(ITEA 2 – 13017)

Enabling of Results from AMALTHEA and others
for Transfer into Application and
building Community around

Deliverable: D 1.3
Design Handbook

Work Package: 1
Continuous Design Flow and Methodology

Task: 1.8
Preparation of the design handbook

Document Type:	Deliverable	**Classification:**	Public
Document Version:	Preliminary	**Contract Start Date:**	01.09.2014
Document Preparation Date:	August 31, 2017	**Duration:**	31.08.2017

Contents

List of Figures

List of Tables

The design handbook will describe all necessary steps for developing automotive multi- and many-core systems with the AMALTHEA4public tool chain. The document describes how the different design steps are supported by APP4MC as well as other third party tools developed by project partners in the course of the project. Furthermore, the handbook gives an example of how APP4MC and other third party tools can be integrated to develop multi-core systems.

1 Introduction

Performance is a critical quality attribute for systems developed in the automotive domain. The introduction of multi- and many-core processors has lead to a significant increase in performance as several tasks in the system can now run in parallel. However, designing and implementing efficient multi-core systems requires specialized tools and processes for development. In this report, we describe necessary steps that need to be carried out when developing multi- and many-core systems and also describe how these steps are supported by the AMALTHEA4public tool chain.

The AMALTHEA4public tool chain is a platform for engineering embedded multi- and many-core software systems. The platform enables the creation and management of complex tool chains including simulation and validation. This tool is now offered as a free and open source Eclipse project called APP4MC (Application Platform Project for MultiCore). From this point forward, we will refer to the AMALTHEA4public tool chain as APP4MC. The aim of this deliverable is to show how the tool can be used to support various design steps and also show how other third party tools can also be integrated with the APP4MC platform to facilitate multi-and may-core development.

1.1 Structure of the Document

The rest of the document is organized as follows: Chapter 2 gives an overview of the design steps that exist when developing embedded systems which were collected and reported in detail in Deliverable 1.1 [1]. Chapter 3, describes how the design steps are supported by the APP4MC platform. Chapter 4 discusses support for design steps provided by third party tools that have been developed as part of the AMALTHEA4Public project. Chapter 5 describes the traceability concept, its importance in systems development, and tools that support traceability developed in the project. Chapter 6 gives an example of how APP4MC can be used in combination with third party tools to develop multi- and many-core systems. Finally the document ends with Chapter 7, which gives a conclusion.

2 Overview of design steps

In this Chapter we discuss the neccessary design steps for developing multi- and many-core systems. Table 2.1, gives and overview of these steps. These steps were collected by sending out an open ended survey to project partners. The project partners were asked to document the steps they use in their current development processes. The results of this has been published in [1] and in [6]. Note that no order in which these steps are carried out is implied since defining a concrete development process with a concrete lifecycle is generally company specific. A wide variety of lifecycles can be applied, including the V-model(cf. 2.1) that is implied by ISO 26262 and AUTOSAR. While some steps are carried out sequentially, others can be done in parallel. The dependencies between the design steps at times imply an iterative approach where they are repeated or at least revisited after other work has been performed. Such an approach is common in iterative-incremental lifecycles. The identified steps cover most aspects of a traditional software development effort, starting from contract negotiation and scope identification and ending at the delivery of the software (with the exception of software maintenance). Some steps and circumstances are specific in the context of the automotive domain, e.g., the differentiation of system and software. Another re-occurring theme is variability management via software product lines, even though this theme will not be regarded in detail in the context of this paper. Some of the design steps we elicited, such as Requirement Engineering and Architecture Design, can be found in nearly all development processes in a similar form. However, there are specific steps steps that are only relevant in multicore development: Partitioning, Task Creation and Target Mapping are, e.g., part of **DS 10: System Integration**. We discuss these design steps in more detail in Chapter 3.

Since the automotive domain is a safety-critical domain and therefore needs to adhere to safety standards, we analyzed the ISO 26262 standard, which is a functional safety standard for road vehicles, for steps that are necessary to ensure that the system developed is safe. From this standard, we derived six more safety related activities that are marked as **DA to DS F** both in Figure 2.1 and Table 2.1.

DS 1: System Requirements Engineering	DS 7: Variant Configuration
DS 2: System Architecture Design	DS 8: Implementation
DS 3: Software Requirements Engineering	DS 9: Validation and Testing
DS 4: Derivation of Product Variants	DS 10: System Integration
DS 5: Definition of Software Architecture	DS 11: Handover
DS 6: Behaviour Modelling	
DS A: Functional safety concept	DS D: Safety Validation
DS B: System safety requirements engineering	DS E: Functional Safety Assessment
DS C: Software safety requirements engineering	DS F: Integration and validation at vehicle level

Table 2.1: Overview of the identified Design Steps

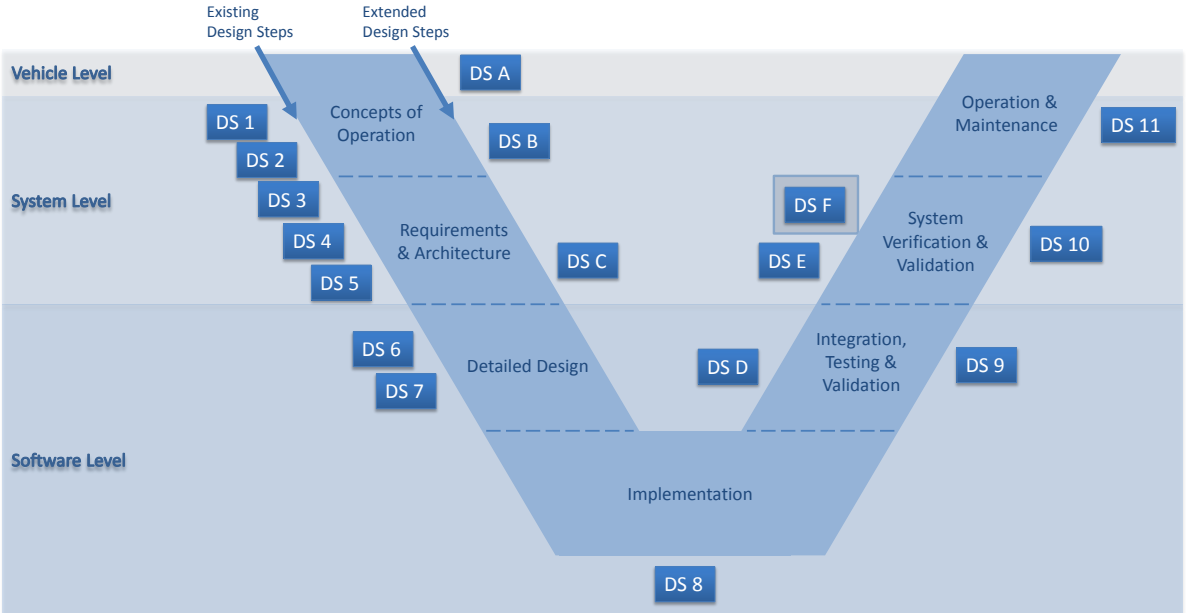

Figure 2.1: Overview of functional and safety-related Design Steps

3 Support for design steps in APP4MC

As it can be seen from the Table 2.1, there are several common steps that apply to all system development e.g., requirements engineering and implementation. The APP4MC platform however, was developed to support multi-and many-core development activities. For this reason most functionality is dedicated towards activities relevant for multi-and many-core system development. However, the APP4MC platform also provides supports for other activities that are not necessarily muti-and many-core relevant. In this section we first describe how the APP4MC platform supports muti-and many-core activities and also describe support for other activities. Table 3.1 shows an overview of design steps that are supported by the APP4MC platform.

Design Step	Support by APP4MC
DS 1: System Requirements Engineering	
DS 2: System Architecture Design	
DS 3: Software Requirements Engineering	
DS 4: Derivation of Product Variants	
DS 5: Definition of Software Architecture	✓
DS 6: Behaviour Modelling	✓
DS 7: Variant Configuration	
DS 8: Implementation	✓
DS 9: Validation and Testing	✓
DS 10: System Integration	
DS 10.1: Create Executables	
DS 10.2: Partitioning	✓
DS 10.3: Task Creation	✓
DS 10.4: Target Mapping	✓
DS 11: Handover	
DS A: Functional safety concept	✓
DS B: System safety requirements engineering	✓
DS C: Software safety requirements engineering	✓
DS D: Safety Validation	
DS E: Functional Safety Assessment	
DS F: Integration and validation at vehicle level	

Table 3.1: Overview of the identified Design Steps and APP4MC support

3.1 Support for Multi-core Activities

The more interesting step in this section is the step that is specific for many-and multi-core development which is **DS 10: System integration**. This step consists of four sub-steps of which three are most relevant, these are **DS 10.2: Partitioning, DS 10.2: Task creation and DS 10.4: Target Mapping**.

The support provided by the APP4MC platform for these steps are described as follows:

3.1.1 Partitioning

Partitioning in multi-and many-core systems development means identifying all tasks in the system, their structure and deriving possible ways in which these tasks can be divided in order for them to be executed in parallel. Especially for large systems, this is a complex task and needs specialized tools that implement efficient partitioning algorithms. The APP4MC platform, supports this task by allowing for the identification of software that can potentially run in parallel under consideration of activations, ASIL(Automotive Safety Integrity Level) safety levels, tags (e.g. for Software Components), Runnable Pairing Constraints, Runnable Core Pairing Constraints, and timing constraints. This is possible because all these constraints can be defined in the AMALTHEA model.

The partitioning component in APPMC utilizes the constraints defined in the AMALTHEA model and in return, produces a partitioned model. Figure 3.1 shows a screen shot of the partitioning plugin in action. It shows options that a programmer can enable or disable such as grouping runnables by their activations or by their ASIL levels. Figure 3.2 on the other hand shows how the partitioning tool can increase the performance of the system due to parallelism. The screen shot shows that partitioning in this case reduced the execution time by 85%.

3.1.2 Task Creation

In this step, tasks are created from a software model. This step is important as it produces a software model with tasks that can then be analyzed in order to identify tasks or group of tasks that can be run in parallel. Tasks agglomerate multiple runnables into a larger group, which is allocated on the target platform. In the APP4MC platform, this activity is supported by the task generator component. This takes the software model as input and produces a software model with tasks and sequencing constraints. Figure 3.3 shows the output of this process in the APP4MC platform.

3.1.3 Target Mapping

This step involves finding a valid and optimal distribution of software elements to hardware components. In APP4MC this is supported by the mapping component. The mapping component utilizes the software model and hardware model, as well as the tasks activation from the stimulation model, and calculates such a distribution. This tool determines the optimal allocation of software elements to hardware components, e.g. tasks to cores, data to memories, etc. The tool is currently capable of minimizing the run-time of a system by load-balancing the cores of a hardware or its energy consumption by selecting appropriate voltage levels.

Figure 3.4 shows the Mapping component in action. As it can be observed from the screen shot, the mapping tool allows the developer to choose between different algorithms of mapping, specifically three algorithms are provided which are DFG (Data Flow Graph), ILP (Integer

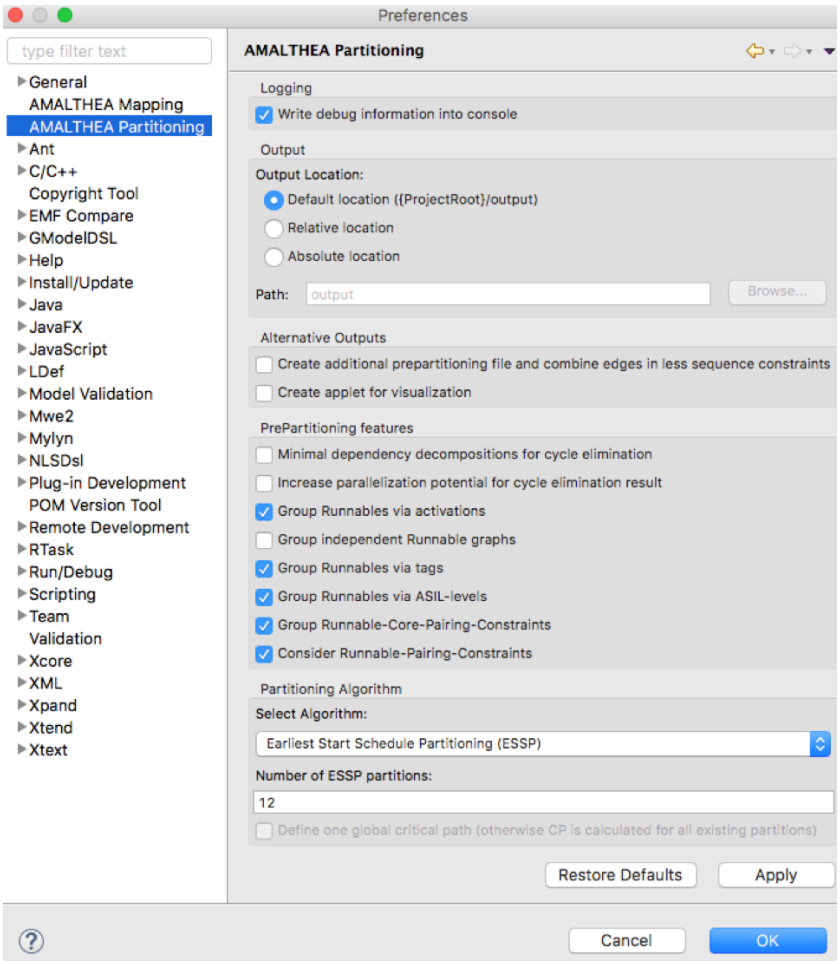

Figure 3.1: Partitioning in APP4MC

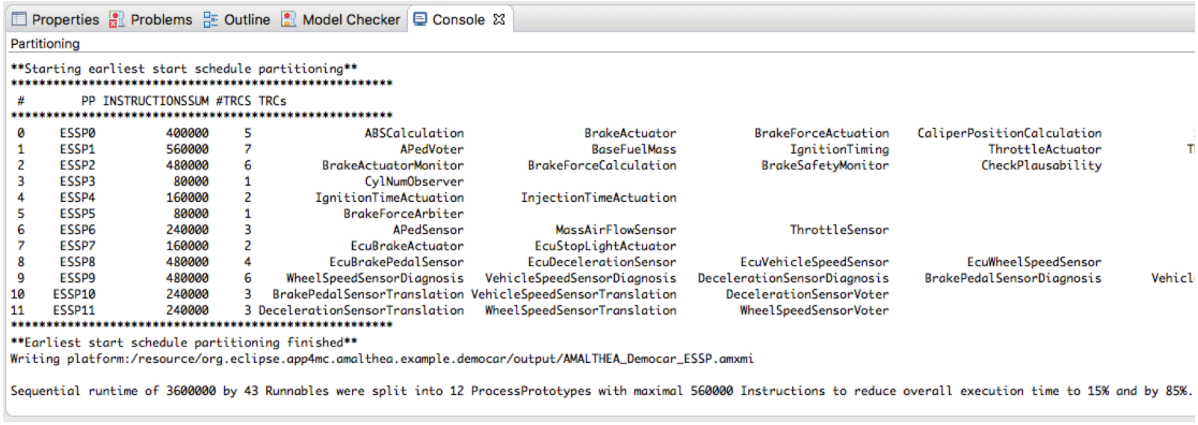

Figure 3.2: Results of Partitioning decrease the overall execution time

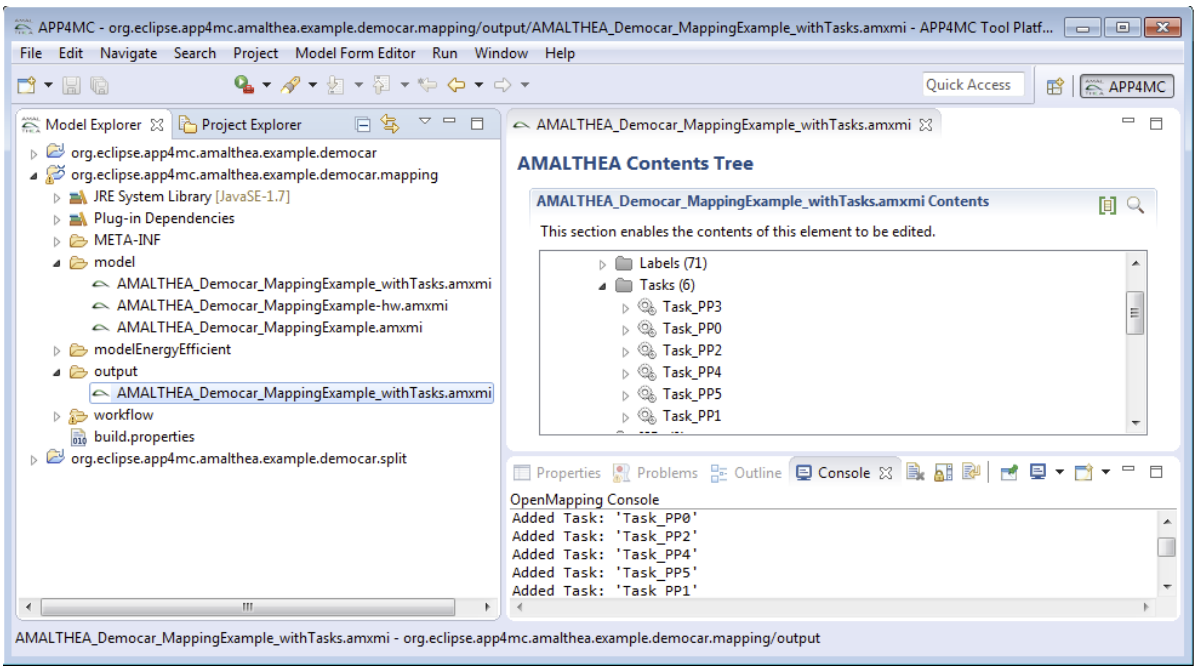

Figure 3.3: Tasks generated from the software model

Linear Programming) and GA(Genetic Algorithm) based balancing. The tool also provides an option to map the software to the hardware based on energy consumption preferences. Note that this feature is still an experimental feature in APP4MC. An example of how the partitioning model produced in this steps is given in Figure 3.5. The result of the mapping activity are stored in a mapping model.

The APP4MC platform also provides another component called Task visualizer which simulates the software based on the outcome of the mapping component, i.e. the execution of tasks of the software model on the cores of the hardware model, possible deadline misses etc. The task visualizer tool provides different visualizations and allows the programmer to set what should be displayed. Figure 3.6 to 3.8 shows three different visualizations supported by the APP4MC task visualizer.

3.2 Support for Other Activities

3.2.1 Definition of Software Architecture

In this step, the software is designed to implement the requirements. Here the components and function groups of the software and their relationships based on the requirements are modelled. To determine the component architecture, software architects use the requirements as well as the corresponding variant model.

The APP4MC platform provides the software model and component model to support this activity. The software model provides an abstract description of the software, e.g., which runnables exist, how many instructions are required for the execution of each runnable, which date is communicated etc. The software model also allows for the definition of constraints. The component model contains the software components of the system, their communication and

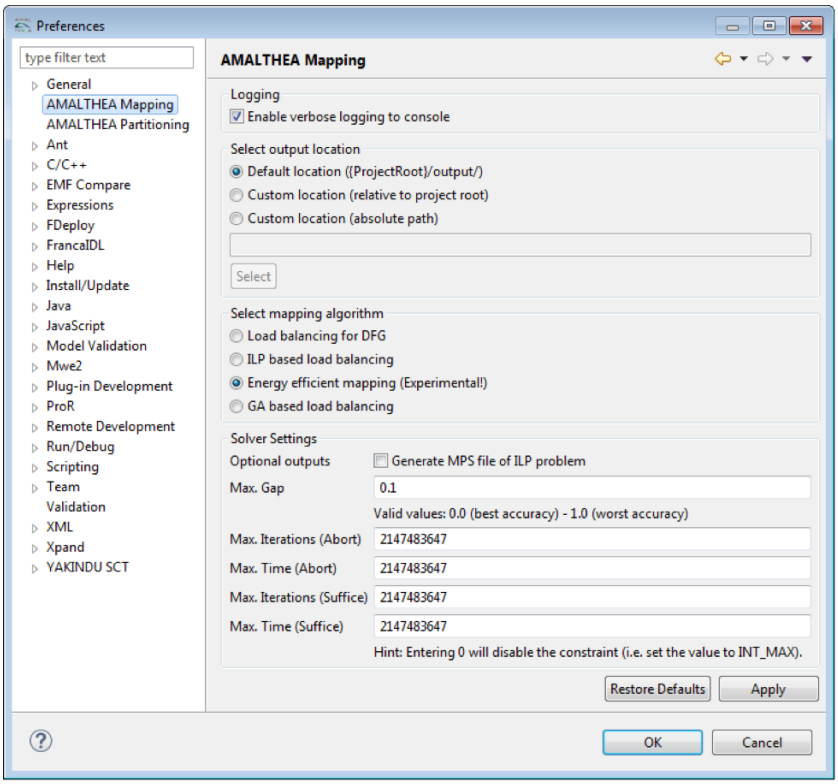

Figure 3.4: Configurable aspects of the APP4MC mapping tool

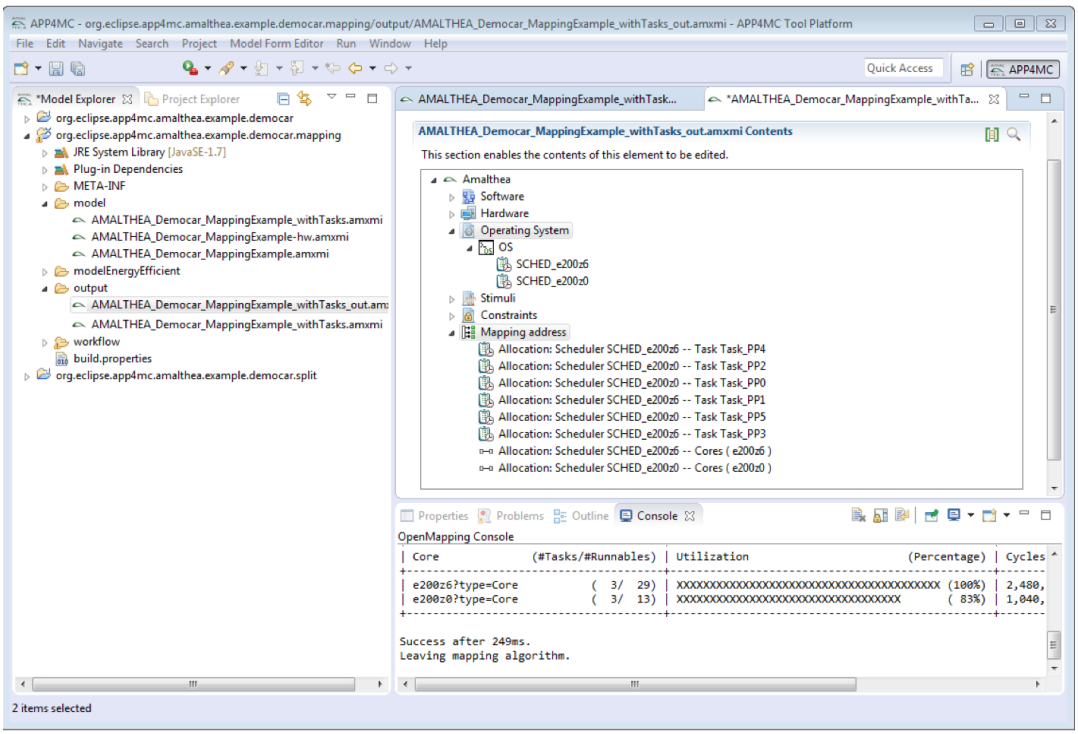

Figure 3.5: The APP4MC mapping tool determines the optimal allocation from tasks to cores (upper half) and illustrates the load on each core (lower half)

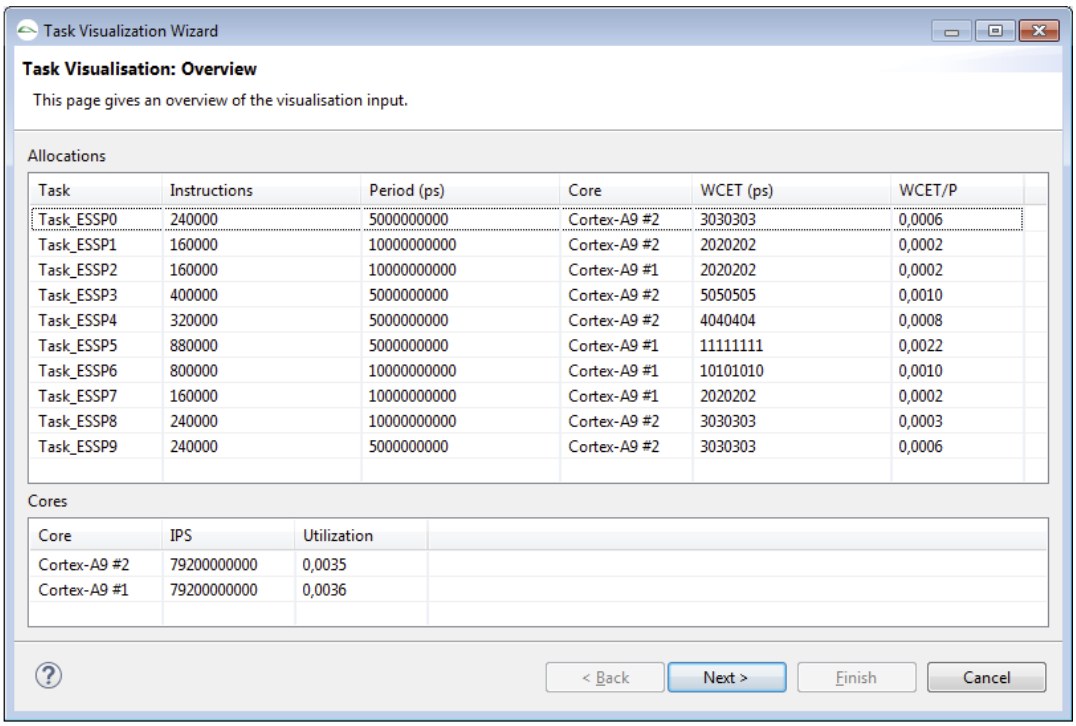

Figure 3.6: The task visualization overview provides basic information about the software as well as the hardware of the system

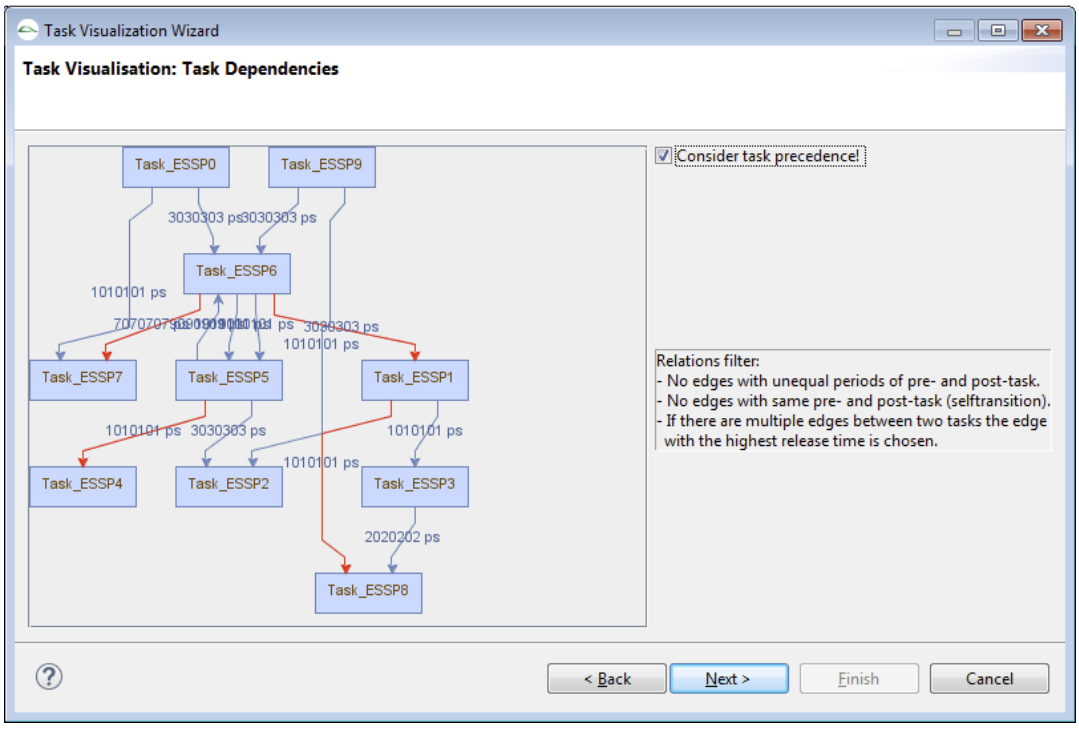

Figure 3.7: The task dependencies overview visualizes the task graph with its inter-task dependencies as well as the communication delay.

Figure 3.8: The gantt chart view of the task visualizer showing the execution of each core. Green means the task is running, yellow ready,red suspended and orange waiting.

inter-dependencies.

3.2.2 Behavior Modeling

In this step the behavior of the software components in the architecture is specified. Behavior models describe the control structure of the system. In some cases the behavior of non-functional requirements is also specified. The APP4MC platform provides support for this activity through the constraint model, component model and software model. Architects can use the models to describe the behavior of the system under development and specify constraints such as timing constraints in the behavior.

3.2.3 Implementation

In this step the required code is produced, tests are developed and executed, the software is integrated and the code is reviewed. The main resulting artifacts are source code and different sets of tests (unit, component, integration) as well as the integrated software and its documentation. The source code can be written from scratch or generated from models in case a model-driven development approach is followed. The software model of the APP4MC platform can be used to generate source code in C.

3.2.4 Validation and Testing

This involves testing of software components to validate if they are working as desired, i.e., according to the specified requirements. For software components that will interact with hardware components, simulations are run in order to fix as much defects as possible before the component can be tested on the actual hardware. Deployable Control Software is a packaged integrated software that is ready to be deployed on a specific hardware.

The APP4MC platform provides two ways of supporting this activity. The first is using check-based validation. The APP4MC platform has a capability to perform check based validation (using Sphinx) on the AMALTHEA model. This validation reveals errors, warnings and information on the model. This validation is run by the programmer when needed. Figure 3.9 show the constraints that can be checked by the validation and Figure 3.10 shows the results of running this check based validation.

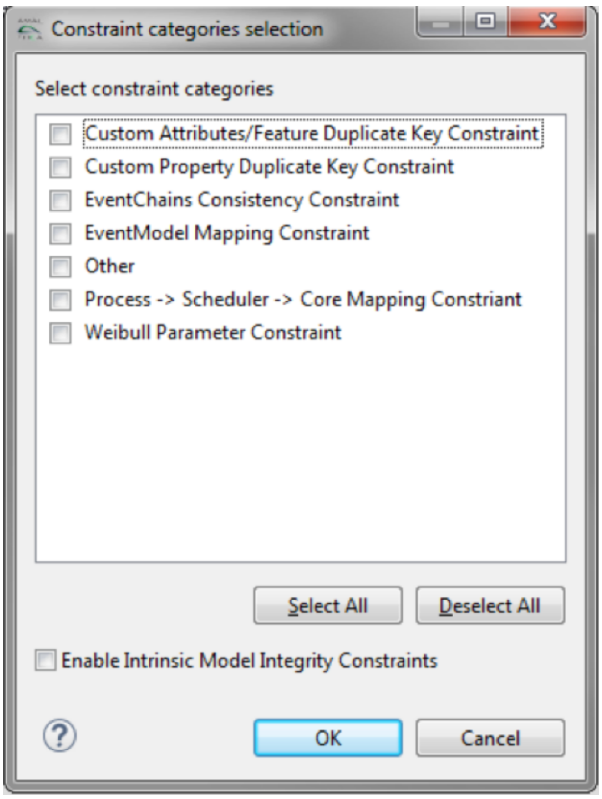

Figure 3.9: Options that can be selected to run the Check-based Validation

The second validation support provided by APP4MC is the Trace model. The trace model gives details on time consumed by tasks to allow refinement of the model to get the most efficient one.

3.2.5 Functional Safety Concept

This is a safety related step where hazard analysis and the risk assessment take place to define the corresponding safety goals as top-level safety requirements. These safety requirements usually lead to constraints that need to be defined and represented somewhere. In the APP4MC platform, the constraints can be defined in the constraints model and other safety requirements can also be represented in the component model.

The APP4MC platform also allows grouping of runnables and labels according to their ASIL levels. This enables the algorithms for partitioning and mapping to take safety information into account.

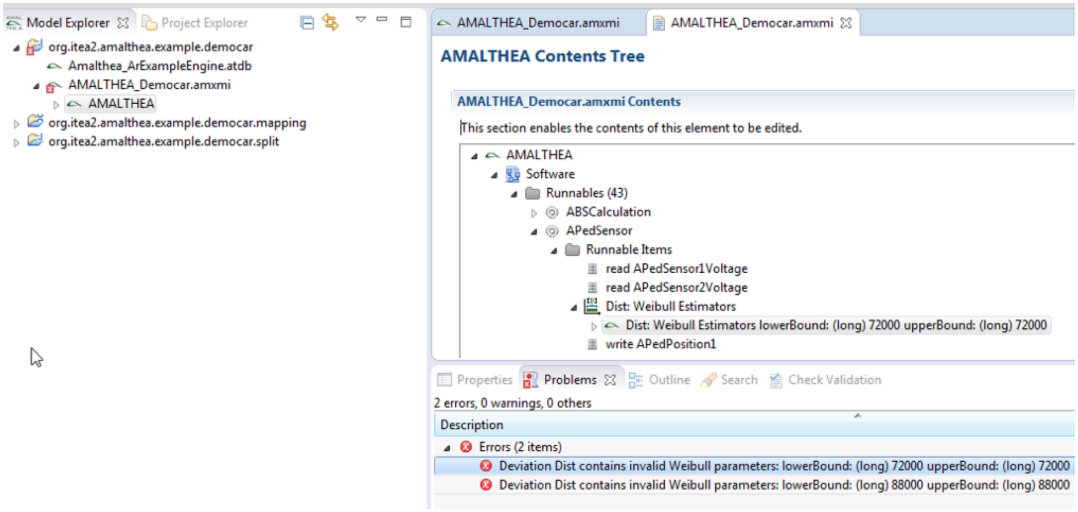

Figure 3.10: Results of running the APP4MC check-bases Validation.

3.2.6 System safety requirements engineering

This activity consists of planning verification/validation activities at system level and definition of system safety requirements based on the systems requirements and the functional safety concept of the system. This also includes the definition of safety-related assumptions caused by the multi-core structure of the system under development. While the APP4MC platform does not provide any support for planning the verification and validation activities, it provides some support for definition of safety constraints. Constraints e.g., timing constraints can be defined in the constraints model and the component model. Hardware related safety constraints can be defined in the hardware model. Also, it must be ensured that the software safety requirements are correct, complete, and consistent with respect to the safety goals and the system design. Again there is no support for the planning of the verification and validation activities, but there is similar support as for **DS A**. The constraint model and the software model can be used to describe and model software safety requirements of the system.

3.2.7 Software safety requirements engineering

This activity is similar to **DS B**, which consists of planning verification/ validation activities at software level, the definition of software safety requirements based on **DS B** (and the system design), and the validation of the software safety requirements against the software requirements (i.e., the part that is not safety-related).

4 Support from third-party tools

As it can be observed from the previous Chapter (Chapter 3), the APP4MC platform does not provide support for all of the design steps. Over the course of the project, this was noted and other tools, which are third party with respect to the APP4MC platform have been developed. The aim of developing these tools was to make sure that there are tools to support all the design steps and that these tools can be integrated with the APP4MC platform. These tools are therefore all Eclipse-based. While most of the tools are open source, a few are commercial but can still be integrated or used together with APP4MC. In this section we describe the tools in details and show which design steps they support. Table 4.1 shows a summary of the design steps that are supported by third party tools developed in the AMALTHEA4Public project.

4.1 Third Party tools

4.1.1 ReqTool

Description: ReqTool is a model and a requirements editor for the Eclipse Capra framework

License: EPL-1.0

Intended Audience: Sales representatives, Project Managers, Software Engineers, Hardware Engineers and Quality Assurance Engineers

Documentation URL: https://github.com/hefloryd/reqtool/blob/master/README.md

Download URL: https://github.com/hefloryd/reqtool/releases

Technology Readiness Level (TRL): TRL 3

Input: Requirements in form of ReqIF, Source code (C/C++, Java, etc.), Test results in form ox XML

Output: XML or Markdown

Design Step Supported: Requiremente related steps which are **DS 1, DS 3, DS 11, DS B and DS C. DS 8, DS 9 and DS D** are supported by allowing for their input and output artifacts to be traced.

Opportunity for Customization: The tools contains a meta-model for requirements in the form of an Eclipse plugin. The plugin can be changed but this currently requires rebuilding the tool. There is not yet any method for dynamically changing the meta-model.

Pre-requisites: The release consists of an Eclipse update site. The update site is not yet hosted anywhere but it can be downloaded and installed from locally.

Design Step	Support by APP4MC
DS 1: System Requirements Engineering	ReqTool, TA Tool Suite, irdleditor, SysML4CONSENS
DS 2: System Architecture Design	TA Tool Suite, SysML4CONSENS
DS 3: Software Requirements Engineering	ReqTool, TA Tool Suite, irdleditor, ScenarioTools
DS 4: Derivation of Product Variants	AMPLE
DS 5: Definition of Software Architecture	TA Tool Suite, MechatronicUML, VaCoMo, SCA2AMALTHEA
DS 6: Behaviour Modelling	TA Tool Suite, irdleditor, MechatronicUML, ConstraintsFromLabelAccesses Workflow Component, SCA2AMALTHEA
DS 7: Variant Configuration	AMPLE
DS 8: Implementation	MechatronicUML
DS 9: Validation and Testing	TA Tool Suite, irdleditor, ScenarioTools, StimuliFromActivations Workflow Component, ConstraintsFromLabelAccesses Workflow Component
DS 10: System Integration	ConstraintsFromLabelAccesses Workflow Component
DS 10.1: Create Executables	
DS 10.2: Partitioning	TA Tool Suite
DS 10.3: Task Creation	TA Tool Suite, StimuliFromActivations Workflow Component
DS 10.4: Target Mapping	TA Tool Suite, StimuliFromActivations Workflow Component
DS 11: Handover	ReqTool
DS A: Functional safety concept	
DS B: System safety requirements engineering	ReqTool
DS C: Software safety requirements engineering	ReqTool
DS D: Safety Validation	TA Tool Suite, MechatronicUML
DS E: Functional Safety Assessment	
DS F: Integration and validation at vehicle level	TA Tool Suite

Table 4.1: Overview of the identified Design Steps and support by third party tools developed in the AMALTHEA4Public project

4.1.2 IFAK RDL Editor

Description: It provides editor support for formal modeling of behavioral requirements. The editor implementation is using Xtext as DSL Framework. Based on the formal requirements, it is possible to generate directly test cases by the integration of external tools for model synthesis and model based test generation. Figure 4.1 shows an example of how requirements can be modeled in text and also shows the graphical representation in terms of a sequence diagram. Figure 4.2 shows the test cases generated from the formal requirements.

License: EPL-1.0

Intended Audience: Test engineers, Requirements engineers

Documentation URL: Not publicly available

Download URL: Not publicly available

Technology Readiness Level (TRL): TRL 4

Input: Requirement specification in form of text

Output: XML

Design Step Supported: **DS 1, DS 3, DS 6 and DS 9**

Opportunity for Customization: Cannot be customized

Pre-requisites: Eclipse Mars (4.5, June 2015)

4.1.3 SysML4CONSENS

Description: This is a SysML profile that allows creating models following the CONSENS specification technique. It is used for both capturing system requirements as well as the system architecture.

License: TBD

Intended Audience: System Engineers

Documentation URL: Gausemeier J, Rammig FJ, Schäfer W (eds.). Design Methodology for Intelligent Technical Systems: Develop Intelligent Technical Systems of the Future. Lecture Notes in Mechanical Engineering, Springer, 2014, Chapter 4.1

Download URL: To be released

Technology Readiness Level (TRL): TRL 3

Input: Initial ScenarioTools specification (Papyrus UML / XMI)

Output: Initial ScenarioTools specification (Papyrus UML / XMI)

Design Step Supported: **DS 1, DS 2 and DS B**

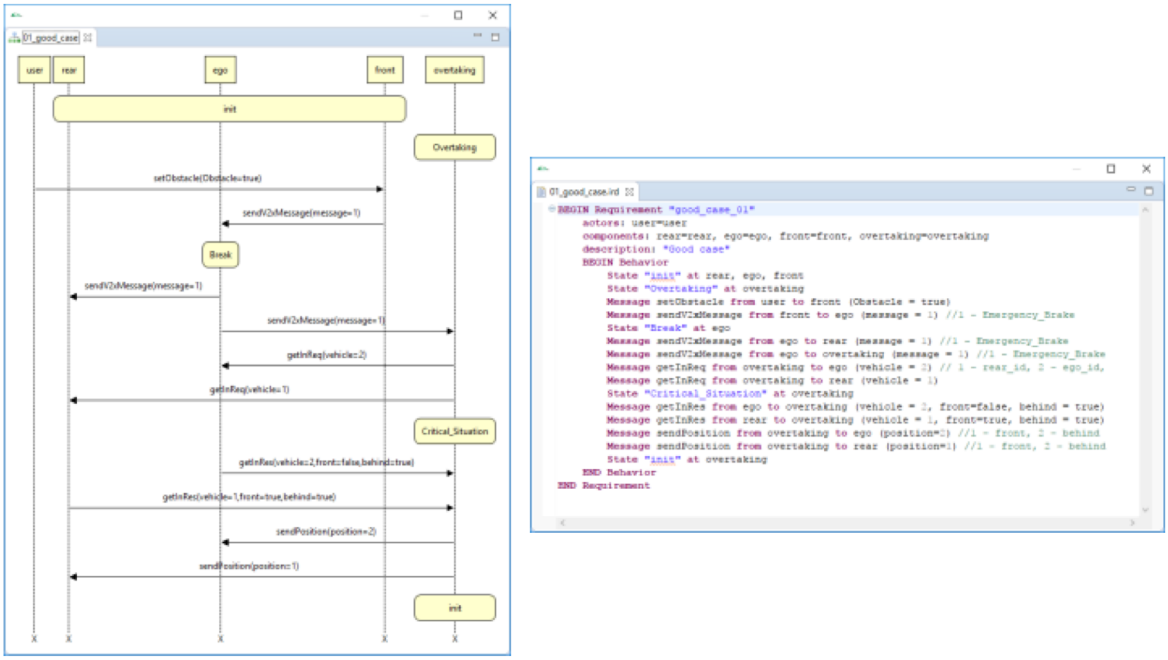

Figure 4.1: Formal requirements modeling.

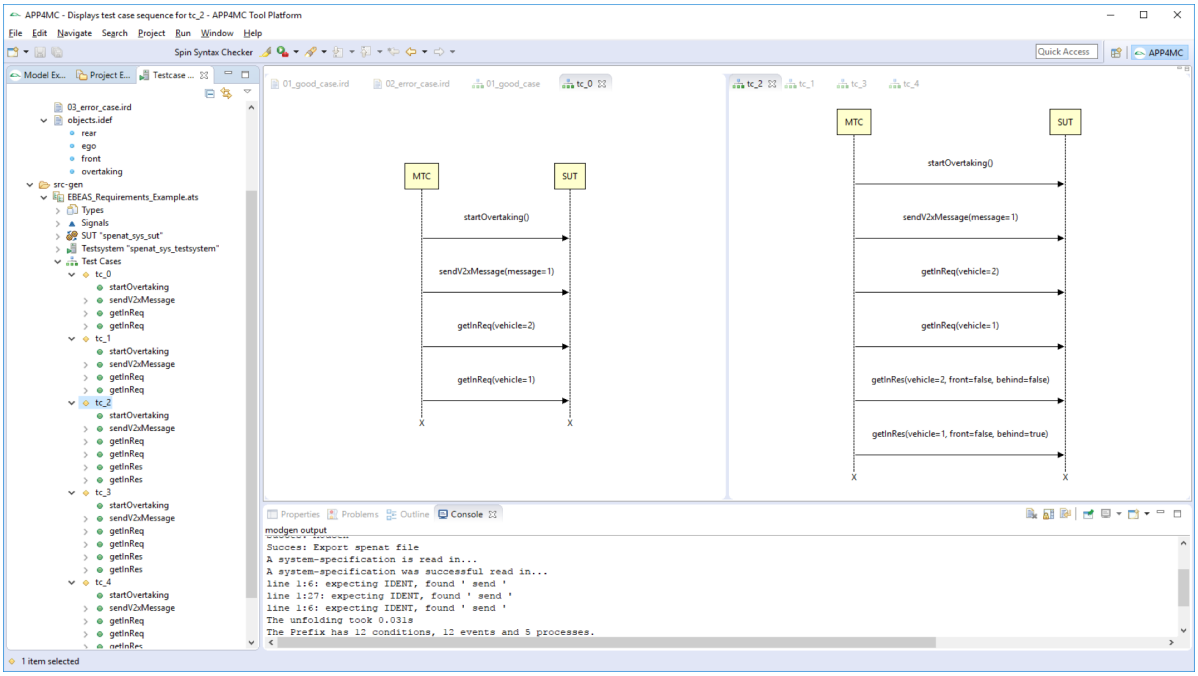

Figure 4.2: Generated Test Cases.

Opportunity for Customization: UML profiles / Papyrus capabilities

Pre-requisites: Papyrus SysML (dependency)

4.1.4 AMPLE

Description: AMPLE realizes a holistic variability management of software and hardware artifacts throughout the entire software development lifecycle (SDLC).A Software Product Line (SPL) is divided into domain and application engineering as well as a problem and solution space. The domain engineering dimension captures all the domain knowledge, i. e. common and variable features of a group of related products, via appropriate models, while application engineering is responsible to derive products based on customer demands. Within the problem space, the scope is stored, i. e. specifications established during the domain analysis and requirements engineering. By contrast, the solution space refers to core assets, e. g. components or code, which form the concrete product. AMPLE make use of a so-called multi variability model (MVM) to manage such an SPL. The MVM integrates two kinds of variability models, where each model takes different characteristics of disciplines like software or hardware into account. Feature Models (FMs) [1] (cf. Figure 4.3), as one type of variability model, are widely used in industry to capture and manage software variability SDLC. A FM comprises a hierarchical arranged set of features connected through different types of association (Mandatory, Optional, Or, Alternative (XOR)) to express variability. Since hardware platforms typically have only few variants, the hardware variability model (HVM) (cf. Figure 4.4) abstract from the associations of a FM and use a simple tree structure instead to cover the hierarchy of a hardware platform along with its elements and properties. Variability is described through specific variation points. Figure 7 denotes the metamodels for the FM as well as the HVM under the term System Family. A System Family allows to describe loosely coupled dependencies between software and hardware elements and enables a combined product configuration (cf. Figure 4.5) by selecting certain product functionalities.

License: TBD

Intended Audience: As system variability as a cross-cutting concern, it affects all abstraction levels, models and artifacts in a SDLC. Thus, all roles for the respective development activities must be aware of variability and the relation to the variability models. Additionally, the initial definition of the product line and the management of the according variability models is done by requirements engineering and product line management.

Documentation URL: DOI: 10.1007/978-3-319-26844-6_32

Download URL: http://193.25.22.150/bitbucket/projects/AMA4PUB/repos/variability_management/

Technology Readiness Level (TRL): TRL 4

Input: N/A

Output: Variability models and product configuration (EMF/XMI), Component Models -via Model-to-Model transformation (Amalthea, Autosar)

Design Step Supported: **DS 4, DS 5 and DS 7**

Opportunity for Customization: The tool can be extended via the extension mechanisms in eclipse (e.g. extension points). This has been demonstrated by the integration of Xtext. Due to the use of EMF and the open meta model, the tool can be customized in any direction, e. g. by adding new properties.

Pre-requisites: Eclipse 4.4

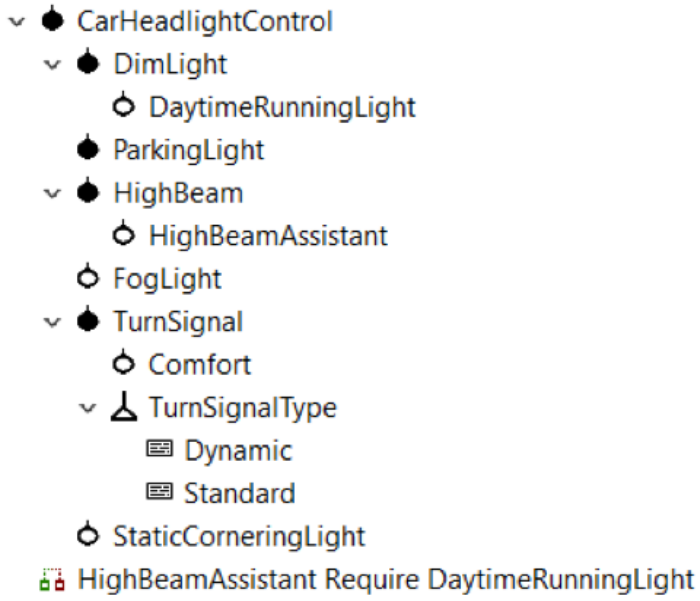

Figure 4.3: Modeling software variability via a feature model.

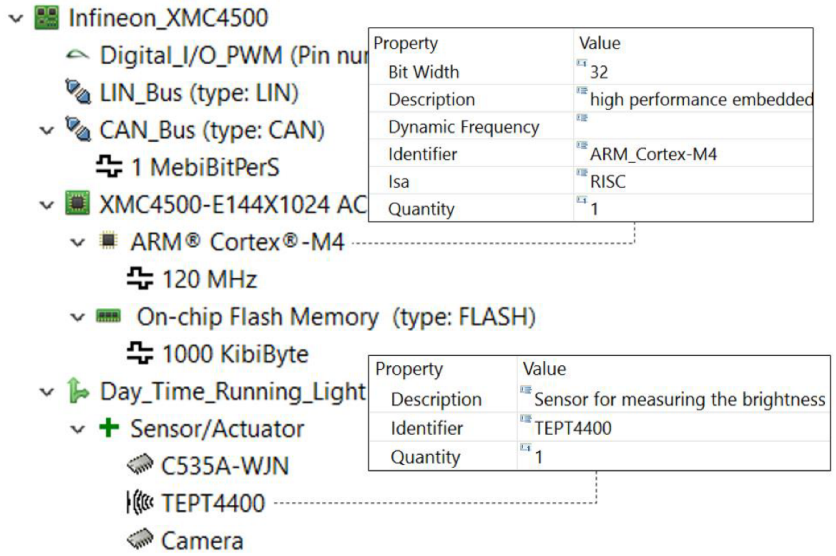

Figure 4.4: Modeling variable hardware platforms and their properties.

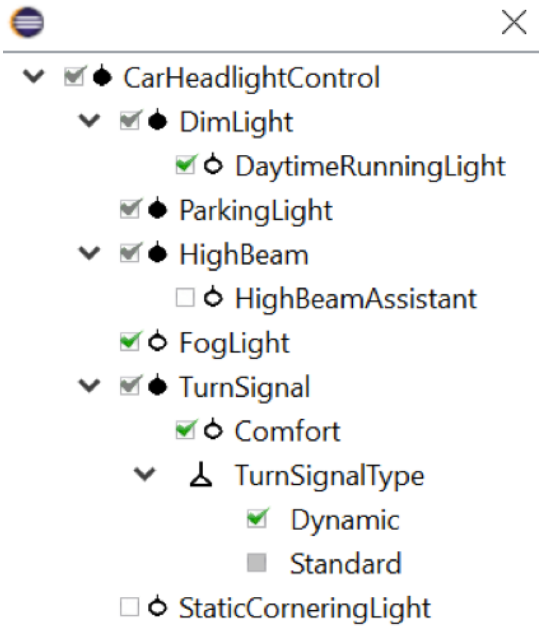

Figure 4.5: Product configuration: Selecting distinct feature of the product.

4.1.5 VaCoMo

Description: In order to apply variability management to the activities of the SDLC, e.g. requirements engineering, models in domain engineering must support variability via variation points. VaCoMo supports the modeling of component models (cf. Figure 4.6), while it covers all product variants within a so-called 150% model. Based on a product configuration, this enables the derivation of product-specific component models by removing optional components and ports. Optional features can be represented by components and ports marked as optional, while an exclusive-or (XOR) is mapped to a Variation Point, which again aggregates a set of optional components. In case components of a Variation Point share the same target port, Variable Connectors can be used. Apart from the variability related structures, VaCoMo make use of common structures: Components represents functional units and are further divided into Hierarchical, Atomic, and Sensor/Actuator. Furthermore, the interfaces of a component can be described via appropriate ports, while connectors allow to connect components among each other via compatible ports. Both, AMPLE (Section 4.1.4) and VaCoMo are realized as Eclipse Plugins by means of customized EMF editors. Based on a product configuration and traceability links between both models, this allows to apply model-to-model transformations for VaCoMo for deriving product-specific component models like AMALTHEA or AUTOSAR.

License: TBD

Intended Audience: VaCoMo is used to define the system's architecture and thus it is relevant for architecture engineering and their roles. However, as the system architecture connects requirements with implementation, also this disciplines may be involved in the system architecture definition.

Documentation URL: DOI: 10.1007/978-3-319-26844-6_32

Download URL: http://193.25.22.150/bitbucket/projects/AMA4PUB/repos/variability_management/

Technology Readiness Level (TRL): TRL 4

Input: N/A

Output: Component Models -via Model-to-Model transformation (Amalthea, Autosar)

Design Step Supported: **DS 5**

Opportunity for Customization: The tool can be extended via the extension mechanisms in eclipse (e.g. extension points). This has been demonstrated by the integration of Xtext. Due to the use of EMF and the open meta model, the tool can be customized in any direction, e. g. by adding new properties.

Pre-requisites: Eclipse 4.4

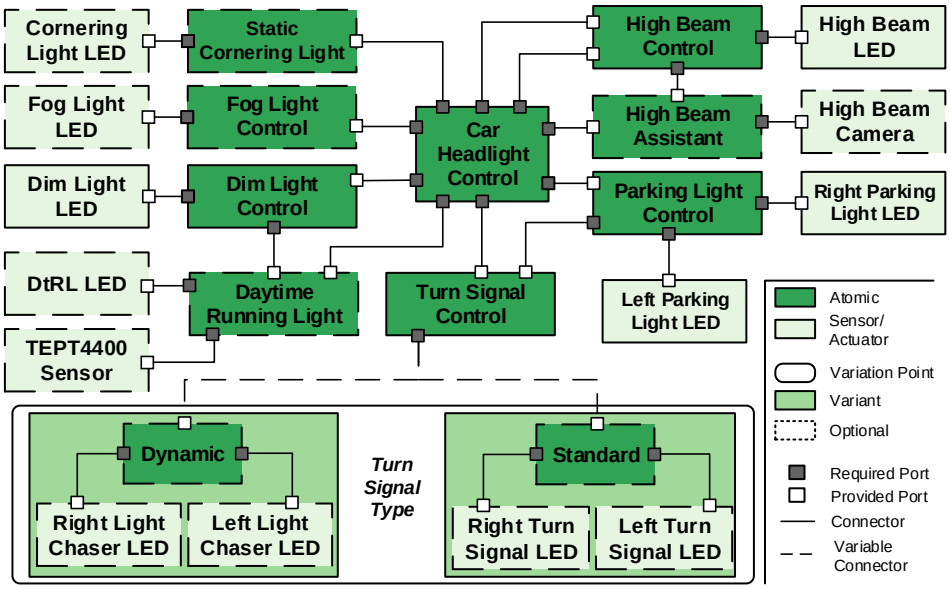

Figure 4.6: Modeling variable component models with VaCoMo.

4.1.6 MechatronicUML

Description: MechatronicUML provides a modeling language, development process, and tooling for engineering the software of software-intensive systems.

License: EPL

Intended Audience: Software Engineers

Documentation URL: http://www.mechatronicuml.org/en/publications.html

Download URL: http://www.mechatronicuml.org/en/download.html

Technology Readiness Level (TRL): TRL 4

Input: N/A

Output: Initial APP4MC model (.amxmi), ANSI C Code (.c*, .h), Timed Automata

Design Step Supported: **DS 5, DS 6, DS 8 and DS D** (timed model-checking)

Opportunity for Customization: Custom properties, metamodel an be extended and custom Eclipse plugins can be integrated.

Pre-requisites: MechatronicUML requires dependencies based on the features one would like to use: E.g., UPPAAL for timed model-checking.

4.1.7 SCA2AMALTHEA

Description: This is a static code analyzer based on C language and LLVM. It is used for analyzing static code out of legacy software artifacts and generating and AMALTHEA model out of the legacy code. This tool is intended for companies that are migrating to multi and many core systems development but already have existing software that also needs to be migrated. Figure 4.7 shows how the tool works to generate an AMALTHEA model.

License: EPL

Intended Audience: SW developer/architects with legacy code

Documentation URL: http://clang.llvm.org/

Download URL: http://releases.llvm.org/download.html

Technology Readiness Level (TRL): TRL 9

Input: C/C++

Output: AMALTHEA datadefintion (.amxmi)

Design Step Supported: **DS 5 and DS 6**

Opportunity for Customization: The inputs can be enhanced by data out of MDX, MSR or AUTOSAR. Additionally, the variance within the source software can be eliminated by using specific configuration files. It is highly customizable according to the user needs.

Pre-requisites: Eclipse installation

4.1.8 Scenario Tools

Description: ScenarioTools provides a formal modeling language for requirements engineering of reactive systems that enables formal analyses techniques (e.g., consistency checks) early in the design process.

License: EPL

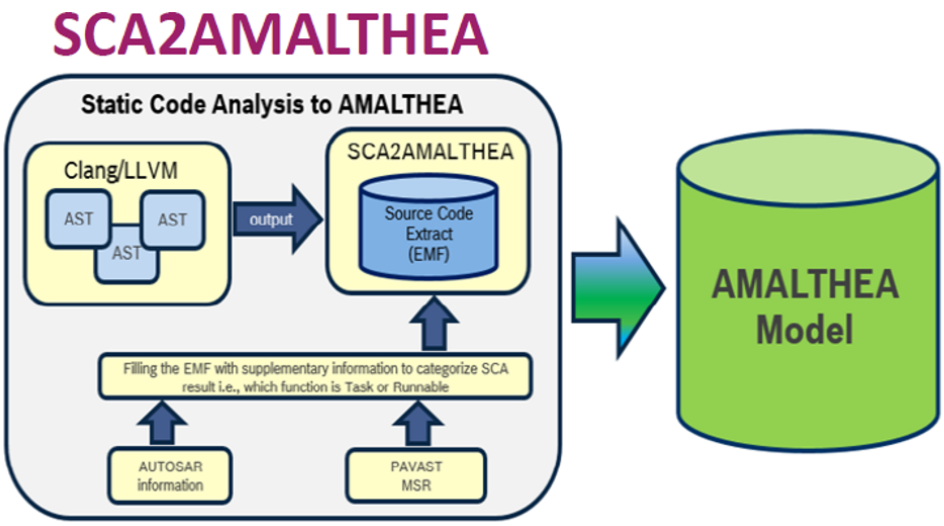

Figure 4.7: Generating an AMALTHEA model from C/C++ code.

Intended Audience: System and Software Requirements Engineers

Documentation URL: http://www.mechatronicuml.org/en/publications.html

Download URL: http://www.mechatronicuml.org/en/download.html

Technology Readiness Level (TRL): TRL 4

Input: Initial ScenarioTools specification (Papyrus UML/XMI)

Output: Complete, formal requirements specification.

Design Step Supported: **DS 3, DS 9** (Formal analysis and simulation) and **DS C** (modeling of safety and timing requirements as well as formal analysis and simulation)

Opportunity for Customization: The metamodel can be extended, custom Eclipse plugins can be integrated.

Pre-requisites: Papyrus (dependency)

4.1.9 Stimuli from Activations Workflow Component

Description: Extend an AMALTHEA model respectively create an AMALTHEA Stimuli model by modeling stimuli based on activations given by AMALTHEA Software model; tasks of the Software model are associated to the newly created stimuli (according to their activations). This is currently restricted to periodic activations.

License: EPL

Intended Audience: System integration, system verification/validation (in general: those disciplines that require a Stimuli model)

Documentation URL: Not publicly available

Download URL: Not publicly available

Technology Readiness Level (TRL): TRL 3

Input: AMALTHEA Software Model (.amxmi)

Output: AMALTHEA Stimuli Model (.amxmi)

Design Step Supported: **DS 9, DS 10.3 and DS 10.4**

Opportunity for Customization: By the workflow component method "setDefaultStimulusName", user can define the base name used for naming newly created stimuli (the stimulus name is combined of this base name and a running number)

Pre-requisites: Eclipse installation

4.1.10 Constraint from Label Access Workflow Component

Description: This tool is used to extend an AMALTHEA model respectively and create an AMALTHEA Constraints model by modeling precedence constraints of runnables and/or tasks based on read and write accesses of runnables to labels. The tool assumes that each label is written only by one runnable and only once.

License: EPL

Intended Audience: System integration, system verification/validation (in general: those disciplines that require a Constraints model)

Documentation URL: Not publicly available

Download URL: Not publicly available

Technology Readiness Level (TRL): TRL 3

Input: AMALTHEA Software Model (.amxml)

Output: AMALTHEA Constraints Model(.amxml)

Design Step Supported: **DS 6** (Derives non-functional requirements (precedence constraints) from label accesses, i.e. data dependencies), **DS 9** (Precedence constraints must be proven to guarantee data consistency) and **DS 10** (Precedence constraints must be considered at basic software configuration e.g,. at task priority assignment.)

Opportunity for Customization: By the workflow component method "setDefaultConstraintsName", user can define the base name used for naming newly created constraints (the constraints name is combined of this base name and a running number)

Pre-requisites: Eclipse installation

4.1.11 TA Tool suite

Description: TA Tool Suite is a fully integrated solution for designing, developing and verifying embedded multi- and many-core systems. Timing-Architects' highly innovative Tool Suite supports the entire project life cycle, including design, simulation, analysis, architecture optimization, and target verification. Figure 4.8 to 4.11 shows how the TA tool suite can be used to model the system (using the TA Explores and Designer modules), simulate how the modeled system will behave (using the TA simulator), optimize the systems (using the TA Optimizer module) and verify timing constraints (using the TA inspector module).

License: Commercial license

Intended Audience: Software Architects, Software Integrator, System Engineers, Testers, Function Engineers and Electronic and Electric Architecture Designers (E/E Architects).

Documentation URL: https://www.timing-architects.com

Download URL: Only available after purchase

Technology Readiness Level (TRL): TRL 9

Input: System Constraints, System Description, ECU Configuration Description (.arxml), Trace recording (csv, OT1, BTF), System constraints, System description (.amxmi), Data specification of an ECUs software system (.mdx), Constraints and requirements(.csv), Field bus configuration (.fibex), Application configuration (.oil)

Output: System Constraints, System Description, ECU Configuration Description (.arxml), System constraints, System description(.amxmi), Data specification of an ECUs software system (.mdx) and Trace recording (.btf)

Design Step Supported: DS 1, DS 2, DS 3, DS 5, DS 6, DS 9, DS 10.2, DS 10.3, DS 10.4, DS D and DS F.

Opportunity for Customization: General Eclipse customization (plug-in support), additional scheduling algorithms via .dlls and command line interfaces to modules for batch automation

Pre-requisites: Minimum Windows 7 OS, Min. 2 GB RAM, Min 4 GB space on HDD, Java 8, CodeMeter software installation from WiBu-Systems AG required and Microsoft Visual C++ 2013 x86 Redistributable required

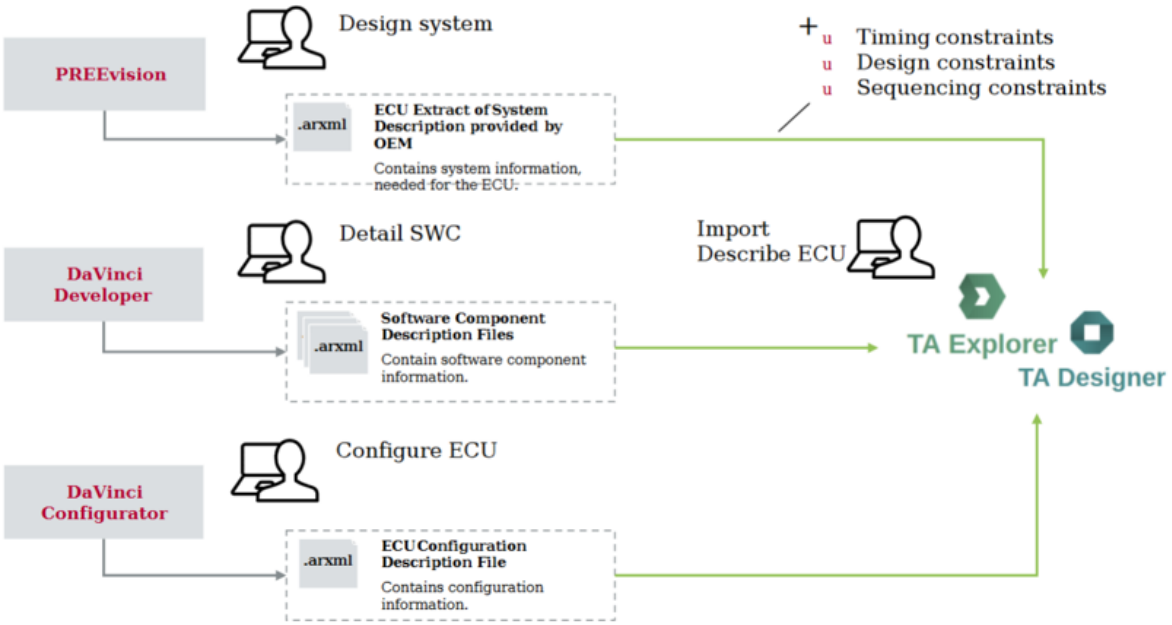

Figure 4.8: System/Software Component Modelling.

Figure 4.9: Simulation of system model based on measured runtimes .

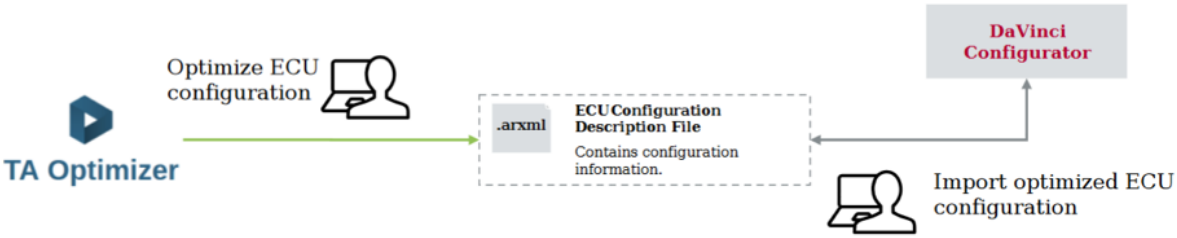

Figure 4.10: Optimization of ECU configuration based on optimized simulation model .

Figure 4.11: Verification of timing violations, bottlenecks, and interactions.

5 Traceability Support

Traceability in software development refers to the ability to link software artifacts like requirements, code, and tests throughout the development life cycle [5]. Traceability facilitates impact analysis, verifying that requirements have been implemented and tested, and in some domains, e.g. in the automotive domain, it is required for fulfillment of safety standards such as ISO 26262 [3]. Traceability is a cross cutting concern since it involves linking not only artifacts from the same design activity but also artifacts that are produced in the different design steps. It is therefore important that a traceability tool can integrate into an existing workflow and process in development companies.

In the AMALTHEA4Public project, we collected requirements for a traceability tool that can integrate into a workflow for the development of embedded systems from a number of industrial partners, mostly in the automotive domain. Based on the collected requirements, Capra[1] has been developed. A major challenge for traceability tool developers is that traceability needs differ from company to company and even from project to project [4, 7]. To build a tool that fits a certain company, one needs to analyze the needs of that company and in most cases the solution will be feasible for that company only. This is not a good business model for commercial tool vendors or open source tool developers who want the same tool to be used in multiple companies. For these reasons, Capra is designed to be a flexible, customizable and extendable tool in order to carter to different needs in industry.

5.1 Capra

Description: Capra is a dedicated traceability management tool that allows the creation, management, visualization, and analysis of traceability links within Eclipse. Traceability links can be created between arbitrary artifacts, including all EMF model elements, all types of source code files supported by the Eclipse Platform through specialized development tools, tickets and bugs managed by Eclipse Mylyn, and all other artifacts for which an appropriate wrapper is provided. Capra is highly configurable and allows users (in a company or project) to define link types that are useful to them. Compared to other similar projects which may have similar features, Capra is not a modeling tool or a tool for requirements management. All functionality is focused on providing traceability capabilities, i.e., the ability to create and visualize links between artifacts modeled in different domain-specific languages. This allows the architecture to be highly modular and the tool to be extremely customizable. The tools also provides notifications when changes are made to artifacts that are connected by traceability links so that the user can also update the traceability links accordingly. Figure 5.1 shows an example of a graphical visualization of traceability links and Figure 5.2 shows an example of a matrix representation of traceability links. Both visualizations are automatically generated by the tool depending on what the user selects.

[1] https://projects.eclipse.org/projects/modeling.capra

License: EPL

Intended Audience: Requirements Engineers, Architects, Testers, Developers

Documentation URL: https://wiki.eclipse.org/Capra

Download URL: `https://projects.eclipse.org/projects/modeling.capra/downloads`

Technology Readiness Level (TRL): TRL 4

Input: Artifacts from different design steps e.g., requirements, models, code, tests, tickets etc.

Output: Traceability model (.xmi)

Design Step Supported: Since traceability is a cross-cutting concern, the tool also provides support for all the design steps

Opportunity for Customization: The traceability metamodel can be extended or even completely replaces, visualization can be customized and the storage mechanisms can also be customized

Pre-requisites: Eclipse Neon installation and PlantUML [2]

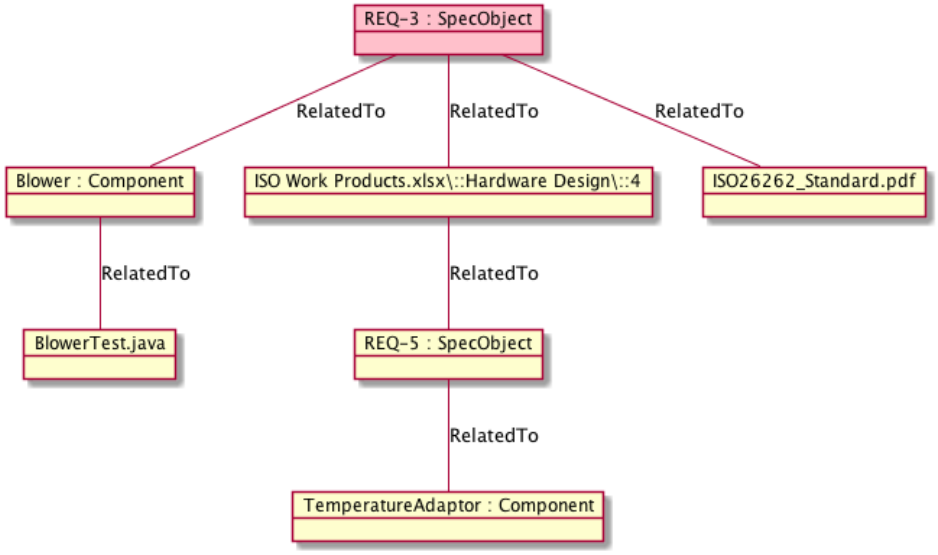

Figure 5.1: Traceability graph resulting from selecting one requirement.

5.2 Use Cases of traceability (with Capra) in Various Contexts

In this section, we give a brief description on how Capra has been customized and used for traceability with various tools developed in the project.

[2] http://plantuml.com/download

	SA_ObstacleInfo_Port : Region	SA_CriticalPointNotifications_Port : Region	SA_LanePositionInfo_Port : Region	A1Situ
SA_ObstacleInfo_Port : Region				X
SA_CriticalPointNotifications_Port : Region				
SA_LanePositionInfo_Port : Region				
A1SituationAnalysisSA_ObstacleInfo_Port_Runnable : Runnable	X			
A1SituationAnalysisSA_CriticalPointNotifications_Port_Runnable : Runnable		X		
A1SituationAnalysisSA_LanePositionInfo_Port_Runnable : Runnable			X	

Figure 5.2: A traceability Matrix. The X mark shows that a link exists between the elements in the specific row and column.

5.2.1 Capra + AMPLE + VaCoMo

As described in Section 4.1.4 and Section 4.1.5, AMPLE is a component modelling tool that allows the definition of variation points and VaCoMo is a variability management too that facilitates modelling of product lines. Together with Capra, these tools provides the functionality to derive product specific requirements and product specific component models. This is done by defining traceability links between the requirements and the feature model and between the feature model and the component model. By deriving specific products from the feature model by selecting which features t include and which features to exclude, corresponding product specific requirements and product specific component models can be derived by following traceability links. This functionality is illustrated in Figure 5.3.

5.2.2 Capra + ReqTool

ReqTool is a requirements management tool that supports creating requirements in an excel like table. ReqTool uses Capra to facilitate traceability to and from requirements. ReqTool demonstrate the capabilities of the Capra framework and the benefits of integrating a requirements tool with Capra and the Eclipse framework. By using these frameworks the ReqTool editor can work with a wide variety of different kinds of artifacts, such as source code elements, Microsoft Word documents and test cases.

This can give software developers a convenient way to manage requirement, their links to other artifacts, and the status of tests for them. The convenience and overview that this provides can result in requirements with a higher quality and better control of the development process and the resulting software.

An example of traceability functionality integrated inti ReqTool is shown in Figure 5.4.

5.3 Integration of APP4MC and DOORS via OSLC

As part of its efforts to support the basic needs of ISO26262 requirements engineering processes, OFFIS focused on integrating the IBM DOORS, an industry standard in requirements engineering, with the Amalthea platform. As a result, OFFIS successfully enabled an OSLC (Open Services for Lifecycle Collaboration [3]) based integration between IBM Rational DOORS Next Generation [4] (in the following IBM DOORS) and the Amalthea OSLC adapter enabling several features such as traceability and visualization of the traceability relations from Amalthea and IBM DOORS. The Amalthea OSLC Adapter provides web access to AMALTHEA Models. Amalthea model elements are provided as HTML and RDF [5] content. The AMALTHEA

[3] http://open-services.net/bin/view/Main/OslcCoreSpecification
[4] https://jazz.net/products/rational-doors-next-generation/
[5] https://www.w3.org/2001/sw/wiki/RDF

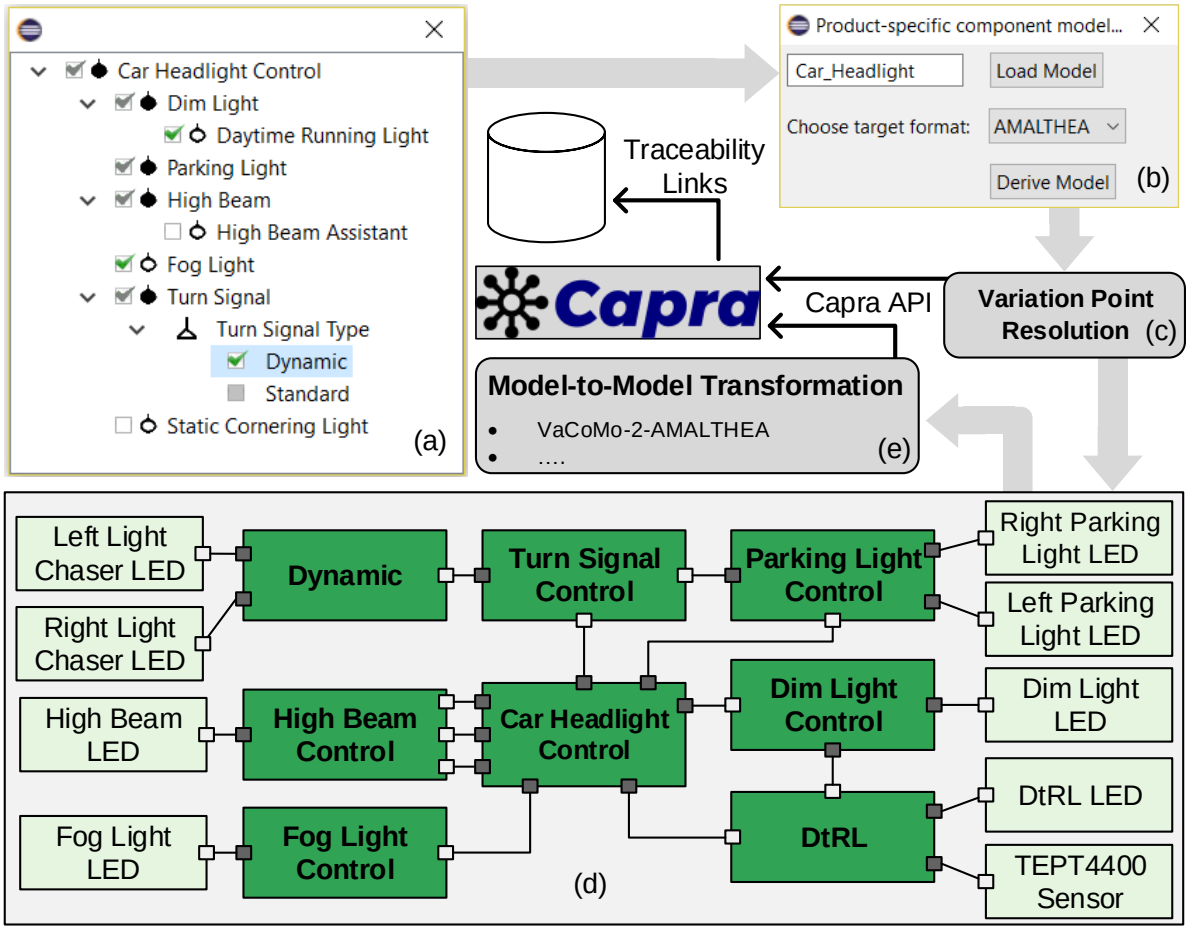

Figure 5.3: Deriving product-specific component models via a product configuration and traceability links.

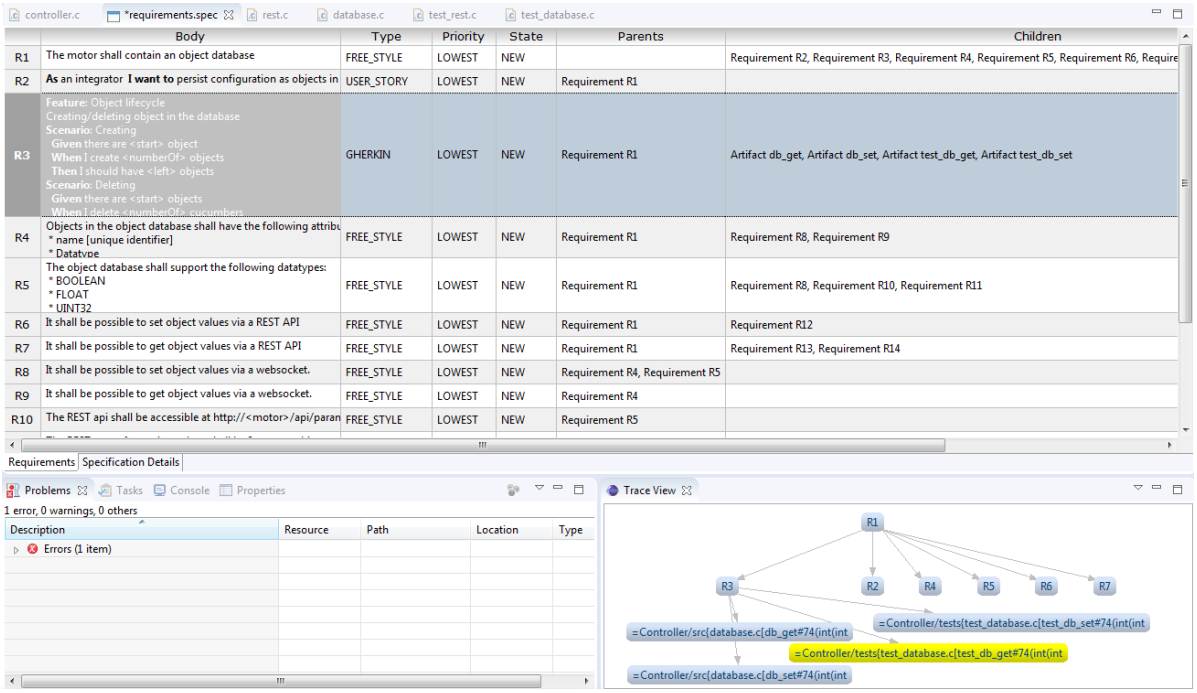

Figure 5.4: The ReqTool editor showing requirements as well as the parents and children traced to the requirements.

OSLC Adapter acts as an OSLC provider for the Architecture Management [6] (AM) domain. It provides all the services required by the IBM Jazz suite for an integration into IBM DOORS. The following services have been implemented:

1. Rootservices [7]
2. OSLC Service providers
3. OSLC Query capability; the OSLC Query language is not supported yet, but we provide a SparQL [8] query engine.
4. OSLC Selection Dialogs: The user can browse the Amalthea model elements in a simple Web page that may be embedded into the web interface of IBM DOORS.
5. OSLC Preview Dialogs: All model elements have also a simple HTML representation. The preview is shown if the user hovers over a link in DOORS or opens an AMALTHEA model element in a web browser.

Some services, such as OAuth authorization, are not fully implemented but provided as stubs. This makes an integration with DOORS possible, although the functionality is still future work.

In order to link between requirements and AMALTHEA model elements, the user must establish a link between the project area in DOORS and the Amalthea model from the adapter.

[6] http://open-services.net/wiki/architecture-management/OSLC-Architecture-Management-Specification-Version-2.0/
[7] https://jazz.net/wiki/bin/view/Main/RootServicesSpec
[8] https://www.w3.org/TR/sparql11-query/

This is done via the project configuration in DOORS; the models that are provided by the AMALTHEA OSLC Adapter are recognized as "Artifact Containers" (Figure 5.5). Now the user can link requirements to model elements via the selection dialog shown in Figure 5.6. After creating the link, it is visible in IBM DOORS. Hovering over the link opens the preview in a speech bubble (Figure 5.7).

The links between requirements and AMALTHEA model elements are currently held in memory by the AMALTHEA OSLC Adapter and are visible in the RDF representation. It is possible to use SparQL to query for linked elements. This makes it possible to use OSLC for analyses on combined models from IBM DOORS and AMALTHEA, for example as OSLC automation on top of IBM Rational Quality Manager [9]. Future work may be to use Capra for storing the links in order to de-couple the inter-domain relations from the models.

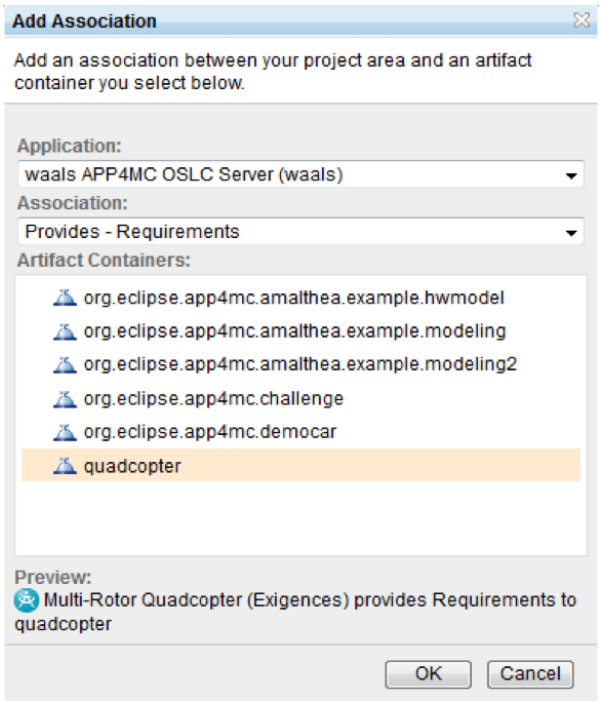

Figure 5.5: Linking a project area in DOORS to an AMALTHEA model.

[9] https://jazz.net/products/rational-quality-manager/

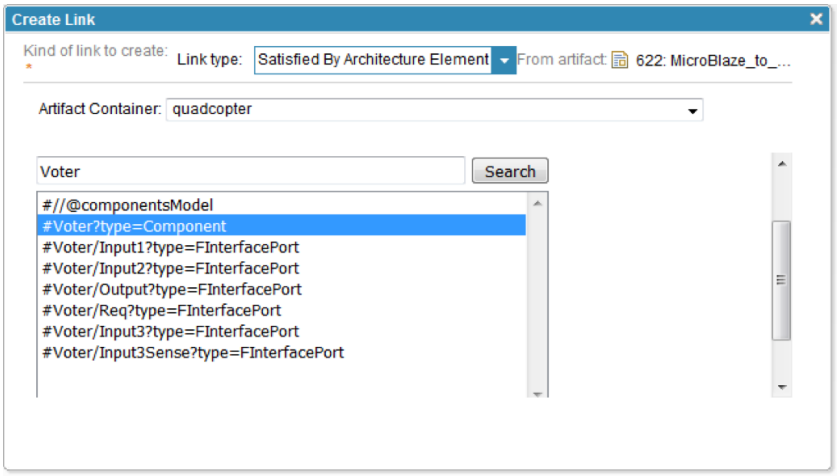

Figure 5.6: Creating a link.

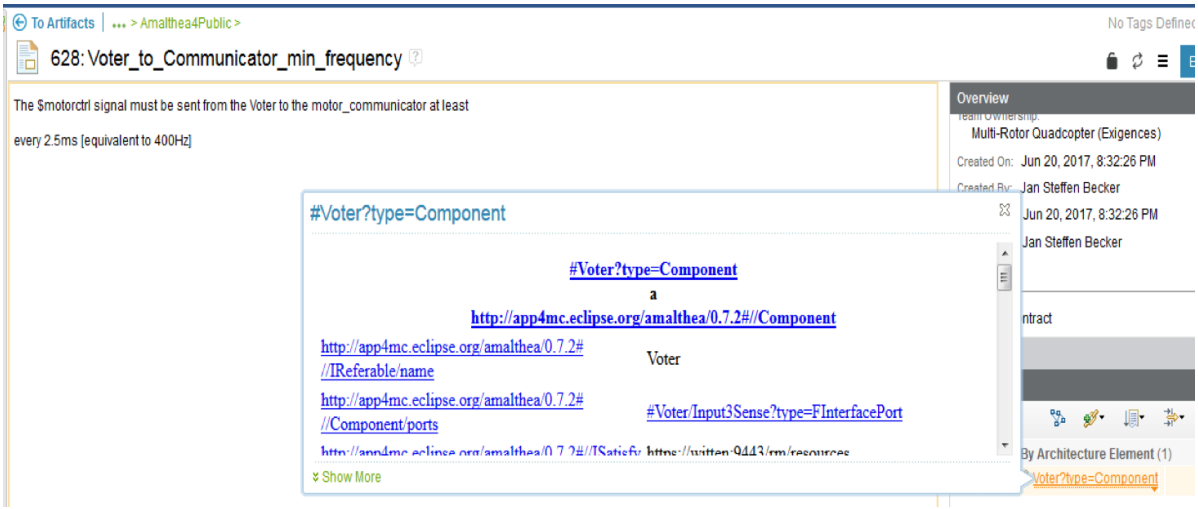

Figure 5.7: OSLC preview.

6 An example of how APP4MC can be integrated with third party tools

This chapter describes how APP4MC can be integrated in a development workflow that uses third party tools. We take an example that uses a model driven approach to develop an embedded system called Emergency Braking & Evasion Assistance System (EBEAS). The system under development (EBEAS) is a system that sends messages to drivers on the road with information on whether they should immediately brake or evade in case of emergencies while driving. An overview of how the system should work is given in Figure 6.1. If the front car detects an obstacle, it should send a message to both the ego, rear and overtaking car with instructions about the obstacle and request the cars to take action to avoid collision.

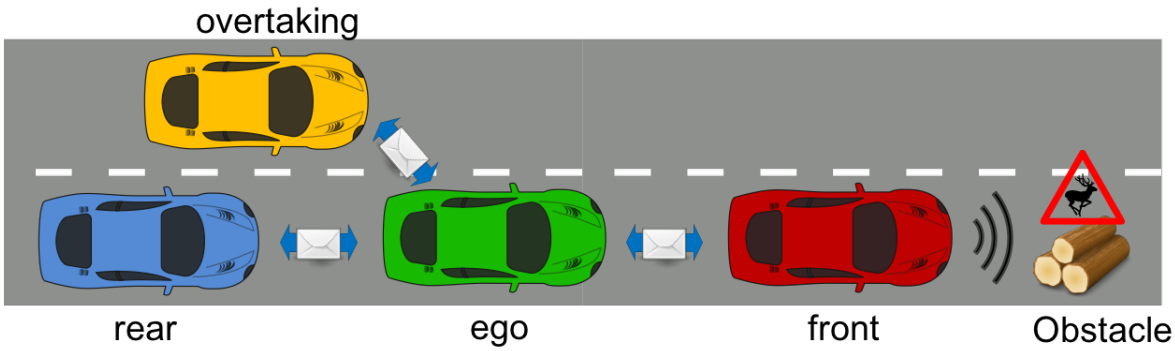

Figure 6.1: Emergency Braking & Evasion Assistance System (EBEAS)

To develop such a system, a V-model is assumed since it is a common development process in the automotive domain. The focus of the development is on the software part of the system and not the hardware part. However, the hardware that the software will run on is still under consideration.

Systems engineering (**DS 1 and DS 2**) is done by using the SYSML profile called CONSES (Chapter 4.1.3). This profile provides several metamodels that can be used to model the System Under Development (SUD) such as the environment model which is used to model the system and its interface to the environment. Once the SUD has been designed, software requirements can be derived and modeled. In this example, the software requirements are modeled using scenarioTools (Chapter 4.1.8). Scenario tools allow the creation of the initial software architecture and specification of formal requirements. Specification of formal requirements means that these formal requirements can also be used for simulation and consistency checks of e.g., timing constraints. The specification of the behavior of the SUD is done using MechatronicUML. MechatronicUML allows for the definition of real-time state charts to describe the behavior of the system. The real-time state charts from Mechatronic UML can be used to generate an AMALTHEA model and the AMALTHEA model generated can be used to generate C/C++

code for the software. This is done through model to model transformations. To connect all the artifacts from system requirements to implementation, Capra (Chapter 5.1) is used. Figure 6.2 shows the tools used in the different steps of the V-model.

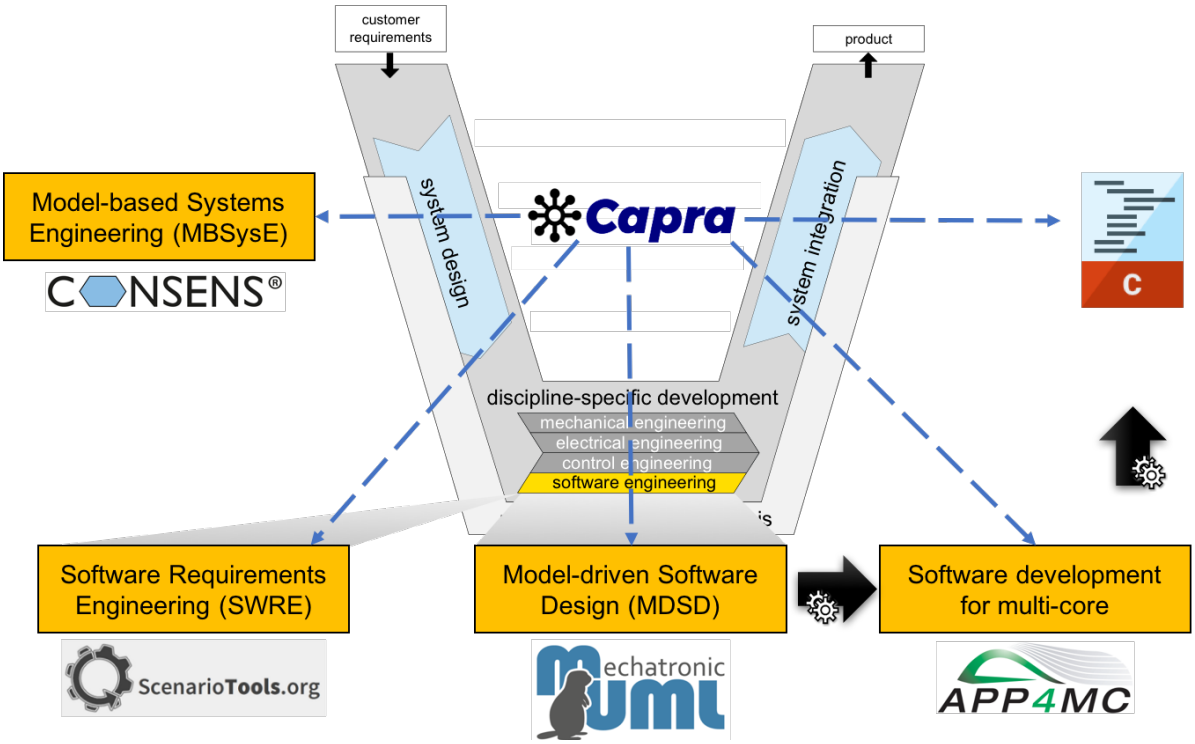

Figure 6.2: Combining the APP4MC platform with third party tools in a development process.

For a more detailed explanation and the EBEAS models created/generated at each step, please refer to [2].

7 Conclusion

This deliverable has given an overview of design steps involved when developing multi-and many-core systems. It has also shown how the tools developed in the project can be used for the various design steps. The aim is to make sure that practitioners both from the AMALTHEA4Public project and outside the project get an understanding of how to develop multi-and many-core systems and also get an overview of the spectrum of tools (mostly open source) available for such development. These tools either emerged from the project or were present and have therefore been further developed by efforts in the project. The deliverable has a special focus on design activities specific for multi-and many-core systems which are **DS 10.2: Partitioning**, **DS 10.3: Task creation** and **DS 10.4: Target mapping**. The APP4MC platform which is one of the main outcome of the project has been designed to support these multi-and many-core activities. Safety related design steps derived from the ISO26262 standard which are necessary when developing safety-critical systems have also been described.

The deliverable has also discussed aspects of traceability and shown that traceability is a cross cutting concern when it comes to systems development. A traceability management tool Capra has been presented with various use cases of how it can be incorporated into the development activities. Further traceability support between APP4MC and a requirements management system known as DOORS, has been demonstrated through an OSLC adapter.

To bring the tools together, an example of an Emergency Braking and Evading System (EBEAS) has been presented which is developed using some of the tools described. The aim here is to show how the different tools can be integrated in a development environment.

As it can be observed from Table 3.1 and Table 4.1, combining the APP4MC platform with third part tools ensures that almost all the design steps have tool support. There is only one step which has no support so far which is **DS E: Functional Safety Assessment**. While this is still an area for further research and development for us, we are aware of commercial tools such as Medini Analyze [1] which have the capability of supporting this step. We therefore recommend that companies use such tools to complement steps not supported by the APP4MC platform and other third party tools presented in this report.

[1] http://www.medini.eu/index.php/en/products/functional-safety

Bibliography

[1] Amalthea4Public Project: D1.1: Analysis of Necessary Design Steps / ITEA. 2015. – Forschungsbericht. Available online

[2] Holtmann, Jörg ; Fockel, Markus ; Koch, Thorsten ; Schmelter, David ; Brenner, Christian ; Bernijazov, Ruslan ; Sander, Marcel: The MechatronicUML Requirements Engineering Method: Process and Language / Software Engineering Department, Fraunhofer IEM / Software Engineering Group, Heinz Nixdorf Institute. Dezember 2016 (tr-ri-16-351). – Forschungsbericht

[3] ISO: *ISO 26262 - Road vehicles — Functional safety.* November 2011

[4] Kirova, Vassilka ; Kirby, Neil ; Kothari, Darshak ; Childress, Glenda: Effective requirements traceability: Models, tools, and practices. In: *Bell Labs Technical Journal* 12 (2008), Nr. 4, S. 143–157

[5] Spanoudakis, George ; Zisman, Andrea: Software traceability: a roadmap. In: *Handbook of Software Engineering and Knowledge Engineering* 3 (2005), S. 395–428

[6] Trei, Maria ; Maro, Salome ; Steghöfer, Jan-Philipp ; Peikenkamp, Thomas: An ISO 26262 Compliant Design Flow and Tool for Automotive Multicore Systems. In: *Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings 17* Springer (Veranst.), 2016, S. 163–180

[7] Winkler, Stefan ; Pilgrim, Jens: A survey of traceability in requirements engineering and model-driven development. In: *Software and Systems Modeling (SoSyM)* 9 (2010), Nr. 4, S. 529–565