AMALTHEA

ITEA 2 - 09013

Model Based Open Source Development Environment for Automotive Multi Core Systems

Work Package 1

Continuous design flow and methodology

Task 1.1

Evaluation/Analysis of existing design flow methods

Deliverable D1.1

State of the art of Design Flow and verification methods and tools

Document type	: Deliverable
Document version	: Final
Document Preparation Date	: 01.07.2011 (delivery: T0+7)
Classification	: public
Contract Start Date	: 01.07.2011
Duration	: 31.12.2013

Final approval	**Name**	**Partner**
Review Task Level	Markus Kelanti	Uni Oulu
Review WP Level	Heiko Adamczyk Ralf Messerschmidt	ifak
Review Board Level	Karlheinz Topp	Bosch

Executive summary

This document is the first deliverable of the itea2 project AMALTHEA. It is established within Task 1.1 *Evaluation/Analysis of existing design flow methods* of WP1 *Tasks Continuous design flow and methodology*.

It provides the relevant state of the art as a common basic knowledge for AMALTHEA concerning the main goal of the project - the development of a consistent, open, expandable, multicore tool platform for automotive engineering, based on the model driven approach as basic engineering methodology.

The document is structured in the following main chapters:

- Chapter 1: Introduction
- Chapter 2: Relevant standards and guidelines
- Chapter 3: Methodology, Concepts and Best Practices
- Chapter 4: State of the art and related work
- Chapter 5: Conclusion

Chapter 2 investigates the important standards and standard-like guidelines. Standards of general industrial importance and derived standards adapted to automotive industry are considered. The standards can be further categorized concerning functional safety related, process reference model related and process assessment related.

Chapter 3 will explain Methodology, Concepts and Best Practices. This chapter uses at first a structure which is based on a given software development model, e.g. well established V-Model. This linear model is a process oriented model and simply divided into a certain number if processes or simply steps. These steps e.g. are Requirement Engineering, Architecture & Design, Coding as well as Verification and Validation. Further aspects which AMALTHEA will take into account are Model Based Design, Safety/Security Design issues and Simulation. One of the important aspects in AMALTHEA is of course the multi core aspects. Hence, the chapter 3 and also the chapter 4 follow a structure based on the above features.

Chapter 4 shows the State of the Art concerning tools and frameworks. The subchapters belongs also to the above descripted feature list. Nevertheless, in the beginning there are tool chains descripted those covering at least two or more features of the overall software development process. Afterwards single tools will be described which are more or less focused on one feature. Most of the selected tools are established in the automotive as well as the industrial automation domain. For sure, the Eclipse based solutions like TOPCASED, Artop and YAKINDU are in main focus of AMALTHEA.

Contents

© The AMALTHEA consortium

List of Tables

List of Figures

© The AMALTHEA consortium

1. Introduction

The goal of this task is to obtain an overview of state of the art design flow methods and to provide the needed information for making a selection of these methods which will be used within this project. The methods cover tasks as: requirement engineering, system architecture design, module design, coding, V&V and system simulation. The main focus lies on the exploration of usability in existing open-source software solutions and their evaluation (e.g. EPF – Eclipse Process Framework – which is based on EMF).

Selected design flow methods (integrating certification aspects) are augmented by domain specific languages in order to fit the needs of multi-ECU and multicore design; which implies tools to be augmented with DSL bridges.

Figure 1-1 gives an overview of the standards with importance for AMALTHEA and shows the relations between these standards, which will be described in the first chapter *Relevant standards and guidelines*.

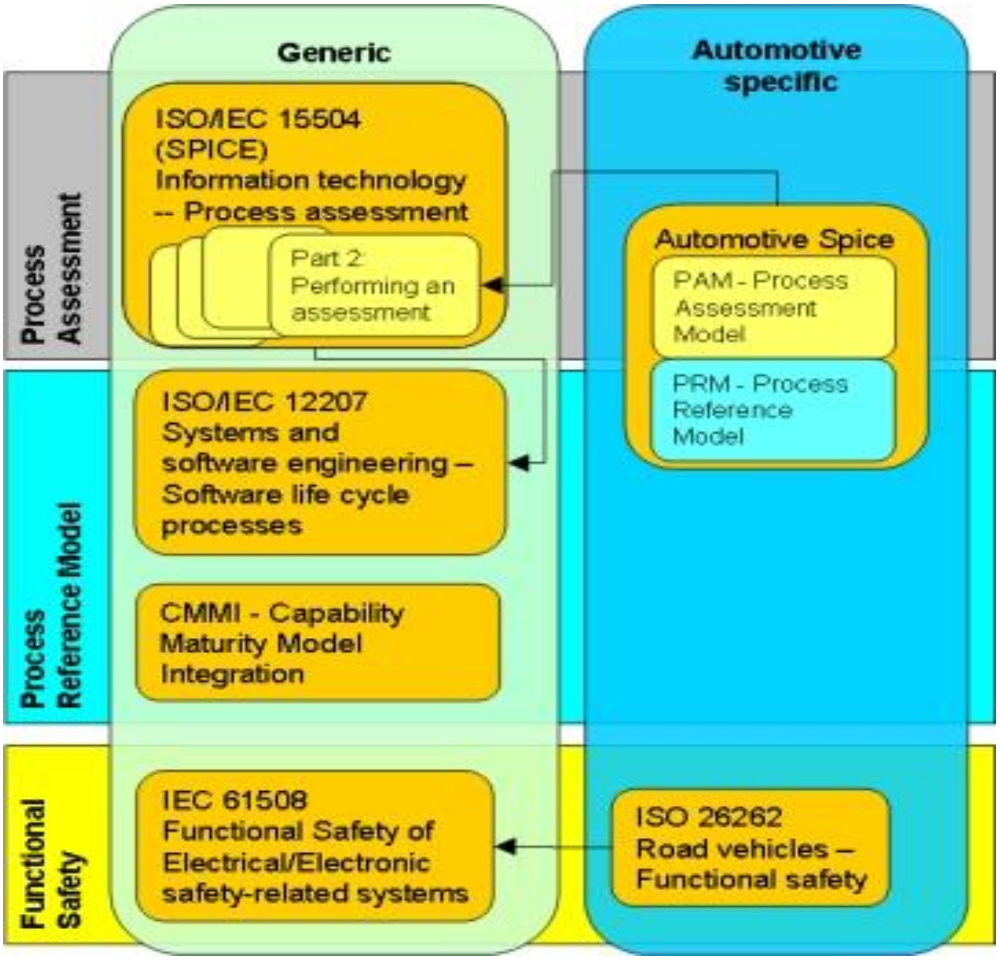

Figure 1-1: Overview of relevant standards for AMALTHEA

© The AMALTHEA consortium

2. Relevant standards and guidelines

The focus of the following chapter is to describe the main relevant standards and guidelines, which AMALTHEA has to take into account. Three different groups can be identified. The first group is dealing with functional safety aspects. The second group can be seen as an extract of the main important software development models. This group explains process reference models and also process assessment related aspects. Finally, the third group considers most relevant standards for the automotive industry sector.

However, many more standards are available those covering the entire lifecycle of a software development process. During the first analysis in this task 1.1, the two selected standards were selected which shows the basic principle of such models. All the over standards identified are close to each other that is why these number of standards wouldn't listed here. Fortunately, during the later specification of the AMALTHEA approach, it could be happen that more standards will be finally considered as the number mentioned here in the next chapters.

2.1. IEC61508

2.1.1. Scope of IEC 61508

An error or failure of safety-related systems and machines may be a risk for people and the environment as well as a threat to the function of the systems. These risks must be identified and judged by the operator of the plant. On the basis of this risk analysis it is necessary to take measures to identify and prevent errors. The object of our efforts is to reduce risks for the whole system.

A part of the whole safety is the functional safety. Functional safety covers systems which carry out safety functions with a certain probability if there are identified causes of errors. If these safety functions use electrical, electronic or programmable electronic systems (E/E/PE), these systems and their components must meet the conditions of IEC 61508.

On the one hand IEC 61508 defines the requirements on safety-related E/E/PE systems regardless of its use. On the other hand the described methods enable a quantitative judgment of safety-related E/E/PE systems.

The described requirements counter risks which are caused by failure of safety-related systems. It is important that IEC 61508 is applied to safety functions of the whole safety-related E/E/EP system. That means, all components – sensor, control, actuator, communication system and errors of the user – are taken into account.

2.1.2. Overview

For the purpose of quantitative judgment of safety-related E/E/PE systems the IEC 61508 distinguishes between "low demand mode of operation" and "high demand or continuous mode of operation". Furthermore the systems are divided into 4 groups

© The AMALTHEA consortium

which are related to their minimum requirement on conditional probability of failure of safety equipment; it is called safety integrity level (see Table 2-1).

Table 2-1: IEC 61508 safety integrity levels

Safety integrity level	**Low demand mode of operation (Average probability of failure to perform its design function on demand)**	**High demand or continuous mode of operation (Probability of a dangerous failure per hour)**
4	>= 10^{-5} to < 10^{-4}	>= 10^{-9} to < 10^{-8}
3	>= 10^{-4} to < 10^{-3}	>= 10^{-8} to < 10^{-7}
2	>= 10^{-3} to < 10^{-2}	>= 10^{-7} to < 10^{-6}
1	>= 10^{-2} to < 10^{-1}	>= 10^{-6} to < 10^{-5}

Furthermore IEC 61508 describes systematically how to meet the requirements of a safety-related E/E/PE system for the whole life cycle. That means, it is not enough to meet the requirements just once, it must be guaranteed to meet the requirements for the period of the whole life cycle of a safety-related E/E/PE system.

For the process of software development, which contains all steps of development and implementation, IEC 61508 refers to the V-model. In detail this means, all steps of development – starting with requirements and coding until integration – must be traceable and described. Besides it is necessary that each step of development can be verified and defined within the design process.

© The AMALTHEA consortium

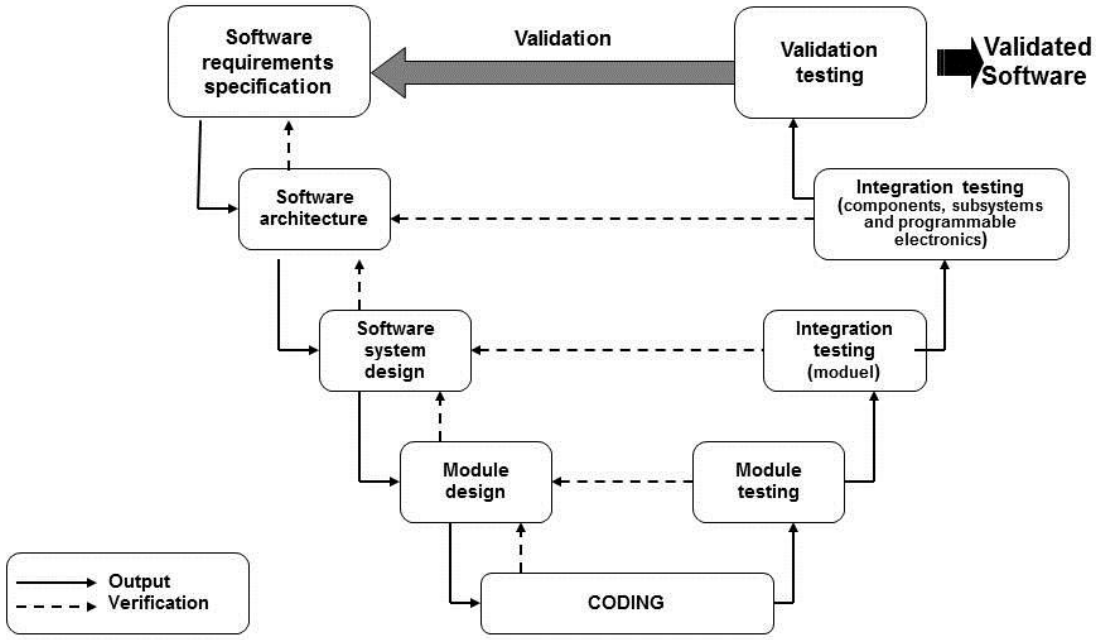

Figure 2-1: Functional safety in the sense of the IEC 61508

2.1.2.1. Safety functions and safety-related systems

Generally, the significant hazards for equipment and any associated control system in its intended environment have to be identified by the specifier or developer via a hazard analysis. The analysis determines whether functional safety is necessary to ensure adequate protection against each significant hazard. If so, it has to be taken into account in an appropriate manner in the design. Functional safety is just one method of dealing with hazards, and other means for their elimination or reduction, such as inherent safety through design, are of primary importance.

The term safety-related is used to describe systems that are required to perform a specific function or functions to ensure risks are kept at an accepted level. Such functions are, by definition, safety functions. Two types of requirements are necessary to achieve functional safety:

- safety function requirements (what the function does) and
- safety integrity requirements (the likelihood of a safety function being performed satisfactorily).

The safety function requirements are derived from the hazard analysis and the safety integrity requirements are derived from a risk assessment. The higher the level of safety integrity, the lower the likelihood of dangerous failure.

© The AMALTHEA consortium

Any system, implemented in any technology, which carries out safety functions is a safety related system. A safety-related system may be separate from any equipment control system or the equipment control system may itself carry out safety functions. In the latter case, the equipment control system will be a safety-related system. Higher levels of safety integrity necessitate greater rigour in the engineering of the safety-related system.

2.1.2.2. Example of functional safety

Consider a machine with a rotating blade that is protected by a hinged solid cover. The blade is accessed for routine cleaning by lifting the cover. The cover is interlocked so that whenever it is lifted an electrical circuit de-energises the motor and applies a brake. In this way, the blade is stopped before it could injure the operator.
In order to ensure that safety is achieved, both hazard analysis and risk assessment are necessary.

1. The hazard analysis identifies the hazards associated with cleaning the blade. For this machine it might show that it should not be possible to lift the hinged cover more than 5 mm without the brake activating and stopping the blade. Further analysis could reveal that the time for the blade to stop shall be 1 s or less. Together, these describe the safety function.
2. The risk assessment determines the performance requirements of the safety function. The aim is to ensure that the safety integrity of the safety function is sufficient to ensure that no one is exposed to an unacceptable risk associated with this hazardous event. The harm resulting from a failure of the safety function could be amputation of the operator's hand or could be just a bruise. The risk also depends on how frequently the cover has to be lifted, which might be many times during daily operation or might be less than once a month. The level of safety integrity required increases with the severity of injury and the frequency of exposure to the hazard.

The safety integrity of the safety function will depend on all the equipment that is necessary for the safety function to be carried out correctly, i.e. the interlock, the associated electrical circuit and the motor and braking system. Both the safety function and its safety integrity specify the required behavior for the systems as a whole within a particular environment. To summarise, the hazard analysis identifies what has to be done to avoid the hazardous event, or events, associated with the blade. The risk assessment gives the safety integrity required of the interlocking system for the risk to be acceptable. These two elements, "What safety function has to be performed?" – the safety function requirements – and "What degree of certainty is necessary that the safety function will be carried out?" – the safety integrity requirements – are the foundations of functional safety.

2.1.2.3. Challenges in achieving functional safety

Safety functions are increasingly being carried out by electrical, electronic or programmable electronic systems. These systems are usually complex, making it impossible in practice to fully determine every failure mode or to test all possible behavior. It is difficult to predict the safety performance, although testing is still

 © The AMALTHEA consortium

essential. The challenge is to design the system in such a way as to prevent dangerous failures or to control them when they arise. Dangerous failures may arise from:

- incorrect specifications of the system, hardware or software;
- omissions in the safety requirements specification (e.g. failure to develop all relevant safety functions during different modes of operation);
- random hardware failure mechanisms;
- systematic hardware failure mechanisms;
- software errors;
- common cause failures;

2.1.2.4. Nominative references and parts framework of IEC 61508

The IEC 61508 consists of 8 parts:

- IEC 61508-0, Functional safety and IEC 61508
- IEC 61508-1, General requirements
- IEC 61508-2, Requirements for electrical/electronic/programmable electronic safety-related systems
- IEC 61508-3, Software requirements
- IEC 61508-4, Definitions and abbreviations
- IEC 61508-5, Examples of methods for the determination of safety integrity levels
- IEC 61508-6, Guidelines on the application of IEC 61508- 2 and IEC 61508-3
- IEC 61508-7, Overview of techniques and measures additional papers:
- IEC Guide 104, The preparation of safety publications and the use of basic safety publications and group safety publications
- ISO/IEC Guide 51, Safety aspects – Guidelines for their inclusion in standards

2.1.2.5. Functional safety of E/E/PE safety-related systems

2.1.2.5.1. Objectives

IEC 61508 aims to:

- release the potential of E/E/PE technology to improve both safety and economic performance;
- enable technological developments to take place within an overall safety framework;
- provide a technically sound, system based approach, with sufficient flexibility for the future;
- provide a risk-based approach for determining the required performance of safety-related systems;
- provide a generically-based standard that can be used directly by industry but can also help with developing sector standards (e.g. machinery, process chemical plants, medical or rail) or product standards (e.g. power drive systems);
- provide a means for users and regulators to gain confidence when using computer-based technology;

- provide requirements based on common underlying principles to facilitate:
    - improved efficiencies in the supply chain for suppliers of subsystems and components to various sectors,
    - improvements in communication and requirements (i.e. to increase clarity of what needs to be specified),
    - the development of techniques and measures that could be used across all sectors, increasing available resources,
    - the development of conformity assessment services if required.

IEC 61508 does not cover the precautions that may be necessary to prevent unauthorised persons damaging, and/or otherwise adversely affecting, the functional safety achieved by E/E/PE safety-related systems.

2.1.2.5.2. E/E/PE safety-related systems

IEC 61508 is concerned with functional safety, achieved by safety-related systems that are primarily implemented in electrical and/or electronic and/or programmable electronic (E/E/PE) technologies, i.e. E/E/PE safety related systems. The standard is generic in that it applies to these systems irrespective of their application. Some requirements of the standard relate to development activities where the implementation technology may not yet have been fully decided. This includes development of the overall safety requirements (concept, scope definition, hazard analysis and risk assessment). If there is a possibility that E/E/PE technologies might be used, the standard should be applied so that the functional safety requirements for any E/E/PE safety-related systems are determined in a methodical, risk-based manner.
Other requirements of the standard are not solely specific to E/E/PE technology, including documentation, management of functional safety, functional safety assessment and competence. All requirements that are not technology-specific might usefully be applied to other safety-related systems although these systems are not within the scope of the standard.

An E/E/PE safety-related system covers all parts of the system that are necessary to carry out the safety function (i.e. from sensor, through control logic and communication systems, to final actuator, including any critical actions of a human operator). Since the definition of E/E/PE safety-related system is derived from the definition of safety, it also concerns freedom from unacceptable risk of both physical injury and damage to the health of people. The harm can arise indirectly as a result of damage to property or the environment. However, some systems will be designed primarily to protect against failures with serious economic implications. IEC 61508 can be used to develop any E/E/PE system that has critical functions, such as the protection of equipment or products.

2.1.2.5.3. Technical approach

The technical approach uses a risk based approach to determine the safety integrity requirements of E/E/PE safety-related systems, and includes a number of examples of how this can be done;

- uses an overall safety lifecycle model as the technical framework for the activities necessary for ensuring functional safety is achieved by the E/E/PE

 © The AMALTHEA consortium

safety-related systems; covers all safety lifecycle activities from initial concept, through hazard analysis and risk assessment, development of the safety requirements, specification, design and implementation, operation and maintenance, and modification, to final decommissioning and/or disposal;
- encompasses system aspects (comprising all the subsystems carrying out the safety functions, including hardware and software) and failure mechanisms (random hardware and systematic);
- contains both requirements for preventing failures (avoiding the introduction of faults) and requirements for controlling failures (ensuring safety even when faults are present);
- specifies the techniques and measures that are necessary to achieve the required safety integrity.

2.1.2.5.4. Safety integrity levels (SIL)

IEC 61508 specifies 4 levels of safety performance for a safety function. These are called safety integrity levels. Safety integrity level 1 (SIL1) is the lowest level of safety integrity and safety integrity level 4 (SIL4) is the highest level. The standard details the requirements necessary to achieve each safety integrity level. These requirements are more rigorous at higher levels of safety integrity in order to achieve the required lower likelihood of dangerous failure.

An E/E/PE safety-related system will usually implement more than one safety function. If the safety integrity requirements for these safety functions differ, unless there is sufficient independence of implementation between them, the requirements applicable to the highest relevant safety integrity level shall apply to the entire E/E/PE safety-related system. If a single E/E/PE system is capable of providing all the required safety functions, and the required safety integrity is less than that specified for SIL1, then IEC 61508 does not apply.

2.1.2.6. Overall safety lifecycle requirements

In order to deal in a systematic manner with all the activities necessary to achieve the required safety integrity level for the E/E/PE safety-related systems, this standard adopts an overall safety lifecycle (see Figure 2-2) as the technical framework. The overall safety lifecycle encompasses the following risk reduction measures:

- E/E/PE safety-related systems;
- other technology safety-related systems;
- external risk reduction facilities.

The portion of the overall safety lifecycle dealing with E/E/PE safety-related systems is expanded and shown in Figure 2-2. This is termed the E/E/PES safety lifecycle and forms the technical framework for IEC 61508-2. The software safety lifecycle is shown in Figure 2-2 and forms the technical framework for IEC 61508-3. The relationship of the overall safety lifecycle to the E/E/PES and software safety lifecycles for safety-related systems is shown in Figure 2-2.

The overall, E/E/PES and software safety lifecycle are simplified views of reality and as such do not show all the iterations relating to specific phases or between phases.

© The AMALTHEA consortium

Iteration, however, is an essential and vital part of development through the overall, E/E/PES and software safety lifecycles.

Activities relating to the management of functional safety (clause 6), verification and functional safety assessment (clause 8) are not shown on the overall, E/E/PES or software safety lifecycles. This has been done in order to reduce the complexity of the overall, E/E/PES and software safety lifecycle figures. These activities, where required, will need to be applied at the relevant phases of the overall, E/E/PES and software safety lifecycles.

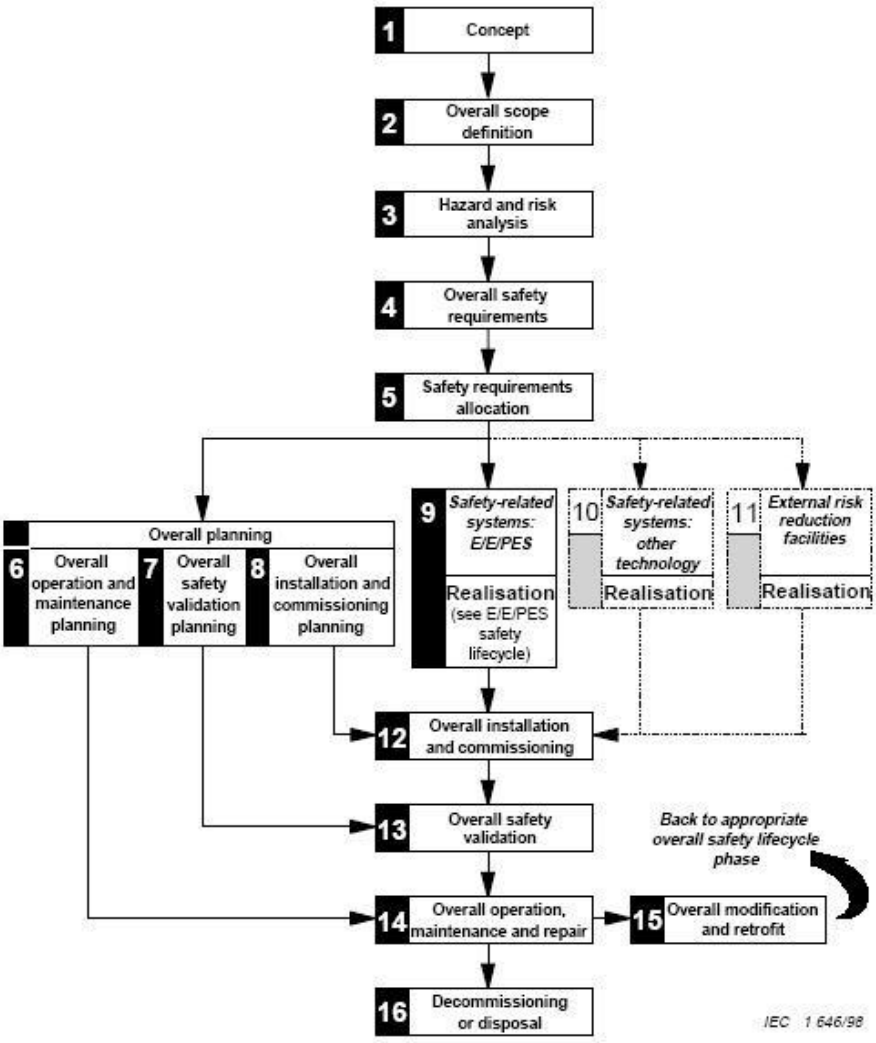

Figure 2-2: Overall safety lifecycle (IEC 61508)

2.1.3. Relevance

"A major objective of this standard is to facilitate the development of product and application sector international standards by the technical committees responsible for the product or application sector."..." A second objective of this standard is to enable the development of E/E/PE safety-related systems where product or application sector international standards do not exist." The later sentence implies that if a

 © The AMALTHEA consortium

safety-related international standard for a special application field exists, this standard has to be applied for product developments at that field-and not the IEC 61508.

2.1.3.1. The IEC 61508 as a basis for other standards

Standard writers need to address functional safety in their safety standard if the hazard analysis carried out by a Technical Committee identifies that this is necessary to adequately protect against a significant hazard or hazardous event.

Parts 1, 2, 3 and 4 of IEC 61508 are IEC basic safety publications. One of the responsibilities of IEC Technical Committees is, wherever practical, to make use of these parts of IEC 61508 in the preparation of their own sector or product standards that have E/E/PE safety-related systems within their scope. For more details see IEC Guide 104 and ISO/IEC Guide 51.

IEC 61508 is the basis for published sector standards (e.g. process sector). It is also currently being used as a basis for developing other sector standards and product standards. It is therefore influencing the development of E/E/PE safety-related systems and products across all sectors.

Sector specific standards based on IEC 61508:

- are aimed at system designers, system integrators and users;
- take account of specific sector practice, which can allow less complex requirements;
- use sector terminology to increase clarity;
- may specify particular constraints appropriate for the sector;
- usually rely on the requirements of IEC 61508 for detailed design of subsystems;
- may allow end users to achieve functional safety without having to consider IEC 61508 themselves.

Such a specific standard derived from IEC 61508 is IEC 26262 which is the adaptation for the specific field of road vehicles.

The basic safety publication status of IEC 61508 described above does not apply for low complexity E/E/PE safety-related systems (see 4.2 of IEC 61508-1). These are E/E/PE safety related systems in which the failure modes of each individual component are well-defined and the behaviour of the system under fault conditions can be completely determined. An example is a system comprising of one or more limit switches, operating one or more contactors to de-energise an electric motor, possibly via interposing electromechanical relays.

2.1.3.2. The IEC 61508 as a stand-alone standard

All parts of IEC 61508 can be used directly by the industry as “stand-alone” publications. This includes use of the standard:

- as a set of general requirements for E/E/PE safety-related systems where no application sector or product standards exist or where they are not appropriate;
- by suppliers of E/E/PE components and subsystems for use in all sectors (e.g. hardware and software of sensors, smart actuators, programmable controllers, data communication);
- by system builders to meet user specifications for E/E/PE safety-related systems;
- by users to specify requirements in terms of the safety functions to be performed together with the performance requirements of those safety functions;
- to facilitate the maintenance of the "as designed" safety integrity of E/E/PE safety-related systems;
- to provide the technical framework for conformity assessment and certification services;
- as a basis for carrying out assessments of safety lifecycle activities.

2.2. ISO 26262

2.2.1. Scope

The following information is taken from [ISO26262]

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

Systems and their components released for production, or systems and their components already under development prior to the publication date of ISO 26262, are exempted from the scope. For further development or alterations based on systems and their components released for production prior to the publication of ISO 26262, only the modifications will be developed in accordance with ISO 26262.

ISO 26262 addresses possible hazards caused by malfunctioning behaviour of E/E safety-related systems, including interaction of these systems. It does not address hazards related to electric shock, fire, smoke, heat, radiation, toxicity, flammability, reactivity, corrosion, release of energy and similar hazards, unless directly caused by malfunctioning behaviour of E/E safety-related systems.

ISO 26262 does not address the nominal performance of E/E systems, even if dedicated functional performance standards exist for these systems (e.g. active and passive safety systems, brake systems, Adaptive Cruise Control).

 © The AMALTHEA consortium

2.2.2. Overview of ISO 26262

ISO 26262:

- provides an automotive safety lifecycle (management, development, production, operation, service, decommissioning) and supports tailoring the necessary activities during these lifecycle phases,
- provides an automotive specific risk-based approach for determining integrity levels (Automotive Safety Integrity Levels (ASIL)),
- uses ASILs for specifying the applicable requirements of ISO 26262 for avoiding unreasonable residual risk,
- provides requirements for validation and confirmation measures to ensure a sufficient and acceptable level of safety is being achieved,
- provides requirements for the relation with suppliers and
- requires an independent assessment of the development process

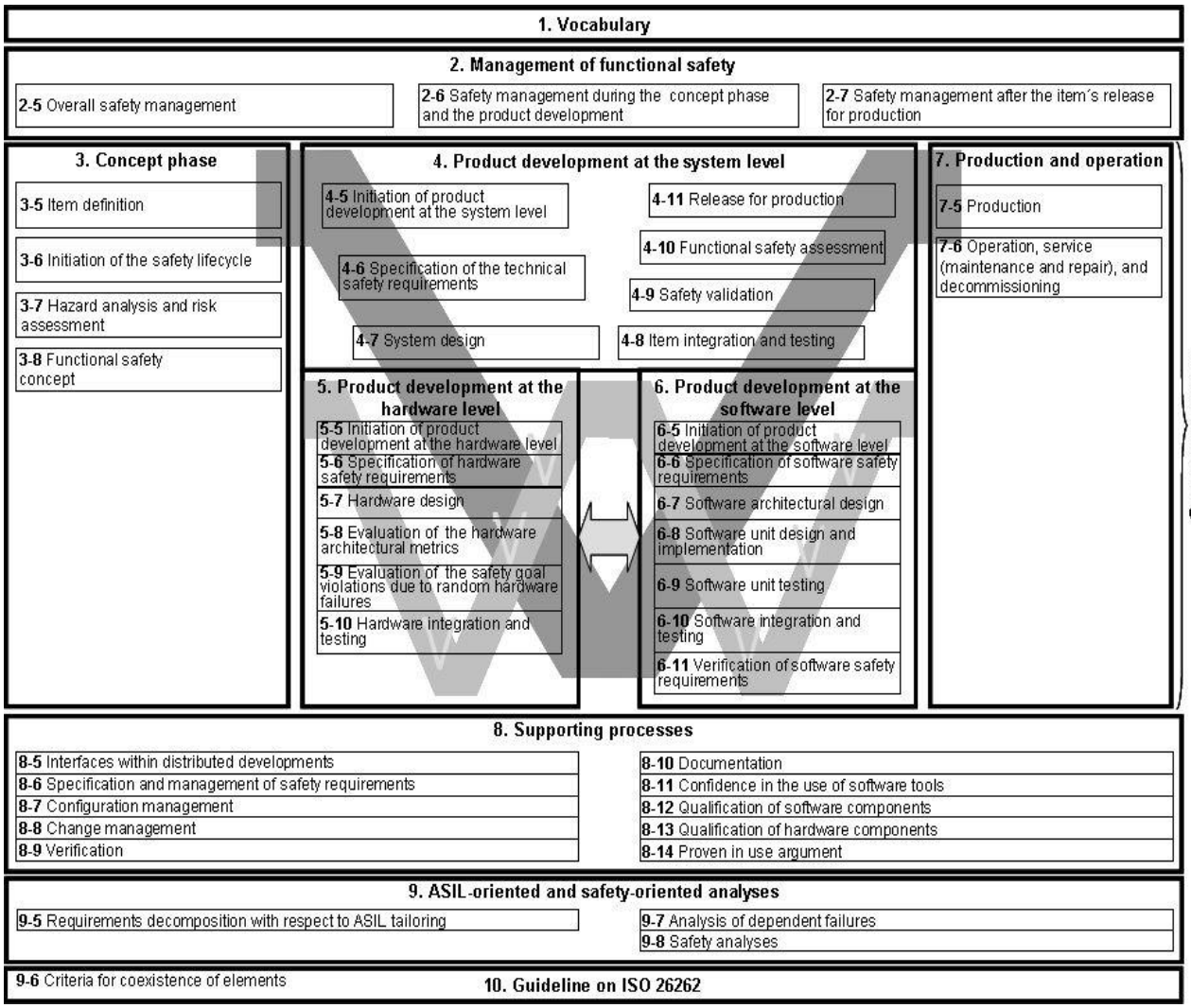

Figure 2-3: Overall structure of ISO 26262

Figure 2-3 shows the overall structure of Baseline 19 from July 2011 of ISO 26262. It has ten parts and is based upon a V-model as a reference process model for the different phases of product development. Within the Figure 2-3:

- the shaded "V"s represent the interconnection between ISO 26262-3, ISO 26262-4, ISO 26262-5, ISO 26262-6 and ISO 26262-7,
- the specific clauses are indicated in the following manner: “m-n”, where “m” represents the number of the part and “n” indicates the number of the clause within that part.

2.2.2.1. Table of contents of ISO 26262

The following information is taken from [ISO26262].

Part 1: Vocabulary
Part 2: Management of functional safety
Part 3: Concept phase
Part 4: Product development at the system level
Part 5: Product development at the hardware level
Part 6: Product development at the software level
Part 7: Production and operation
Part 8: Supporting processes
Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses
Part 10: Guideline on ISO 26262

ISO 26262-1:2011 specifies the terms, definitions and abbreviated terms for application in all parts of ISO 26262.

ISO 26262-2:2011 specifies the requirements for functional safety management for automotive applications, including the following:

- project-independent requirements with regard to the organizations involved (overall safety management), and
- project-specific requirements with regard to the management activities in the safety lifecycle (i.e. management during the concept phase and product development, and after the release for production).

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

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

ISO 26262-4:2011 specifies the requirements for product development at the system level for automotive applications, including the following:

- requirements for the initiation of product development at the system level,

 © The AMALTHEA consortium

- specification of the technical safety requirements,
- the technical safety concept,
- system design,
- item integration and testing,
- safety validation,
- functional safety assessment, and
- product release.

ISO 26262-5:2011 specifies the requirements for product development at the hardware level for automotive applications, including the following:

- requirements for the initiation of product development at the hardware level,
- specification of the hardware safety requirements,
- hardware design,
- hardware architectural metrics, and
- evaluation of violation of the safety goal due to random hardware failures and hardware integration and testing.

The requirements of ISO 26262-5:2011 for hardware elements are applicable both to non-programmable and programmable elements, such as ASIC, FPGA and PLD. Furthermore, for programmable electronic elements, requirements in ISO 26262-6, ISO 26262-8:2011, Clause 11, and ISO 26262-8:2011, Clause 12, are applicable.

ISO 26262-6:2011 specifies the requirements for product development at the software level for automotive applications, including the following:

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

ISO 26262-7:2011 specifies the requirements for production, operation, service and decommissioning.

ISO 26262-8:2011 specifies the requirements for supporting processes, including the following:

- interfaces within distributed developments,
- overall management of safety requirements,
- configuration management,
- change management,
- verification,
- documentation,
- confidence in the use of software tools,
- qualification of software components,
- qualification of hardware components, and
- proven in use argument.

© The AMALTHEA consortium

ISO 26262-9:2011 specifies the requirements for Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses, including the following:

- requirements decomposition with respect to ASIL tailoring,
- criteria for coexistence of elements,
- analysis of dependent failures, and
- safety analyses.

End of material from ISO Web-page

2.2.3. Relevance for Amalthea

For the purpose of Amalthea, Parts 6 and 8 are most important, especially chapter *8.11 Confidence in the Use of Software Tools*

An example of the application of ISO 26262, is provided in [Mathw2010] where the authors show how to apply ISO 26262 to software tools and how they got qualification.

Please note, that there has been a change in nomenclature since July 2010.

1. Henceforward, there are only three tool confidence levels instead of four. With some reservations, this may be seen as a merging of TCL2old and TCL3old to TCL2new.
2. Tool impact has been renamed (TI0old -> TI1new , TI1old -> TI2new) See the changes referring the former nomenclature depicted in the following diagram:

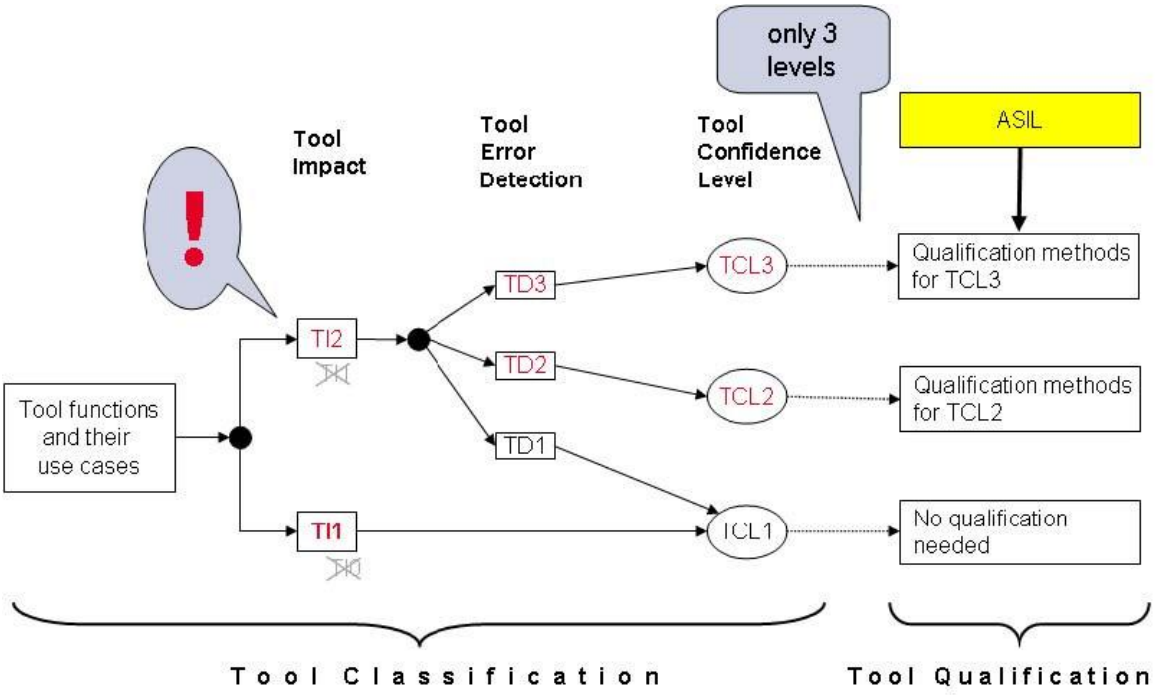

Figure 2-4: Tool classification and tool qualification levels in ISO26262

 © The AMALTHEA consortium

References

[ISO26262] ISO 26262:2011, www.iso.org.

[Mathw2010] Conrad, Mirko; Munier, Patrick; Rauch Frank: Qualifying Software Tools According to ISO 26262, MBEES2010, Dagstuhl-Workshop MBEES: Model-Based Development of Embedded Systems, Munich 2010.

2.3. ISO / IEC 12207:2008 "Systems and software engineering – Software life cycle processes"

2.3.1. Scope

This standard for software industry "establishes a common framework for software life cycle processes, with well-defined terminology" [ISO/IEC12207:2008].

The Process Reference Model does not represent a particular process implementation approach nor does it prescribe a system/software life cycle model, methodology or technique. Instead the reference model is intended to be adopted by an organization based on its business needs and application domain. The organization's defined process is adopted by the organization's projects in the context of the customer requirements.
The following process fields are defined:

1. Agreement Processes — two processes
2. Organizational Project-Enabling Processes — five processes
3. Project Processes — seven processes
4. Technical Processes — eleven processes
5. Software Implementation Processes — seven processes
6. Software Support Processes — eight processes
7. Software Reuse Processes — three processes

ISO 12207 is related to software, the corresponding standard for systems is the ISO 15288 - System life cycle processes, both standards are using the same terminology and are now harmonized. "In many cases, the processes of this International Standard directly correspond to processes of ISO/IEC 15288 but with some specialization for software products and services" [ISO/IEC12207:2008]. ISO 12207 emphasizes, that software should be "considered as an integral part of the system and system design processes".

2.3.2. Overview

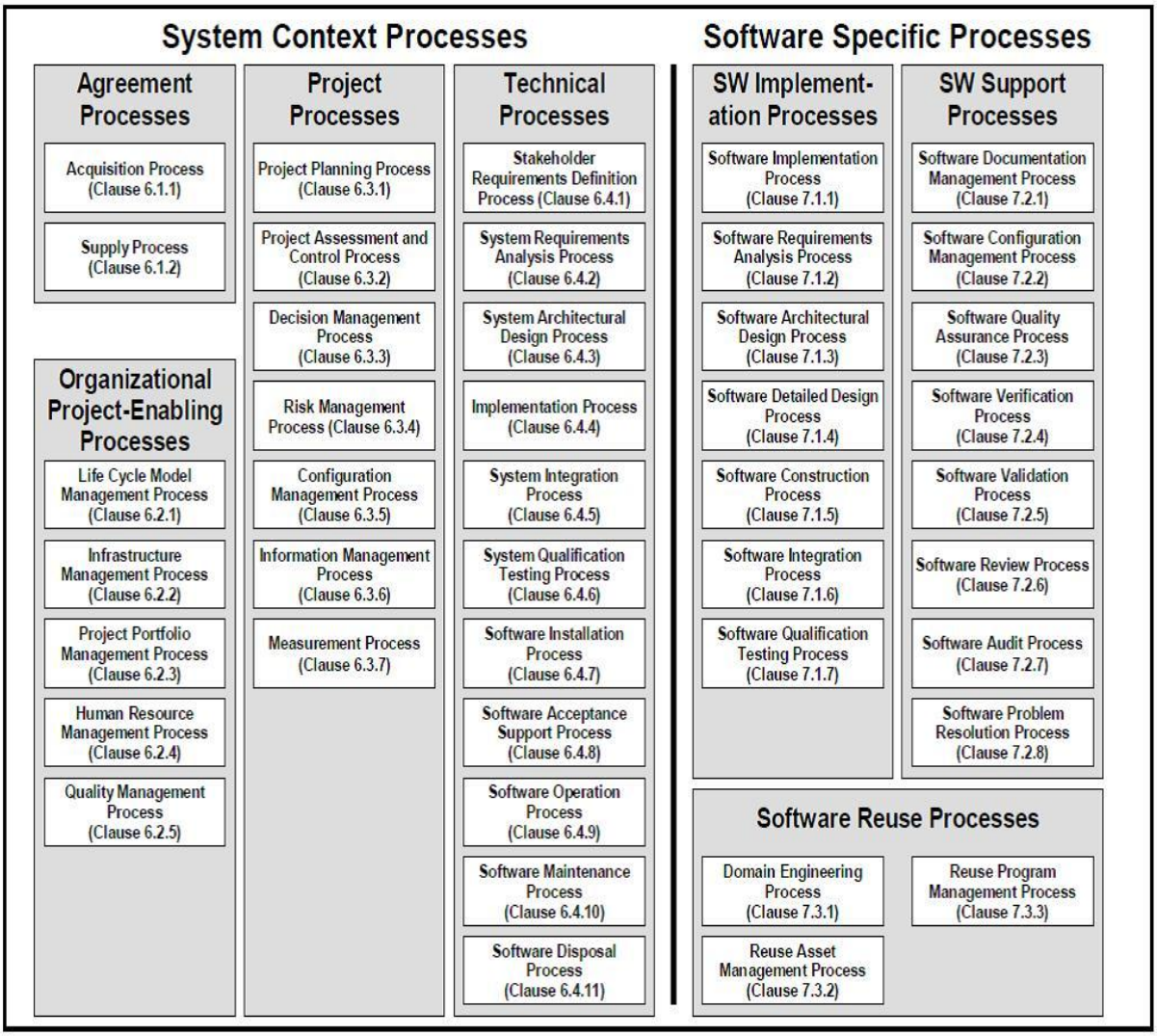

Figure 2-5: ISO 12207 Lifecycle process groups

Important for AMALTHEA are:

- Software Implementation Processes
- Software Requirements Analysis Process
- Software Architectural Design Process
- Software Detailed Design Process
- Software Construction Process
- Software Integration Process
- Software Configuration Management Process
- Software Verification and Validation Process

the following processes have to be considered:

- Software Quality Assurance Process
- Software Documentation Management Process
- Software Qualification Testing Process

 © The AMALTHEA consortium

2.3.3. Relevance

As stated in the scope the reference model of ISO 12207 is generic and is expected to be adapted to specific fields. Such an adaptation for automotive industry is included in Automotive SPICE. Automotive industry is an important application field for AMALTHEA but not the only application field, so the generic versions of standards are an important aspect for the usability of AMALTHEA outcome in various application fields.

2.4. Capability Maturity Model Integration

2.4.1. Scope

Capability Maturity Model Integration (CMMI) is a process improvement approach with the goal to help organizations to improve their efficiency and performance. CMMI can be used to guide process improvement across a project, a division, or an entire organization. Currently supported is CMMI Version 1.3 (release date November 1, 2010). CMMI has its roots in an initiative of the US defense department and the Software Engineering Institute (SEI) of the Carnegie Mellon University/Pittsburgh for the development of an evaluation system for the maturity of software. The technical reports are freely downloadable at: http://www.sei.cmu.edu/cmmi

2.4.2. Overview

The reports describe intensively general organizational relationships and practices in a praxis-related way. Beside three Integration models

- CMMI for Acquisition (CMMI-ACQ),
- CMMI for Development (CMMI-DEV),
- CMMI for Services (CMMI-SVC), - also a
- People CMM has been released.

The models are collections of best practices but they are not processes or process descriptions. The actual processes used in an organization depend on many factors, including application domains and organization structure and size. In particular, the process areas of a CMMI model typically do not map one to one with the processes used in a specific organization.

The best practices can be distinguished as following: there are general practices - belonging to all models, shared practices which are part of more than one model, and specific practices which are uniquely used in one model. The best practices are clustered in process areas and the process areas are allocated to categories. There are the three categories Project Management, Support, and Process Management which are used in all models. Furthermore there are the model specific categories Engineering (CMMI-DEV), Acquisition (CMMI-ACQ), and Service Establishment and Delivery (CMMI-SVC).

For the evaluation of organization's processes two different kinds of levels are used in CMMI. Maturity levels - used to describe the degree of process improvement

© The AMALTHEA consortium

across a predefined set of process areas in which all goals in the set are attained. Capability levels - used to describe the achievement of process improvement within an individual process area. Both capability levels and maturity levels provide a way to improve the processes of an organization and measure how well organizations can and do improve their processes. By this the levels itself are also considered describing an evolutionary improvement-path.

Four capability levels are defined (incomplete, performed, managed and defined) and there are fife maturity levels defined (inital, managed, defined, quantitatively managed and optimising).

For applying CMMI contents to an organization a concrete target of improvement is strictly recommended.

Within a so called appraisal an organization's processes will be appraised (assessed) and a maturity level will be allocated to the organization.

CMMI for Acquisition (CMMI-ACQ)

Provides best practices for acquiring products and services.

CMMI for Development (CMMI-DEV)

Provides best practices for developing products and services. It covers the product's lifecycle from conception through delivery and maintenance. It was the first of the three CMMI models released in year 2000.

CMMI-DEV contains the 22 process areas listed in the table below. Of those process areas, 16 are core process areas, 1 is a shared process area, and 5 (category Engineering) are development specific process areas.

Table: CMMI-DEV process areas with associated categories and maturity levels

Process Area	Category	Maturity Level
Project Planning (PP)	Project Management	2
Requirements Management (REQM)	Project Management	2
Supplier Agreement Management (SAM)	Project Management	2
Project Monitoring and Control (PMC)	Project Management	2
Risk Management (RSKM)	Project Management	3
Integrated Project Management (IPM)	Project Management	3
Quantitative Project Management (QPM)	Project Management	4
Product Integration (PI)	Engineering	3
Requirements Development (RD)	Engineering	3
Technical Solution (TS)	Engineering	3
Validation (VAL)	Engineering	3

 © The AMALTHEA consortium

Verification (VER)	Engineering	3
Organizational Process Definition (OPD)	Process Management	3
Organizational Process Focus (OPF)	Process Management	3
Organizational Training (OT)	Process Management	3
Organizational Process Performance (OPP)	Process Management	4
Organizational Performance Management (OPM)	Process Management	5
Configuration Management (CM/SCM)	Support	2
Measurement and Analysis (MA)	Support	2
Process and Product Quality Assurance (PPQA)	Support	2
Decision Analysis and Resolution (DAR)	Support	3
Causal Analysis and Resolution (CAR)	Support	5

CMMI for Services (CMMI-SVC)

Provides best practices for organizations providing services.

People CMM

People CMM addresses the critical human capital issue of organizations by providing best practices for managing and developing an organization's workforce. Example issues are: human resources, knowledge management, or organizational development.

2.4.3. Relevance

CMMI for Development (CMMI-DEV) will be the most relevant part for AMALTHEA.

2.4.3.1. Comparing with other norms

CMMI follows the same intension as DIN EN ISO 9001, while ISO 9001 (Quality management systems – Requirements) covers the entirety of organizational process fields, CMMI goes more in depth but for a restricted number of process fields. Beside CMMI there are also ISO/IEC 12207 (Systems and software engineering – Software life cycle processes) for software development and ISO 15288 (Systems and software engineering – System life cycle processes) for system development. Both norms have equal demands as CMMI-Dev. The explanations in CMMI are more in depth, while the structure of ISO 12207 is easier to grasp.
The process assessment model for the ISO 12207 defined in ISO/IEC 15504 (also referred to as SPICE) part 5 (Information technology, Process Assessment, Part 5: An exemplar Process Assessment Model) is CMMI-independent, but CMMI follows the same process requirements from ISO/IEC 15504.

2.4.4. References

http://www.sei.cmu.edu/cmmi/start/index.cfm

2.5. Automotive SPICE

2.5.1. Scope

Automotive SPICE is a domain specific derivate of the international standard ISO/IEC 15504 (SPICE). This variant focuses on the development of embedded automotive electronic control units and is a subset of the ISO/IEC 15504. Additionally, about 15% of the text has been rewritten in order to better fit to the development of embedded systems.

2.5.2. Overview

Automotive SPICE defines a process reference model (PRM) and an process assessment model (PAM).

2.5.2.1. Automotive SPICE - Process Reference Model (PRM)

The Automotive SPICE process reference model is derived from ISO/IEC 15504 (part 2) which refers to the process model defined in ISO/IEC 12207. Automotive SPICE focuses on 26 of the 48 processes defined in ISO/IEC 15504 and adds 5 additional processes. Since the effort for performing an assessment of all 31 Automotive SPICE processes is still very high, the Hersteller-Initiative Software (HIS) a group of German car manufacturers (Audi, BMW, Daimler, Porsche and Volkswagen) has defined a minimal subset of Automotive SPICE processes which should be considered during an assessment.

The following Figure 2-6 gives an overview over the processes defined in ISO/IEC 15504 and the focus in Automotive SPICE as well as the scope defined by the HIS group. (See also Höhn, Sechser, Dussa-Zieger, Messnarz, Hindel. "Software Engineering nach Automotive SPICE").

 © The AMALTHEA consortium

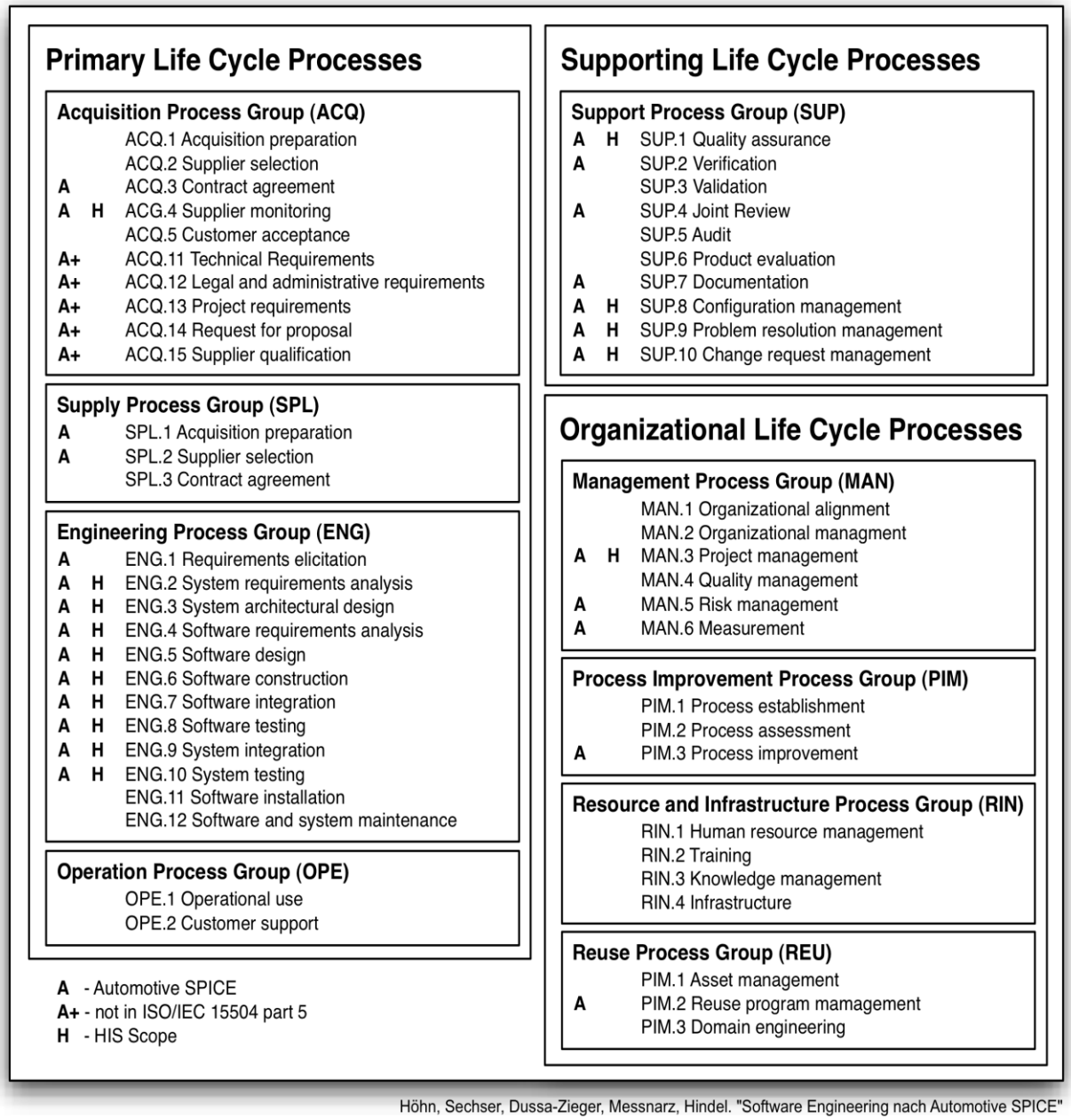

Figure 2-6: Overview Automotive SPICE processes

The following text is an extract from the Automotive SPICE process reference model specification that describes the process groups:

- The Acquisition process group (ACQ) consists of processes that are performed by the customer, or by the supplier when acting as a customer for its own suppliers, in order to acquire a product and/or service.
- The Supply process group (SPL) consists of processes performed by the supplier in order to supply a product and/or a service.
- The Engineering process group (ENG) consists of processes that directly elicit and manage the customer's requirements, specify, implement, or maintain the software product, its relation to the system.

© The AMALTHEA consortium

- The Supporting life cycle processes category (SUP) consists of processes in the PRM that may be employed by any of the other processes at various points in the life cycle.
- The Management process group (MAN) consists of processes that contain practices that may be used by anyone who manages any type of project or process within the life cycle.
- The Process Improvement process group (PIM) consists of processes performed in order to define, deploy and improve the processes performed in the organizational unit.
- The Reuse process group (REU) consists of processes performed in order to systematically exploit reuse opportunities in organization's reuse programs.

2.5.2.2. Automotive SPICE - Process Assessment Model (PAM)

The process assessment model (PAM) defines assessment indicators for the performance and capability of the aforementioned Automotive SPICE processes. The following capability levels are defined:

- **Level 0: Incomplete process**
  The process is not implemented, or fails to achieve its process purpose. At this level, there is little or no evidence of any systematic achievement of the process purpose.

- **Level 1: Performed process**
  The implemented process achieves its process purpose.

- **Level 2: Managed process**
  The previously described Performed process is now implemented in a managed fashion (planned, monitored and adjusted) and its work products are appropriately established, controlled and maintained.

- **Level 3: Established process**
  The previously described Managed process is now implemented using a defined process that is capable of achieving its process outcomes

- **Level 4: Predictable process**
  The previously described Established process now operates within defined limits to achieve its process outcomes.

- **Level 5: Optimizing process**
  The previously described Predictable process is continuously improved to meet relevant current and projected business goals.

The PAM defines a set of base practices for each process that need to be performed in order to demonstrate at least capability level 1. Additionally, generic practices are defined for each capability level which allow measuring the performance of the processes.

 © The AMALTHEA consortium

2.5.3. Relevance

In the automotive domain suppliers are often required to prove that the processes used during a given development project are performed according to Automotive SPICE level 2 or even level 3. Proving the bidirectional traceability from requirements via architecture, design and implementation to tests is often considered to be a challenge.

2.5.4. References

[AutoSpice1] Automotive SIG. "Automotive SPICE® Process Assessment Model (v2.5)". 2010-05-10. URL: http://www.automotivespice.com/web/download.html
[AutoSpice2] Automotive SIG. "Automotive SPICE® Process Reference Model (v4.5)". 2010-05-10. URL: http://www.automotivespice.com/web/download.html
[Hoehn3] Höhn, Sechser, Dussa-Zieger, Messnarz, Hindel. "Software Engineering nach Automotive SPICE. Entwicklungsprozesse in der Praxis - Ein Continental-Projekt auf dem Weg zu Level 3". dpunkt.verlag. Heidelbert. 2009.

2.6. AUTOSAR

2.6.1. Scope

The AUTOSAR scope includes all vehicle domains. AUTOSAR focuses on body, power train and chassis domains first. All vehicle control applications are addressed, particularly with respect to distributed functions (e.g. via busses). In addition to the automotive domain, products that are derived from a commercially available and proven in use automotive AUTOSAR product may be applied in technical fields such as railway, agriculture and forestry machinery, construction machinery, compressors, pumps or power generators and marine including military transportation vessels. Please note, that the following fields are explicitly excluded from the AUTOSAR scope: aviation, aerospace, nuclear power, chemical and biological reactors and in the petrochemical and military sectors and more generally for ultra hazardous applications.

2.6.2. Overview

The Automotive Open System Architecture (AUTOSAR) is a model-based design approach commonly used for automotive software development. It has been developed by leading automotive manufacturers and suppliers. AUTOSAR provides standardized functional interfaces that enable different suppliers and OEMs to develop their vehicle applications more independent of the hardware. Therefore, on the highest level of abstraction the AUTOSAR specifications distinguish three software layers that run on a microcontroller:

- Application
- Runtime Environment (RTE)
- Basic Software (BSW)

© The AMALTHEA consortium

In the AUTOSAR specification, all the modeling elements are defined in the AUTOSAR meta-model. On the top-level, applications are described through software components, each encapsulating a certain piece of functionality, and connections among them. These software components include so called runnables (often a C-function), which can be seen as the smallest unit of code that can be scheduled.

To allow hardware independent development of the application, the communication between the different software components as well as the communication between a software component and the hardware is made transparent by the RTE. The underlying modeling concept in AUTOSAR is called virtual functional bus (VFB). To avoid a mapping of applications to the hardware in early design stages, all communications are modeled the same way, i.e. regardless of whether it is inter- or intra-ECU communication. In this way system developers can model applications and their connections independent of the technology on that they are running at the end of the development process.

Finally, when the SWCs have been mapped to the ECUs, the RTE implements the VFB-functionality used by a SWC on the specific ECU. Therefore, the RTE uses the BSW, which provides infrastructural functionality on the ECUs. The BSW itself consists of different BSW-modules, which might be hardware dependent, that for example contain services such as an operating system and bus communication (cf. [AR_Homepage]).

With AUTOSAR Release 4.0 additional concepts that are highly relevant for the Automotive Industry have been added to the standard. Those concept include

- Timing Extension
- Multicore
- Traceability of Requirements
- Variant Handling

2.6.2.1. Timing Extensions

As already mentioned above the AUTOSAR methodology provides several well-defined process steps and with those a set of artifacts that are provided or needed. With AUTOSAR Release 4.0 an additional methodology focusing on the timing specification was introduced. Therefore depending on the availability of necessary information, the role a certain artifact is playing and the development phase five different views were introduced:

 © The AMALTHEA consortium

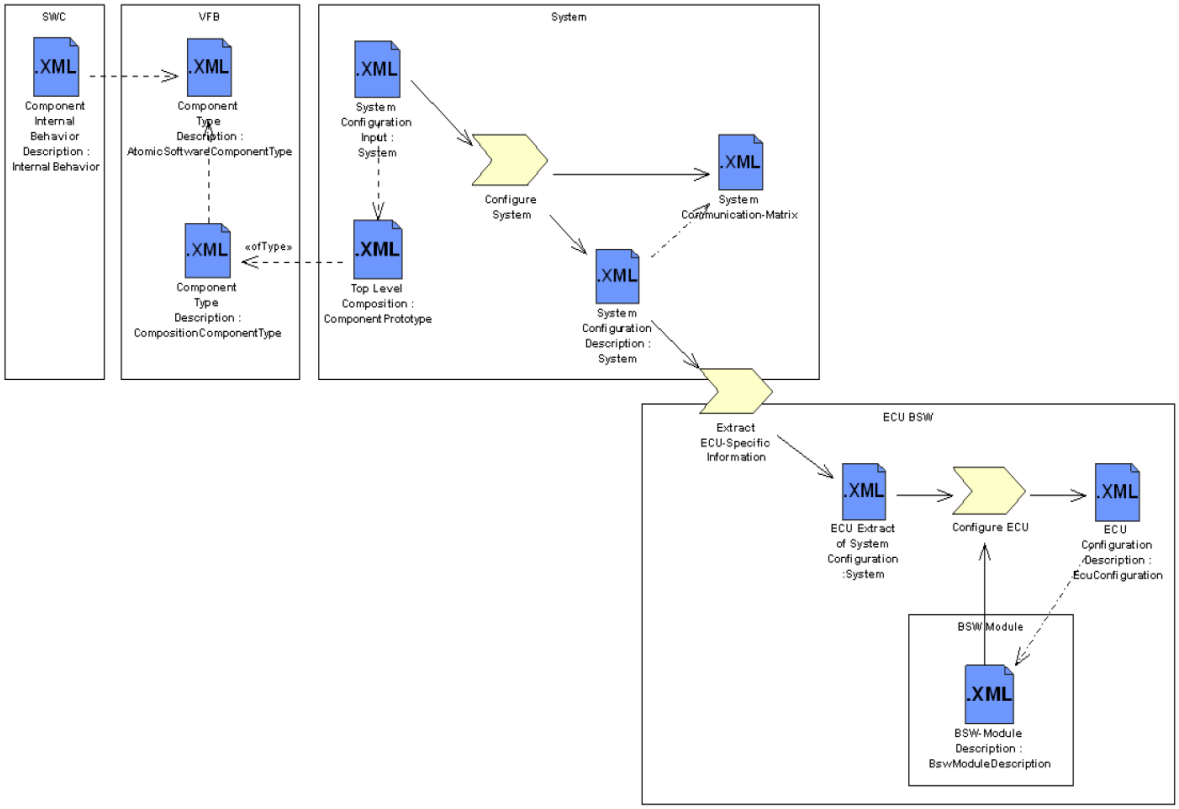

Figure 2-7: Overview of the AUTOSAR methodology and timing specification [6]

- VfbTiming
  This view deals with timing information related to the interaction of SwComponentTypes at VFB level meaning they only refer to SwCmponentTypes, PortPrototypes and their connections. Typically, real end-to-end timing constraints are captured in this view.
- SwcTiming
  This view deals with timing information related to the SwcInternalBehavior of AtomicSwComponentTypes, which specify the component's behavioral decomposition into RunnableEntities. Thus constraints can refer to the activation, start, and termination of the execution of RunnableEntities.
- SystemTiming
  This view deals with timing information related to a System, utilizing information about the mapping of software components to concrete target hardware, where local and remote communication has to be considered.
- BswModuleTiming
  This view deals with timing information related to the BswInternalBehavior of BSW modules. Similar to the SwcTiming view, constraints can refer to the activation, start and end of the execution of BswModuleEntities.
- EcuTiming
  This view deals with timing information related to the EcucValueCollection. Constraints refer to isolated BSW modules as well as to the inter BSW module relations on one specific ECU.

To be able to capture the observable behavior within a system at a certain point in time, AUTOSAR extends the formal basis of events by TimingDescriptionEvent. Additionally TimingDescriptionEventChain is introduced to relate timing events to

each other. Based on those events and event chains, it is possible to express various specific timing constraints. Depending on the concrete view, different types of timing events and thus event chains, as well as different timing constraints are available to specify the timing behavior.

2.6.2.2. Multicore

The main motivations for multi-core in the automotive domain are functional safety and performance.

1. Functional safety:
    - a multi-core processor can run in the so-called lock-step mode. In this mode, two cores are executing the same software and the results are compared. The system can be operated with existing software and behaves like a single processor from the outside view.
    - a multi-core processor can support partitioning between independent applications with different ASILs. If each application uses it's own core(s), temporal interference can be avoided.
2. Performance: To increase the overall performance the software has to be distributed to different cores that execute parts of the software in parallel.

In AUTOSAR R4.0.1, multi-core support for the system was introduced as optional part of the standard. This release mainly describes the multi-core extensions to the operating system and gives hints to the overall architecture.

2.6.2.2.1. Software extensions

The basic mechanisms are built into the AUTOSAR operating system (OS) and consist of the following features:

- **OS applications** are the basis for the mapping of application code to cores. This means that all objects that pertain to the same OS application (tasks, ISRs, alarms, etc.) are placed on the same core.
- The **scheduling policy** has not changed, but each core has its “own” (local) scheduler which selects the (local) tasks to be run.
- Cross-core TASK activation and setting of EVENTS across cores is supported.
- Communication between cores is implemented with an **Inter OS-Application Communication** (IOC).
- The IOC is based on the principles of (queued / unqueued) messages and can be used from the RTE. Its internal functionality is closely connected to the Operating System.
- For synchronization between basic software (BSW) modules or complex device drivers (CDD), spinlocks were introduced. Spinlocks can be used to build critical sections, which allow protecting resources which are accessed from different cores.
- A **spinlock** is a locking mechanism where the TASK waits in a loop ("spins") repeatedly checking for a shared variable that indicates whether the lock is free or not. In Multi-Core systems the comparison and changing of the variable typically requires an atomic operation. As the TASK remains active but is not doing anything useful, a spinlock is a busy waiting mechanism.

 © The AMALTHEA consortium

This first application of a multi-core system only enforces minimal changes to the single-core system. Therefore the BSW is declared "safe for single-core use only", and force the system to execute the BSW (with few exceptions like OS and ECU state manager) always on the same core, called the BSW or master core.

The challenge for AUTOSAR projects is the mapping of applications and basic software to the different cores. The mapping decides whether it is possible to use the full performance of such a multicore MCU.

2.6.2.2.2. Hardware assumptions

CPU Core features

- The hardware supports atomic read and atomic write operations for a fixed word length depending on the hardware.

Memory features

- Shared RAM is available to all cores; at least all cores can share a substantial part of the memory.
- Flash shall be shared between all cores at least. However, performance can be improved if Flash/RAM can be partitioned so that there are separate pathways from cores to Flash.
- A single address space is assumed, at least in the shared parts of the memory address space.
- The AUTOSAR Multi-Core architecture shall be capable to run on systems that do and do not support memory protection. If memory protection exists, all cores are covered by a hardware based memory protection.

2.6.2.2.3. Software assumptions

Decisions made in AUTOSAR R4.0:

- The scheduling algorithm does not assign tasks dynamically to cores.
- The AUTOSAR OS RESOURCE algorithm is not supported across cores.
- Interrupts can only be disabled on the same core.
- All BSW modules that would be present on a Single-Core system usually reside on the master core.

2.6.2.2.4. Next steps

The definition of Multi-Core concepts is going on. AUTOSAR R4.0.4 already includes new concepts like “Enhanced BSW allocation in partitioned systems“ (Concept #24) and the “Definition of resource locking behavior” (Concept #26). Other concepts like the "Distribution / splitting of SW-Components” (Concept #25) are still under discussion and are postponed to later releases.

2.6.2.3. Requirement and Traceability

In order to improve the overall consistency of the AUTOSAR standard documents and in order to follow the requirements of bidirectional traceability between artifacts

within the development cycle, AUTOSAR has integrated a formal approach for establishing traceability between requirements, use cases, specification items, etc. of the AUTOSAR standard documents.

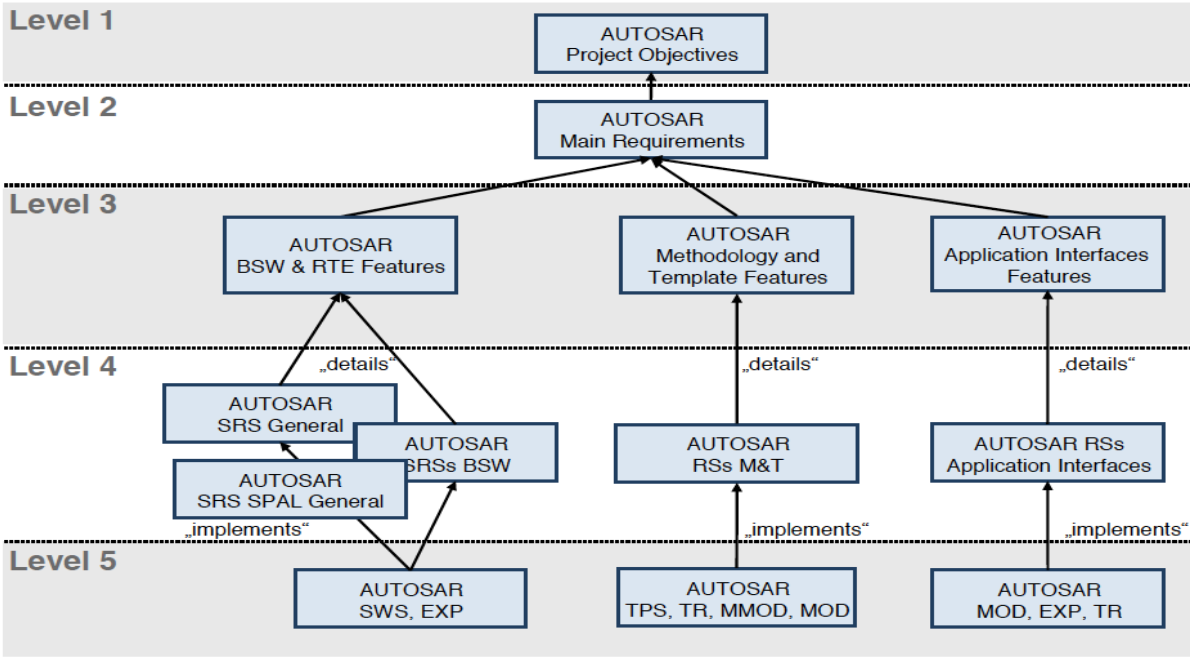

Figure 2-8: AUTOSAR traceability concept

This traceability concept is highly optimized for traceability between items within the AUTOSAR standard specifications and does not replace more general requirements management and traceability solutions that e.g. allow adding custom attributes for requirements or support following the life of a requirement through the complete development process.

2.6.2.4. Variant Handling

The motivations for Variant Handling in AUTOSAR are to build a bridge between OEM's and suppliers, to avoid redundancy between artifacts, and to provide a basis for expressing basic product lines in AUTOSAR. Of course, variant handling concepts do already exist at most companies, but they are typically not standardized (beyond company borders), and thus it is difficult for OEM's and suppliers to talk to each other on this subject. Consider the following example. An OEM sends a model which contains variants to a supplier. The supplier generates code from this model, but does not resolve all variants. What the OEM gets back is object code with some variants bound, and other variants left "open" for binding at load time. This can only work if both parties speak the same language, and have the same understanding about variants. And quite often, more than two parties are involved.

Hence, variant handling in AUTOSAR is mostly about **documenting** variants:

 © The AMALTHEA consortium

- **Variation Points** are locations in the model that are variable. That is, they may not exist in all variants, or may have different characteristics in different variants.
- The **Binding Time** is the latest possible time when a variation point may be bound.
- **Binding Expressions** specify under which condition(s) a variable element exists, or determine certain variable characteristics.

Since AUTOSAR Release 4.0, the concept of variant handling is integrated in the AUTOSAR DSL. In order to reduce complexity of the variant handling concept, the binding expressions refer to a set of central system constants. A specific variant is selected by assigning values to these system constants.

The AUTOSAR variant handling concepts allows for exchanging information about variability in AUTOSAR models between the different stakeholders within the system development process. However, it doesn't provide a solution for managing the general question about how to manage variability and how to define an architecture of a good product line.

2.6.3. Relevance

The AUTOSAR core partners (BMW, Bosch, Continental, Daimler, Ford, GM, PSA, Toyota, Volkswagen) are already using AUTOSAR or are planning to adapt to AUTOSAR in the near future. In 2016 more than 200 Million AUTOSAR based ECUs are expected on the market (see Figure 2-9).

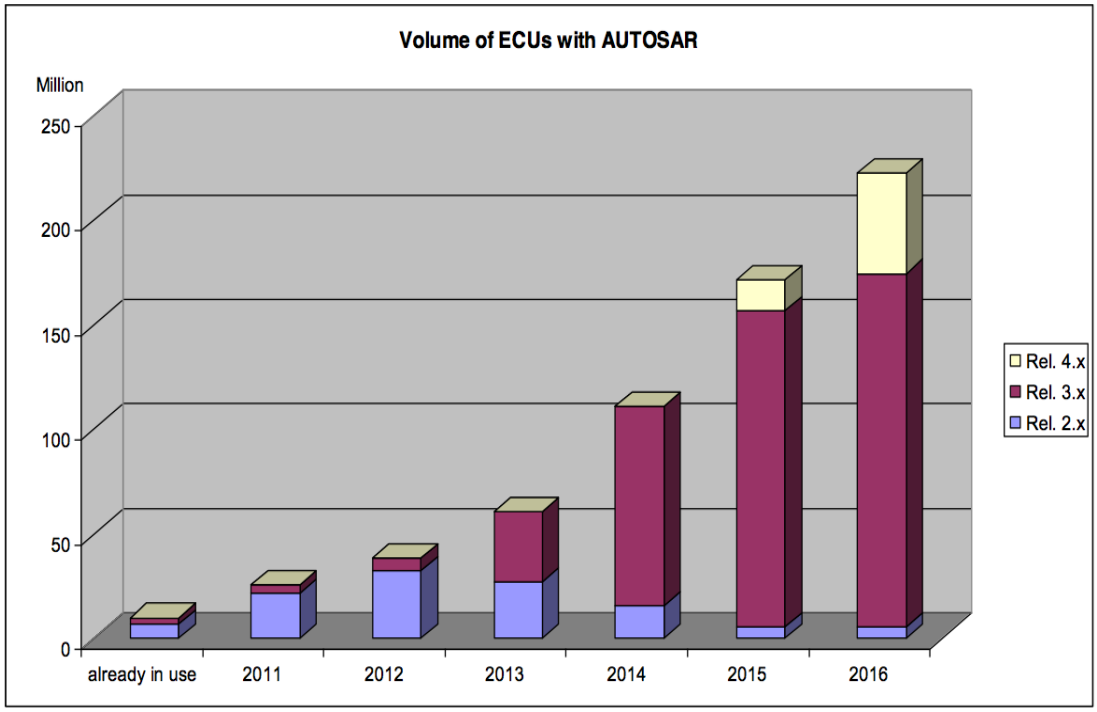

Figure 2-9: Volume of ECUs with AUTOSAR

2.6.4. References

[AR_HOMPAGE] AUTOSAR, "Homepage", http://www.autosar.org
[AR_FAQ] AUTOSAR, "FAQ", http://www.autosar.org/index.php?p=1&up=6&uup=0
[AR_DERIVED_APPS] AUTOSAR, "Development Partnership AUTOSAR to extend scope of applications to non-automotive areas", http://www.autosar.org/download/media_release/Development%20Partnership%20AUTOSAR%20to%20extend%20scope%20of%20applications%20to%20non-automotive%20areas.pdf
[AR_EXPLOITATION] AUTOSAR, "Core Partner Exploitation Plans", http://www.autosar.org/download/conferencedocs11/04_AUTOSAR_CP_Exploitation_Plan_2011_OpenConf2011.pdf
[AR_TPS_GENERIC_STRUCTURE] AUTOSAR, "Generic Structure Template", http://www.autosar.org/download/R4.0/AUTOSAR_TPS_GenericStructureTemplate.pdf
[AR_TPS_TIMING] AUTOSAR, "Timing Extensions", http://autosar.org/download/R4.0/AUTOSAR_TPS_TimingExtensions.pdf
[AR_SWS_MULTICORE] AUTOSAR, "Multi-core OS", http://www.autosar.org/download/R4.0/AUTOSAR_SWS_MultiCoreOS.pdf

 © The AMALTHEA consortium

3. Methodology, Concepts and Best Practices

3.1. Activities for the overall development process

This chapter describes activities that are related to all phases of the software development process:

- Traceability
- Variant Handling

3.1.1. Traceability

Traceability is an essential activity within state of the art software development. It is used to e.g. ensure that the right artifacts are created at each phase of the software development life cycle, to track the progress during development and to reduce the effort required to determine the impact of requested changes.

3.1.1.1. Definition

Automotive SPICE and the IEEE Standard Glossary of Software Engineering Terminology define traceability as “The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another”.
CMMI defines it as “A discernable association among two or more logical entities such as requirements, system elements, verifications, or tasks.”

Forward traceability refers to tracing a source artifact (origin) to its resulting artifacts. E.g. trace each unique requirement forward into the design elements that implement that requirement, trace each design element to the code units that implement the design element or trace each code unit to the test cases that validate the code unit.

Backward traceability refers to tracing each unique artifact (e.g., architecture element, design element, code unit, and test) back to its origin, e.g. tracing each architecture element backward to the implemented requirements (the origins).

Bidirectional traceability refers to tracing in both directions (backwards and forward).

3.1.1.2. Traceability - a good practice

The process standard Automotive SPICE requires “Ensuring consistency and bilateral traceability” between subsequent work products within the development life cycle. Why is this important?

© The AMALTHEA consortium

Forward traceability ensures proper direction of the evolving product (that we are building the right product) and indicates the completeness of the subsequent implementation. For example, if a requirement can't be traced forward to one or more elements within the architecture, then the architecture is incomplete and the resulting product might not meet the requirements.

Additionally, if good forward traceability is maintained, changed requirements can be traced forward to the subsequent artifacts such as architecture elements, design elements, code units, test cases, etc. This helps determining the impact and effort for implementing the change request in the product. Furthermore, it helps keeping the overall system and its documentation in a consistent state. E.g. If an automotive instrument cluster that was developed for the European market is to be adapted for the American market, it needs to handle the speed in miles per hour instead of kilometer per hour. Forward traceability helps identifying all development artifacts including architecture and design specifications, code units, human machine interface layouts and tests that have to be updated. Incomplete modifications would most likely lead to defective software and inconsistent documentation.

Backwards traceability helps ensure that the evolving product remains on the correct track with regards to the original and/or evolving requirements (that we are building the product right).

The objective is to focus on the original scope of the product and to avoid adding unintended functionality (i.e., „gold plating"). Examples of scenarios that benefit from backwards traceability are:

- If during implementation a developer comes up with an innovative new solution, then the solution should be traced backwards to the original requirements in order to ensure that the solution still fits to the original scope of the project.
- If during development it turns out that an artifact cannot be traced backwards, then this might indicate that this artifact is not required or that the preceding artefact's are not complete and need to be updated. The overall impact of this change can again be determined using forward traceability.
- If a defect is identified, then backwards traceability can help identifying the root of the problem which might not only exist in the code unit but possibly already in the architecture specification or even in the requirements specification.

To summarize, typical use cases that highly benefit from traceability are:

- impact analysis
- coverage analysis (checking for completeness)
- determine project status
- help understanding the system

3.1.1.3. State of the art methodologies

Traceability is identified as a good practice for improving the quality within system and software development and is thus required by process standards such as

 © The AMALTHEA consortium

SPICE, CMMI and ISO 26262. Especially when applying agile methods where functionality is implemented in small increments, it is crucial to understand the impact of the new features on the overall system.

The classical methodology for implementing traceability is the construction of trace matrixes. A trace matrix contains one column for the source artifacts and one column for each resulting artifact. Using this approach, the trace information is kept as a separate work product during the development process.

Another approach for implementing traceability are trace tags. These tags usually identify the origin of each artifact. The tags are maintained directly within the development artifacts. This allows easy creation of backwards traceability. However, forward traceability requires additional tool support.

A precondition for the creation of a trace matrix and of using the trace tag approach is the unique identification of each traced element.

3.1.1.4. Challenges for implementation

The biggest challenge for implementing traceability from requirements via architecture, design and implementation to code and documentation is the maintenance of the trace information. Although some tools already have some built-in support for the creation, maintenance and analysis of trace information, these tools often do not cover the complete development life cycle and only have limited connectivity to traceable elements that are constructed in other tools. This results in high manual effort and poor quality of trace data.

3.1.1.5. Relevance

Today, this traceability is often only established for parts of he development process. Additionally, it is often created and maintained manually in late phases of the system development process in order to fulfill requirements imposed by conformity to Automotive SPICE, etc. Adding and maintaining the traceability early in the development process has high potential for increasing the understandability and consistency of the overall systems. This of course requires a solid underlying methodology and tool support that helps keeping the effort for maintaining the traceability as low as possible.

3.1.1.6. References

[WESTFALL] Linda Westfall, "Bidirectional Requirements Traceability", http://www.compaid.com/caiinternet/ezine/westfall-bidirectional.pdf

3.1.2. Variant Handling

Software products showing a high variability are difficult to manage because of different aspects. On the one hand it is a complex task to manage the lifecycle due to dependencies between different components, which may vary in each final product, on the other hand such a product will be further developed due to new requirements or improvements. To manage these aspects in the software development, software product lines [CLEMENTS2007] can be used.

© The AMALTHEA consortium

The central element for the representation of common and variable parts of a product line is one or more feature model [KANG1990, APEL2008]. Each feature model has many features, which represent stakeholder requirements or product characteristics [APEL2009, CZARNECKI2002, CZANRECKI2005]. These features are arranged hierarchically and may have dependencies to other features. Supplementary each feature has a semantic which indicates the meaning within a group (optional, mandatory, or, alternative). The feature itself is associated with a concrete software component or software artifact, so that it is possible to produce software based on a selection of features and their dependencies. A defined and valid selection of features represents one configuration (product) of the product line.

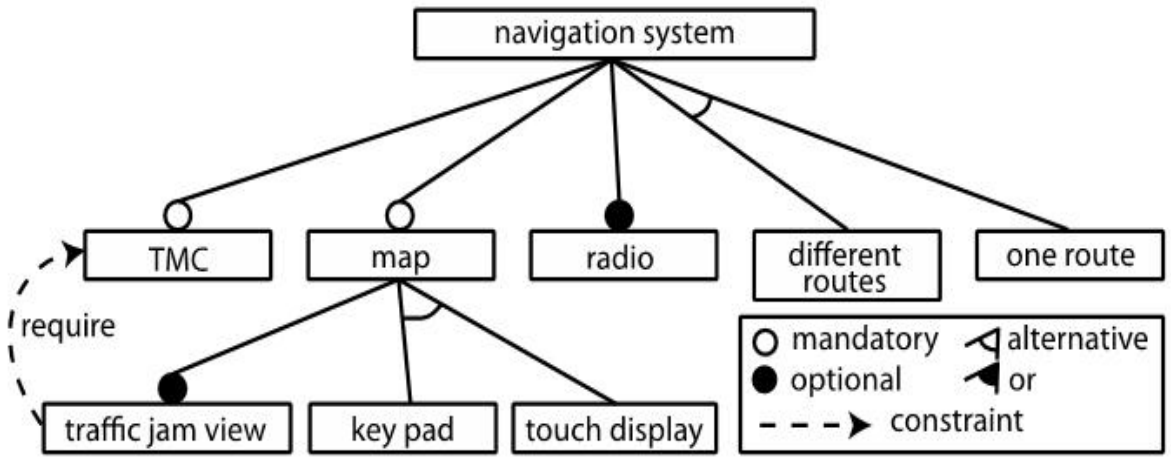

Figure 3-1: Feature model

Figure 3-1 shows an example of a feature model for the software of a (simplified) navigation system. The feature model let's you select for example, if you want the Traffic Message Channel (TMC) service to be included in your system or not. Based on this kind of description, many different features and their relationships of a configurable system can be developed and finally lead to a highly variable product. To handle the variability of the product and to support the development of products based on software features, the concept of feature models has been extended in numerous approaches [APEL2008, CZARNECKI2002, CZANRECKI2005, APEL2006, BENAVIDES2005]. Apart from the support to handle the variability of a system, a software product line can also be used to provide support for the quality management of a product. While the variability of a system brings challenges in the field of testing due to the large number of different products that can be created, the product line can also be used to provide different test cases according to the selected configuration.

3.1.2.1. References

[CLEMENTS2007] Paul Clements and Linda Northrop.*Software Product Lines: Practices and Patterns*, Addison Wesley, 6th edit, 2007.
[KANG1990] K.C. Kang. Feature-oriented domain analysis (FODA) feasibility study.*Report, DTIC Document*, 1990.
[APEL2008] S. Apel, T. Leich and G. Saake. Aspectual feature modules.In IEEE Transactions on Software Engineering, pages 162--180.IEEE Computer Society, 2008.

 © The AMALTHEA consortium

[APEL2009] Sven Apel and Christian Kästner.An overview of feature-oriented software development. _Journal of Object Technology (JOT), pages 49--84, 2009.
[CZARNECKI2002] Krzysztof Czarnecki, Thomas Bednasch, Peter Unger and Ulrich Eisenecker. Generative Programming for Embedded Software ,*An Industrial Experience Report*, Pages 156--172, 2002.
[CZANRECKI2005] Krzysztof Czarnecki, Chang Hwan and Peter Kim. Cardinality-Based Feature Modeling and Constraints :*A Progress Report*, 2005.
[APEL2006] S. Apel, M. Kuhlemann and T. Leich. Generic feature modules: Two-staged program customization. In Proceedings of the International Conference on Software and Data Technologies (ICSOFT), pages 127--132, 2006.
[BENAVIDES2005] D. Benavides and P. Trinidad. Using Constraint Programming to Reason on Feature Models.In The Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE'05), 2005.

3.2. Requirement engineering

3.2.1. Requirements traceability

The traceability of requirements through the overall development process is an essential activity for improving software development processes. More details about traceability is described in the Activities for the overall development process chapter.

3.2.2. Complexity

Complexity increases when system in question grows in size, in other words it includes more entities and connections between them. In software systems the reason for increasing complexity is more demands for richer functionality by customers along with shortening development times as functionality is needed quicker [CLARK2008]. The advancements in hardware and network technologies introduce more software-based systems in order to provide the services current society needs. This leads to ever-increasing complexity in embedded systems as they need to operate in an environment that can contain different devices like mobile phones, sensors, personal laptops and so on. And yet they need to be able to communicate between each other or share information (E-mail, phone calls, video streams..) [GRACE2005,ZHANG2005].

When the amount of stakeholders (often customers) rise, it can lead to tens of thousands of requirements increasing the complexity as the interconnections between these requirements and the sheer count of them makes their management and analysis a time consuming task. Regnell et al. [REGNELL2008] use the term "Very Large-Scale Requirements Engineering" (VLSRE) to describe this situation and identify three potential research opportunities to reduce the complexity:

- **Sustainable requirements architectures**. The information content and data structure of requirements should aim for simpler and sustainable form that can cope with the increasing amount of requirements. Often in VLSRE the sheer cumulative amount of the information from the requirements and the connections between these requirements grow and cause information overload. Therefore, the question of deciding what information is important

and what structure and format can sustain tens of thousands of requirements and connections between them becomes critical.

- **Effective requirements abstractions**. In VLSRE where interdependencies (prioritisation, resource estimation, change impact analysis...) among and between requirements are important, tens of thousands requirements with such connections between increase the complexity of the RE. In case of 20-50 requirements, finding all interdependencies is still possible but when the number increases to thousands, it becomes almost impossible. This problem can be tackled with abstraction mechanisms and experience-based heuristics by bundling the requirements and choosing the level-of-detail. Still, the problem lies on finding the actual interconnections, how to support humans to navigate and search in the RE database and how to display only relevant information in a huge database containing tens of thousands of requirements.
- **Emergent quality predictions**. In order to match the competitive market and large set of demanding stakeholders, fighting the over-scoping becomes a critical question in order to meet the quality expectations of the product. With a huge set of requirements, predicting the system level quality aspects becomes increasingly hard from the requirement's details and increase the risk of over-scoping in the platform development.

Even the with the advancement with development environments and software programming languages, leaning only to code-centric development and technologies when developing software systems proves to be difficult and costly [FRANCE2007]. MDD has arisen as one of the possible solutions for the complexity problem. It shifts the development from programming in solution space into modelling the problem space first. The purpose is to cover the underlying complexities in implementation platform from the developer as the platform itself can include varying amount of computers, networks, middleware and other entities and functionality needed for the system to work. Models therefore dictate the development in MDD; they are the main artefacts and the models are then transformed into the actual running systems [FRANCE2007]. If strong complexity management is required, it has been often shown that automation of mundane development tasks and support for concern separation is needed [FRANCE2006]. MDD tries to do just that by promoting the use of models and model technologies to increase the abstraction levels in development process. This would allow the process itself ot be less complex still maintaining enough formality to allow automation.

3.2.3. Timing

The influence of timing constraints in a system engineering process starts right at the beginning with the definition of user requirements. Thereby it is specified what the system should be able to provide. While functional requirements are used to define the specific behaviours of the system, additional non-functional requirements emerge. Those describe how a system should behave and are especially defined by the safety requirements of norms and standards which the system has to achieve. Additional timing constraints come along with the specification of the technical system architecture. When the specific hardware and the distribution of functionality is defined, real-time requirements are generated according to the information flow between all participating communication partners in order to ensure a timely interaction with a reactive environment.

 © The AMALTHEA consortium

Typically two different types of timing requirements are known: rate of execution and response time. The rate of execution deals with the event-to-event timing within a software function. This can be the timing between a new sensor value and the respective actuator output, time between samples of an input, or some combination of both. Response time, on the other hand, is the time between the first occurrence of an event and the time of the first response to that event. Thus those requirements determine how quickly a program must recognize an event and begin executing the appropriate software routine to handle it.

3.2.4 References

[CLARK2008] T. Clark, P. Sammut& J. Willans, “Applied Metamodelling: A Foundation for Language Driven Development”, Second Edition, Ceteva 2008.
[FRANCE2007] R. B. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in FOSE '07: 2007 Future of Software Engineering. Washington, DC, USA: IEEE Computer Society, 2007, pp. 37-54.
[FRANCE2006] R. B. France, S. Ghosh, T. Dinh-Trong, and A. Solberg, "Modeldriven development using uml 2.0: promises and pitfalls," Computer, vol. 39, no. 2, pp. 59-66, 2006.
[GRACE2005] P. Grace, G. Coulson, G. S. Blair, and B. Porter. Deep middleware for the divergent grid. In Proceedings of the IFIP/ACM/USENIX Middleware 2005, Grenoble, France, November 2005.
[ZHANG2005] J. Zhang and B. H. C. Cheng. Model-based development of dynamically adaptive software. In Proceedings of the IEEE International Conference on Software Engineering (ICSE06), Shanghai, China, May 2006.
[REGNELL2008] Regnell, B., Svensson, R., B., Wnuk, K. Can We Beat the Complexity of Very Large-Scale Requirements Engineering? Proceedings of the 14th international conference on Requirements Engineering (REFSQ 08), pp. 123-128, 2008.

3.3. Architecture & design

Software Architecture and Architecture description standard

Development of software system usually starts with elicitation of needs. Needs expressed by various stakeholders in this phase can be seen as a foundation on which a software system is developed. However, expressed needs can often be met with more than one design solution, which in turn can yield more or less valuable outcome. Each of those design solutions is determined with decisions made during the software architecting phase. Most often, Software Architecture (SA) is defined as the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time [GARLAN1995]. Beside this definition, literature offers many other interpretation of SA. Most of them can be found in [SWARCHITECTURE2012].

Development of SA is recognized as one of the technical processes defined by ISO 12207:2008 standard for “Systems and software engineering – software life cycle processes” and a significant process for AMALTHEA project. Because SA is abstract in nature and it always depends on domain, software type or on specific group or stakeholders, SA is not a candidate for standardization process. However,

conceptualization of SA is. International standard ISO/IEC/IEEE 42010:2011 for “Systems and software engineering – Architecture description” specifies the manner how SA descriptions are organized and expressed. It provides terms, concept definitions and motivation behind them thus enabling SA community to share the same conceptual framework for SA description. This International Standard addresses the creation, analysis and sustainment of architectures of systems through the use of architecture descriptions. Remaining text is based on ("Systems and software engineering – Architecture description," ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)) [ISO/IEC/IEEE 42010:2011]

Table 3-1: Basic terminology for software architecture

Term	Description
architecture	(system) fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution
architecture description	work product used to express an architecture (products of architecting)
architecture framework	conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders
architecture view	work product expressing the architecture of a system from the perspective of specific system concerns
architecture viewpoint	work product establishing the conventions for the construction, interpretation and use of architecture views to frame specific system concerns
concern	(system) interest in a system relevant to one or more of its stakeholders
environment	(system) context determining the setting and circumstances of all influences upon a system
model kind	conventions for a type of modelling
stakeholder	(system) individual, team, organization, or classes thereof, having an interest in a system

© The AMALTHEA consortium

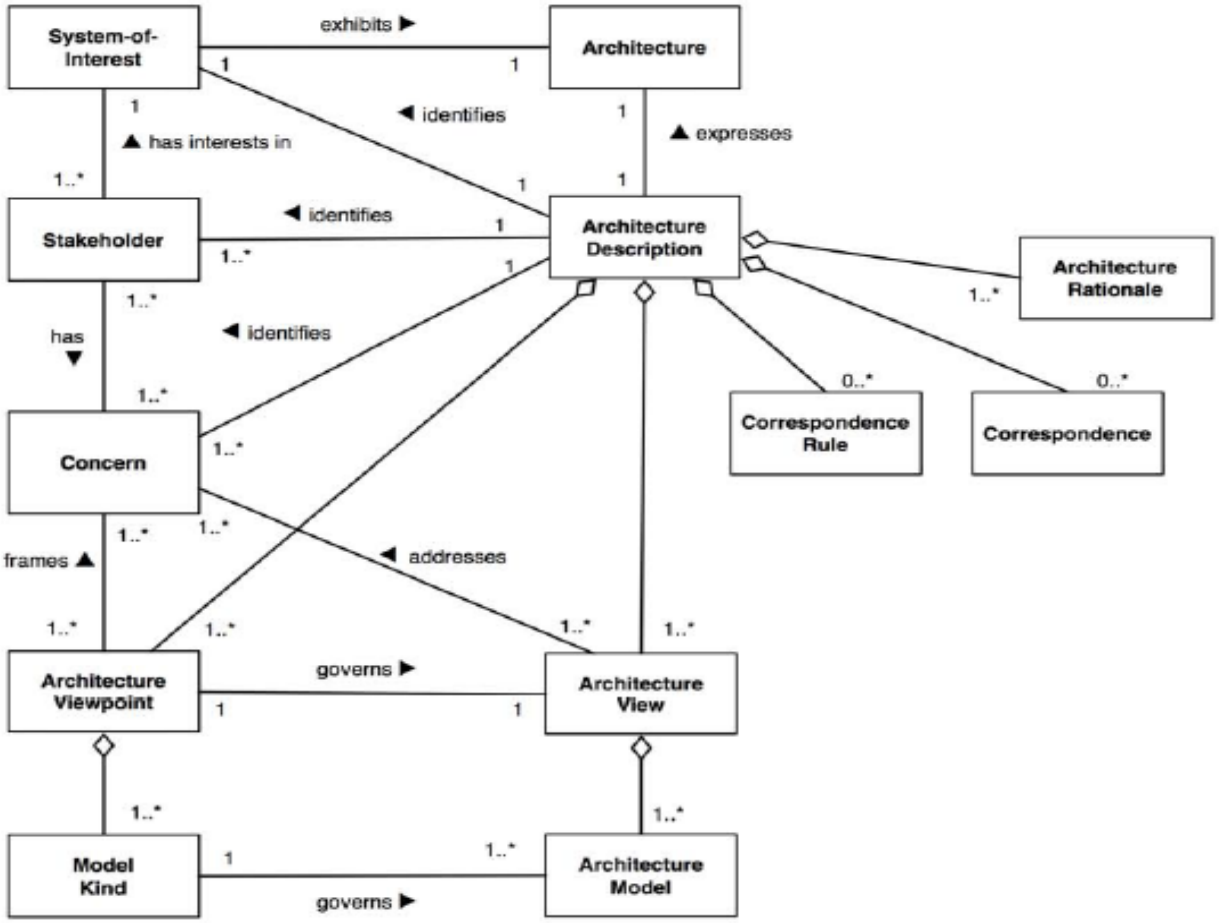

Figure 3-2: Conceptual model of ISO 42010:2011

This International Standard distinguishes an architecture of a system from an architecture description. Architecture descriptions, not architectures, are the subject of this International Standard. Whereas an architecture description is a work product, an architecture is abstract, consisting of concepts and properties. This International Standard specifies requirements on architecture descriptions. There are no requirements in this International Standard pertaining to architectures, or to systems or to their environments.

This International Standard does not specify any format or media for recording architecture descriptions. It is intended to be usable for a range of approaches to architecture description including document-centric, model-based, and repository-based techniques.

This International Standard does not prescribe the process or method used to produce architecture descriptions. This International Standard does not assume or prescribe specific architecting methods, models, notations or techniques used to produce architecture descriptions.

Two most often used ways to develop SA are frameworks (FW) and architecture description languages (ADL). For clarity reasons, it is a good practice to build FWs and ADLs on top of concepts that are defined in this standard. Purpose of the FW is to establish a common practice for creating, interpreting,

analysing and using architecture descriptions within a particular domain of application or stakeholder community. ADLs, on the other hand, are framing stakeholders concerns using one or more model kinds. More often than FWs, ADLs are supported with automated tool.

Table 3-2: Examples of Frameworks and Architecture Description Languages

Frameworks	**Architecture Description Languages**
Zachman Information System Architecture Framework	Rapide
TOGAF	Wright
Kruchten's 4+1 view	SysML
Siemens 4w	ArchiMate
RM-ODP	RM-ODP viewpoint language

3.4. Model-based design/Model-driven design

In [SELIC2006], B. Selic describes model-based design as an approach to software development that raises the level of abstraction compared to classical high-level programming languages, such as Java, C, C++ and makes extensive use of tool automation. Although these classical programming languages also have underlying models, designers have to deal with implementation issues that are irrelevant for solving the design problem. In the model-based approach, the goal is to use higher-level formalisms, abstracting from the unnecessary low-level details, which allow describing things directly in the high-level problem domain. The whole system developed finally consists of the set of models, all describing certain parts of the system, potentially with different levels of abstraction. To take advantage of the model-based approach, a tool chain is needed that allows automated code generation out of the whole set of models (system specification).

The whole set of models can be seen as a system (wide) model. This means, it serves as a reference whenever subgroups want to test/include new/improved parts of the system. Therefore, it allows the creation of the system among domain frontiers. Here, you can see another fundamental idea of model-based design: evolving systems. Experimentation, already possible in early design phases, helps taking the right decisions. The system then evolves from abstract descriptions by adding more and more details to the final system specification, which is found in an iterative way (and round-trips).

To achieve these advantages of model-based design, B. Selic specifies some requirements concerning the characteristics of a model (cf. [SELIC2006]):

- Abstraction
- Understandable (i.e. intuitive notation/expression)
- Accurate (i.e. aspects of interest have to be described correctly)
- Predictive (i.e. the model has to provide accurate predictions concerning the aspects of interest)

 © The AMALTHEA consortium

Models must be distinguished from their representation. A model itself is a theoretical construct whereas its representation is a description/visualization of the model. Hence there can be different representations, each highlighting another aspect of the same model.

Model-based design finally raises a number of challenges. Some questions that have to be answered are: When is a system completely described and which models are needed? Some glue might be needed that keeps all the models together. It is important to identify any ambiguous parts in the whole set of models. Another challenge comes with the tooling platform, as tools need to be adaptable for each domain and for different development steps. Therefore, well specified interfaces between tools and model-transformation rules have to be an integral part of a tool chain.

3.5. Safety/Security design

During the architectural and functional design, there are a lot of patterns available. This is the fact that no software projects begins really at zero. There are always existing software artefacts available and the new software project has to integrate it (best practice approach).

The following design principles will be considered in AMALTHEA:

- Economy of mechanism: use a simple design where possible
- Need-To-Know principle: allow user to access the system with a minimum access right and privilege needed to perform the requested service
- Fail-safe defaults: design no access by default and define rules, which allows user to have access according to their rights
- Open design: open the design of the system to a group of experts those can evaluate and improve the system design
- Usability: the functionality and particularly the handling of the system should be simply and ambiguity, the GUI should be clearly arranged and represented
- Defence-In-Depth principle: implementing several complementary security techniques at multiple system levels
- Redundancy principle: reduce the likelihood and impact of problems which occur due to excessive consumption of system resources like DoS

3.6. Resource mapping

The term resource mapping shall cover all aspects in the design and development process related to mapping software to software/hardware. This means that design decisions in the software development process with impact on the hardware/software mapping should be considered already in early stages of the development process. For example, there could be an early validation through low detail system simulation with the help of abstract hardware models. One important requirement for the mapping solution is that the non-functional requirements, meaning timing, performance and dependability constraints, of the application must be fulfilled by the solution. Tool support for this mapping and automation in generation of running systems through code generation helps designers to increase the efficiency of this job and keep track with different variants and the amount of possible configurations.

© The AMALTHEA consortium

The problem that finally has to be solved is to partition and allocate the executable pieces of software on the available processing cores, so that they can be scheduled by the OS without violating any timing constraints. However, additional goals for reaching an optimal mapping solution, such as a reliable solution which guarantees timing constraints also in case of underestimated resource efforts of different software parts, might have to be fulfilled. The final allocation cannot be made before all necessary information, which includes the application code as well as all hardware dependent software modules (in AUTOSAR this refers to the Basic Software modules), for the mapping is available. In the following subsections, an overview of the important steps related to target mapping, namely partitioning and allocation, scheduling and timing simulation, is given.

3.6.1. Model-based Partitioning and Allocation

Model-based partitioning is the mapping of software components into parts, which can be used by the model based allocation in order to allocate them to available scheduler of hardware elements, i.e. the cores in the case of a multicore processor. The benefit of allocating software parts first to a scheduler, allows using advanced global scheduling algorithms.

The goal of this two-staged, but iterative, process is to find a mapping which is feasible at minimal costs. Those costs can be for example the worst case execution time or CPU utilization. This however is a very complex problem and known to be NP-hard.
In general once the non-functional behaviour of the software components is defined at least in an abstract manner, parts of the software, i.e. functions or sub-functions, are assigned to a task or Interrupt Service Routine. At some stage, the execution times, communication costs, data and control dependencies of all the tasks become known and each task can be assigned to a specific scheduler, which i.e. manages a number of processors or cores. Based on this methodology it is possible to determine the actual costs by using a scheduling analysis or evaluation. This procedure maybe repeated until the mapping satisfies all requirements.

3.6.2. Scheduling

Solutions to the scheduling problem yield a splitting or fusion from tasks to time slots during that these are executed on a processor, or rather core (cf. [MARWEDEL2011]). The challenge is to find a mapping for a given set of tasks and their properties, such as deadlines, periods and execution times, on a given set of cores that allows an operating system scheduler to fulfil all the requirements and constraints of the application. In addition to timing constraints, there might be precedence constraints and resource constraints as well. Finding a solution to the scheduling problem is therefore highly related to the partitioning problem. Scheduling analysis can be used to guarantee that an operating system scheduler, following a certain scheduling algorithm, is able to keep all these constraints at run-time and that is optimal with respect to a specific criterion. Some characteristics of a schedule, which can be used as optimality criterion, are

- Max. lateness (max. delay of a task completion with respect to its deadline)

 © The AMALTHEA consortium

- Latency (time, that is spent for bus communication, memory access etc. during task execution)
- Throughput (average number of completed tasks per time unit)

Because some of these attributes are in conflict (e.g. throughput and latency), there is no universal best scheduling algorithm available. So, a feasible scheduling algorithm has to be chosen with respect to the specific system and its requirements. For example, at a hard real-time embedded system all tasks have to keep their deadlines. Hence, in this case the focus of a scheduling algorithm will be set less on maximizing throughput and stronger on minimizing the max. lateness. A scheduler needs processing time just as the considered task set. Hence, another aspect of scheduling that must be taken into account is the complexity and run-time behaviour of the scheduling algorithm, as scheduling causes computation-overhead by itself.

Well known scheduling algorithms of single-core systems are, for example, EDF (Earliest Deadline First) and Rate Monotonic (cf. [BUTTAZZO2011]). At multi-core systems, scheduling gets much harder than at single-core systems, because additionally to the aspects of a single-core system, for example, there must be considered the inter-core-communication as well as the access of shared memory by different cores. Therefore scheduling algorithms known from single-core systems need to be adapted or rather extended to work also at multi-core systems.

Multi-core scheduling approaches can be divided into two domains, namely global and local scheduling. Global scheduling captures the scheduling algorithms that allow migration, i.e. tasks are allocated to cores at runtime. An example of such an algorithm is EDZL (Earliest Deadline Zero Laxity, cf. [CIRINEI2007]), which is based on the already mentioned EDF.

In contrast, when using local scheduling tasks are allocated to cores already offline, i.e. migration of tasks is not allowed at runtime. Hence, there can be used single-core scheduling algorithms at each core of the multi-core system. However, to use local scheduling, a feasible allocation of tasks to the cores of the multi-core system is needed. Once again, this shows the high relation between the scheduling and allocation problem.

3.6.3. Model-based Timing Simulation

By model-based timing simulation we understand a discrete-event simulation of a given system model. Its goal is to provide the user a data basis for a clear statement about the timing behaviour of the system. Hardware and software objects are modelled by agents, whereas its behaviour is modelled by state machines. Each agent is able to register at a central component in order to get triggered at a certain time and sending an event to itself or to other agents. The interaction between the different agents is traced with timestamp and exact identification, in order to apply metrics on the events of the trace, as shown as example in the following Figure 3-3.

© The AMALTHEA consortium

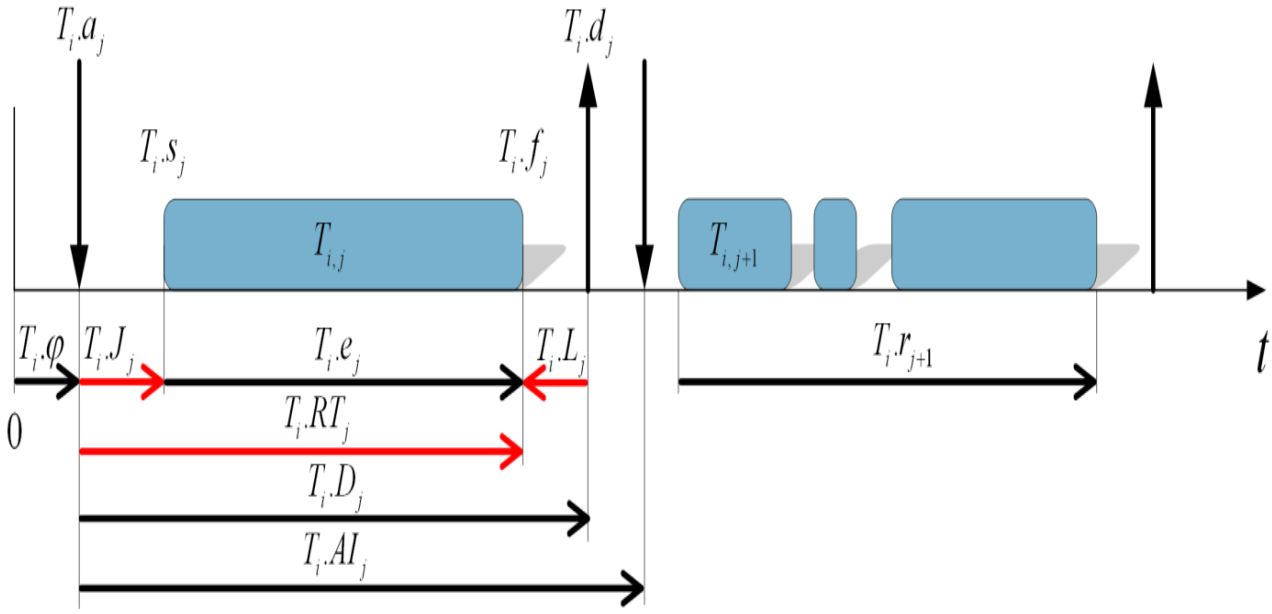

Figure 3-3: Example for time stamp tracing of interactions between different agents

Analysis values - result from the specification of the system or the stimulation:

- Activation Delay $T_i.\varphi$ is the timespan passed after the first activation of instance $T_{i,j}$
- Activation Interval $T_i.AI_j$ is the timespan between the activation of the instances $T_{i,j+1}$ and $T_{i,j}$. The Activation Interval $T_i.AI_j$ is determined for tasks with periodic execution $T_i.AI_j = T_i.AI$ in the design step or result from tasks activated by an event from their stimulation.
- Relative Deadline $T_i.D_j$ is the timespan relative to $T_i.a_j$ in which the instance $T_{i,j}$ has to complete

Recorded values - are clocked by use of profiling:

- Starting Point $T_i.s_j$ is the point in time, where the execution of instance $T_{i,j}$ is started
- Finalization $T_i.f_j$ is the point in time, where the execution of instance $T_{i,j}$ is completed

Determined values - can be calculated from the values above:

- Activation $T_i.a_j$ is the point in time, where instance $T_{i,j}$ gets activated. This is given for periodic tasks by $T_i.a_j = T_i.\varphi + (j-1)T_i.AI$. For tasks activated by an event, $T_i.a_j$ is the point in time, where the event is triggered
- Absolute Deadline $T_i.d_j$ calculates from $T_i.d_j = T_i.a_j + T_i.D_j$
- Execution Time $T_i.e_j$ is the timespan a task takes for calculation
- Runtime $T_i.r_j$ is the timespan a task takes from its starting point $T_i.s_j$ until its completion $T_i.f_j$, $T_i.r_j = T_i.f_j - T_i.s_j$, where $T_i.r_j = T_i.e_j$ holds only, if the task was not interrupted

Parameters of Real-time:
Release Jitter $T_i.J_j$, Response Time $T_j.RT_j$ und Lateness $T_j.L_j$

 © The AMALTHEA consortium

Metrics on the trace can be calculated online during simulation calculation time or after simulation by using the stored trace file. The results of this analysis can then be compared to the specified timing constraints of the system to verify whether the system satisfies the specified requirements in the simulated time frame or not.

3.6.4 References

[BUTTAZZO2011] Buttazzo, G. C.: Hard Real-Time Computing Systems – Predictable Scheduling Algorithms and Applications, Springer, 2011

[CIRINEI2007] Cirinei, M.; Baker, T.P.: “EDZL Scheduling Analysis”, in Proc. 19. Euromicro Conference on Real-Time Systems, Pisa, Italy, July 2007, pp. 9-18

[GARLAN1995] David Garlan, D. P. (1995). Introduction to the Special Issue on Software Architecture. IEEE Transactions on Software Engineering- Special issue on software architecture .

[ISO/IEC/IEEE 42010:2011] "Systems and software engineering – Architecture description," ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000). (n.d.). Standard: IEEESTD.2011.6129467 .

[MARWEDEL2011] Marwedel, P.; Teich, J.; Kouveli, G.; Bacivarov, I.; Thiele, L.; Ha, S.; Lee, C.; Xu, Q.; Huang, L.: „Mapping of Applications to MPSoCs,“ in Proc. 9. International Conference on Hardware/Software Codesign and System Synthesis, Taipei, Taiwan, October 2011, pp. 109-118

[SELIC2006] Selic, B.: "Model-driven development: its essence and opportunities," in Proc. 9. IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, Gyeongju, Korea, April 2006, pp. 313-319

[SWARCHITECTURE2012] Defining Software Architecture. (2012). Retrieved 2012, from Software Engineering Institute: http://www.sei.cmu.edu/architecture/start/definitions.cfm

3.7. Verification & validation

3.7.1. Methods for Model analysis and validation

For using a model as basis for the automatic execution of necessary activities within a development process, it must be assured the specification model represents all the test-relevant aspects for the composition of co-operating systems semantically-correct. This verification is essential for achieving trustworthy results from any model-depending, automatically executed activities.
Different validation and verification methods have been established in theory and practice for approving designated capabilities of a specification model. In general it can be distinguished between:

- •formal verification methods(static verification),
- •model based simulation (dynamic verification)and

- •model analysis based on given design patterns respectively templates (static verification).
- •syntactic validation and verification methods(static verification)

With methods of formal verification (e.g. model checking) defined characteristics of a specification model can be guarantied. Therefore, these methods are applied especially for approving documented evidence of security capabilities of a specification model. But especially applied to complex models formal verification methods can be very time and effort intensive.

In contrast to that with scenario-based simulation designated system capabilities can be investigated also in case of complex system models. However this validation is restricted to the considered simulation scenario, a guaranty of the considered system capabilities cannot be achieved by scenario-based simulation. With model-analysis methods based on given design pattern resp. templates basic, designated characteristics especially of the model syntax can be verified. Characteristics of the behaviour cannot be investigated with these methods. In fact good models are assumed to meet certain templates, which have been extracted from experiences with analog problem cases. Also the analysis of specification-models concerning a given syntax is a task of the model validation.

All V&V methods are classified as either dynamic or static. For reliable software projects, both dynamic and static methods are required. Static methods include semantic and syntactic verification, model analysis and metric calculations. Dynamic methods are all sorts of tests in small and large scales:

- Unit test,
- Module test,
- Integration test,
- System test,
- Functionality test,
- Stress test.

3.7.2. Test cases generation

The systematic generation of appropriate test-scenarios is a special challenge for complex test-objects, which is often hardly achievable. This complexity further increases if the test object is a distributed system with several components, in which the communication of these components is subject of the testing. For the creation of suitable test scenarios a systematic procedure is necessary fulfilling the requirements of high specification coverage of the test cases and small test costs (small number of test cases and/or test steps) for the test realization respectively. There are several mostly model based methods for the automatic creation of test cases. Basis of these methods is a model of the required behaviour, a specification model of the test object. Therefore it is necessary to verify that the specification model is semantically correct regarding the requirements, otherwise wrong test scenarios would be created with model based test generation methods.

In general model based test generation methods explore the specification model with different strategies in order to derive valid test scenarios. Thereby the most methods

 © The AMALTHEA consortium

don't operate on the whole state space of the specification model being able to generate test scenarios for complex specification models. With these approaches the specification model is stimulated with valid input data, the model is executed and the model reaction is observed. Valid input data are often deduced by genetic algorithms or intelligent heuristics operating on the specification model. This procedure is repeated until an abort criterion (e. g. specification coverage, max. execution time) is reached. High coverage criteria cannot be guaranteed with these test generation methods. This is only possible with test generation methods operating on the whole state space of the specification model. But with these methods often a high number of test scenarios are generated increasing the test costs. In practice it is necessary to choose the most suitable test generation method for the used test process and the specified requirements for the test cases. Here a trade-off between specification coverage by generated test cases and resulting test costs must be found.

3.7.3 Syntactic Verification And Validation Methods

Syntactic methods are essentially the opposite of the formal methods where the software is neither security nor safety critical. In model based systems' case, each model node's implementation basically involves the usage of abstract dependency graphs of the functions used in the software module.

Syntactic methods and formal methods can be applied simultaneously to the separate modules of the whole project. The choice is made depending on the balance between the cost and the robustness needs of each module. For instance, the low-level framework of the ECU has high security and security requirements; therefore it should be verified by using formal methods. But an integrated mobile application that retrieves the temperature inside the car could be verified by syntactic methods.

3.7.4. Timing validation

The correctness of embedded systems depends not only on the results of the implemented functionality, but also on their timeliness. Especially hard real-time systems have to meet rigid timing constraints, which are determined by the surrounding physical environment. Embedded systems repeatedly acquire data from the environment through a variety of sensors, process the data and finally respond via actuators. One of the biggest challenges however is to guarantee that the timing constraints of all executed tasks will be met taking account of a given processor architecture, varying communication times over a bus or interconnection network and the variable scheduling of tasks on the processing units. Thus the purpose of a timing validation is to formally analyse the defined model of a system to calculate its minimum and maximum timing. This is achieved by a static scheduling analysis.

© The AMALTHEA consortium

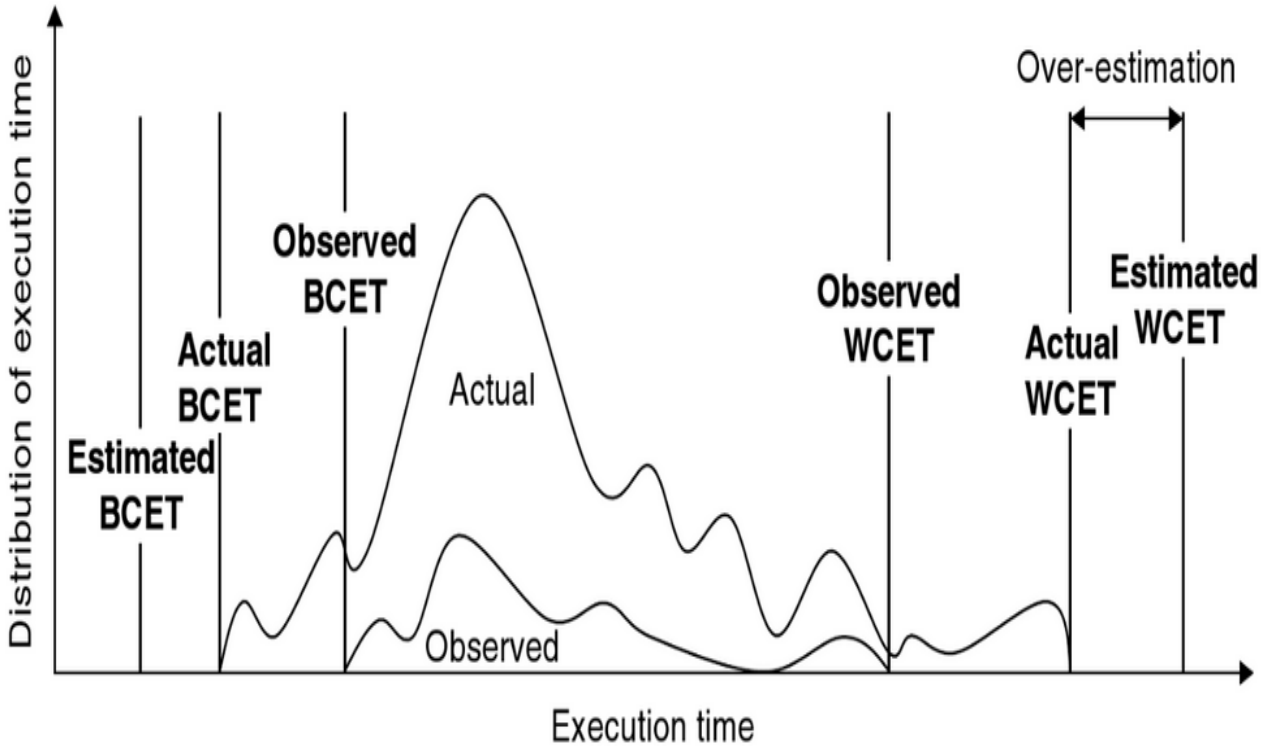

Figure 3-4: Estimated, actual, and observed WCET/BCET [1]

The goal of the static scheduling analysis is to guarantee for a system that all tasks are schedulable in a way that no timing requirements are violated regardless of the input. Therefore the worst-case response times (WCRTs) / best-case response times (BCRTs) of all tasks in the system based on their worst-case execution times (WCETs) / best-case execution times (BCETs), their periods, the chosen scheduling algorithm, etc. are determined. That way a safe overestimate is calculated that is greater than the actual WCET but never less and respectively smaller than the actual BCET but never greater. The resulting calculated interval includes consequently the real maximum and minimal timing. Finally, the WCRTs can be compared with the defined timing constraints to determine if they can be met under all conditions.

3.7.5. References

Roychoudhury, Abhik: Embedded Systems and Software Validation, 2009, Morgan Kaufmann Publishers, ISBN 978-0123742308

4. State of the art tools and frameworks

4.1. Selected tools and frameworks

4.1.1. Eclipse Community tools and frameworks

Eclipse is an open source community, whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. The Eclipse Foundation is a not-for-profit, member supported corporation that hosts the Eclipse

 © The AMALTHEA consortium

projects and helps cultivate both an open source community and an ecosystem of complementary products and services.

The Eclipse Project was originally created by IBM in November 2001 and supported by a consortium of software vendors. The Eclipse Foundation was created in January 2004 as an independent not-for-profit corporation to act as the steward of the Eclipse community. The independent not-for-profit corporation was created to allow a vendor neutral and open, transparent community to be established around Eclipse. Today, the Eclipse community consists of individuals and organizations from a cross section of the software industry.

4.1.1.1. Eclipse Modelling Framework (Modeling.EMF)

The EMF project is a modelling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

4.1.1.2. Eclipse Model-to-Model Transformation (Modeling.M2M)

Model-to-model transformation is a key aspect of model-driven development (MDD). The M2M project will deliver a framework for model-to-model transformation languages. The core part is the transformation infrastructure. Transformations are executed by transformation engines that are plugged into the infrastructure. There are three transformation engines that are developed in the scope of this project. Each of the three represents a different category, which validates the functionality of the infrastructure from multiple contexts.

4.1.1.3. Eclipse Model-to-Text Transformation (Modeling.M2T)

The Model to Text (M2T) project focuses on the generation of textual artifacts from models. Its purpose is threefold:

1. Provide implementations of industry standard and defacto Eclipse standard model-to-text engines
2. Provide exemplary development tools for these languages
3. Provide common infrastructure for this languages.

4.1.1.4. Eclipse Model Development Tools (Modeling.MDT)

The objectives of this project are to provide

1. an implementation of industry standard metamodels
2. and exemplary tools for developing models based on those metamodels.

4.1.1.4.1. Eclipse Sphinx (Modeling.MDT.Sphinx)

Sphinx provides an extensible platform that eases the creation of integrated modelling tool environments supporting individual or multiple modelling languages

(which can be UML-based or native DSLs) and has a particular focus on industrial strength and interoperability.

Features that are provided by Sphinx are:

- **Workspace Management** This component is built on EMF, EMF Transaction, and Eclipse Platform. It provides services for managing the lifecycle and editing domains of model instances that need to be centrally provided and shared in the workspace of Sphinx-based modelling tool applications.

- **Navigator View and Editor Sockets** This component is built on EMF, EMF Transaction, Eclipse Common Navigator Framework, Eclipse UI Forms, and GMF. It provides common logic for creating explorer views, form editors, and graphical editors operating on shared model instances in Sphinx-based modelling tool applications.

- **Validation Runtime Extensions** This component is built on EMF, EMF Transaction, EMF Validation, and Eclipse Platform. It provides extended runtime-level services for validating models or model fragments in Sphinx-based modelling tool applications and visualizing validation results.

- **Compare & Merge Integration** This component is built on EMF, EMF Transaction, EMF Compare, and Eclipse Compare Support. It provides extensions enabling model-based compare/merge operations to be carried out on shared model instances in Sphinx-based modelling tool applications.

- **EMF Runtime & Eclipse Platform Extensions** This component is built on EMF, EMF Transaction, and Eclipse Platform. It provides common runtime-level enhancements such as description of meta-models, description of shared model instances including their scopes, meta-model compatibility services, as well as a couple of performance optimizations and utilities. They are used by all other Sphinx platform components and are also available to Sphinx-based modelling tool applications.

4.1.1.4.2. Eclipse Requirements Modelling Framework (Modeling.MDT.RMF)

The Requirements Modelling Framework is an implementation of the Requirements Interchange Format, including a GUI.

4.1.1.4.3. Eclipse Unified Modelling Language (Modeling.MDT.UML2)

Eclipse UML2 is an EMF-based implementation of the UML 2.x metamodel for the Eclipse platform.

4.1.1.4.4. Eclipse Papyrus (Modeling.MDT.Papyrus)

Papyrus is aiming at providing an integrated and user-consumable environment for editing any kind of EMF model and particularly supporting UML and related modelling languages such as SysML and MARTE. Papyrus provides diagram editors for EMF-based modelling languages amongst them UML 2 and SysML and the glue required for integrating these editors (GMF-based or not) with other MBD and MDSD tools.

 © The AMALTHEA consortium

Papyrus also offers a very advanced support of UML profiles that enables users to define editors for DSLs based on the UML 2 standard. The main feature of Papyrus regarding this latter point is a set of very powerful customization mechanisms which can be leveraged to create user-defined Papyrus perspectives and give it the same look and feel as a "pure" DSL editor.

4.1.1.5. Eclipse Textual Modelling Framework (Modeling.TMF)

The Textual Modelling Framework is an umbrella project for textual modelling frameworks at Eclipse.

4.1.1.5.1. Eclipse Xtext (Modeling.TMF.Xtext)

Xtext is a framework/tool for development of external textual DSLs. Just describe your very own DSL using Xtext's simple EBNF grammar language and the generator will create a parser, an AST-meta model (implemented in EMF) as well as a full-featured Eclipse Text Editor from that.

The Framework integrates with technology from Eclipse Modelling such as EMF, GMF, M2T and parts of EMFT. Development with Xtext is optimized for short turnarounds, so that adding new features to an existing DSL can be done in seconds. Still with the new version more sophisticated programming languages can be implemented.

4.1.1.5.2. C/C++ Development Tooling (CDT)

The CDT Project provides a fully functional C and C++ Integrated Development Environment based on the Eclipse platform. Features include:

- support for project creation and managed build for various tool chains
- standard make build
- source navigation
- various source knowledge tools, such as type hierarchy, call graph, include browser, macro definition browser, code editor with syntax highlighting, folding and hyperlink navigation, source code refactoring, code generation
- visual debugging tools, including memory, registers, and disassembly viewers.

4.1.1.6. Relevance

Eclipse is a large, vibrant, well-established open source community with over 200 open source projects, close to 1,000 committers, 170-plus member companies, thousands of companies embedding Eclipse into products and applications, and millions of users. Eclipse began as a Java IDE but has evolved into a much larger and more diverse open source community. Eclipse has become a major destination for people involved in developing software that includes open source software [ECLIPSE_SURVEY2011]. Due to this great acceptance in the industry, Eclipse is of very high relevance for the implementation of the AMALTHEA tool chain.

4.1.1.7. References

[EMF] Eclipse, "Modelling Framework (EMF)", http://www.eclipse.org/modeling/emf
[M2M] Eclipse, "Model to Model Transformation (M2M)", http://www.eclipse.org/modeling/m2m
[M2T] Eclipse, "Model to Text Transformation (M2T)", http://www.eclipse.org/modeling/m2t
[MDT] Eclipse, "Model Development Tools (MDT)", http://www.eclipse.org/modeling/mdt
[RMF] Eclipse, "Requirements Modeling Framework (RMF)", http://www.eclipse.org/rmf
[SPHINX] Eclipse, "Sphinx (Sphinx)", http://www.eclipse.org/sphinx
[UML2] Eclipse, "Unified Modeling Language (UML2)", http://www.eclipse.org/projects/project.php?id=modeling.mdt.uml2#
[TMF] Eclipse, "Textual Modeling Framework (TMF)", http://www.eclipse.org/modeling/tmf
[XTEXT] Eclipse, "Xtext (Xtext)", http://www.eclipse.org/Xtext
[CDT] Eclipse, "C/C++ Development Tooling (CDT)", http://www.eclipse.org/cdt/
[ECLIPSE_SURVEY2011] Eclipse, "The OpenSource Developer Report, 2011 Eclipse Community Survey", http://www.eclipse.org/org/community_survey/Eclipse_Survey_2011_Report.pdf

4.1.1.8. Toolkit in OPen-source for Critical Application and SystEms Development (TOPCASED)

TOPCASED is a modular, open-source, Eclipse-based software environment providing methods and tools for critical embedded systems development, ranging from system and architecture specifications to software and hardware implementation through equipment definition. TOPCASED promotes model-driven engineering and formal methods as key technologies.

TOPCASED includes the following components:

- **TOPCASED - UML and SysML Editor** The TOPCASED UML and SysML Editor is moved from the TOPCASED project into the Eclipse Community and is now available as Eclipse MDT Papyrus.
- **TOPCASED - Requirements** provides an editor for managing requirements, import/export functionality as well as support for traceability between requirements and between requirements and model elements. TOPCASED is planning to use the Eclipse RMF for managing requirements. Note: The traceability is currently limited to editors that are based on the Papyrus backbone.
- **TOPCASED - Gendoc2** allows for generating text documents out of models. The generation is driven by templates described using MS Word or OpenOffice.
- **TOPCASED - Simulation** aims to simulate UML activity diagrams and state machine diagrams and to visualize the simulation in the diagrams.
- **TOPCASED – OCL Tools** provide an editor and a validation engine for OCL constraints.
- **TOPCASED – Model Transformation & Scripting** integrates support for model to model transformation and for scripting operations on models.

 © The AMALTHEA consortium

The following Figure 4-1 describes the Eclipse components which TOPCASED is built on and the TOPCASED components that are planned to be contributed to the Eclipse community in 2012-2014.

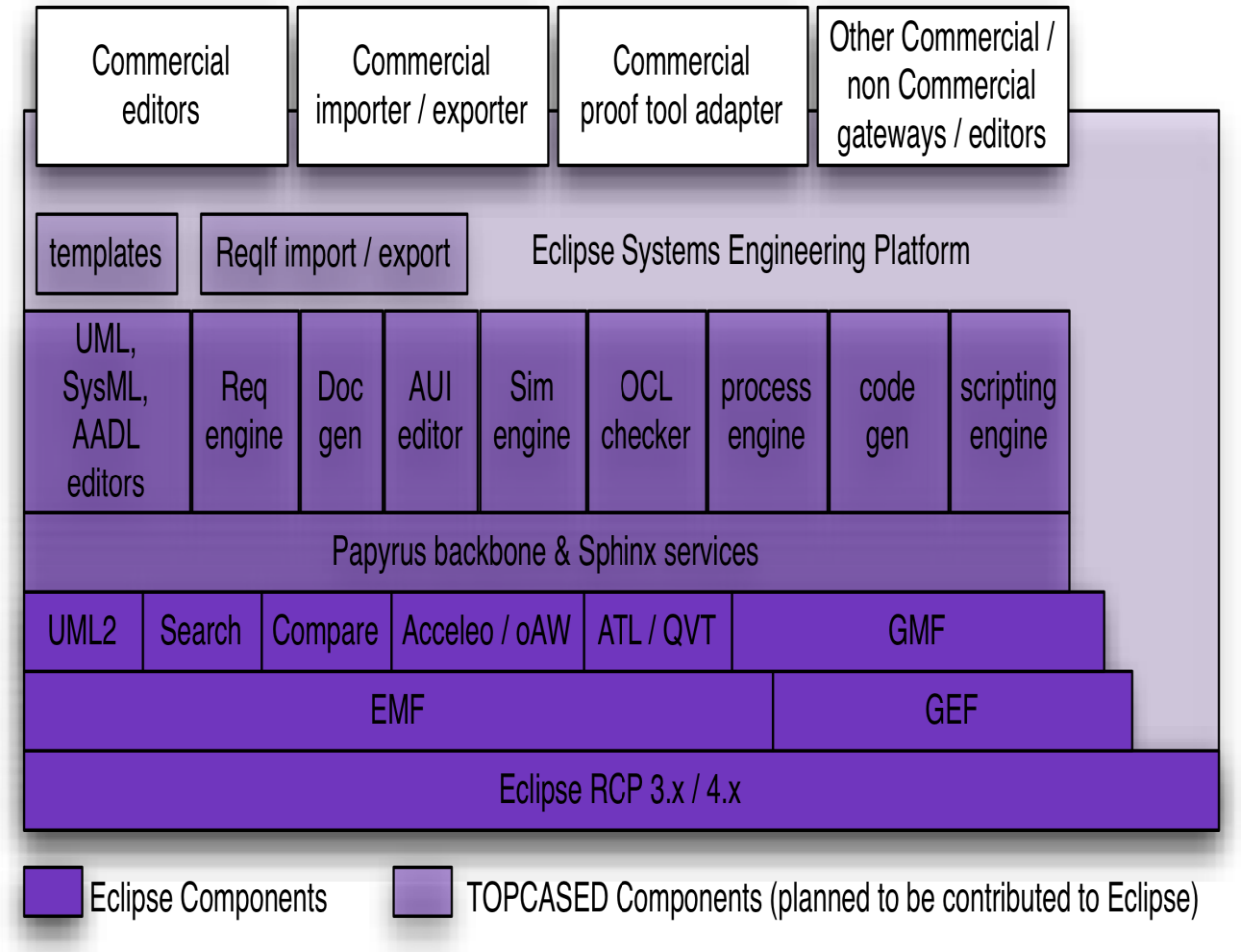

Figure 4-1: TOPCASED components structure

4.1.1.8.1. References

[TOPCASED_HOMEPAGE] TOPCASED Homepage, http://www.topcased.org/
[TOPCASED_STATUS2011] The TOPCASED status : components view and Industrial usage, http://gforge.enseeiht.fr/docman/view.php/52/4485/Eclipse+IDD+-+Berlin+2011+-+TOPCASED+presentation-ed2.pdf

4.1.1.9. AUTOSAR Tool Platform (Artop)

4.1.1.9.1. Overview

The AUTOSAR Tool Platform (Artop) is an implementation of common base functionality for AUTOSAR development tools. Artop, including its source code, is available free of charge to all AUTOSAR members and partners. The Artop development process is transparent and based on a community approach driven by AUTOSAR members and partners. The community that develops Artop is organized as the Artop User Group.

Artop is based on Eclipse, especially on EMF and Sphinx. The following AUTOSAR specific components are provided by Artop:

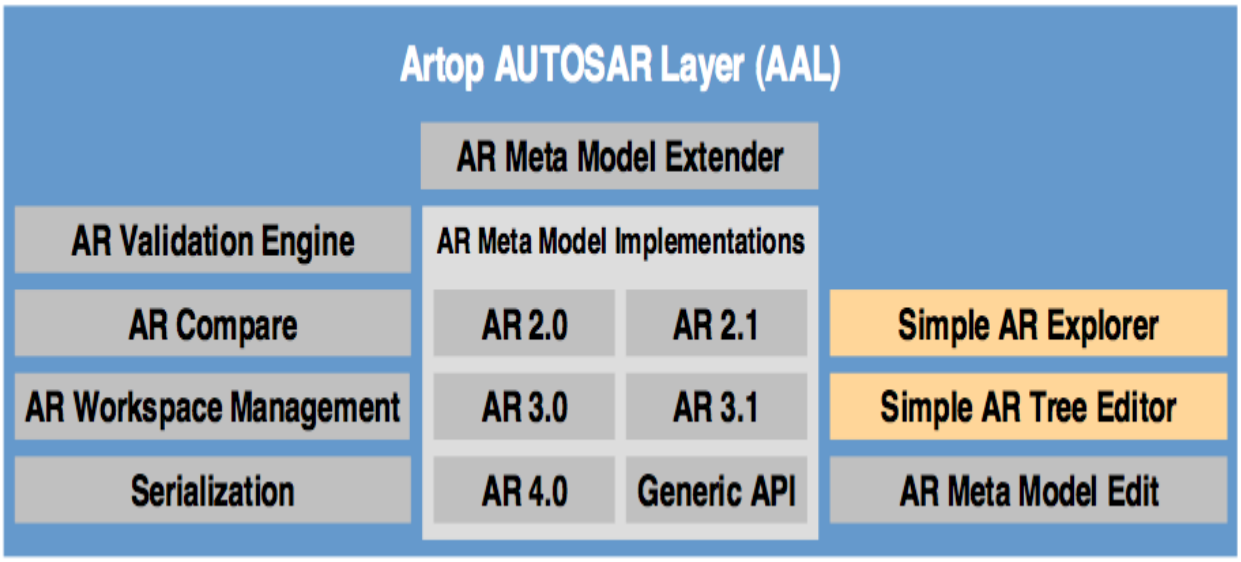

Figure 4-2: Problem Frame 6

- **AUTOSAR Metamodel Implementations** The most important part of the Artop platform is an implementation of the AUTOSAR metamodel. The metamodel is implemented using the Java programming language and the Eclipse Modelling Framework (EMF). Currently Artop contains support for AUTOSAR release 2.0, 2.1, 3.0, 3.1, 3.2 as well as AUTOSAR release 4.0. Support for upcoming AUTOSAR metamodel releases are added with later versions of Artop. The goal is to provide metamodel implementations in Artop close to the point in time when AUTOSAR releases them. Artop also provides a Generic API Metamodel API that allows developing plug-ins which process or manipulate AUTOSAR models of different AUTOSAR releases with one common implementation. The Generic API is provided for certain commonly used areas of the metamodel. Naturally the Generic API cannot cover the full metamodels.

- **AUTOSAR Serialization** The Serialization component provides file-based persistence for AUTOSAR models. It allows for serializing and de-serializing AUTOSAR models to and from AUTOSAR XML files, based on the XSD schema defined by AUTOSAR.

- **AUTOSAR Workspace Management** The AUTOSAR Workspace Management supports managing of AUTOSAR models, which are spread over more than one AUTOSAR XML file. It integrates into the Eclipse workspace and provides the capability to load and save various file formats.

- **AUTOSAR Metamodel Edit** Metamodel Edit helps to create user interfaces on top of the AUTOSAR model by enabling in-memory AUTOSAR models to be displayed and modified. It is based on the EMF.Edit framework and provides key features like content and label provider classes, property source classes and a non-transactional command framework to create, modify and delete model elements.

 © The AMALTHEA consortium

- **AUTOSAR Validation Engine and Constraints** Artop contains a generic validation engine that minimizes the effort to create validation constraints for AUTOSAR models. This component can validate constraints that correspond to the standard AUTOSAR constraints, as well as project specific or custom constraints that can easily be added via the extension point mechanism.

- **AUTOSAR Compare** Artop provides functionality that allows comparing AUTOSAR models. This allows to easily tracking changes between different versions of an AUTOSAR model file. The functionality is based on EMF Compare that additionally allows visualizing the changes between models.

- **AUTOSAR Example Components** Some example components, mainly UI components are also provided with Artop (orange components). These components show the usage of Artop, but are not meant to be used by an end-user. They will not be fully tested and documented and there is also no guarantee that the API of these components will be stable.

4.1.1.9.2. Relevance

Artop is the only open platform that provides common basic functionality for AUTOSAR development tools.

4.1.1.9.3. References

[ARTOP_HOMEPAGE] Artop Homepage, http://www.artop.org
[ARTOP_WHITEPAPER] Artop Whitepaper, http://www.artop.org/artop_whitepaper.pdf

4.1.1.10. YAKINDU

4.1.1.10.1. Overview

YAKINDU [YAKINDU] is an open-source-toolkit built on Eclipse for the model-driven development of embedded systems. Through the systematic use of models, it aims at an integrated development process and an increase in quality and maintainability. YAKINDU is not a monolithic application but a set of independent language modules. These modules currently comprise the following list:

- SCT - statechart tools
- Damos - data-flow oriented modelling
- Mscript - math oriented scripting
- CReMa - (requirements) traceability
- CoMo - component model

The language modules are not bound to any specific methodology. They are extendable and can be used in any domain specific tool chain independent from each other. As Figure 4-3 below shows, the only common basis of the language modules is a typesystem implementation with respect to the SI-units standard.

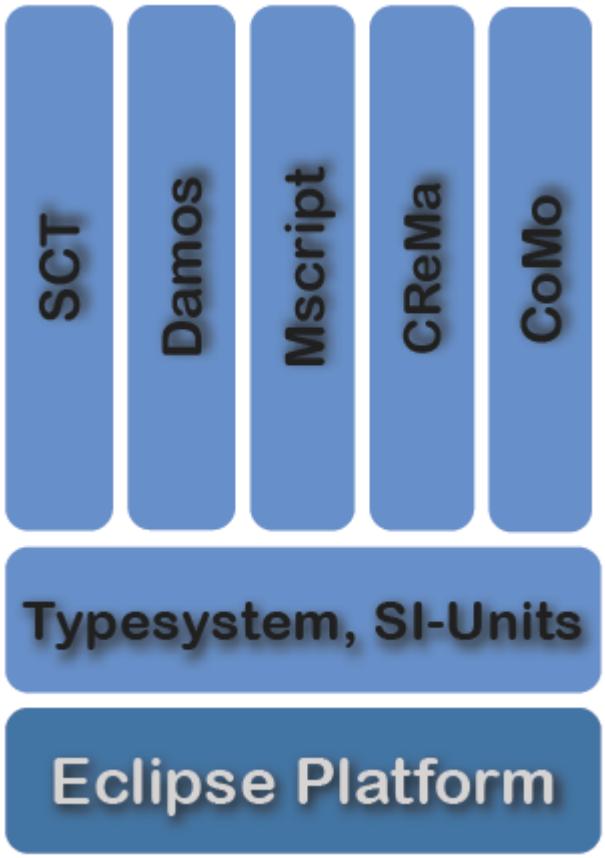

Figure 4-3: YAKINDU Architecture

The statechart tools (SCT) module is used to model reactive systems. They are designed to continuously interact with the environment the system is embedded in. Beside the fact that events from the environment asynchronously trigger the transitions of the statecharts, their core semantics is cycle driven. Thus, processing of concurrent events is enabled. A model interpreter is used to simulate the behavior of the statecharts. During simulation, transitions and current states are visualized and events from the embedding environment can be triggered by the developer. Various code generators exist to bring the statecharts onto the embedded device. Anyway, the model interpreter for simulation and the generated code follow the same core semantics.

In contrast to SCT, Damos [Ung] is a data-flow oriented modelling tool. It uses block diagrams as syntax to describe system component functions and data-flow connections between them. Data could be for example physical quantities like current or voltage. Damos is used to model control systems or digital signal processing. Damos supports a "two-dimensional" structuring of models. In the first dimension, subsystems can be used to hierarchically structure the models for enabling various implementations. A subsystem defines its provided interface. Subsystem realization models implement those interfaces. Thus, product line engineering for example is supported by implementing different realizations depending on the various product's specifications. The second dimension is established by system fragments which can be used to divide the model into multiple parts. For example, to support simulation of the model, it might be divided into a functional fragment and a simulation fragment. Thus, the "outer" world the model should be embedded in can be encapsulated into a

 © The AMALTHEA consortium

separate fragment which in turn can be removed once the model is brought into the productive environment.

The Mscript module is a mathematical DSL to define functions independently from the typesystem and its type ranges or type conversions. It can be used for example in Damos to model system component functions. The code generation maps the MScript functions to the concrete typesystem used on the embedding environment.

The typesystem the YAKINDU toolkit builds upon handles numeric datatypes by incorporating their units of measurement with respect to the SI-units standard. When no units are specified, a dimensionless value is assumed. Units are automatically converted and calculated. The typesystem is used among others for model validation and simulation.

The Cross Relational Manager (CReMa) is a tool that brings a tracing infrastructure into any development environment. It connects pairs of so called tracepoints to define traces between two development artifacts. A tracepoint binds an artifact of any kind - a piece of source code, a requirement, a document, a model element, etc. - to a trace. Artifacts might be assigned to multiple tracepoints and therefore to multiple traces. Thus, traces for example from a requirement specification via some model element to a piece of source code can be established to follow some specification to its implementation. CReMa manages these traces and provides a user interface to create and follow the traces. Its extendable architecture allows for implementation of new tracepoint providers. They are a non-invasive means to enable traceability also for other development artifacts. CReMa already ships with a requirements model based on ReqIf, an OMG standard for requirements specification.

The component model (CoMo) module is currently under development. It can be used to specify architectural elements of the system to be modelled. In the context of the YAKINDU toolkit, components might be implemented by statecharts or block diagrams. Anyway, the component model will be open to also be used by any other tool in a tool chain that incorporates YAKINDU.

4.1.1.10.2. Relevance

The statechart tools as well as Damos perfectly fit into the AMALTHEA tool chain. However, they still need to be extended for Multi-Core and product line engineering support. CReMa could be the basis for traceability in the AMALTHEA tool chain. Trace point providers can be implemented for each of the tools in the tool chain, thus, enabling traceability for the whole tool chain. It is not clear yet, if CoMo might be used in AMALTHEA, since AUTOSAR already defines a component model. Anyway, in the telecommunication domain CoMo might be useful.

4.1.1.10.3. References

[Ung] Andreas Unger, Damos Blog - http://andreasunger.com/

[YAKINDU] YAKINDU - http://www.yakindu.org/

4.1.2. Eclipse Industry Working Groups

4.1.2.1. Eclipse Automotive Industry Working Group

4.1.2.1.1. Overview

The Eclipse Automotive Industry Working Group defines and implements a standard platform for the software development tools used in the automotive industry. The main goals are:

- To provide an infrastructure for tool development required by the automotive industry
- To address and support the needs for the whole automotive software development cycle
- To avoid that the same non-competitive basic tool functionality is redeveloped over and over again
- To join forces and meet current and future requirements in terms of tool runtime performance and memory consumption

4.1.2.1.2. Relevance

The Eclipse Automotive Industry Working Group can be seen as a central gateway between projects and other activities within the Automotive Industry and the activities within the Eclipse Community.

4.1.2.1.3. References

[AUTO_IWG] Eclipse, "Automotive Industry Working Group", http://wiki.eclipse.org/Auto_IWG

4.1.2.2. Eclipse Polarsys

4.1.2.2.1. Overview

The Polarsys initiative was founded at the end of 2011. The goal of the Polarsys group is to collaboratively define, build and maintain open source tools for safety-critical and embedded system development in demanding engineering domains, such as aerospace, defence and security, transportation, energy, healthcare, telecommunications.

These domains typically require maintenance of tool chains for the very long term – from 30 to up to 50 to 70 years in some cases – which creates some very unique issues. Another characteristic of the working group is to address material for qualification processes – like in the DO 178 for Aircraft or ISO 26262 for Automotive.

Additionally, Polarsys intends to foster exchanges between academics and industrial partners in these domains and manage the maturity of tools from early research prototypes to the obsolescence of tools.

To implement this vision, the Polarsys group and the Eclipse Foundation will collaborate to provide services and to:

 © The AMALTHEA consortium

- Maintain the infrastructure required for the hosting of Polarsys projects, including software repositories, software build and test facilities.
- Proceed to IP due diligence in order to provide clean open source software released under EPL or other licenses validated by Polarsys such as BSD-like and LGPL.
- Define the software tools to be contained in each release of the tool chains (with specific roadmaps and release trains).
- Create a labelling process to recognize projects maturity and companies know-how and commitment .
- Create a software quality assessment kit to define the maturity level of each component.
- Digitally sign certified software releases made available under the Polarsys banner.
- Make those binary software releases available only to the members of Polarsys.

Polarsys focuses on techniques and tools to fulfil the Polarsys goals and vision, and mainly on:

- System, Hardware and Software Modelling
- Code analysis (static code analysis)
- Debugging, tracing and other integration tools
- Transversal process support tools : Configuration Management, Change tracking, Technical facts management, Project reporting
- Test and verification tools targeting embedded software methods, simulation and early validation
- Embedded components (RTOS, Middleware, ...)
- System on Chip simulation and Hardware logic (VHDL, SystemC, ...)

Polarsys will build on the technology and innovation created in the very successful TOPCASED open source project of the French cluster Aerospace Valley. Key contributors and source code from the TopCased project will be moved into the Polarsys working group.

4.1.2.2.2. References

[POLARSYS] Eclipse, "Polarsys Industry Working Group", http://www.polarsys.org/

4.1.2.3. IBM RATIONAL RHAPSODY

The Rational Rhapsody Developer from IBM is a commercial, UML/SysML-based model-driven development (MDD) environment focused on embedded and real time development. It provides:

- visual software development based on industry standard UML/SysML
- full behavioural code generation for C, C++, Java, Ada
- rapid prototyping
- visual debugging and animation of statecharts, activity and sequence diagrams
- round trip and reverse engineering
- integration with Eclipse development environment

- requirements traceability to design, code and test artifacts
- AUTOSAR 4.0 support for automotive applications
- MISRA-C and MISRA-C++ code to support automotive and safety related development process
- model multicore affinity, generate code targeting multicore processors and visualize multicore execution
- customizable documentation generation through ReporterPlus Add On
- automate testing and validation with Rational Rhapsody TestConductor Add On
- integration to requirements management solutions through the Rational Rhapsody Gateway Add On
- domain-specific language support for graphical C, MARTE or create your own through profiles

4.1.2.3.1. References

Rhapsody, http://www-01.ibm.com/software/awdtools/rhapsody/

4.2. Requirement engineering

The information should be gathered in systematic manner using following strategy:

- Manufacturer webpage for initial information
- Scientific papers giving evaluations for the tools in question

Manufacturer's webpage will provide the initial information about what the tool is capable of doing and the scientific papers provide the information that can be used to compare against the information provided by manufacturer. In short, this way it is possible to collect the basic information and the possible evaluation information (problems, comparisons...)

GUIDE FOR WRITING TOOL DESCRIPTIONS:

Each tool evaluation should have following topics written:

- Tool description (Short description about the tool and its usage)
- Relevance to AMALTHEA (Why the tool is relevant to the project, for example will it set restrictions to data, is it required that the design flow interacts with the tool and so on.)

4.2.1. Topcased

TOPCASED is strongly model--oriented. The main features concerning requirement management are:

- Topcased allows a requirement import, supported formats are *.docx (Office 2007) and *.odt (Open Office), *.xlsx (Office 2007), *.ods (Open Office) and *.csv (an older Office format must be converted to a newer version, i.e. *.doc into *.docx)

 © The AMALTHEA consortium

- structured requirements can be used as it is, but it can also be adapt/extend (e.g. predefined structure elements using Word: Req_ID, Req_Abstract, Req_Text; self-defined elements using Excel)
- attach a requirements file to a di-model
- a user can define filter rules for the import process and he can search
- a uploaded requirements can when be attached to a model by simply drag & drop (visible via “link to”)
- a dedicated “requirement view” in Topcased GUI shows all the requirements in a list by reference, and in the details there are e.g. the “link to” to see
- in a properties view all requirements can be edited, managed including deletion and insertion of a new req. without upstream
- if a model element is deleted, the corresponding req. will be listed as “Not Affected”
- if a req. will be deleted, the req. will go to the end of the list (category “deleted”)
- if a req. was updated and if this req. has been changed, it will be explicitly shown as a warning and the user can set a valid flag to validate the changes (the warning will when disappear)
- new req. are not highlighted in the list to show the user that this req. is not used/linked
- it is also possible to split a model (e.g. in a collaborative work) and to duplicate the single req. file
- after the distributed parallel work the submodels can be merged and the corresponding req. files too
- This aims at exporting a matrix with traceability for: Upstreamed req. / Current req. / Model elements
  TOPCASED announces that there is an ongoing work on the requirements management functionality. Until now, the process uses a triplet of models: the DSL model, the graphical model (diagrams) and the requirement model which are not in line with exting methods, e.g. sysML. Therefore, one objective will be to keep the same principle without having a dedicated requirement model. SysML will store the requirements and links to the model elements.

4.2.1.1. References

FAUDOU, FAURE, GABEL, MERTZ: Topcased Requirement: a Model Driven, Open-Source and Generic Solution to Manage Requirement Traceability; ERTS; 19-21 May 2010; TOULOUSE (FRANCE)

4.2.2. Rational Doors

DOORS (Dynamic Object Oriented Requirements System) is originally created by QSS Ltd and is today known as Rational Doors as a part of the IBM’s Rational software product line. Rational DOORS is a multi-platform, enterprise-wide requirements management tool including requirement traceability in system engineering. Rational DOORS allows cross-functional teams to collaborate on development projects and enables the usage of views to navigate through different items in the database allowing user easily to obtain the information needed for the work.

© The AMALTHEA consortium

Rational DOORS' key feature for the requirements management is a central repository for requirements that allow users to store their requirements in one place allowing each team member to access and edit the same information. In addition, Rational DOORS also supports changing requirement's management, lifecycle management and also provides a system for change proposals. Second key feature is requirement's traceability where Rational Doors allows users to track the requirement's history data. This is enabled by either adding links manually for those items the traceability should be enabled or allowing Rational DOORS to do it manually using user defined rules and attributes.

Rational DOORS provides import and export capabilities for a range of tools and formats, including RTF, Word, WordPerfect, Excel, Lotus, Access, Plain Text, HTML, PowerPoint, MS Project, Outlook and many others. A direct export from external tool into Rational DOORS is also possible but requires a specifically made plug-in for the tool.

Rational DOORS also supports UML modeling directly in the program if the Rational DOORS Analyst Add On is installed. Additional Web Access add on provides a web client with similar functionality as the Rational DOORS allowing users to access the requirements database without the actual client installed into their computer.

In short, Rational DOORS:

- Provides a platform for requirements management
- GUI support for requirements management
- Allows web browser access for the requirements database (with additional add on)
- Supports the Requirements Interchange Format
- Ability to link requirements to other items like test cases, system designs and various other items
- Enables users to discuss and comment requirements
- Supports the Open Services for Lifecycle Collaboration (OSLC) specifications for requirements management, change management and quality management
- Import and export supports wide range of tools and formats

4.2.2.1. References

http://www-01.ibm.com/software/awdtools/doors/

4.2.3. Polarion

Polarion Requirements is a commercial collaborative web based requirements management solution with traceability and forensic level links and accountability. Polarion Requirements supports any type of process or methodology for organizations of any size in any industry. Polarion Requirements promises better requirements elicitation and collaboration at the lowest cost of ownership in the market, and to be as easy as MS Office.

Polarion Requirements support requirements document creation. Creation of professional online requirements documents. Imported documents still look and edit

 © The AMALTHEA consortium

like documents wit the usability of Microsoft Word without any of the limitations. The Polarion Import Wizard easily recognizes and imports documents that contain requirements, test cases, defects, etc. to a modern, web-based management platform. Supports PDF, MS Word, Excel, XML, HTML. Requirements and/or other entities are recognized both based on keywords (Must, Should, ”Ability to”, etc) or using custom document styles.

To support reuse Polarion Requirements preserves and tracks the complete lifecycle of a user story over time. Stakeholders can leverage knowledge and reuse for future projects. For example, manage a requirements specification & related test cases in one place and reuse some or all of them when implementing some variant of the base specification in another project.

Traceability and audit trails are supported. Change control and impact analysis are a vital part of the requirements management process. Polarion Requirements offer visual impact analysis allowing the determination of the potential impact that the changes may have on other requirements, release timelines, resource allocations, etc. Application also allows linking; one to many or many to one and also across projects.

Simplifies communication and collaboration (for example requirements negotiation or prioritization) between developers by offering facilities for web based collaboration via support for discussions, wikis, polling, subscriptions, alerts, and more.

- Real time collaboration across tools and teams. Subscriptions let to select events to monitor and be notified of any changes.
- Enter and browse document comments from within Word documents or from online collaborators.
- Vote or rank the importance of change requests, enhancements and defects.
- Read and sign documents electronically.
  All aspects of collaboration such as discussion threads are recorded so they can be trace and investigated to understand how various decisions were agreed upon.

Polarion Requirements is flexible and adaptable, and it supports any methodology or workflow:

- Agile/Lean, Traditional, Hybrid, or custom environments.
- FDD, XP, Scrum, RUP, Kanban, etc
- Attain compliance with a wide spectrum of industry standards like FDA, FAA, CMMI, ISO, IEC, CPI, SPICE, and many more.
- Customizable workflow support - covers all the artifacts from requirements.

Test case management support that allows specifying, manages, and executes test cases on a single platform. Easy to create test cases and easy to link them to their corresponding work items such as requirements, change requests, other test cases, etc. Polarion Requirements automatically create bug reports and tasks for developers based on test failures. Offers bi-directional synch with automated testing suites such as HP Quality Center.

© The AMALTHEA consortium

The facts about the Polarion requirements are:

- Document-like authoring. Author requirements, test cases and other work items in a document-like editor.
- Database-like authoring. Create and manage requirements in a tree-table interface by those who prefer to work with objects not documents.
- Web Client. A fully featured web client that allows people to collaborate online on the documents and work items, and access any kind of reports live and online.
- Microsoft Word Round-trip. Import and recognize the requirements from Microsoft Word Documents. Export as professional looking Word Documents. Seamlessly import back changes made outside of Polarion.
- Wiki collaboration and discussion. Supports discussions, comments, polling, wikis, subscriptions, alerts, and more.
- Editable traceability matrix. Manage and edit many to many relationships between different artifacts in the traceability matrix editor.
- Impact trees. Navigate through any level of links in the traceability and impact trees to understand easily the impact of any change.
- Requirements workflow. Define and reuse a company or industry standard document templates to speed up the specification process and ensure that all the documents follow the corporate identity standards.
- Fully auditable with Electronic Signatures. Control the requirements lifecycle via workflow engine, who and under what conditions can approve requirements, automate routine task by workflow transition triggers, fully customizable for any process you follow.
- Requirements templates. Attain compliance with a wide spectrum of industry standards lie FDA, CMMI, FAA, ISO, IEC, CPI, SPICE, and many more that require features like; electronic signatures, version and audit trails for all artifacts, traceability across any number of levels and artifacts.
- Custom link types. Ability to link and qualify the link role for all the different one to many or many to one relationships and do so across projects.
- Requirements Change Management. Workflow driven change management process so you know any time what is the status of related requirements or you can estimate the impact.
- Test Case Management. Specify, manage, and execute test cases on a single platform.
- Cross project querying & reporting. Search and report across any number or projects, any level of hierarchy easily.

Reuse:

- Polarion Requirements preserves and tracks the complete lifecycle of a user story over time.
- Stakeholders can leverage knowledge and reuse for future projects.
- For example, manage a requirements specification & related test cases in one place and reuse some or all of them when implementing some variant of the base specification in another project.

 © The AMALTHEA consortium

4.2.3.1. References

POLARION REQUIREMENTS,
http://www.polarion.com/products/requirements/index.php

4.2.4. Rational Focal Point

Rational Focal Point (FP) is configurable, web based product and portfolio management solution. Its main purpose is to facilitate product planning and management capabilities. FP covers tasks from strategic planning and goals to tactical tasks. Strategic tasks and management is facilitated with options to work on product portfolio level, while the tactical tasks are covered with specific product and requirements management functionalities.
Focal point's main goal is to enable the development of the right product for the right market at the right time. Its main users are portfolio managers, product managers and requirement managers. Two main selling points are requirement management and product planning functionalities.
This tool supports work with requirements by providing common repository, by allowing mangers to customize their information content, and to perform different analytical operation on top of them. Requirements from different sources are recorded in common repository. Most common way of recording requirements is via web browser access, but other ways (like custom applications access) are possible to.
Requirements for different products have different information content. For this reason it is important that tool offers ways to customize this content. In Focal Point, managers are able to define the basic concepts like requirements and features, and to define information content in a way it fits to all stakeholders needs.
Analysis of requirements spans form trivial functionality like visualization, coloring, sorting and filtering to complex functionalities like prioritization according to qualitative and quantitative metrics, ranking requirements according to value that they bring to different groups of users, assigning weight factors to categories, diagram representation of results, etc.
Observed from design flow point of view, Focal Point covers initial development phases – product definition and requirements management. In order to allow seamless integration with other tools in the chain, Focal Point offers a set of APIs implemented with web service technology for access to its functionality.

In short, Focal Point offers:

- Completely web based solution|Extreme configurability and flexibility|Workflow support|
- Role based access|Full history with audit trails|Integration possibilities|
- Excel import/export|Prioritizationsupport|Visualizations|
- Built-in calculations|Filteringcapabilities|Versioning and base-lining|
- Report generation (Excel, PDF, RTF)|Homepage / Dashboard|Word import|
- Views and filtering|TimeGrid|Investment Analyzer|
- Traceability views|Trendcharts|Gantt charts|
- Notifications|Mailimporter|and many more...|

© The AMALTHEA consortium

4.2.4.1. References

Focal Point, http://www-01.ibm.com/software/awdtools/focalpoint/

4.2.5. Accept 360

Accept 360 is an suit that includes strategic and portfolio planning tools, requirements management tools, road-mapping tools, tools for project execution (agile, traditional), and tool for idea harvesting and incubation. Accept360 Requirements is one of the tools from Accept 360 family. It creates a living repository for product intelligence. Manage requirements in the context of your company's own market drivers. Capture, collaborate and prioritize requirements with stakeholders and create road-maps.

Accept 360 is web based application. It can be installed in client's own IT infrastructure or it can be used as SaaS. In first case, the burden of running and maintaining the application is in scope of company's IT personnel, while in case of SaaS the provider of the service is responsible for running and maintenance. Accept's functionality is usually accessed via web browser, but other application like outlook express or eclipse development tool can be enabled for access to. Accept 360 is not only a requirement tool, it is an end to end solution for the initial development phases. Tool covers idea generation, strategy, requirements, and execution of requirements. In order to allow seamless integration with other tools in the chain, Accept 360 offers a set of APIs. The Accept Requirements API is designed to enable easy integration of information from third-party information systems into Accept Requirements and to allow lightweight reporting on Accept Requirements data. The API is standards based, uses web services technology, and supports any modern development environment. It was designed with the goal of making the integration as simple, easy and seamless as possible.
In short Accept 360 offers:

- End-to-End Solution - Ideation, Strategy, Requirements Management and Execution
- Single System of Record for faster, more confident decision making
- Real-Time Alignment between company and product strategy
- Visibility and Predictability at all levels
- Integrated Collaboration and Communication
- Best practices to maximize the value of your investment
- Enterprise scalable across teams, locations, product lines

4.2.5.1. References

Accept 360, http://www.accept360.com/solutions/accept360-requirements/
Accept Req.,
http://d1p939z8j66sl5.cloudfront.net/Accept_Requirements_DS022011.pdf

4.2.6. IBM Rational Rhapsody

Rhapsody supports the requirements engineering with standard UML or SysML Requirements. Requirements are stored in the model and relations (ids,

 © The AMALTHEA consortium

specifications) can be generated into the design elements and test cases to support the requirements traceability in the model and implementation code. Advanced requirements analysis and traceability capabilities can be enabled by the export of model information to requirements engineering tools via the Rhapsody Gateway. The Rational Rhapsody Tools and Utilities Add On includes advanced capabilities for bi-directional information exchange with third-party requirements management /authoring tools, including Microsoft Word, Excel and PowerPoint, PDF, ASCII, Adobe Acrobat 6.0+, Rational DOORS, Rational RequisitePro, Borland CalibreRM, Mathworks Simulink.

The main used UML elements are use case/sequence diagram and informal descriptions with text and charts.

4.2.6.1. References

IBM Rational Rhapsody, http://www-01.ibm.com/software/awdtools/rhapsody/

4.3. System architecture design

4.3.1. IBM Rational Rhapsody

Rhapsody uses particularly UML elements component and class diagram for the system architecture.

4.3.2. Atego - Artisan Studio

Artisan Studio is a system and software modelling tool suite that features:

- Complete support for OMG UML and SysML
- Extended modelling notations to provide domain specific modelling profiles including UPDM, DoDAF, MODAF, MARTE, IDL, ARINC 653
- Fully extensible meta-model and diagram notations with a graphical profile editor, which implements consistent stereotyping and actively grammar checker
- Provides collaboration between teams and team members through a multi-user repository
- End-to-end traceability across all models
- Artifact synchronization with The Mathworks MATLAB/Simulink and IBM Rational DOORS
- Configurable Microsoft Word and HTML document generation
- UML profiles, generators and reversers for ARINC 653, IDL3/ IDL3+, OMG XMI, MARTE, and SPT
- Code synchronization using the Automatic Code Synchronizer (ACS) and the Transformation Development Kit (TDK) for C, C++, C#, VB, Ada, Java, and IDL
- Generation of traceability and allocation tables into Microsoft Excel
- Atego, http://www.atego.com/products/artisan-studio/

4.3.3. Sparx Systems - Enterprise Architect

http://www.sparxsystems.com/products/ea/index.html

Enterprise Architect is a visual modeling platform that provides full life cycle modeling for:

- Business and IT systems
- Software and Systems Engineering
- Real-time and embedded development

It features built-in requirements management capabilities that makes it possible to trace high-level specifications to analysis, design, implementation, test and maintenance models using UML, SysML, BPMN and other open standards. Moreover Enterprise Architect supports generation and reverse engineering of source code for many popular languages, including:

- ActionScript
- Ada
- C and C++
- C#
- Java
- Delphi
- Verilog
- PHP
- VHDL
- Python
- System C
- VB.Net
- Visual Basic

A built-in source code editor allows navigating from model directly to source code in the same environment. Reverse engineering from source code, jar files or .Net binary assemblies helps to visualize applications and handle complexities. It furthermore features among others:

- Version Control Integration
- Model Driven Architecture (MDA) transformations for C#, DDL, EJB, Java, JUnit, NUnit, WSDL, XSD
- Built-in Debugging, Compiling of Executing Code and Testing Management
- Database Modeling for DB2, InterBase, Informix, Ingres MS Access, MySQL, MS SQL Server, Oracle, PostgreSQL, Sybase ASE, ASA, Firebird
- Systems Engineering and Simulation
- Automation API and Scripting with JScript, VBScript and Javascript
- Report Generation: PDF, HTML and Rich-Text

© The AMALTHEA consortium

4.4. Module design

4.4.1. IBM Rational Rhapsody

Rhapsody uses UML elements particularly class diagram, state machine, activity diagram. Rational Rhapsody provides also a domain specific language (DSL) to create own unique diagrams and diagram elements.

4.4.2. Mathworks - Matlab/Simulink

While MATLAB is a numerical computing environment, SIMULINK focuses on the modeling, simulating and analyzing of multi-domain dynamic systems. MATLAB specializes in

- Math, Statistics and Optimization
- Application Deployment
- Database Access and Reporting

Simulink however features

- Fixed-Point Modeling
- Event-Based Modeling
- Physical Modeling
- Rapid Prototyping and HIL Simulation
- Verification, Validation, and Test
- Simulation Graphics and Reporting

One of their key features is the code generation. Either Verilog or VHDL code for FPGAs and ASICs or C / C++ code with built-in support for AUTOSAR and ASAP2 for embedded systems can be generated from Simulink and Stateflow diagrams or MATLAB functions. Furthermore a basis to perform parallel computations on multicore computers, GPUs, and computer clusters is provided.

4.4.2.1 References

http://www.mathworks.de/products/simulink/index.html

4.4.3. YAKINDU - Damos

The YAKINDU Damos tools [Ung] support a data-flow oriented modeling of modules. For this purpose they include a block diagram editor that could be used e.g. for control systems engineering or digital signal processing. Such use cases usually model the flow of various physical quantities like temperatures, pressures, or voltages. The conversion of such quantities often lead to errors. To avoid these errors, Damos supports the usage of units of measurement within the block diagrams, thus, enabling an early validation already within the design phase.

© The AMALTHEA consortium

Beside the support for system structuring by subsystems and system fragments as described in section "Selected tools and frameworks", Damos also includes a simulator and a C code generator. In simulation, the continuous state of continuous blocks need to be computed by numerical integration. Damos offers different solvers for the integration method, like Euler, Runge-Kutta, or Dormand-Prince, each of them with different accuracy and computation time. The C code generator is designed to produce highly optimized C99 compliant code.

4.4.3.1 References

[Ung] Andreas Unger, Damos Blog - http://andreasunger.com/

4.5. Coding (Target Mapping)

4.5.1. IBM Rational Rhapsody

The Rational Rhapsody Developer supports full behavioral code generation for C, C++, Java, Ada including rapid prototyping, round trip and reverse engineering. Rhapsody generates code from object model diagrams and state charts (so-called "constructive" UML elements). The Code generation can be controlled by code generation properties.

Rhapsody in C comes with a number of specialized C language profiles, e.g. MicroC for applications/targets without multi-tasking operation systems and limited memory resources. The code generation is compliant with MISRA-C:1998.

4.5.2. Visual Studio 2010

http://www.microsoft.com/visualstudio/en-us

Microsoft Visual Studio is an Integrated Development Environment (IDE) that currently supports the programming languages

- Visual Basic .NET
- C
- C++
- C++/CLI
- C#
- F#

Besides utilizing the built-in Microsoft debuggers it also provides methods like Static Code Analysis, Profiling, Code Metrics for an additional diagnostic analysis. Modeling the architectural design but also reverse engineering can be done in one of the following UML compliant diagram types:

- Activity
- Use Case
- Sequence
- Class
- Component

 © The AMALTHEA consortium

Extensive support for software testing, i.e. Unit Testing, Code Coverage, Test Impact Analysis, Coded UI Test, or Load Testing is provided by the Test Manager included in Visual Studio. The functionality, like utilizing different compilers, version control interfaces or test frameworks, can also be increased through add-ins.

4.5.3. Mercurial (revision control)

Mercurial is a platform independent, distributed source control management for software development licensed under the open source GNU GPLv2. In contrast to traditional version control systems based on client-server architectures, Mercurial is distributed, which means that each developer holds a local copy of the entire development history. That way it works independent of network access or a central server. Mercurial in general provides the following functionality:

- Create
  Creating a repository
- Browse
  Browsing files in the repository and viewing the history of changes
- Revert
  Reverting the content of files to a previous state
- Merge
  Merging different development branches of a repository
- Update
  Synchronizing the content in a development branch
- Publish
  Publishing local changes in the development branch to the repository

In addition its functionality can be further increased with extensions. Although Mercurial is primarily run from command line graphical user-interfaces are also available for all platforms. Different IDEs like Visual Studio or Eclipse can even make Mercurial directly accessible in their user interface through plugins.

4.5.4. Xpand

Xpand [XPAND] is an open source code generation framework based on Eclipse. It provides a statically-typed template language featuring

- polymorphic template invocation,
- aspect oriented programming,
- functional extensions,
- a flexible type system abstraction,
- model transformation,
- model validation and much more.

It includes an editor which provides features like syntax coloring, error highlighting, navigation, refactoring, and code completion.

4.5.5. References

Mercurial, http://mercurial.selenic.com/

[XPAND] Eclipse, Xpand, http://www.eclipse.org/modeling/m2t/?project=xpand

4.6. Verification & validation

4.6.1. IBM Rational Rhapsody

The Rational Rhapsody Developer provides design level debugging of the model via visual debugging and animation of statecharts, activity and sequence diagrams, event injection and sequence diagram generation from the running executable. The sequence diagrams created can be compared with the expected behavior to validate proper design operation.

The Rational Rhapsody TestConductor Add On provides a code-based test generation and validation suite. It generates a test frame for the design to be tested. Within this test frame the test cases can be defined as code, flowchart, statechart or as a scenario. The TestConductor manages the test case execution, drives inputs, monitors design responses and provides the results of the test, including model coverage information for each test case.

The Rational Rhapsody Automatic Test Generation Add On is a UML model-based testing solution generating test cases to test individual components for specific purposes, such as state and transition coverage, MC/DC coverage, or isolation of a particular class from the overall design. Additionally it supports the identification of not covered model elements (dead code).

The Rational Rhapsody Tools and Utilities Add On supports the creation of graphical user interfaces for rapid prototyping with graphical panels and various widgets such as knobs, buttons, dials, meters and displays.

4.6.1.1. References

IBM Rational Rhapsody, http://www-01.ibm.com/software/awdtools/rhapsody/

4.7. Simulation

4.7.1. Inchron - chronSIM

chronSIM is a development tool for the simulation and analysis of real-time critical systems. Therefore, the user has to define the soft- and hardware architecture of an individual system or several networked ones at first. Applications can be modelled by using either the GUI or with C code, which can be imported from existing source code. Various stimuli are used to emulate the environment of systems in real operating conditions. With help of the following types of timing requirements it is possible to validate the real-time behaviour:

- Call Nesting: Limitation of nesting depth for function calls
- Recursion: Limitation of function recursion
- RTOS Failures
- Requirement Group: Container element for hierarchically structuring requirements

 © The AMALTHEA consortium

- Response Time: Limitation of the response time for processes
- Event Timing: Limitation of the time between two events
- Load: Limitation of workload for a CPU or bus resource
- Event Chain: Limitation of time between two steps of an event-chain
- Event Chain Break Off: Incomplete processing of an event-chain
- Event Chain Multiple Processing: Event-chains that show multiple processing of identical steps

The simulator then emulates the run-time behaviour on the projected processors or different CPU cores of the dynamic embedded systems according the set scheduling algorithm. Available therefore are FCFS, fixed priority, Round Robin, TDMA, OSEK and AUTOSAR Task. Finally the results are visualized in various diagrams:

- Sequence Diagram: Visualizes the information flow between tasks as well as between tasks and ISR-routines
- Stack Diagram: Visualizes the stack consumption of the tasks
- State Diagram: Visualizes in chronological sequence the activity of processes based on the states it can have and messages on the CAN- and FlexRay-busses
- Nesting Diagram: Visualizes the nesting depth of function calls for individual processes
- Event Chain Diagram: Visualizes the progress of event-chains
- Load Diagram: Visualizes the workload of resources
- Histogram: Bar chart that visualizes the statistical distribution of events

Based on that information and the validation of the predefined timing requirements, the user can analyse the static as well as the flow dependent execution path. For documentation a HTML reporting with an overview of the defined system and the results of the requirement evaluation can be created. Additionally a plain XML output for a personal analysis is available. A batch processing enables an automated analysis.

4.7.1.1. References

Inchron, http://www.inchron.com/chronsim.html

4.7.2. Inchron - chronVal

chronVAL is a real-time analysis tool for the analysis and validation of embedded systems. Therefore, the user has to define the soft- and hardware architecture at first. In this process individual or distributed systems are supported as well as multi-processor ones. Applications can be modelled by using either the GUI or with C code, which can be imported from existing source code. Various stimuli are used to emulate the environment of systems in real operating conditions. Models of the hard- and software system architecture are therefore abstracted on the level of individual tasks. Especially the dynamic run-time behaviour of software and the bus communication across microprocessors can be evaluated. It mathematically analyses the real-time behaviour of software taking account of a specific hardware under the influence of external events. chronVAL displays detailed system information about maximum CPU loads, burst lengths as well as task blocking and suspension times,

which can then be used to determine the best and worst cases for end-to-end timings, task suspension times, bus and CPU loads.

4.7.2.1. References

Inchron, http://www.inchron.com/chronval.html

4.7.3. Symtavision - Symta/S

SymTA/S is an Eclipse-based tool to design and verify the timing behaviour of embedded controllers, networks and distributed real-time systems. Therefore, the user has to define the soft- and hardware architecture of the target system in hierarchical levels at first. In this process individual or distributed systems are supported as well as multi-core ones. There is also the possibility to import existing configurations from an AUTOSAR, FIBEX or DBC file. Based on that information a mathematical model is built, which then is used to analyse the timing behaviour in a worst-case scheduling analysis as well as a distribution analysis and scenario analysis. The scheduling analysis provides the following results:

- Resource consumption time: Time required by an element to be executed on a resource excluding activation and preemptions
- Response time (best case, worst case, distribution): Time required by an element to be executed on a resource including activation and preemptions
- Load (total, overhead, for entire resource or each element individually): Long term load for an endless time range
- Execution backlog buffer: Required buffer size for each element that no activation gets lost
- Deadline violations: Report if an element response time exceeds its deadline

The system and the results are then visualized by SymTA/S in various ways:

- System graph: Architecture overview of the defined system
- Path Graph: Visualizes the information flow between tasks
- Task Graph: Visualizes tasks on an ECU as well as their variable and chaining connections
- Gantt Chart: Illustrates the schedule of a task or runable
- Line Chart: Visualizes the min./max. response time for tasks
- Pie chart: Visualizes the workload for tasks, cores, CPUs or resources
- Histogram Chart: Visualizes the distribution of response times
- Box & Whisker Chart: Visualizes the distribution of response times

With all the information determined during the scheduling analysis the user can design, verify and optimize a system considering specific parameters. A report generation to PDF or HTML containing all properties of the model as well as the results of the analysis helps to track and archive the engineering process.

4.7.3.1. References

Symtavision, http://www.symtavision.com/symtas.html

 © The AMALTHEA consortium

4.7.4. Synopsys - Platform Architect

Platform Architect is a graphical environment to capture, simulate and analyse the system-level performance of a SoC as well as multi-core systems. Therefore the user captures at first a SystemC based performance model of the platform in the graphical user interface. In addition stimuli have to be defined to represent the application workload. The simulator then executes the workload model on the platform model and gathers information on performance metrics like latency, throughput, and utilization. Once finished the results get listed in tabular form and can be used for a performance analysis. To support this, individual constraints for the gathered metrics can be specified manually, which then get color-coded in case of violations. To explore the impact of design parameters on the system performance metrics permutations of different design parameter scenarios can be defined in a csv-file. The results of this sensitivity analysis are stored in another csv-file, which can then be used to aggregate and explore the results in a spreadsheet application using various types of charts, like bar or pivot charts. Based on this analysis the design can iteratively be modify and analysed to get an optimal architecture, where all performance requirements are fulfilled.

4.7.4.1. References

Synopsys,
http://www.synopsys.com/systems/architecturedesign/pages/platformarchitect.aspx

4.7.5. Windriver - Simics

Simics is a full-system simulator based on the Eclipse platform that emulates the hardware of any digital system in a virtual platform in order to run software the way it does on the real physical hardware. The systems can thereby be composed of multiple heterogeneous and multi-core processors running multiple software stacks and operating systems and can in addition be connected via local bus, rack backplane, internal network or the internet. On those virtual platforms then real system software can be executed and their performance evaluated without any alterations to the original source code. Simics can therefore single-step any code forward as well as backward, break on hardware accesses and exceptions using the built-in Eclipse C/C++ debugger. Information on the memory management unit state, both virtual and physical memory contents, and the precise state on any processor are accessible at any time. In addition a full system process list, system execution time line, and code coverage report in HTML or plain text is available to analyse the software applications running on the virtual platform. To also be able to run what-if scenarios and investigate different system architectures Simics supports functional models of hardware devices written in C, C++, SystemC, and Python.

4.7.5.1. References

Windriver, http://www.windriver.com/products/simics/

4.7.6. criticalblue - prism

Prism is a software analysis environment based on the Eclipse framework to analyze, design, implement, verify and tune software for multicore platforms. Existing C or C++ projects of the Eclipse CDT (C/C++ Development Tooling) can be used without any need of modifications. At first a dynamic analysis is used to profile the sequential execution of the initial source code. The information of the resulting application traces are then visualized in different views:

- Graph: Graphical representation of the structure and runtime behaviour of the application
- Functions: Lists all functions and their caused workload
- Call Hierarchy: Shows function inter-relationships and frequency
- Dependencies: Lists data dependencies and their types
- Races: Lists data races and their types
- Code Performance: Lists regions of the code where most of the runtime is spent
- Schedule: Illustrates the schedule of a task in a Gantt chart
- Wait Hotspots: Overview of task activity
- Memory: Details about the regions of memory accessed, created and destroyed
- Cache Performance: Lists and visualizes the performance of tasks on the cache level

Based on that information the user can easily identify functions which consume the most execution time and understand the data dependencies between those. Before modifying any code, different parallel scenarios can be explored. The user therefore can emulate different numbers of cores, threads and dependencies in the system and specific tasks can be mapped to individual cores. The impact of those approaches on the target system is then updated in all the views mentioned above. That way the most suitable strategy for introducing parallelism can be selected and finally implemented accordingly. This however is not done automatically by Prism but has to be done by the user itself using POSIX Threads. Tracing the application after these changes however can now be used to confirm that the implementation safely achieves the desired results. Finally opportunities for further parallelization can be analyzed by reapplying the previous steps until the desired degree of parallelization is reached.

4.7.6.1. References

criticalblue, http://www.criticalblue.com/prism/

4.7.7. Fraunhofer First - Precision Pro

PrecisionPro is a planning tool to generate static schedules for safety-critical systems based on multi-core processors. At first the user has to define the timing requirements and the relational dependencies that a valid schedule has to fulfill. This is done in a simple ASCII file using a special problem description language and an

 © The AMALTHEA consortium

arbitrary text editor. After the system starts up such a problem specification can be loaded and displayed in the user interface. The lower part of the user interface displays in a table the information of the problem specification like priority, execution time, and periods among others. This data is enriched by additional information like workload and min/max execution time once the results of the scheduling are available. The schedules of the individual tasks and their assignment to the available CPU cores on the other hand are visualized with colored rectangles, which represent the time slices, in the upper segment. In that view moreover already scheduled time slices can be removed from the processor assignment as well as yet unplanned ones scheduled based on their minimal or maximal time value or at an individual starting time. The generated schedule can finally be exported to an ASCII file in a predefined format.

4.7.7.1. References

Precision Pro,
http://www.first.fraunhofer.de/en/department_systems_architecture/research_group_embedded_multicore/

4.7.8. Vector Fabrics - vfEmbedded

vfEmbedded is a web based environment to analyse, partition, and map a C program onto a specific multicore platform. It can perform both a static and dynamic analysis. The user therefore defines the hardware architecture of an individual system or several networked ones. Applications are imported by uploading existing C source code. The relative operating sequence of that program is then visualized in a list view as well as in a 3D rectangle representation. Complementing the latter one a graph illustrating the dependencies is available in addition. All representations indicate the nesting relationships of invocations and the hierarchy of tasks. Various detailed pieces of information on the individual components are provided to identify possible bottlenecks. The components can then be selected to be parallelized and as a result vfEmbedded determines the impact of those actions on the performance. Finally, the original source code provided with statements for multitasking according the developed design can be downloaded.

4.7.8.1. References

vfEmbedded, http://www.vectorfabrics.com/products/vfembedded

4.7.9. Cheddar

Cheddar is an open-source scheduling simulator for real-time systems developed and maintained by a team composed of the LISyC laboratory/Université de Bretagne Occidentale and Ellidiss Technologies and published under the GNU General Public License. It is designed to validate the temporal constraints of tasks on a real-time system. The system therefore can either be described with AADL or in the Cheddar architecture design language by using the graphical editor. Cheddar features the possibility to run feasibility tests on the one hand but it also has a scheduling engine on the other hand, which can generate a schedule and automatically look for violated timing constraints. The tool therefore supports a variety of scheduling algorithms as

© The AMALTHEA consortium

well as properties, like assigning tasks to different processors in a multi-processor system.
The generated schedule is modestly displayed in a Gantt chart with an overview of activity for each task. Additional textual information about the results is also provided. Cheddar can be run as a stand-alone application but a TOPCASED plugin is also provided.

4.7.9.1. References

Cheddar, http://beru.univ-brest.fr/~singhoff/cheddar/#]

4.7.10. OMNeT++

OMNeT++ is a discrete event simulation environment, primarily for building network simulators. It is free for academic and non-profit use; commercial users must obtain a license. OMNeT++ offers

- an component-based C++ framework with topology description language NED,
- an Eclipse-based IDE with graphical NED editor,
- a graphical runtime environment for simulation execution,
- a command-line user interface for simulation execution,
- event logging function and sequence chart generator
- result logging function and analysis tool
- documentation generator

Omnet runs on Linux, Mac OS X, other Unix-like systems and on Windows. Independent model frameworks include domain-specific functionality such as support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling, photonic networks, etc.

OMNeT++ is based on component architecture. The simulation models are specified in the Network Description (NED) language. It comprises mainly elements for the architecture design.

A model is called network and consists of communicating, hierarchically nested modules. There are two types of modules: compound modules build the hierarchical system structure, simple modules contain the component behaviour (written in C++). Modules can be connected via gates and channels and communicate with messages through these channels. The messages may contain data and attributes.

Components (modules) can be further structured in subordinated modules, but in the end the behavior is specified in the simple modules (leaves of module tree) as C++ code. Simple module types will be defined by subclassing the cSimpleModule class of the OMNeT++ class library and redefining it's methods for module generation, initialization and finishing as well as the receive packet handlers.

The OMNeT++ environment contains a compiler for the NED topology description language. The other parts of models (simple module behaviour) and framework (OMNet++ class library) are directly written in C++.

 © The AMALTHEA consortium

OMNeT++ contains both a graphical runtime environment and a command-line user interface for simulation execution. Furthermore powerful capabilities for event logging and sequence chart generation as well as for result recording are integrated. Result analysis and chart generation can be done with the IDE integrated analysis tool or alternatively with several third-party tools. For example the OMNeT++ R package supports loading the content of OMNeT++ result files into GNU R, a free software environment for statistical computing and graphics with powerful plotting capabilities.

4.7.10.1. References

OMNeT++, http://www.omnetpp.org/

4.8. Functional Safety and IT Security

4.8.1. Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles (MAENAD)

MAENAD is an FP7 project funded by the European Commission that started in September 2010 for a period of three years. It focuses on the model-based analysis and engineering of novel architectures for dependable electric vehicles. Due to the fact that the challenges faced in engineering Fully Electric Vehicles (FEVs) are already partly met by EAST-ADL2, an emerging automotive architecture description language that is compliant with AUTOSAR, MAENADs main goal is to refine the current EAST-ADL standard by advanced capabilities to facilitate development of dependable, efficient and affordable FEVs. The following thematic priorities are scheduled to be achieved by this project:

- Provision of support for the automotive safety standard ISO 26262, including a novel approach for automatic allocation of safety requirements to components of an evolving architecture
- Provision of an effective model-based prediction of quality attributes for FEVs such as the dependability and performance via use of advanced, scalable and automated techniques
- Provision of an automated exploration of potentially huge design spaces to achieve better or optimal trade-offs among dependability, performance & cost

Since the approach of the AMALTHEA project is also model-based the results of MAENAD might also benefit the development of the tool chain.

4.8.1.1. References

MAENAD, http://www.maenad.eu/index.html

4.8.2. Open Vulnerability Assessment System

OpenVAS - Open Vulnerability Assessment System - is a framework offering services and tools providing security vulnerability scanning and vulnerability management solution. OpenVAS is a free implementation of the well-known commercial framework Nessus, it is built and further developed upon the last free version of Nessus (2.2).

All OpenVAS products are Free Software. Most components are licensed under the GNU General Public License (GNU GPL).

Figure 4-4 shows the structure and main components of the framework.

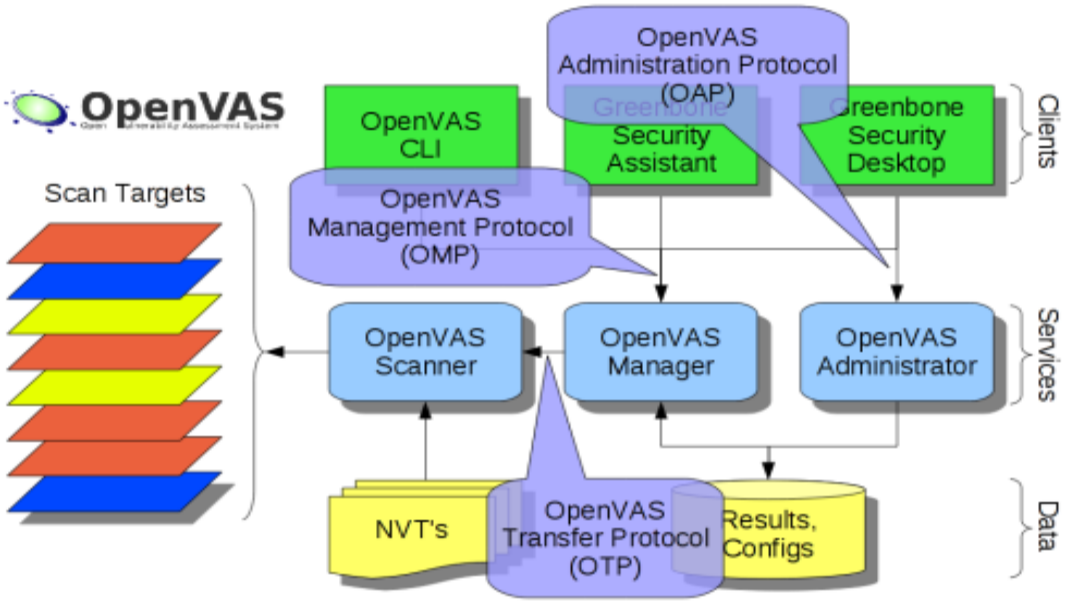

Figure 4-4: OpenVAS architecture

The core of this SSL-secured service-oriented architecture is the OpenVAS Scanner. The scanner executes the actual Network Vulnerability Tests (NVTs). NVTs are test-cases written in a script language and loaded as plug-in. So own NVTs can be written and loaded. But there are over 20,000 freely accessible NVTs and there is a daily update service.

The OpenVAS Manager is the central service that consolidates plain vulnerability scanning into a full vulnerability management solution. The Manager controls the Scanner via OTP (OpenVAS Transfer Protocol) and itself offers the XML-based, stateless OpenVAS Management Protocol (OMP). All intelligence is implemented in the Manager so that it is possible to implement various lean clients that will behave consistently e.g. with regard to filtering or sorting scan results. The Manager also controls a SQL database (sqlite-based) where all configuration and scan result data is centrally stored. The Manger also supports False Positive management.

A couple of different OMP clients are available: The Greenbone Security Assistant (GSA) is a lean web service offering a user interface for web browsers. GSA uses XSL transformation stylesheet that converts OMP responses into HTML.

The Greenbone Security Desktop (GSD) is a Qt-based desktop client for OMP. It runs on various Linux, Windows and other operating systems.
OpenVAS CLI contains the command line (cli) tool "omp" which allows creating batch processes to drive OpenVAS Manager.
The OpenVAS Administrator acts as a command line tool or as a full service daemon offering the OpenVAS Administration Protocol (OAP). The most important tasks are

 © The AMALTHEA consortium

the user management and feed management. GSA support OAP and users with the role "Admin" can access the OAP functionality.

Most of the above tools share functionality that is aggregated in the OpenVAS Libraries.

The OpenVAS Scanner offers the communication protocol OTP (OpenVAS Transfer Protocol) which allows to control the scan execution. This protocol is subject to be eventually replaced and thus it is not recommended to develop OTP clients. Traditionally, the desktop- and cli-tool OpenVAS Client acts as a direct OTP client.

4.8.2.1. References

OpenVAS, http://www.openvas.org

4.9. Domain Specific Languages and Editors

4.9.1. Xtext&Xtend

Xtext [XTEXT] is an open source framework for the development of domain specific languages (DSLs) or programming languages. It is part of the Eclipse modelling framework (EMF) project. In contrast to common parser generators Xtext does not only generate a parser but it also generates a model for the abstract syntax tree and tool support for the Eclipse environment. This includes a fully-fledged text editor that supports syntax coloring based on the lexical structure and the semantic data of DSL source code documents. An Xtext-generated editor proposes valid code completions at any place in the document, helping the users with the syntactical details of the DSL. Xtext supports for static analysis and validation of the models described by the DSL documents. Errors and warnings in the DSL documents are visualized. Custom quick fixes can be implemented to correct these with a single keystroke. The infrastructure generated for the DSL also comprises linking, thus enabling the model to be splitted into multiple DSL source code documents linking each other. An infrastructure for refactoring enables the consistent renaming of model elements that will be incorporated into all places where the model element is linked in the various DSL documents.

Software systems are not only made up of structure. At some point behavior and computations have to be defined. These aspects are implemented using expressions. Xtext ships with Xbase which is a predefined set of expressions. It can easily be embedded into any language developed with Xtext. It not only provides the necessary grammar, but also comes with a reusable compiler, an interpreter, and Eclipse tooling. Xbase supports static typing, type inference, closures and operator overloading. The language can be easily extended to add further expressions to a DSL.

The parsed model will be represented as an EMF model. Thus, it can be easily processed by interpreters, code generators, or model transformators. For code generation, Xtext also ships with Xtend. Xtend is a statically-typed template language, which eases the development and maintenance of code generators. It compiles directly to readable Java code, and its syntax is similar to Java but is much

© The AMALTHEA consortium

less verbose. The language itself is developed with Xtext and is based on Xbase. Code generators make heavy use of string concatenation. Therefore they are usually implemented using a template language. So far people tend to reuse a template language used in web development, which lack static typing and easy reuse of template snippets. Xtend's template expressions allow the developer to write readable and maintainable code generators.

4.9.2. Timing Augmented Description Language

The Timing Augmented Description Language (TADL) was developed in the ITEA2 project TIMMO, which ran from April 2007 to September 2009. The projects main goal was to develop a formal, standardized approach to describing timing-related information in embedded-system design for the automotive industry. The resulting TADL provides a method for formal specification, analysis and verification of timing constraints at different levels of abstractions as well as throughout the entire development process. It is based on the modelling concepts of the common infrastructures of EAST-ADL and AUTOSAR. Those concepts are then augmented by adding information related to timing and events referring to structural elements. EAST-ADL is used to model the higher abstraction levels (vehicle, analysis, and design) while AUTOSAR specifies the implementation level. The timing constraints however are defined separately from the structural modelling in TADL. They are then associated with structural elements via Event Chains and Events according the meta-model shown in Figure 4-5.

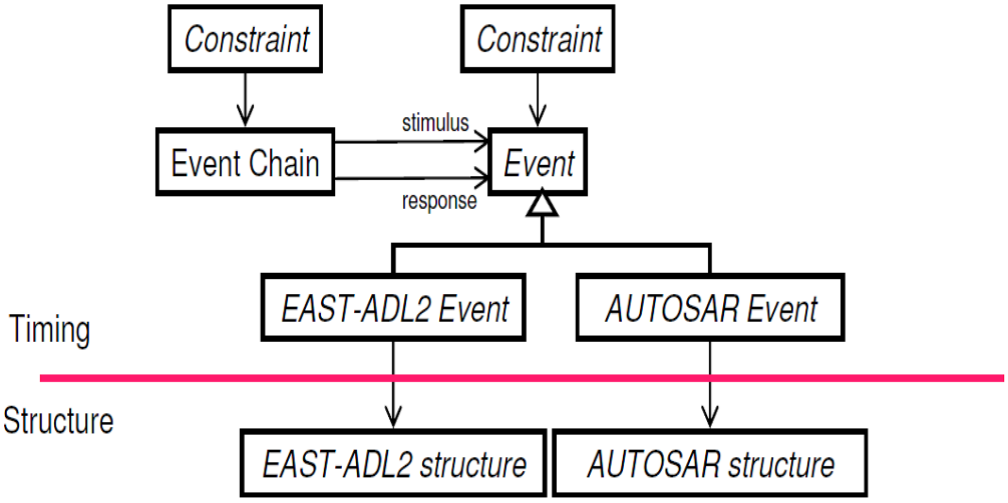

Figure 4-5: Simplified extract of the TADL UML meta-model (TIMMO Open Workshop 26.03.2009)

As specified by MARTE, the UML profile for Modelling and Analysis of Real-Time and Embedded Systems, constraints defined by TADL also have a constraint kind which specifies the distinction between whether a constraint is offered, required or a contract. Additionally all TADL constraints have a timing bound and a mode. In general the following three different end-to-end constrains can be differentiated in TADL:

- Reaction Constraint
  defines the maximum difference between a stimulus and its response

 © The AMALTHEA consortium

- Output Synchronization Constraint
  defines the maximum difference between a set of responses
- Input Synchronization Constraint
  defines the maximum difference between a set of stimuli

TADL also benefits of the requirement support provided by EAST-ADL, which allows for tracing from solutions modelled in the structural model to requirements and from verification cases to requirements. The TADL constraints are thereby defined as refinements of existing requirements. Further support in EAST-ADL includes the tracing between abstraction levels by realization for the structural elements and derivation between requirements.

For all the functionality mentioned above the following relationships are available in EAST-ADL:

- ADLSatisfy
  Indicates, that the modelled function satisfies a specific requirement
- ADLDeriveReqt
  Indicates, that a requirement is derived from another
- ADLRefine
  Used to refine a requirement by timing constraints
- ADLVerify
  Indicates, that a verification case verifies a specific requirement
- ADLRealization
  Traces requirements between EAST-ADL abstraction levels

However, not only a way to model timing information was developed by the project but also a methodology of how to apply the TADL.

The methodology follows once again EAST-ADL and AUTOSAR. That is why the Software Process Engineering Meta-model (SPEM), a standard for modeling arbitrary software and system development processes, was chosen for the underlying meta-model. Furthermore, the process is inspired by the V-Model development approach. Based on the abstraction levels available in EAST-ADL (vehicle, analysis, design and implementation level) and the different views of AUTOSAR (VFB, System and ECU) the methodology describes the tasks that are necessary to process the timing information. Therefore a textual description in form of a table in the main description is added to exactly specify which input work products are used as source for an individual output work product. The basic idea behind the methodology is to introduce time budgeting using the capability to specify Events, Event Chains and Event Chain Segments respectively. TADL enables then the subsequent refinement of time budgets across different level of abstractions and therefore in different phases of the development process. This timing information can then be iteratively exchanged between car manufacturer and suppliers to systematically control and validate the design stages. Finally the methodology also provides recommendations on when a set of tools, like simulators, can be applied to validate the model, what data are required to make use of the mentioned validation method and what the possible results are.

© The AMALTHEA consortium

A follow-on ITEA2 project is already available. TIMMO-2-USE started in October 2010 with two years of duration. Its goal is the development of novel tools, algorithms, languages and a methodology validated by use cases. The AMALTHEA project might benefit of their results in the area of partitioning and modeling of multi-core systems and of the integration of TADL into the tool chain as one of the underlying DSLs.

4.9.2.1. References

Timing Augmented Description Language, http://timmo-2-use.org/timmo/index.htm

4.9.3. Secure Use Cases

The so called secure use cases are to be used to examine possible attacks in different use case scenarios. Use cases based on simple text and figures are well accepted in the requirement phase. Based on the depicted attack scenarios a possible mitigation by dedicated countermeasures can be defined. This approach can also be seen as the definition of IT security related requirements. The right responses to the attacks and the attacks themselves can be viewed as patterns. They can be taken from catalogues such as the hazard and actions catalogues of the German Federal Office for Information Security (BSI). Hence, a certain use case on which an attack is possible, i.e. confidential or sensitive data was readed by an unauthorized threat agent, must be protected. Then a general IT security objective would be to warrant authorization. In result only authorized agents should have access 1 in the given use case. This scenario can be fully depicted using security use cases. Therein different IT security relevant interactions will be shown. It is also possible to define the roles and they rights for the acteuers, i.e. persons or objects.

Consideration of the Secure-use cases is limited to authorization problems. Security protection goals such as integrity or availability cannot be displayed. The restriction to the use-case view the created models are easy to understand, even people who are not otherwise make use of the UML in a position to capture the contents of the models quickly.

4.9.4. MisUse-Cases

The MisUse-Cases are originally developed by Sindre and Opdahl 2 and it has been refined by Ian Alexander 3. However, they use a similar principle as the Secure Use Cases but the concept was extended, i.e. elements. In contrast to the Secure Use Cases, the MisUse Cases is based on the objectives of different actors and not from systems operations. The essential element is the "MisUser" that is a person or object that intentionally or unintentionally cause a certain risk to a given use case scenario. Between the Use Cases and MisUse Cases there are various relations exists. These relations are "threaten", "reinforced", "mitigated" and "conflict". In addition, there are also the standard UML relations "extend" and "include". The relations are defined as follows 4:

- „MisUse-Case A “threatened a „Use Case B“ if achieving the goal of A, and the system cannot reach its own goal B.

 © The AMALTHEA consortium

- „Use Case A“ is mitigating a „MisUse Case B“ if the effects like threats will be reduced.
- A given „Use Case“ or „MisUse Case A“ amplifies an also given „MisUse-Case B“, if the likelihood of success or the damage of B increases.
- Two „Use Cases“ A and B are in conflict, if during the way to A the reaching of the goal of B is more difficult and vice versa.
- The goals of actors are the main part of the considerations, so they must be identified at first. The objectives of the actors, which may interact with the object to be developed, are shown in white ellipses (authorized agents). The goals of the attacker (a MisUse) are shown in color-inverted ellipses. In the literature there are not many details about the specific procedure for the implementation of MisUse-Case diagrams. However, it is clear that an approach in which all the actors and objectives of a system are shown in just one diagram is not optimal. The final model looks then overloaded and a requirement analysis is the more difficult.
  For this reason, the following procedure has been established. First of all, the MisUse Case model should be created with just two actors. One actor represent the major user of the system and the other one the general attacker. Afterwards, both actors are defined by they goals and between these goals relationships will be assigned (using mitigation, conflicts etc.). This procedure is like a first iteration. The next iterations will do the same thing but going into more details, e.g. more or all possible threats should be mentioned in this model.
  With such a simple syntax, the MisUse Case is based on the well-known Use Case approach. It extents Use Case with a possible attacker view. Finally it is easy to follow such method and the modelling approach is also simple because of they easy syntax.
  Use Cases are widely accepted and also more and more used for specifications. Particularly in safety related automotive and industrial applications, MisUse Cases could enlarge the threat risk analysis based approach.

4.9.5. Security Problem Frames

The security problem frames are an extension developed by Michael Jackson, Problem frames and method used to describe problems during software development. Jackson himself wrote about problem frames: A problem frame is a kind of pattern. It defines an identifiable problem in terms of its context, the characteristics of its domain and its requirements and interfaces 5.
The goal of general problem frame approach is to develop a so called "machine" which can influence or improve the behaviour of the environment in which the end-user application is working. This machine can be considered as a software system. Problem Frames are defined using context diagrams those consist of rectangles and connections between them. The rectangles represent domains (problem areas) and are divided into "causal" (physical rules), "lexical" (representing data) and "biddable" (people).
The context diagram is used to identify the problems that exist in a reaction of the machine and to recognize their relationship. Requirements are not defined in these diagrams. After creating the context diagram, several sub-problems will be derived. This is formalized and extended by requirements to develop problem-diagrams.

Requirements are shown with a dashed ellipse. Also between the requirements and the domains represented by lines interfaces exists. If the line is dashed, it represents a dependence of the request if it also has an arrow, it is a limiting function.

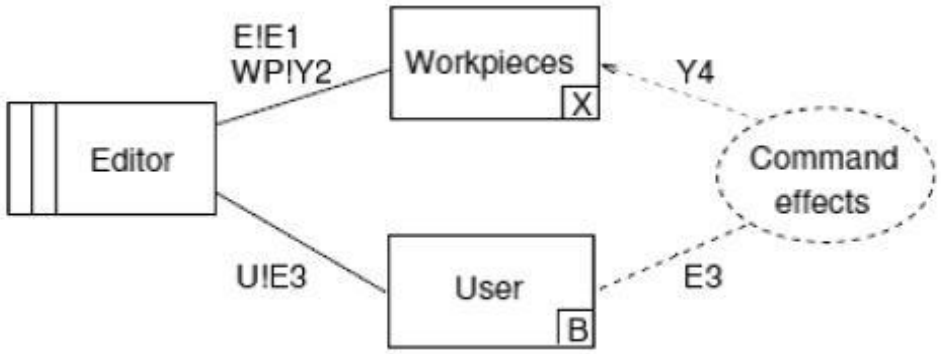

Figure 4-6: Problem Frame 6 [MaTRoRo]

Figure 4-6 shows an "X", that means that the domain is lexical Workpieces. The "B" indicates that the user is "biddable". A domain, which is "causal" would be marked by a "C". The interfaces between the domains contain comments, these have the following meaning: The comment "E! E1" indicates that the property is the domain of E1 E (the editor) controls. The comment "U! E3" indicates that the property E3 (user commands) through the user-controlled domain. The subdivision of context diagrams takes place so that the resulting problem diagrams to fit existing problem frames. For this, the problem frames "instantiated", so described by the exact requirements, domains and interfaces. If the problem diagrams structured exactly like the problem frames, they have similar characteristics. The solution to a problem that has frames and similar characteristics. Jackson has defined five basic problems. The problem frame is described by him to the "Work Pieces" problem shown in figure 2-6. As a complement to the frame problem developed Jackson Abuse-frames, in this place needs to be anti-mapped requirements, these are the demands of an aggressor. Security problem frames are specially concerned the security problem frames, they were developed at the University of Duisburg under the supervision of Prof. Dr. Maritta Heisel. They treat problems such as problem-frames only, they do not prescribe solutions. However, they can be more accurately described by "Concretized security problem frames". These included approaches and can sometimes even be associated with the design phase. Security problem frames can thus be viewed as a kind of transition between the phases.

In various publications (e.g. 6) there are patterns descripted which can be helpful in the AMALTHEA project. Most of the patterns coming from the software engineering group at the University of Duisburg. The main security objectives are integrity, confidentiality and authenticity. The approaches can be taken in the design phase. Unfortunately, the notation and the modelling approach of this method are unfamiliar, but well described.

 © The AMALTHEA consortium

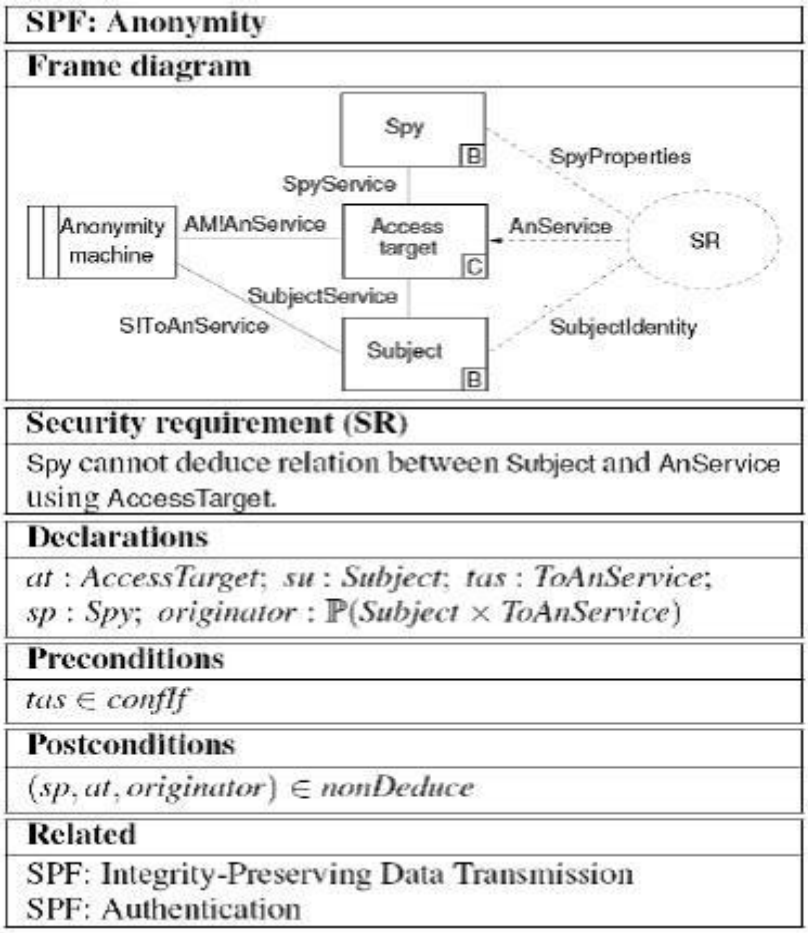

Figure 4-7: Security Problem Frame template 7 [MaTRoRo]

4.9.5.1. References

1 E. Fernandez-Medina and M. Piattini, “Designing secure databases, ”Information and Software Technology, vol. 47, no. 7, pp. 463–477, 2005.
2 G. Sindre and A. L. Opdahl, “Capturing security requirements through misuse cases,” 2001.
3 I. F. Alexander, “Modelling the interplay of conflicting goals with use and misuse cases,” in GBPM, 2002.
4 I. Alexander, “2002-initial industrial experience of misuse cases in trade-off analysis,” in Proceedings of IEEE Joint International Requirements Engineering Conference, pp. 61–68, 2002.
5 M. Jackson, Problem Frames: Analysing and Structuring Software Development Problems.Addison-Wesley, 2000.
6 D. Hatebur, M. Heisel, and H. Schmidt, “Using problem frames for security engineering,” tech. rep., Faculty Of Engineering, University of Duisburg-Essen, 2006.
7 D. Hatebur, M. Heisel, and H. Schmidt, “A security engineering process based on patterns,” in Proceedings of the International Workshop on Secure Systems Methodologies using Patterns (SPatterns), DEXA 2007, pp. 734–738, IEEE Computer Society, 2007.
[XTEXT] Xtext - http://www.eclipse.org/Xtext/.
[MaTRoRo] Master Thesis Roland Rothe, Otto-von-Guericke University Magdeburg

5. Conclusion

The main contribution of this deliverable is the selection and analysis of relevant standards, academic approaches and available tools in the context of the intended AMALTHEA goals.
All of the listed standards, methodologies and concept as well as the list of available well established tools and tool chains can be understand as the initial position of the AMALTHEA approach. Based on this deliverable, the next steps in AMATHEA project particularly work package 1 is to identify requirements to the overall AMALTHEA approach. Anyway, a good basis to identify requirements is in one hand the herein listed standards and in the other hand the analyses of end user stories or respectively use cases.

© The AMALTHEA consortium

6. Glossary

Expression	Explanation
AADL	Architecture Analysis & Design Language
BCRT / WCRT	Best-case / Worst-case Response Time
BCET / WCET	Best-case / Worst-case Response Time
BSW	Basic Software, abstraction level in AUTOSAR
CMMI	Capability Maturity Model Integration
DSL	Domain-specific Language
E/E/PE	Electrical, Electronic or Programmable Electronic
ECU	Engine Control Unit
IDE	Integrated Development Environment
ISR	Interrupt Service Routine
MAENAD	Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles, itea2 project
MDD / MDA	Model-driven Development / Architecture
RTOS	Real-time Operating System
SoC	System on a Chip
SPICE	Software Process Improvement and Capability Determination
SysML	Systems Modeling Language
TADL	Timing Augmented Description Language
TDMA	Time Division Multiple Access, scheduling algorithm
UML	Unified Modeling Language
VFB	Virtual Functional Bus, abstraction level in AUTOSAR

© The AMALTHEA consortium