Distributed Real-time Architecture for Mixed Criticality Systems

Description of Development Process with Model Transformations
D 1.3.1

Project Acronym	DREAMS	**Grant Agreement Number**		FP7-ICT-2013.3.4-610640	
Document Version	1.0	**Date**	2014-07-31	**Deliverable No.**	1.3.1
Contact Person	Leire Rubio	**Organisation**		IK4-IKERLAN	
Phone	+34-943712400	**E-Mail**		lrubio@ikerlan.es	

Contributors

Name	Partner
Leire Rubio, Asier Larrucea	IKL
Gebhard Bouwer, Gernot Klaes	TÜV
Simon Barner, Alexander Diewald	FORTISS
José Enrique Simo Ten, Alfons Crespo	UPV/FENTISS
Thomas Koller, Obaid Ur-Rehman	USIEGEN
Jörn Migge	RTaW
Arjan Geven	TTT
Anton-Aart Trapman	ALSTOM

Table of Contents

Contributors 2
Table of Contents 3
Figure Index 5
Table Index 6
Glossary 7
1 Introduction 9
1.1 Context 9
1.2 Objectives of the document 9
1.3 Structure of the document 9
1.4 IEC 61508 Safety Life Cycle Overview 9
2 Definition of DREAMS Development Process 13
2.1 Safety Approach 13
2.1.1 Impact of DREAMS characteristics on the development process 14
2.1.2 DREAMS Safety Life-Cycle Specification 22
2.1.3 Measures for fault avoidance within DREAMS 22
2.2 Security Approach 24
2.2.1 Impact of security aspects on the DREAMS development process 24
2.2.2 DREAMS Security Lifecycle Specification 28
2.2.3 Measures for fault avoidance within DREAMS 32
2.3 Timing Approach 32
2.3.1 GMP 32
2.3.2 Impact of DREAMS characteristics on the development process 35
2.3.3 DREAMS Timing Lifecycle Specification 37
2.4 Summary of DREAMS Development Process 39
3 DREAMS Meta-Models and Model-Transformations 41
3.1 Impact of DREAMS Characteristics 41
3.2 Introduction 43
3.2.1 (Meta-)Modelling 43
3.2.2 MDE tool-chain 45
3.2.3 DREAMS System Model 47
3.3 AutoFocus3 Meta-Model 49
3.3.1 Requirements perspective 50
3.3.2 Logical Perspective 51
3.3.3 Technical Perspective 54
3.3.4 Deployment Perspective 56
3.4 Timing requirements Meta-Model 56
3.5 MultiPARTES Meta-Model 61

3.5.1 Overview: Meta-Models, Toolset and Transformations 61
3.5.2 Safety Consistency Model 64
3.5.3 Safety Compliance Model 64
3.5.4 Diagnostics Techniques and Measure Model 65
3.5.5 Safety Integrity Levels Model 66
3.6 Model-to-model Transformations 66
3.6.1 Generic DREAMS Development Process 66
3.6.2 Exemplary Development Process using Concrete Technologies 73
4 Requirements Matrix 79
5 Bibliography 83
Terminology 84

Figure Index

Figure 1: Overall Safety Life Cycle 10
Figure 2: E/E/PE Safety Life Cycle. 11
Figure 3: Software Safety Life Cycle. 11
Figure 4: Software Development Process (V-model). 12
Figure 5: Hardware Development Process (V-model). 12
Figure 6: ASIC development safety life cycle (V-model). 13
Figure 7: System development safety lifecycle (the V-model) 14
Figure 8: Meet-in-the-middle design flow of DREAMS. 17
Figure 9: Modular design of DREAMS. 19
Figure 10: DREAMS System and Element Development Processes. 20
Figure 11: Component based DREAMS Development process. 20
Figure 12: Domain Engineering and Product Engineering. 21
Figure 13: Safety-related Development Process of DREAMS. 23
Figure 14: Security requirements in the meet-in-the-middle design flow of DREAMS 26
Figure 15: Security Requirements and Threat Model for DREAMS 26
Figure 16: Modular design of DREAMS. 27
Figure 17: Modular design example with untrusted components 28
Figure 18: Trustworthy Security Development Lifecycle (SDL) proposed by Microsoft. 29
Figure 19: Security Development Lifecycle based on ISO/IEC 21827 30
Figure 20: Security Development Process of DREAMS 31
Figure 21: GMP for Timing 33
Figure 22: Abstracting Timing Properties (TIMMO-2-USE) 36
Figure 23: Timing-related Development Process. 38
Figure 24: DREAMS Development Process. 40
Figure 25: Comparison of object technology and model-driven engineering (MDE) 44
Figure 26: 4-level model architecture 45
Figure 27: General architecture of MDE tool-chain. 46
Figure 28: System Structure of Application (Logical View) and Structure of Platform (Physical View) [D1.2.1] 47
Figure 29: Dimensions of abstraction: granularity levels, and development perspectives 49
Figure 30: AutoFOCUS3 Requirements Perspective 51
Figure 31: AutoFOCUS3 Logical Perspective 52
Figure 32: Meta-Model of Logical Perspective 52
Figure 33: AutoFOCUS3 Technical Perspective and Deployment Perspective 54
Figure 34: Meta-Model of Technical Perspective 56
Figure 35: Timing Extension 57
Figure 36: Timing Constrains 58
Figure 37: Modular synchronization constraints 59
Figure 38: DREAMS Timing Extensions 60
Figure 39: DREAMS Timing Events at software component ports 60
Figure 40: Safety Consistency Check, Useful Documents for Certification and Generation of Restrictions. 63
Figure 41: Safety Consistency Model 64
Figure 42: Safety Compliant Item and Safety Manual Models 65
Figure 43: Diagnostic Techniques and Measure Model. 65
Figure 44: Instance of the Diagnostic Techniques and Measure Model for IEC-61508. 66
Figure 45: Overview of DREAMS software development & deployment workflow 67
Figure 46: Checking process and diagnostic techniques information panel. 69
Figure 47: Example of Wind Power certification document. 70
Figure 48: Overview of Offline Resource Allocation and Exploration Process 71
Figure 49: DREAMS software development workflow: Tools, transformations and implementation artefacts. 74
Figure 50: Context of XtratuM configuration model 76
Figure 51: Parts of XtratuM configuration model. 77
Figure 52: TTE tool-chain workflow 78
Figure 53: TTE tool-chain dataflow 78

Table Index

TABLE 1: ANALYSIS OF DREAMS REQUIREMENTS. 15
TABLE 2: TRACEABILITY OF DREAMS DEVELOPMENT PROCESS. 18
TABLE 3: IMPACT OF SECURITY ASPECTS INTO THE DREAMS DEVELOPMENT PROCESS. 25
TABLE 4: ANALYSIS OF DREAMS TIMING REQUIREMENTS. 35
TABLE 5: IMPACT OF DREAMS CHARACTERISTICS ONTO META-MODELS 42
TABLE 6: IMPACT OF DREAMS CHARACTERISTICS ONTO MODEL-TRANSFORMATIONS 43
TABLE 7: DREAMS REQUIREMENT MATRIX. 82

Glossary

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
ASIL	Automotive Safety Integrity Level
ATL	Atlas Transformation Language
CMOF	Complete MOF
CPLD	Complex Programmable Logic Device
DREAMS	Distributed Real-Time Architecture for Mixed Criticality Systems
E /E /PE	Electrical/Electronic/Programmable Electronic
EAST-ADL	EAST Architectural Description Language
ECU	Electronic Control Unit
EMF	Eclipse Modelling Framework
EMOF	Essential Meta-Object Facility
FPGA	Field-Programmable Gate Array
GMP	Generic Methodology Pattern
ID	Identification
IEC	International Electrotechnical Commission
JAXB	Java Architecture for XML Binding
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MIRA	Model-based Requirements Analysis
MOF	Meta Model Facility
NI	Network Interface
NoC	Network on Chip
OMG	Object Management Group
PIM	Platform Independent Model
PLD	Programmable Logic Device
PSM	Platform Specific Model
QVT	Query/View/Transformation
SDK	Software Development Kit
SIL	Safety Integrity Level
STNoC	System on Chip
TADL	Timing-Augmented Description Language
TIMMO	TIMming MOdel
TIMMO-2-USE	Timing Model – TOols, algorithms, languages, methodology, USE cases.
TTE	Time Triggered Ethernet
VL	Virtual Link
VLID	Virtual Link ID
WCET	Worst Case Execution Time
WCTT	Worst Case Traversal Time
XCM	Extended Configuration Management

XML	Extensible Markup Language
XSD	XML Schema

1 Introduction

1.1 Context

The objective of DREAMS is to develop a cross-domain architecture and design tools for networked complex systems where application subsystems of different criticality, executing on networked multi-core chips, are supported. DREAMS will deliver architectural concepts, meta-models, virtualization technologies, model-driven development methods, tools, adaptation strategies and validation, verification and certification methods for the seamless integration of mixed-criticality to establish security, safety, real-time performance as well as data, energy and system integrity.

Engineering of safety systems typically implies enforcing a strict development process. V-shape processes have been used frequently to attain certification.

1.2 Objectives of the document

This delivery will define a DREAMS development process that is realized on the top of DREAMS platform with the building blocks from WP2-WP5 and which will be applied in the way that will be possible, on the DREAMS three demonstrators (WP6-WP8). The overall development process, which is based on IEC 61508, will be complemented by abstracting the different aspects that must be covered by the models, the models used in individual domains, and the necessary tool support for model-to-model transformation, verification and code generation.

1.3 Structure of the document

This document contains the following structure. Section 2 describes the approaches to safety (IEC 61508), security and timing for development of DREAMS project based development process. Section 3 describes the required meta-models and model transformations for a tool-supported workflow. Finally Section 4 provides a relation matrix of requirements and sections related with these requirements.

1.4 IEC 61508 Safety Life Cycle Overview

A safety life cycle is a series of phases from initiation or specification of safety requirements, to cover and develop of safety features in safety-critical system, and ending in decommissioning of that system. The IEC 61508 standard covers safety-related systems where a system incorporates electrical/electronic/programmable electronic devices. The standard covers possible hazards caused by failures of the safety functions of E/E/PE safety related systems. The detection of a potentially dangerous condition that results in the action of a protective or corrective mechanism to prevent hazardous events is defined as *functional safety*. IEC 61508 is concerned with the E/E/PE safety-related systems whose failure could have an impact on the safety of persons and/or environment.

The standard has two fundamental points: the safety life cycle and the safety integrity levels. The safety life cycle is defined as a process that includes all necessary steps to achieve the required functional safety.

Figure 1 show the safety life cycle defined by IEC 61508 [1], which is basis, followed and complained by the proposed DREAMS development process in Section 2.

Phases 1 and 2 entail the considerations of the safety implications of the EUC and the control systems, at the system level. In Phase 3, first two phases' risk identification and analysis, assessed

against tolerable criteria, are done. In phase 4, the risk-reduction measures of safety requirements are specified, and in phase 5 these are translated into the design of safety functions, which are implemented in safety-related systems, depending on the selected manner of implementation in phases 9, 10 and 11. No claim for safety can be made unless its planning considers the overall safety context reflected in phases 6, 7 and 8. The further phases, phases 12-16 are performed when the system has been built. In Phase 12, the system must be installed and commissioned and in Phase 13, the system is checked to verify that all the safety related requirements have been identified and handled during building and installation. Then the system may be put into operation, where there are safety and maintenance activities. It is also foreseen that the system can be modified during operations and therefore, incorporation of some modifications will be needed. The final phase, phase 16, is related with the disposal of the system (e.g., separations of the battery or the toxic elements to dispose them separately).

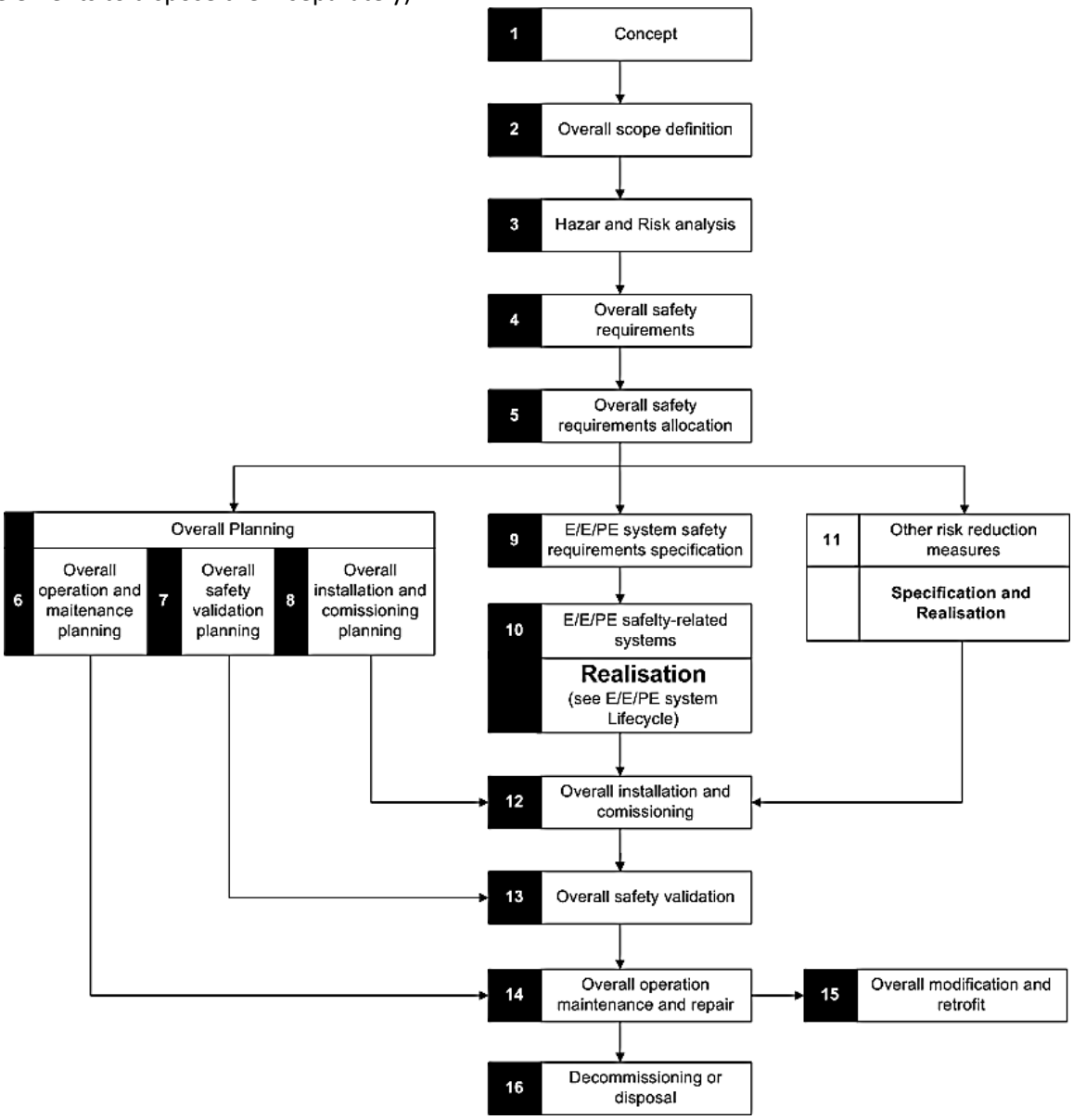


Figure 1: Overall Safety Life Cycle.

IEC 61508 defines detailed lifecycle stages for stage 10 in the overall life cycle concerned to E/E/PE and software systems developments. The E/E/PE system and software safety life cycles that shall be defined and used according to IEC 61508 are specified in Figure 2 and Figure 3.

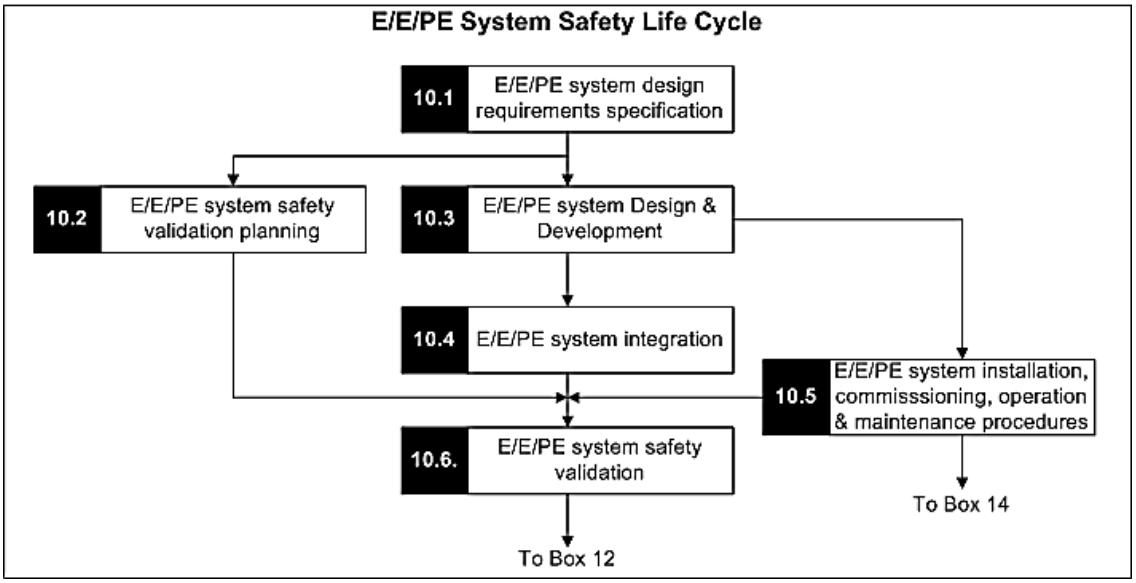


Figure 2: E/E/PE Safety Life Cycle.

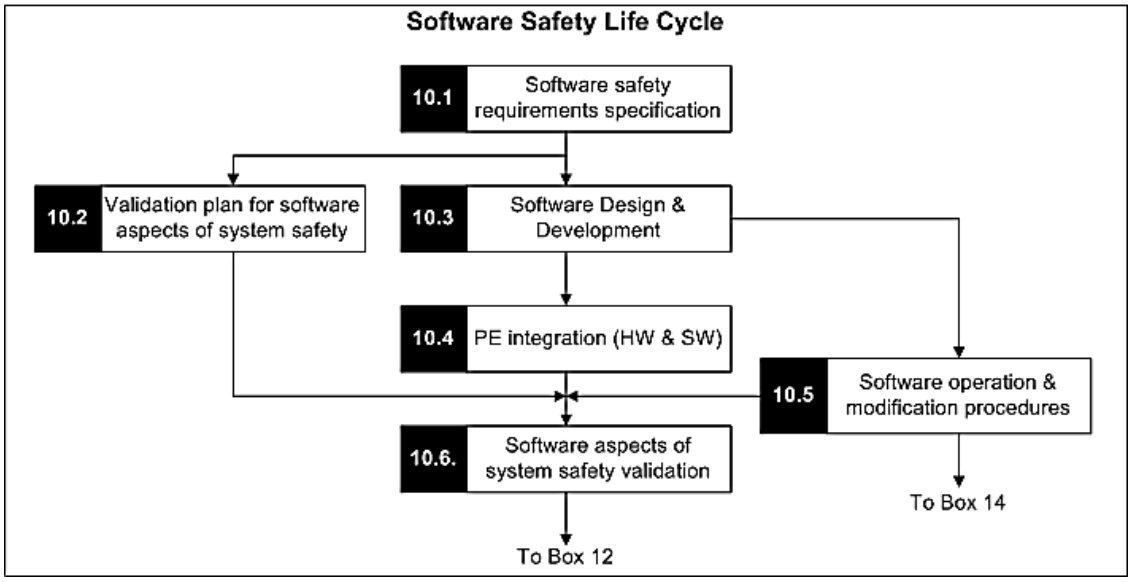


Figure 3: Software Safety Life Cycle.

Figure 4 shows the V-model of the development life cycle for the design of ASICs. For IEC 61508, the term ASIC covers standard integrated circuits, core-based and cell-based ASICs, gate arrays, FPGAs, PLDs and CPLDs (IEC 61508-4 3.2.15). As shown, there are similarities between ASIC and software safety life-cycles.

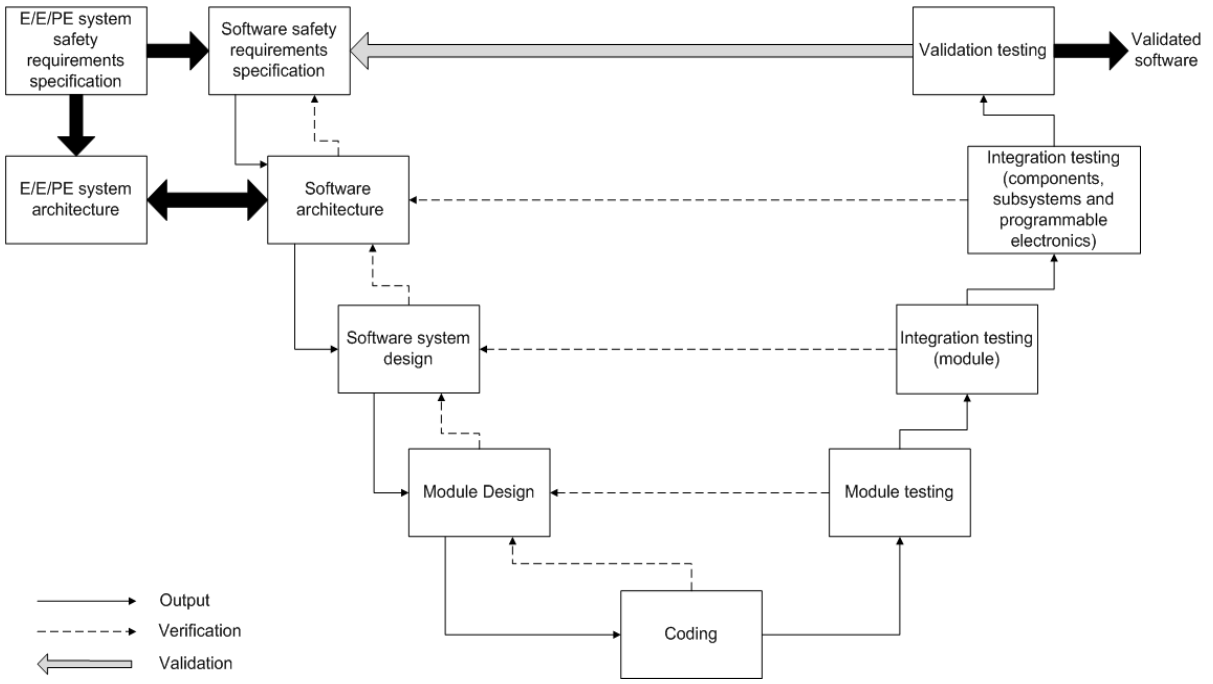


Figure 4: Software Development Process (V-model).

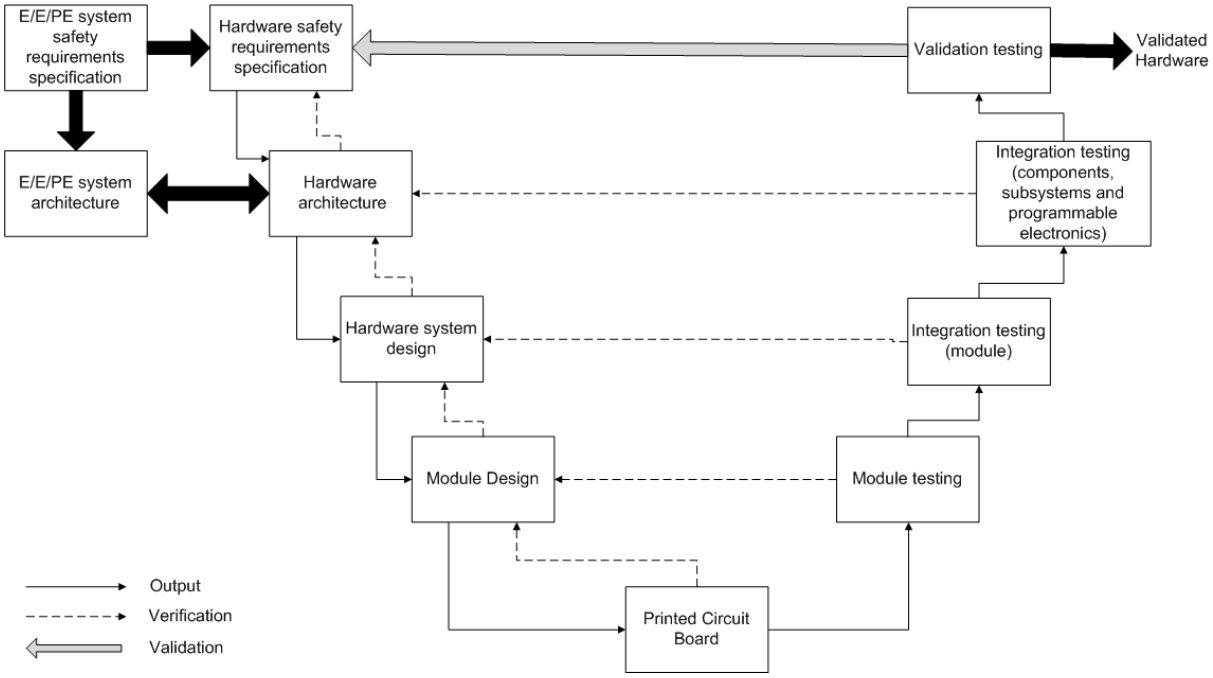


Figure 5: Hardware Development Process (V-model).

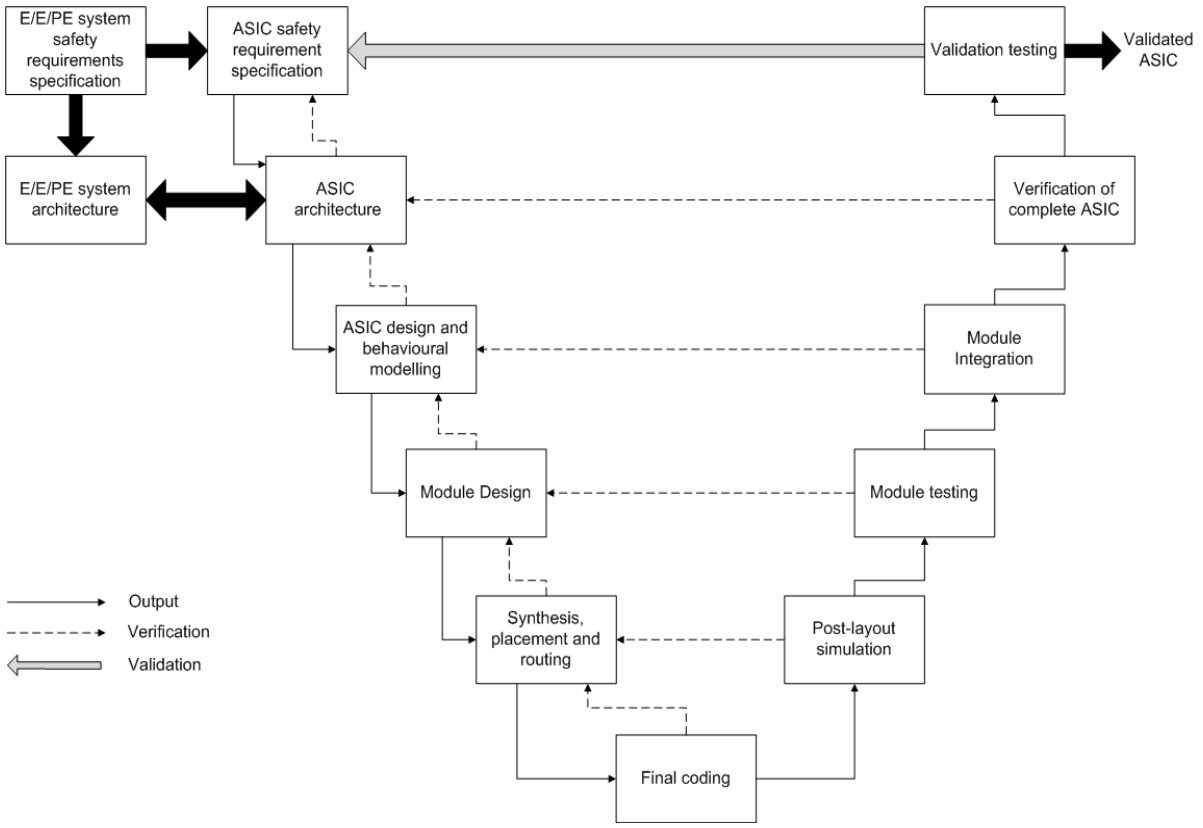


Figure 6: ASIC development safety life cycle (V-model).

If another safety life-cycle is used, it shall be specified as part of the management of functional safety activities (clause 6 of IEC 61508-1) and all objectives and requirements of all sub-clause of IEC 61508-2 shall be met.

2 Definition of DREAMS Development Process

One of the main objectives of the document is to expose the DREAMS development process, which although is not intended to develop hardware in its specific context, is intended to develop software. The main goal is not hardware development, although according to IEC 61508-2, sometimes (e.g., implementation of communication network inside a FPGA (e.g., STNoC)) is considered as HW, so, the development process of DREAMS shall be compatible for hardware and software development.

In this section, overall results of the analysis of the DREAMS project requirements are detailed. Then the impact of the DREAMS approach on the V-model is presented. And finally, the DREAMS project-specific development process is detailed and explained.

2.1 Safety Approach

Commonly, V-shape processes have been used to attain certification. IEC 61508 recommends the following V model development process. Figure 7 shows a lifecycle that is composed by two branches: I) a design branch and II) a testing branch. This V-model can be easily customized for any domain (e.g., automotive, railway, etc.).

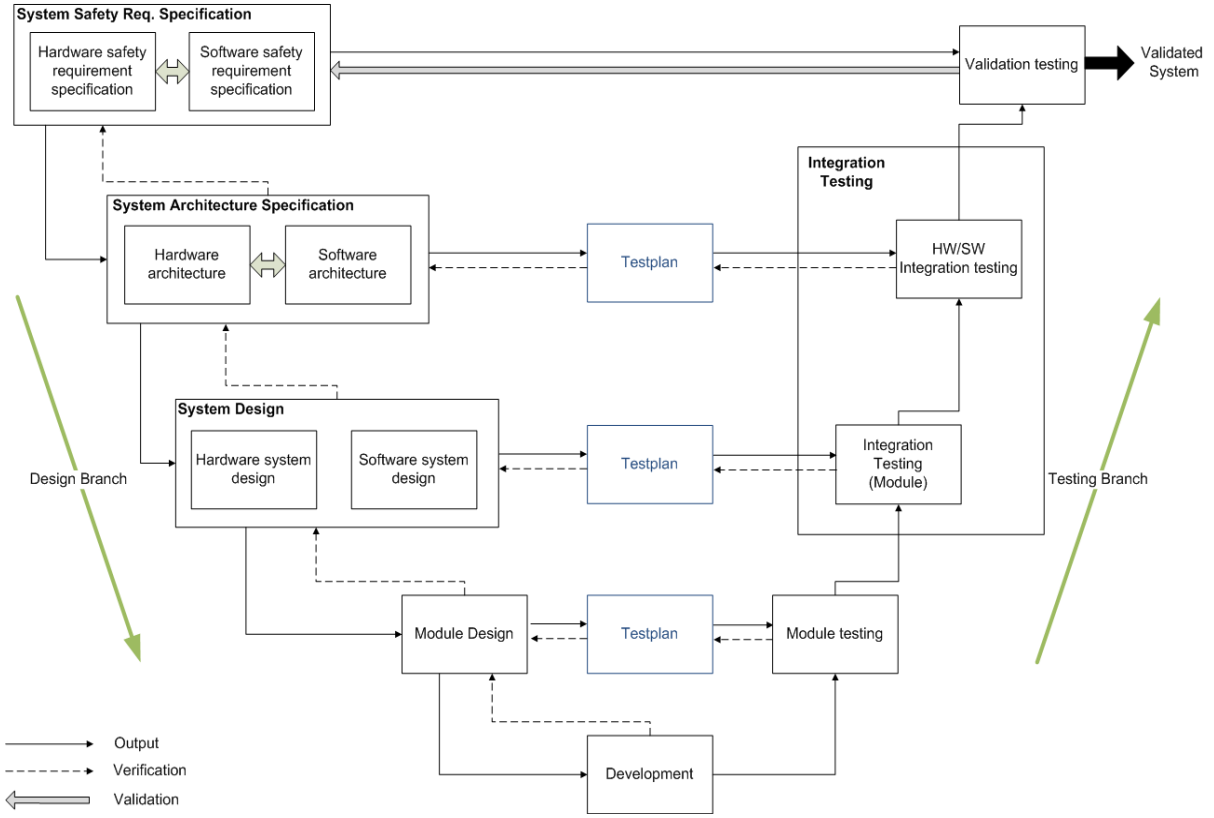


Figure 7: System development safety lifecycle (the V-model)

The continuous arrows of Figure 7 show the dependencies between phases. At the end of each phase the verification is carried out (discontinuous vertical arrows) against the results of each phase. This is done in order to check the consistency between the inputs and outputs of each phase. As can also be seen, the branches of the Design and Testing are linked by the Test Plans. The solid arrows on the left side of each Test Plan indicate that each test plan, which covers all the requirements, has been generated from the specification of the requirements. Regarding the solid arrows on the right side, these arrows denote the actions to perform the testing. The dashed lines to/from test plans are the verification activities.

In case of hardware development, the techniques and measures defined by IEC 61508-2 must be applied. In case of software development, the techniques and measures defined by IEC 61508-3 must be taken into account.

2.1.1 Impact of DREAMS characteristics on the development process

In delivery D1.1.1 the DREAMS project requirements have been collected. The DREAMS project has some requirements that have to be taken into account for the definition of the development process. These relevant requirements are referred to the meet-in-the-middle development approach, traceability and modularity concepts.

Req. ID	Description	Reference to
R1.8.1	The architecture should allow design methodologies where top-down and bottom-up design styles are combined.	Meet-in-the-middle development
R 9.12.1	The development process should support "top-down" and "bottom-up" development of DREAMS-based applications (“meet-in-the-middle methodology”), if possible aligned with existing practices and workflows .	

R1.1.1	The architecture shall assure that the behaviour of a subsystem in the value and time domain before integration into a larger systems equals the behaviour after integration.	Scalability
R1.6.1	The architecture shall leverage multi-core platforms for a system perspective of mixed-criticality applications combining the chip-level and cluster-level.	
R 9.9.4	DREAMS systems need to be automatically adaptive, and this requirement will help automating the production of a configuration in particular in adapting to different platform technologies (e.g. hardware).	
R9.6.3	The meta-models should support the traceability between the artefacts used in the different steps of the development process.	Traceability
R9.13.3	The development process shall support the traceability for requirements regarding: safety, security, etc. Traceability will help in the avoidance of systematic faults in both HW (IEC 61508-2 Table B.6) and SW (IEC 61508-3 Tables A.1 to A.10) development.	
R1.10.1	The architecture shall support different models of computation with corresponding interaction mechanisms on on-chip and off-chip networks: predictable time-triggered communication, event-triggered communication with dynamic arbitration and shared memories.	Heterogeneity
R1.10.2	Communication between components in an application subsystem should be performed explicitly using communication primitives provided by the architecture.	
R1.10.3	The architectural style and the development methodology shall consider multiple types of communication and computational activities (periodic, sporadic and aperiodic activities).	
R2.7.1	The on-chip network shall provide different interaction mechanisms required for different models of computation such as Time-Triggered, rate-constrained, best-effort communication and shared memory access.	
R 2.7.2	The architecture shall provide support of different processors and/or hardware accelerators with shared memory access.	
R4.2.1	Variability modelling and analysis tools shall be enhanced to achieve by automatic means as well as guided manual means an optimal or best effort configuration of DREAMS platforms and DREAMS systems.	Variability
R5.1.1	Mixed-criticality product line shall be supported to enable certification of product-lines with variability management.	
R9.9.1	The variability meta-model shall allow specifying variations of base models in order to define product lines.	
R9.9.2	The variability meta-model shall allow to describe different feature sets of applications.	
R9.9.3	The variability meta-model shall allow to describe different implementation alternatives of applications.	
R9.14.1	The development process shall define how variability is bound.	

Table 1: Analysis of DREAMS requirements.

Below each characteristic of the DREAMS architecture is defined and the respective changes (if necessary) are specified in the development process according to IEC 61508.

2.1.1.1 Meet-in-the-middle Methodology

The meet-in-the-middle approach can be considered as a successive refinement method going alternatively from a top-down methodology to a bottom-up methodology, in order to converge into a hardware or software solution. It is this characteristic that makes platform-based design a novel design method, which can be used for the design of complex and heterogeneous embedded systems. The meet-in-the-middle process approach is not a top-down process in which the software is designed in the first place and hardware is developed secondly [2]. It is used when pre-existing products, for which there are some implementations, are partially mapped onto a new service, functionality or process definition. This option involves the usage of old and new services, functionalities or process definitions. The meet-in-the-middle development strategy offers a middle ground between previous methodologies, that attempts to take advantages of other approaches, while attenuating some of their most notable risks and problems [3]. For attenuation of risks and problems, from the point of view of safety, it is important to show that the existing products are sufficiently free of systematic faults. According to IEC 61508-2 and IEC 61508-3, there are two routes which can be used to probe the non existence of systematic faults.

- **Route 2_S[1] - Proven-in-use approach:** Compliance with the requirement for proven in use elements is established. An element shall only be regarded as proven-in-use when it has a clearly restricted and specified functionality and when there is demonstration that the systematic faults are low enough. (IEC 61508-2 Section 7.4.10)

- **Route 3_S - Pre-existing software:** This option is the compliance with the requirements of IEC 61508-3, where it is defined that the pre-existing software elements that are reused to implement all or part of the safety functions, shall meet the following requirements:
  - Meet the requirements of one of the following compliance routes
    - Route 1_S: compliant development. Compliant with the requirements of the IEC-61508 standard for the avoidance and control of systematic faults in software.
    - Route 2_S: proven in use. Provide evidence that the element is proven in use.
    - Route 3_S: assessment of non-compliant development. Compliance with IEC-61508-3 7.4.2.13.
  - Provide a safety manual (Annex D of IEC 61508-2 and IEC 61508-3) that gives a sufficiently precise and complete description of the pre-existent elements to make possible an assessment of the integrity of a specific safety function that depends wholly or partly on the pre-existing software elements.

In general, a meet-in-the-middle methodology applies a top-down design (application design) from higher abstraction level and a bottom-up to lower level (platform design). These two processes necessarily meet at some point, when the platform is ready to host an application, and when the application is ready to be hosted in a platform. This point is called meet-in-the-middle point, where performance analysis and architectural exploration takes place.

[1] $_S$: Designates systematic safety integrity to distinguish it from Route 1_H for hardware safety integrity.

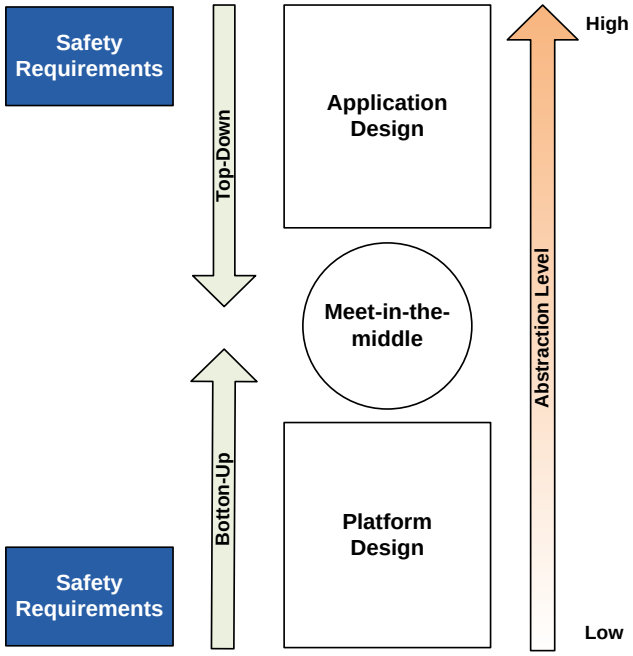


Figure 8: Meet-in-the-middle design flow of DREAMS.

Figure 8 illustrates the meet-in-the-middle design flow approach for DREAMS, where, a top-down methodology is applied for higher abstraction levels and a bottom-up methodology is applied to lower abstraction levels. At the same time, at the hardware perspective it supports the bottom-up approach (low to high abstraction level), this way enabling hardware adaptation at design-time as well as at runtime using dynamic and partial reconfiguration.

2.1.1.2 Traceability

In order to ensure that the software that results from the life cycle activities meets safety-related requirements, it is essential to ensure the consistency between the life cycle stages. Traceability is the impact analysis to check that the decisions made at an earlier stage are adequately implemented in later stages (forward traceability) and that decisions made at later stage are actually required and mandated by earlier decisions.

According to IEC 61508, the following traceability techniques must be used during the development process of DREAMS for assurance of traceability.

- IEC 61508-2 [4] [4]specifies hardware traceability, which should be between specifications, design, circuit diagram and parts lists. It should be computer-aided and based on defined methods (IEC 61508, table B.6 and section 7.2.2.2 of IEC 61508-2).
- In case of software, traceability is done between all phases of the development process, in compliance also with IEC 61508-3. The following table shows the traceability techniques specified by IEC 61508-3 that are interpreted and is going to be used among phases of DREAMS Development process.

A prerequisite for traceability is, that the requirements can be clearly identified. For this, the requirements usually get a unique identifier. Subsequently, for example, in a tables, the relation between the identifier is shown tracked through the development process (with derived requirements and respective test cases). These mentioned tables have to be reviewed (amongst others) to ensure that all requirements have been considered in the respective development document in the respective development phase.

Technique	Ref.	Table	Interpretation
Forward traceability between the system safety requirements and the software safety requirements.	C 2.11	A.1	Review to ensure that all system safety requirements are addressed by software safety requirements.
Backward traceability between the safety requirements and the perceived safety needs.	C 2.11	A.1	Review to ensure that all software safety requirements are actually needed to address systems safety requirements.
Forward traceability between the software safety requirements specification and software architecture.	C 2.11	A.2	Review to ensure that all software safety requirements are addressed by the software architecture.
Backward traceability between software safety requirements specification and software architecture.	C 2.11	A.2	Review to ensure that all architecture safety requirements are actually needed to address software safety requirements.
Forward traceability between the software safety requirements specification and software design.	C 2.11	A.4	Review to ensure that all software safety requirements are addresses by the software design.
Forward traceability between the software design specification and the module and integration tests specifications.	C 2.11	A.5	Review to ensure that a adequate test is planned to examine the functionality of all modules and their integration with appropriately related modules.
Forward traceability between the system and software design requirements for hardware/software integration and the hardware/software integration test specifications	C 2.11	A.6	Review to ensure that the hardware/software integration tests are adequate/ sufficient.
Forward traceability between the software safety requirements specification and the software safety validation plan.	C 2.11	A.7	Review to ensure that adequate software validation tests are planned to address the software safety requirements.
Backward traceability between the software safety validation plan and the software safety requirements specification.	C 2.11	A.7	Review to ensure that all validation tests are relevant.
Forward traceability between the software safety requirements specification and the software modification plan (including reverification and revalidation)	C 2.11	A.8	Adequate modification procedures to achieve the software safety requirements.
Backward traceability between the software modification plan (including reverification and revalidation) and the software safety requirements specification	C 2.11	A.8	Adequate modification procedures to achieve the software safety requirements.
Forward traceability between the software design specification and the software verification (including data verification) plan.	C 2.11	A.9	Review to ensure adequate test of functionality.
Backward traceability between the software verification (including data verification) plan and the software design specification.	C 2.11	A.9	Review to ensure that all verification tests are relevant.
Forward traceability between the requirements of IEC 61508-1 Clause 8 and the plan for software functional safety assessment	C 2.11	A.10	Check completeness of coverage of the functional safety assessment.

Table 2: Traceability of DREAMS Development Process.

2.1.1.3 Modularity

Modular design or modularity in design is an approach that subdivides a system into smaller parts (modules) that can be independently created and then used by different systems to drive different functionalities; a compliant item. The following figure shows the DREAMS modular structure that is decomposed into four levels (System-of-Systems, systems, subsystems and components). A system of system is a composition of independent and interoperable systems intended to achieve unique goals collectively.

For example, considering a series of DREAMS platforms (see D1.2.1, figure 49) as a System of system, the cluster domain can be considered as a system, the Node Domain can be considered as a Sub-System and the Virtualization Layer Domain as an element of the sub-system. In the same way (D1.2.1, figure 50), if we focus on the Node domain, we can consider it as SoS, and its internal elements can be abstracted as systems (e.g., I/O, gateways, applications, etc.), subsystems (e.g., Processor Cores, Local Memory, Network Interface, etc.) and elements (e.g., MON, LRS, Application components).

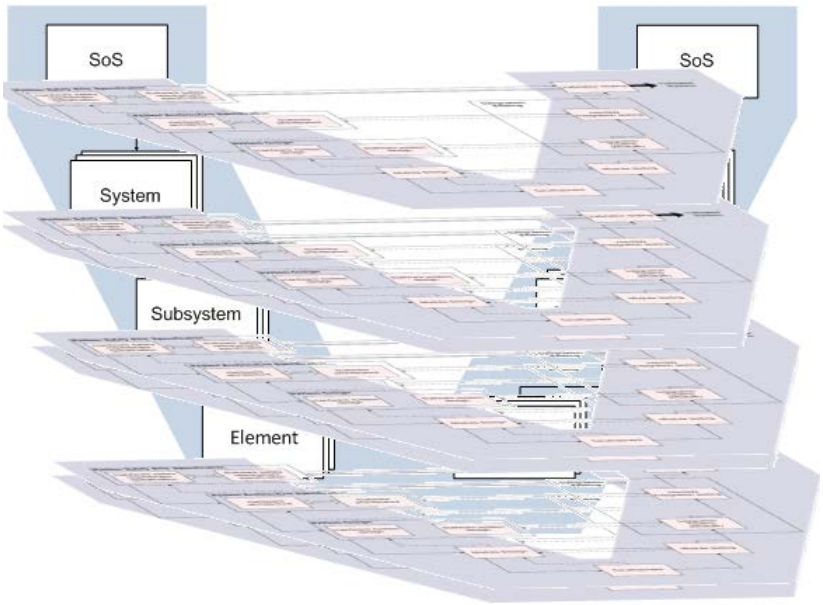


Figure 9: Modular design of DREAMS.

To apply the modularity approach in the DREAMS development process, IEC 61508 defines some rules, techniques and measures (IEC 61508-2 Annex A-B) that must be followed and used in order to comply with the standard. In case of hardware, IEC 61508-2 defines that modules must be limited by size and must also be isolated. In case of software modification, an impact analysis is carried out to determine how the effect of modifications is limited by the modularity of the overall system.

Modularity aims at the decomposition of the software system into small comprehensible parts (element) in order to limit the complexity of the system. It addresses typical development process methods, although there is one difference; a element-based development process focuses on questions related to elements whereas a system development process does not. In that sense, we distinguish the element HW/SW (e.g., dedicated tile/core, etc.) development process and system elements development for the extension of the platform and application elements. The activities or goals of these two processes are different. In case of the system development, the emphasis is to find the proper element and to verify them. In the other hand, in case of the element development, is required the independence and the reusability of the elements.

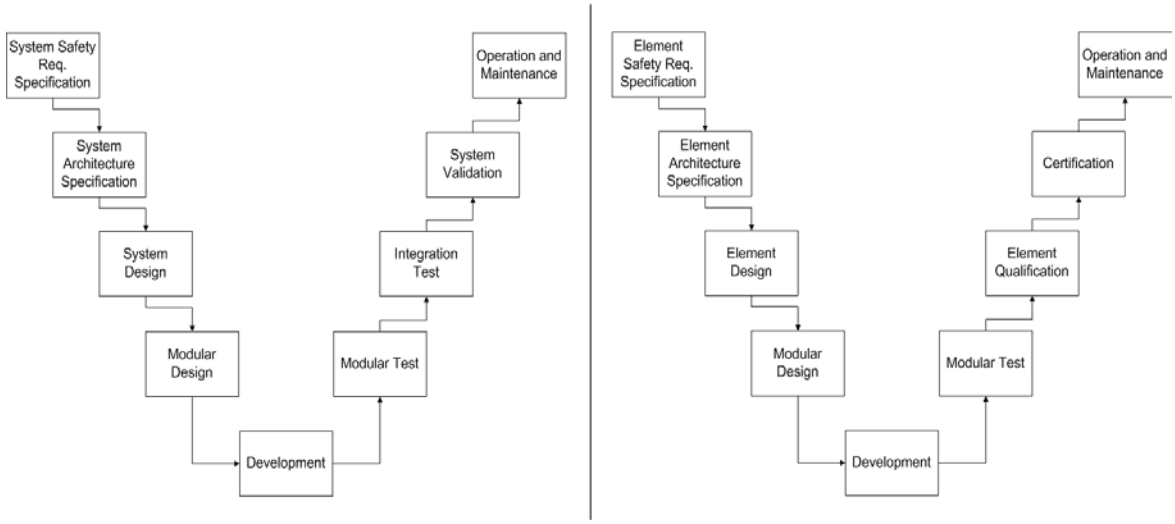


Figure 10: DREAMS System and Element Development Processes.

Figure 11 shows the DREAMS component-based development process, which assumes that there are closed components (developed elements), that can be integrated into the system through an adaptation process. This way, a element can be reused into diverse systems development at the same time, thereby reducing the required development time and cost.

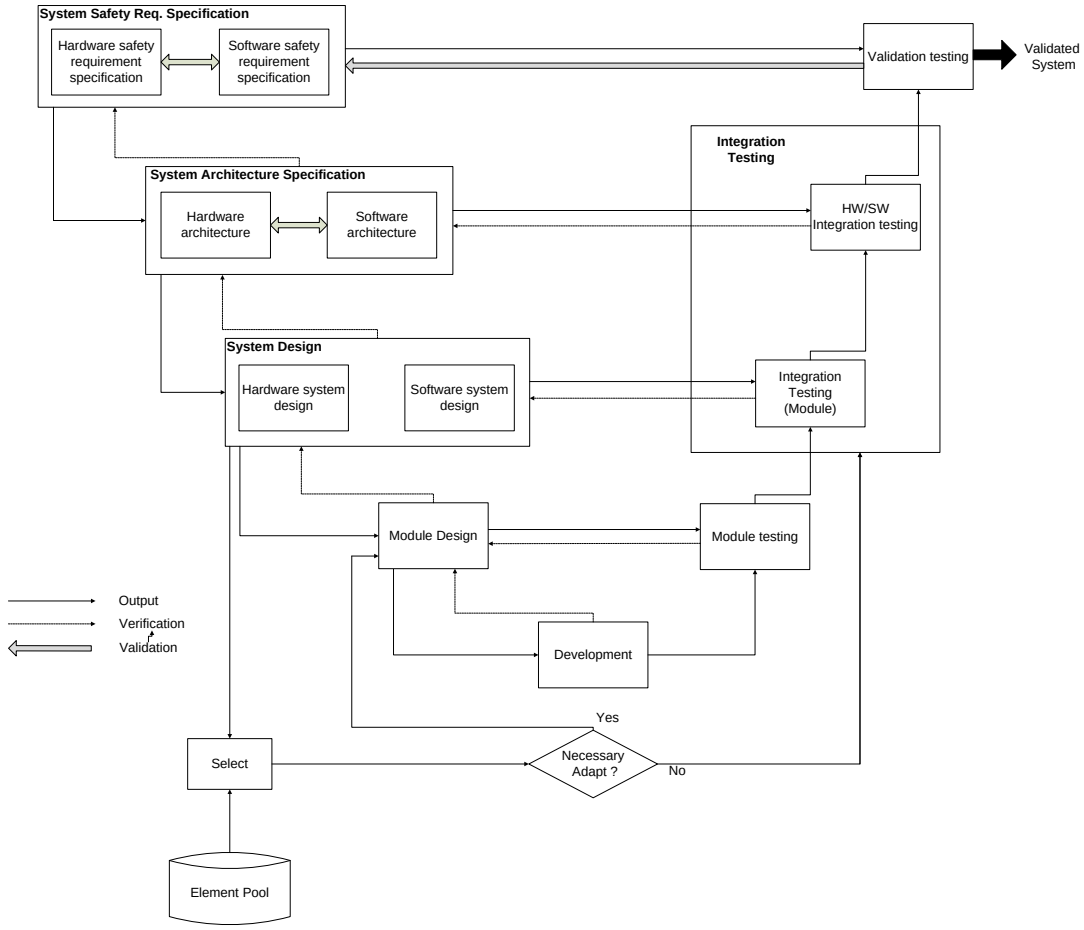


Figure 11: Component based DREAMS Development process.

Whereas modularity is useful for the decomposition of a system into smaller parts in order to develop independent elements, some features of these elements can be variable, depending of the needs of system/subsystem. Therefore, DREAMS development process also takes into account the variability of both HW and SW. Hardware variability is the ability to change (customize, extend) a hardware platform for a specific context. Software variability is the ability to change (configure,

customize, extend) software artefacts (e.g., code, product, domain requirements, models, design, documentation, test cases) for a specific context. The reusability of any software artefact is determined by its ability to support the variability required from it.

One of the purposes of DREAMS is to offer a core system, whose can be updated and upgraded with specific features, but always with the same nucleus. The conjunction of DREAMS characteristics provide a diverse, independent and re-usable element based architecture, which can be used in future changes or updates (change or update of functionality, platform, etc) of the product line.

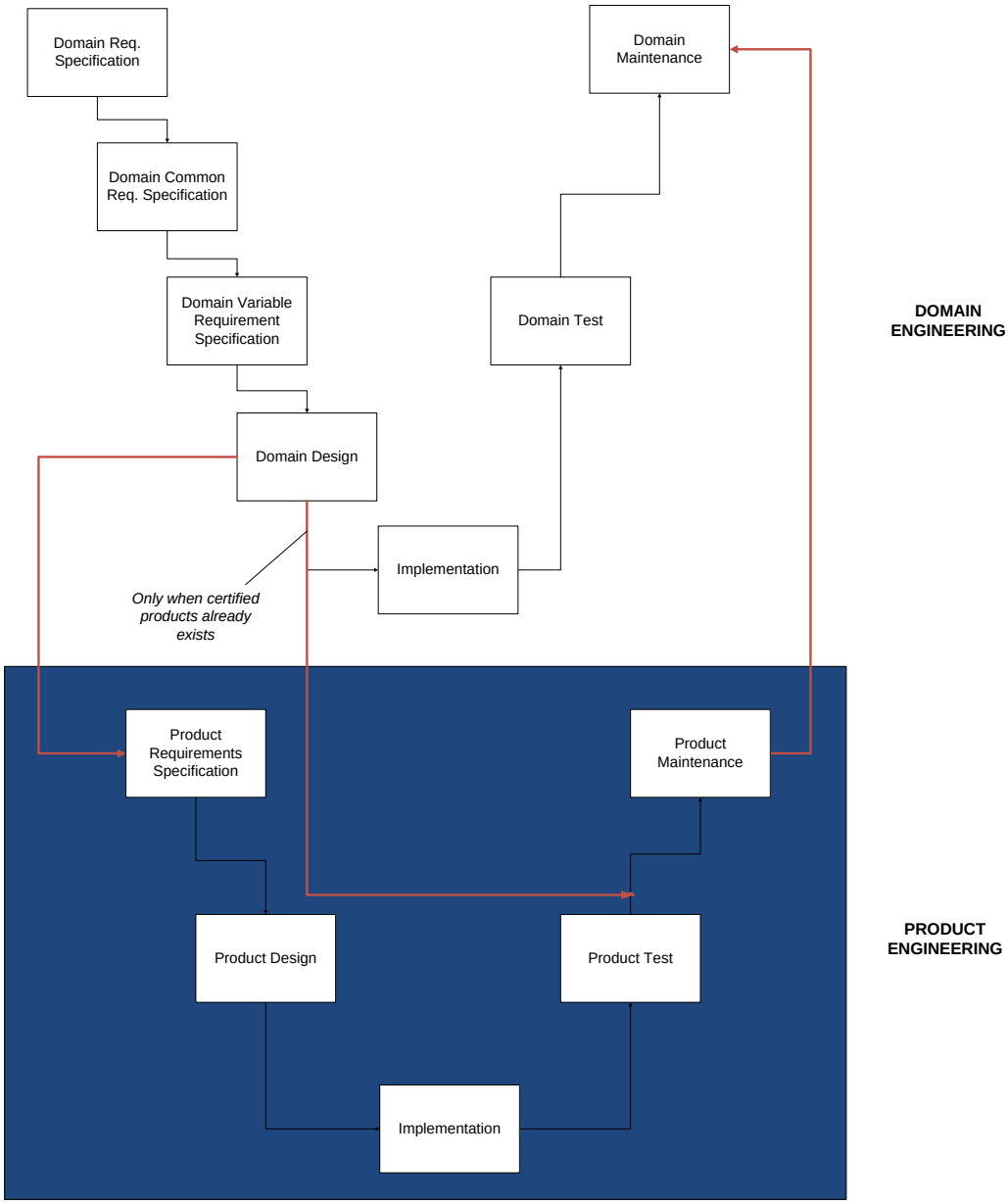


Figure 12: Domain Engineering and Product Engineering.

The process of reusing product-line knowledge in the production of new systems, also called domain engineering, is designed to improve the quality or characteristics of the development through the reuse of artefacts. Domain engineering is applied to all phases of the development process, although it is focused on the three primary phases: Domain analysis, design and implementation paralleling application engineering, which produces a set of implemented components that are relevant to the domain, reusable and configurable. Domain engineering focuses on a family of systems (products).

Figure 12 shows that in domain design, where all domain requirements are available and can be used to define the product specifications. After the product is tested, it can be used for domain implementation together with many other products. The domain implementation must not have direct effect on the product implementation but it has to go back to the domain design and product specification.

2.1.2 DREAMS Safety Life-Cycle Specification

According to points raised in the previous section, this section is intended to summarize the most important aspects of the DREAMS development process related to the safety approach, as shown in Figure 13.

2.1.3 Measures for fault avoidance within DREAMS

For each level of Figure 13 (Domain Level, System Level and Element Level) techniques and measures for the avoidance of systematic failures during the different phases of the lifecycle have to be defined. The basis for these techniques and measures are the tables in IEC61508-2 Annex B and IEC61508-3 Annex A, B. Reasonable techniques and measures have to be selected for each phase depending on the selected criticality. In case the criticality level is unclear, the measures for SIL3 shall be selected.

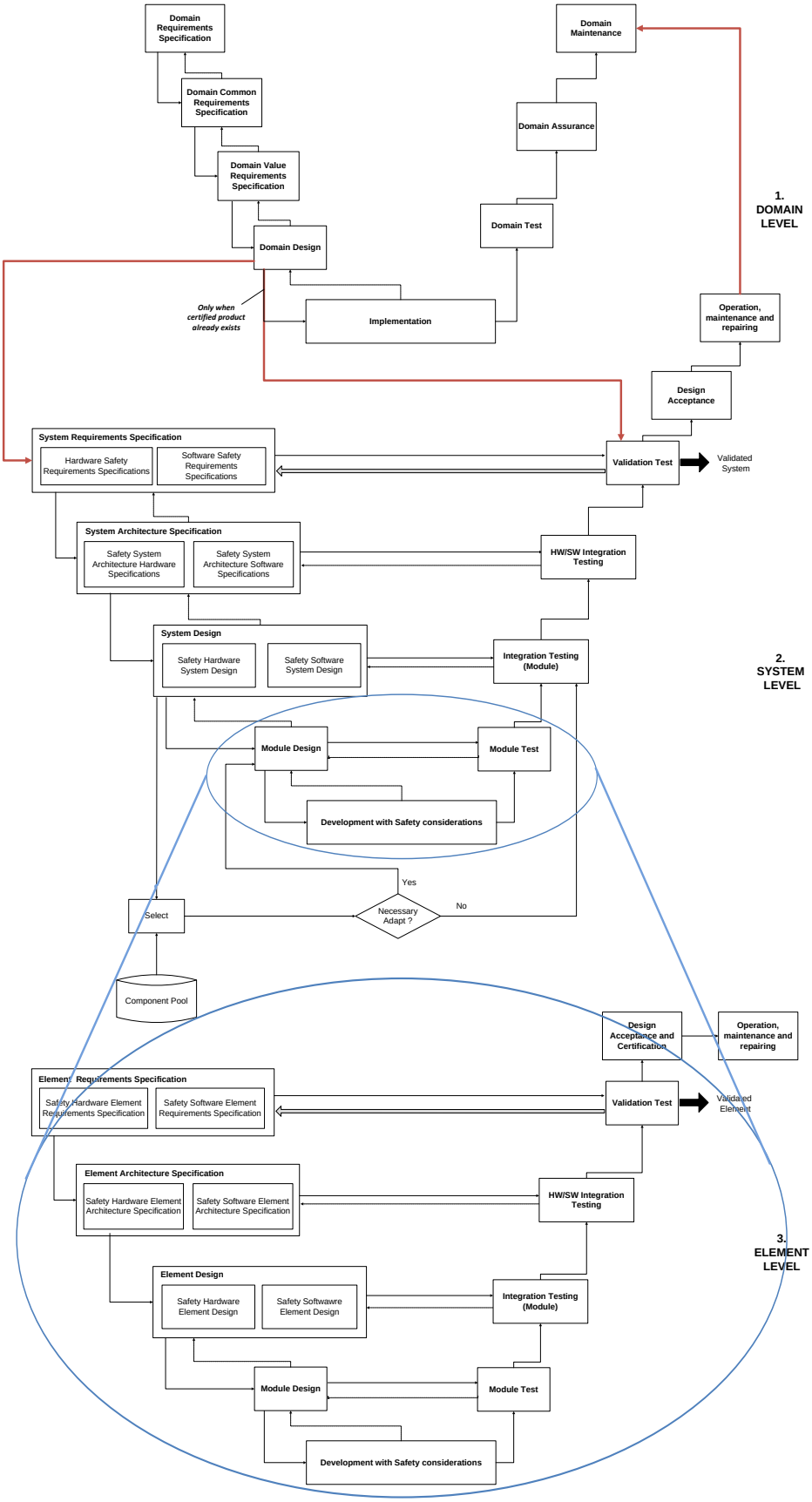


Figure 13: Safety-related Development Process of DREAMS.

2.2 Security Approach

As mentioned in 2.1, the V-shape process is used in DREAMS which can be customized for different industrial domains (e.g., avionics, wind power, healthcare). This is due to the fact that the engineering of safety systems implies enforcing a strict development process and V-shape processes have been used frequently to attain certification in the safety domain. However, the engineering of security systems is not based on the V-Model. In the IT security domain, the evaluations for certification are usually based on ISO/IEC 15408, commonly known as the “Common Criteria” [5]. The Common Criteria (CC) supports an objective evaluation of IT products or systems to validate that a particular product or system satisfies a defined set of security requirements. Although the focus of the CC is evaluation, it presents a standard that should be of interest to those who develop security requirements. However, the CC does not give methodological support for security engineering because its focus is on the evaluation of the security of the IT products and is not dependent on the process which is used to develop the product. Integration of security into the software development process is not specified in the CC and therefore any well known industrial approach can be adapted. Nevertheless, the evaluation role of CC makes it of interest to those who develop security products. The Common Criteria allows for seven Evaluation Assurance Levels (EALs). At the time of this writing, CC version 3.1 (revision 4) is in use. Sixteen countries around the world, including many countries of the EU as well as the US, Canada, Japan and Australia, are a part of the Common Criteria Recognition Act (CCRA). This means that the IT products evaluated in one of these countries will be approved in all of the 16 countries. Additionally, the IT products evaluated on the CC approach get certifications which are accepted in 26 countries.

The Common Criteria is flexible in what to evaluate and is therefore not tied to the boundaries of IT products. The CC approach uses Protection Profiles (PP). A PP is an implementation-independent set of security requirements for a class of Targets of Evaluation (TOEs). PP provides customer desires, needs, and requirements on what is needed. PP is used for a Security Target (ST), which states how the PP will be satisfied by the supplier and what will be provided. A TOE is defined as a set of software, firmware and/or hardware, possibly accompanied by guidance that meets specific consumer needs. A TOE can be a complete system or a subsystem. Examples of a TOE are (a) a complete software application, (b) an operating system, (c) the cryptographic co-processor of a smart card integrated circuit or (d) an IT product or system, together with its documentation and administration that is the subject of a CC evaluation. Finally, an Evaluated System (ES) show that the three representations discussed above, i.e., the PP, the ST and the TOE are all consistent.

Though the evaluation and certification of security systems is supported by the CC, the integration of security in the development process of IT systems is missing. Different approaches for secure software development have been proposed in other standards which can be used to achieve security by design. This includes ISO/IEC 27000 [6], ISO/IEC 21827 [7] and the Trustworthy Security Development Lifecycle (SDL) proposed by Microsoft. This also affects the security aspects concerning security requirements and the corresponding security services.

2.2.1 Impact of security aspects on the DREAMS development process

The security requirements have been collected in the deliverable D1.1.1. Below the most significant security requirements concerning the development process and model transformation are listed. They are classified into development process, threat model, security requirements, modelling and verification.

Req. ID	Description	Reference to
R11.6.	Integration of security in the development process, i.e., security by	Development

1	design.	Process
R11.1.1	The core security services shall include secure communications, secure time distribution and secure execution environment.	Threat Model
R11.1.2	Identification of core and optional security services, security policies and threat models in the DREAMS architecture and provision of security mechanisms to address those identified services.	
R11.2.4	Mechanisms for protection against physical attacks, such as side channel attacks, shall be evaluated and provided if found adequate, e.g., if they do not affect the QoS requirements.	Security Requirements
R11.3.3	A choice of cipher suites shall be provided. A cipher suite includes cryptographic algorithms and their parameters, e.g., key sizes etc.	
R11.3.4	Core security services on the cluster level shall be identified and provided. This includes services such as end-to-end security (e.g., privacy and authentication).	
R11.3.7	Key management for secure communication between the entities on a cluster shall be provided (Mechanisms for key generation, key distribution/exchange, key destruction etc.).	
R9.8.1	The Security Meta-Model for Data Confidentiality shall allow modelling the varying needs of confidentiality.	Modelling
R9.8.2	The Security Meta-Model for Data Integrity shall allow modelling the varying needs of data integrity.	
R9.8.3	The Security Meta-Model for Authentication shall allow modelling the needs for establishing the authenticity of communication partner and the authentication of data origin.	
R11.4.1	Integrity, authenticity and availability shall be ensured for communications and communications partners, in the presence of security threats, such as message sniffing, insertion, modification and denial of service.	Verification
R11.4.2	Security services shall be validated using reasonable attack scenarios and related penetration tests. Attack scenarios in the context of the DREAMS architecture need to be envisaged and implemented to validate the strength of the security services of DREAMS.	

Table 3: Impact of Security aspects into the DREAMS development process.

2.2.1.1 Meet-in-the-middle Methodology

In DREAMS, in order to converge at a final security solution, the security requirements should be fulfilled by the bottom up as well as top down designs. The security requirements should be fulfilled by the platform design as well as the application design to meet at a common middle point. Figure 14 illustrates the meet-in-the-middle design flow approach for DREAMS (section 2.1.1.1). In DREAMS there are various security requirements defined affecting software and hardware. Therefore there are requirements on both design flows, at the top-down design and at the bottom-up design. In the meet-in-the-middle point, all security requirements have to be fulfilled and consistent. This has to be done thoroughly, because attacks on one side could aim on targets on the other side. E.g., a component or module developed for the platform design in the bottom-up could be vulnerable to attacks from a component or module developed in the top-down process for the application design.

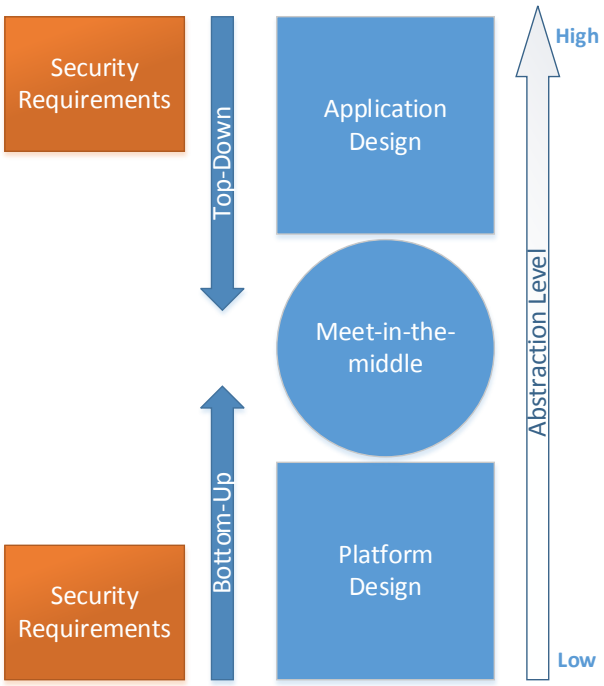


Figure 14: Security requirements in the meet-in-the-middle design flow of DREAMS

2.2.1.2 Threat Model

The security development process uses threat analysis to assess the security needs and identify the security risks. This information is then used to develop security requirements. The security requirements stated in D1.1.1 permit a first view on the security challenges. To get a more detailed view into the security threats, a threat model is needed, which will be an input to the secure development process.

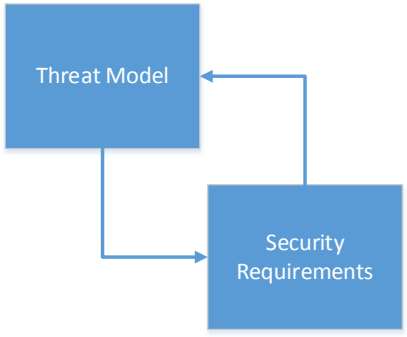


Figure 15: Security Requirements and Threat Model for DREAMS

A threat model describes and analyses the security risks associated with the system. It identifies potential threats to the system and its vulnerabilities. The threat model covers the different individual parts as well as the system in its entirety. Various parts of the system have different attack goals and can lead to diverse benefits for the attacker. Hence, the threat model discusses types of attacks, their functional principles and their impact on the system [8].

Figure 15 shows the relationship between security requirements and the treat model in DREAMS. The threats for the DREAMS architecture identified in the threat model allow a more accurate view on the security requirements. In the threat model, different types of attackers are explained, diverse threats are classified and attack scenarios are illustrated.

The identified threats and attack scenarios can be used to analyse the protection of the system against different attack types in the testing and the verification phase of the development process. This includes different phases like module testing or integration testing.

The threat model and security attacks are described in the deliverable D1.2.1. A thorough analysis and further details for off-chip communication are given in the deliverable D3.3.1.

2.2.1.3 Security Requirements and Modularity

As mentioned in section 2.1.1.3, modularity is an approach to divide a system into smaller parts to reuse them in different systems. The modular structure of DREAMS is decomposed into the four levels system-of-system, system, subsystem and component (Figure 16), which is related to the architectural style as already discussed in section 2.1.1.3. The use of modular design reduces the interdependence between elements of a system and thus reduces the risk that a change or error in one module will have effects on some or all of the other modules. The security requirements describe demands between modules on the same level. There are end-to-end or point-to-point security requirements. Whereas end-to-end security ensures the provision of the requested security services between two modules through the complete communication path between these modules, point-to-point security only ensures the provision of the requested security services between two modules on the path of the communication. E.g., security requirements between two applications are end-to-end requirements and security requirements between two gateways in the communication between two modules are point-to-point requirements.

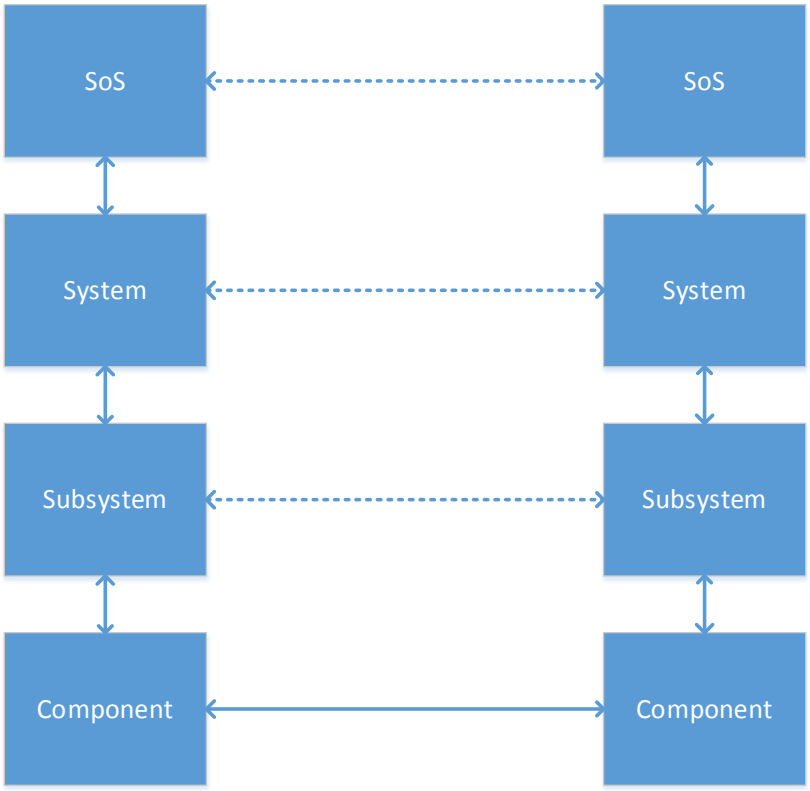


Figure 16: Modular design of DREAMS

However, parts of the system between the modules communicating in a secure way cannot be trusted and have to be treated as insecure. The same applies to the four levels of the modular structure of DREAMS. Underlying modules are not necessarily trustable. Hence, the security services between two modules, e.g., two subsystems, have to assume that there are insecure components.

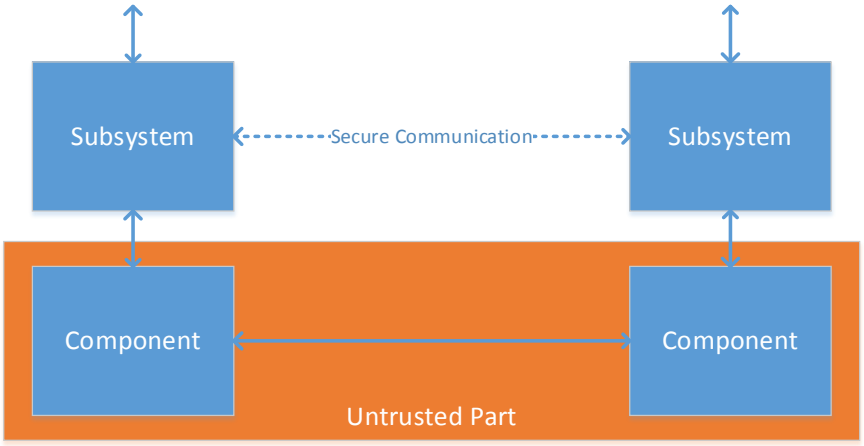


Figure 17: Modular design example with untrusted components

Figure 17 shows an example for a secure communication between two subsystems. They use the components to communicate with each other, but the components are untrusted. Therefore, the communication does not depend on the underlying components. The subsystems can communicate through a black channel and the channel itself does not have to be considered.

Not every communication has to be secured. Nevertheless, parts of the communication path can use security services anyway. E.g., the communication between two gateways is encrypted, but the communicating applications have not requested the encryption. Otherwise, if they need a specific security service, they cannot assume that the specific security service is provided by a module from a lower level.

2.2.2 DREAMS Security Lifecycle Specification

In order to integrate the security into the software development life cycle of DREAMS, different approaches based on [6, 7, 9] can be considered. The SDL proposed by Microsoft [9] is the most relevant approach to software development based on the V-Model.

The DREAMS security development lifecycle can be roughly based on the Microsoft's SDL as discussed above. The SDL process is depicted in Figure 18.

The core phases of the SDL, i.e., requirements, design, implementation, verification/testing and release roughly correspond to the core phases of the V-Model based software development process.

The SDL integrates effective security practices into each phase of the software development lifecycle to improve awareness of security risks. It is also used to realize time and cost-saving benefits from discovering and eliminating the security issues early in the development process instead of fixing the issues as they are discovered later after the software release. This is more of a security by design approach.

The security approach for the engineering process given in ISO/IEC 21827 [7] is shown in Figure 19. The security engineering process is composed of three main areas, i.e., the risk analysis, engineering and assurance.

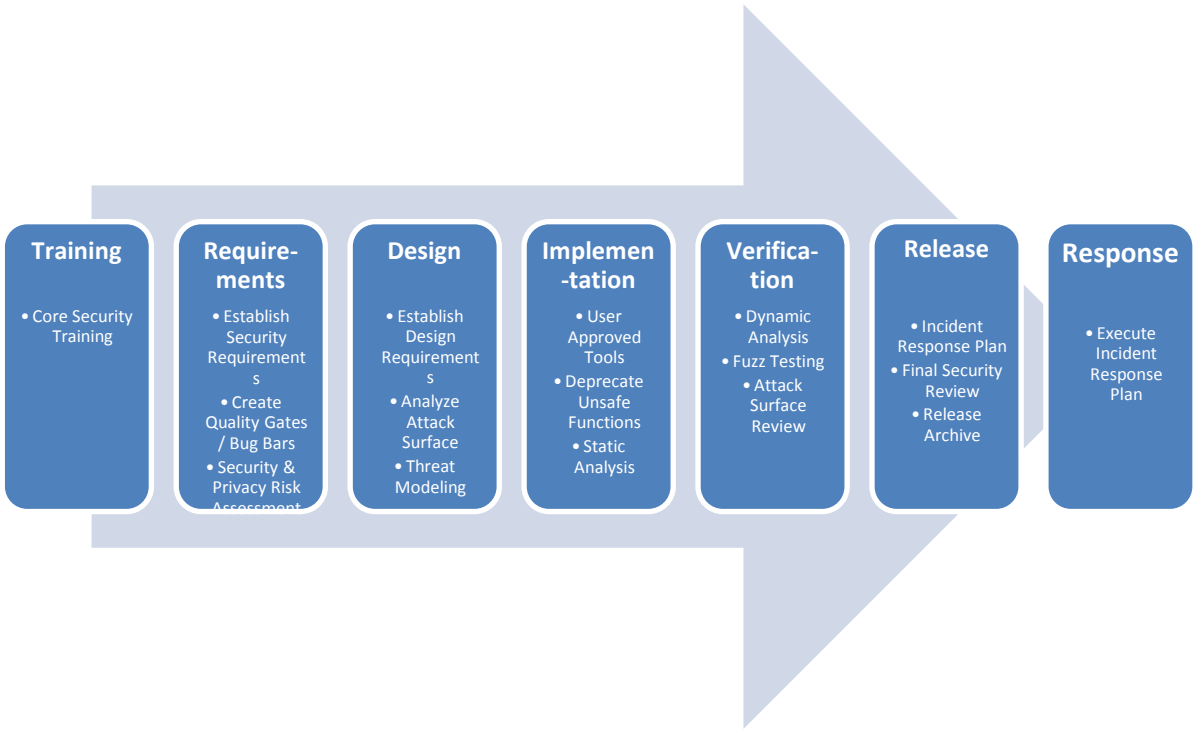


Figure 18: Trustworthy Security Development Lifecycle (SDL) proposed by Microsoft.

The risk analysis identifies and prioritizes the dangers inherent to the product or system being developed. Risk assessment is the process of identifying potential problems to the system. Risks are assessed by examining the likelihood of a threat (or vulnerabilities) and by considering the potential impact of an unwanted incident. An unwanted incident is composed of a threat, vulnerability, and impact.

The security engineering works on the produced risk information to find and implement solutions to the problems presented by dangers. Security engineering is a process that proceeds through concept, requirements, design, implementation, testing and deployment phases. Throughout this process, security engineers work very closely with the system engineering team. Coordination between the security and the system development engineers ensure that security is an integral part of the larger process, and not a separate activity. Using the information from the risk process and other auxiliary information such as the relevant laws and policies etc., the security engineers identify security needs together with the customer. This is followed by the identification of specific requirements. The process of proposing solutions to security problems involves identifying possible alternatives and evaluating the alternatives to determining the most promising one. This process is shown in Figure 19.

The assurance process helps the customers in establishing trust in the security solutions. Assurance is defined as the degree of confidence that security needs are satisfied. It is a very important product of security engineering. There are many forms of assurance, e.g., it might be defined as the confidence in the repeatability of the results from the security engineering process. Although assurance does not add any additional controls to counter the risks related to security, but it does provide the confidence that the controls that have been implemented will reduce the anticipated risk.

Any one or both of the above approaches explained above for security engineering can be used in DREAMS for a secure development process. For uniformity with the safety development lifecycle based on the V-Model the best points from the above approaches are integrated into the V-Model as described next.

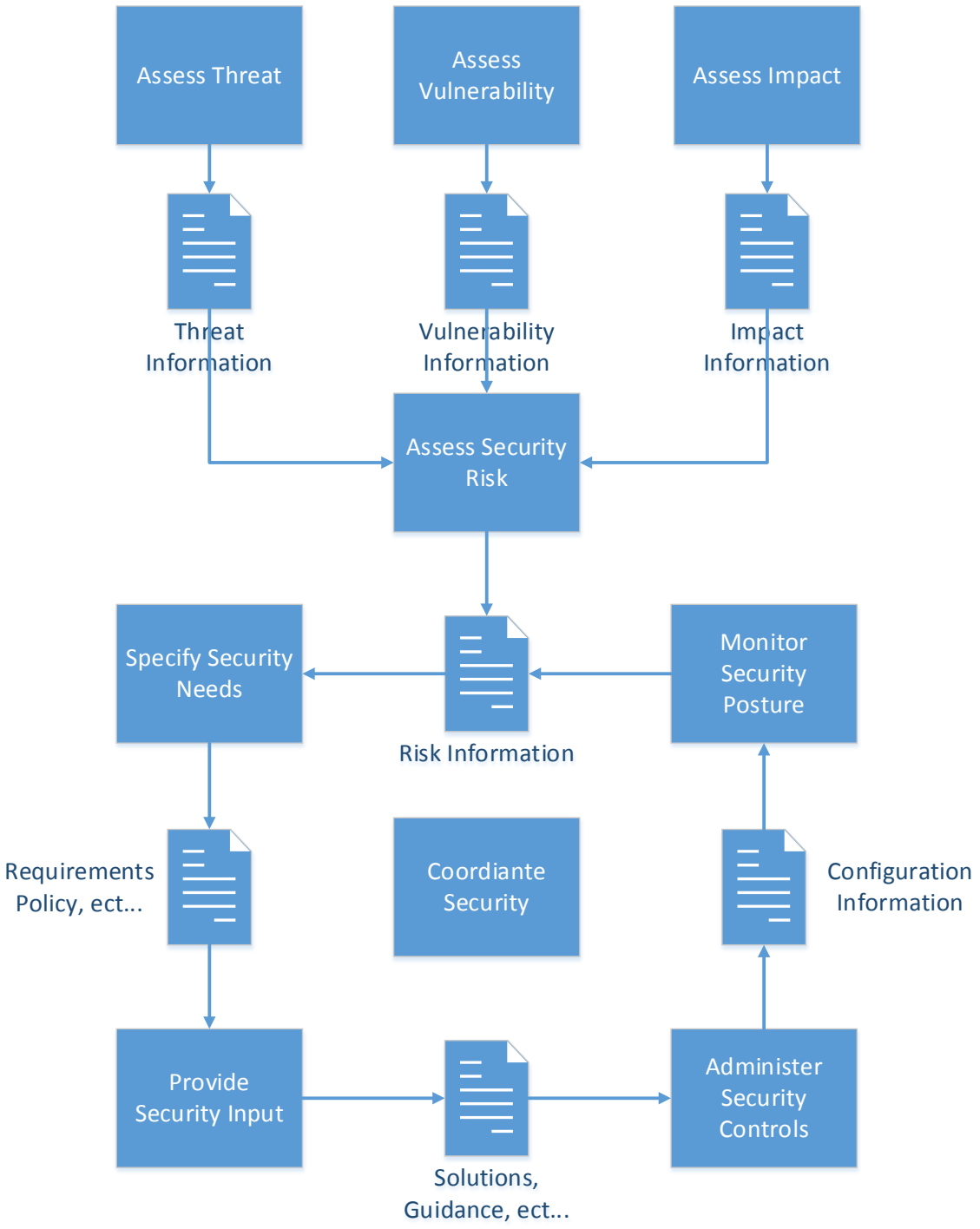


Figure 19: Security Development Lifecycle based on ISO/IEC 21827

However, the safety engineering process shown in Figure 13 is based on the V-Model. In order to come up with a common approach for the development for DREAMS, the different phases of the proposed security engineering architectures of SDL and ISO/IEC 21827 can be mapped to the various phases of the V-Model based approach shown in Figure 13. Based on the combination of these approaches, the secure engineering approach looks as shown in Figure 20.

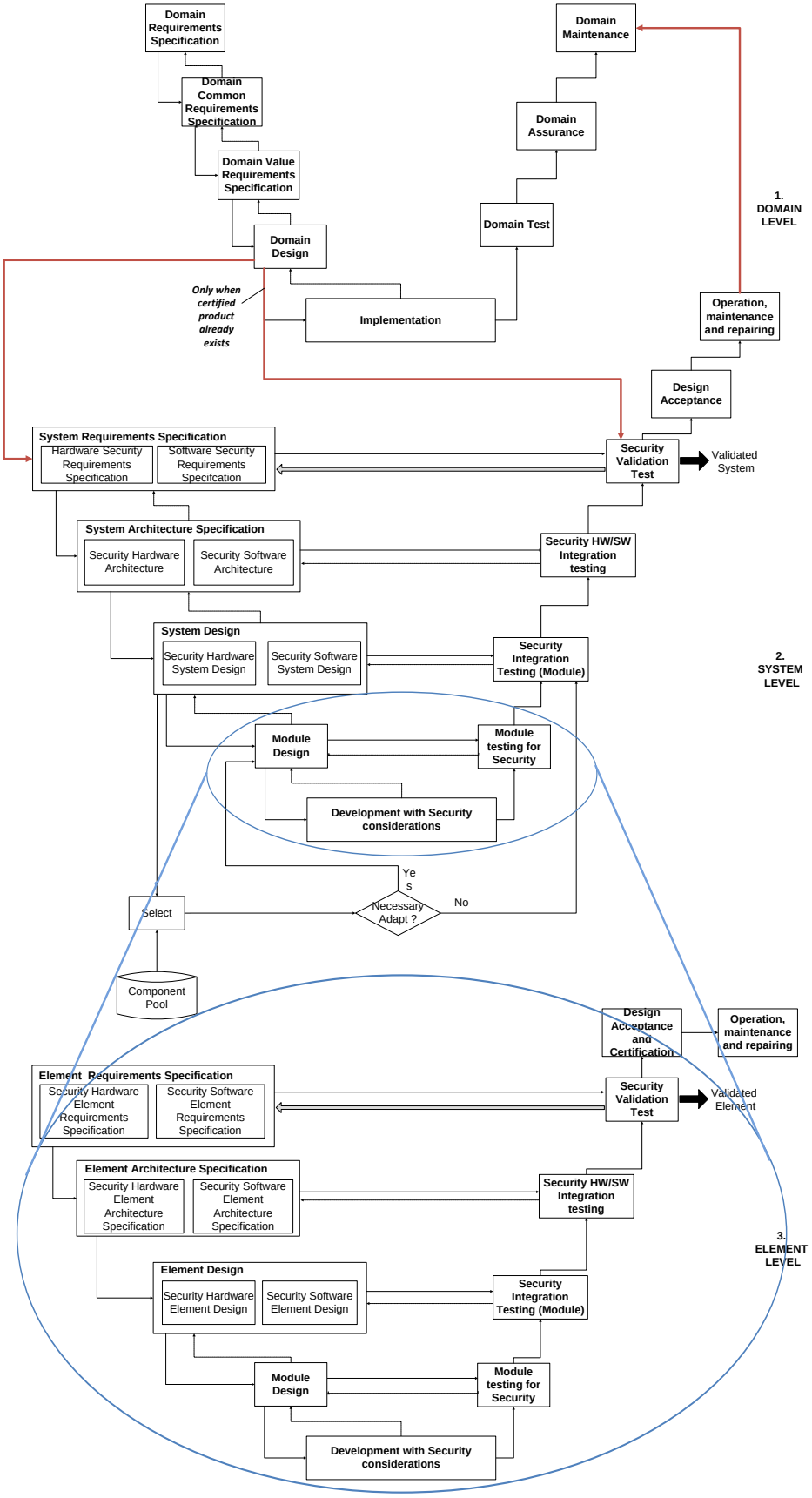


Figure 20: Security Development Process of DREAMS

2.2.3 Measures for fault avoidance within DREAMS

By following the Security by Design approaches described in the previous section different faults due to security flaws such as DoS attacks, buffer overflow attacks, etc. can be avoided. These need to be considered during the risk analysis phase and the output of the risk analysis should be an input to the secure software engineering phase.

2.3 Timing Approach

The consideration of timing constraints and their verification is often neglected but should be considered as a subject on its own: if control orders do not arrive in time at the actuators or are not updated sufficiently often, then the system may get “out of control”. The consequences may be damage to the system or harm to people and thus, without considering timing requirements during the development process, it is not possible to design a safe system. However, timing constraints and their verification through prediction techniques such as worst-case analysis or simulation are based on the assumption that in the real system all parts behave as supposed. If at some time an application actually sends its data over a network much more often than foreseen, it may hinder control orders of other applications to arrive in time, which could lead to failures. This is where safety considerations must come into play, in order to establish acceptable failure probabilities, which are then achieved through an appropriate safety design. In this sense, the Safety and Timing approaches are complementary.

The goal of the TIMMO / TIMMO-2-USE (https://itea3.org/project/timmo-2-use.html) projects was to elaborate a meta-model and a methodology for modelling and verifying timing constraints in the automotive domain. The main results, Timing-Augmented Description Language (TADL) and the Generic Methodology Pattern (GMP) for timing, are however general enough to allow their application to other domains. For this reason it is considered in DREAMS as basis for timing considerations.

2.3.1 GMP

The Generic Methodology Pattern (GMP) is a set of process steps that identify design tasks that are relevant for considering timing constraints and their validation during the development of electronic systems, see [10].

The GMP consists in a generic sequence of tasks that can be executed at every abstraction level of the development process, see Figure 21. These tasks are described in more detail in the following sections.

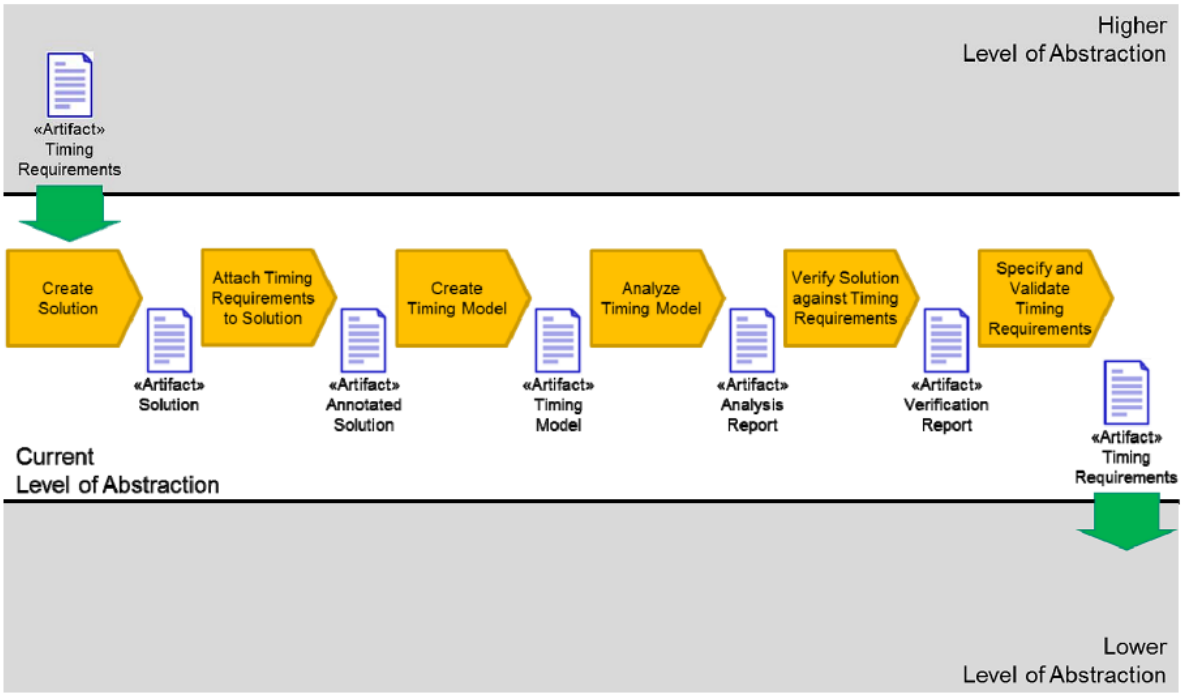


Figure 21: GMP for Timing

The natural flow of execution is from higher abstraction levels to low abstraction levels, but the GMP can be performed on top-down or bottom-up direction.

2.3.1.1 "Create Solution"

This task describes the definition of solution architecture without any timing information. In the GMP, it is a (big) placeholder for all other design activities at the current design level / system view.

2.3.1.2 "Attach Timing Requirements to Solution"

This task "describes the formulation of timing requirements in terms of the current" design level / system view.

This means that for the "input" timing requirements from the previous level/view, the corresponding entities need to be (re)defined in the current view:

- definition of TimingEvents, attached to the structural entities of the current system view
- redefinition of TimingChains in terms of the translated TimingEvents
- redefinition of TimingConstraint with references to the translated TimingEvents

Let us consider the example of latency constraints in the technical view. As explained Section 3.4, a latency constraint is based on a "stimulus" and a "response" event (Figure 36). In the technical view, these TimingEvents would be those related to the reception and the production of data in component ports. As suggested in Section 3.4, these could be called DataReceived (stimulus) and DataSentEvent (response).

2.3.1.3 "Create Timing Model"

This task "describes the definition of a formalized model for the calculation of specific timing characteristics based on properties of the current" design level / system view.

The goal is to define a model (or several models) that will serve as input for the "timing analysis" to be performed in the next task. This may mean to define additional information, like the decomposition of a TimingChain into segments.

Let us return to the latency constraint example, but this time at mapping level. The work of this task would consist in decomposing the "end-to-end" timing chain into segments which span each a different execution / communication perimeter: execution on a tile, communication over NoC, communication over off-chip networks. This decomposition is generally needed so that worst-case analysis or simulation can be applied in order to evaluate timing.

2.3.1.4 "Analyze Timing Model"

This task "describes the actual execution and evaluation of all necessary calculations according to the timing model".

This task consists in feeding the analysis tool(s) with the "timing model(s)" built in the previous step, before running the analysis and finally retrieving the results for verification.

2.3.1.5 "Verify Solution against Timing Requirements"

This task "describes the comparison of the obtained analysis results with the specified timing requirements".

The simple part of this task consists in comparing analysis results with requirements.

In our latency constraint example, the verification would consist in simply checking that the upper bound on (end-to-end) delays is smaller than the latency constraint.

Notice however that if several kinds of analyses have been performed, the results might first have to be merged, before being able to perform the simple comparison. If two algorithms produce upper bounds on Worst Case Traversal Time (WCTT), and if algorithm 1 provides a bound larger than the constraint, then it can only be concluded that we algo1 cannot prove that the WCTT are below the constraint. But if for the same WCTT algorithm 2 provide a bound below the constraint then it can be concluded that the constraint is met. The problem with algorithm 1 is simply that the provided upper bound on WCTT is too pessimistic to allow a positive conclusion. The merging of the analysis result would consist in taking the minimum of the bound computed by the different algorithms.

Notice also that the main objective of this design task is to decide "whether the numbers are good enough for progressing" or "whether those numbers have to be revised". If not, it is necessary to return back to an earlier development step/level (iteration)". The "numbers might not be good enough" if slack is needed for future extensions or in order to compensate for lack of precision in the estimation of timing characteristics.

2.3.1.6 "Specify and Validate Timing Requirements"

This task "describes the identification of mandatory timing characteristics and their promotion to timing requirements for the next development phase".

It is about stating which timing constraints are the inputs for the next design phase and whose satisfaction implies the satisfaction of the original input requirements of the current step. For example, in the case of latency constraints, one can either decide to only keep the end-to-end constraint or to impose sub-latency constraints (=sub-time budgets), for the different perimeters (processing, on-chip-, off-chip networks) that are covered by the end-to-end constraint. In the

second case the solution space is reduced, but the global problem is divided into sub-problems with lower complexity.

2.3.2 Impact of DREAMS characteristics on the development process

In the deliverable D1.1.1 the DREAMS project requirements have been collected. Some of these requirements have to be taken into account for the definition of the development process. Regarding the Timing Approach, the relevant requirements are listed in the following table and referred to as the, modularity and timing concepts. Other concept such as meet-in-the-middle development approach, traceability and some requirements of modularity and heterogeneity are listed in Section 2.1.1.

Req. ID	Description	Reference to
R9.13.5	Integration of an additional application subsystems into an existing system	Modularity / Scalability
R 4.4.3	Methods and tool should at least be suitable for all application domains represented by the demonstrators.	Heterogeneity
R 1.2.2	The architecture shall ensure that all safety-critical subsystems of a system see a sequence of critical events (e.g. reception of messages) in the same order or can re-establish the temporal order.	
R 6.1.2	The end-to-end communication time between two tasks of an application subsystem on the DREAMS architecture shall be less than 50ms.	Timing
R 6.1.3	The DREAMS architecture shall ensure that it is possible to design a solution where two subsystems can be connected in which the time between a data generation from one application subsystem and the time the data has been delivered to another application subsystem is less than 1s. This latency shall be ensured even when the two application subsystems are in different clusters.	
R7.4.1	The wind turbine shall achieve the stop state (safe state) when the speed of the blades is greater than or equal MAX_BLADE_SPEED. The wind turbine shall be in the safe state until a manual reset of the system.	
R 8.4.1	Soft real-time applications shall be combined with non real-time applications in a reliable manner. The focus is on reliable and predictable communication and synchronization between on- and off-chip domains.	
R 9.5.1	The Timing Requirements Meta-Model shall allow the specification of latency constraints (local or end-to-end).	
R 9.5.2	The Timing Requirements Meta-Model shall allow the specification of repetition constraints.	
R 9.5.3	The Timing Requirements Meta-Model shall allow the specification of synchronization constraints (based on events).	
R4.1.3	End-to-end response time analysis algorithm shall be developed which are able to account for scheduling algorithms considered by the resource allocation strategy.	
R10.4.1	Bounded Reconfiguration Time	
R10.4.4	Timely communication between GRM and LRM	

Table 4: Analysis of DREAMS Timing requirements.

In the following section we analyze whether the above mentioned aspects are covered or supported by the GMP and identify missing elements or needed adaptations.

2.3.2.1 Timing

The "timing" related requirements listed in the table are either

- concrete example of latency constraints from the demonstrators (R6.*, R7.*, R8.*)
- requests for the possibility to specify certain kinds of timing requirements (R9.*)
- requests for the possibility to perform timing analyses (R4.*)
- concrete example of timing constraints related to resource management services (R10.*)

The GMP developed in the TIMMO-2-USE project [10] and the associated meta-model [11], cover or support, in principle, all listed requirement related to "timing". "Depending on the specificities of the demonstrators, some adaptations or extension may be necessary.

2.3.2.2 Meet-in-the-middle Methodology

The core entities of the Timing meta-model (TimingExtension, TimingEvents, TimingChains and TimingConstraint) are defined for a certain system view / design level and form a self-contained set of information (together with the system view). Furthermore, the input timing constraints may either come from a higher abstraction level (previous design step) or from a lower abstraction level (existing system). Thus, the GMP does not impose an order in which the timing extension must be defined for the different system view/levels.

Additionally, the GMP foresees a "bottom-up task" called "Abstraction of Timing Properties": refinement of timing properties used at a current level, form estimations drawn from an existing implementation, see Figure 22. One would typically refine the estimation of WCET, based on feedback from implementation.

Thus the GMP is a meet-in-the-middle methodology.

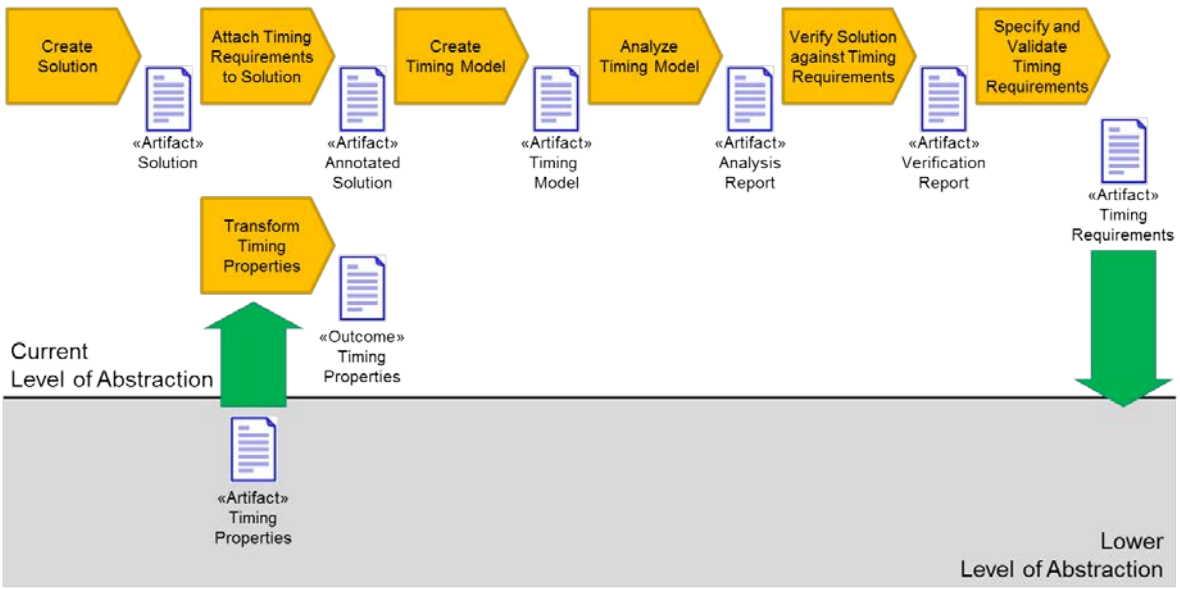


Figure 22: Abstracting Timing Properties (TIMMO-2-USE)

2.3.2.3 Traceability

TIMMO-2-USE TimingConstaint are actually EAST-ADL requirements and inherit, in principle, the related traceability features. Since traceability is an essential aspect of requirements modelling and

may imply relations between different kinds of requirements, a common traceability framework should be used. In the upcoming deliverable D1.4.1 (DREAMS meta-model) will be defined in details how the TimingConstraints will be integrated with the other requirements of the DREAMS meta-model.

2.3.2.4 Modularity

By definition, smaller timing chains can be composed into larger timing chains and larger timing chains can hierarchically be decomposed into sub-chains. In parallel, latency constraints with a larger perimeter can be decomposed in sub- latency constraints with smaller perimeters. Thus, latency constraints and timing chains are naturally compatible with modular design approaches. Synchronization constraints on the other hand, are not modular by nature: a control algorithm, implemented in a software component may require that the different sensor values, needed as input, are measured at the "same" time. In order to express this in the form of a synchronization constraints, it is necessary to refer to "measurement events" in the sensor, but the sensor hardware and the corresponding drivers are not (necessarily) part of the control software component, because they may be located "at the other end" of the system, and the data possibly travels over some network or simply because the sensors and drivers may be provided by different suppliers. In order to overcome this difficulty and preserve traceability, we propose to use "modular" synchronization constraints, which are expressed in terms of events related to the local input and output ports, but with the meaning of placeholders for the actual events in the sensors and actuators, which can only be determined when the component is used in the context of an architecture, see Section 3.3. Thus, the GMP and the Timing-requirements meta-model are compatible with modular system description approaches.

Since the timing model is an "orthogonal" or "annotation" meta-model, it should be possible to combine it with other orthogonal meta-model such as those used for describing variability.

2.3.3 DREAMS Timing Lifecycle Specification

The Generic Methodology Pattern (GMP), for the consideration of timing requirements throughout the development process (see Section 2.3.1), can be mapped to the V-Model based safety related development process of DREAMS (Figure 13), as shown in Figure 23.

The GMP can be "instantiated" at and in between design steps, where entities are considered that consume inputs and/or produce outputs. At a higher level one typically considers a "functional architecture" and at a lower level, a software architecture, consisting of software (modules). Notice that timing requirements can not apply to software alone, because software needs execution (and communication) resources. Thus, timing related activities generally concern the combination of software and hardware, unless a function is implemented only in hardware. For this reason hardware (HW) and software (SW) are not explicitly mentioned in the names of the timing related design step in Figure 23.

Timing requirements can be seen as constraints, imposed in a top-down manner or as properties of existing entities (bottom-up). When working in top-down manner (vertical solid lines in Figure 23), one intents to design entities that satisfy the constraints coming from the previous higher level. The timing properties of the resulting entities are also expressed as timing requirements, since they play the role of constraints for (a) the design of the entities at the next lower level and in (b) the tests of the actual implementations in the corresponding timing related activities in the right branch of the V-Model (horizontal dotted lines in Figure 23). Only during the verification activities in the left branch of the V-Model (vertical dotted lines in Figure 23), are timing requirements of some lower level seen as properties that must satisfy some higher level constraints.

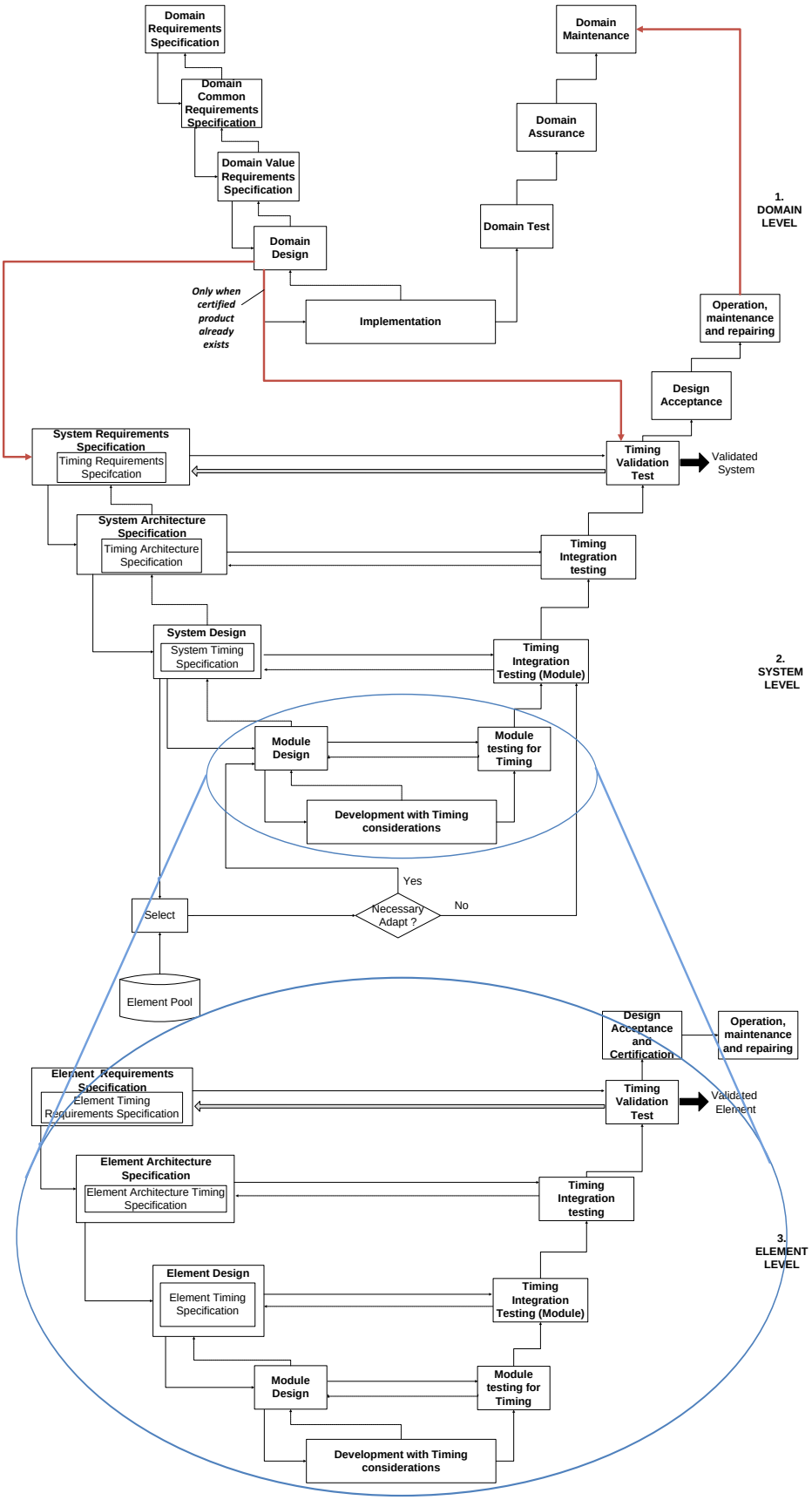


Figure 23: Timing-related Development Process.

2.4 Summary of DREAMS Development Process

Figure 24 shows the DREAMS Development process which addresses the requirements of safety, security and timing approaches. The development processes of each approach have been shown in Figure 13, Figure 20 and Figure 23, but without numeration. The summary of the development process, shown in Figure 24, shows that for each approach, there is one block for each phase of the development process that includes both HW and SW. Therefore, the numeration of phases of each approach is not possible, because each development process of the safety, security and timing approaches, show that for each phase of the development process, there is one block for HW and other for SW, which has been merged in case of the summarized figure.

As shown, DREAMS development process has three levels of development, beginning with the element development process, continuing with system development process and, finishing with the domain level development process. Each level of abstraction follows the V-shape, which is based on the life cycle defined by IEC 61508.

Between each level of development process, are defined the relations/connections that have between them. For example, as shown in Figure 24, the entire element level development process is focused in the phases from 2.4 to 2.6 of the system level development. Therefore, the pre-existent certified elements can be used and reused at system level, although sometimes, some changes on elements shall be needed for adapt them to the system needs. Therefore the recertification process shall be carried out, to prove that upgraded system is enough safe. Same way, a certified system can be used on domain level to generate a product-line products.

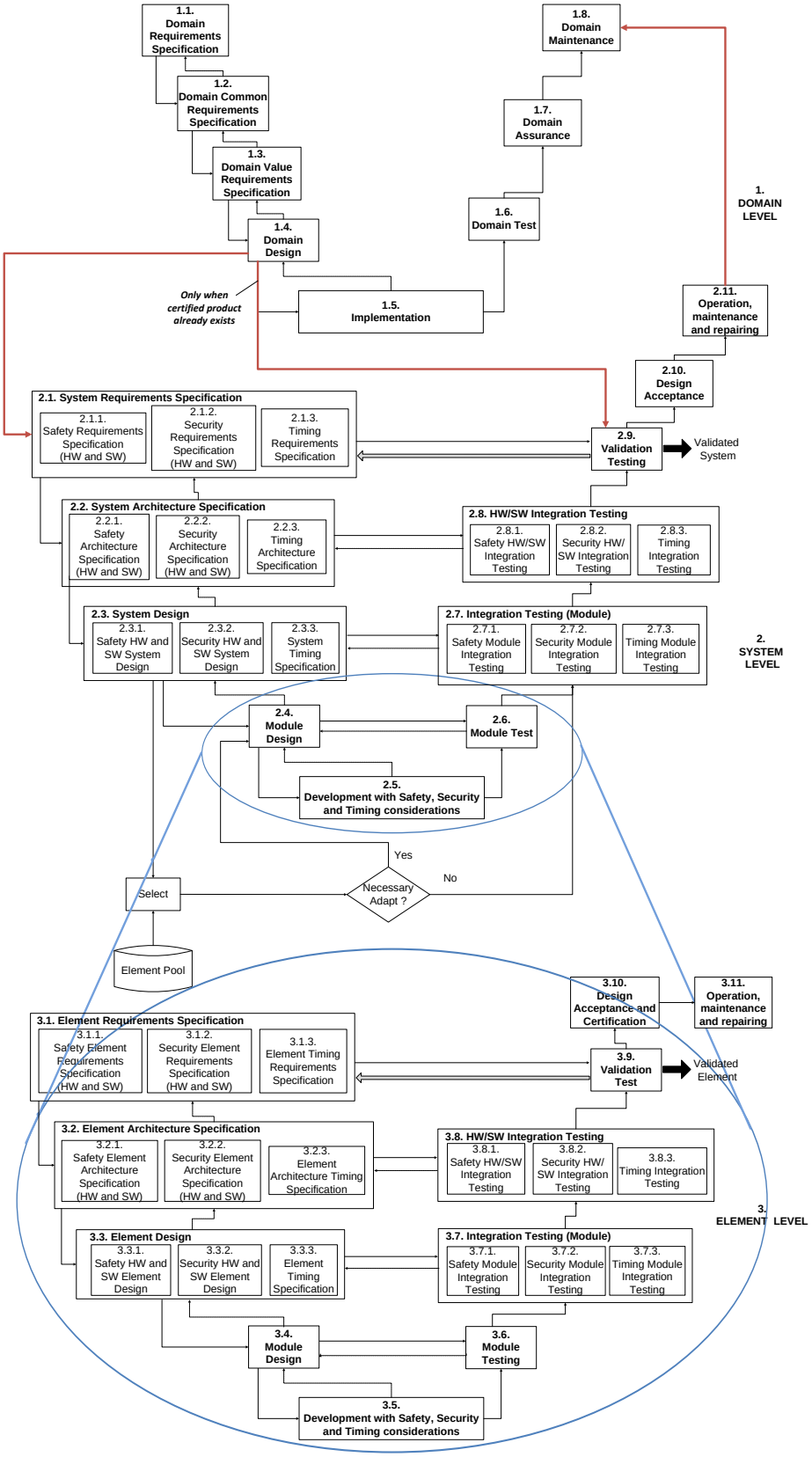


Figure 24: DREAMS Development Process.

3 DREAMS Meta-Models and Model-Transformations

This chapter will provide an overview of the DREAMS meta-models and the model-transformations required for a tool-supported workflow. It is structured as follows:

Section 3.1 starts with an analysis of the impact of the DREAMS characteristics onto meta-models and model-transformations. After that, Section 3.2 provides a brief introduction on meta-modelling, model-driven engineering (MDE) tool-chains as well as a summary of the DREAMS system model introduced in D1.2.1 since it will serve as an input to the definition of the DREAMS meta-model.

After that, a coarse overview of the meta-model used in DREAMS will be presented whose purpose is to provide a specification language for DREAMS-based systems (section 3.3). Furthermore, the timing meta-model which will be applied in DREAMS and the MultiPARTES meta-model will be introduced. Both meta-models will be briefly introduced. The detailed definitions of the meta-models are due in documents D1.4.1 and D1.6.1.

In Section 3.6, the model-transformations required to exchange data between the different tools involved in the development of a DREAMS system will be sketched. This section includes an exemplary instantiation of the toolchain. It covers configuration formats for hardware/software IPs to be integrated into the DREAMS architecture

3.1 Impact of DREAMS Characteristics

In deliverable D1.1.1, the DREAMS project requirements have been collected. Some of these requirements have to be taken into account for the definition of the development process. Regarding the Meta-Models and the Model-to-Model-Transformations, the relevant requirements are listed in the following two tables. The analysis of the requirements onto the meta-models resulted in the following groups (see Table 5): overall design of meta-models, architecture, safety, security, timing, variability and requirements. The aspects safety, security and timing are investigated in Chapter 2 of this document. A preliminary investigation of the majority of the remaining aspects (which are due in documents D 1.4.1 and D1.6.1) is performed in this Chapter.

Req. ID	Description	Reference to
R 9.1.1	Separation of concerns: The meta-model shall be organized in such a way that different aspects are covered by sub-meta-models.	Overall design of Meta-Models
R 9.1.2	The meta-models should exhibit an adequate degree of abstraction, i.e. abstracting irrelevant details while providing the information required for the methods and tools based on them.	
R 9.1.3	The proposed meta-models shall be domain-independent.	
R 9.2.1	The application meta-model / PIM shall capture the structure of applications in terms of their component architecture.	Architecture
R 9.2.2	For the application meta-model, precise (platform-independent) execution semantics should be defined.	
R 9.3.1	The platform meta-model shall capture the topology and the hierarchic structure of instances of the DREAMS architecture	
R 9.3.2	The platform-meta model shall distinguish different types of building blocks / services contained in instances of the DREAMS architecture.	
R 9.4.1	The platform-specific meta-model shall provide means to describe applications that are deployed to instances of the DREAMS architecture.	
R 9.6.1	The Reliability / Safety Meta-Model should allow to specify policies	Safety

	according to IEC-61508 realization phase (system architecture definition). Each safety compliant item of a mixed criticality system can specify a failure probability (e.g., an assigned SIL level). So it is required that during modelling process safety policies are checked (e.g., Chosen SIL level cannot be higher than the maximum allowable SIL).	
R 9.6.2	The Reliability / Safety Meta-Model shall allow specifying criticality-levels according to IEC-61508.	
R 4.2.1	Variability modelling and analysis tools shall be enhanced to achieve by automatic means as well as guided manual means an optimal or best effort configuration of DREAMS platforms and DREAMS systems.	
R 9.8.1	The Security Meta-Model for Data Confidentiality shall allow modelling the varying needs of confidentiality.	Security
R 9.8.2	The Security Meta-Model for Data Integrity shall allow modelling the varying needs of data integrity.	
R 9.8.3	The Security Meta-Model for Authentication shall allow modelling the needs for establishing the authenticity of communication partner and the authentication of data origin.	
R 9.12.3	The development (design, verification, validation) process shall foresee the definition of application timing requirements.	Timing
R 9.5.1	The Timing Requirements Meta-Model shall allow the specification of latency constraints (local or end-to-end).	
R 9.5.2	The Timing Requirements Meta-Model shall allow the specification of repetition constraints.	
R 9.5.3	The Timing Requirements Meta-Model shall allow the specification of synchronization constraints (based on events).Synchronization constraints	
R 9.9.1	The variability meta-model shall allow specifying variations of base models in order to define product lines.	Variability
R 9.9.2	The variability meta-model shall allow to describe different feature sets of applications	
R 9.9.3	The variability meta-model shall allow to describe different implementation alternatives of applications.	
R 9.9.4	The variability meta-model shall allow to describe instances of the DREAMS architecture	
R 5.1.1	Mixed-criticality product line shall be supported to enable certification of product-lines with variability management.	
R 9.6.3	The meta-models should support the traceability between the artefacts used in the different steps of the development process.	Requirements
R 9.13.3	The development process shall support the traceability for requirements regarding: safety, security, etc.	
R 9.7.2	The Energy / Power requirements meta-model should be suitable to define requirements on the energy / power consumption of a DREAMS system at the system-level.	
R 9.2.3	The DREAMS application architecture model should provide means for defining the memory needs.	

Table 5: Impact of DREAMS characteristics onto Meta-Models

Table 6 clusters the requirements on model-transformations in a tool-supported development process into the following groups: complexity management, as well as integration of resource allocation / exploration and analysis methods into the development process (see Section 3.6).

Req. ID	Description	Reference to
R 4.4.1	Design activities of the DREAMS development process shall be supported by a tool chain.	Complexity Management
R 4.4.2	The exchange of data between consecutive tools in the DREAMS development process shall be automated so that it can be performed without “manual” recopying or reworking of the data.	
R 4.5.1	The generation of the configuration files of the DREAMS platform for an instance of the DREAMS architecture, shall be supported by tools that use the system model as input.	
R 9.10.1	The development process should define the model-to-model transformations required to implement application subsystems on top of the DREAMS platform.	
R 9.10.2	The development methodology should allow the definition of the implementation artefacts, i.e. its end products that have to be produced for a DREAMS-based system.	
R 9.11.1	The development process shall support offline real-time scheduling methods for mixed-criticality systems (allocation of functional parts to partitions, time triggered schedules, static virtual circuit arbitration and port scheduling).	Integration of resource allocation and exploration into development process.
R 9.11.2	The development methodology shall support online resource allocation and management strategies, and their relationship to the static methods described in R10.2.1.	
R 9.11.3	The development process should support the design space exploration (DSE) method that provides means for semi-automatic architectural exploration.	
R 9.14.1	The development process shall define how variability is bound.	
R 9.11.4	The development process should support timing analyses, such as the analysis of end-to-end response times induced by a given allocation and scheduling configuration.	Integration of analyses into development process
R 9.13.6	The development process should support a reliability analysis, and methods for the introduction of active redundancy that derive a fault-tolerant design from the original design.	
R 9.7.1	A high level analytical model covering average static and dynamic power consumption of the NoC at the system-level shall be provided, e.g., a mathematical function.	

Table 6: Impact of DREAMS characteristics onto Model-Transformations

3.2 Introduction

3.2.1 (Meta-)Modelling

DREAMS promotes the use of a model-driven engineering (MDE) approach in order to provide the different stakeholders in the system development process with appropriate means to describe the system under design in order to support the different activities in the development process. Here, MDE is defined as the combined use of models for the representation of the system with development processes and analysis methods [12]. The goal of MDE is to provide models for as many development artefacts as possible, which provide different views onto the same system (“everything is a model” [13]).

In general, MDE distinguishes two major model types [14, 15]: Product models are used to describe design and implementation artefacts that are created manually by the developer, or automatically using appropriate model-transformations. In contrast to that, process models provide a formalization of the development process and are used to specify the relation of product models stemming from different phases of the development process. In DREAMS, only product models are considered, since they are intended to be used as data-containers for tools supporting the development of DREAMS-based systems.

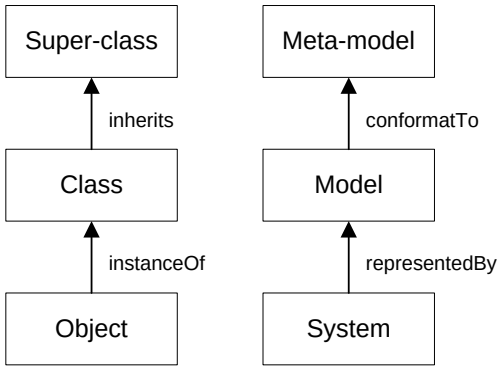


Figure 25: Comparison of object technology and model-driven engineering (MDE)

While the concepts used in object-technology and MDE are identical at first glance [13], there are some subtle differences that will be discussed in the following. The left-hand side of Figure 25 illustrates the well-known abstraction provided by object-oriented languages based on the two relations *instanceOf* and *inherits*. In contrast to that, the right-hand side of the figure illustrates the basic principle of MDE where "a particular view (or aspect) of a system can be captured by a model and that each model is written in the language of its meta-model" [13] (relation *representedBy*). While this looks very similar to the relation *instanceOf* from object-technology, it is important to note that there are arbitrarily many models of some (real-world) system, which individually provide suitable abstractions required for the specific purpose for which the model has been created. In contrast to that, the *instanceOf* is always a 1:1 relation between one particular object (instance) and its type (class). Accordingly, there is a difference between the well-known relations: conformatTo and inherits, which expresses that a model, is expressed in terms of its meta-model. Therefore, a meta-model can be interpreted as the abstract syntax of a language that is used to formalize a given model.

The bottom part of Figure 26 repeats the relationship between system, model and meta-model discussed before (also referred to levels M0, M1 and M2). As it can be seen, a meta-model is expressed in the "language" provided by the meta-meta model (at level M3), that provides basic concepts like classes, references, attributes, atomic types. This principle could be continued infinitely, i.e. the meta-meta-model at level M3 is linguistically expressed in terms of a "meta-meta-meta-model". However, the figure illustrates a different approach where the meta-meta-model at level M3 conforms to itself. This has the advantage that the modelling architecture is built upon a "self-contained" meta-meta-modelling kernel where it is possible to reason about M3 (in terms of M3 concepts) in a similar fashion like M3 concepts can be used to reflect M2.

Since there is obviously no unique choice of the meta-meta-model, numerous suggestions have been made (see [16] for an overview). However, the efforts of the Model-Driven Architecture (MDA) initiative to establish standards for MDE, resulted in the definition of the Meta Object Facility [17], a meta-meta-model that is basically a simplified version of the class modelling capabilities provided by UML 2.0. The MOF 2 core specification [17] provides the basis for further standards in the area of model serialization and exchange, model life-cycle management, transformation languages and

constraint specifications. MOF defines two compliance levels for implementations: Essential MOF (EMOF) and complete MOF (CMOF) [17].

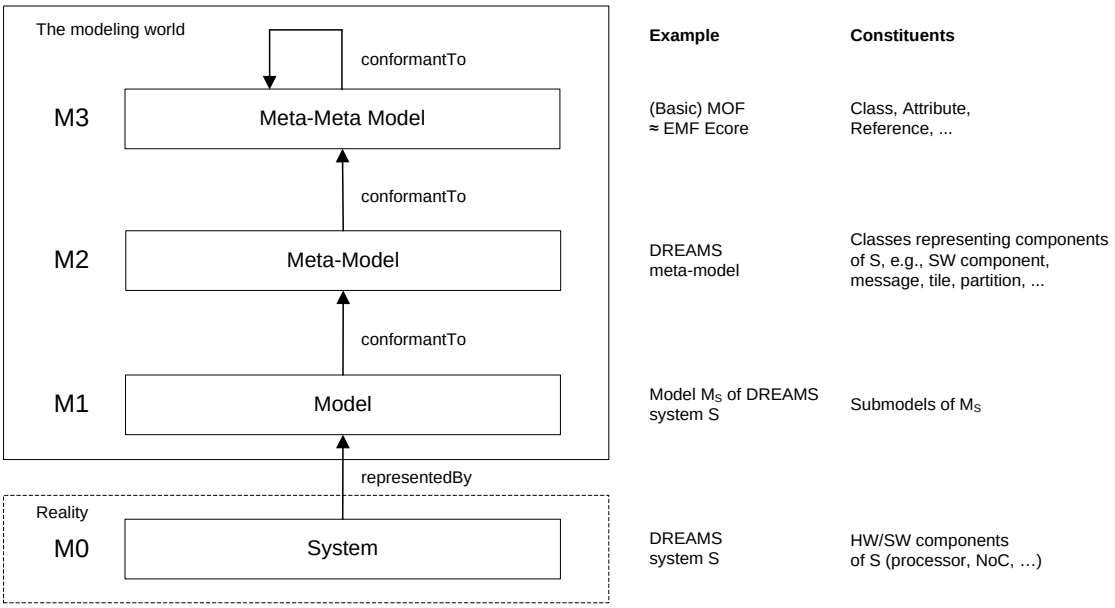


Figure 26: 4-level model architecture

For the DREAMS meta-model, mainly the EMOF compliance level is relevant because the Eclipse modelling framework (EMF), which is the basis for the majority of tools to be developed and/or extend in the course of the project, provides the Ecore meta-meta-model which – to a large extend – is compliant to EMOF (mainly naming differences)[18].

3.2.2 MDE tool-chain

Before the description how an MDE tool-chain can be created based on the concepts introduced in the last section, the application of such tool-chains for safety-critical systems will be briefly discussed. As pointed out in Section 2.1, the DREAMS development process suggests the application of IEC 61508 for the development of safety-critical systems. In this context, IEC 61508-4 classifies tools into the three categories described below that demand different degrees of verification to qualify tools for use in an IEC 61508 development process. The degree of verification needed for tools depends on the possibility and simplicity to verify the output of tools against its input.

- T3 (transformation tool): Tool generates outputs which can directly or indirectly contribute to the executable code of the safety related system. Examples: translator, compiler, linker, assembler…
- T2 (verification tool): Tool supports the test or verification of the design or executable code, where errors in the tool can fail to reveal defects but cannot directly create errors in the executable software. Example: static code analysis, emulator, simulator, test tools…
- T1 (data entry tool): Tool generates no outputs which can directly or indirectly contribute to the executable code (including data) of the safety related system. Example: text editor.

As pointed out in the requirements analysis in deliverable D1.1.1 (requirement R 5.2.2), tools will not be qualified during the project, but the provision of basic requirements will be assessed for tools in categories T2 and T3. These requirements include the availability of a tool manual, a tool errata sheet, at least one test case for each requirement, the management of revisions using a configuration management tool, and the development of libraries according to IEC 61508-3 (safety code).

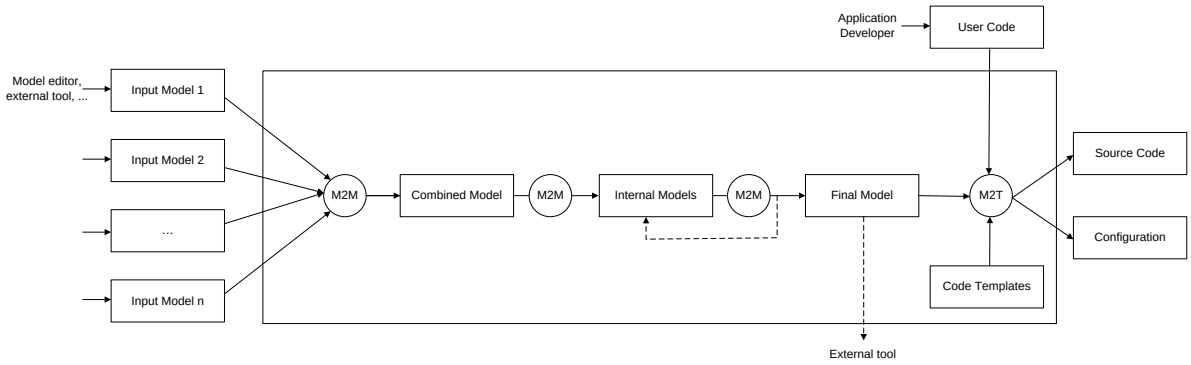


Figure 27: General architecture of MDE tool-chain.

Figure 27 provides an overview of the general architecture of MDE tool-chains. Generally, a tool-chain (or one of its modules) processes a set of input models, which are either created in an internal editor, or using an external (third-party) tool. Based on these inputs, it produces a final output model which is either processed by an external tool, or from which textual artefacts such as source code or configuration data are generated.

As indicated in Figure 27, the transformations required to obtain the final model from the input models can involve several stages. The transition from a model M (conforming to its meta-model T) at a given stage to its successor model M' (conforming to meta-model T') is implemented by a model-to-model transformation T → T'. Here, the complexity of transformations can range from simple aggregation of information or assignment of ID-s to complicated calculations (such as static scheduling).

In order to obtain textual artefacts from the final model, template-based code generators rely on code templates that contain both hard-coded static contents of the output file, and dynamic parts that emit outputs based on references to objects of the input model. For the sake of clarity, the model-to-text transformation usually does not involve extensive calculations, but implements a more or less direct transformation of the final model to a corresponding textual representation by serialization.

The use of model-transformations and code generators in a fully automated work-flow – eliminating manual review steps generally required on intermediate products – typically requires the classification of the corresponding tool to be a member of class T3. As an alternative, the ultimate output of these tools (i.e., program code in a 3rd generation language such as C) can undergo the same (generally both automatic and manual) validation and verification steps as program code that has been manually crafted in the first place.

In the following, it will be sketched how the above tool architecture can be implemented based on the EMF technology. On the one hand, the use of *Ecore* as a common meta-meta-model ensures the syntactical interoperability the different (sub)-meta-model defined within the project. On the other hand, the EMF provides the technological infrastructure that eases the import of XML-based models created by existing (third-party) tool-chains, as well as the implementation of the transformation and generation steps sketched above. The EMF supports the automatic generation of the following components from an Ecore specification of a meta-model, which are required for the implementation of the corresponding modules of the tool-chain (Eclipse plugin):

- Java classes for the in-memory representation of the instances of the user-specified meta-model including the required code to serialize and load (parse) the models to/from an XML representation.

- XML schemas specifying the structure of the XML representation (to ensure interoperability with external tools). Conversely, it is also possible to convert a XML schema specification (e.g., XSD) to an Ecore meta-model, which can be used to create an importer for foreign XML-based formats into an EMF-based tool-chain.
- Basic, but fully functional tree editors that allow the instantiation and manipulation of EMF-based models.

Furthermore, the EMF is accompanied by a variety of tools to ease the processing of models and model-to-model transformations and model-to-text transformations ("code generation"):

- A pragmatic yet efficient way of implementing model-to-model transformations is to code the required calculations directly in Java. As already mentioned, EMF provides automatically generated Java representations of meta-models which provide a comfortable API for instantiation and manipulation
- Dedicated model-transformation languages allow the specification of model-to-model transformations (e.g., *QVT declarative*, *QVT operational*, *ATL*). While these approaches offer a higher degree of abstraction, it should be considered that they increase the number of technologies used in the tool-chain.
- For model-to-text transformations ("code generation"), template-based code generation engines are generally preferred over hand-crafted code generators ("printf"). Here, dedicated template languages (e.g., *Xtend2, Xpand, Acceleo (MOFM2T))* allows the specification of templates that contain both hard-coded static contents of the output file, and dynamic parts that emit code based on references to objects of the input model.

3.2.3 DREAMS System Model

In this section, the DREAMS system model and architectural style introduced in D1.2.1 will be summarized and will be used to relate the meta-models and model-transformations introduced in the remainder of this chapter to the DREAMS architectural style. [2]

In Figure 28, the structure of a DREAMS system is sketched. It is divided into a logical view (of the application) and a physical view (of the platform).

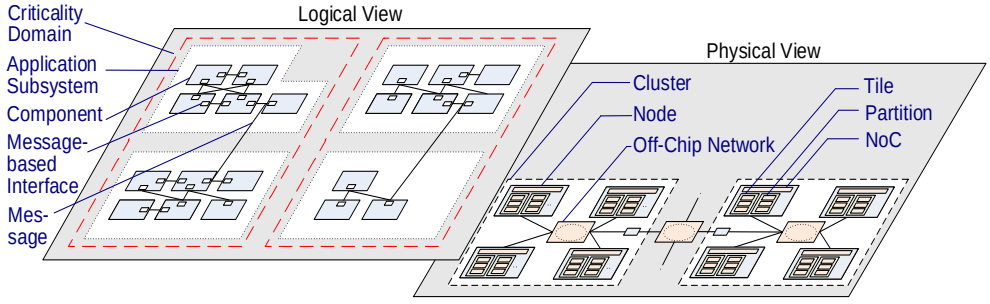


Figure 28: System Structure of Application (Logical View) and Structure of Platform (Physical View) [D1.2.1]

[2] As pointed out at the beginning of this section, this document contains only preliminary information on the DREAMS meta-models. Hence, the presentation of this conceptual system model as well as the description of the corresponding meta-models focuses on the aspects application architecture and behaviour, platform architecture, timing, as well as safety and variability. In the upcoming deliverables D1.4.1 and D1.6.1, additional meta-models will be presented that also cover the aspects security, reliability, and energy. Preliminary information on reliability meta-models are included as part of the object-specification meta-model in Section 11.1.3 of deliverable D4.1.1.

3.2.3.1 Logical view

The purpose of the logical view is to provide the following information about mixed-criticality applications in a platform-independent way:

- Criticality-levels: A system is structured into criticality levels from the different application domains (e.g., DAL A to E in avionics, ASIL A to D in automotive and SIL 1-4 in multiple domains according to IEC-61508).
- Component service: The specification of a component's interface defines its services, which is the intended observable behaviour as perceived by the transmission of messages as a response to inputs, state and the progression of time. Three types of messages are distinguished based on their timing, namely *periodic messages*, *sporadic messages*, and *aperiodic messages.*

As pointed out in D1.2.1, the above definition of the logical view induces the logical namespace *Criticality.*Subsystem*.Component.Message*. The mapping to the physical namespace can be performed by a (run-time) translation layer in hardware or software, or at design-time (using a fixed mapping).

3.2.3.2 Physical view

In this section, the physical system structure of the DREAMS platform consisting of networked multi-core chips will be described. It is described by the physical view that defines the following hierarchical structure:

- The overall system is physically structured into a set of *clusters*.
- Each cluster consists of *nodes*.
- Each node is a multi-core chip containing *tiles*.
- A tile can be a processor cluster with several processor *cores*, caches, local memories and I/O resources. Alternatively, a tile can also be a single processor core or an IP core.
- The processor cores within a tile can run a *hypervisor* that establishes time-and-space partitions, each of which executes a corresponding software component.

The communication between the above entities is provided by the following components:

- The connection between clusters is provided by *inter-cluster gateways* that are formed by *off-chip gateways* located between two clusters.
- Nodes are interconnected by an *off-chip real-time communication network*.
- Tiles are interconnected by a *Network-on-Chip* (NoC). Each tile provides a *Network Interface* (NI) to the NoC offering ports for the transmission or reception of NoC messages.
- A *on/off-chip gateway* is responsible for the redirection of messages between the NoC and the off-chip communication network.
- *Off-chip* and *on-chip networks* are responsible ensuring for time and space partitioning as well as the integrity of messages between the respective communication partners.

The physical view induces the following namespace: *Cluster.Node.Tile.Port.*

3.2.3.3 Application-platform mapping

In order to provide their specified services, components from the logical view must be mapped to the resources of the physical platform:

- Components must be assigned to partitions with suitable computational resources

- Messages must be mapped to the communication networks, taking into account the required timing and reliability properties, and the routing between different parts of the physical platform.

In order to address the challenge of mapping (logical) messages to a (potentially multi-hop route) in the physical platform, the architectural style provides *Virtual Links (VLs)* as an abstraction hiding the physical system structure of the platform from the component. The timing and reliability properties of a VL are determined by the properties of the constituent physical networks. It is defined as an end-to-end multicast channel between the output port of one sender component and the input ports of multiple receiver components, and can be identified using its Virtual Link ID (VLID). VLs are defined for time-triggered transmission of periodic messages, and the rate-constraint transport of sporadic messages. In contrast to that, aperiodic messages do not employ the VL abstraction, but are subject to connectionless transfer and include the physical name of the receiver for the routing through the network instead.

Here, the Network Interfaces (NI) mentioned above either send messages locally to a receiver within the same node, or direct it to the corresponding gateway. The gateway will use the VLID (or the physical receiver name in case of aperiodic messages) to generate the off-chip path to the gateway at the receiver node that will route the message to the corresponding tile.

3.3 AutoFocus3 Meta-Model

In the following, a conceptual overview[3] of the relevant parts of the AutoFOCUS3[4] meta-model will be presented, which will provide the basis for parts of the DREAMS meta-model. AutoFOCUS3 is an Eclipse-based tool prototype that supports the development of embedded systems based on the Focus modelling theory [5]. In its meta-model, systems are described using component models of the software architectures which are enriched with specifications of the executable behaviour (e.g., I/O automata).

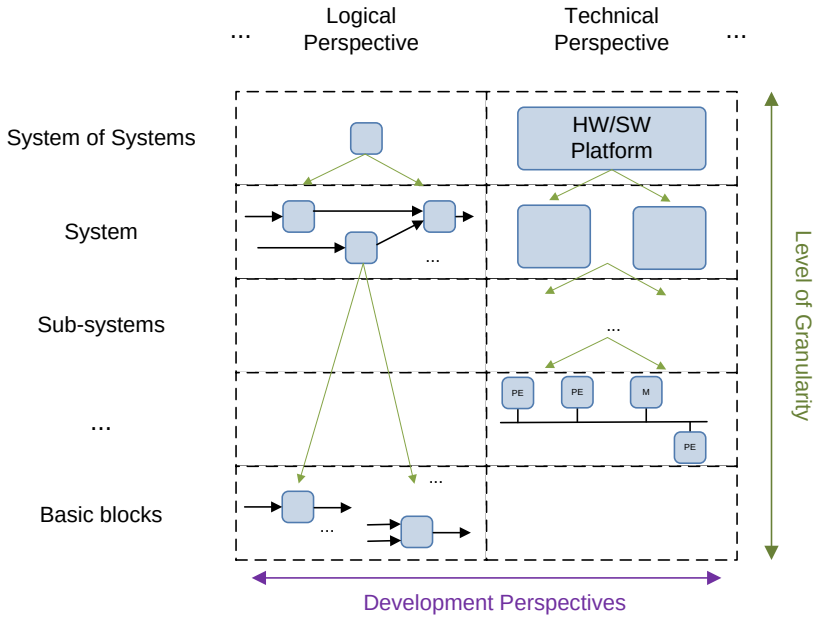


Figure 29: Dimensions of abstraction: granularity levels, and development perspectives

[3] For the sake of clarity, the presentation abstracts from the tool implementation, and focuses on core concepts.

[4] http://af3.fortiss.org/

The execution platform is described using a topology model containing the corresponding hardware and software elements (e.g., execution units, communication transmission units and endpoints, etc.). Finally, a deployment model is used to specify how an application is a mapped to the platform (e.g., application components to execution units).

In AutoFOCUS3, two dimensions of abstraction are applied (see Figure 29). In the granularity dimension, a system is decomposed into sub-systems which are at a lower granularity level and which can themselves be regarded as systems. The process can be applied recursively, until finally basic building blocks are reached. On the other hand, a system can be regarded from different perspectives (or views) that provide different information about the system. In Figure 29 only those perspectives are shown that will be used in DREAMS for the description of mixed-criticality systems (i.e., logical and technical architecture). For these perspectives, also the principle structure of the corresponding meta-models will be sketched in the subsequent sections. AutoFOCUS3 also provides further perspectives that will be briefly described in the next sections (i.e., requirements and deployment perspective).

3.3.1 Requirements perspective

Figure 30 shows a screenshot of the AutoFOCUS3 requirements perspective. It is provided by the MIRA (Model-based Requirements Analysis) module that can be used to create the following models:

- Elicitation and specification of system context
    - Glossary
    - Requirements sources
- Specification of system requirements
    - Requirement models: Template for specification of requirements. It contains information such as ID, title, description, rational, author, requirements source, references to external resources.
    - Use case models can be used to describe a desired functionality in a specific scope of the system. Use cases on the one hand provide similar information as the requirement models described above. On the other hand, they also provide the possibility to describe the involved actors, the trigger conditions, the required inputs, outputs, and pre-conditions as well as the guarantees that hold after the execution.
- Management and tracing of requirements
    - Requirements attributes: In addition to the descriptive attributes mentioned above, also attributes that describe the state of a requirement in the course of the development process can be modelled (e.g., status, priority, opens issues, etc.).
    - Requirements relations: Both requirement models and use cases provide the possibility to create traces to related requirement models, and to components of the system specification.

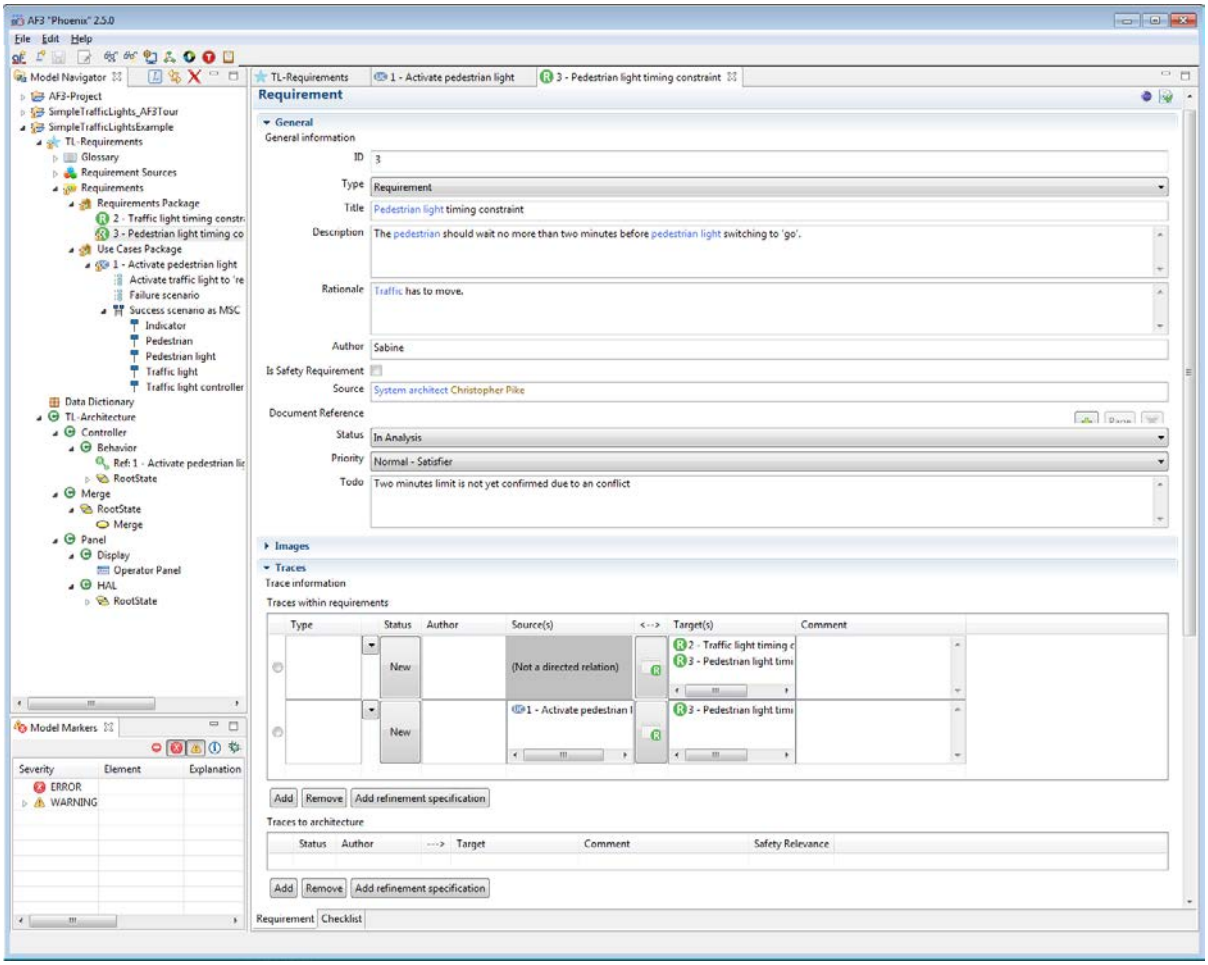


Figure 30: AutoFOCUS3 Requirements Perspective

The requirements perspective can be used to support the software "requirements specification" phase in the V-shape Software development process depicted in Figure 24.

3.3.2 Logical Perspective

The logical perspective describes the application of the system by a network of communicating and cooperating components (see Figure 31). Hence, the logical perspective corresponds to the platform-independent model (PIM) in the MDA terminology, and provides the following abstraction:

- Hierarchical structuring of the application by means of components.
- Definition of the syntactical interfaces and information flow between the system and its environment (input and output communication) and between the different sub-systems (local communication).
- Implementation of the system functionality and its interaction with the environment by means of behaviour specifications.

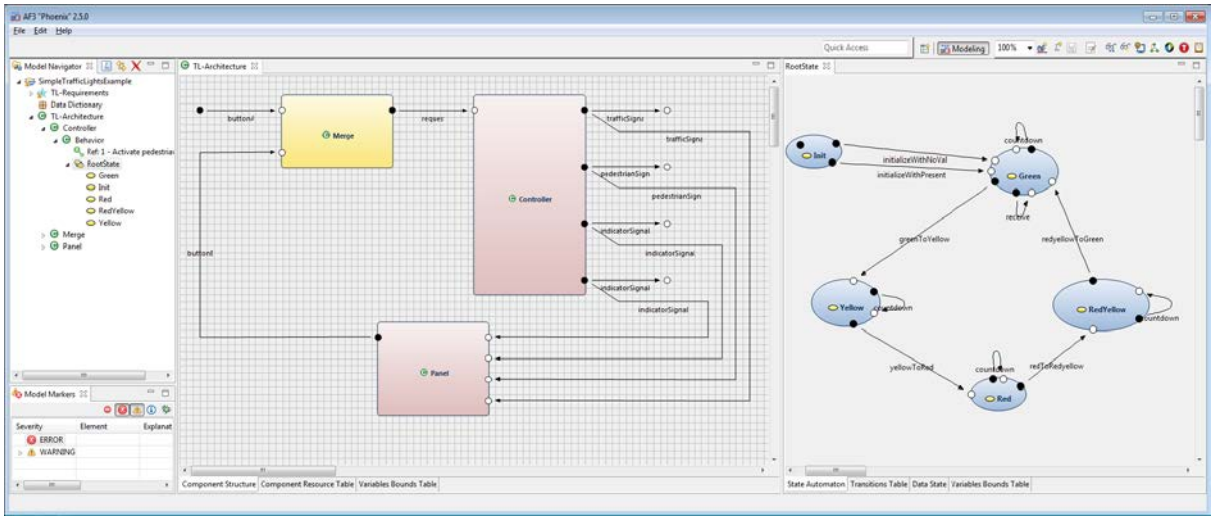


Figure 31: AutoFOCUS3 Logical Perspective

The logical perspective can be used to support the "software architecture", "software system design" and software "module design" phases in the V-shape software development process depicted in Figure 24. In conjunction with a code generator (see Section 3.2), also the software "development" phase can (but does not need to) be based on the models provided by the logical architecture.

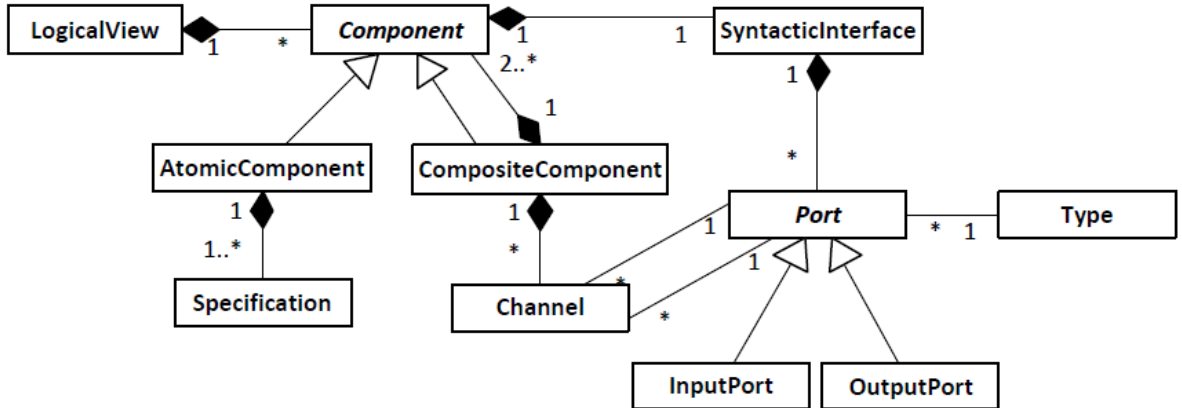


Figure 32: Meta-Model of Logical Perspective

3.3.2.1 Summary of Meta-Model

In the following, the main concepts[3] of the meta-model of the logical perspective will be described (see Figure 32).

- **Component**:
  The logical view consists of a set of components. A component is a syntactical and semantic projection of the overall system with the outermost component giving the interface between the system and its environment. In the FOCUS theory, a component is mathematically defined as a mapping from the valuations of input stream histories to the valuations of the output stream histories [19]. A component is a named element that has a syntactical interface and may be atomic or combined. An example of component architecture is shown in on the top left diagram of Figure 31 (components *Merge, Controller, Panel*).

- **Atomic Component:**
  An atomic component is given by one or more (alternative) behaviour specifications describing the input/output behaviour of that component.

- **Composite Component:**
  A composite component consists of a network of two or more components (each being atomic or, again, a composite). Composite components are used to structure the system architecture into smaller units and connecting them with communication links (called channels). The behaviour of the composite component is defined by the parallel composition of the behaviours of the sub-components [19].

- **Specification:**
  A specification is a generic mechanism used to provide additional information to model elements such as components, ports and channels. For instance, it is used to define the behaviour of a component and using different specification formalisms (e.g., I/O automata as shown in the right diagram of Figure 31, or table-driven specification). Further examples for specifications will be presented in Section 3.3.2.2.

- **Syntactic Interface:**
  A component has a syntactic interface consisting of (named) input and output ports.

- **Port:**
  A port is an interaction point between the system and its environment. Note, that any port has at most one source of information, i.e., atomic output ports receive the data to be sent from the atomic behaviour, while composite output ports are connected to at most one atomic output port via a channel. Analogously, atomic input ports receive their data from a composite input port or an atomic output port, respectively.

- **Type:**
  Each port is either associated with an atomic data type, or a composed data type (structure, array, enumeration).

- **Channel:**
  A channel is a named element that specifies a directed connection between an output port and an input port of two components.

3.3.2.2 DREAMS logical view

In this section, the relation of the excerpt of the meta-model implementing the logical perspective of AutoFOCUS3 and the logical view of the DREAMS system model (see Section 3.2.3.1) will be discussed.

There is an apparent overlap between DREAMS logical view and the meta-model summarized above:

- Components: Atomic and composite components can be modelled using the corresponding classes introduced above. Their syntactical interface can be described using typed (and named) ports.
- The observable behaviour of components can be modelled using the specifications mentioned above (e.g. I/O automata).

Furthermore, specifications can be used to enrich the model elements with additional information required to describe mixed-criticality applications (by defining dedicated specification classes). Examples include:

- Criticality-level can be annotated to components using a dedicated specification type (*SafetySpecification*).
- Static memory demands of components
- Channels can be annotated in order to distinguish between periodic, sporadic and aperiodic messages.
- Following a similar approach, a meta-model for security properties can be integrated.

3.3.3 Technical Perspective

The technical perspective describes the platform on which the system is to be executed on. It corresponds to the platform model (PM) in MDA terminology. The left part of Figure 33 shows an example of a simple technical architecture that consists of two ECUs that are connected by a network.

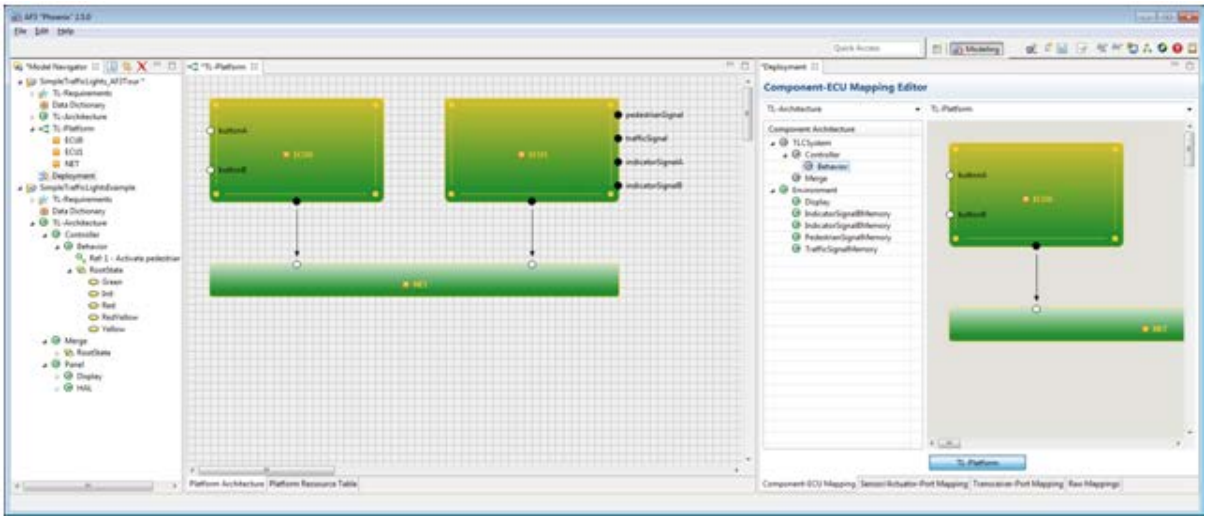


Figure 33: AutoFOCUS3 Technical Perspective and Deployment Perspective

The technical perspective can be used to create models of the HW/SW architecture of the target platform. On the one hand, these models abstract the artefacts from the “Hardware Architecture” phase in Figure 7. It should be emphasized that these models are not actually used in a hardware development process, but are used to support the “HW/SW integration” phase. On the other hand, the models of the technical perspective also provide an abstraction of the platform software components (e.g., partitions provided by a hypervisor).

3.3.3.1 Summary of Meta-Model

Figure 34 shows the basic concepts[3] of the platform meta-model (in black) and examples of platform-specific specializations (in red).

The technical meta-model can be extended for specific technical platforms by providing specializations of the respective base classes. In T1.4, an extension of the technical meta-model will be developed that captures the technical architecture of the DREAMS platform.

The technical perspective provides the following abstractions:

- Hierarchical structuring of the processing and communication units of the execution platform.
- Description of specialized peripheral components (like sensors and actuators).

In the following, the different base classes will be briefly presented:

- **Computing Unit:**
  A computing unit provides an abstraction of an execution container for the application. A model of a computing unit may contain various types of (physical) ports as an interface to its environment. Furthermore, each computing unit may have an internal structure decomposing it into smaller computing and transmission units.

- **Transmission Unit:**
  A transmission unit connects multiple computing units via transceiver ports.

- **Port:**
  A port is a generalization of hardware units or virtual machine parts that connect computing units with their environment and allows data exchange.

- **Type:**
  (Physical) ports are usually typed with low-level types like integer or Boolean.

- **Transmitter:**
  A computing unit can provide information to its environment via transmitter ports, representing abstractions of communication interfaces provided by the platform (e.g., bus interface, off-chip gateway). The output ports of logical components are mapped to transmitter ports in the technical architecture.

- **Receiver:**
  A computing unit can acquire information from its environment via receiver ports. Input ports of logical components are mapped to receiver ports.

- **Transceiver:**
  A transceiver allows the computing unit to send and receive information. Transceiver ports are connected to some transmission unit. They usually represent network driver software provided by the operating system or the virtual machine of the computing unit.

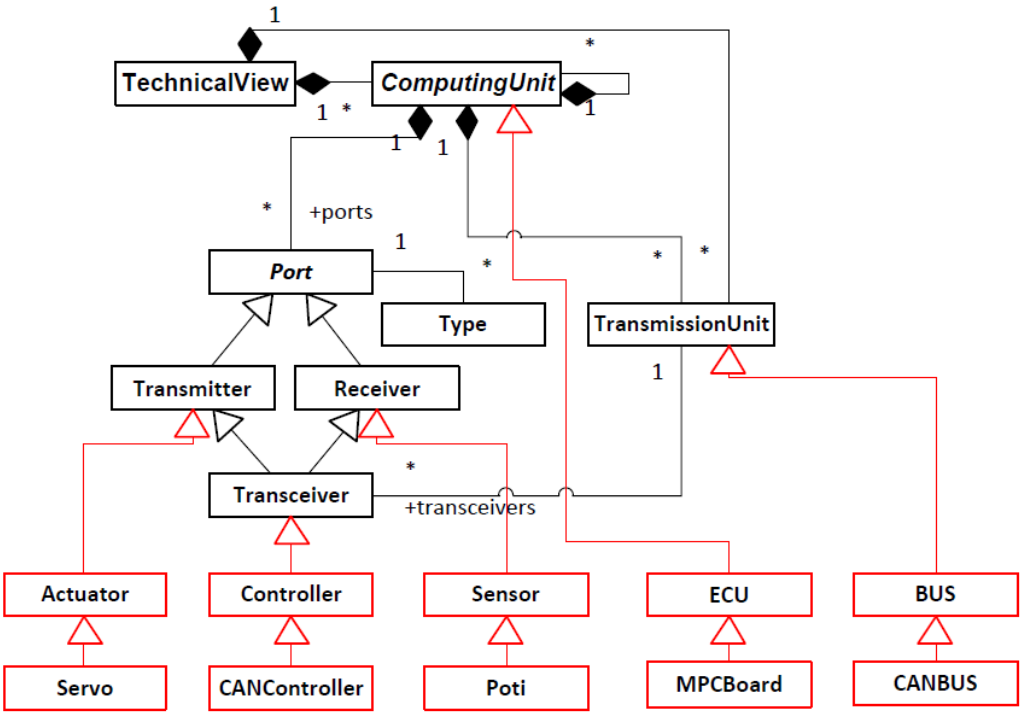


Figure 34: Meta-Model of Technical Perspective

3.3.3.2 DREAMS technical architecture

As pointed out above, the AutoFOCUS approach suggests the implementation of dedicated technical architecture meta-models. For the DREAMS project, this meta-model will cover the following aspects of the physical view of the DREAMS system model (see 3.2.3.2)

- The platform organisation into nodes, tiles, the processor cores of a tile, the partitions provided by a hypervisor will be reflected by specialized *computing unit* types.
- On-chip, off-chip and inter-partition communication will be described using derived *transmission units*.
- The meta-model will be extended with a mechanism to model gateways between different levels of the platform hierarchy, e.g. between the on- and off-chip level.

3.3.4 Deployment Perspective

The deployment perspective (see right part of Figure 33 for an example) is used to model how components from the logical architecture are mapped to the technical architecture. Hence, the models of this perspective support the platform/software integration activities preceding the "Integration Testing" phase of a V-shape model Figure 24.

In the following two sections, the TIMMO-2-USE and the MultiPARTES project are introduced. They are relevant for the fine-grained definition of the DREAMS meta-model that follows in D1.4.1 and D1.6.1.

3.4 Timing requirements Meta-Model

The meta-model developed in the TIMMO-2-USE project for the description of timing requirements [11], was designed to allow the extension of different systems views (of existing meta-models) with timing related information [10]. This way, the TADL language provided by TIMMO2-USE provides

mechanisms to reference to an external meta-model (e.g., AutoFOCUS3). The details of the reference to an external meta-model are specified in D1.4.1.

In case of the DREAMS development process, as specified on Section 3.6.2, Figure 49, the different modelling languages (AUTOFOCUS, TIMMO-2-USE) are used to describe the different aspects structure/behaviour and timing.

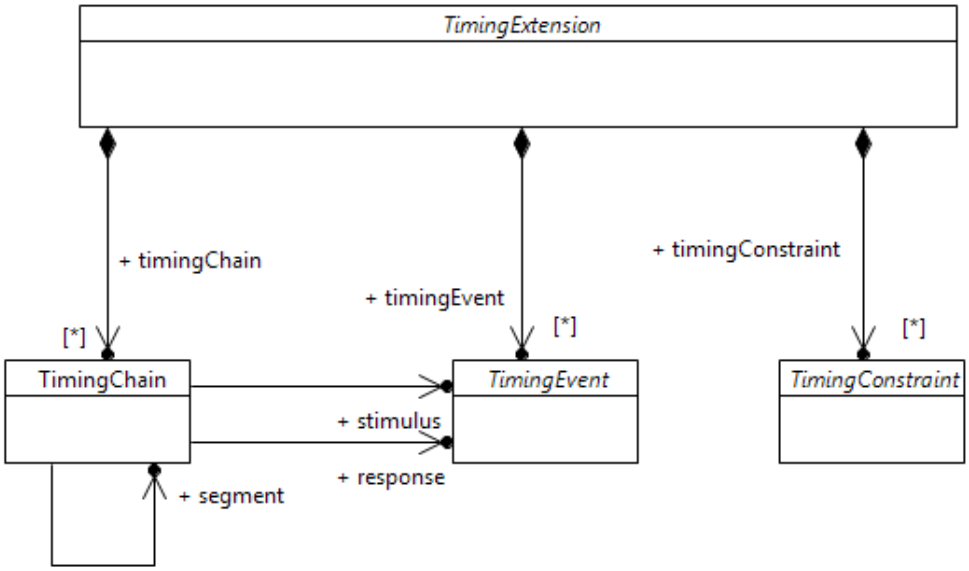


Figure 35: Timing Extension

This has been applied during the TIMMO-2-USE project to the different system views of the EAST-ADL and AUTOSAR meta-models. The same approach will be applied to extend the logical, technical, and mapping view of the DREAMS meta-model. In this section, an overview to the TIMMO-2-USE timing model is provided. A more detailed description as well as the integration of with the other meta-models used in DREAMS will be provided in D1.4.1.

The core concept of the timing meta-model is the so-called TimingEvent class, see Figure 35. It "denotes a distinct form of state change in a running system, taking place at distinct points in time called occurrence of the event" and to which the timing meta-model allows to attach TimingConstraints directly, or indirectly through TimingChains, see Figure 36. The timing related information of a system view is grouped into a TimingExtension entity, see Figure 35.

A TimingChain is a container for a pair of TimingEvents that are causally related. The "stimulus" event is supposed to trigger actions that lead to the "response" event. It is not necessary to know these actions, just that the "stimulus" event does lead to the "response" event. A TimingChain can be hierarchically decomposed into "segments", which are TimingChains themselves. This allows refining a TimingChain along with the refinement of the system description in the same or a different system view.

A whole range of different kinds of Timing Requirements has been defined in TIMMO-2-USE. In Figure 36, we show three latency constraints that are likely to be relevant for DREAMS. More Timing Requirements, such as Repetition, Order and Offset Constraints might also be needed, but this will depend on the specific needs of the demonstrators and will be defined in the upcoming deliverable D1.4.1, which defines the DREAMS meta-model.

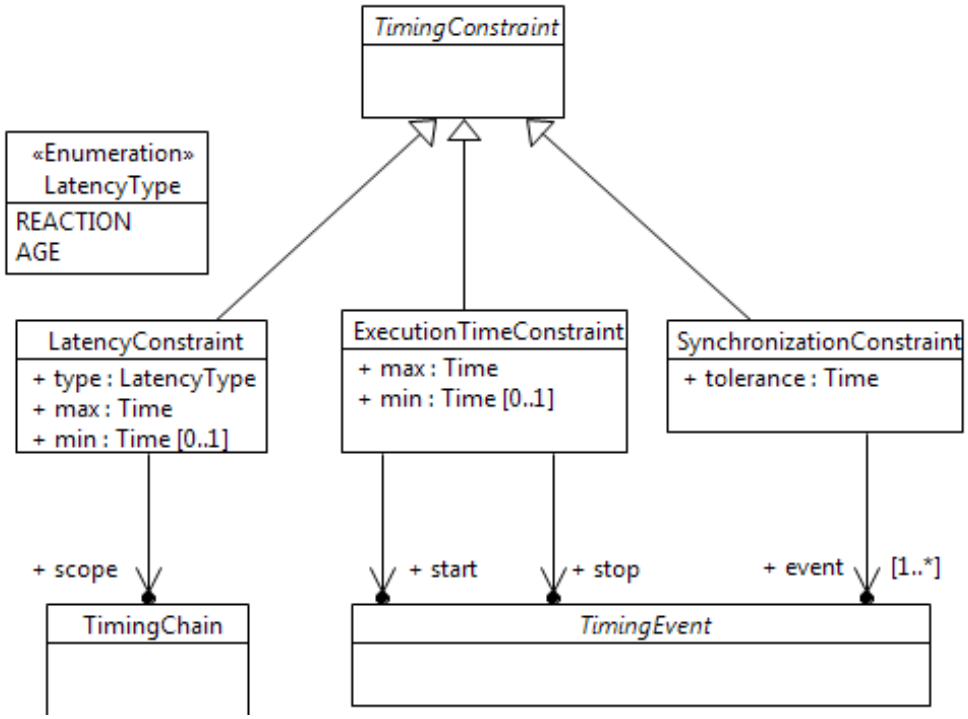


Figure 36: Timing Constrains

The first kind of constraint is the so-called LatencyConstraint that comes in two variants:

- "A ReactionConstraint defines how long after the occurrence of a stimulus a corresponding response must occur". This requirement is useful, if it is important that a system reacts to a stimulus in a limited amount of time.
- "An AgeConstraint defines how long before each response a corresponding stimulus must have occurred". This requirement is useful, if it is important that sensor values, used by a control algorithm are not "too old".

The second kind of constraint is the SynchronizationConstraint that "describes how tightly the occurrences of a group of events follow each other", see Figure 36. These events are typically related to the measurement of sensor values or the applying of actuator values. In order to describe a (software) component that implements a control algorithm in a self-consistent way, when a modular approach is used, where the implementation of a control algorithm and the needed sensors and actuators are handled separately, a dedicated variant, called modular synchronization constraints can be used, see Figure 37. The modular synchronization constraint is part of the component that implements the algorithm and it references events related to input and output ports of the component, but considers them as place-holders for the events in the sensor and actuators that are actually connected in a concrete architecture.

The third kind is ExecutionTimeConstraints that "limit the time between the starting and stopping of an executable entity (function), see Figure 36, not counting the intervals when the execution of such an executable entity (function) has been interrupted". ExecutionTimeConstraints impose limits on WCET of software functions, for example. ExecutionTimeConstraints are similar to LatencyConstraints, since both limit the duration of an activity. But LatencyConstraints also include the time periods where the activity is actually not progressing because the needed resources are allocated to other activities, whereas ExecutionTimeConstraints exclude interruption times.

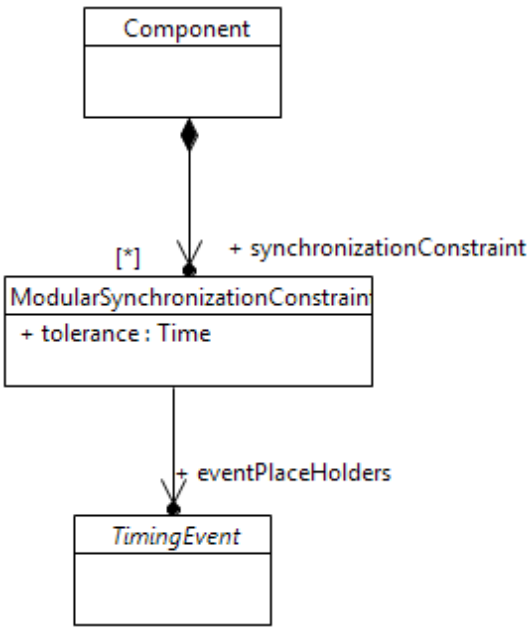


Figure 37: Modular synchronization constraints

Let us underline here that seen top-down, timing constraints are indeed constraints on the properties of artefacts of the following design steps, but that seen bottom-up, the timing constraints play the role of properties. This is in particular true for ExecutionTimeConstraints: if a system is designed from scratch, i.e. necessarily top-down, it may become necessary at some time to make resource allocation decision, before (bounds) WCETs are available. In that case, resource allocation decision can be based on ExecutionTimeConstraints, which play the role of assumptions regarding the later software implementation. When software implementations have been implemented, and their WCET are below the ExecutionTimeConstraints (on the chosen execution platform), then the ExecutionTimeConstraints are actually also properties - in the sense that their execution times are lower than the upper bound declared by the ExecutionTimeConstraint.

All DREAMS relevant timing requirements will be listed in the upcoming deliverable D1.4.1, which defines the DREAMS meta-model.

Since a TimingExtension entity does group all timing information related to a certain view of the system, we define view specific sub-classes that inherit from TimeExtension and point to the system view that is concerned, see Figure 38.

TimingConstaints impose constraints on the occurrences of events. But the concrete definition of these events depends on the system view that is extended with timing information. In Figure 39 we show examples of events that could be relevant in the DREAMS technical view. The DataReceivedEvent represents the fact that input data is ready in an input port of a component. The DataSentEvent represents the fact that the component has made available new output data in an output port. This would for example allow defining a ReactionConstraint on a component that implements a control function: the delay between the availability of sensor data (at an input port) and the production of the corresponding control data (at an output port), must not be longer than a specified maximal time.

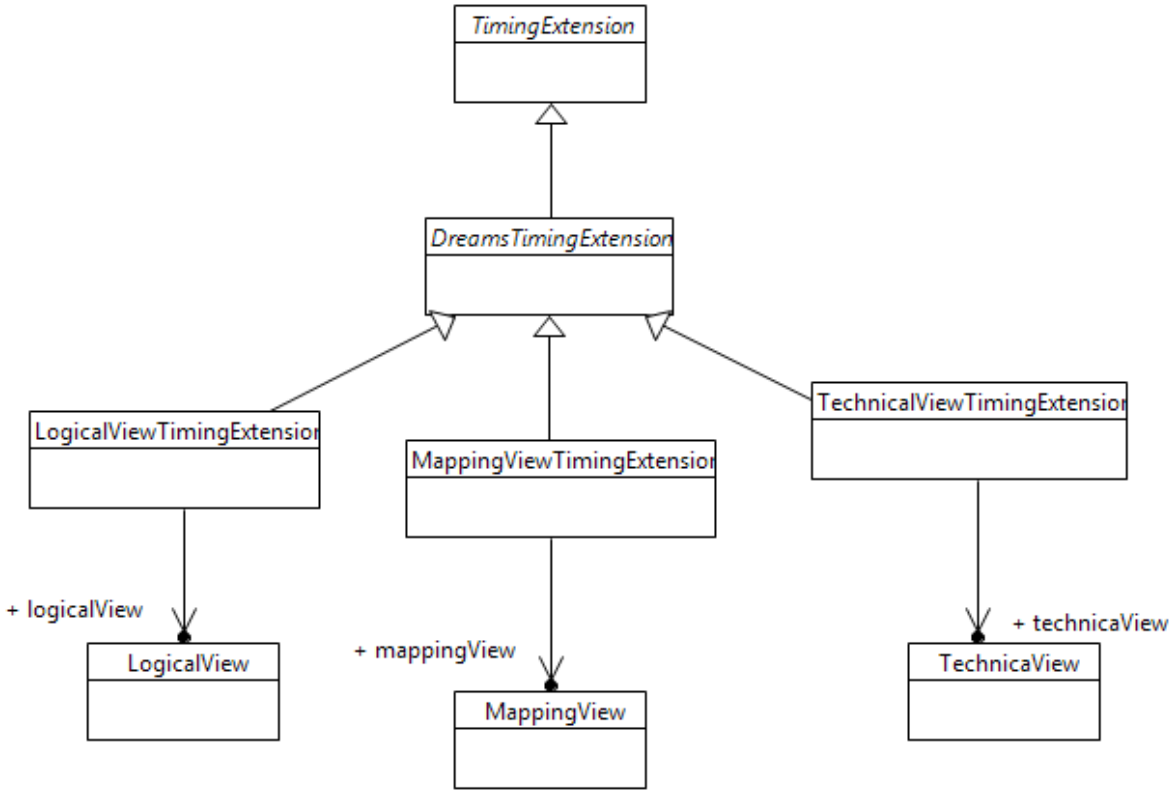


Figure 38**: DREAMS Timing Extensions**

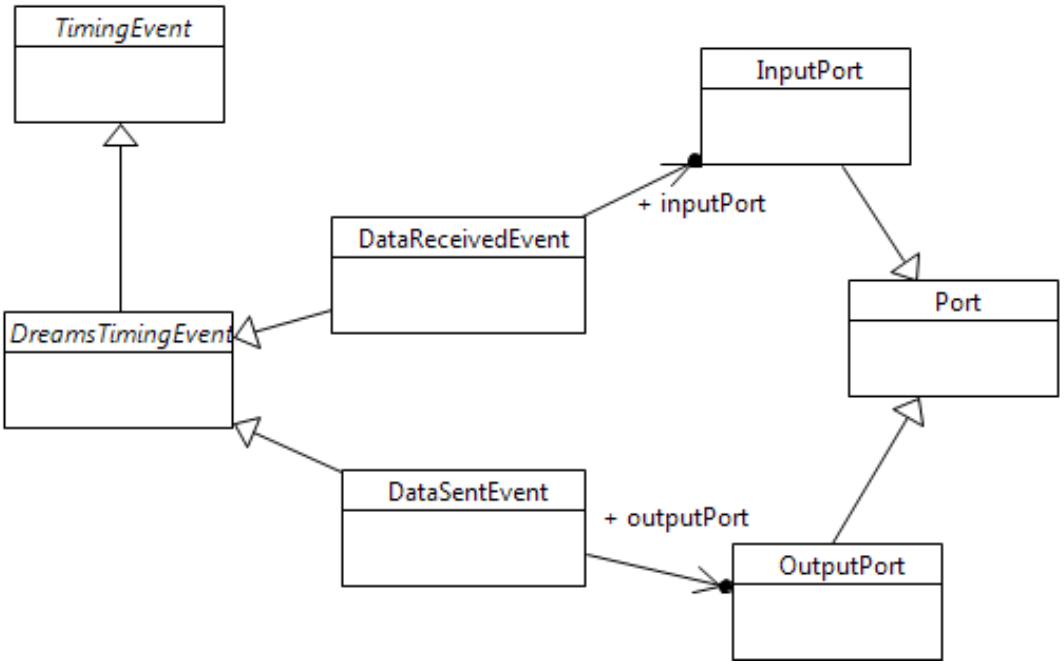


Figure 39: DREAMS Timing Events at software component ports

The exact list of the events and their semantic will be defined in the upcoming deliverable D1.4.1, which defines the DREAMS meta-model.

3.5 MultiPARTES Meta-Model

This section contains a description of the work carried out in the European Project FP7 - MultiPARTES (Project Nº 287702), in the following MPT. Following subsections describes main models, metamodels and functionalities of the toolset developed in MPT focusing on safety and certification aspects. The aim is to take advantage of the work done about Safety and Certification and, together with Variability Concepts, not covered in MPT, enrich the model, metamodels and transformations defined in DREAMS.

The proposal is reusing and adapting safety related models developed in MultiPARTES, more precisely:

- Safety Consistency Model
- Safety Compliance Model
- Diagnosis Techniques and Measure Models
- Safety Standard Model

These models (described briefly in section 3.5.2 that capture safety related concepts) would keep the appropriate pointers to the other models of the project where needed. Along with this, the proposal is reusing the Safety Validation Rules Checking Mechanism and Certification Document generation depicted in section 3.5.3 and 3.5.4.

Figure 49 shows where safety related models and even product line (not developed in MultiPARTES) would fit in the development process, mainly in the left side of the V-shape model, but also helping in certification aspects in the right side of the V-shape development process.

Precise relation with other models would need a very detailed study that would be done as part of next task T1.4

3.5.1 Overview: Meta-Models, Toolset and Transformations

3.5.1.1 Meta-Models

As part of the project, the work in MPT has focused on defining models and tools for managing safety concepts, checking the safety consistency of the platform, and modelling mixed-criticality systems in a multi-core platform based in hypervisor partitioning technology. Two kinds of models have been defined:

Models to define the SW and HW System

- Application Model
- Deployment Model
- Platform Meta Model
    - Hardware Platform Model
    - Hypervisor Model
    - Operating System Model
- Real Time Policy model
- Partitioning Restriction Model

In principle, it is not planned to use these models in DREAMS because we assume that AutoFOCUS and TIMMO-2-USE models provides these modelling capabilities. However, perhaps it would be

interesting for DREAMS to check in next Task T1.4 if parts of these models can enrich or complement DREAMS models.

Last model “Partitioning Restriction Mode” is used to help a Hypervisor Partitioning Algorithm developed in MultiPARTES. Perhaps it would be interesting to investigate in the next task if partitioning algorithm developed in MultiPARTES can be used in DREAMS in conjunction with other models.

Models to represent Safety Related Concepts

- **Safety Consistency Model** - that takes into account properties/attributes related to safety and IEC-61508 specific concepts. The model is basically a hierarchy of Safety Compliant Items (SCI) to be described later.

- **Safety Compliance Model** - that sets IEC-61508 related safety considerations to a SCI. ‘Safety Manual’ of each SCI includes diagnostics techniques and measures.

- **Diagnostic Techniques and Measures Model -** that sets diagnostic techniques defined in IEC-61508 to avoid and control failures during operation.

- **Safety Standard Model -** that enumerates a set of integrity levels for each Safety Standard.

3.5.1.2 Toolset

The toolset developed in MultiPARTES to manage these models include, very briefly described:

- **System Modelling**: Allowing defining the platform model, the application model (with annotations for safety, real time, etc) and generating a restriction model to help partitioning tool.
- **Partitioning tool**: In charge of generating the partitioning that will be represented in the deployment model, the assignment of application to partitions, the characteristics of the partitions, operating system, resources, etc.
- **Generation of Artefacts**: When the system partitioning is correct, a number of transformation tools generate a set of outcomes that are necessary for creating and building the final system: XtratuM configuration files, System building files, Source code Skeletons, etc.
- **Safety Consistency Checking:** tool with the following functionality
  - **Checking Consistency Rules:** checking constraints for:
    - **Safety consistency -** as for example, ‘Rule 2-61508: Safety certification standard supported by any 'compliant item' must be compliant with the system certification standard all the components of a system must be compliant with the certification standard of the system. If any component is not compliant with this standard, the system will not be considered consistent. If any component is not compliant with system’s standard, but supports a derived standard it will be considered consistent, but a warning message will be shown.
    - **System integrity -** as for example, ‘Applications can only be mapped to supported operating systems’
  - **Certification Documents**: producing useful documents for certification.

- **Generation of Restrictions**: populating 'restrictions' models from consistency information.

Figure 40 summarises functionality described above:

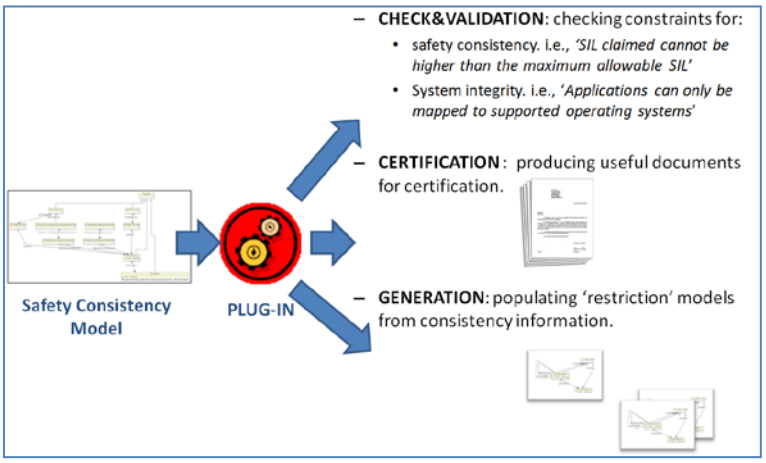


Figure 40: Safety Consistency Check, Useful Documents for Certification and Generation of Restrictions.

3.5.1.3 Model-to-model transformations

These transformations include pat of the intelligence of the toolset. The most relevant model-to-model transformations are:

- **Input models to deployment model**: This deployment model will contain the information of the resulting partitioning schema. It is the foundation for the integration of the partitioning tool with external tools (e.g. safety analysis, real-time analysis…). It defines the resulting partitions, where each partition has a list of allocated applications, required hardware, required operating system, etc.
- **Safety consistency model to partitioning constraint model**: The tool pre-processes safety information and defines restrictions to the resulting partitioning, based on a comparison between safety information of each compliant item. For instance, an application must be or must not be allocated on a specific partition based on their respective safety integrity levels.

3.5.1.4 Model-to-text generators

There is a number of model-to-text generators, some of them are:

- **Deployment model to XtratuM configuration files**: Based on the deployment model, it generates the XML file required to configure XtratuM.
- **Deployment model to makefiles**: It generates the required files (makefiles, scripts, etc.) for building the **whole** system.
- **Safety Consistency Model to HTML code**: It generates a useful document for safety certification that collects all information of safety models.
- **Etc**.

The next sections describe safety consistency and checking related models. For the sake of simplicity, entities of the model are not described in details as they can be found in MPT deliverables.

3.5.2 Safety Consistency Model

The different components of the system can be referred as Safety Compliant Items (SCIs) and this model contains the properties related to safety and IEC-61508 specific concepts for each system component. The SCIs can be defined as system, platform, partition or node. Furthermore, user can assign to each application its IEC-61508 safety integrity level. Figure 41 shows the Ecore diagram of the models.

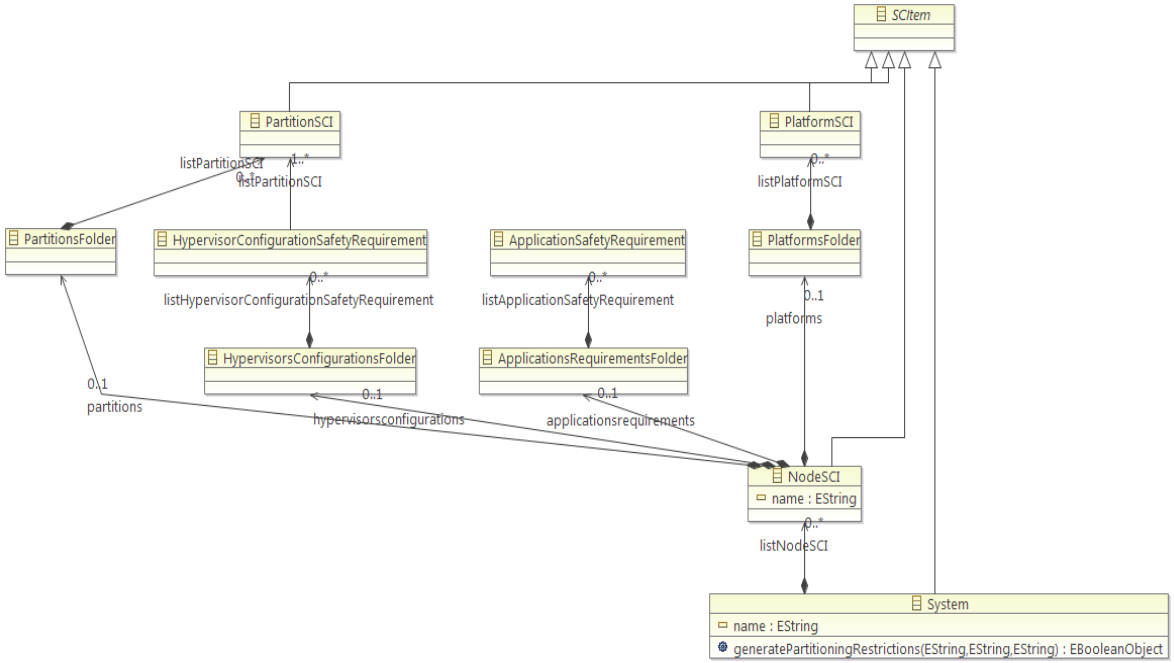


Figure 41: Safety Consistency Model

This model is basically related to 2.1.1 Safety Requirement Specification and 2.2.1 Safety Architecture Specification, at system and element level (3.1.1 and 3.2.1), of the V-shape model as it enable to specify safety requirements (modelled as a Safety Compliance Model described in the next section) for any SCI item (safety compliant item) as system, platform, hypervisor, partition and application node.

3.5.3 Safety Compliance Model

This model provides IEC-61508 related safety considerations encapsulated into a Safety Manual to each SCI. This model, linked to previous model, is used basically to represent safety requirements, fault management and faults hypothesis information.

- **Management of Functional Safety (FSM**). It is a collection of the supported safety integrity levels by SCI.
- **Faults Management**. It contains information such as the diagnostic coverage level, hardware fault tolerance level (HFT), and a collection of diagnostic techniques and measures employed to achieve the safety integrity level assigned in the FSM.
- **Hypothesis value and hypothesis range**. Collections that specify assumptions about the types of faults, the rate at which components fail and how components may fail.

Figure 42 shows the Ecore diagram of the models.

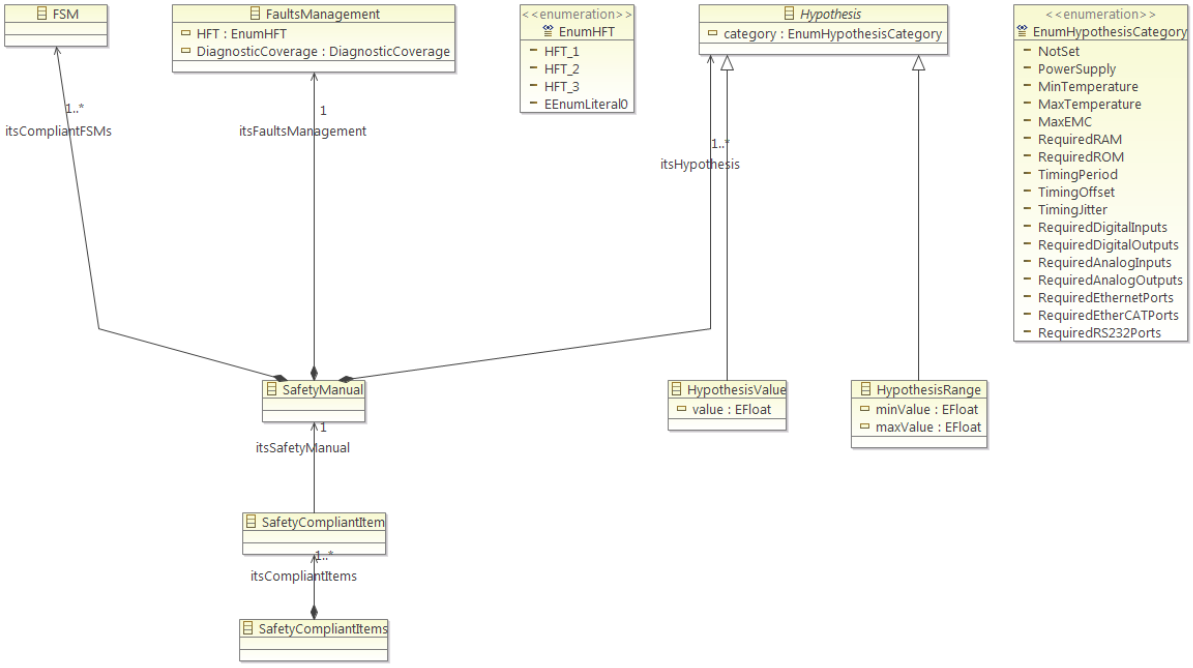


Figure 42: Safety Compliant Item and Safety Manual Models

3.5.4 Diagnostics Techniques and Measure Model

This model represents the diagnostic techniques and measures defined in IEC-61508 standard to avoid and control failures during operation. This model, which is linked to the previous models, is used to represent the requirements of systems and SCI items nodes about the standard IEC-61508.

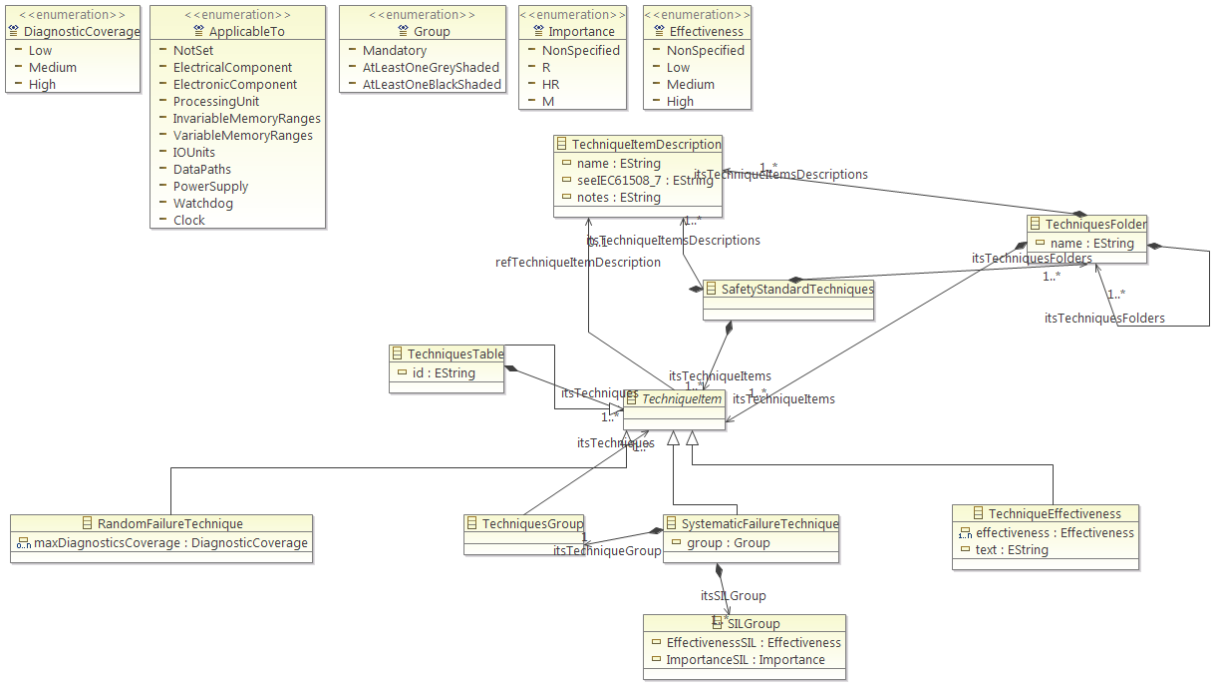


Figure 43: Diagnostic Techniques and Measure Model.

To help understanding this model, Figure 44 shows an instance of the model for IEC-61508 standard.

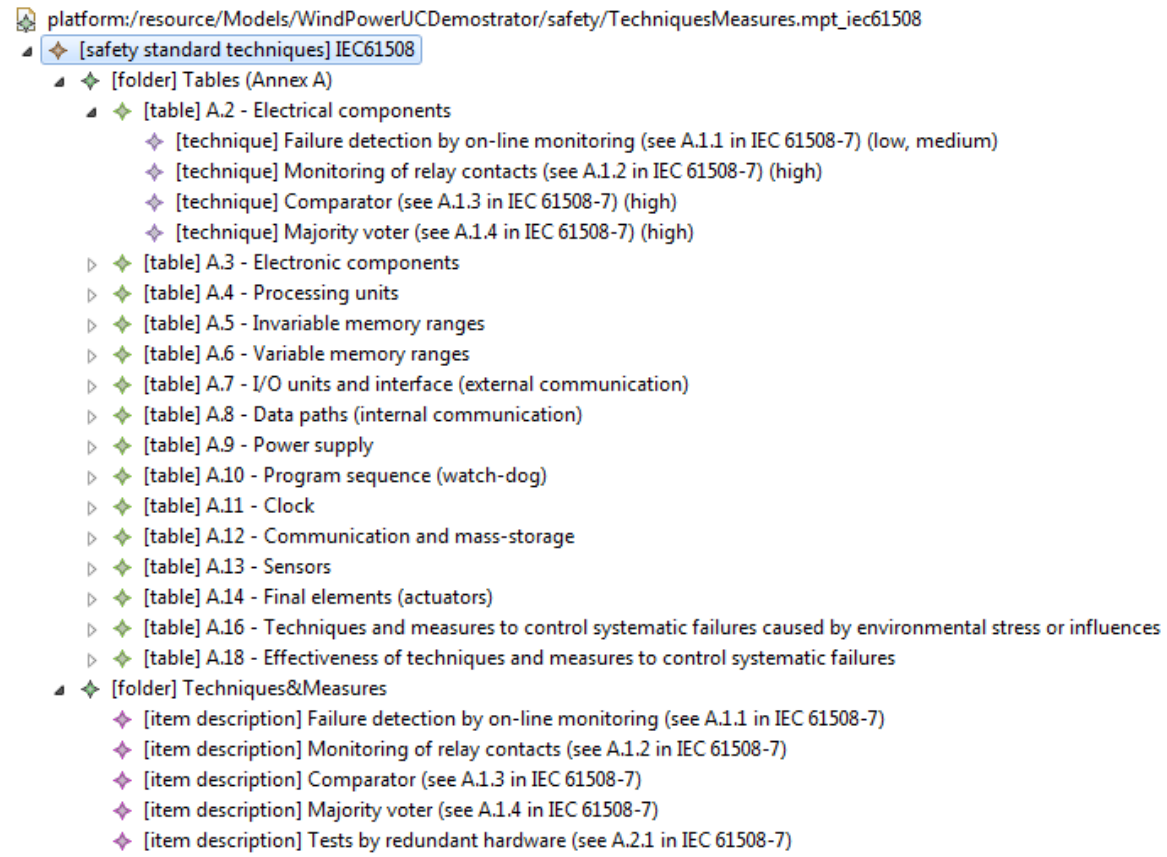


Figure 44: Instance of the Diagnostic Techniques and Measure Model for IEC-61508.

3.5.5 Safety Integrity Levels Model

Finally, this model is used to represent just an enumeration of the integrity levels of safety standards (i.e. SIL, ASIL). In the next section, the model-to-model transformations are introduced which ground the development process used in DREAMS.

3.6 Model-to-model Transformations

3.6.1 Generic DREAMS Development Process

Figure 45 provides a coarse overview of the development workflow suggested to implement software for DREAMS-based systems, and to deploy it to the platform. The focus of this description is on the specification and implementation phase of the development process ("left branch" of V-shape model).

The numbers in the yellow boxes indicate the phases of the DREAMS development process depicted in Figure 24[5]. An exemplary version of the workflow that details the use of concrete technologies implementing the DREAMS architectural style can be found in Section 3.6.2.

As sketched in the figure, the workflow involves a number of frontend tools (to be operated by the different stakeholders of the development process) as well as internal tools (e.g., analyses, optimizations, transformations, and generators) that form the backend of the workflow.

[5] We append "(SW)" to phase specification to indicate that the step does not involve hardware development. Two-digit phase specifications (e.g., 2.3 (SW)) express that the step is not specific to one of the three aspects safety, security, and timing. However, the step may (but not necessarily needs to) support one or more of these aspects.

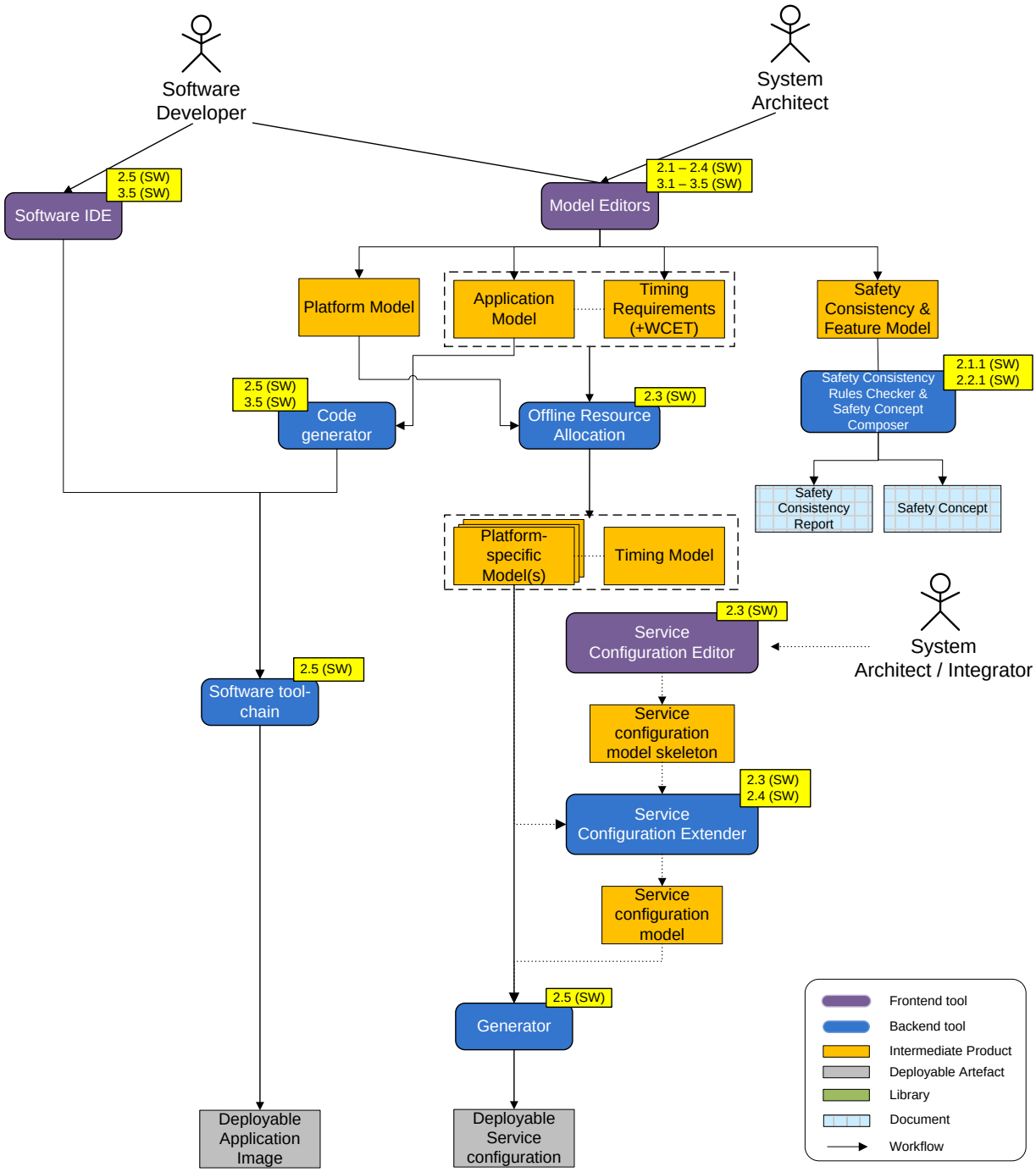


Figure 45: Overview of DREAMS software development & deployment workflow

The workflow implements a chain of transformations from the input models via a set of intermediate models and textual artefacts to the final artefacts to be deployed to the target platform (e.g., software images, configuration for devices and software services).

3.6.1.1 System Architecture Development Process

In the first step, the *system architect* uses modelling tools to create a model of the overall system. It covers the following aspects:

- Application architecture (with annotations).
  - Desired temporal behaviour.
  - Criticality domains.

- Platform architecture: structure of target platform.
    - Clusters, nodes, tiles, networks, etc.
    - System software (e.g., partitions to be provided by hypervisor).
- Safety Consistency & Feature Model: Safety Consistency Model and Variability Model of the Safety Consistency Model.
    - Safety Manuals for SCI Nodes, Platform, Hypervisor, Partitions, Applications, etc.
    - Variability with features of the Safety Consistency Model.

This step covers phases 2.1-3 (SW) of the DREAMS development process (see Figure 24). As pointed out in Section 3.3.3, abstract models of the HW/SW platforms are used to support the software/platform integration step in phase 2.3.

3.6.1.2 Software Development Process

The actual functionality of the different application subsystem is provided by *software developers* who implement software components to be integrated into the overall system as pointed out below. On the one hand, the software developer can follow a fully model-driven software development approach and specify the functionality (and the desired temporal behaviour) using models that refine the application architecture specified by the system architect (corresponding to phase 2.1-4 (SW) of the development process). In this case, a code generator is used to produce the source code of the application (phase 2.5 (SW)).

On the other hand, the software developer can also implement components directly in a programming language supported by the integration platform (e.g., C) (phase 2.5 (SW)). In this case, application subsystems are provided as black box components with regard to the system model which is still used to support the integration phase. Although not shown explicitly in the figure, the software developer may also use tools to support the preceding phases of the development process (i.e., 2.1-4 (SW)).

In either case the resulting source code is compiled (phase 2.5 (SW)) and further transformed into deployable images supported by the software integration platform (see below for details). Also, both software development approaches may be used in a development process at the element level (corresponding to phases 3.1-5 (SW)).

3.6.1.3 Safety Management

3.6.1.3.1. Safety Consistency Rules

Safety consistency are checked by the tool by means of the consistency rules that are enforced by the toolset by means of validation constraints defined in EOL (Epsilon Validation Language http://www.eclipse.org/epsilon/doc/evl/ by Epsilon Research group at York University) that is a validation language developed on top of EOL. Some examples of the rules implemented in the toolset include:

- **Rule 1-61508: SIL claimed cannot be higher than the maximum allowable SIL.** SIL level claimed to a Safety Compliant Item cannot be higher than the allowable SIL value calculated depending on the diagnostics used for compliant item.
- **Rule 2-61508: Safety certification standard supported by any 'compliant item' must be compliant with the system certification standard.** All the components of a system must be compliant with the certification standard of the system. If any component is not compliant with this standard, the system will not be considered consistent. If any component is not

compliant with system's standard, but supports a derived standard it will be considered consistent, but a warning message will be shown.

- **Rule 3-61508: FSM used in the development of any 'compliant item' must be compliant with the system FSM.** FSM for each component of a system must be compliant with FSM for the System.
- **Rule 4-61508: SIL level required for the Application is provided by the Platform.** The platform where an application is deployed on must provide the level of integrity required by the application.

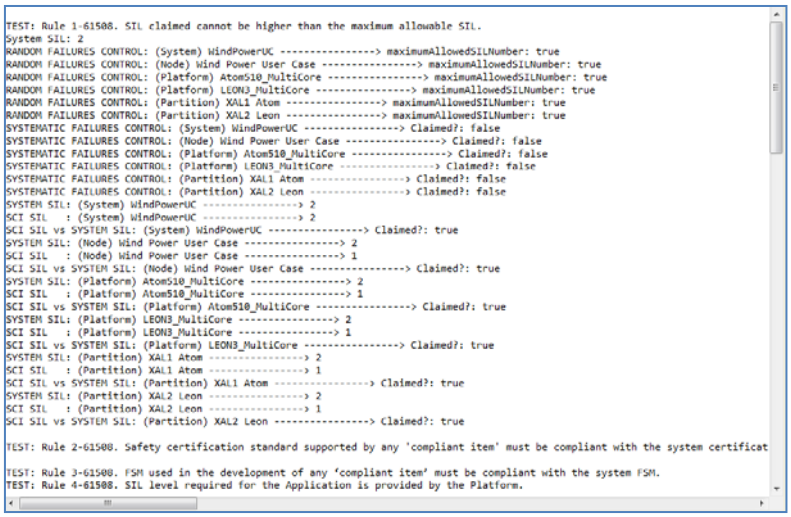


Figure 46: Checking process and diagnostic techniques information panel.

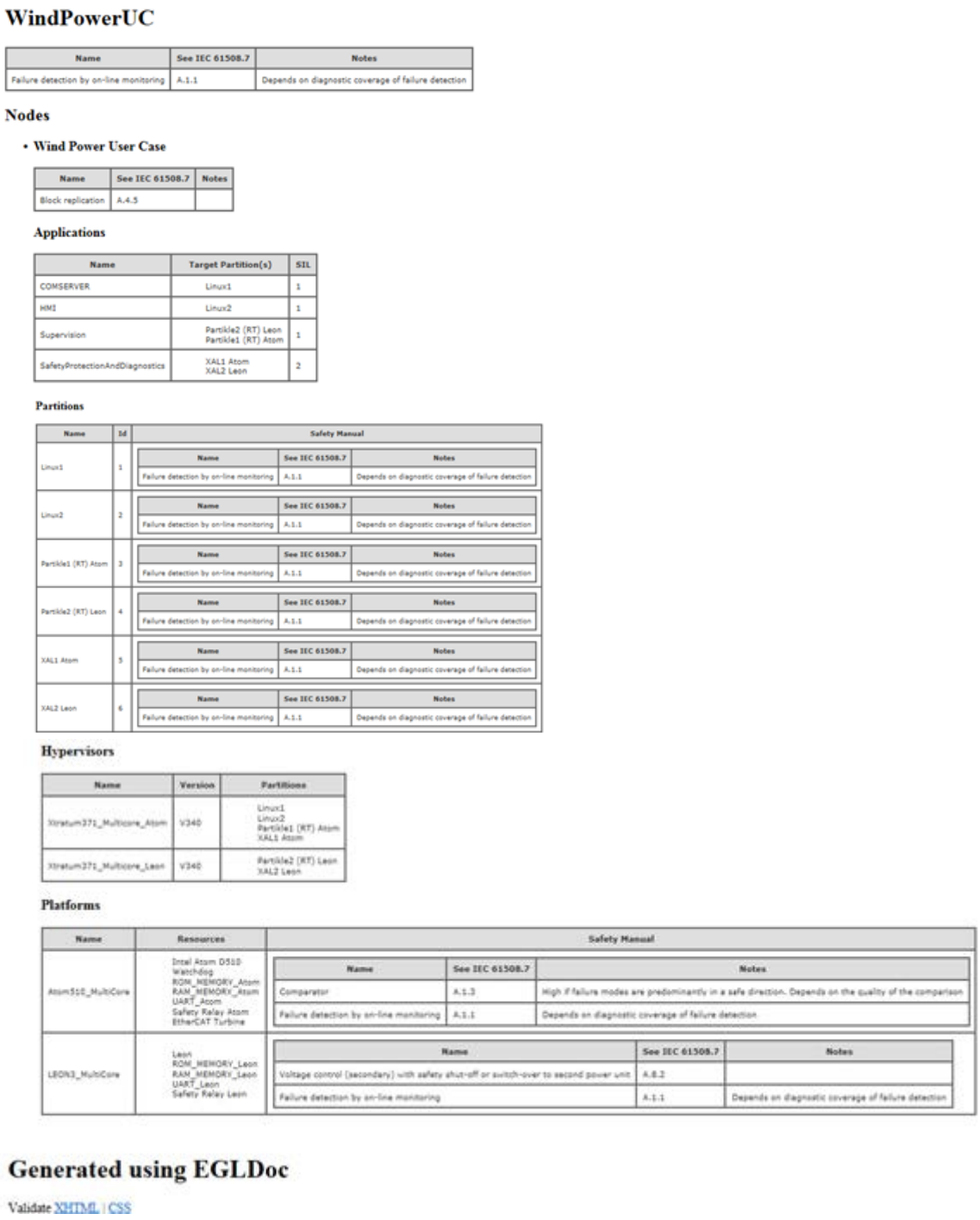
Under user requirement EVL constraints are triggered and checked, and a summary of constraint failures is given by the system to alert the user. In addition to this, additional information to know details of the checking process (Figure 46) and the diagnostics techniques that appear in the safety manuals are shown for each SCI component. This information may be useful for certification purposes.

3.6.1.3.2. Generation of Documentation for Certification

As mentioned before the toolset generates documents useful for certification. The functionality is developed in Epsilon EGL language and is based on a set of templates that generate a set of html document useful for certification. The main templates are:

- Template for SCI Nodes (Safety Compliant Item) – For each safety item:
  - Documents the safety manual
  - Documents Application Safety Requirement
  - Documents the Partition Safety Manual
  - Documents the Hypervisor configuration Safety Requirement
  - Documents the Platform
- Template for Safety Manual of each SCI item
  - Documents the Standard an SIL level
  - Documents the Diagnostic Technique used
  - Etc.

Figure 47 shows an example of Certification Document generated for the Wind Power use case.



WindPowerUC

Name	See IEC 61508.7	Notes
Failure detection by on-line monitoring	A.1.1	Depends on diagnostic coverage of failure detection

Nodes

- **Wind Power User Case**

Name	See IEC 61508.7	Notes
Block replication	A.4.5	

Applications

Name	Target Partition(s)	SIL
COMSERVER	Linux1	1
HMI	Linux2	1
Supervision	Partikle2 (RT) Leon Partikle1 (RT) Atom	1
SafetyProtectionAndDiagnostics	XAL1 Atom XAL2 Leon	2

Partitions

Name	Id	Safety Manual
Linux1	1	Name: Failure detection by on-line monitoring; See IEC 61508.7: A.1.1; Notes: Depends on diagnostic coverage of failure detection
Linux2	2	Name: Failure detection by on-line monitoring; See IEC 61508.7: A.1.1; Notes: Depends on diagnostic coverage of failure detection
Partikle1 (RT) Atom	3	Name: Failure detection by on-line monitoring; See IEC 61508.7: A.1.1; Notes: Depends on diagnostic coverage of failure detection
Partikle2 (RT) Leon	4	Name: Failure detection by on-line monitoring; See IEC 61508.7: A.1.1; Notes: Depends on diagnostic coverage of failure detection
XAL1 Atom	5	Name: Failure detection by on-line monitoring; See IEC 61508.7: A.1.1; Notes: Depends on diagnostic coverage of failure detection
XAL2 Leon	6	Name: Failure detection by on-line monitoring; See IEC 61508.7: A.1.1; Notes: Depends on diagnostic coverage of failure detection

Hypervisors

Name	Version	Partitions
Xtratum371_Multicore_Atom	V340	Linux1 Linux2 Partikle1 (RT) Atom XAL1 Atom
Xtratum371_Multicore_Leon	V340	Partikle2 (RT) Leon XAL2 Leon

Platforms

Name	Resources	Safety Manual
Atom510_MultiCore	Intel Atom D510 Watchdog ROM_MEMORY_Atom RAM_MEMORY_Atom UART_Atom Safety Relay Atom EtherCAT Turbine	Name: Comparator; See IEC 61508.7: A.1.3; Notes: High if failure modes are predominantly in a safe direction. Depends on the quality of the comparison Name: Failure detection by on-line monitoring; See IEC 61508.7: A.1.1; Notes: Depends on diagnostic coverage of failure detection
LEON3_MultiCore	Leon ROM_MEMORY_Leon RAM_MEMORY_Leon UART_Leon Safety Relay Leon	Name: Voltage control (secondary) with safety shut-off or switch-over to second power unit; See IEC 61508.7: A.8.2; Notes: Name: Failure detection by on-line monitoring; See IEC 61508.7: A.1.1; Notes: Depends on diagnostic coverage of failure detection

Generated using EGLDoc

Validate XHTML | CSS

Figure 47: Example of Wind Power certification document.

3.6.1.4 Offline Resource Allocation

The key property of mixed-criticality systems is the integration of different application subsystems of different criticality levels into a single target system. In the DREAMS architecture, both design-time and runtime methods are used to support this integration task. Figure 45 depicts the role and workflow of the *system integrator* who integrates the components provided by application developers into the overall system model designed by the system architect. This step corresponds to phase 2.3 (SW) and involves the following main steps:

- Mapping of tasks to execution containers (i.e., partitions).
- Computation of offline task and message schedules.

- Configuration of online adaptation strategies (e.g., pre-computed static schedules for different system modes).

In these tasks, the system integrator is supported by the offline resource allocation algorithms provided by WP4, potentially based on a partial manual integration (e.g., task mapping according to application-specific requirements). The result of this offline integration step is stored in the platform-specific model (PSM) of the application that is obtained from the initial application and platform model using a model-transformation. The PSM refines platform-independent model artefacts from the application model (e.g., messages between components) into platform-specific constructs (e.g., instantiation of specific communication channels depending on the mapping of the communication endpoints, such as inter-partition communication provided by the hypervisor, as well as on-chip and off-chip communication). Furthermore, the PSM contains detailed information about the deployed application (e.g., it serves as a container for the computed mapping information and schedules).

In the subsequent steps of the workflow, the PSM serves as input model for the generation of platform-specific code and platform configuration data:

- The PSM can be used to directly generate configuration data for specific services of the DREAMS platform such as the global and local resource managers (phase 2.5 (SW)).
- Alternatively, the PSM can be merged using appropriate model-transformations ("configuration extender" in Figure 45) with skeleton configurations created using the vendor-specific configuration tools. This approach corresponds to phases 2.3 (SW) / 2.4 (SW) and ensures the creation of valid vendor-specific configuration files for the different technical domains of the target platform that are augmented with the results obtained during the offline resource allocation step. Here, the system model (and the PSM derived from it) provides a holistic view onto the overall DREAMS system and ensures the consistence of the overall configuration.

The offline resource allocation and exploration step is illustrated in Figure 48 in more detail, which presents the involved processing chain (coloured ovals in the figure, from left to right). It provides two entry points that starts at a different level of abstraction. The basic workflow involves the following steps:

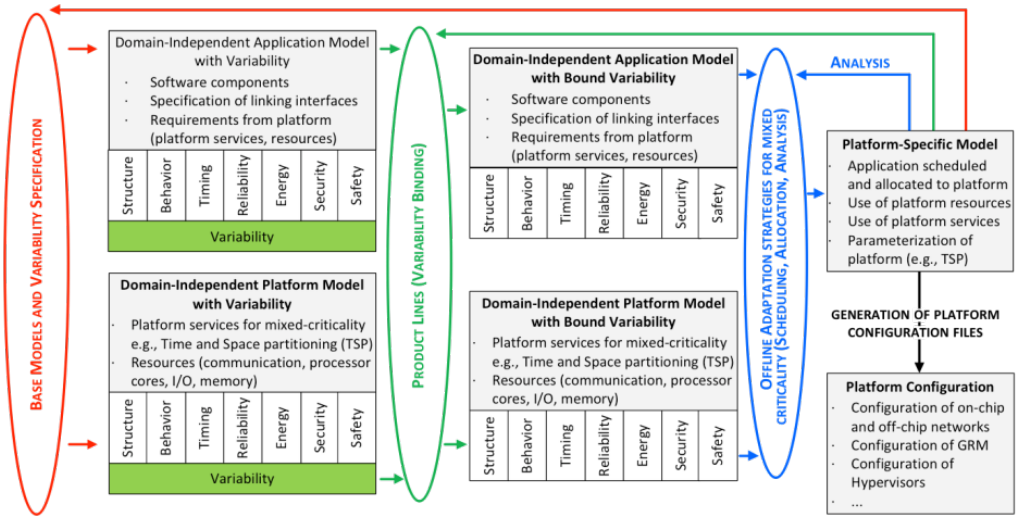


Figure 48: Overview of Offline Resource Allocation and Exploration Process

- *Offline adaptation strategies for mixed criticality systems,* (blue oval) serve as the entry point to the development process. These methods are used to compute a deployment of an application to an instance of the DREAMS platform and also take into account the DREAMS

online resource management strategies. This step uses models of the application subsystems and a platform model as input. The result of this process is a platform-specific model that contains information about the deployed application.

- In the figure, the blue backward arrow ("analysis") indicates that this step is an optimization process where different deployment alternatives are explored and that it provides analysis methods to rate the eligibility or quality of a particular solution, such as analyses of different timing properties, or the energy consumption of the NoC. It also includes an analysis to estimate the reliability, which is used to explore different fault-tolerant versions of a given design (based on active redundancy)2. Note that several parts of the solution analysis require HW-specific models that allow an estimation how the platform behaves given a certain configuration. For the examples given above this can be the execution time of instructions for the timing properties and the energy demand of the instructions and the peripherals for the energy consumption.

- In the next step (bottom right in the figure), the platform-specific model is used as input for the backend of the processing chain that *generates configuration files* for the different HW/SW components of the target platform.

3.6.1.5 Product-Line Design

3.6.1.5.1. Product-Line Exploration Process

Based on the above basic workflow that handles the configuration of a single system, the following extended development process can be defined which considers entire product-line families.

- In this case, the process starts with the definition of base models and variability specification (red oval in Figure 48). Here, a system model consisting of an application model and a platform model (see above) are used as base models. Additionally, the system designer provides a (separate) variability specification that defines which parts of the base model can be varied. Hence, the base model serves as template which is augmented with appropriate variation points.
- Together, both models span an entire product-line family from which particular members can be selected. In the figure, this selection step is designated as variability binding (green oval), since for all variation points, a concrete choice is made. The result of this process is a system model that can be further processed using the basic workflow pointed out above.
- The green and red backward arrows in the figure indicate that also in this workflow, the eligibility and quality of the deployed system is rated. In case the selected solution does not satisfy all requirements, the following two options exist: At first, a different variability binding is selected (indicated by the green backward arrow), i.e. a different member of the product-line family is used as input for the basic workflow. In case the last step was not successful, the designer changes the definition of the product-line by modifying the base model and/or the variability specification (red arrow).

3.6.1.5.2. Variability and its impact in Safety and Certification

Finally, one of the aims in DREAMS is to study the possibility of adding software product line techniques to some of the tools to be developed in the project.

MPT did not address variability concepts. The proposal here in DREAMS is to use CVL (Common Variability Language) Variation Points language developed by SINTEF to represent variability (in a separate model) for Safety Consistency Model.

The definition of how variability affects Safety Consistency Model and its impact in the other models will depend on the analysis of variability needs for the demonstrators use cases.

3.6.2 Exemplary Development Process using Concrete Technologies

In Section 3.6.1 , a generic development process for DREAMS-based systems has been introduced. In this section, a concrete instantiation of this process will be presented that shows the integration of tools for provided by DREAMS project partners and which will be applied to develop the application demonstrators. Figure 49 illustrates this instantiation of the DREAMS development workflow that follows the generic approach shown in Figure 45.

In the following, the roles of the tools for the system architecture as well as the software development process will be summarized.

- AutoFOCUS3: The system architect uses “AutoFOCUS3” to create models of the application and the instance of the DREAMS platform. As pointed out in Section 3.3, AutoFOCUS3 can be used to support the specification of software requirements, software architecture and design specification as well as the software development phase of the development process (2.1-2.5 (SW), and 3.1-3.5 (SW)) if the tool is used to define and implement software modules). The software/platform integration step in phase 2.3 supports modelling the HW/SW platforms (i.e. the SW deployment and configuration (number of devices, inter-connections, etc.) of existing HW blocks). Furthermore, AutoFOCUS3 contains modules to support several phases on the right-hand side of the development process depicted in Figure 24 at system and element level such as formal verification, simulation, synthesis of test-cases and test-code (not shown in Figure 49).
- RTaW Timing tools: The system architect uses the “RTaW Timing” tools to describe the desired timing properties of the system. This toolset cover phases 2.1.3, 2.2.3, 2.3.3, 2.4 and 3.1.3, 3.2.3, 3.3.3 of the design branch, and phases 2.7.3, 2.8.3 and 3.7.3, 3.8.3 of the testing branch (see Figure 24).

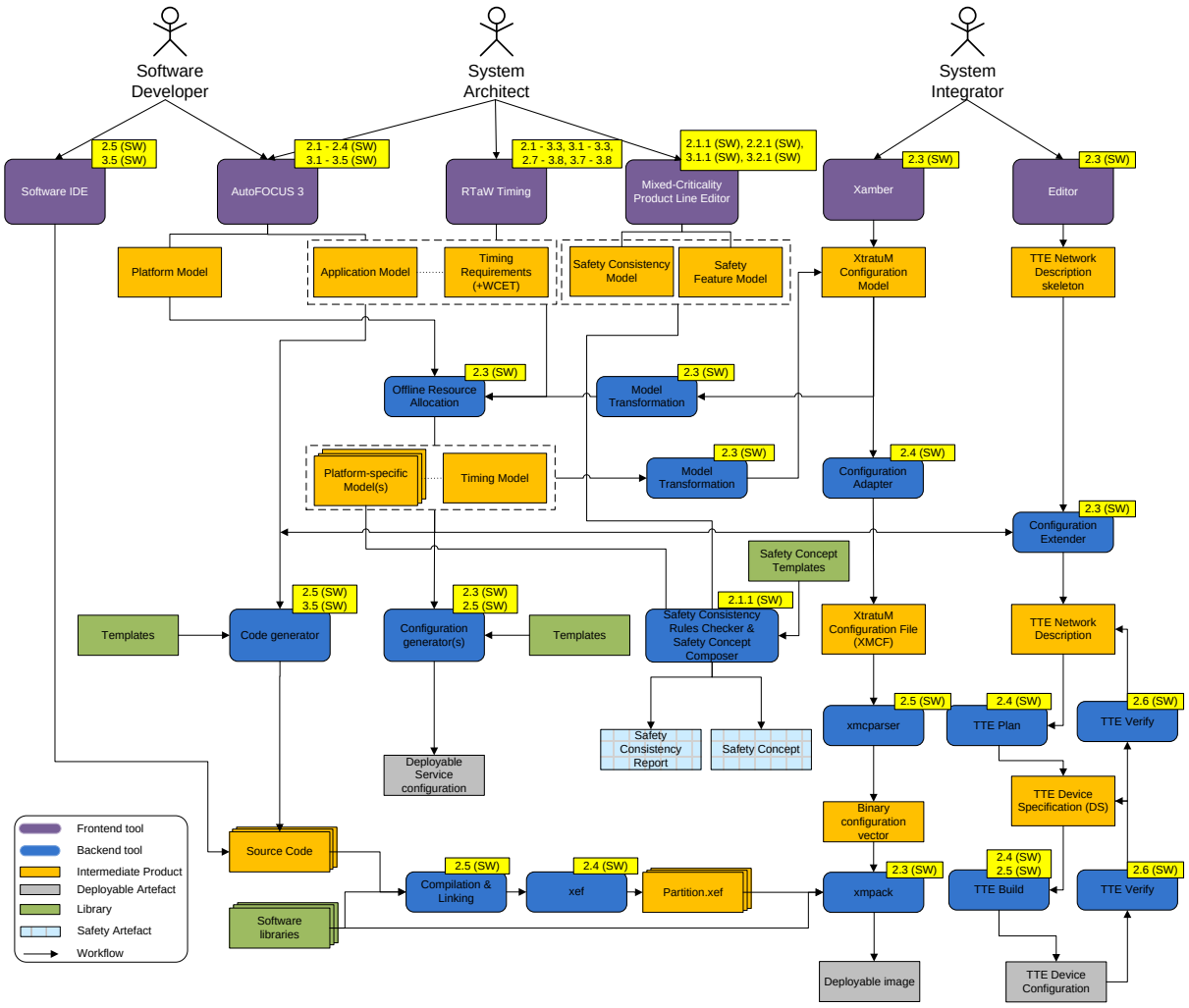


Figure 49: DREAMS software development workflow: Tools, transformations and implementation artefacts.

The DREAMS architectural style suggests the use of hypervisors and off-chip communication networks in order to enable the efficient implementation of mixed-criticality systems. Since these elements are hidden in the blocks *Offline Resource Allocation, Service Configuration Editor* and *Platform Specific Model(s)* in the generic workflow shown in Figure 45 , but are an essential part of the approach, a more detailed presentation using the XtratuM hypervisor and the TTEthernet off-chip network as examples for these technologies are provided in Figure 49 .

Here, the XtratuM hypervisor and the TTEthernet off-chip network have been selected as examples for these technologies. More details on the configuration of these components using technology-dependent tools provided by the respective vendors will be presented in Sections 3.6.2.1 and 3.6.2.2, respectively.

- For the configuration of the XtratuM hypervisor, the *system integrator* creates an “XtratuM Configuration Model” (phase 2.3 (SW)) that is used as additional input to the offline resource allocation step. The offline resource allocation module required for the configuration of the virtualization services of the XtratuM hypervisor is provided by “Xamber” (phase 2.3 (SW)). In the next step, these results are used to obtain an updated “XtratuM Configuration Model” that contains the information computed during the offline resource allocation. Thereafter, this model is transformed into a “XtratuM Configuration File” (phase 2.4 (SW)) which in turn is transformed into a deployable binary hypervisor configuration vector in phase 2.5 (SW) (see Section for details).

- The figure also shows how the configuration vector for the hypervisor service is combined with the executable images of the application subsystems in order to obtain deployable images. Here, the source code for the application subsystems (which has either been crafted manually using a "Software IDE" or that has been generated from the application model) is compiled, linked (phase 2.5 (SW)) against the required libraries, and transformed into the partition image format (phase 2.4 (SW)). In the module integration phase, for each hypervisor instance the required partition images are combined with the corresponding "Binary Configuration vector" into a deployable image (phase 2.3 (SW)).
- For the configuration of the TTEthernet off-chip communication, the *system integrator* creates an initial "TTE Network Description" skeleton (phase 2.3 (SW)) that is extended with the information computed in the offline resource allocation step (phase 2.3 (SW)). After that, the resulting "TTE Network Description" model is transformed (phases 2.4 (SW), 2.5 (SW) and verified (phase 2.6 (SW)) by the TTEthernet tool-chain using the workflow described in Section 3.6.2.2.

Finally, Figure 49 also details the integration of the safety and variability management toolsets into the workflow.

- To represent the Safety Consistency Model and the Variability of the Safety Consistency Model the System Architect defines a model in which safety requirements are defined for System, SCI Nodes, Platform, Hypervisor, Partitions and Applications with its features. As described in previous sections, these are separate models from the base models.
- In the next step, the Safety Consistency Rules Checker and the Safety Concept Composer can be called to check consistency rules for specific platform and applications and to produce documents useful for certification according to safety concept templates.

3.6.2.1 *Configuration of XtratuM hypervisor*

Virtualization is the enabling technology to run simultaneously into the same hardware independent subsystems with strong isolation and mixing different criticality levels. As shown in Figure 49 , the Xtratum hypervisor is configured during some phases of the development process, phases 2.1 to 2.5 (SW), where the Xtratum configuration file will be configured. The configured Xtratum configuration file will be tested during the rising branch of the development process, phases (2.6 to 2.9).

XtratuM is a "Type 1" hypervisor intended to embed system applications. It provides a para-virtualization interface and mainly supports the following features:

- Temporal isolation: fixed scheduling of CPU time.
- Spatial isolation: fixed assignment of physical memory and device resources.
- Static resource assignment: The system designer assigns the platform resources (time, memory and devices) to partitions.
- Inter-partition communication mechanism: Supporting "sampling" and "queuing" channels.
- Heath monitoring mechanism.
- System manager: Privileged (and trusted) "system partition" can monitor and manage other partitions

XtratuM is available for a limited variety of platforms. From scratch, it must be ported in order to support specific underlying architecture mechanisms. In the DREAMS context, ARM, PowerPC and x86 will be supported.

Once having a specific porting of XtratuM, during the development process, it must be configured according to the physically available platform components and, on the other hand, according to the application requirements. In order to deploy an XtratuM-based application, the provided SDK includes tools to generate a container packing the following components:

- XtratuM hypervisor image
- Configuration information
- Full binary code for all partitions

This section focus on the "Configuration information" that in fact is a model including information about a particular view of the system. Therefore, the configuration information can be seen as the result of a deployment-specific transformation of different DREAMS models: platform, requirements, and application (see Figure 50).

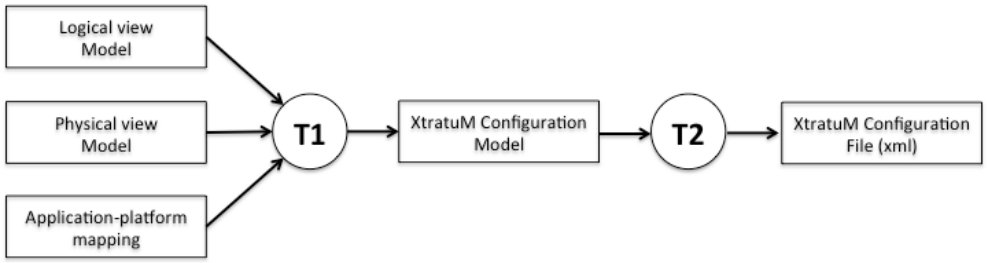


Figure 50: Context of XtratuM configuration model.

The XtratuM SDK packing tool stands for an XML configuration file matching all architectural and application restrictions of a specific deployment. This file includes detailed information about:

- Hardware description
- Hypervisor properties
- Partition Table: List of partitions
- Inter-partition communication channels: List of channels.

The syntax of this file is narrowed by the chosen XtratuM and must follow a deployment-specific model. This model must follow an XML schema specification automatically generated and provided together with each XtratuM compilation. This mechanism allows the very early detection of configuration errors during XtratuM deployment packing.

In the DREAMS context, this XML schema meta-model is not general enough because it assumes a variety of implicit restrictions. In fact, due the variety of platforms and configurations expected in DREAMS we should manage a family of valid schemas. The mandatory generalization of XtratuM configuration leads to the concept of a general "Configuration Model" which includes model restrictions explicitly together with the configuration information. From this model, a correct XtratuM Configuration File can be generated by model transformations (see "T2" in Figure 50).

The "XtratuM Configuration General Model" (XCM) has two parts: the first one including restrictions about underlying hardware and hypervisor internals and the second one including non-restricted XtratuM configuration fields that nevertheless must fit the restrictions declared in the first part of the model (see Figure 48). This provides the required flexibility to generate, by model transformations, configuration files for all supported platform-XtratuM variations.

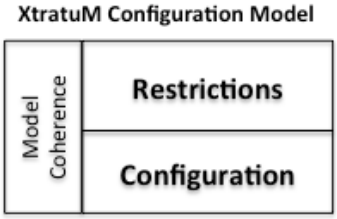


Figure 51: Parts of XtratuM configuration model.

In order to manage the creation and manipulation of XtratuM configurations, a Java-based helping tool "Xamber" provides a graphical user interface to edit models and restrictions. From a practical point of view, Xamber handles XCM files and, among other manipulation features, includes import/export options to generate (or read) specific "XtratuM Configuration Files" as well as platform specifications.

All models to be manipulated with Xamber must follow a meta-model written in XML Schema syntax. Internally, Java/XML bindings are implemented using JAXB facilities. In this sense, it is desirable that all models interfacing Xamber could have a meta-model specified as an XML Schema file.

3.6.2.2 *Configuration of TTEthernet off-chip network*

The generation of time-triggered schedules for TTEthernet (TT-network-schedules) as well as configuration artefacts for TTEthernet nodes is carried out by the TTE-tool-chain. As shown in Figure 49 , the TTEthernet off-chip network is configured during some phases of the development process, phases 2.1 to 2.5 (SW), where the TTEthernet off-chip network will be configured. The configured TTEthernet off-chip network will be tested during the rising branch of the development process, phases (2.6 to 2.9).

Figure 52 and Figure 53 show the tool workflow and the related dataflow on a typical tool-chain run respectively. The connection of the different tools is made via well-defined interfaces, in form of XML files. This provides a human readable format of the input as well as generated data and enables the manipulation of these files manually or via automated tools.

TTE-Plan is used to generate network configurations for TTEthernet systems. It helps the user to model the topology of a TTEthernet network, and to generate a communication schedule for that network (i.e. TT-network-schedule).

TTE-Plan is a command-line tool that takes a network description XML file as input and generates a network configuration, which consists of a main <*.network_config*> file and additional files referenced by this file. In the network description file, the user configures aspects of the network such as topology, virtual links, and synchronization parameters. The network configuration file can be used to generate device configuration *HEX* files for each device in the network using TTE-Build, and these file s can be downloaded to the switches using TTE-Load, and to the end systems user application. When the network is in operation, the data traffic can be viewed using either an oscilloscope or TTE-View, a *Wireshark* distribution with a dissector plug-in for TTEthernet frames.

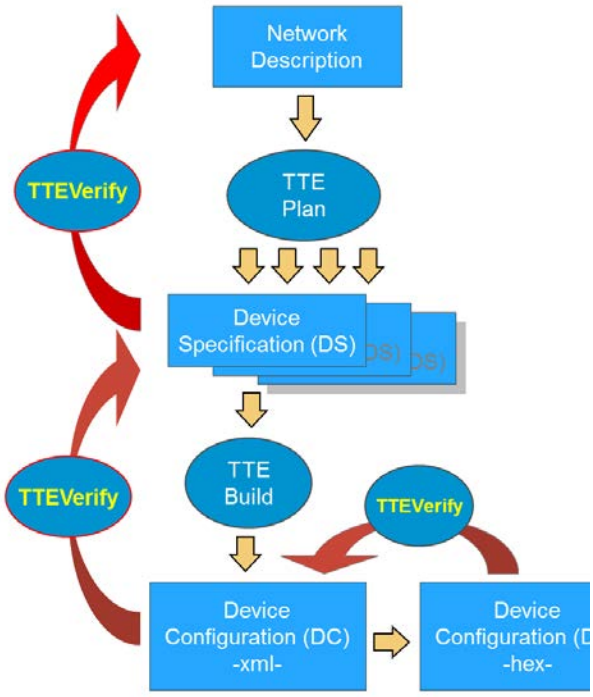


Figure 52: TTE tool-chain workflow

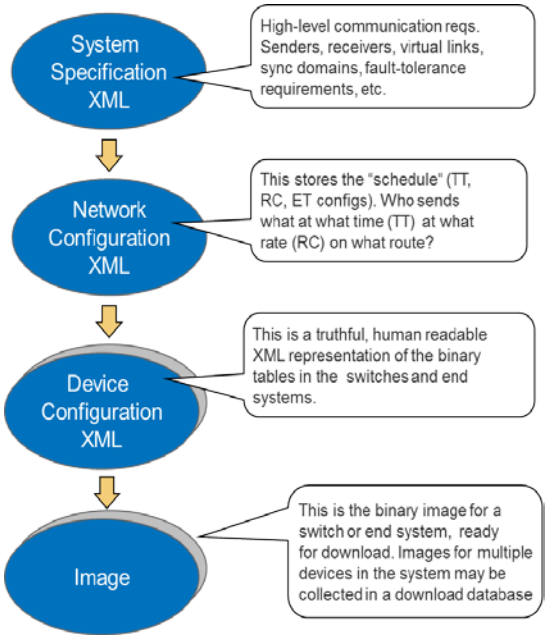


Figure 53: TTE tool-chain dataflow

An overview of this tool chain showing input and output files is shown in Figure 53. For the configuration of a TTEthernet network, TTE-Plan expects a network description XML file corresponding to the network topology and preferences. This file is used as input to TTE-Plan so it can generate the corresponding network configuration files, which constitute the input to TTE-Build and the following tools.

A complete network description contains several sections specifying the necessary TTEthernet preferences. Namely,

- timing parameters, including redundancy, global speed, and network periods;
- network synchronization configuration;
- network topology, specifying switches, end systems and their physical connections;
- time-triggered and rate-constrained virtual links, as well as best-effort traffic;
- frame buffers;
- specific constraints and scheduling guidance parameters.

This concludes section 3.6.2 which introduces one exemplary instantiation of the generic development process that has been shown in the section before. Thereby, the general presentation of the meta-models which will be used in the DREAMS project is finalized and a more detailed version with higher granular models follows in the deliverables D1.4.1 and D1.6.1.

4 Requirements Matrix

Requirement	Ref. to Section	Comment
R 1.1.1	2.1.1.3, 2.3.2.4	Assurance of value and timing in case of integration into a larger system.
R 1.2.2	2.3.2.3	Safety critical subsystem shall see sequences of critical event in a given order
R 1.6.1	2.1.1.3	Scalability (Multi-core platforms, mixed-critical, etc…)
R 1.8.1	2.1.1.1	Meet-in-the-middle
R 1.10.1	2.1.1.3	Different models of computation support
R 1.10.2	2.1.1.3	Communication between components
R 1.10.3	2.1.1.3, 2.3.2.3	Architectural style and development methodology
R 2.7.1	2.1.1.3	The on-chip network shall provide different interaction mechanisms required for different models of computation such as Time-Triggered, rate-constrained, best-effort communication and shared memory access.
R 2.7.2	2.1.1.3	The architecture shall provide support of different processors and/or hardware accelerators with shared memory access.
R 4.1.3	2.3.2.1	End-to-end response time analysis algorithm shall be developed which are able to account for scheduling algorithms considered by the resource allocation strategy.
R 4.2.1	2.1.1.3,3, 3.5	Variability modelling and analysis tools shall be enhanced to achieve by automatic means as well as guided manual means an optimal or best effort configuration of DREAMS platforms and DREAMS systems.
R 4.4.1	3.6	Design activities of the DREAMS development process shall be supported by a tool chain.
R 4.4.2	3.6	The exchange of data between consecutive tools in the DREAMS development process shall be automated so that it can be performed without “manual” recopying or reworking of the data.
R 4.4.3	2.3.2.3	Methods and tool should at least be suitable for all application domains represented by the demonstrators.
R 4.5.1	3.6	The generation of the configuration files of the DREAMS platform for an instance of the DREAMS architecture, shall be supported by tools that use the system model as input.
R 5.1.1	2.1.1.3, 3, 3.5	Mixed-criticality product line shall be supported to enable certification of product-lines with variability management.
R 6.1.2	2.3.2.1	The end-to-end communication time between two tasks of an application subsystem on the DREAMS architecture shall be less than 50ms.

R 6.1.3	2.3.2.1	The DREAMS architecture shall ensure that it is possible to design a solution where two subsystems can be connected in which the time between a data generation from one application subsystem and the time the data has been delivered to another application subsystem is less than 1s. This latency shall be ensured even when the two application subsystems are in different clusters.
R 7.4.1	2.3.2.1	The wind turbine shall achieve the stop state (safe state) when the speed of the blades is greater than or equal MAX_BLADE_SPEED.
R 8.4.1	2.3.2.1	The wind turbine shall be in the safe state until a manual reset of the system.
R 9.1.1	3.2	Separation of concerns: The meta-model shall be organized in such a way that different aspects are covered by sub-meta-models.
R 9.1.2	3.2	The meta-models should exhibit an adequate degree of abstraction, i.e. abstracting irrelevant details while providing the information required for the methods and tools based on them.
R 9.1.3	3.2.3, 3.3.2	The proposed meta-models shall be domain-independent.
R 9.2.1	3.2.3, 3.3.2	The application meta-model / PIM shall capture the structure of applications in terms of their component architecture.
R 9.2.2	3.2.3, 3.3	For the application meta-model, precise (platform-independent) execution semantics should be defined.
R 9.2.3	3, 3.3.2.2	The DREAMS application architecture model should provide means for defining the memory needs.
R 9.3.1	3.2.3, 3.3.3	The platform meta-model shall capture the topology and the hierarchic structure of instances of the DREAMS architecture
R 9.3.2	3.2.3, 3.3.3	The platform-meta model shall distinguish different types of building blocks / services contained in instances of the DREAMS architecture.
R 9.4.1	3.2.3, 3.5, 3.6.2.2	The platform-specific meta-model shall provide means to describe applications that are deployed to instances of the DREAMS architecture.
R 9.5.1	2.3.2.1, 3.4	The Timing Requirements Meta-Model shall allow the specification of latency constraints (local or end-to-end).
R 9.5.2	2.3.2.1, 3.4	The Timing Requirements Meta-Model shall allow the specification of repetition constraints.
R 9.5.3	2.3.2.1, 3.4	The Timing Requirements Meta-Model shall allow the specification of synchronization constraints (based on events).
R 9.6.1	3, 3.5	The Reliability / Safety Meta-Model should allow to specify policies according to IEC-61508 realization phase (system architecture definition). Each safety compliant item of a mixed criticality system can specify a failure probability (e.g., an assigned SIL level). So it is required that during modelling process safety policies are checked (e.g., chosen SIL level cannot be higher than the maximum allowable

		SIL).
R 9.6.2	3.3.2.2, 3.5	The Reliability / Safety Meta-Model shall allow specifying criticality-levels according to IEC-61508.
R 9.6.3	2.1.1.2, 2.3.2.3, 3	The meta-models should support the traceability between the artefacts used in the different steps of the development process.
R 9.7.1	3.6.1.4	A high level analytical model covering average static and dynamic power consumption of the NoC at the system-level shall be provided, e.g., a mathematical function.
R 9.7.2	3.6.1.4	The Energy / Power requirements meta-model should be suitable to define requirements on the energy / power consumption of a DREAMS system at the system-level.
R 9.8.1	2.2.1.3, 3.3.2.2	The Security Meta-Model for Data Confidentiality shall allow modelling the varying needs of confidentiality.
R 9.8.2	2.2.1.3, 3.3.2.2	The Security Meta-Model for Data Integrity shall allow modelling the varying needs of data integrity.
R 9.8.3	2.2.1.3, 3.3.2.2	The Security Meta-Model for Authentication shall allow modelling the needs for establishing the authenticity of communication partner and the authentication of data origin.
R 9.9.1	2.1.1.3, 3.6.1.4	The variability meta-model shall allow specifying variations of base models in order to define product lines.
R 9.9.2	2.1.1.3, 3.6.1.4	The variability meta-model shall allow to describe different feature sets of applications.
R 9.9.3	2.1.1.3, 3.6.1.4	The variability meta-model shall allow to describe different implementation alternatives of applications.
R 9.9.4	2.1.1.3, 3, 3.5	DREAMS systems need to be automatically adaptive, and this requirement will help automating the production of a configuration in particular in adapting to different platform technologies (e.g. hardware).
R 9.10.1	3.6	The development process should define the model-to-model transformations required to implement application subsystems on top of the DREAMS platform.
R 9.10.2	3.6	The development methodology should allow the definition of the implementation artefacts, i.e. its end products that have to be produced for a DREAMS-based system.
R 9.11.1	3.6	Offline real-time scheduling methods for mixed criticality systems.
R 9.11.2	3.6	The development process shall support online resource allocation and management strategies.
R 9.11.3	3.6	The development process should support the Design Space Exploration (DSE) method.
R 9.11.4	3, 3.4, 3.6	The development process shall support timing and end-to-end response analysis.
R 9.12.1	2.1.1.1, 2.3.2.2, 3.5	Meet-in-the-middle
R 9.12.3	3.4, 3.5	The development process shall foresee the timing requirements.

R 9.13.3	2.1.1.2, 2.3.2.3, 3	The development process shall support the traceability for requirements regarding: safety, security, etc.
R 9.13.5	2.3.2.4	The development process should support the integration of an additional application subsystem into an existing system.
R 9.13.6	3.6.1.4	The development process should support a reliability analysis, and methods for the introduction of active redundancy that derive a fault-tolerant design from the original design.
R 9.14.1	2.1.1.3, 3, 3.6	The development process shall define how variability is bound.
R 10.4.1	2.3.2.1	The system can be reconfigured upon foreseen and unforeseen changes in its operational and environmental conditions within in a predictable time span.
R 10.4.4	2.3.2.1	Information about the state of individual resources, as provided by the local resource managers, has to be transmitted to the GRM in a timely manner. Similarly, the decisions about resources stated taken by the GRM have be communicated to the LRMs in a timely fashion.
R 11.1.1	2.2.1.2	Core security services
R 11.1.2	2.2.1.2	Identification of core and optional services, security policies and thread models
R 11.2.4	2.2.1.3	Mechanisms for protection against physical attacks, such as side channel attacks, shall be evaluated and provided if found adequate, e.g., if they do not affect the QoS requirements.
R 11.3.3	2.2.1.3	A choice of cipher suites shall be provided. A cipher suite includes cryptographic algorithms and their parameters, e.g., key sizes etc.
R 11.3.4	2.2.1.3	Core security services on the cluster level shall be identified and provided. This includes services such as end-to-end security (e.g., privacy and authentication).
R 11.3.7	2.2.1.3	Key management for secure communication between the entities on a cluster shall be provided (Mechanisms for key generation, key distribution/exchange, key destruction etc.).
R 11.4.1	2.2.1.2	Integrity, authenticity and availability shall be ensured for communications and communications partners, in the presence of security threats, such as message sniffing, insertion, modification and denial of service.
R 11.4.2	2.2.1.2	Security services shall be validated using reasonable attack scenarios and related penetration tests. Attack scenarios in the context of the DREAMS architecture need to be envisaged and implemented to validate the strength of the security services of DREAMS.
R 11.6.1	2.2.2	Integration of security in the development process

Table 7: DREAMS Requirement Matrix.

5 Bibliography

[1] IEC, "IEC 61508-1: General Requirements," in *Requirements for electrical/ electronical / programmable electronic safety-related systems*, ed, 2010.

[2] Y.-Y. Fan Jiang, J. Y. Kuo, and S.-P. Ma, "An Embedded Software Modeling and Process by Using Aspect-Oritented Approach," *Software Engineering and Applications, Journal of,* vol. 4, pp. 106-122, April 2011.

[3] M. P. Papazoglou and W.-J. van den Heuvel, "Service-Ortiented Design and Development Methodology," *Web Engineering and Technology (IJWET), International Journal of,* vol. 2, pp. 412-442, 2006.

[4] IEC, "IEC 61508-2: Requirements for electrical/electronic/programmable electronic safety-related systems," in *Requirements for electrical/ electronical / programmable electronic safety-related systems*, ed, 2010.

[5] ISO, "ISO/IEC 15408-1/2/3," ed, 2005.

[6] ISO, "ISO/IEC 27001," ed, 2013.

[7] ISO, "ISO/IEC 21827 ", ed, 2008.

[8] F. Swiderski and W. Snyder. (2004) Threat modeling. *Microsoft Press*.

[9] S. Lipner and M. Howard, "The Trustworthy Computing Security Development Lifecycle," presented at the Annual Computer Security Applications Conference Tucson, Arizona, 2004.

[10] TIMMO-2-USE. *Methodology description V2*. Available: https://itea3.org/project/workpackage/document/download/861/09033-TIMMO-2-USE-WP-4-D13MethodologydescriptionV2.pdf

[11] TIMMO-2-USE. *Language syntax, semantics, metamodel V2*. Available: https://itea3.org/project/workpackage/document/download/850/09033-TIMMO-2-USE-WP-2-D11Languagesyntax,semantics,metamodelV2.pdf

[12] S. Kent, "Model Driven Engineering," in *Integrated Formal Methods*. vol. 2335, M. Butler, L. Petre, and K. Sere, Eds., ed Berlin Heidelberg: Springer, 2002, pp. 286-298.

[13] J. Bézivin, "In Search of a Basic Principle for Model Driven Engineering," *Upgrade,* vol. 5, pp. 21-24, April 2004.

[14] J. Bézivin, "On the unification power of models," *Softw. Syst. Model.,* vol. 4, pp. 171-188, May 2005.

[15] B. Schätz, A. Pretschner, F. Huber, and J. Philipps, "Model-Based Development of Embedded Systems," in *Advances in Object-Oriented Information Systems*. vol. 2426, J.-M. Bruel and Z. Bellahsene, Eds., ed Berlin Heidelberg: Springer, 2002, pp. 298-311.

[16] H. Kern, A. Hummel, and S. Kühne, "Towards a comparative analysis of meta-metamodels," in *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11 (SPLASH '11 Workshops)*, New York, NY, USA, 2011, pp. 7-12.

[17] OMG, "Meta Object Facility (MOF) Core Specification," ed, 2011.

[18] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, 2nd ed.: Addison-Wesley Longman, Amsterdam, 2008.

[19] M. Broy and K. Stølen, *Specification and development of interactive systems: focus on streams, interfaces, and refinement*. Secaucus, NJ, USA: Springer, 2001.

Terminology

Next there are some of the terminologies used during this deliverable. For more terminology, see deliverable D1.1.1.

Application Specific Integrated Circuit (ASIC)

Integrated circuit designed and manufactured for specific function, where its functionality is defined by the product developer.

Architecture

Specific configuration of hardware and software element in a system.

Element

Part of a subsystem comprising a single component or any group of components that performs one or more element safety functions.

Equipment under control

Equipment, machinery, apparatus or plant used for manufacturing, process, transportation, medical or other activities.

Functional Safety

Part of the overall safety, related to the EUC and the EUC control system that depends on the correct functioning of the E/E/PE safety-related systems and other risk reduction measures.

Pre-existing software

Software element which already exist and is not developed specifically for the current project or safety related system.

Programmable Electronic (PE)

Based on computer technology, which may be composed by hardware, software, and of input and/or output units.

Proven in use

Demonstration, based on analysis of operational experience for a specific configuration of an element, that the likelihood of dangerous systematic faults is low enough so that every safety function that uses the element achieves its required safety integrity level.

Safety

Freedom from unacceptable risk.

Safety Lifecycle

Necessary activities involved in the implementation of safety-related systems, occurring during a period of time that starts at the concept phase of a project and finishes when all of the E/E/PE safety related systems and other risk reduction measures are no longer available for use.

Safety Manual for Compliant Systems

Document that provides all the information relating to the functional safety of an element, in respect of specified element safety functions, that is required to ensure that the system meets the requirements of IEC 61508 series.

Safety related Software

Software that is used to implement safety functions in a safety related system.

Safety Related System

Designated system that implements the required safety functions necessary to achieve or maintain a safe state of/or EUC and is intended to achieve, on its own or with other E/E/PE safety-related systems and other risk reduction measures, the necessary safety integrity for the required safety functions.

Software

Intellectual creation that comprises the programs, procedures, data, rules and any associated documentation pertaining to the operation of a data processing system.

Software Lifecycle

Activities occurring during a period of time that starts when software is conceived and ends when the software is permanently decommissioned.

Software Module

Construct that consist of procedures and/or data declarations and that can also interact with other such constructs.

Subsystem

Entity of the top-level architectural design of a safety-related system, where a dangerous failure of the subsystem results in dangerous failure of a safety function.

Validation

Confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use are fulfilled.

Verification

Confirmation by examination and provision of objective evidence that the requirements have been fulfilled.