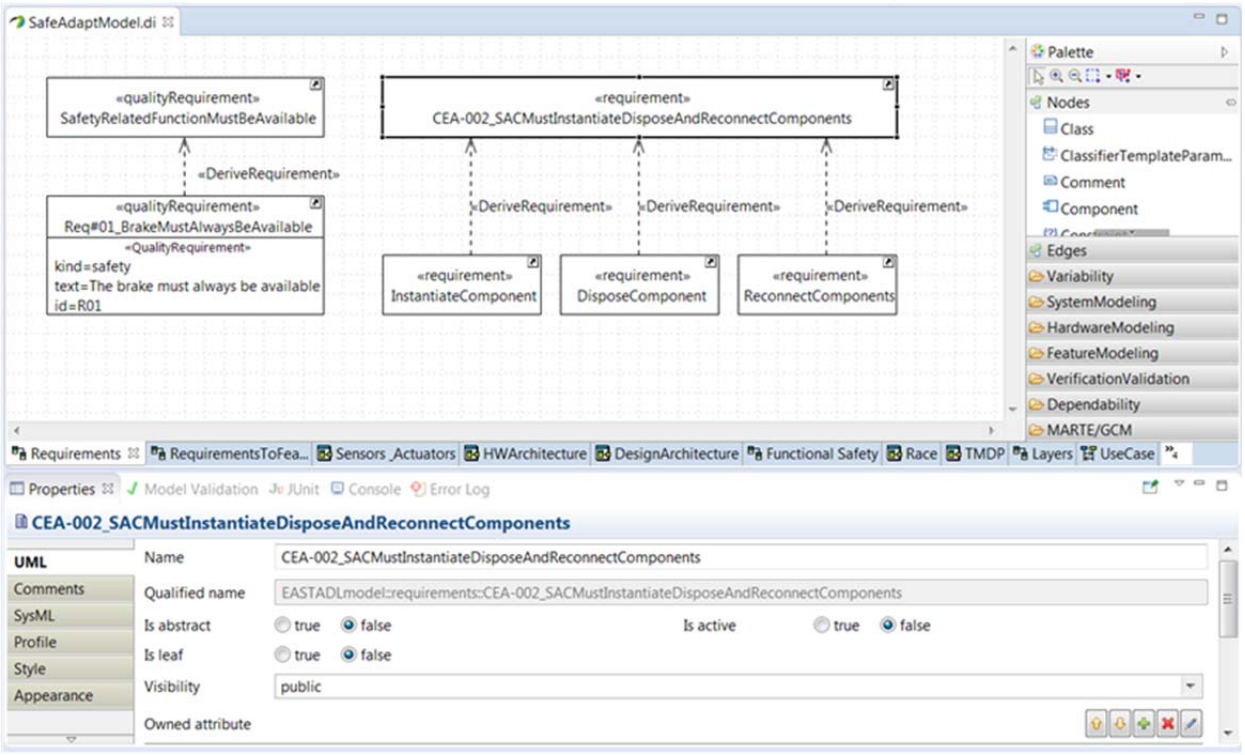


Figure 7: Requirements in Papyrus tool

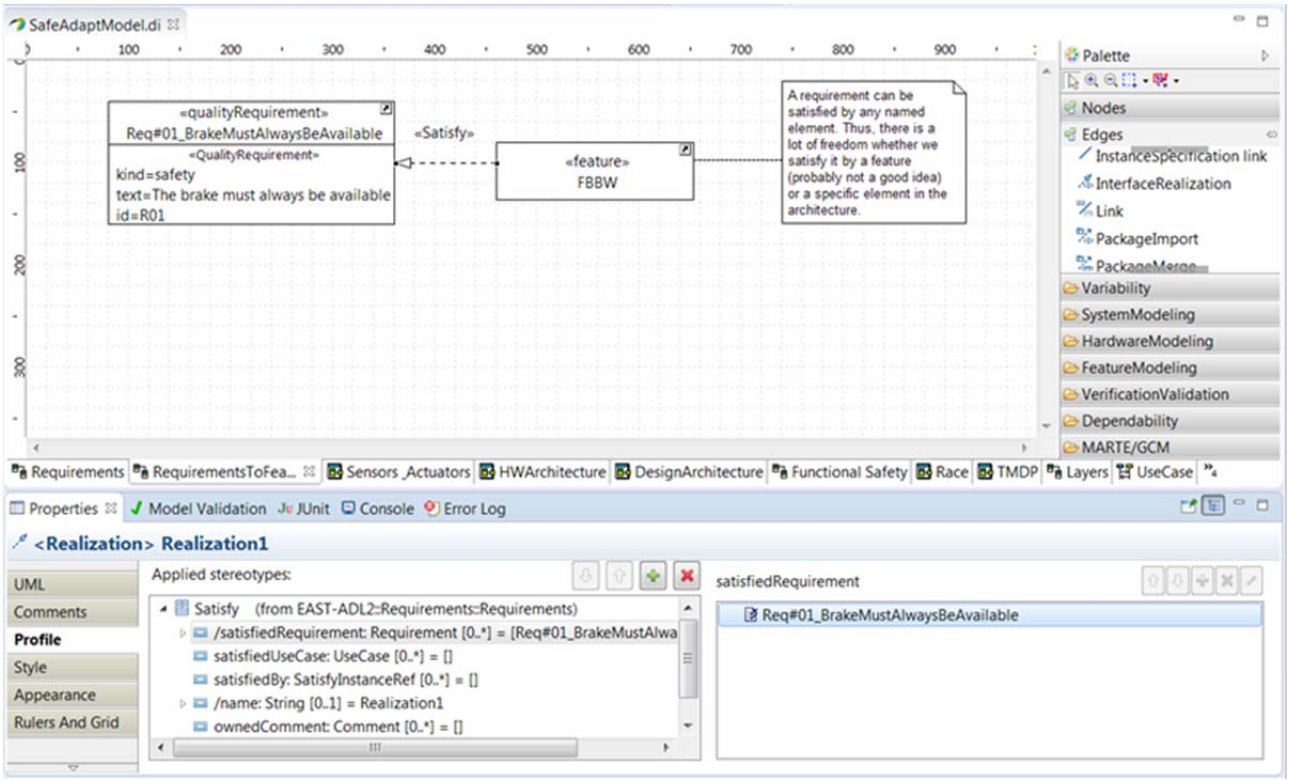


Figure 8: Allocations of requirements to features in Papyrus

➢ 3-6 Initiation of the safety lifecycle

As defined in the ISO 26262 standard, the initiation of the safety lifecycle phase has two objectives. The first objective is to make the distinction between a new item development and a modification to an existing item. The second objective is to define the safety lifecycle activities to be carried out in such case.

Consequently, in case of a modification of an already existing item or its environment, an impact analysis is required and a tailored safety lifecycle is advisable. In this special case, and according to ISO 26262, the impact analysis shall identify and describe the intended modification applied to the item or its environment and assess the impact of these modifications.

Therefore, with the hypothesis that the safety analysis is already available (inherited from original item), the most convenient approach is the bottom-up one, i.e. by verifying the impact in terms of differences in hazard list and risk assessment outcomes.

➢ 3-7 Hazard analysis and risk assessment

In terms of ISO 26262, the objective of the hazard analysis and risk assessment is to identify and to categorise the hazards that malfunctions in the item can trigger and to formulate the safety goals related to the prevention or mitigation of the hazardous events, in order to avoid unreasonable risk. As a final step, the hazard analysis, risk assessment and the safety goals should be verified.

In order to evaluate the risk associated with the item under safety analysis, a risk assessment is carried out. A risk assessment considers the functionality of the item and a relevant set of scenarios (operating conditions & environmental conditions). To identify hazards, the potential sources of harm, it is helpful to define the malfunction(s) related to the item. If the item target function(s) has been correctly identified and described, the malfunction can be always defined in terms of anomalies of function activation. To assess the risk level, hazardous events, the hazard in concomitance with a particular scenario, is considered.

As required by ISO 26262, for each identified hazardous event, the severity, controllability and exposure values should be ranked, to determine the associated Automotive Safety Integrity Level (ASIL) that shows the level of risk. It is important to remark that the controllability levels assigned to the various situations should be assessed through specific testing on the road such as fault injection testing. ASIL specifies the item's necessary safety requirements for achieving an acceptable residual risk. A risk (R) can basically be described as a function F of the frequency (f) of occurrence of a hazardous event, the ability of avoiding the specific harm through opportune reactions of the involved persons (C = Controllability) and the potential severity of the resulting harm or damage (S = severity); the frequency of occurrence depends only on the probability of the driving scenario taking place in which the hazardous event can occur (E = exposure).

During the concept phase a safety goal shall be defined for each hazardous event. This is a fundamental task, since the safety goal is the top level safety requirement, and it will be the base on which the functional and technical safety requirements are defined. The safety goal leads to item characteristics needed to avert the hazard or to reduce risk associated with the hazard to an acceptable level. Each safety goal is assigned an ASIL value to indicate the required integrity level in consonance with which the goal shall be fulfilled. For every Safety goal a Safe state, if

applicable, shall be identified in order to declare a system state to be maintained or to be reached when the failure is detected, so to allow a failure mitigation action without any violation of the associated safety goal. For each safety goal and safe state (if applicable) that are the results of the risk assessment, at least one safety requirement shall be specified.

Papyrus and Qompass tools allow accomplishing these stage objectives. In addition, these tools are complemented with Excel templates in order to make easier the process to non-expert users.

The following process has been set up in order to apply a Hazard Analysis and Risk Assessment (HARA) file template (in Excel format):

- Vehicles, Items and Components Identification. The different elements involved are identified and classified according to the concepts defined in ISO 26262. These elements are vehicle (passenger car), functional items and components.
- Situations Definition. Different situations in which a vehicle can be successfully tested are defined. These situations relate to possible items malfunctioning and exhibiting unintended behaviour, and will be described by operation conditions (possible factors of the environment as well as driver and vehicle status).
- Malfunctions Identification. ISO 26262 addresses possible hazards caused by a malfunction of a safety-critical E/E system, including interaction between such systems. Therefore, the use cases will account for these malfunctions and subsequently identify possible conditions or triggers that cause them. At this point, FMEDAexpress (FMEA analysis) can be used for the extraction of new functional and non-functional hazards at item level not previously considered.
- ASIL Determination. As it said before, the hazard analysis and risk assessment estimates the probability of exposure, the controllability, and the severity of the hazardous events. In conjunction, these parameters determine the ASILs of the hazardous events. At this point, malfunctions will be associated with situations focusing on controllability, loss and damages, and probability of exposure. With this information, the ASIL classification will be determined. ASIL C and D will drive the use cases selection as these are at the centre of interest.

As a result of this process, the hazard analysis and risk assessment can be conducted such as it is shown in the following picture.

	Nr.	Situation (Safety Relevant)									Persons at Risk	Failure/Malfunction/ Misuse (effects in terms of functional outputs)	Hazard		Potential Effect	ASIL
		Locality					Driver/Vehicle [status before failure]									
Function		Location	Road conditions	Environment Conditions	Other char	Traffic Situation	Vehicle Speed	Manoeuvres	Driver condition	Other characteristics			ID	Description		
ACC	3	Highway	Any	Any		Other vehicles [preceding, lateral, oncoming, following]	V < 120 kph	Overtaken	On board medium careful		Driver, passenger, other drivers [vehicles, motorcycles]	Acceleration of the vehicle to weak	ACC3	. possible rear-end collision with vehicle in front	It takes to long to reach the target speed	QM
ACC	4	Highway	Any	Any		Other vehicles [preceding, lateral, oncoming, following]	V < 120 kph	Any	On board medium careful	other car is overtaking us	Driver, passenger, other drivers [vehicles, motorcycles]	Deceleration of the vehicle too hard	ACC4	Another car is following the ACC-vehicle. Rear end collision with the ACC-vehicle	Worst Case: unintended deceleration until standstill due to brake intervention	C
AEB	10	City	Any	Any		Other vehicles, motor/bicycles , pedestrian	V < 50 kph	Any	On board medium careful		Driver, passenger, cyclist, pedestrian, other drivers [vehicles, motorcycles]	AEB will brake unintentionally to full stop	AEB2	Destabilisation of the vehicle, departure of the lane, collision with objects	Worst Case: unintended deceleration until standstill due to brake intervention	C
BMS	13	Parking	na	na		na	V = 0 kph	Parking	Out of board		Driver, Passenger	Unintended charging of HV-Battery from grid	BMS1	Driver and passangers can be dammaged by a fire or explosion	overcharging of Battery- degassing fire, explosion(depending	D

Figure 9: Preliminary HARA in Excel format

Then, the specified HARA is introduced/updated into Papyrus tool.

Therefore, it is already possible to perform a Hazard analysis and Risk assessment to preliminarily evaluate the “safety relevance” of the Item under safety analysis. For this purpose, the hazards should be evaluated in different scenarios for assessing Severity, Controllability and Exposure. The hazard under analysis, when applied to the various operational situations (operative & environmental conditions), result in the so called “Hazardous Events” (HE), as you can see in the diagram in Figure 10.

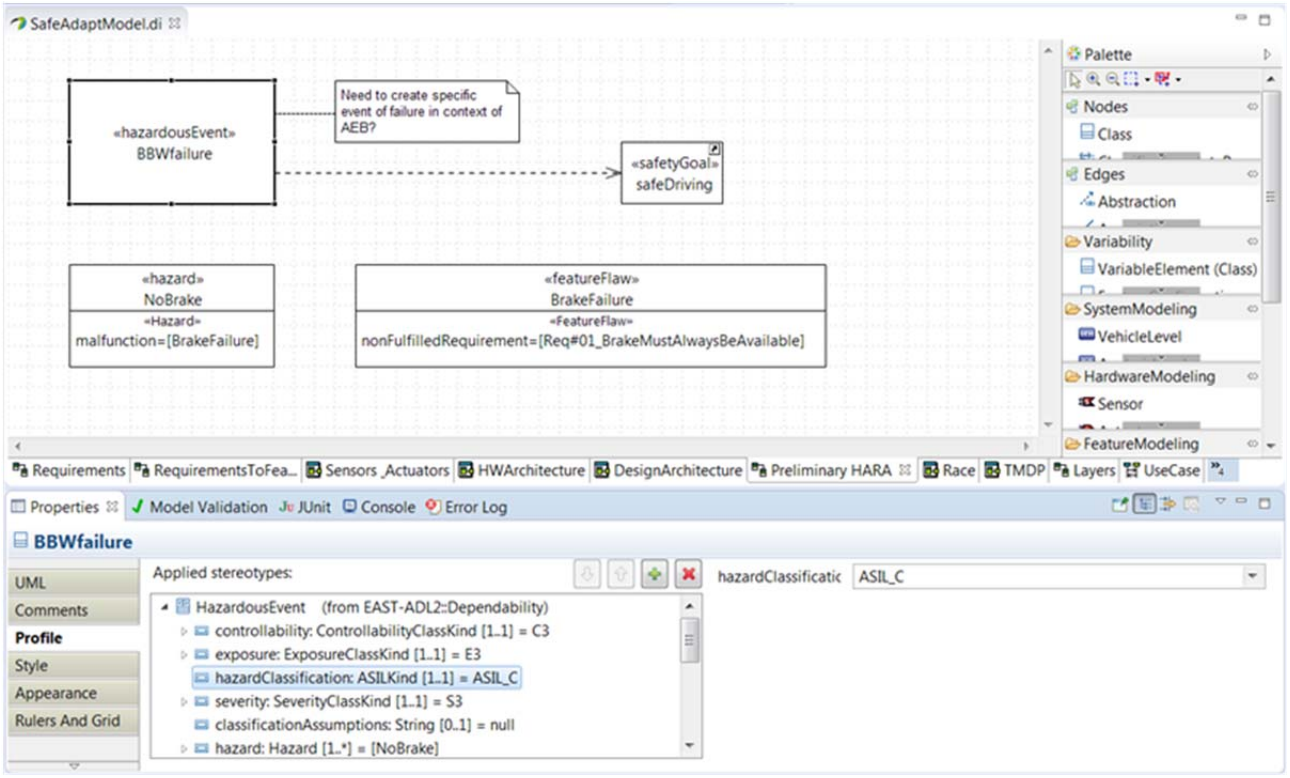


Figure 10: Preliminary HARA in Papyrus tool

Each hazardous event has to be classified in terms of associated risk defined as its ASIL. Since the identified hazardous events are related to a target feature, it makes sense to define (for each hazardous event that appears safety relevant) the safety goals. In EAST-ADL2, the safety goal artefact is modelled as a specialization of requirement. The ASIL determined for the hazardous event should be assigned to the corresponding safety goal. ASIL and safe state are attributes of the safety goal metaclass.

To verify the correctness and completeness of the preliminary hazard analysis and risk assessment performed previously, a deeper analysis has to be performed, by looking at architectural level. Therefore, the target function should be defined by deriving it from the target feature introduced at the upper abstraction level. At this point it is possible to define the malfunction as anomalies of the item's outputs. This serves as a more concrete basis for hazard identification and risk assessment, and therefore offers an opportunity for validation.

Note that this process may be iterative and parallel: hazards and risks may be identified and assessed at any abstraction level, but the information is solution independent and hazards, safety goals and safe states are managed at vehicle level.

➢ 3-8 Functional safety concept

As defined in the ISO 26262 standard, the objective of the functional safety concept is to derive the functional safety requirements, from the safety goals, and to allocate them to the preliminary architectural elements of the item, or to external measures.

The functional safety concept also describes the safety measures that are needed to avoid violation of safety goals. It shall contain assumptions about necessary driver actions if needed. Traceability between the item feature that causes the safety relevant failure and its related safety measures shall also be included.

So, the first step shall be to derive the functional safety requirements to fulfil the specified safety goals based on the information given in the item definition and the results of the HARA.

Papyrus allows accomplishing these objectives as follows. Safety goals and safe states are the results of the risk assessment. For each of these, at least one functional safety requirement must be specified. Note that what is expressed in the ISO 26262 standard as “preliminary architectural assumptions” is the purpose of the analysis architecture in the EAST-ADL2 language. At this level, the goal is to verify that the functional safety concept realises all the previously defined safety goals. More than one safety requirement could be associated with the same function.

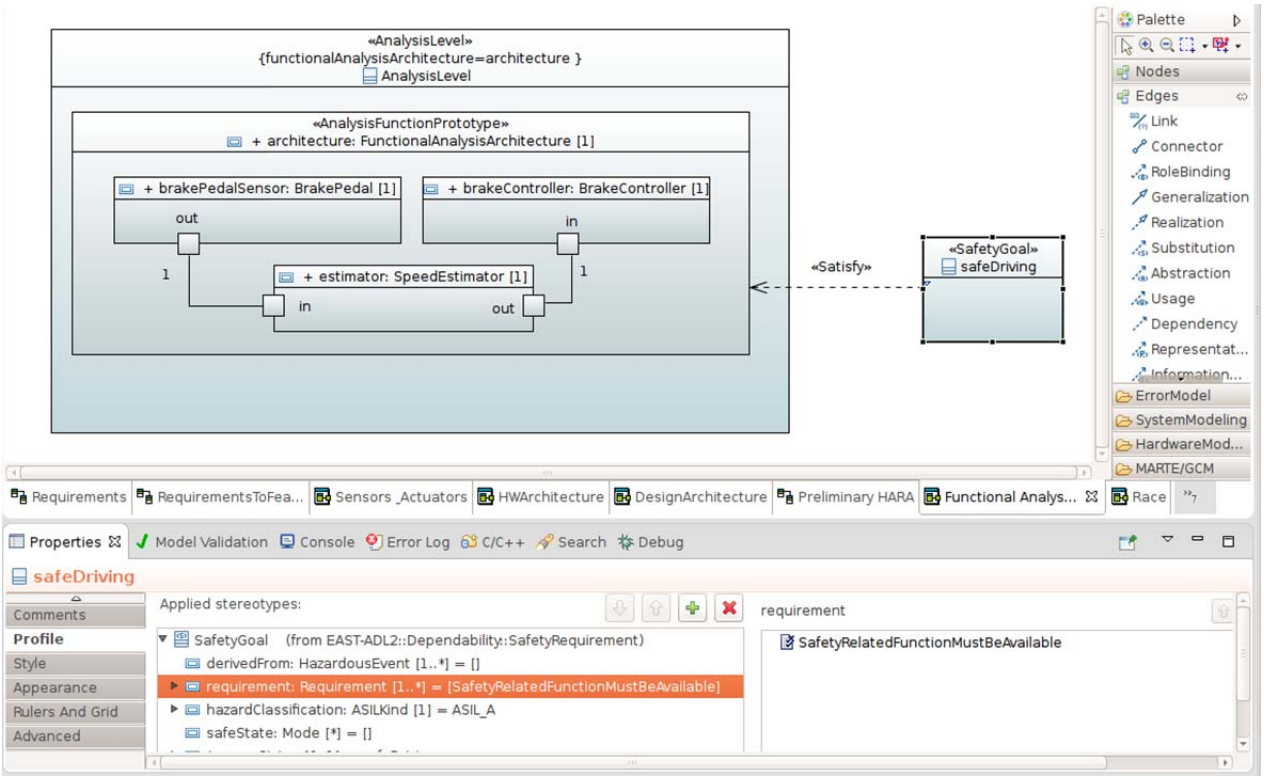


Figure 11: Functional analysis architecture in Papyrus tool

As shown in **Figure 11**, the “requirement” attribute of the safety goal “safeDriving” points to the “SafetyRelatedFunctionsMustBeAvailable” requirement (note that in general, it points to more than one attribute). The satisfy relationship maps it to the analysis architecture.

The definition of the safety architecture should also include the specification of the warning and degradation concept. The warning- and degradation concept is the specification of how to alert the driver of potentially reduced functionality and of how to provide this reduced functionality to reach a safe state. The specification of the warning and degradation concepts and the necessary actions of the driver and other persons who are potentially at risk shall be used as input for the user manual of the item.

A degradation concept is handled in the context of design pattern support within Papyrus (currently experimental). More details can be found in the SafeAdapt deliverable D4.1.

On the other hand, the safety analyses performed by composeR (CFT) and FMEDAexpress(FMEA) support the activity of deriving functional safety requirements from safety goals and safe states (Standard, 2011).

The main focus of FMEDAexpress (FMEA) during concept phase or C-FMEA (SESAMO Project) is the extraction of potential failures modes associated with the proposed functions or caused by interactions between system components. This method allows the analysis of concepts in early phases i.e. before the design is defined. In the same way, the best concept alternative can be found. In addition to the aforementioned benefits, these safety analyses allow the identification of system level testing requirements.

A deeper description about how to address both systematic and random hardware failures by applying safety analysis is depicted in ISO 26262-9:2011, clause 8.

As last task of the functional safety concept a traceability based argument can be used in order to argue about the consistency and compliance of the functional safety concept with the safety goals. This means that if the item complies with the functional safety requirements it will comply with the safety goals as well. In this area, Prossurance supports the easy development and maintenance of safety case guiding the process of collecting evidence and deducing safety arguments. That traceability based safety argumentation benefits the tight integration between system engineering, safety analysis and safety case processes.

5.2 Product development: system level

ISO 26262-4:2011 specifies the requirements for Product Development at System Level for automotive applications, including the following:

- requirements for the initiation of product development at the system level,
- specification of the technical safety requirements,
- the technical safety concept,
- system design,
- item integration and testing,
- safety validation,

- functional safety assessment, and
- product release

As shown in next figure, the Product Development at System Level phase is digitalised with the Prossurance tool using its prescriptive knowledge management functionality. This functionality manages standards information as well as any other information derived from them, such as interpretations about intents, mapping between standards, etc. The process and activities which are required to be carried out in order to address this phase are defined with Prossurance tool. In addition, required and produced assets and assurance artefacts are identified. In latter case this identification is shown through green relationships while red dotted relationships are used in case of required workproducts.

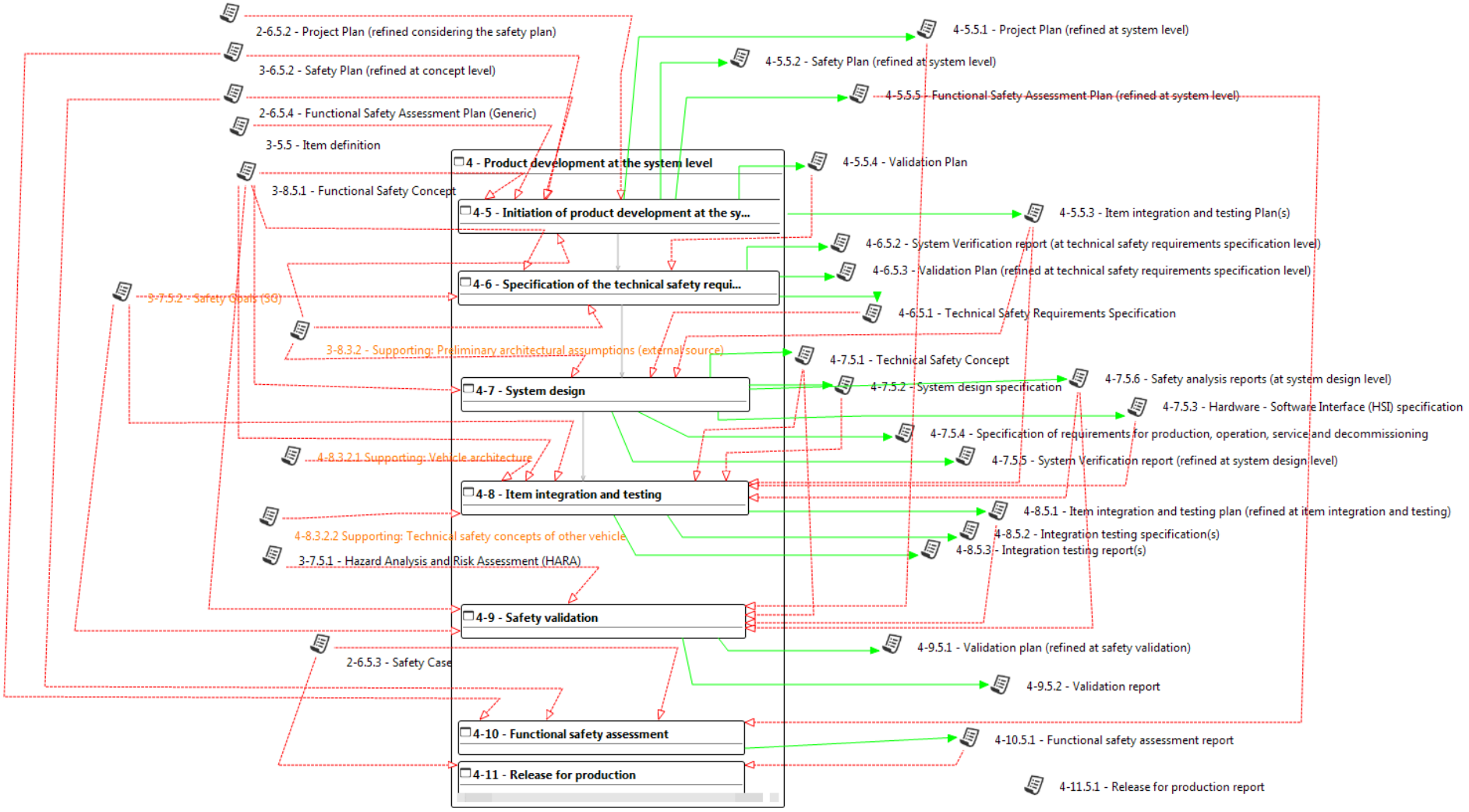


Figure 12: ISO 26262 Product development at system level in Prossurance tool

In the following chapters detailed guidelines are presented in order to address Product Development at the System Level phase in accordance with the standard.

- 4-5 Initiation of product development at system level

The main aims of this phase are to establish and plan the upcoming subphases of system development i.e. the specification of system architecture, the allocation of the technical safety requirements to hardware and software together with the hardware-software interface specification.

- 4-6 Specification of the technical safety requirements

The objective of this subphase is to develop the technical safety requirements, which refine the functional safety concept considering the preliminary architectural design.

Moreover, a second objective is to verify through analysis that technical safety requirements comply with the functional safety requirements.

In SafeAdapt, the technical safety requirements are introduced/updated into the Papyrus tool as shown in previous Figure 7. Papyrus differentiates between functional and quality requirements (which typically focus on some non-functional property of the system). The user can specify several types of relationships. A requirement element is linked to any other element using a Satisfy relation. A Derive relation between requirements supports tracing between an original and derived requirement. The Refine relation links the requirement and the constraint, or other element used to specify the textual requirement in more detail. Finally, requirements can be grouped and structured using the RequirementContainer construct. Safety mechanisms should be inherited from the technical safety requirements defining the fault detection or their control within the system (see Figure 7). This includes the ability to detect random hardware faults and if appropriated, systematic faults as well. At the same time, the measures for the detection or control of failure modes in the communication channels need to be taken in account. In fact, all the necessary measures need to put in place so that a safe state can be achieved.

- 4-7 System design

In this sub-phase, at first step, system design and the technical safety concept shall comply with the functional requirements and the technical safety requirements specification of the item. As a final step, safety analyses shall be executed as central topic of the product development to identify safety relevant failures caused by any element of the item under development that are able to cause harm to people.

Thus, the first target of this sub-phase will be addressed with Papyrus tool. With this tool it is possible to follow the standard recommendations about the use of hierarchical design and avoidance of unnecessary complexity to achieve an adequate level of granularity. Such as it is shown in the following pictures, at design level, functions from the higher level are refined into sub-functions which can be either composite (can aggregate other functions) or atomic (non-concurrent entities). In addition abstract hardware platform is provided at this stage, together with

an allocation of functions. Figure 13 shows the concrete functional architecture of a critical system in Papyrus.

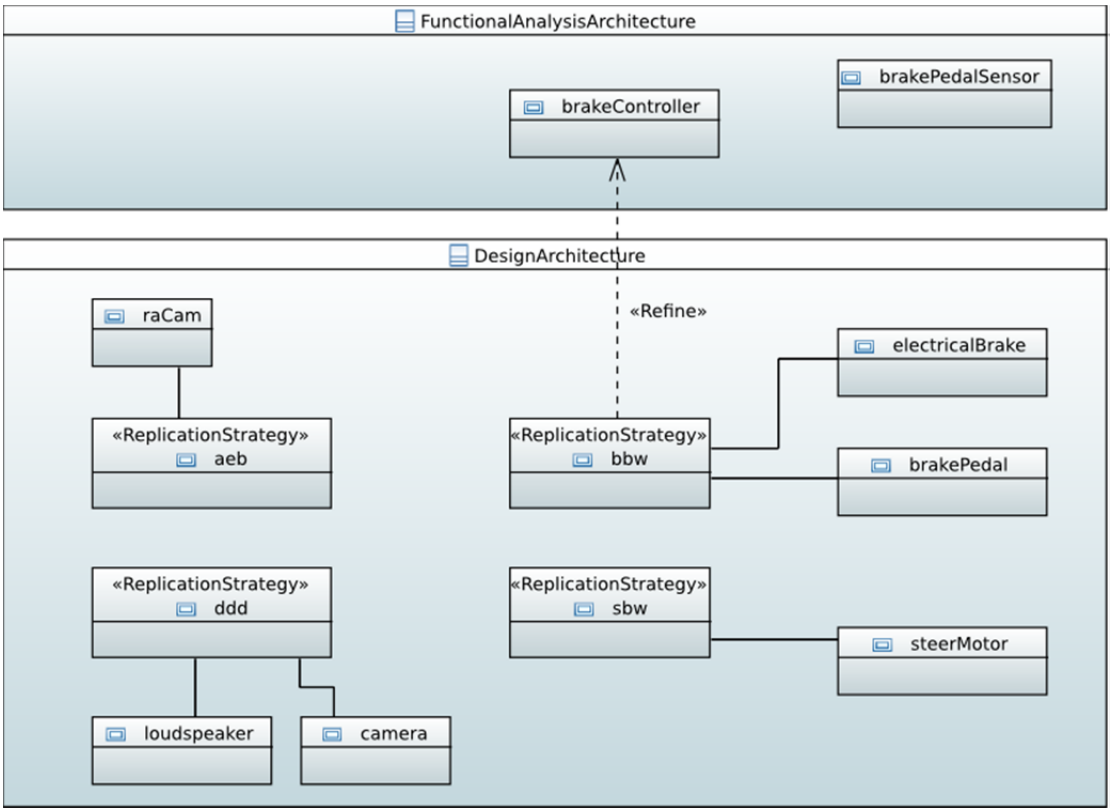


Figure 13: Concrete functional architecture of a critical system in Papyrus tool

Allocation is performed at design level, where the item is realized with concrete functional elements. At this level, function prototypes of the Functional Design Architecture (FDA) are allocated to nodes in the Hardware Design Architecture (HDA) as shown in Figure 14. Moreover, every technical safety requirement should be allocated to hardware and/or software. Hence, these activities can be performed only at DesignLevel, when the item is realised with concrete functional elements (Papyrus and Qompass). As a result of the system design analysis, probably new requirements are identified.

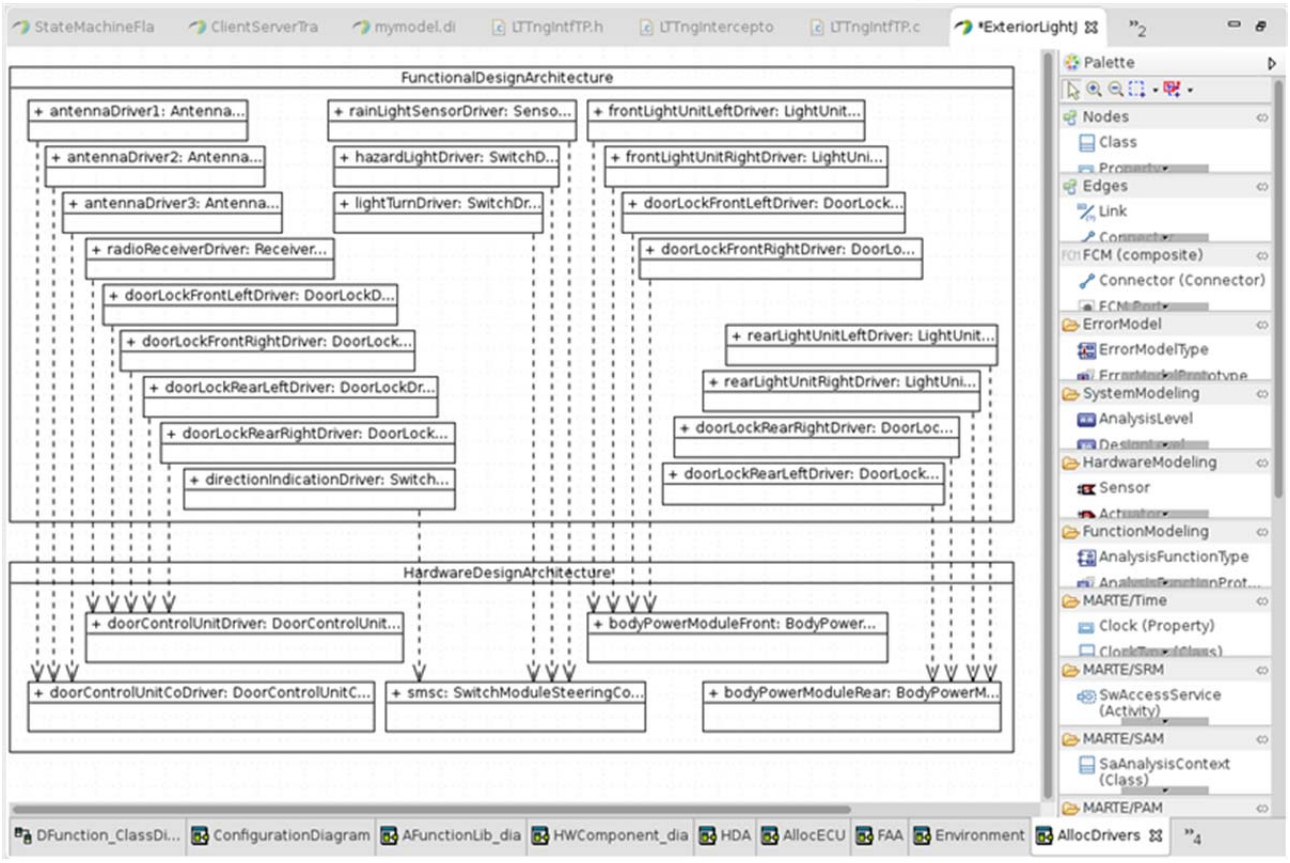


Figure 14: Function-to-node allocation in a critical system in Papyrus tool

Once the Hardware Technical Safety Concept built by hardware components has been captured, their initial failure rate data is defined. In fact, the technical safety requirements and safety measures are allocated into the different hardware elements. At this point, the Hardware Architectural Metric target values i.e. SPFM and LFM are defined at item level. These values are ASIL dependant and they will be verified at HW component level later on. In addition, the estimated value for Diagnostic Coverage (Latent and Residual) needs to be set at HW component level.

Here another key challenge is that internal and external interfaces of each safety-related architectural element have to be defined to avoid safety-related effects on other elements. On the one hand, the implementation of the architectural elements could meet the criteria for coexistence (see part 9 chapter 6 of the automotive standard). On the other hand, Hardware Software Interface Specification (HSI) should describe all safety-relevant dependencies between hardware and software. This HSI will be refined during the hardware and the software development phases. Such models need to be abstract enough to ensure later tests efficiency; however, they also should be accurate to avoid false positives.

Once all these models are available, EAST-ADL2 makes possible to perform safety simulations and analyses through external analysis tools. It must be taken into account that in an architecture specification, an error is allowed to propagate via design specific architectural relationships when such relationships also imply behavioural or operational dependencies (e.g. between software and hardware). Consequently, EAST-ADL2 models can be optimized in terms of cost, safety and performance with the framework Papyrus while all the necessary information (timing, redundancy

strategy, hardware platform) can be modelled with the input models of Qompass tool. Later on, the results of the optimization can be fed back to these models as shown in Figure 15. The blue, red and yellow bars in the chart correspond to results for different allocation strategies (a manual allocation and two automatic allocations).

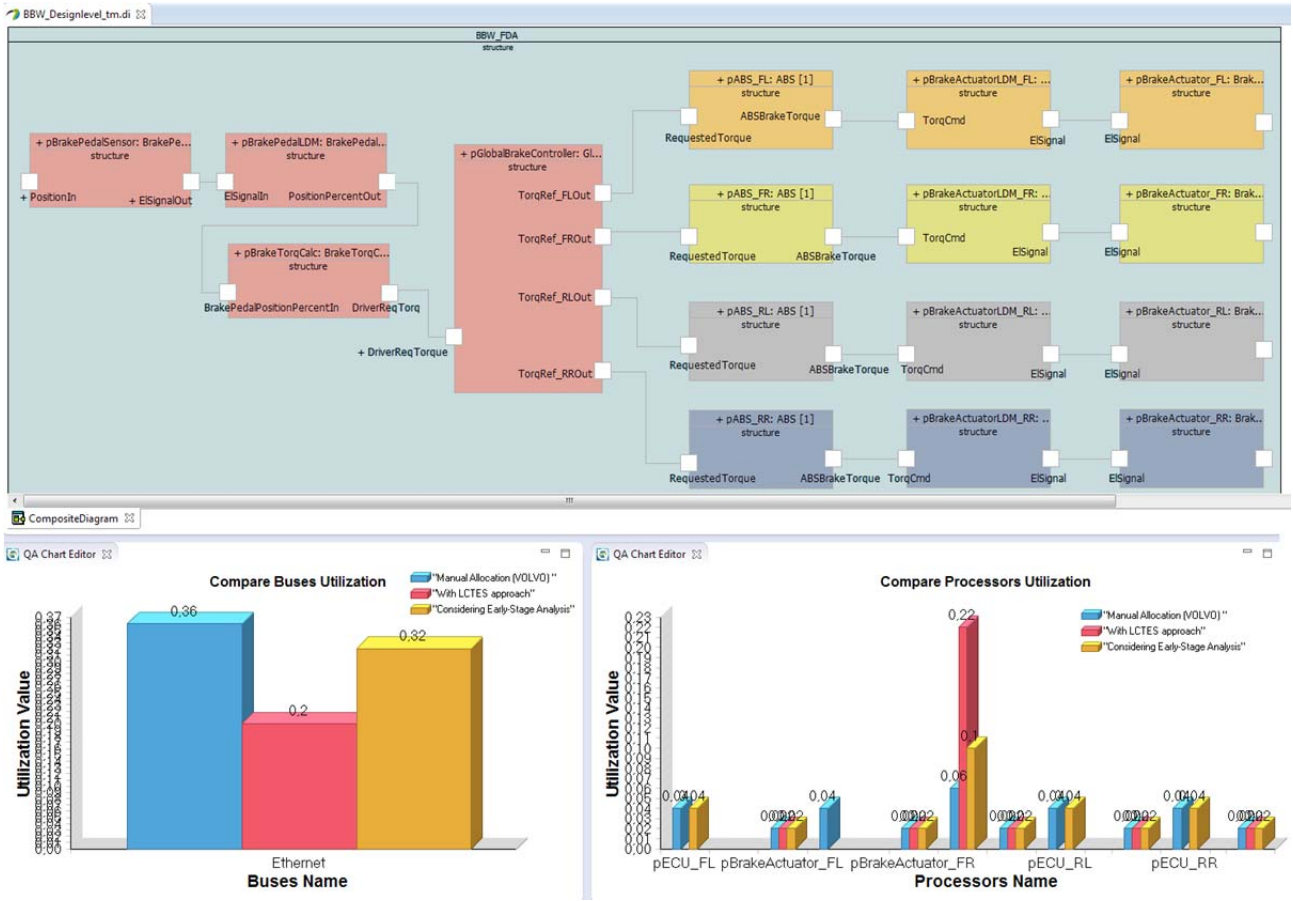


Figure 15: Results from the analysis/optimization in Papyrus (Qompass) tool

The second target of this sub-phase is to provide evidence for compliance of technical safety requirements and functional safety requirements. In this way, to verify the system design, several complementary methods shall be applied such as: inspection and walk-through, simulation and safety analyses.

Inspection and walk-through may be conducted depending on critical nature of components. Formal Inspection is a technical examination process during which a product is examined with the purpose of finding and removing defects. A defect is any occurrence in a software product (design, pseudo-code, code, comments, etc.) that is determined to be incomplete or incorrect with respect to software requirements and/or program standards. Walk-through is a form of software peer review" in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems" (IEEE Std. 1028-2008). Subsequently, the possible actions to take as a result of verification activities will be decided.

For system design analysis based on simulation techniques the ERNEST tool will be used since, in early design stages, this simulation framework enables the analysis of non-functional properties as well as adaptive behaviour at a system-wide level (simulation of computation and communication within networked embedded systems are covered). The tool is interfaced to the Papyrus modelling tool obtaining the software model and the rough hardware design (ECU and buses). In addition, this interface enables to link the simulation results with modelled system requirements via back-propagation.

As input information a deployment plan is also used for allocating the software functions to ECUs. The ERNEST tool needs this information to collect groups of software functions to be scheduled. Finally, network descriptions and timing constraints for end-to-end timing chains are supported. These timing chains describe a path through the system, starting in a sensor, going through several functions realized by software components on ECUs and ending in actuators. For each function the worst case execution time and for each connection the worst case transmission delay of a message is considered.

Obviously, the system developer has to provide the constraints for validation. Figure 16 depicts the definition of a timing constraint between two ports in ERNEST.

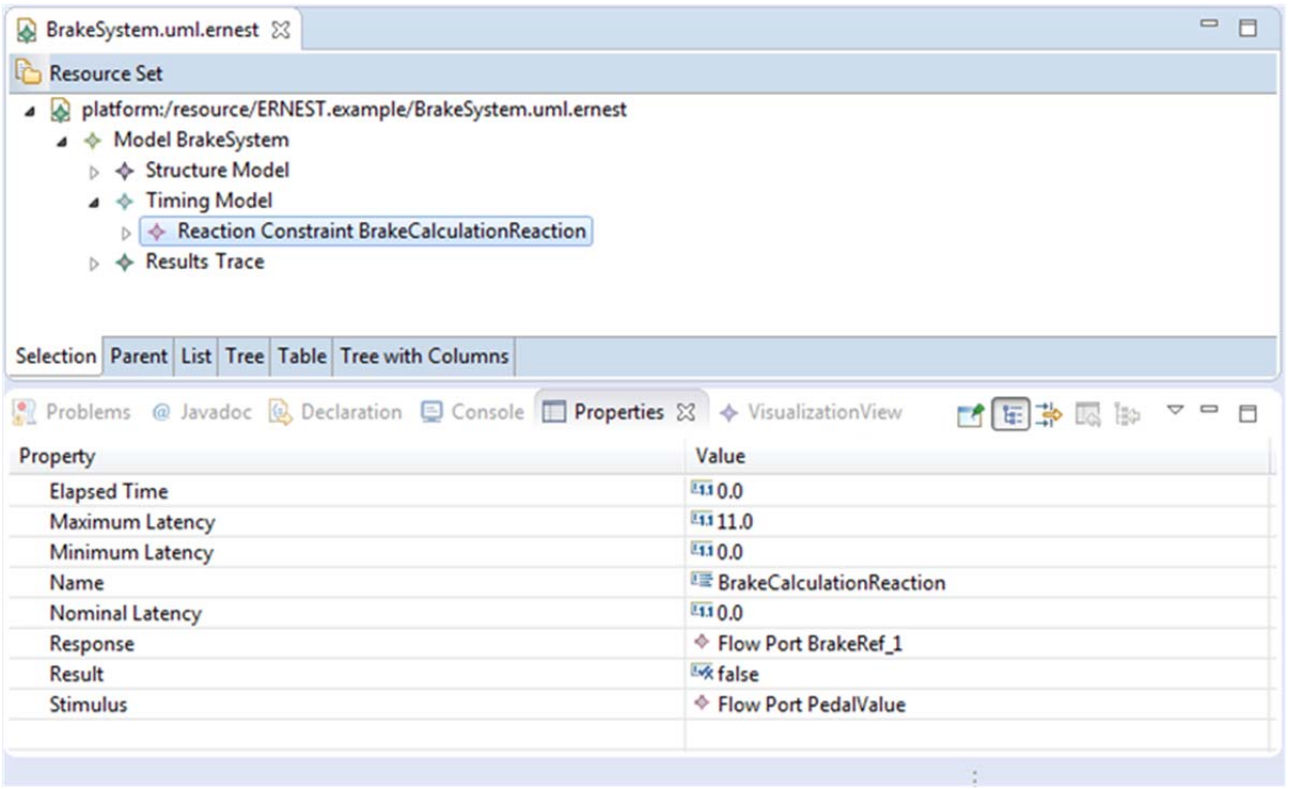


Figure 16: Definition of a timing constraint in the ERNEST tool

Finally, the ERNEST output is the validation of every constraint based on the generated timing traces during the simulation. The results are stored internally in the ERNEST model as so called “constraint validations” along with an indication when a violation occurred. In Figure 17 the visualization of the analysis results is shown where all events (stimulus and response) for a

constraint are visible. The time frame starts with a stimulus and it is only valid if a response occurs within the previously specified timing interval.

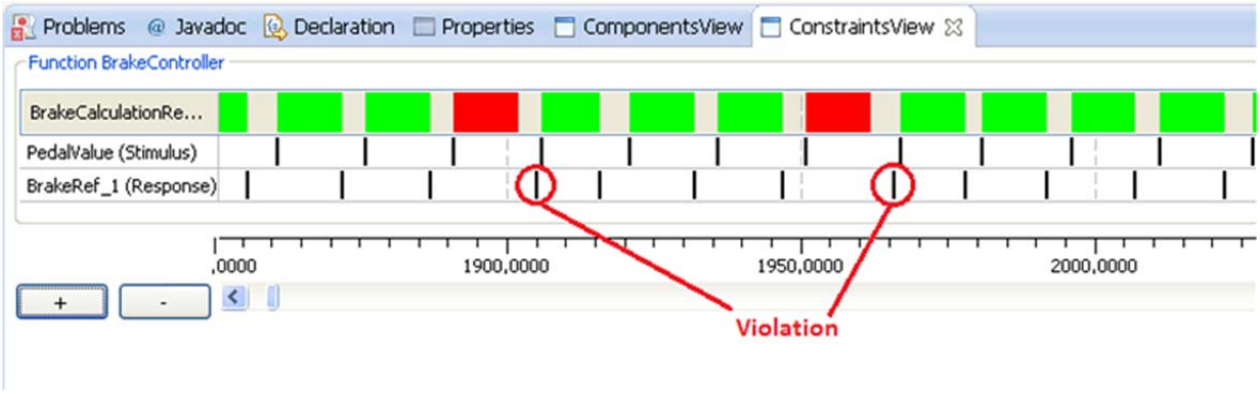


Figure 17: Visualization of analysis results in the ERNEST tool

In case constraints are violated under certain conditions, changes on the hardware or software architecture should be accomplished to fulfil all requirements. Finally, the simulation results can be back propagated into the model in the Papyrus tool, to mark the respective connections and functions regarding the failed timing constraints.

For the safety analysis, error models capture the system internal faults, failures, error logic and propagations. Currently, a State-Machine (SM) based definition of error behaviour is supported through the EAST-ADL temporal behaviour constraint specification. Given an error model, the analysis of the causes and consequences of failure behaviour can be automated through tools. In SafeAdapt project, composeR allows static safety analysis in terms of FTA whereas FMEDAexpress addresses both qualitative and quantitative FMEA. The analysis leverage includes fault trees from functional failures to software and hardware failures, minimal cut sets, FMEA tables for component errors and their effects on the behaviour and reliability of the entire system. So, it is possible to evaluate the system architecture versus alternative architectures through fast checks which detect whether certain unsafe systems states are reachable and what the corresponding probability is.

For a safety critical development both systematic and random hardware failures need to be managed. Systematic failures can be eliminated changing the design, modifying the manufacturing process or the operational procedure, documenting and using the documentation properly, etc. In other words, ensuring fault avoidance and fault removal of the design. In fact, they must be avoided and controlled. In contrast, random hardware failures occur unpredictably during the lifetime of a hardware element. This second type of failure follows a probability distribution and it is needed to control and mitigate their effects.

As depicted in ISO 26262-4:2011 clause 7, the different measures for the avoidance of systematic failures are stated in Table 2 (o = the method has no recommendation for or against its usage for the identified ASIL, + = recommended, ++ = highly recommended).

Deductive analyses (FTA, RBD, Ishikawa diagram) are highly recommended for ASIL C and D whereas inductive analyses (FMEA, ETA, Markov modelling) are highly recommended for all ASIL levels. As a result of this step, the causes of systematic failures and the effects of systematic faults are identified.

Methods	ASIL			
	A	B	C	D
Deductive Analysis	O	+	++	++
Inductive Analysis	++	++	++	++

Table 2. Methods for safety analysis on the system design (II)

As the main purpose of conducting such a safety analysis is the assistance in the design, qualitative methods can be sufficient. In any case, if necessary quantitative ones can be carried out as well.

Consequently, and thanks to this analysis, external and internal causes of systematic failures are identified and the corresponding next measures are taken to either eliminate or reduce their effect.

FMEDAexpress (Qualitative FMEA)

As previously mentioned, qualitative methods at system level are highly recommended by ISO 26262. Moreover, during this process not previously identified top level system malfunctions during HARA could arise.

At this point, components are black boxes having their own functions described. As first step, the effect of each component failure mode without safety mechanism is analysed. These effects are the top level system malfunctions already identified in HARA. The previous step allows assessing the most critical failure modes or malfunctions of the system components.

Afterwards, fault tolerance is achieved by means of defining internal or external safety mechanisms in order to control or mitigate these failure modes. In other words, the most critical malfunctions of the components are not propagated to others.

Once the safety mechanisms are defined, they are taken into consideration to redefine the new effects. Hence if any extra safety mechanism was still needed, this lack would be found out.

composeR (Qualitative FTA)

In contrast to FMEA, Fault Tree Analysis is highly recommended for ASIL C and D, being just recommended for ASIL B.

It is a complementary safety analysis to FMEA. It consists of performing Boolean logic diagrams to analyse causes and their combination into a top event or top level system malfunction violating the safety goal. To prevent the failure of the system to happen, safety mechanisms are introduced mitigating the component failure.

The repetition of an event in several branches leads to a common cause failure.

Figure 18 presents the example of a system FTA with composeR.

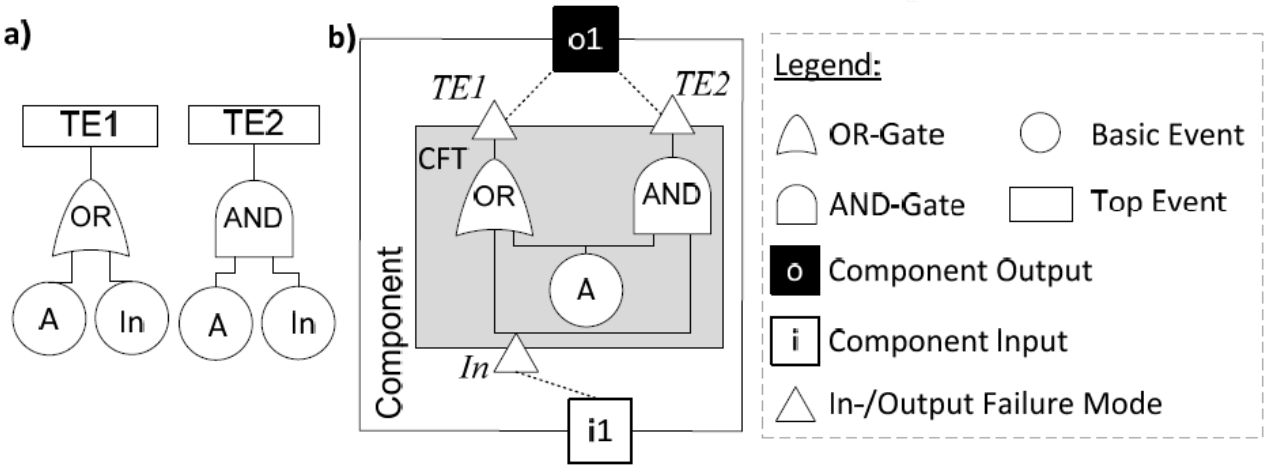
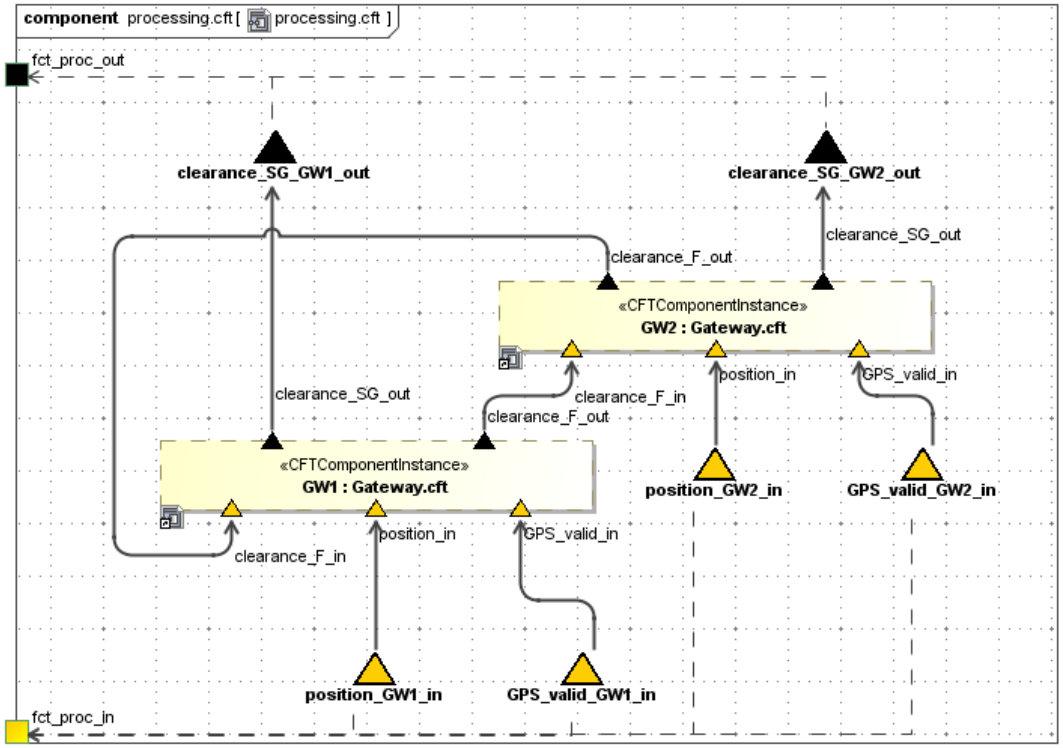


Figure 18: Example of a system FTA with composeR: (a) Classic Fault Tree and (b) Component Fault Tree. For more information, please see (Jessica Jung, 2013).

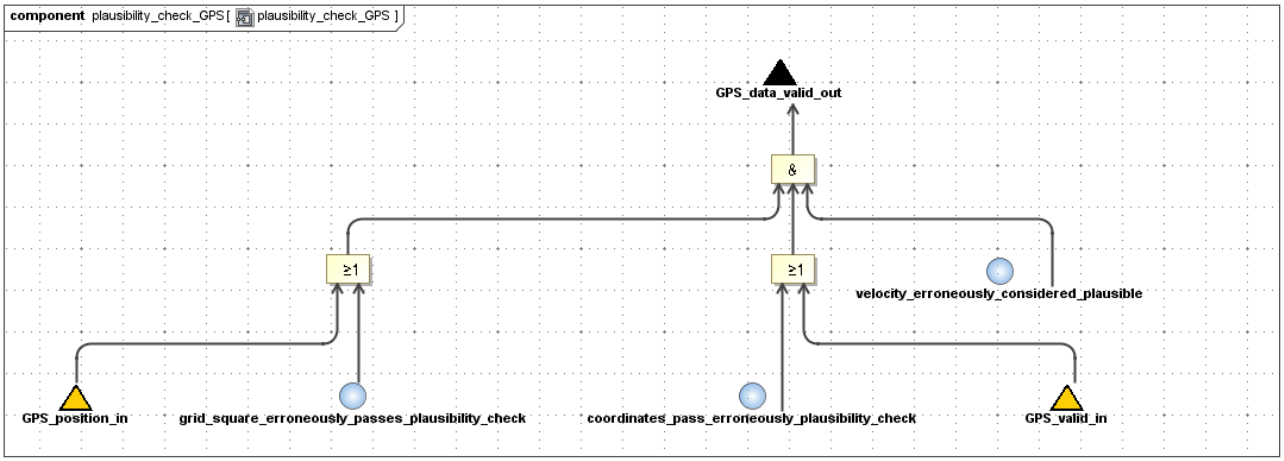
composeR (Quantitative system FTA)

This method whereby the residual risk from each safety goal is allocated into components is an extension of qualitative FTA.

After assigning the residual risk target for each safety goal which is the same that the top event needs to achieve, each event in the fault tree needs to be completed with a value starting from the top until the bottom. Figure 19 presents an example of a FTA with failure probabilities with composeR.



(a)



(b)

Figure 19: Example of a FTA with Failure probabilities with composeR: (a) At component level, failure modes propagate from one component to another and (b). Within a component fault tree, the failure behavior of a component is modeled.

In fact, in agreement with ISO 26262-5:2011, fault tree analysis is available for this purpose. To demonstrate the coverage of first failures, FMEA is recommended. During the application of FMEDA, a quantified version of FMEA, random hardware faults can be included which use

architectural metrics allocated to components to calculate the effects of random hardware faults using Single-point fault metric (SPFM) or Latent-fault metric (LFM).

The target value for SPFM and LFM for the element of the item architecture at system level shall be specified in the System Design.

➢ 4-8 Item integration and testing

The integration and testing phase comprises three phases and two primary goals as described below: the first phase is the integration of the hardware and software of each element that the item comprises. The second phase is the integration of the elements that comprise an item to form a complete system. The third phase is the integration of the item with other systems within a vehicle and with the vehicle itself.

The first objective of the integration process is to test compliance with each safety requirement in accordance with its specification and ASIL classification. The second objective is to verify that the "System design" covering the safety requirements is correctly implemented by the entire item.

According to Spanfelner et al. (Spanfelner, B.; Richter, D.; Ebel, S.; Wilhelm, U.; Branz, W.; Patz, C. , Mai 2012), Figure 20 shows the trace of the different safety requirements. It contains also the design and test flow especially for software development. This trace is also pictured in ISO 26262-10:2011 and here supplemented with additional information about the responsibility of the different levels of safety requirements. As shown in this figure, this sub-phase is carried out by several companies at different levels.

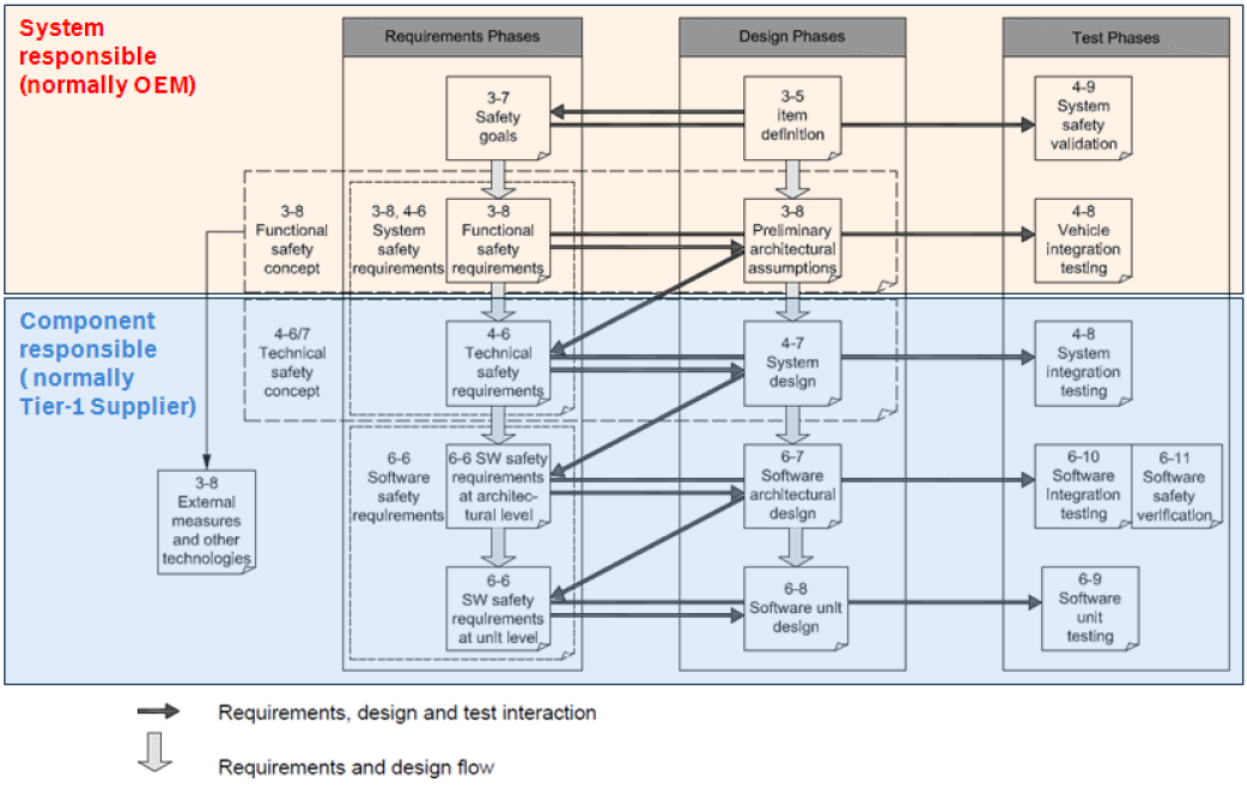


Figure 20: Safety requirements, design and test flow from concept to software

➢ 4-9 Safety validation

The first aim of this sub-phase is to provide evidence of compliance with the safety goals and that the functional safety concepts are appropriate for the functional safety of the item. The second aim is to provide evidence that the safety goals are correct, complete and fully achieved at vehicle level.

In other words, the safety validation criteria shall be specified based on the functional safety requirements and refined based on the technical safety requirements. During safety validation evidence shall be provided that the planned external measures are implemented as specified in the safety requirement documentation and that the technical solution satisfies the allocated safety goals.

The Papyrus extension Sophia tool offers detailed means to model artefacts of verification and validation activities (using Verification and Validation extension) and to relate these artefacts to requirements. This facilitates planning and tracking V&V activities and their impact on the system parallel to the system's development.

Moreover, ISO 26262 establishes that the safety validation is performed through the assurance (based on examination and tests) that the safety goals are sufficient and have been achieved. To carry out this, the following methods allowed by the standard shall be applied:

- Analyses through composeR, FMEDAexpress, UNISIM-VP and Dynacar RT (simulation based)
- Reviews

➢ 4-10 Functional safety assessment

The safety case is considered as an input of the Functional Safety Assessment, but the "Functional Safety Assessment Report" is an input for the Safety Case. It shows that activities should be performed in parallel. After a successful run of a functional safety assessment, ISO 26262 defines the "Release for Series Production" in its chapter/clause 11.

Due to permanent need of human interactions for analysis, verifications, design decisions, validations etc. in this project the "Functional Safety Assessment" could be only partially considered. Some of the described methods for verification give already the hint that for complete functional assessment a complete tailored safety lifecycle has to be considered, including human influences.

➢ 4-11 Release for production

This part of the ISO 26262 standard is out of scope of SafeAdapt project.

5.3 Product development: hardware level

The current SafeAdapt project does not fully cover this part of ISO 26262. Only main chapters addressed by the project have been included.

The tool chain is intended for development of systems with more focus on software than hardware development. Hardware is mainly seen as purchased part and only included for safety analysis and configuration aspects.

➢ 5-8 Evaluation of the hardware architectural metrics

FMEDAexpress enables the calculation of hardware quantitative measures required by ISO 26262 for hardware architectural metrics and the safety goal evaluation due to random hardware failures. In other words, these metrics capture the single contribution of each violating failure mode as a specific failure rate, according to its characterization. In short, these metrics have to be verified from the detailed design architecture using electronics parts reliability data consisting of two types of base FIT rates: persistent FIT rates (related to permanent faults) and SER (Soft Error Rate) information (related to transient faults or soft errors).

The tool calculates FIT rates such as λ_{RF}, $\lambda_{SPF,}$ λ_{MPF_I} and Diagnostic Coverage with respect to latent and residual faults which are computed to get the final SPFM and LFM metrics. The resulting values should be verified against the expected ones determined by the corresponding ASIL level.

Table 3 depicts the possible source for the derivation of the target ISO 26262 Single-point fault metric and Latent-Fault Metric value.

	ASIL A	ASIL C	ASIL D	ASIL D
Single-point fault metric	n.a.	≥90 %	≥97 %	≥99 %
Latent-fault metric	n.a.	≥60 %	≥80 %	≥90 %

Table 3. “Single-point fault metric” and “latent-fault metric” values

➢ 5-9 Evaluation of the safety goals violations due to random hardware failures

ISO 26262 Part 5 Chapter 9 proposes two alternative methods to evaluate whether the residual risk of a safety goal violation because of random hardware failures of the item is comparable to residual risk of other items already in use. This process evaluates that the residual risk of violating a safety goal due to single-point faults, residual faults and dual-point faults is low enough. In SafeAdapt both methods are applied.

The method 1 is based on a quantified FTA to calculate the so called “Probabilistic Metric for random Hardware Failures” (PMHF). There are clearly gaps in FTA tools of the market to calculate accurately this PMHF value thanks to formal representation and impact and of diagnosis coverage of safety mechanism (Cuenot, P.; Adler, N.; Otten, S., 2013).

PMHF = single point faults failure rate + residual faults failure rate + (total safety related faults failure rate / 10^{-9} * delta) * latent multiple point faults failure rate

Whereas PMHF presents a global approach, the failure rate class method (FRC) evaluates each hardware component individually. It especially addresses the individual evaluation of each residual and single-point fault and of each dual-point failure leading to the violation of the considered safety

goal. This method is based on Failure Rate Class and it is performed during quantified FMEDA. Most market available tools can calculate architectural metrics and residual risk based on method 2, but systematic reuse is missing.

5.4 Product development: software level

ISO 26262-6:2011 specifies the requirements for Product Development at Software Level for automotive applications, including the following:

- requirements for initiation of product development at software level,
- specification of the software safety requirements,
- software architectural design,
- software unit design and implementation,
- software unit testing,
- software integration and testing, and
- verification of software safety requirements.

As shown in next figure, the Product Development at Software Level phase is digitalised with the Prossurance tool using its prescriptive knowledge management functionality. This functionality manages standards information as well as any other information derived from them, such as interpretations about intents, mapping between standards, etc. The process and activities which are required to be carried out in order to address this phase are defined with Prossurance tool. In addition, required and produced assets and assurance artefacts are identified. In latter case this identification is shown through green relationships while red dotted relationships are used in case of required work products.

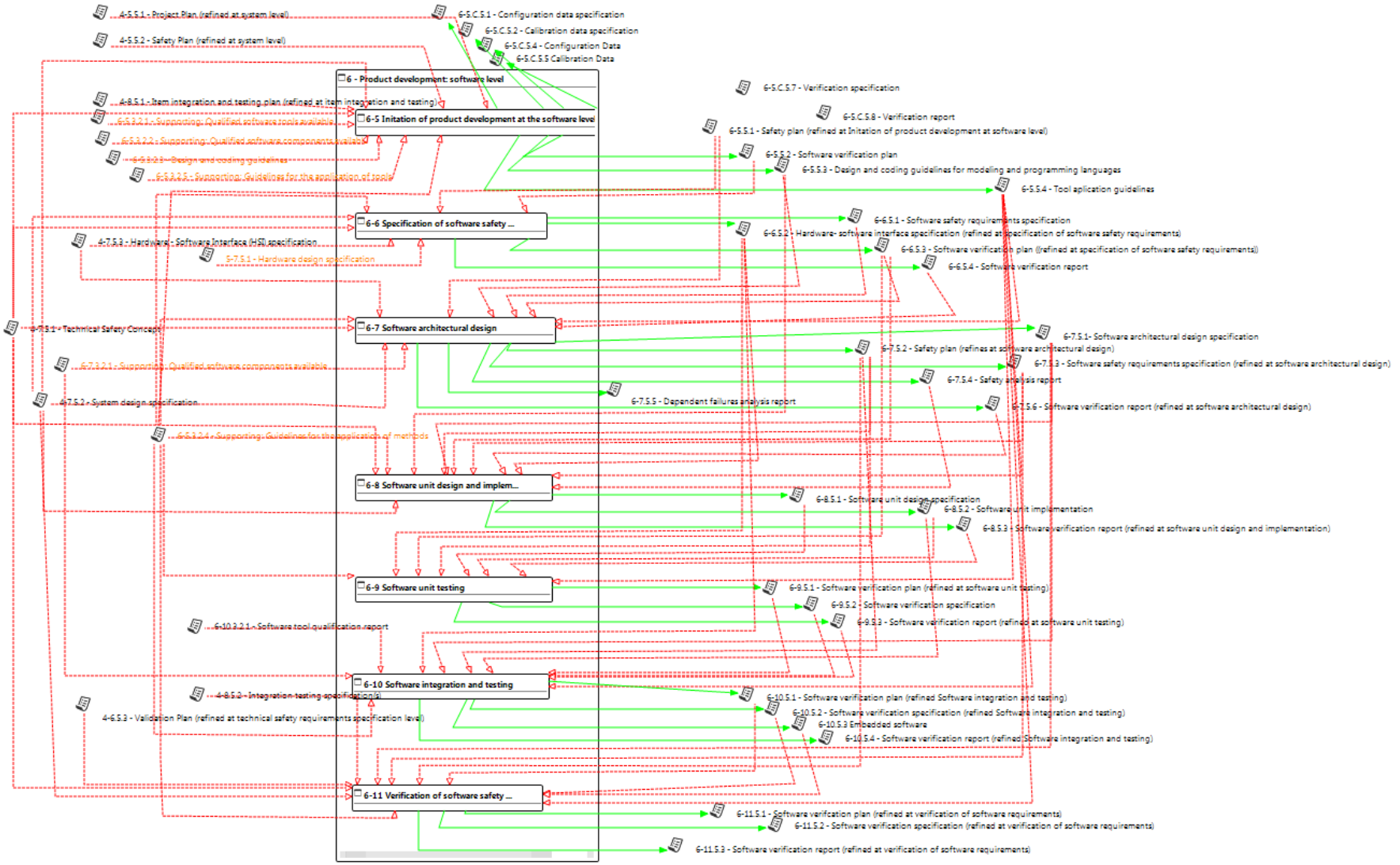


Figure 21: ISO 26262 Product development at software level in Prossurance tool

In the following chapters detailed guidelines are presented in order to address Product Development at the Software Level phase in accordance with the standard.

➢ 6-5 Initiation of product development at software level

To perform a dependent failure analysis at module level, software safety requirements resulting from this part need at least be available.

➢ 6-6 Specification of the software safety requirements

Software safety requirements can be modelled in EAST-ADL using Papyrus as shown in Figure 7. Such specification of the software safety requirements also considers constraints of the hardware and the impact of these constraints on the software.

➢ 6-7 Software architectural design

As stated in ISO 26262, the aim of the software architecture phase is to:

- Design a software architecture that realizes the software safety requirements.
- Verify the software architectural design

The aim of the software architectural specification is to represent all software components and their interactions in a hierarchical structure. Static aspects as well as dynamic aspects are described. Moreover, every software component used in the software architectural design shall be categorized as:

- newly developed
- reused with modifications
- reused without modifications. In this category, safety-related software components shall be qualified

In this subphase the software safety requirements shall be allocated to sub-systems or software components, so that each inherits the highest ASIL of any requirement allocated to it.

As shown in Figure 22, SafeAdapt methodology is aligned to EAST-ADL2 in which the implementation level is based on AUTOSAR standard. Thus, system models at implementation level specify the actual software and hardware architectures according to AUTOSAR. At the highest level of the AUTOSAR model, the definition of software architecture (set of software components and their relations) and their internal behaviour are represented by the set of runnable entities. Runnable entities are non-concurrent entities that are the allocation units of the Operating System (OS) tasks.

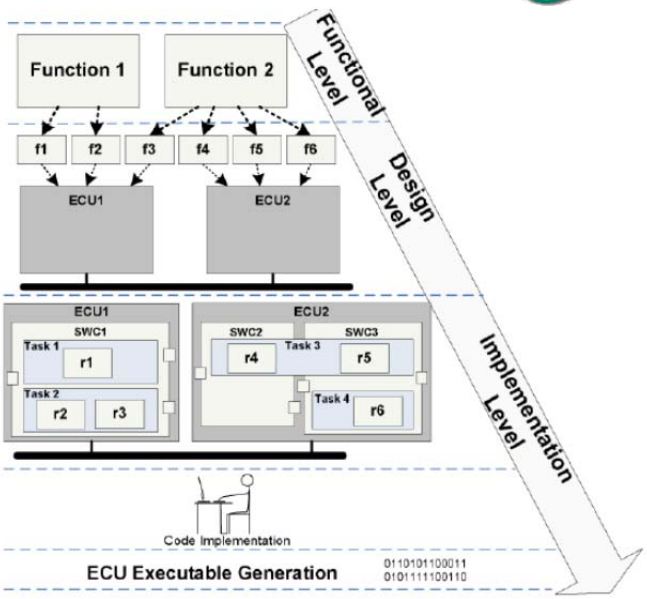


Figure 22: EAST-ADL and AUTOSAR scopes

Transformation from the design (EAST-ADL2) to the implementation (AUTOSAR) level is a crucial step. In this project, the AUTOSAR gateway tool provides an enhanced transformation from EAST-ADL2 design architecture to AUTOSAR vehicle architecture design and initial system configuration. The preliminary AUTOSAR model is generated following some predefined strategies. For instance, atomic functions from design level are transformed into runnable entities. Also hardware platform according to the AUTOSAR concept is generated from the EAST-ADL2 hardware platform specification. In addition, runnables are grouped into software components, which are allocated to the Electronic Control Units (ECUs) based on runnables allocation. Generation of software components corresponds to the compositional structure of functions. However, a final user can specify his constraints according to which runnable will be grouped in software components.

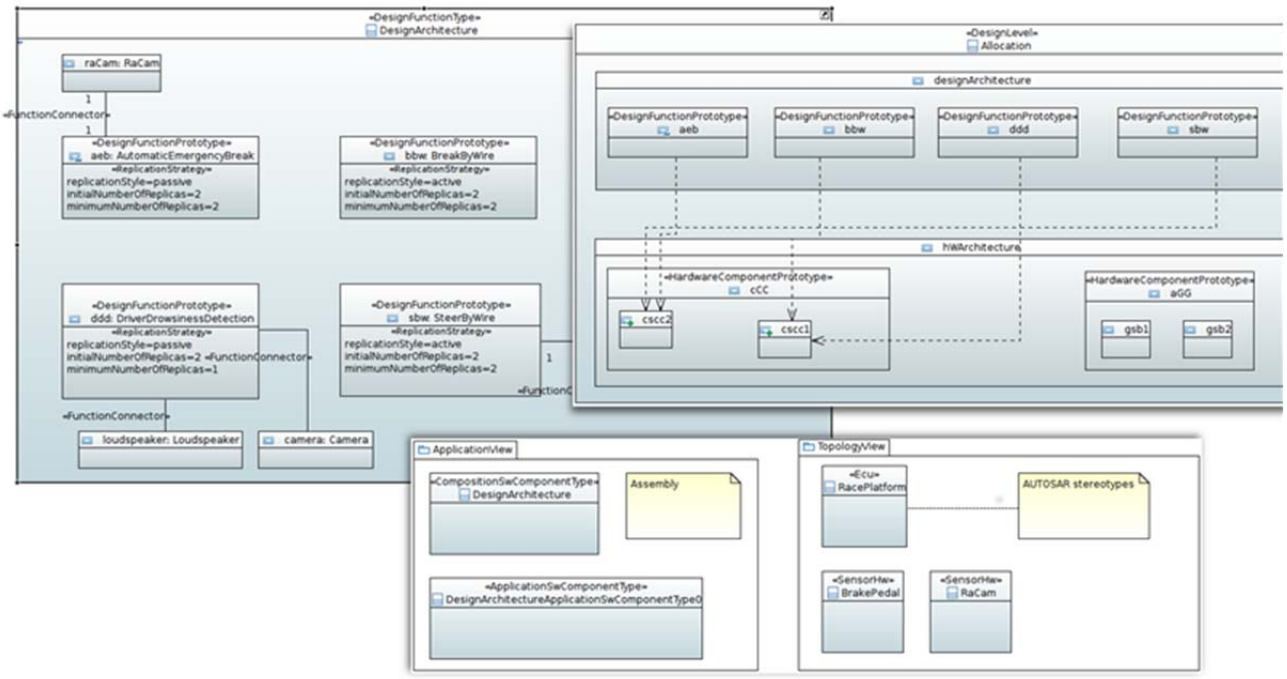


Figure 23: AUTOSAR model in AR gateway tool

The upper part of Figure 23 shows the design architecture of the SafeAdapt model along with the allocation of application components to ECUs. The lower part shows a subset of the resulting AUTOSAR model in form of a UML model applying the AUTOSAR profile. The option to generate UML has the advantage that it enables a refinement of the AUTOSAR model in the same environment as the original EAST-ADL model. In a second step, it is also possible to generate ARXML which allows exchanging AUTOSAR models between tools through arxml (AUTOSAR XML) files as shown in Figure 24

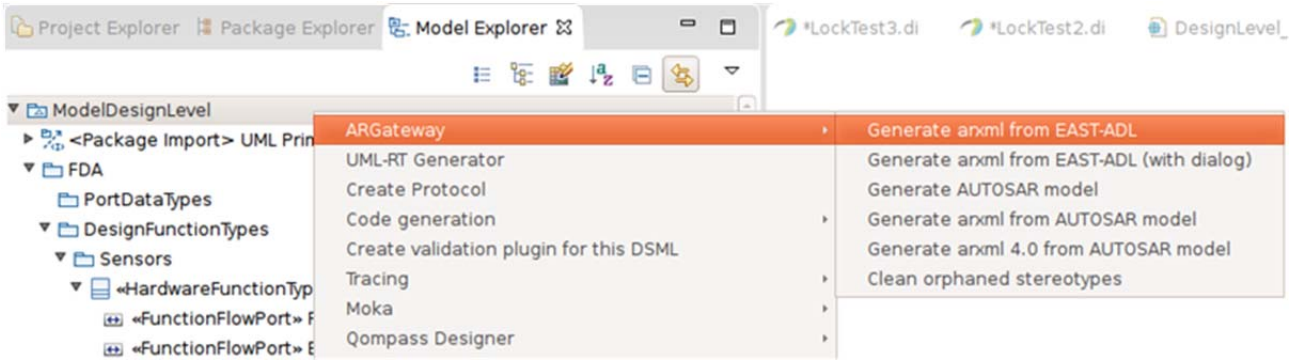


Figure 24: Generation of arxml files from AUTOSAR model in AR gateway tool

This enables further refinement with tools supporting the import of this format, e.g. Arctic Studio. Using Arctic Studio, together with information about allocation of functions to the nodes, designers can improve the configuration at implementation level. Then it can further be refined and evolved by applying design space exploration that will partition runnable entities in tasks and schedule

them in a way that will optimize system end-to- end responses. In brief, Arctic Studio tool allows designers to define:

- Hardware entities and topology with enough detail to support software configuration
- Software components with runnables
- Mapping to tasks and frames
- Mapping to ECUs and busses

Figure 25 shows a software architecture in Artic Studio.

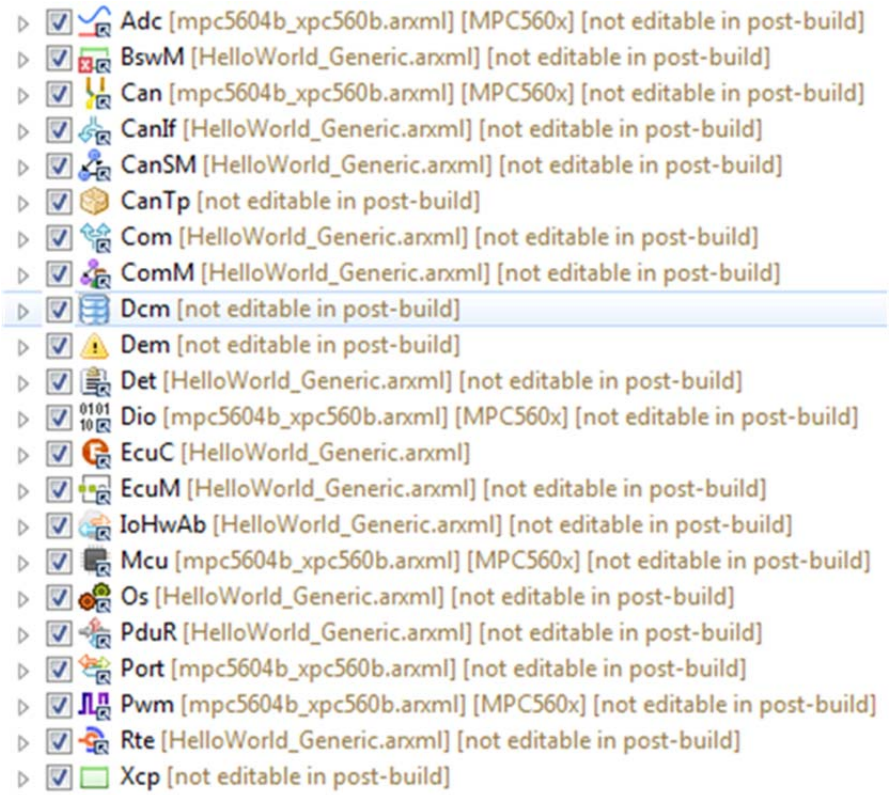


Figure 25: Software architecture in Arctic Studio tool

In addition, it is possible to define mappings at different abstract levels such as between elementary or composite functions and appropriate AR software components or runnable. Through this realization relationship, software architecture can be traced back to functions, features and requirements.

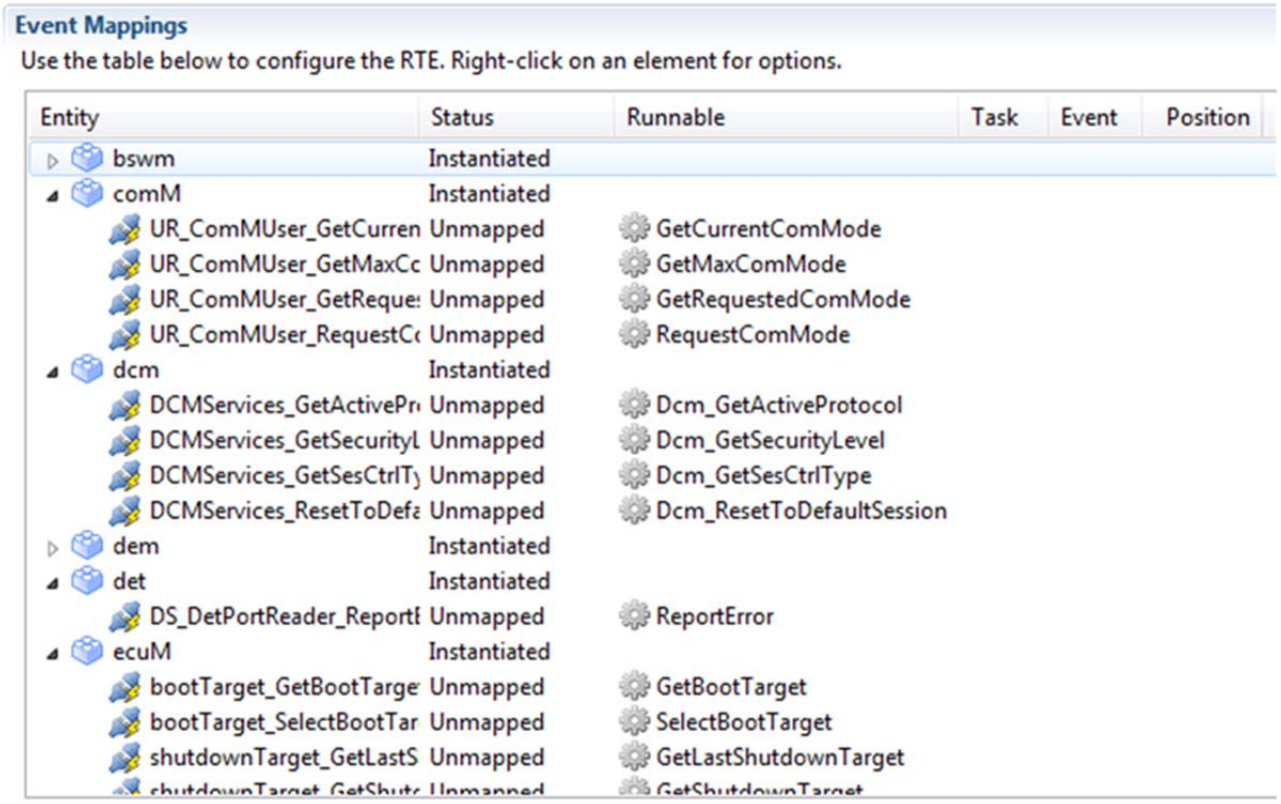


Figure 26: RTE configuration in Arctic Studio tool

From above we can deduce that Arctic Studio tool is used to represent the final software architecture of automotive embedded systems. As such, it includes the definition of the software components, their interfaces, execution timing, middleware (basic software) interactions, and so on. The model is sufficiently detailed to automatically generate and configure the platform software and integrate it on ECUs.

Additionally, developers can take advantage of the support provided by Arctic Studio in relation to safety mechanisms such as:

- Built-in self-test mechanisms for detecting hardware faults (testing and monitoring) in relation to Memory and Core
- Run-time mechanisms for detecting software faults during the execution of software (Watch Dog)
- Run-time mechanisms for preventing fault interference (memory partitioning for SW-Cs and time partitioning for applications). In this case and according to ISO 26262, part 6, clause 7.4.11, the partitioning scheme and the partitioning framework of the operating system shall be specified to ensure freedom from interference.
- Run-time mechanisms for protecting the End-to-End (E2E) communication protection for SW-Cs
- Run-time mechanisms for error handling (on the basic software and hardware)

For all implemented safety mechanisms a safety manual is needed containing the fault model according to which the safety mechanism was developed and the constraints that must be fulfilled when applying the safety mechanism.

The safety analysis method described below has to be performed during the software architecture phase in order to:

- identify or confirm the safety-related parts of the software; and
- support the specification and verify the efficiency of the safety mechanisms

The analysis has to be performed according to the description in ISO 26262 part 9, clause 8. Fault Tree Analysis (FTA) and Failure Mode and Effects Analysis (FMEA) are methods that can be used to verify the software architecture and SW safety concept. This type of analysis is done on software module level once the software architecture is released.

Hence, composeR (FMEA) and FMEDAexpress are used to either identify or confirm the safety-related parts and it supports the specification and verification of the safety mechanisms to mitigate both random hardware failures and software faults e.g. diverse software design. Such safety analyses should be performed separately for each SW module that has at least one ASIL requirement (i.e. not QM).

Software failures, as systematic failures, do not require quantitative analyses but only qualitative analyses.

To be sure about the independence between software components an analysis of dependent failures should be carried out as well. The aim of this analysis is to identify single cause that can invalidate a required freedom from interference between two software elements and lead to the violation of a safety requirement or safety goal. This type of analysis is done on software elements that could be affected by common cause failure or cascading failure.

The identification of dependent failures (systematic/random) can be supported by FMEA. Similar parts with similar failures modes that appear several times in FMEDAexpress can give more information about these types of failures. ComposeR (FTA) can help in this identification as well. Examination of cut sets or repeated identical events of a FTA can indicate potential for dependant failures. The analysis has to be performed according to the description in ISO 26262 part 9, clause 7.

Finally, if as a result of such analyses new hazards are identified, they shall be introduced and analysed in the hazard analysis and risk assessment.

➢ 6-8 Software unit design and implementation

At this phase, the goal is to specify the software units in accordance with the software architectural design and the associated software safety requirements, to implement the software units as specified and to verify the design of the software units and their implementation.

Here, the first activity is to define and establish the specific rules for design and programming to be followed in the current project. Once these rules are clear, the team proceeds to use Arctic Studio tool. With this tool it is possible to follow the standard 26262 recommendations about the use of hierarchical design and avoidance of unnecessary complexity to achieve an adequate level of

granularity. Arctic Studio includes the definition of the software components, their interfaces, execution timing, middleware (basic software) interactions, etc. The model is sufficiently detailed to automatically generate and configure the platform software and integrate it on ECUs as it is illustrated in Figure 27. The last step, called Build, compiles all the generated files and runnables and builds the executable (.elf) file which is loaded in ECU for debugging and testing. Debugging helps to check the application whether it is working correctly as anticipated. If modification or correction is required then development process can be resumed again from the previous step.

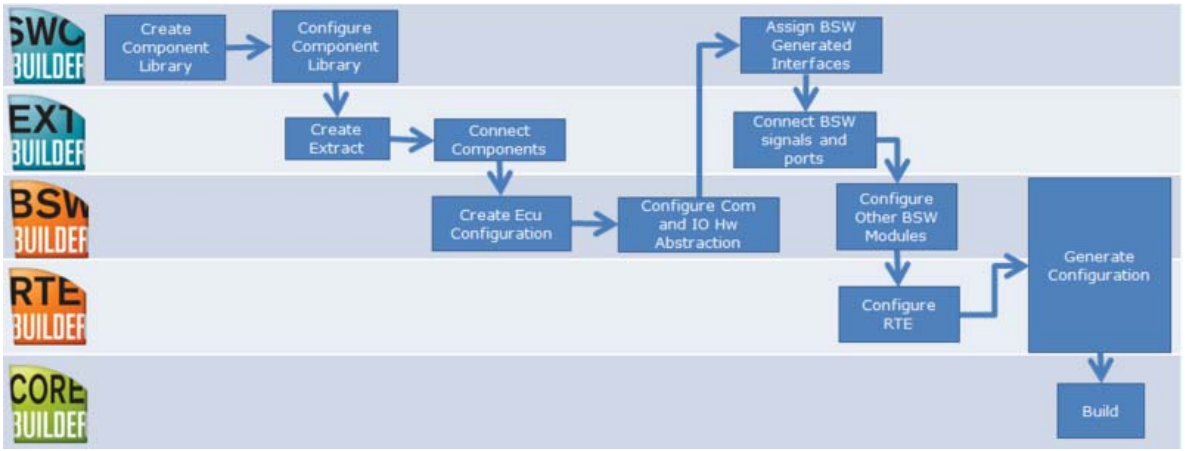


Figure 27: Arctic Studio tool workflow

In parallel, an ASIL (ASIL-A to ASIL-D) must be allocated to each function. Typically all SW functions inherit the same ASIL as the SW module in a top-down approach. Moreover, every SW function is assigned a criticality class (C0 to C3, where C0 is not safety related):

- C1: Interference free. No interference with safety related functionality
- C2: Safety relevant. Latent fault
- C3: Safety critical. Single-point fault

The combination of ASIL target and criticality defines for each SW function what kind of safety measures should be considered for implementation for that SW function with reference to ISO 26262-6:2011: Table 4 or similar. For example, if an ASIL D software module contains a function that is not safety related (i.e. does not have any safety requirement), then this function shall only implement measures to ensure independence and freedom from interference.

On the other hand, the TTEthernet tools enable the developer / design engineer to conduct seamless design, create configuration and provide data loading for TTEthernet based networks (see Figure 28). The tools capture system-level communication requirements and automatically generate network- and device configuration- files, thus enabling seamless integration with existing design processes.

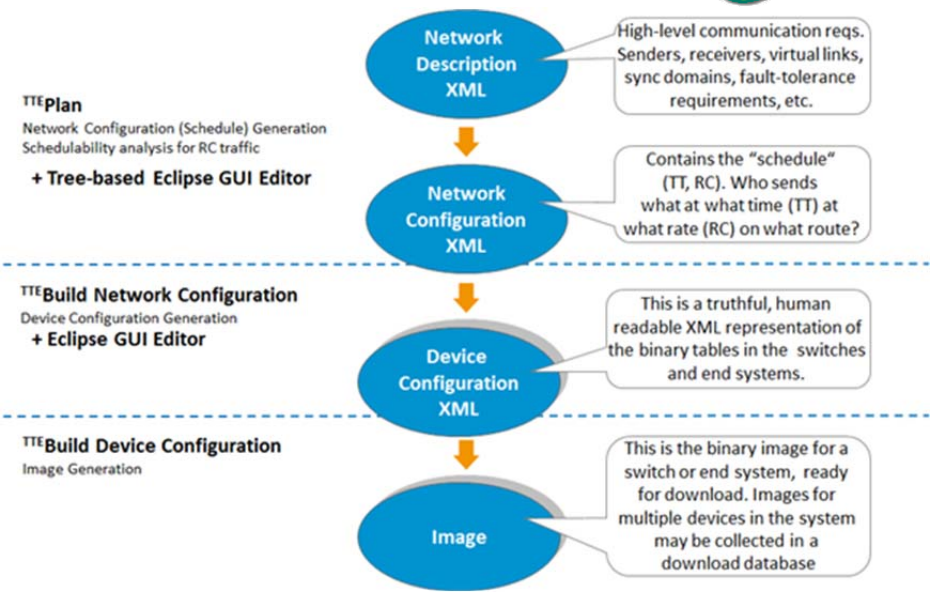


Figure 28: TTEthernet data flow for configuring a network

By using TTEPlan, TTEbuild and TTELoad, the tools generate all files and data bases needed for the TTEthernet data communication based network, including a tool for loading appropriate files (TTELoad) to the hardware of the target network. Within the following, the work flow to configure a TTEthernet based network is described (see Figure 29).

The development/design process is started by using TTEPlan. The designer will reply to a set of interactively lead input masks entering requirements and parameters. By this computer aided process the high level communication requirements for the network will be described and defined. This includes parameters for the physical and the logical topology of the network under consideration. Furthermore, the virtual links, including their IDs, timing requirements and possible frame sizes will be defined. Finally, synchronization parameters and requirements are defined (i.e. the SAE AS6802 clock). All information is stores in XML file format for further processing. The network schedule is available as soon as all steps are completed and the data is automatically processed by TTEPlan.

In a next step, the resulting XML file from the TTEPlan tool is used as an input file to TTEBuild Network Configuration tool supporting the network configuration process. Everything is implemented in a set of XML files. The network schedule calculated by TTEPlan is included in this set of XML files. The network configuration is independent from the target hardware. It simply describes all details necessary to configure the network accordingly. This comprises the schedule(s), the port assignments, the buffer allocation for all devices planned in the network (number of devices and types need to be known). Potentially it might be useful to modify or even create parts of the network configuration by third party tools, which is possible and it is supported by the TTEthernet tools.

The next step is to introduce the resulting XML file set into the TTEBuild Device Configuration Database tool. This tool generates one device configuration file per device planned in the target network (i.e. switch or end system). The TTEBuild Device Configuration Database tool provides its results both, in XML format and in binary code (different image formats, HEX or BIN can be selected) ready for direct download to the target device. The device configuration is

device/hardware specific and describes all configuration parameters at bit level. Fine-tuning of device parameters is possible at this level.

In a next step, the binary code is downloaded to the switches in the network using the TTELoad tool. TTELoad connects to the management interface of the switch and provides a safe unlocking procedure before reprogramming the static configuration memory of the switch. It also supports bootstrap configurations of the TTEthernet switches. For the End-systems the information is loaded to them using the drivers included in the End-system delivery package.

TTTech provides Eclipse plug-ins for TTEPlan and TTEBuild. With editors for all TTETool databases, as well as a schedule visualization feature, Eclipse then provides a convenient user interface for most TTETool use cases. Basic database validation and generation of validation reports is also possible using Eclipse.

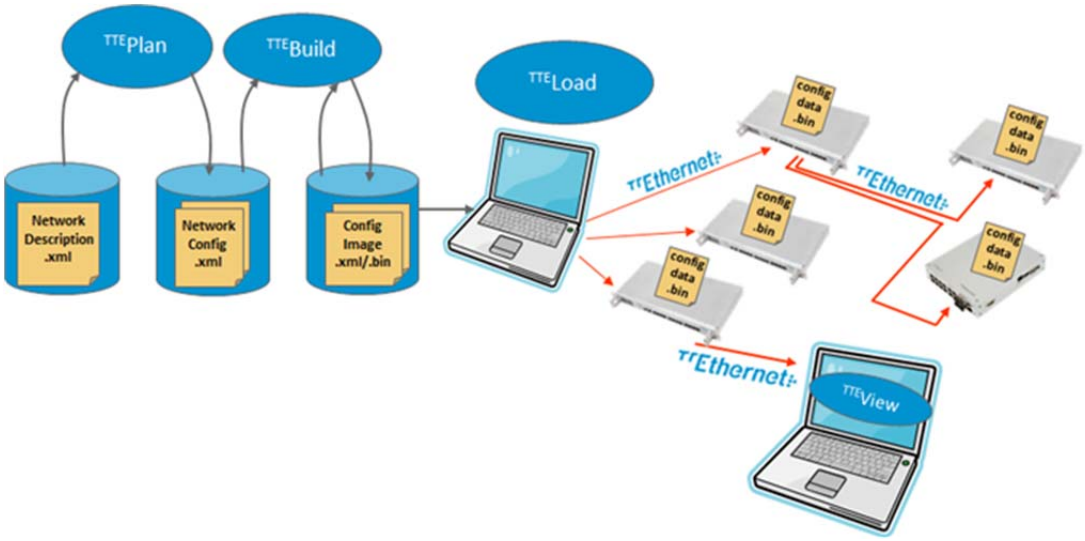


Figure 29: TTEthernet tools development suite

At the last activity of “Software unit design and implementation” phase, the detailed design and its implementation are statically verified before proceeding to the software unit testing phase. To do this, inspection and walk-through may be conducted depending on critical nature of components. Formal inspection is a technical examination process during which a product is examined with the purpose of finding and removing defects. A defect is any occurrence in a software product (design, pseudo-code, code, comments, etc.) that is determined to be incomplete or incorrect with respect to software requirements and/or program standards. While walk-throughs is a form of software peer review "in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems". Subsequently, the possible actions to take as a result of verification activities will be decided.

- 6-9 Software unit testing

Software unit tests should be executed to verify that the software units fulfil the software unit design specifications and do not contain undesired functionality. To demonstrate that there is no

unintended functionality structural coverage metrics should be measured for safety relevant software units.

In SafeAdapt, unit test cases are generated manually. However, according to the 26262 standard, Dynacar RT tool can be used as a test environment in order to execute both software-in-the-loop and hardware-in-the-loop tests. Instead of deploying separate environments to test different components of a vehicle and rebuilding and redeploying models, Dynacar RT delivers a common test platform that can be used through the whole powertrain design process, allowing rapid prototyping, implementation and real-time testing of ECUs and powertrain components since Dynacar RT supports an open architecture.

Prior running the tests, the user should set up the test environment. This process includes three main steps: the configuration and implementation of the vehicle model, the definition of the test tracks and the configuration of the driving cases.

a) Configuration and implementation of the vehicle model

At the first step, in order to determine the vehicle dynamics behaviour in a driving situation the vehicle model should be adjusted to the vehicle real parameters. By default, there are 10 vehicle configurations in Dynacar RT which can run the tests, but the parameters of any vehicle can be changed.

In Figure 30, the user selects and updates the car skin.

Figure 30: Real vehicle 3D visualization in Dynacar RT

Once the base model of the vehicle is loaded, the detailed vehicle setup can be completed through the definition of domain parameters such as: body (mass, GOG, wheelbase, track, etc.), aerodynamics, wheels, suspension & steering, powertrain and brake system.

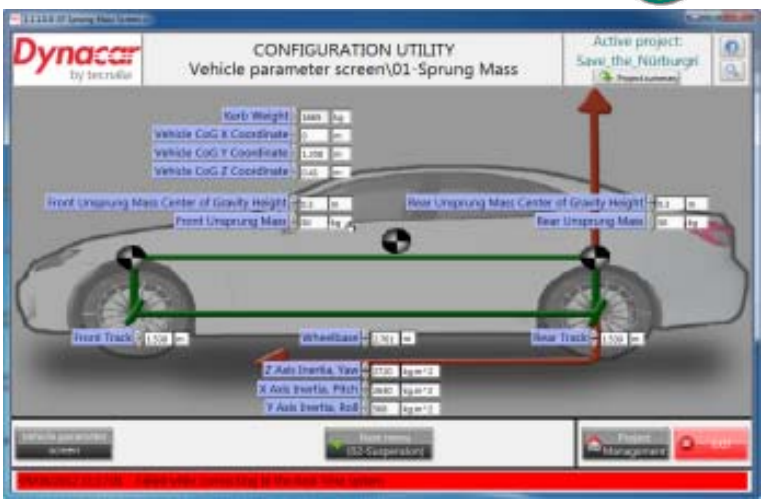


Figure 31: Vehicle parameters definition in Dynacar RT

To complete the vehicle model, the custom control algorithms and simulation models generated with other languages (Labview, Simulink, Dymola, Maplesim, C/C++, etc.) can be integrated directly into the base Dynacar RT vehicle model in dll format. In this way, engineers can customize the model by changing parameters on the fly using the graphical user interface or by plugging in their own models. This feature allows carrying out software-in-the-loop tests as shown in Figure 32.

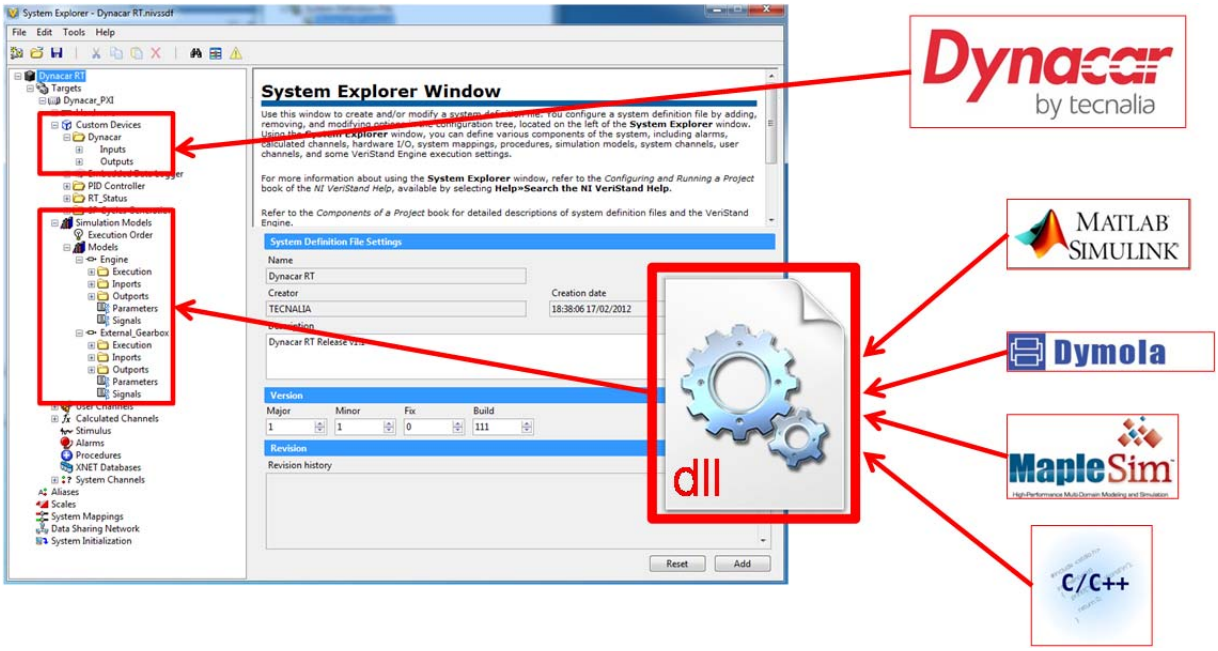


Figure 32: External model integration and virtual ECU (SIL) in Dynacar RT

On the other hand, it is possible to implement a hardware-in-the-loop testing environment by installing custom I/O hardware and its application.

b) Definition of the test tracks

At the second step, the test tracks can be accurately modelled using accurate terrain information and on site measurements. By default, 5 circuits are included such as, for instance, the test track at INTA facilities (Madrid, Spain) shown in Figure 33. However, with the Scenario Editor, users can

design and modify scenarios in several ways: importing csv file with X-Y-Z road data, importing gpx file with GPS data, importing Google maps route and adding fixed objects to the scenario. In addition, the vehicle start position within the circuit should be established.

,.

Figure 33: Accurate test track model in Dynacar RT

c) Configuration of the driving cases

At the third step, the driving tests should be configured. Dynacar RT allows to introduce different data from excel files. In order to help user generating this data Dynacar RT provides several predefined excel templates grouped in the following concepts: autonomous cycle (speed vs time, height vs distance, height vs time).

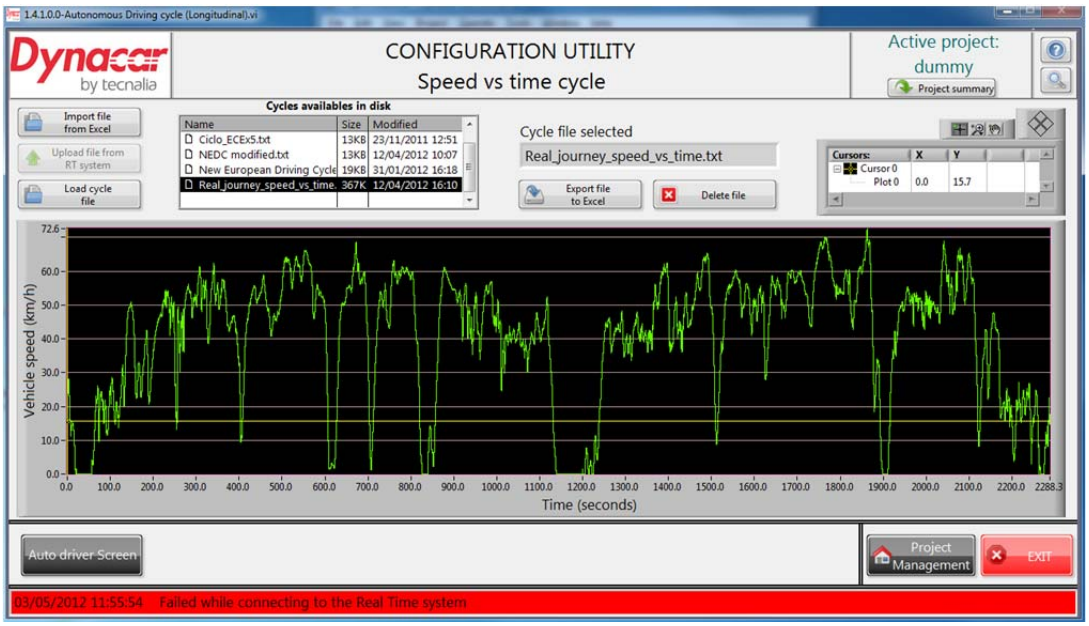


Figure 34: Example of test case input template in Dynacar RT

In addition, other model inputs/outputs can be linked in case of being provided by another model/controller installed on the same HW or can be linked using any kind of communication

(CAN, FLEXRAY, PROFIBUS, A/D, Ethernet and so on). For instance, several outputs can be used as virtual sensors that can be connected to other software or models (SIL) or hardware (HIL).

Once configuration is performed Dynacar RT is ready as a test environment. The simulation should be run in Dynacar RT in order to generate plausible vehicle behaviour data. At the first step, the user should choose the test type: with human driver or driverless just following driving cycles in autonomous mode. After that, by selecting the download project option, the simulation will start with the new parameters and the selected scenario. After the simulation a manual analysis of the readouts should be achieved by the user. These results can help developers to verify that the software fulfil the design specifications and do not contain undesired functionality.

➢ 6-10 Software integration and testing

The scope is to integrate the software components and demonstrate that the software architecture is correctly realized. Integration levels are tested against the architectural design.

To verify the software units, it is advisable to use an appropriate combination of the following verification mechanisms:

- Mechanisms for error detection at software architectural level
- Mechanisms for error handling at software architectural level
- Methods for the verification of the software architectural design

In SafeAdapt, test cases are generated manually. However, according to the 26262 standard, Dynacar RT tool can be used as a test environment in order to execute both software-in-the-loop and hardware-in-the-loop tests. Depending on the scope of the tests and the hierarchical level of integration, Dynacar RT can be used in combination with the target processor for final integration, or a processor emulator or a development system for the previous integration steps. For a detailed explanation about the use of Dynacar RT, please, consult chapter “6-9 Software unit testing”.

➢ 6-11 Verification of software safety requirements

The goal is to demonstrate that the embedded software fulfils the software safety requirements.

According to the 26262 standard, this verification can be conducted with the support of Dynacar RT tool since it works as a "Virtual Rolling Chassis" concept (or test mule virtual car). This testing shall be executed on the target hardware. For a detailed explanation about the use of Dynacar RT, please, consult chapter “6-9 Software unit testing”.

Moreover, composeR can be used as a verification method to confirm that all possible failure modes are covered by appropriate safety measures (as specified in the software technical safety concept). Hence composeR and FMEDAexpress (FMEA) are used to either identify or confirm the safety-related parts and it supports the specification and verification of the safety mechanisms to mitigate both random hardware and software faults e.g. diverse software design.

5.5 ASIL-oriented and safety-oriented analyses

The current SafeAdapt project does not fully cover this part of the ISO 26262 standard. Only main chapters addressed by the project have been included.

➢ 9-7 Analysis of dependent failures

Dependent failures are defined by ISO 26262 as failures whose probability of simultaneous or successive occurrence cannot be expressed as the simple product of the unconditional probabilities of each of them. Such dependent failures should be identified from the results of safety analyses and they can be classified as Common Cause Failure or Cascading Failure.

As shown in the following figure, a Common Cause Initiator (CCI) exists and triggers the same or different systematic software faults (2 and 2') in both SW elements, causing both to fail. The combination of the resulting failures leads to the violation of the safety goals.

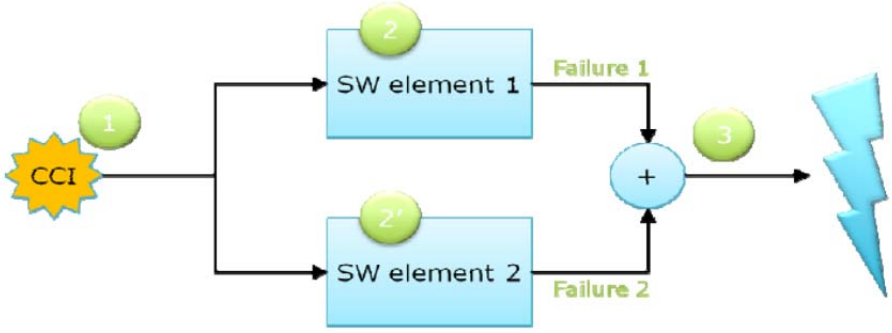


Figure 35: Common Cause Failure

In the following figure a cascading failure is shown. In this example, a fault (1) on SW element 1 leads to its failure. Due to insufficient fault containment on SW element 1 or insufficient independence of SW element 2 a coupling mechanism (2) exists, which leads to a failure of SW element 2. The combination of the two resulting (cascading) failures or the resulting (cascading) failure 2 leads to a violation of the safety goal (3).

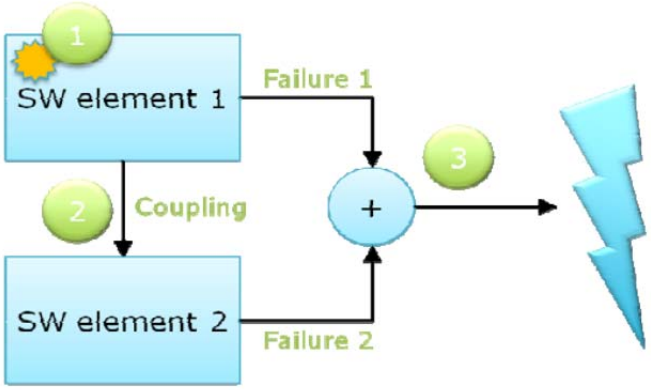


Figure 36: Cascading Failure

In SafeAdapt, the identification of dependent failures (systematic/random) can be supported by FMEA. Similar parts or components with similar failures modes that appear several times on composeR (FMEA) can give information about these types of failures. composeR (FTA) can help in this identification as well. Examination of cut sets or repeated identical events of a FTA can indicate potential for dependent failures.

Any dependability related to robust design could not be evaluated by analysis of the architecture. The realized product, a simulation of the realized design or a model which is completely validated versus the design is compulsory.

Typically the software elements that can be affected by common cause initiator are:

- The redundant or diverse elements (e.g. resulting from an ASIL decomposition)
- The software module and its safety mechanisms (e.g. software functional part and the watchdog)
- The software functions using identical SW modules (e.g. libraries)
- The software that rely on the same hardware or same input signal

ISO 26262 gives a list of possible root causes for hardware and software dependent failures. The list can be reduced to the ones related to software:

- Random hardware failures (e.g. CPU and memory structures)
- Development faults (e.g. during software development process)
- Installation faults (e.g. during configuration, integration)
- Environmental factors (e.g. impact on input values, interrupts)
- Failures of common internal and external resources (e.g. libraries)

Based on the list, the possible CCI and software faults must be retrieved.

In order to make a more systematic analysis about the possible CCI or software faults, the type of software properties that can be affected by the root causes is proposed:

- Data
- Control flow
- Timing
- Code and configuration
- Shared HW resources

Based on the matrix of root causes and affected software properties, a better systematic analysis of the possible CCI, software faults or the resulting failures can be done.

As being the search for basic events that trigger multiple top events of a fault tree analysis, Common Cause Analysis is also possible in component fault tree analysis. The identification of common causes is drastically increased since all basic events and all top events are included within one single model. In contrast to that, common cause analysis can be complex in classic fault trees since a common cause can be in multiple separated trees.

➢ 9-8 Safety analyses

Safety Analyses examine the consequences of faults and failures on items considering their functions, behaviour and design. It also provides information on conditions and causes that could bring violations to a safety goal or requirement. Last, it could indicate new hazards not found during the hazard analysis and risk assessment.

These are two most common techniques for analysing system fault modes: FMEA and FTA. The FMEA is an inductive (bottom up) approach focusing on the individual parts of the system, how they can fail and the impact of these failures on the system. FMEA starts from known causes and forecasting unknown effects while FTA is a deductive (top down) approach starting with the undesired system behaviour and determining the possible causes of this behaviour. FTA starts from known effects and pursues unknown causes.

In other words, FMEA focuses on each system component and it examines before-the-fact all things that could possible go wrong with that component. While FTA focuses on failure outcome and it examines the applicable components, processes and conditions retroactively to identify all possible contributing factors that could have worked alone or in combination to cause the outcome. FMEA and FTA complement each other. FMEA yields the possible system failures, which are the inputs of FTA (Rolf, 2006). In practise, a FTA is performed for lager systems. When a problem is detected within a certain subsystem an FMEA on the smaller subsystem is performed to find out how it behaves. Figure 37 shows how the FMEA and the FTA complement each other.

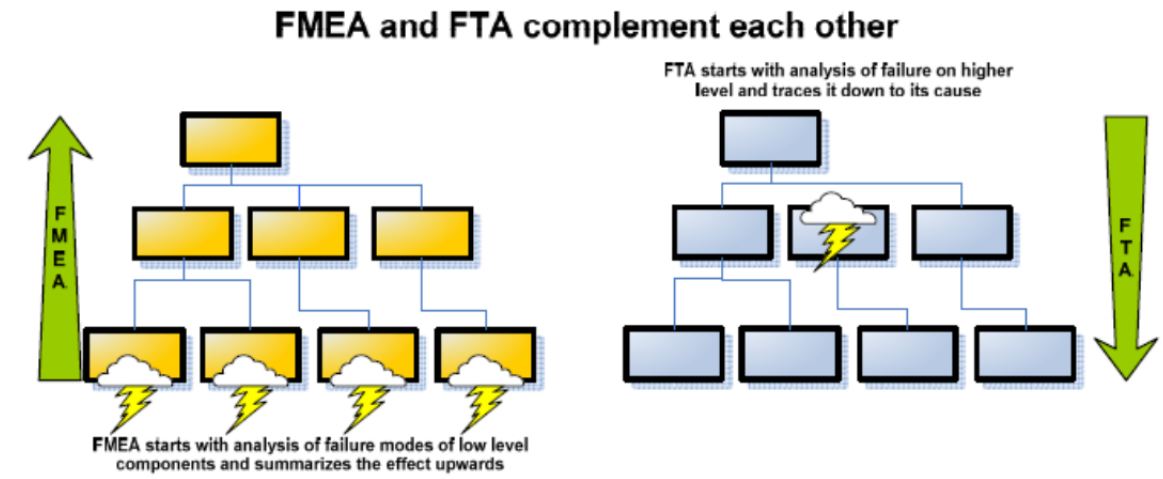


Figure 37: (a) FMEA and (b) FTA [3]

Such methods could be classified as “Qualitative” or “Quantitative”. Qualitative analysis methods identify failures without predicting how often they occur. Qualitative FMEA at system, design or process level together with qualitative FTA are some of these techniques. As ISO 26262-6:2011 depicts, this method could even be applied to software where no more appropriate software-specific analysis methods exist.

On the contrary, quantitative analysis methods predict the frequency of failures as well. The quantitative analysis methods include quantitative FMEA and FTA. Software failures, as systematic failures, do not require quantitative analyses but only qualitative analyses. These types of methods mostly address random hardware failures. They are used to verify a hardware design against defined targets for the evaluation of the hardware architectural metrics and the evaluation of safety goal violations due to random hardware failures (see ISO 26262-5:2011). Quantitative safety analyses require additional knowledge of the quantitative failure rates of the hardware elements.

Quantitative analysis, when dealing with HW random faults, is called FMEDA (Failure Mode Effect and Diagnostic Analysis). FMEDA permits to calculate the architectural metrics (Single Point Fault Metrics and Latent Fault Metrics) by introducing safety mechanisms with their diagnostic coverage (detection rate of the fault) stopping or mitigating the fault propagation as proposed in ISO 26262 Part 5.

In SafeAdapt there are two safety analysis tools: composeR (which performs FTA) and FMEDAexpress (which performs FMEDA analysis). Table 4 illustrates the tools capabilities addressing ISO 26262.

	FME(D)A	**CFT**
Availability to perform qualitative and quantitative analysis	FMEA: qualitative FMEDA: quantitative	Both
Addresses random failures	YES: FMEDA	YES
Addresses systematic failures	YES:FMEA	Yes but low of interest
Application to different architectural levels	YES	YES
Used to calculate SPFM and LFM	Yes FMEDA	Not direct
Support dependent failure analysis	YES	YES
Analysis generated from models	YES	NO

Table 4. Tool capabilities addressing ISO 26262

Such safety analysis techniques are performed at the appropriate level of abstraction during the concept and product development phases of ISO 26262. If the analysis determined that a safety goal or requirements is not compliant with, such results should be used for deriving prevention or mitigation measures for the causes of the violation.

Below the methodology to use these two safety analysis tools is given.

FMEA Analysis with composeR

Failure Modes and Effects Analysis (FMEA) (Sweeden, 2013) is a detailed bottom up inductive analysis of a system, subsystem, process, design or function so that potential failure modes, their causes and their effects can be identified. At the same time, it helps in the process of either controlling or avoiding the undesired effects of these failure modes. It focuses on the individual parts of the system, how they can fail and the impact of these failures on the system. Since this

bottom-up safety analysis method starts with a detailed list of all components within a system, their failure modes are identified and the analysis of their impact at higher levels is performed. Their effects at the highest system level are foreseen.

The composeR tool allows an XML-based import of the component structure with arbitrary hierarchy. After the component structure is being imported, component fault tree analysis and FMEDA analysis can be performed.

It usually consists of the following steps as shown in Figure 39.

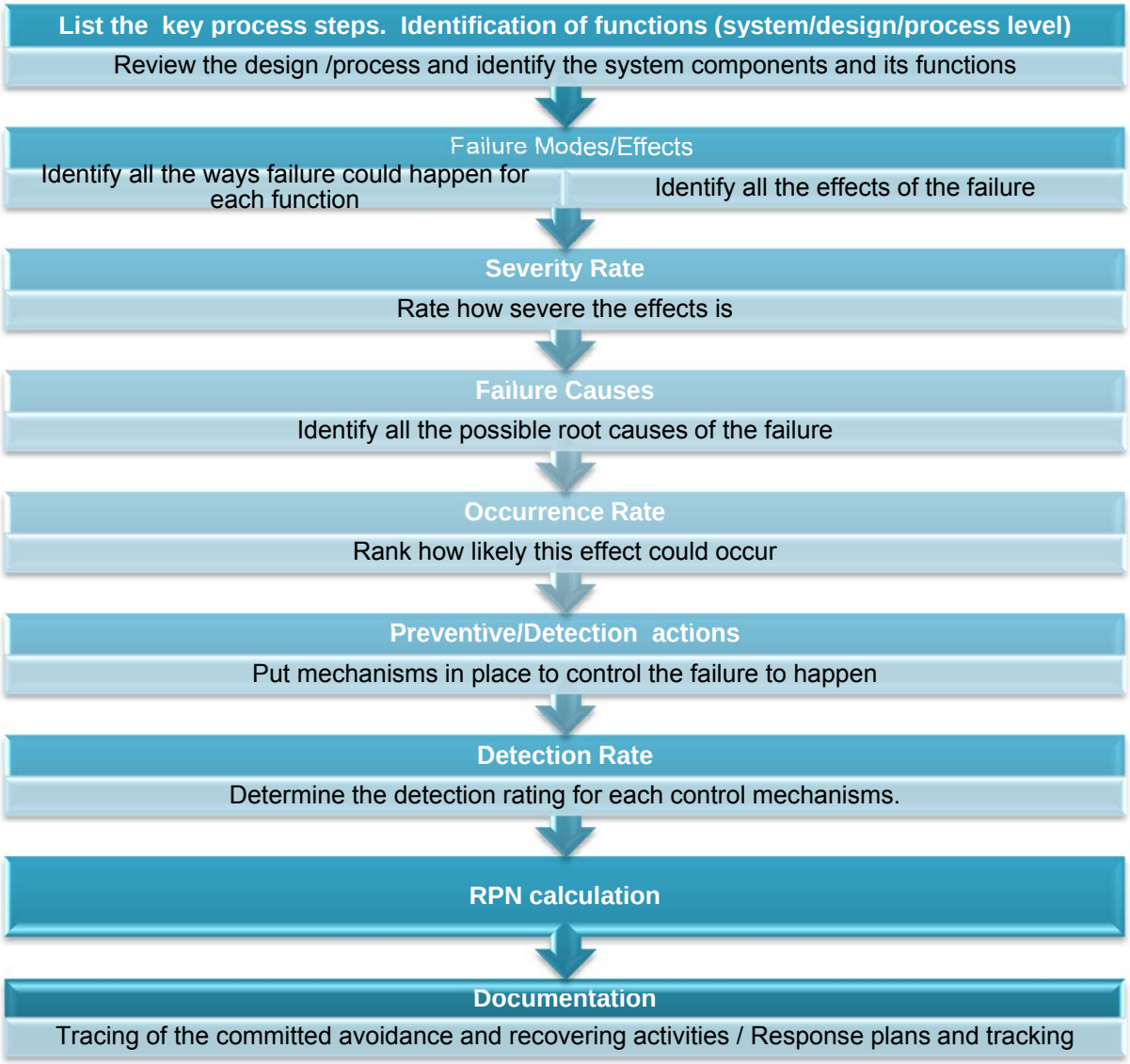


Figure 38: General FMEA analysis process

At the final step, in composeR all relevant information is documented in a worksheet as shown in Table 5.

Item/Function	Failure Mode	Failure Causes	Failure Effects	Severity Rate(S)	Occurrence Rate(O)	Detection Rate (D)	Preventive Actions/ Detection Actions	RPN	Recommended Actions

Table 5. General FMEA table

Where

a) Function: identify the functions of the scope by means of identifying the purpose of the system, design, process or service. The scope is usually divided into different subsystems, items, parts or process steps.

b) Failure Mode: fill it with all the possible failure modes that may affect the considered function. It answers the question about what could go wrong.

c) Failure Causes: define all the possible failure causes for each failure mode i.e. why would the failure happen?

d) Failure Effects: determine all the consequences on the system, related systems, process, and related processes for each failure mode i.e. what would be the consequences of failure?

e) Severity Rate (S): 1–10, 10 = most severe effect

f) Preventive Actions/Detection Actions: for each cause, it identifies how to control them by means of either procedures or mechanisms in place. Specifically, the actions prevent the failure to happen, reduce the possible likelihood or detects the failure.

g) Occurrence Rate (O): 1–10, 10 = very likely to occur

h) Detection Rate (D): 1–10, 10 = very unlikely to detect

i) Risk Priority Number (RPN): SxOxD

j) Recommended Actions: design or process changes to make severity or occurrence lower. For instance, actions to reduce the hazard rate increase the potential of finding failures…

In addition, this data is usually completed with useful information such as: controls to improve detection process, responsible person, deadlines, remarks, etc.

Component Fault Tree Analysis with composeR

FTA (Headquarters) is a top down, deductive failure analysis technique to evaluate the safety and reliability of a given system based on its architecture. It backwardly deduces the causes of a given event, discovering the root causes of failures.

The composeR tool allows an XML-based import of the component structure with arbitrary hierarchy. After the component structure is being imported, component fault tree analysis and FMEDA analysis can be performed.

During the whole development phase of safety-critical embedded systems the automation capabilities and the integration of dependability analyses into the design process can save great

effort and therefore also money. This is the reason why it is becoming a significant concern within the design community.

Model safety analysis allows an early safety assessment in the system design process. Methods such as Fault Tree Analysis or FMEA tables can be automatically generated, rather than creating them manually as it has been performed so far (A. Joshi, 2005) . Model-driven safety analyses are applied in the architecture design phase and this automation is based on an architecture design specification together with the specification of the failure behaviour of architectural components. Hence, both system level faults and design architecture itself need to be modelled together, extending the model with faults and failure modes. These faults can be classified as transient/permanent, non-deterministic/inverted/stuck-at, based on its fault propagations or dependency of faults, fault hierarchies i.e. failure mode of a component as a function of its subcomponent, digital faults (HW/SW), etc. The modelling of error propagation, error masking and filtering is carried out in such a way that the inheritance of these rules from hierarchical components is automatically achieved (Ana Rugina, 2007). Moreover the automatic generation of safety analysis from design artefacts is not directly possible.

These abstract models assist analysts in understanding in a better way how the faults are propagated through the different components of the embedded architecture and eventually cause hazardous effects at system level.

As it has been previously affirmed, there are currently commercial tools in the market addressing the assistance of performing tables and filling data. Yet the intelligent part of accomplishing either a FMEA or FTA stays quite manual and time-consuming.

Among the different model based safety analysis techniques, Siemens composeR ESSaReL (The ESSarel research project & tool) based tool comprises different analysis models. Even if ESSaReL approves Markov chains and State charts as well, these two safety analyses methods are out of scope in SafeAdapt project. Consequently only Component Fault Trees and State Event Fault Trees will be explained.

a) Component fault trees (CFT)

This method consists of modifying the classical fault tree into an extended one i.e. a modular version of traditional fault trees is proposed. In other words, independent sub-trees are considered as modules being the main goal the generation of fault trees for different system components in order to combine these ones afterwards to get the results for the system. It is important to point out that many components have the same probability distribution (Weibull, Exponential, etc.) and parameters are modelled only once.

CFTs are classified and can be instantiated in different projects. The analysis is conducted at architectural level, constructing a system level CFT based on the previous defined architectural specifications. Besides all the architecture elements are commented with low-level CFTs.

One of the main differences compared to the traditional fault tree is the appearance of two new symbols. Additionally, each component can have input and output ports. These new symbols are the basis to make an interconnection between components and higher-system levels.

Fault Tree Gates, the ones from traditional FTA together with input and output ports are shown in Table 6.

Gate	CFT Symbol (IEC 61025)	Description
AND	&	The output event occurs if all input event occur
OR	>=1	The output event occurs if at least one of the input events occurs
Voter gate	k:n	The output event occurs if k or more of the input events occur
Input failure ports (Causal Inport)		Describe possible points for failure propagation
Output failure ports (Causal Outport)		Describe possible points for failure propagation
Internal fault events		Similar to basic events
XOR gate	=1	The input event occurs if all input events occur and an additional conditional event occurs
NOT gate	Not	The output event occurs if the contrary of the input occurs
Causal Edge		Connection edge

Table 6. Symbols for CFT

Figure 39 presents an example of a CFT.

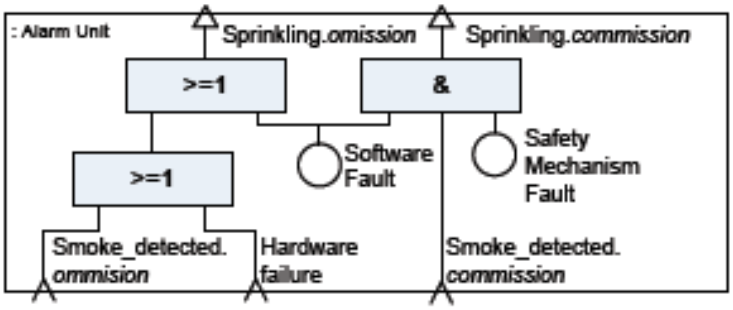


Figure 39: CFT example

The information regarding a component can be developed independently and stored into a XML library.

Still both quantitative and qualitative analysis can be carried out whereby CFT and Binary Decision Diagram (BDD) concept can be applied as well. Qualitative analyses determine which combinations of failure need to happen simultaneously so that the top event is caused. On the other hand, quantitative ones compute the global probability of failure of the top event calculating it from the basic events up to the top.

CFT used in ESSaReL supports graphical specification and efficient evaluation of CFTs via probabilistic evaluation and minimal cut set analysis. Another feature is the feasibility of defining a partial fault tree for each output failure port; whereas CFT can be appraised as a function of the input ports and the internal fault events.

Two of the most important advantages of this automatic method are that repeated events are represented only once and the “Cause and Effect Graphs” can contain several top-events which is not possible at all in the classical fault trees. Having several top events allow the examination of several failure modes and their influence at a time. As stated before, the top event probability is calculated by the standard algorithms.

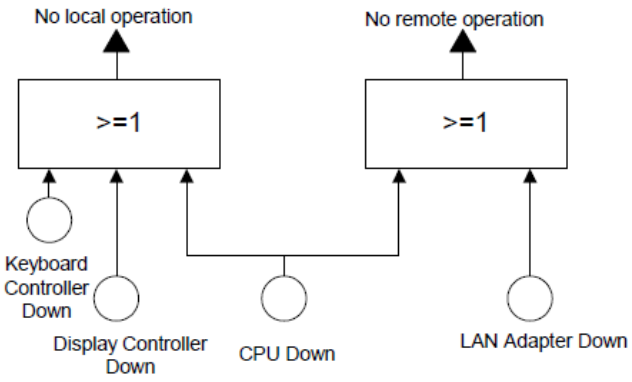


Figure 40: Two top-events (B. Kaiser, 2003)

In the same way traditional fault trees are extended into CFTs, FTs could be extended into SaveCCM models (Grunske, 2006) (M. Kerholm, 2007).

Thanks to this extended mode, fault tree generation is not anymore manually generated, allowing saving large quantities of time, money and effort. Furthermore the possible failure propagation between different components and their dependencies are examined. Considering their names and types, input and output failure ports are matched as well.

In short, the main features of the proposed CFT method are detailed below (Han, 2008).

Method	Fundamental Modelling Formalism	Graphical/Textual Modelling	Reuse of Safety Evaluation Annotations	Modelling of Architectural Dependencies	Masking, Filtering, and Renaming of Error/Failure Propagation	Modelling of Interaction Between Errors and Operational Modes
CFT	Purely-Event-Based	Graphical modelling (models are saved in a XML based representation)	CFTs are error types and can be instantiated multiple times	Should be specified in the underlying architectural model	Extra modelling for masking, filtering and renaming required	Supported only by a recent extension of CFTs

Table 7. CFT modelling support architecture-based

Method	Identification and Specification of Hazard Conditions & Safety Requirements	Architecture Specification including Architectural Dependencies	Identification of an Error Model of a Basic Architectural Components	Generation of ErrorModels for Hierarchical Components
CFT	Hazard conditions can be identified with SHARD and specified directly in the CFT formalism	General purpose language, limited support for architectural modelling in ROOM and SaveCCM	Based on SHARD & IF-FMEA	Generation of hierarchical CFTs based on name matching of incoming and outgoing failure ports with limited support of architectural dependencies (currently communication connection only)

Table 8. CFT process support of architecture-based safety evaluation

Method	Tool Description	Automatic Support for the Generation of Error Models of Hierarchical Components	Probabilistic Model Analysis (Tool Back-end)	Generation of Standard Fault Trees	Generation of FMEA tables
CFT	UWG3 & ESSaReL (Windows-based with drag and drop GUI)	Manual tool guided generation of error models for hierarchical components	Probabilistic evaluation by translation of the CFTs into BDD	Automatic flattening of CFTs to standard fault trees	Currently not supported, however FMEA table generation similar to should be possible

Table 9. CFT support of architecture-based safety evaluation

A component fault tree is a Boolean model associated to system development elements such as components. It has the same expressive power as classic fault trees. As classic fault trees, also component fault trees are used to model failure behaviour of safety critical systems. This failure behaviour is used to document that a system is safe and can also be used to identify drawbacks of the design of a system.

A separate component fault tree element is related to a component. Failures that are visible at the outport of a component are models using Output Failure Modes which are related to the specific outport. To model how specific failures propagate from an inport of a component to the outport,

Input Failure Modes are used. The inner failure behaviour that also influences the output failure modes is modelled using the gates NOT, AND, OR, and Basic Event.

Every component fault tree can be transformed to a classic fault tree by removing the input and output failure modes elements. Figure 41 (a) shows a classic fault tree and Figure 41 (b) a component fault tree. In both trees, the top events or output events TE1 and TE2 are modelled. The component fault tree model allows, additionally to the Boolean formulae that are also modelled within the classic fault tree, to associate the specific top events to the corresponding ports where these failures can appear. Top event TE1 for example appears at port O1. Using this methodology of components also within fault tree models, benefits during the development can be observed, for example an increased maintainability of the safety analysis model (Jessica Jung, 2013).

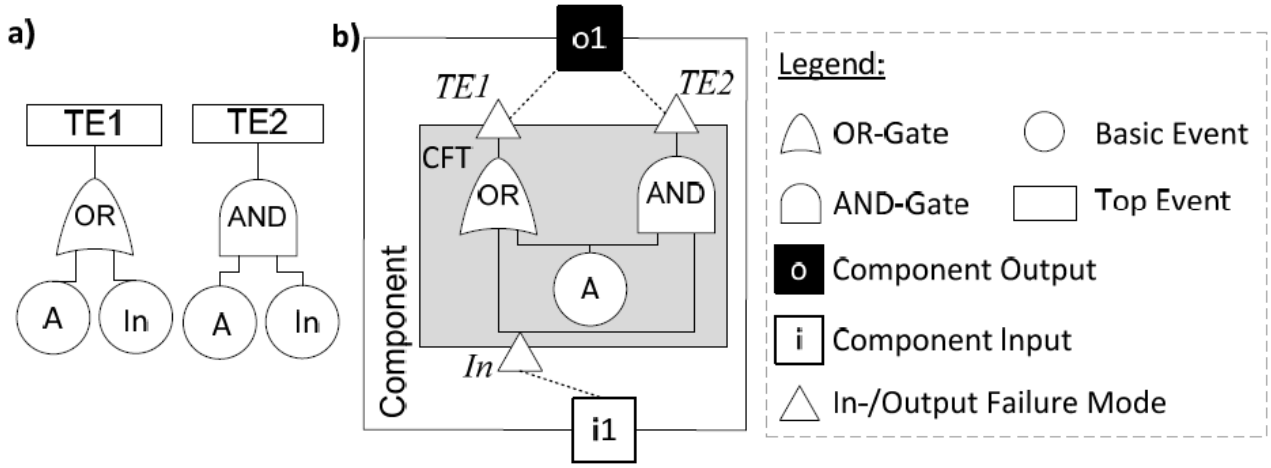


Figure 41: Example of a system FTA with composeR. (a) Classic Fault Tree and (b) Component Fault Tree. For more information, please see (Jung, J.; Hoefig, K.; Domis, D.; Jedlitschka, A.; Hiller, M., Oct 2013).

FMEDA Analysis with FMEDAexpress

Figure 42 shows a screenshot of a tool implementing the FMEDA methodology. This tool allows splitting up the analyzed system into assemblies. Each assembly holds a certain set of parts to be analyzed. Each part has a list of associated failure modes, effects and measures. The failure modes come from a part list which contains parts and their failure modes. Every time the part is used in an assembly, a new instance of that part is generated. If the failure modes of a part are altered, the changes are automatically distributed to all references where the part is implemented. The effects of a failure mode can also be reused. Once initiated, an existing failure mode can be selected via a drop-down menu and associated to another failure mode. The same mechanism is available for the measures. Furthermore, local effects are supplied as a textual field to add a reason for the reuse of an effect. Another feature that is not available in classic FMEDA analyzes is to color code a failure mode. Here we used the classifications done, in progress and critical to mark failure modes as already analyzed, to be investigated further, e.g., by performing test, and as critical if redesign is required. For further information, please take a look at (Höfig, 2014) and (Reliability Engineering Resource Website).

As previously stated, this tool enables the calculation of hardware architectural metrics required by ISO 26262-5:2011.

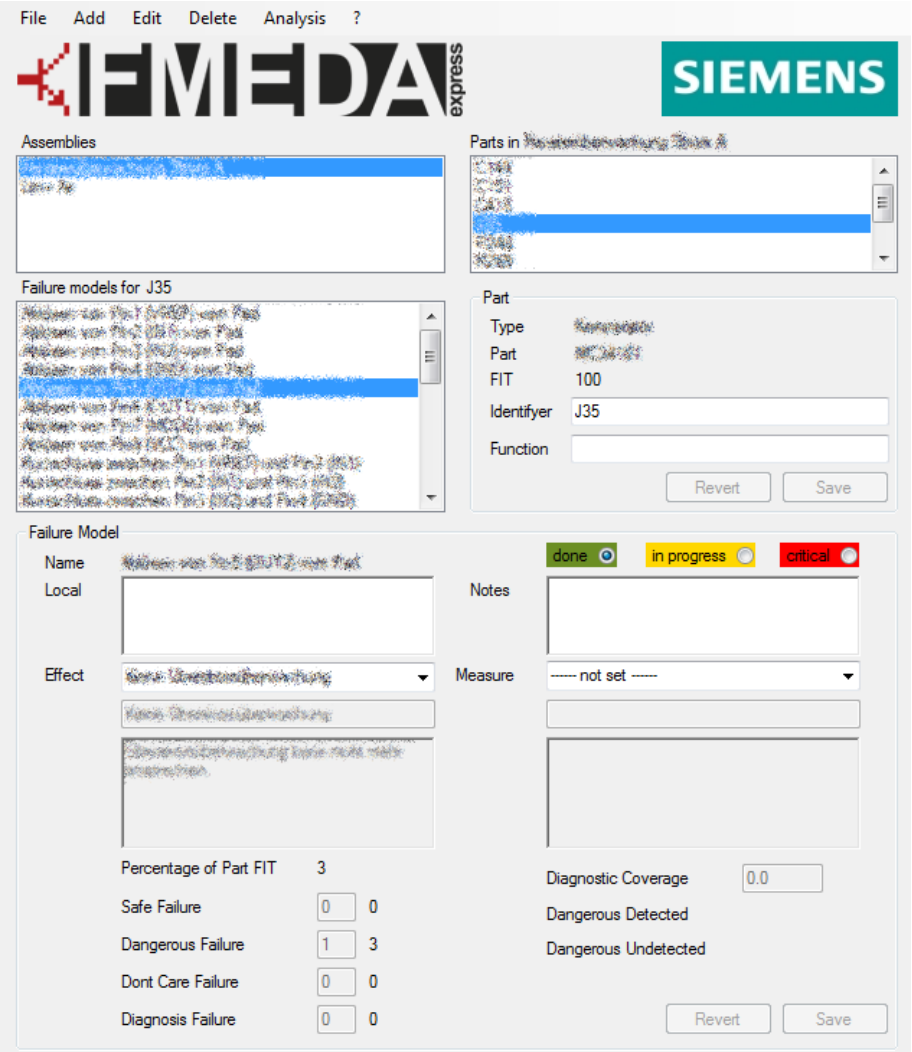


Figure 42: Screenshot of the FMEDAexpress interface

6 Examples of Tool Chain Use

In this section, the most common ways of using the SafeAdapt methodology by industry are presented.

6.1 Model-to-code approach

This paragraph outlines how to use the SafeAdapt toolchain in order to generate the system code thanks to the model transformation mechanisms. Figure 43 shows the collaboration of tools in the context of code generation.

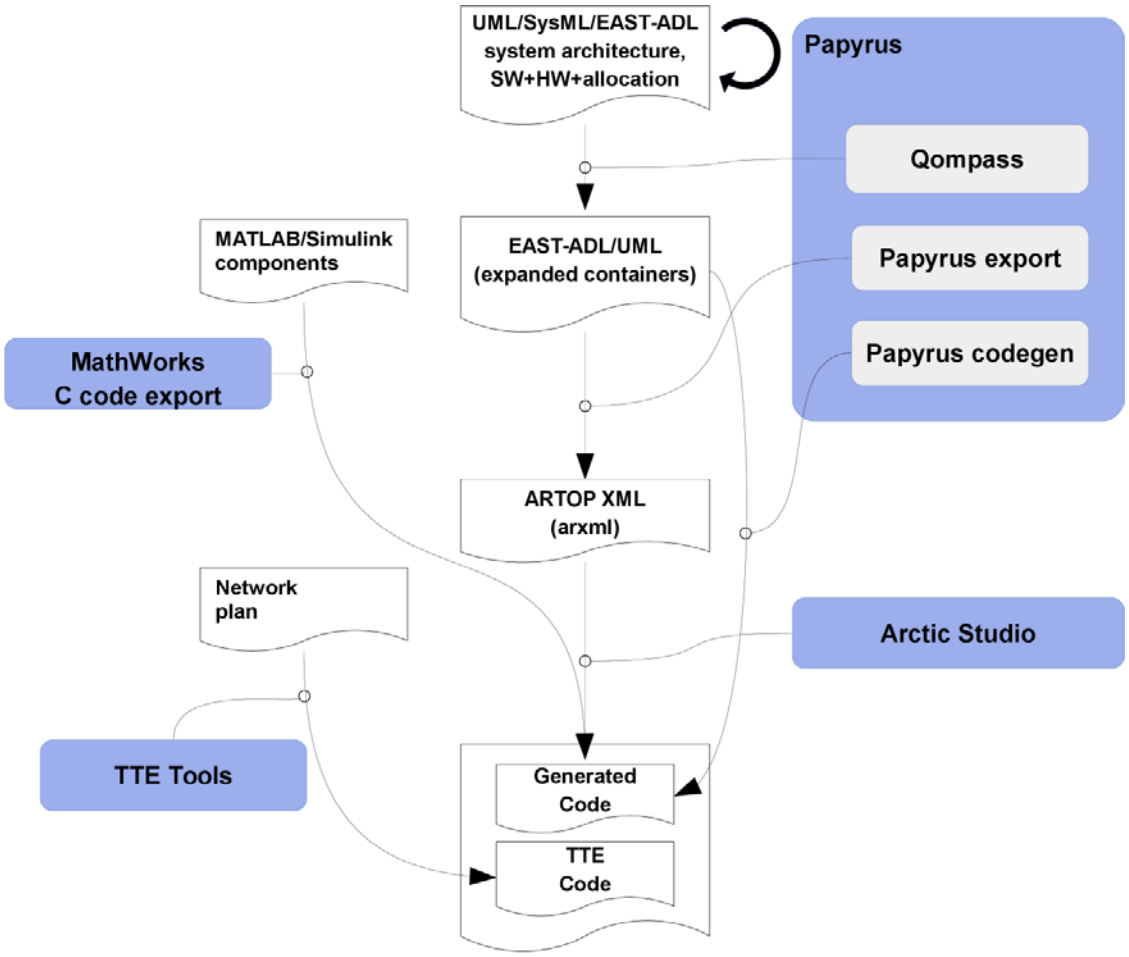


Figure 43: Tool chain with focus on code generation

The starting point is the UML/EAST-ADL model. The Qompass model transformations can add so-called containers: additional composite classes that encapsulate the original components and can provide additional services, notably reflective data.

From this model, Papyrus export utility can be used to produce an AUTOSAR model, notably a model using the ARTOP XML format. This model might need manual refinement, and then target code can be produced using an AUTOSAR implementation like Arctic Studio. In the context of SafeAdapt, several components are implemented using MATLAB Simulink and MathWorks tools can produce resulting C code

6.2 Model-to-simulation approach

This paragraph outlines how to use the SafeAdapt toolchain in order to simulate and analyse the behaviour of the system model, as shown in Figure 44. Simulation enables the designer to quantitatively assess deployment and reconfiguration strategies. There are two simulation tools, UNISIM-VP and ERNEST.

UNISIM-VP can simulate a vehicle distributed computing system including ECUs and networks (CAN, LIN, FlexRay, etc.). UNISIM-VP simulator provides instrumentation capabilities and hardware fault injection mechanisms (ECU, memory, interconnects and peripherals). The simulation employs the same binary code that is also used for target hardware. Thus, code is produced in the same way as described in the previous paragraph (since the currently chosen processors for the demonstrators are not supported yet, the code needs to be recompiled for a processor architecture supported by UNISIM-VP). Besides the code, a scenario and platform descriptions are inputs for the simulation. The scenario drives simulation that produces an execution trace. The scenario could contain different fault conditions (e.g. ECU not responding, sensor defect, transient failure). These faults can be for instance produced in a specific memory region or CAN messages. Simulation can serve to validate the software and reconfiguration strategy in terms of time (deadline, reconfiguration delay, bandwidth and data traffic) and memory space. In comparison to the demonstrators, the simulated distributed system may contain more than two ECUs and thus enabling more complex (failure) scenarios.

In case of ERNEST, a different code is generated from the System model (partly SystemC) which is then simulated at a higher level of abstraction. Compared to UNISIM-VP, simulation results are obtained more quickly but reflect less platform details.

In both cases, the results have an impact on the original system model, which gets updated in order to make a new iteration of the process.

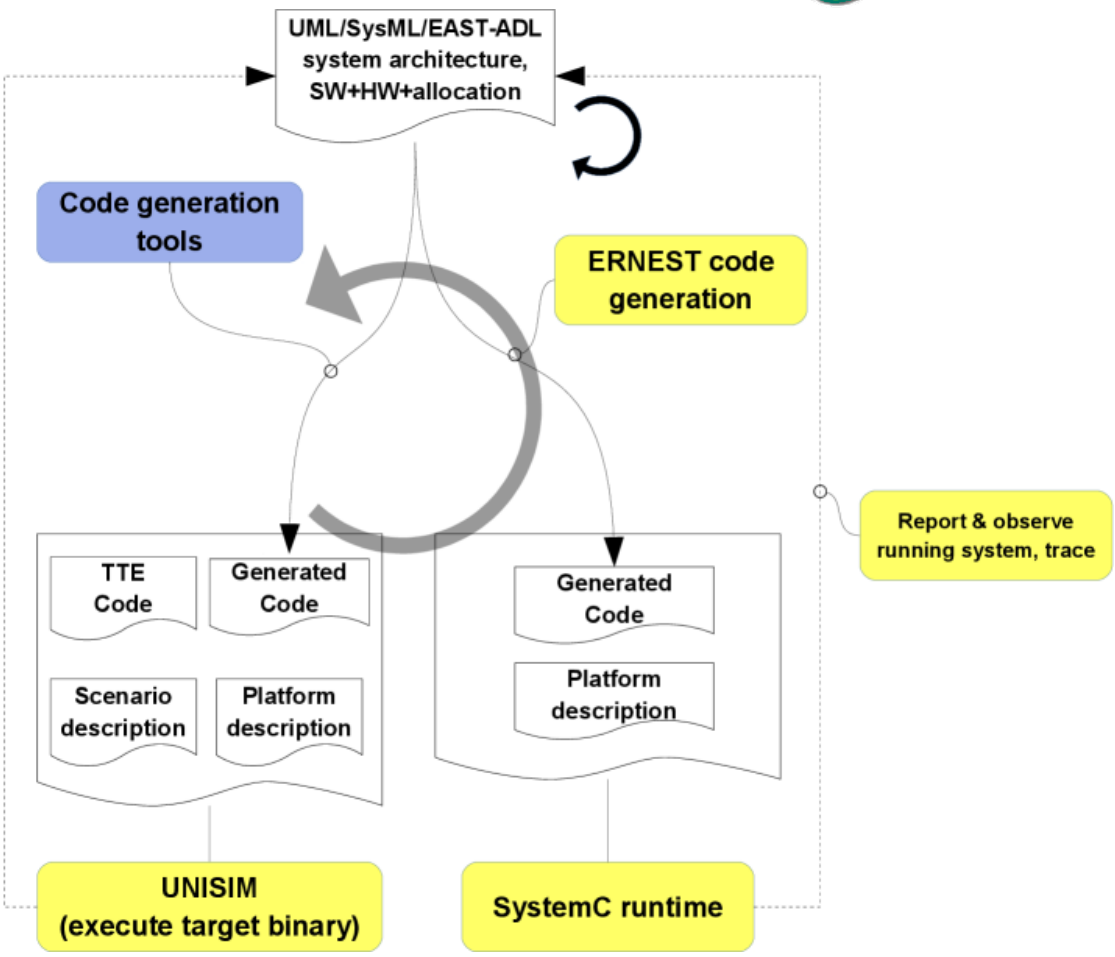


Figure 44: Tool chain with focus on simulation

6.3 Risk analysis approach

This paragraph outlines how to use SafeAdapt in order to fully address ISO 26262 throughout the system lifecycle (specification, design and implementation phases). Figure 45 shows the tool chain with a focus on this lifecycle. The generated code, including for instance the MATLAB components is used for a Dynacar integration test. The test results are available in form of an XML file with additional data. FMEA express (optionally FMEDAexpress) and composeR can be used to create an FMEA analysis report from this data. The report is then used to update the EAST-ADL model and the MATLAB component.

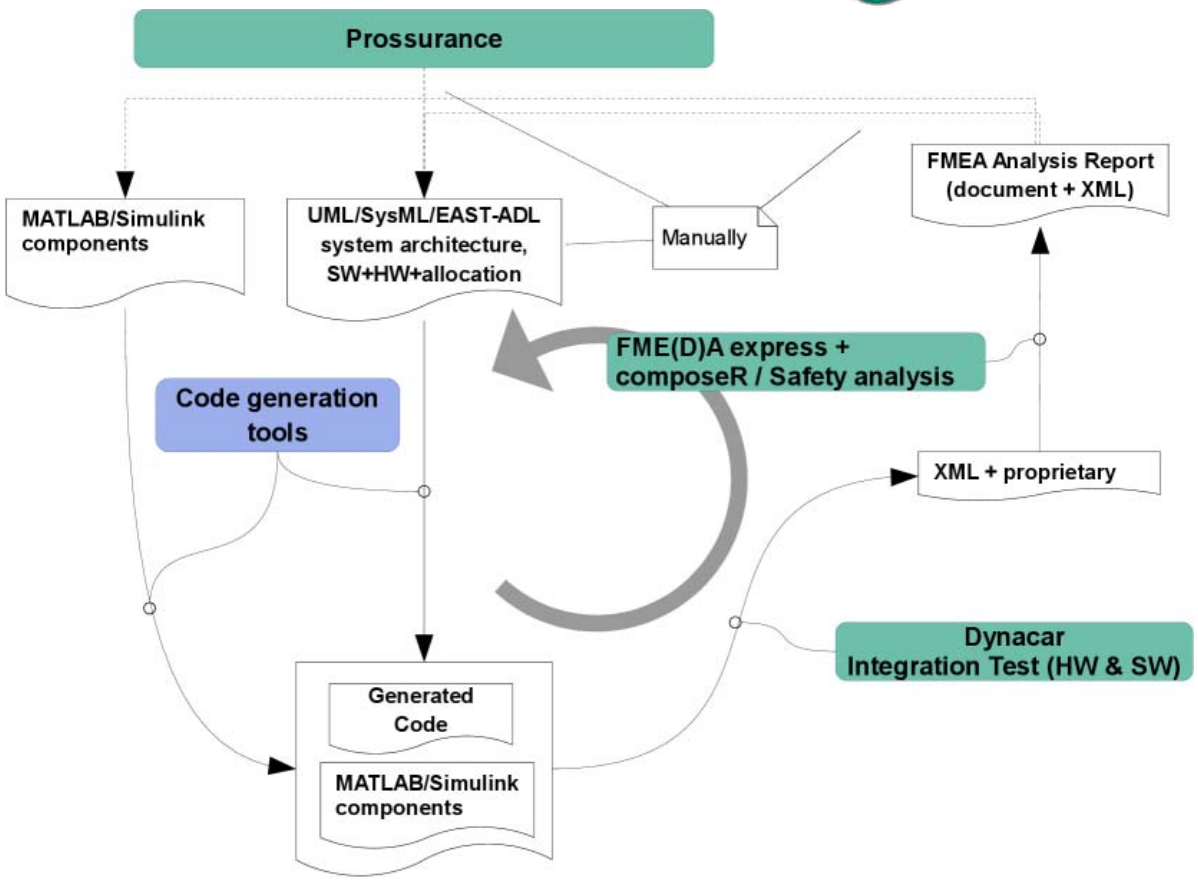


Figure 45: Tool chain with focus on safety-analysis tools

7 Conclusions

In this document, the processes and tools that are used within the SafeAdapt project have been shown. There is a focus on the verification and validation aspects for safe adaptive embedded systems, since these are of particular importance for the certifiability of adaptive embedded systems. In the context of the automotive domain the international ISO standard 26262 must be respected, enabling the more efficient creation of safety-critical systems.

Three representative exploitations of the tool chain have been shown, namely code generation for target, code generation for simulation and safety/risk analysis.

Several existing tools, partly provided by the partners of SafeAdapt need to collaborate to achieve this goal. The use of well accepted international standards, namely UML/SysML, EAST-ADL, MARTE, ARText and AUTOSAR mitigates interoperability problems. However, the used tools have different input and output data format and gateways are needed to render them compatible. While we will provide some gateways (for instance from the central UML/EAST-ADL architecture model to AUTOSAR), the focus of SafeAdapt project is not on tool integration. Therefore, the data for some tool will be converted manually as a proof-of-concept for their conceptual interoperability.

Bibliography

(n.d.).

A. Joshi, M. W. (2005). *Model-Based Safety Analysis Final Report.*

al., M. K. (2007). The SAVE approach to component-based development of vehicular systems. *Journal of Systems and Software*, 80(5):655–667.

Ana Rugina, P. F. (2007). *Dependability Modelling with the Architecture Analysis and Design Language (AADL).* Technical report, CMU/SEI-2007-TN-043.

ARText. (n.d.). Retrieved from https://www.artop.org/artext/

AUTOSAR. (n.d.). Retrieved from http://www.autosar.org/

B. Kaiser, P. L. (2003). *A New Component Concept for Fault Trees.*

Commission, U. N. (1981). *Fault Tree Handbook.*

Cuenot, P.; Adler, N.; Otten, S. (2013, 02 28). *Safe Automotive soFtware architEcture (SAFE).* Retrieved from WP3 - Deliverable D3.2.2 Proposal for extension of Meta model for hardware modeling.

Dardar, R. (n.d.). *Building a Safety Case in Compliance. Master Thesis in Intelligent Embedded Systems.*

EAST-ADL. (n.d.). Retrieved from http://www.east-adl.info/

Grunske, L. (2006). Towards an Integration of StandardComponent-Based Safety Evaluation Techniques with SaveCCM. *Second Int. Conf. on Quality of Software Architectures, QoSA 2006, volume 4214 of LNCS, pages 199–213. Springer.*

Han, L. G. (2008). A Comparative Study into Architecture-Based Safety Evaluation Methodologies using AADL's Error Annex and Failure Propagation Models. *11th IEEE High Assurance Systems Engineering Symposium, Faculty of ICT.* Swinburne University of Technology, Hawthorn, VIC 3122, Australia.

Headquarters, B. V. (n.d.). *The Powers of Fault Tree Analysis.*

Höfig, K. Z. (2014). metaFMEA-A Framework for Reusable FMEAs. In *proceedings of the 4th International Symposium on Model Based Safety Assessment (IMBSA).*

IEEE Std. 1028-2008. (n.d.). *IEEE Standard for Software Reviews and Audits, clause 3.8.*

II, C. E. (n.d.). Hands on the ISO 26262 Standard. Functional Safety In Automotive Electronics.

International Organization for Standardization (ISO). (2011). ISO/DIS 26262: Road vehicles - Functional safety.

Jessica Jung, A. J. (2013). A controlled experiment on component fault trees. In J. G. Friedemann Bitsch, *Computer Safety, Reliability, and Security, volume 8153 of Lecture Notes in Computer Science* (pp. 285-292). Heidelberg: Springer Berlin.

Jung, J.; Hoefig, K.; Domis, D.; Jedlitschka, A.; Hiller, M. (Oct 2013, Oct). Experimental Comparison of Two Safety Analysis Methods and Its Replication. *Empirical Software Engineering and Measurement. ACM / IEEE International Symposium*, (pp. 223,232).

MARTE. (n.d.). Retrieved from http://www.omgmarte.org/

OPENCOSS project. (n.d.). Retrieved from www.opencoss-project.eu

Reliability Engineering Resource Website. (n.d.). Retrieved from http://www.weibull.com/basics/fault-tree/

Rolf, I. (2006). Fault-Diagnosis Systems. In *An Introduction from Fault Detection to Fault Tolerance.* Springer.

SafeAdapt. (2015, March 17). *Safe Adaptive Software for Fully Electric Vehicles*.

SESAMO Project. (n.d.). Retrieved from http://sesamo-project.eu/

Spanfelner, B.; Richter, D.; Ebel, S.; Wilhelm, U.; Branz, W.; Patz, C. . (Mai 2012). Challenges in applying the ISO 26262 for driver assistance systems. *5. Tagung Fahrerassistenz.* München.

Standard, I. (2011). Road vehicles — Functional safety.

Sweeden, R. D. (2013). *Master Thesis: Building a Safety Case in Compliance with ISO 26262 for Fuel Level Estimation and Display System.*

SysML. (n.d.). Retrieved from http://sysml.org/

The ESSarel research project & tool. (n.d.). Retrieved from http://www.essarel.de/

UML. (n.d.). Retrieved from http://www.uml.org/

List of abbreviations

Abbreviation	Definition
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
CSD	UML Composite Structure Diagram
CFT	Component Fault Tree
DC	Diagnostic Coverage
EAST-ADL	Electronics Architecture and Software Technology - Architecture Description Language
E/E	Electrical or/and Electronics
ECU	Electronic Control Unit
FCM	Federation Conceptual Model
FIT	Failures In Time
FMEA	Failure Mode and Effects Analysis
FMEDA	Failure Modes, Effects and Diagnostic Analysis
FPGA	Field-Programmable Gate Array
FTA	Fault Tree Analysis
HARA	Hazard Analysis and Risk Assessment
HIL	Hardware-in-the-Loop
HW	Hardware
LFM	Latent-fault metric
MARTE	Modelling and Analysis of Real-Time and Embedded systems
MIL	Model-In-the-Loop
PMHF	Probabilistic Metric for random Hardware Failures
SIL	Software–In-the-Loop
SPFM	Single-point fault metric
SW	Software
SysML	System Modelling Language
TADL	Timing Augmented Description Language
UML	Unified Modelling Language
V&V	Verification & Validation

XMI	XML Metadata Interchange
XML	eXtensible Markup Language