 THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE EUROPEAN UNION'S SEVENTH FRAMEWORK PROGRAM (FP7/2007-2013) FOR CRYSTAL – CRITICAL SYSTEM ENGINEERING ACCELERATION JOINT UNDERTAKING UNDER GRANT AGREEMENT N° 332830 AND FROM SPECIFIC NATIONAL PROGRAMS AND / OR FUNDING AUTHORITIES.

CRitical S**YST**em Engineering **A**cce**L**eration

Use – Case Definition

D301.010

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	Use – Case Definition
Deliverable No.	D301.010
Dissemination Level	CO
Nature	R
Document Version	V1.0
Date	2013-10-29
Contact	Oscar Ljungkrantz
Organization	VOLVO
Phone	
E-Mail	oscar.ljungkrantz@volvo.com

AUTHORS TABLE

Name	Company	E-Mail
Oscar Ljungkrantz	VOLVO	oscar.ljungkrantz@volvo.com
Cecilia Ekelin	VOLVO	cecilia.ekelin@volvo.com

REVIEW TABLE

Version	Date	Reviewer
Internal Review	2013-10-10	Mats Larsson
External Review	2013-10-18	Joerg Settelmeier
External Review	2013-10-21	Gerald Stieglbauer

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected

CONTENT

1 Introduction

1.1 Role of Deliverable

This document has the following major purposes:

- Define the overall use case, including a detailed description of the underlying development processes and the set of involved process activities and engineering methods
- Provide input to WP601 (IOS Development) required to derive specific IOS-related requirements
- Provide input to WP602 (Platform Builder) required to derive adequate meta models
- Provide input to the other work packages in WP6 that contain the bricks associated with the use case
- Establish the technology baseline with respect to the use case, and the expected progress beyond (existing functionalities vs. functionalities that are expected to be developed in CRYSTAL)

2 Use Case Process Description

2.1 Use Case

This use case addresses the development process at Volvo Trucks used when developing a new electronic architecture including vehicle functions. In the current process at Volvo, tools for systems engineering and for software development according to *AUTOSAR* [1] are important parts. The AUTOSAR architecture addresses software standardization and enables a common market for automotive software components.

ArcCore, which is involved in this use case, provides a set of tools for AUTOSAR compliant software development. Systemite provides a systems engineering tool called SystemWeaver that is currently used at Volvo for requirements handling, functional design and early architectural and software design, like topology and decomposition of functional components into software components, respectively. Volvo also investigates the use of behavioural modelling, both at early stages to validate requirements, and at later stages for software components verification and generation of code. ASD:Suite by VERUM might be used for the latter case. Simulink is a tool that might be used for both mentioned types of modelling, although the developer MathWorks is not part of the use case. Chalmers investigates the possibility to use sequence/scenario modelling as a complement. Tools for timing analysis and test case generation are also investigated. DTFSim by AIT, Rubus by Arcticus, and Orca by OFFIS might be used for this timing analysis; actually Rubus is currently used at some parts of Volvo. MoMuT by AIT might be used for the test case generation.

The use case presented here is thus a mix of current process at Volvo together with methods and tools that are interesting to investigate and may represent future possibilities. The purpose of the use case is to describe a comprehensive integrated development process although not all parts of it necessarily will be covered in depth within CRYSTAL. The main use case is divided into six sub-use cases, which details different parts of the development process; see Figure 2-1 for an illustration. Although not explicitly depicted in Figure 2-1, the sub-use cases can be performed in an iterative manner as required. Each sub-use case is described in more detail in Section 3.

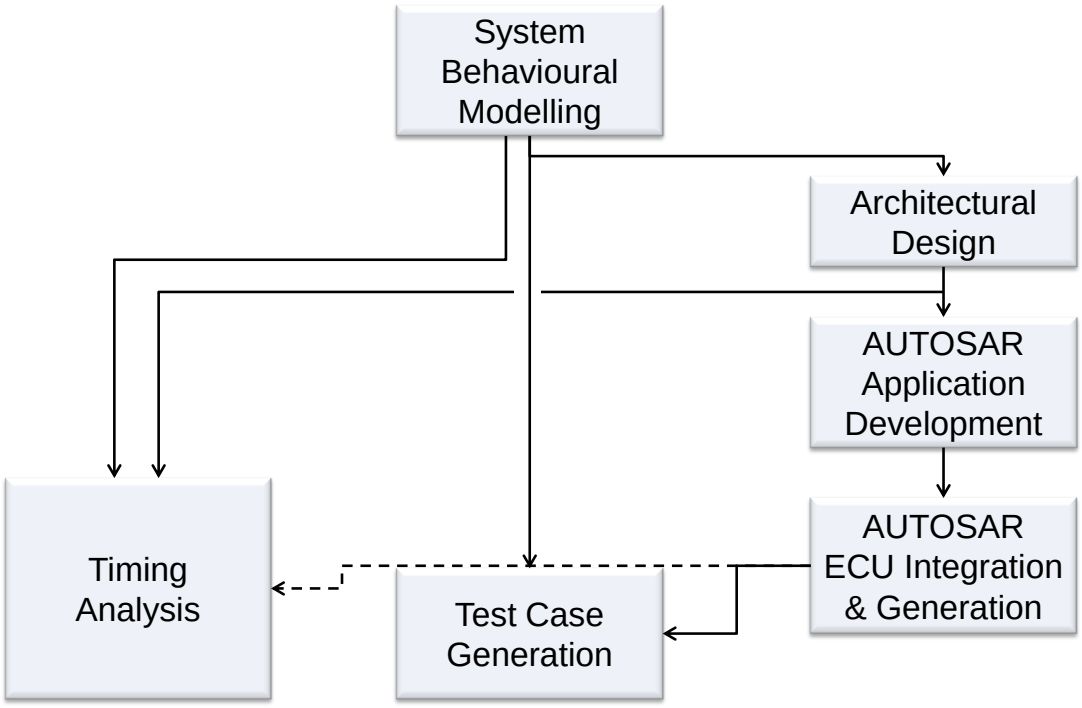

Figure 2-1: Use case overview.

To make the use case tangible it will be applied on a selected functionality called Adjustable Speed Limiter (ASL) which is part of the Vehicle Speed Control and Limitation function. The ASL is used to let the driver activate and adjust a set speed of the vehicle whereupon the engine torque is limited not to exceed the set speed. The main principles of the ASL are simple, but it has a number of conditions for activation and deactivation/overriding making it challenging enough so that the methodology can be fully validated. The main reason for choosing this example, instead of the air suspension system mentioned in the CRYSTAL description of work, is that ASL has a more complete set of requirements and that *behavioural models* (also known as *executable specifications*) in Simulink exist for this example.

2.2 Process

The process is based on the V-model (specification of the system on the left-hand side of the V, including requirements, design and implementation, and test and integration on the right-hand side) but uses a model-based approach which enables early verification of system properties, before reaching the bottom of the V-model. The *activities* contained in the sub-use cases are illustrated as mapped to the V-model in Figure 2-2. Although not explicitly depicted in Figure 2-2, all activities and sequences of activities can be repeated an arbitrary number of times as required, thus creating an iterative process.

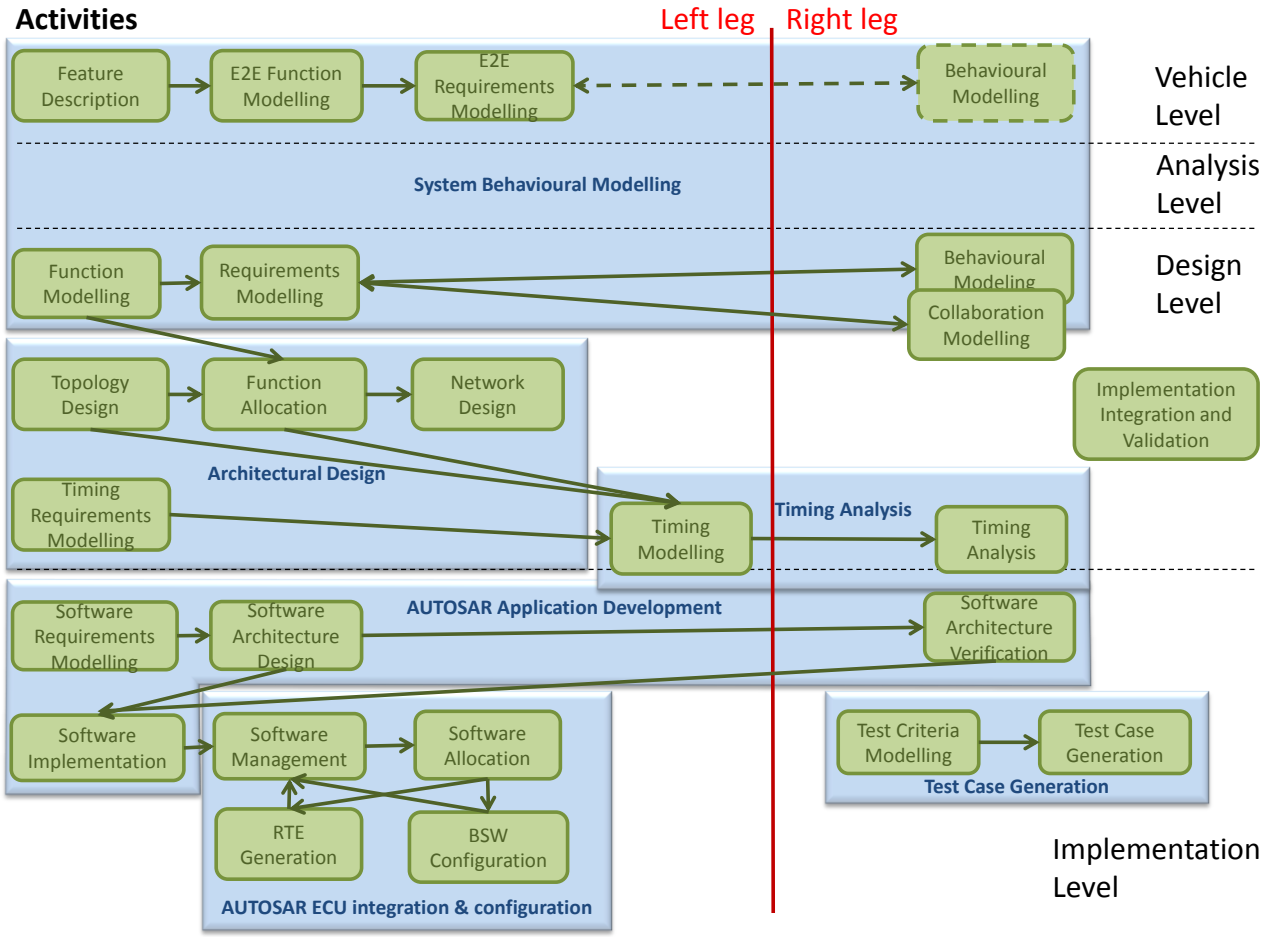

Figure 2-2: The activities of the use case, mapped to the V-model and to the six sub-use cases. The four abstraction levels of EAST-ADL [2] are also shown, but only three of them are currently used: vehicle level, design level and implementation level. The activities to the left are activities about *specification* (requirements, design, implementation, etc.) of the system, while the connected activities to the right are model based *verification* of the system. The unconnected activity to the right represents a conventional right-hand-side V-model activity, in which the results from the implementation level is integrated and verified at the design level. Finally, the dashed part represents a possibility that is not yet defined in the use case.

3 Detailed Description of the Use Case Process

Section 3.1 describes the six sub-use cases in detail, Section 3.2 outlines the interoperability challenges, and Section 3.3 summarises the involved stakeholders.

3.1 Activities

The activities are shown in Figure 2-2 and are further explained here using illustrations of the work flow together with a textual explanation. All used abbreviations and the involved tools developed by CRYSTAL brick providers are summarized in Section 5. In the figures of this section, blue rounded boxes denote the tools used and red arrows indicate the work flow. The work flow arrows may also imply a tool supported data flow. Exactly which data flows that will be implemented remains to be investigated as part of CRYSTAL. Blue lines are used to denote links between data. Entities within the blue rounded boxes represent different assets. Note that although the red arrows indicate a certain work flow, the activities and involved steps can be performed in an iterative manner as required.

3.1.1 System Behavioural Modelling

System behavioural modelling involves determining the vehicle functions, their decomposition into design functions and modelling of their behaviour. An illustration can be seen in Figure 3-1; the numbered steps are explained as follows. The first three steps are on the vehicle level of EAST-ADL and the remaining are on design level.

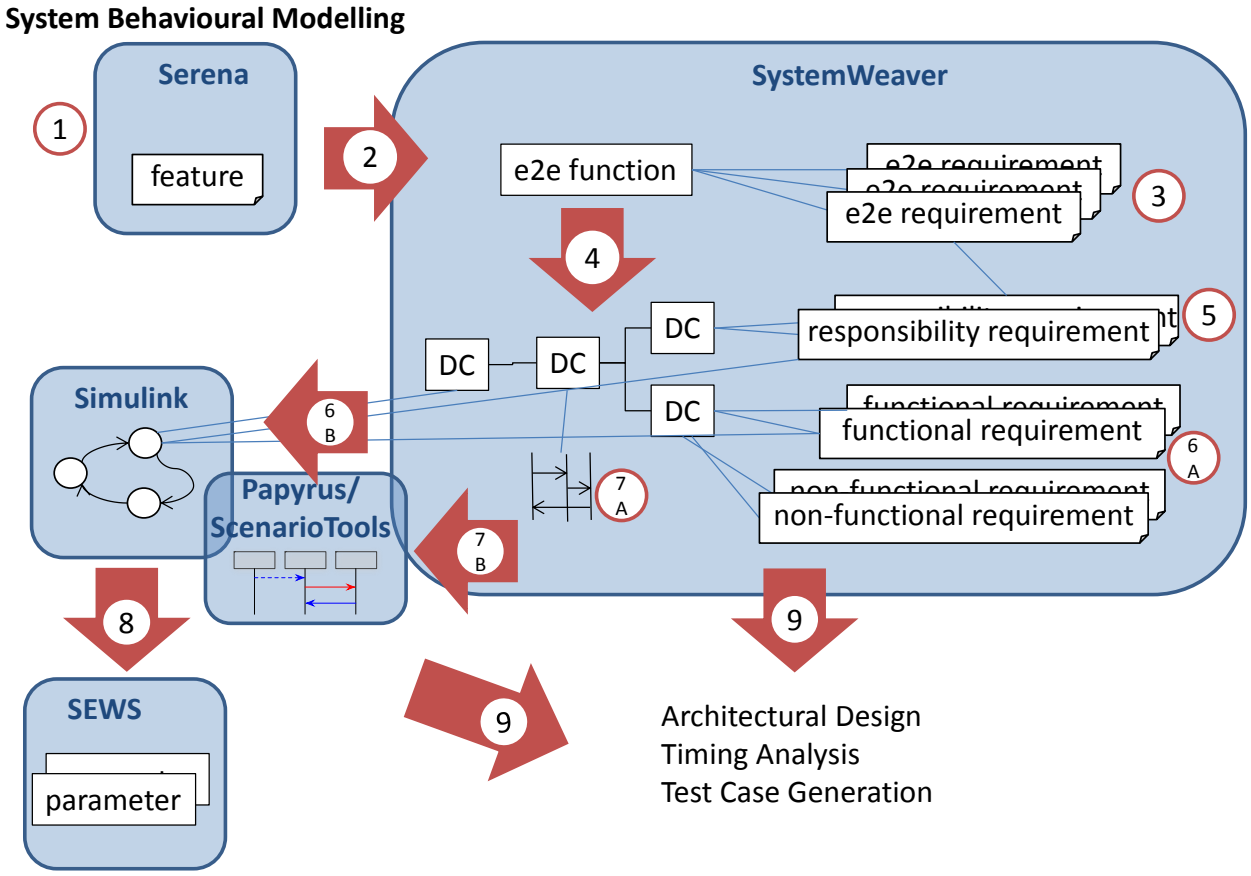

Figure 3-1: The system behavioural modelling sub-use case.

Steps involved:

1. The product planning department wants to introduce a new function. They provide a high-level description of its *features*.
2. The function owner defines *end-to-end* (*e2e*) *functions* to describe features (to establish a "contract" with product planning). The e2e functions can be broken down to use cases; the ASL example is described by a use case.
3. The function owner defines high-level textual *e2e requirements* and links them to e2e functions. At this level, there could be non-functional requirements as well, such as configurability and HMI requirements.
4. The function owner defines functional components, denoted *design components* (*DCs*), to realise the e2e functions. Today at Volvo, the DCs are not defined by the function owner though, but rather taken from a predefined set of Logical Design Components (LDCs). The LDCs are design components that are defined centrally by a group of system architects and that are grouped as, or broken down from, Logical Design Architectures (LDAs).
5. The function owner defines high-level textual requirements denoted *responsibility requirements* on functional components and links them to DCs. In addition, interfaces and variability on functional components are also specified here.
6. *Functional and non-functional requirements*

   A. The algorithms and functionality fulfilling the responsibility requirements can be described as textual algorithmic requirements called *functional requirements*. Textual definition of *non-functional requirements* that cannot be included in behavioural models is also done and linked to the DCs.

   B. The algorithms and functionality fulfilling the responsibility requirements can also be refined into *behavioural models* describing the intended behaviour on the interfaces of the functional components. These models could either be used to validate the textual functional requirements, or replacing them and thus to be treated as requirements. Links to the textual requirements and DCs are made.

7. *Collaborations and scenarios*

   A. *Collaborations* are defined to describe a certain functionality described by an e2e function or a use case in an e2e function. Hence, a collaboration may span several DCs. A collaboration is used for "desktop simulation" (allows manual simulation by the user, no conventional tool supported simulation is performed) to verify function chains; the ASL example is a collaboration. The behaviour of a collaboration is described using *Message Sequence Charts* (*MSCs*) and *scenarios* where each MSC has at least one scenario. A Scenario is a requirement and an MSC acts as a prototype of a test case.

   B. *Modal Sequence Diagrams* (*MSDs*) may also be defined, directly from the textual requirements or as a refinement from the Message Sequence Charts.

8. Parameters related to the behavioural models are extracted.
9. Behavioural models, functional components, MSCs, MSDs and requirements are available for other sub-use cases.

3.1.2 Architectural Design

The architectural design involves the *topology* specification, which consists of *Electronic Control Units* (*ECUs*) and the network (e.g. busses like CAN) that connects the ECUs, as well as the deployment of functional components to the ECUs. Furthermore the *communication infrastructure* is created, to detail the signals and frames on the network. Since it is important that the topology, function allocation and communication infrastructure respect the functional timing requirements, these requirements are also formulated as part of this activity. An illustration can be seen in Figure 3-2. In the following, the involved design steps are presented; all steps are on the design level of EAST-ADL.

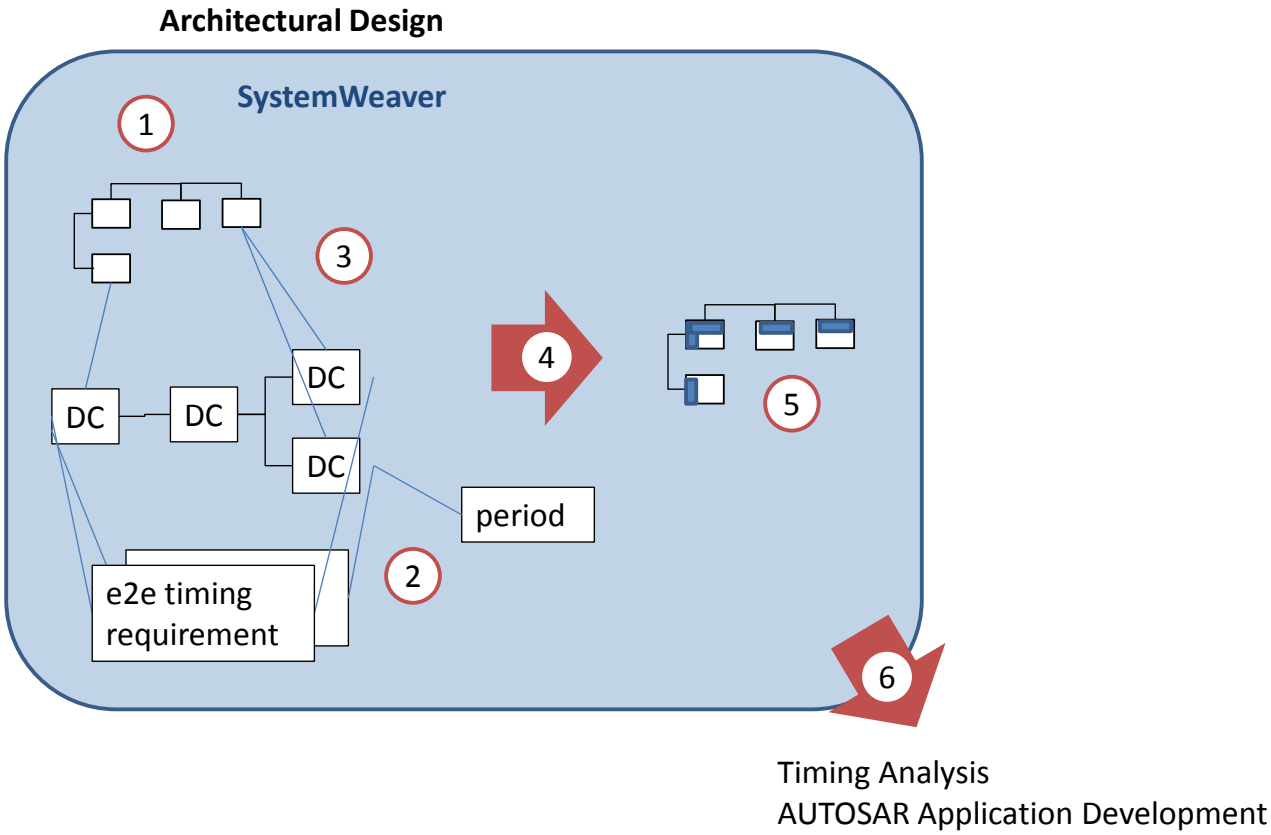

Figure 3-2: The architectural design sub-use case.

Steps involved:

1. The system topology is defined by the system architect. This physical structure is developed in parallel with, and independent of, the logical model represented by the DCs.

   Optionally, the system architect can model architecture alternatives and costs for ECUs. This could be useful:
   - When initially the system is developed and more than one design alternative is possible;
   - When the system is developed in an iterative manner;
   - When a “legacy system” should be extended with new functionalities (with less costs).

2. End-to-end latency requirements are formulated by the function developer using *Logical Component Sequences*. A logical component sequence consists of a sequence of DCs and signals that represent a functional timing dependency together with a maximum allowed latency for the sequence. The DCs are also annotated with execution periods.
3. Functional components (=DCs) are allocated onto the ECUs. Links are introduced to indicate the mapping.
4. The communication infrastructure is created. This involves the following parts:
   a. The system signals (signals that should be transported on the bus) are calculated from the allocated system model.
   b. System signals are packaged into frames.
   c. The frames are scheduled.

5. The output from the allocation and communication work is a complete AUTOSAR system model describing the allocated system architecture. From this it is also possible to generate system models per ECU as input to “AUTOSAR application development”; see Section 3.1.3.
6. Timing requirements, topology, functional allocation and communication infrastructure are available for other sub-use cases.

3.1.3 AUTOSAR Application Development

The AUTOSAR application development involves the specification and implementation of *AUTOSAR software components* (*SWCs*) based on the DCs in an AUTOSAR environment. This is typically done per ECU, and often by a supplier. An illustration is shown in Figure 3-3. In the following, the involved steps are presented; all steps are on the implementation level of EAST-ADL.

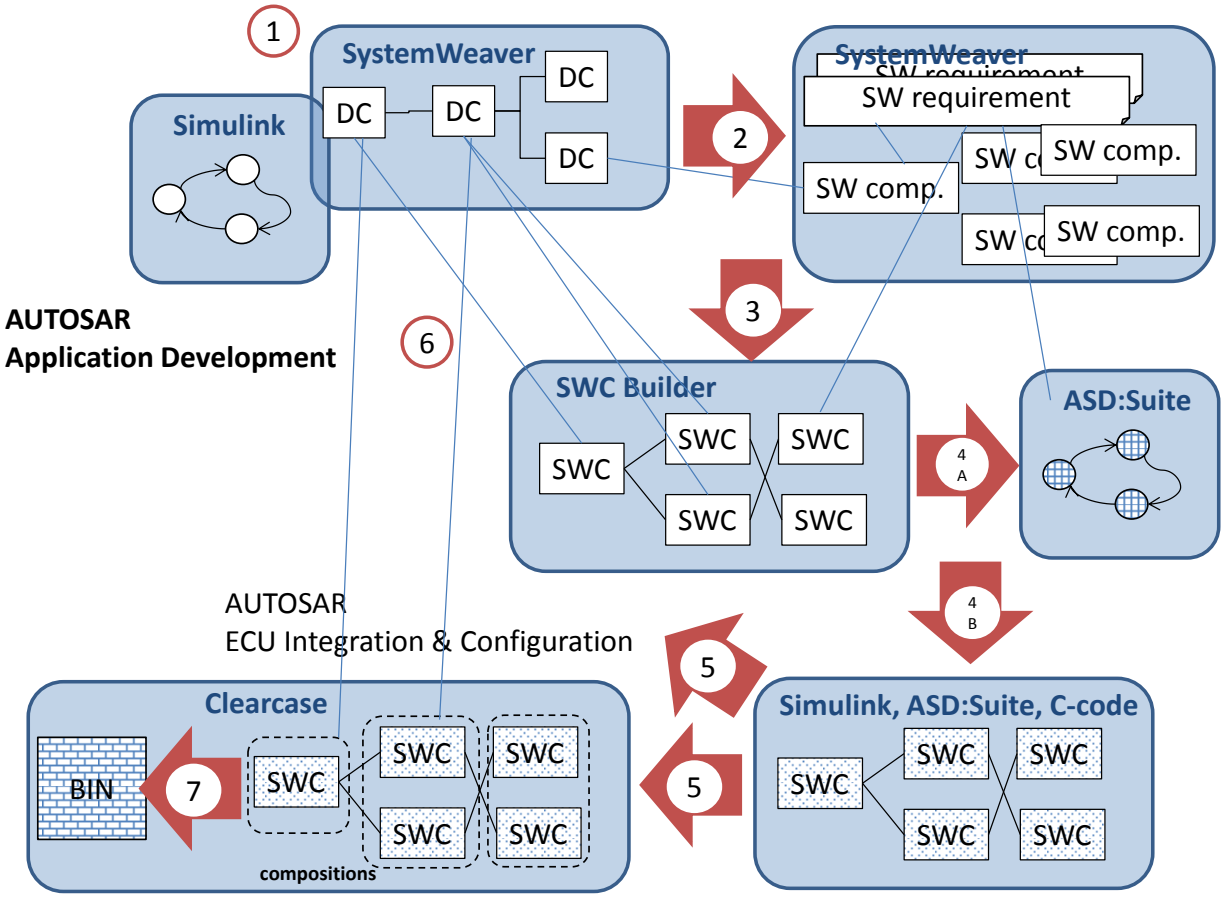

Figure 3-3: The AUTOSAR application development sub-use case.

Steps involved:

1. A system architecture model (DCs + topology and signal interfaces), requirements and behavioural models of the functional components exist.
2. The architecture and behavioural models are made available to the application developer. For in-house development this may mean direct access to the models and tools used by system engineers. For external suppliers, the same approach is assumed here, but in practice specifications in form of documents or standard data formats may be used instead (this also means that the system architect must in practice later integrate and validate the AUTOSAR models from different suppliers from a system perspective; this is covered by the activity “implementation integration and validation” and the related engineering method “validate implementation towards design”, see Figure 2-2 and Section 4

respectively, but is not depicted here). *Software* (*SW*) *requirements* are formulated and a decomposition of DCs into *SW components* is done. SW requirements are linked to the SW components, and SW components are linked to the DCs.

3. The SW components are modelled as *AUTOSAR software components* (*SWCs*). SWCs are linked to the DCs and SWCs are linked to the SW requirements.
4. *SWC behaviour and implementation*

A. For selected components, component behaviour is modelled and is used to verify for completeness and correctness. The models are based on the SWC structure and are linked to the related SW requirements.
B. The behaviour of the SWCs is *implemented* (i.e. code generation or manual coding).

5. The application developer makes the implemented and interconnected SWCs available in a *composition* (per functional component) for "AUTOSAR ECU integration & generation"; see Section 3.1.4. This may be achieved through direct tool interaction or using AUTOSAR XML-files together with binary files or object code.
6. SWC compositions are linked to functional components.
7. Binaries, configuration files and source code are published, that is, checked in for version management.

3.1.4 AUTOSAR ECU Integration & Generation

AUTOSAR ECU integration and generation involves the integration of software components in an AUTOSAR platform. This includes generation and configuration of platform services. An illustration is shown in Figure 3-4. In the following, the involved steps are presented; all steps are on the implementation level of EAST-ADL.

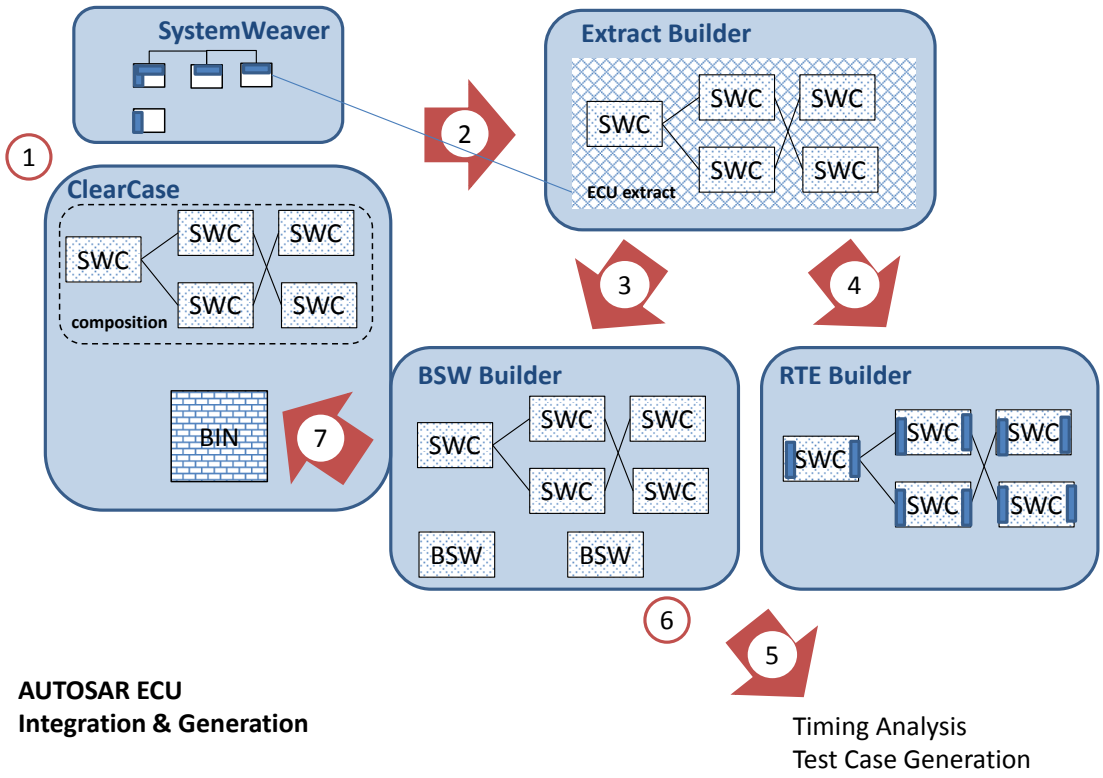

Figure 3-4: The AUTOSAR ECU integration and generation sub-use case.

Steps involved:

1. DCs have been implemented as compositions of interconnected SWCs. Source code and/or object code are available. The communication infrastructure including signal packaging and allocation information is available from the sub-use case “architectural design”; see Section 3.1.2. Additional diagnostic specification and scheduling requirements may be available as well, for instance from the SEWS tool; see Figure 3-1.
2. The compositions are integrated into one *ECU Extract* per ECU by the ECU integrator. The ECU extract in Extract Builder is linked to corresponding ECU element in SystemWeaver.
3. All needed *Basic SW* (*BSW*) modules are configured.
4. The *Run-Time Environment* (*RTE*) is configured and so called runnables are mapped to operating system tasks.
5. Configuration parameters and relevant data are made available to the sub-use cases “timing analysis” and “test case generation”; see Section 3.1.5 and 3.1.6, respectively.
6. Source code for RTE, BSW and other AUTOSAR-related platform artefacts are generated.
7. The AUTOSAR architecture, generated code and binaries are published, that is, checked in for version management.

3.1.5 Timing Analysis

Timing analysis particularly involves verification of the timing requirements but could also include analysis of the timing behaviour. An illustration is shown in Figure 3-5; the numbered steps are explained as follows. The steps are done on design level or implementation level of EAST-ADL, depending on the characteristics and capabilities of the tools, and on the information available.

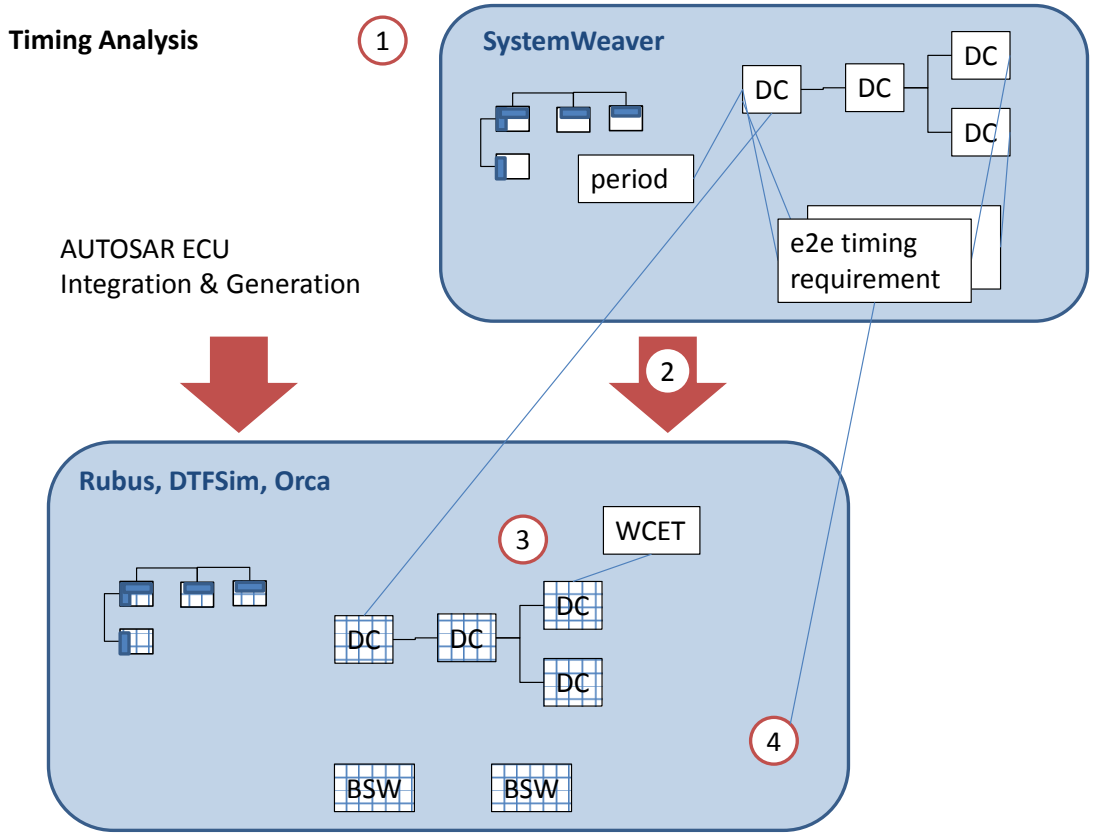

Figure 3-5: The timing analysis sub-use case.

Steps involved:

1. Functional components, timing requirements, system topology, and communication infrastructure have been defined in the sub-use cases “system behavioural modelling” and “architectural design”; see Section 3.1.1 and 3.1.2, respectively.
2. System model, topology and timing requirements are made available for the timing analysis tools.

   (Optional) For the Orca tool, alternative architectures and costs/safety requirements should also be available in order to optimize the task deployment with respect to costs as well as timing and safety analysis.
3. Functional components are annotated with important timing properties - possibly based also on data from the sub-use case “AUTOSAR ECU integration & generation”; see Section 3.1.4. Corresponding model elements in the timing tool are linked to the elements in the system modelling tool (e.g. DCs, periods).

   DTFSim needs besides the topology, network parameters (communication infrastructure plus details from the AUTOSAR ECU Integration & Generation), and the end-to-end latency requirements, also component parameters. Some of the components parameters are available but other parts such as *Worst Case Execution Time (WCET*) need to be assumed based on rules of thumb.

   Rubus needs similar information as for DTFSim. Timing information from each ECU like periods, priorities and WCETs are needed, where the two latter need to be assumed based on rules of thumb. As part of this use case it will also be investigated whether or not Rubus can be adjusted and utilized to perform timing analysis with less precision on higher abstraction levels, that is, earlier on in the development process; in that case, the use case description will be complemented accordingly.

   For Orca to be able to perform optimisation regarding architecture and cost, different alternative design architectures with different costs must be available. Optimized solutions regarding architecture design and costs are then provided.
4. The system is analysed for compliance with the end-to-end latency requirements and the results are linked with the e2e timing requirements.

   The DTFSim tool is a discrete-event simulation environment which focuses on design and analysis of the network architecture of electronic control systems. For this purpose, so-called event chains from sensors to actuators, including buses (CAN, FlexRay or Ethernet), are modelled and simulated.

   The Rubus tool is not based on simulations but performs pre-runtime analysis (different types of response-time analysis as well as end-to-end latency analysis). It can analyse data based on EAST-ADL and it supports CAN buses today; Ethernet support is under investigation.

   The Orca tool can perform scheduling analysis and architectural optimisation. In the latter case, links back to the architecture is made. For the timing analysis, exact results are provided. Automatic deployment of tasks based on timing analysis results can be also envisaged.

3.1.6 Test Case Generation

Test case generation involves the generation of test cases based on requirements and behavioural models. An illustration can be seen in Figure 3-6. In the following, the involved design steps are presented; all steps are on the implementation level of EAST-ADL.

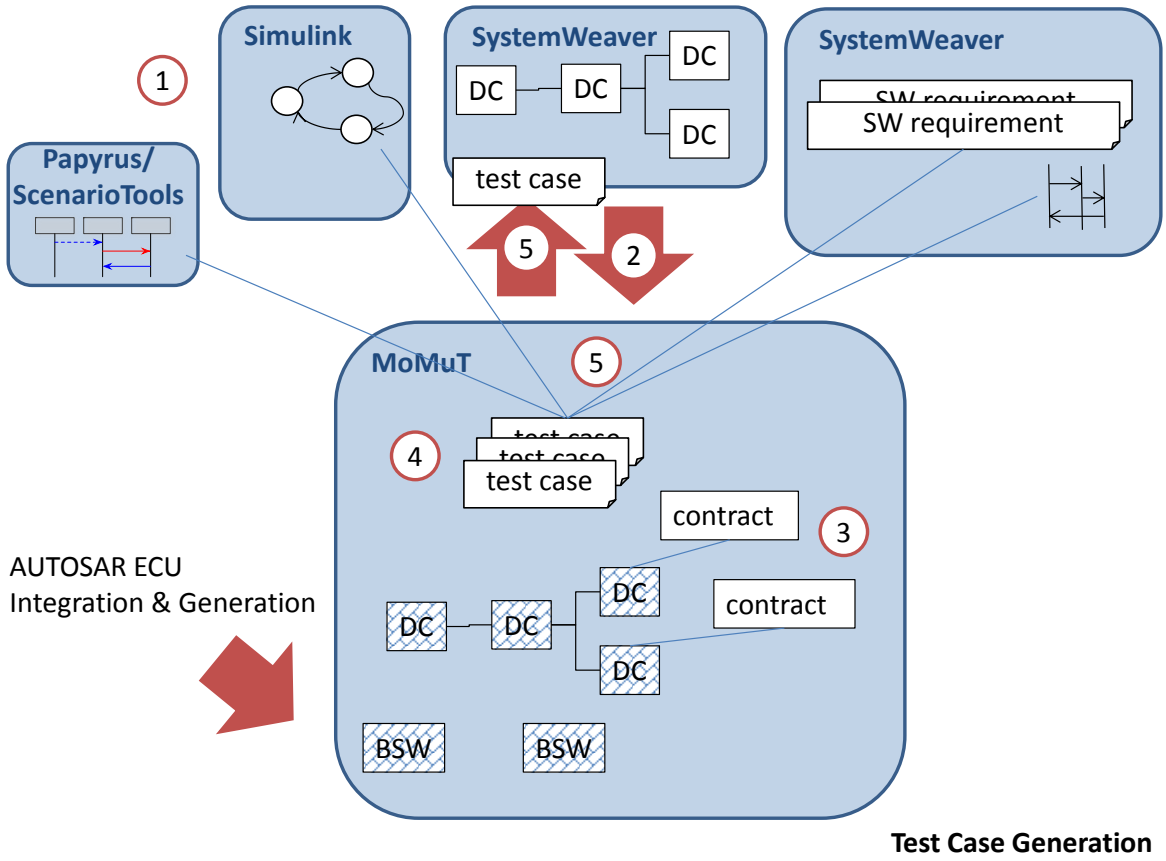

Figure 3-6: The test case generation sub-use case.

Steps involved:

1. Behavioural models, requirements, MSCs, MSDs, and functional components have been defined in the sub-use case “system behavioural modelling”; see Section 3.1.1.
2. The models are made available to the test case tool.
3. The system models are augmented with information needed for test case generation – possibly using data from the sub-use case “AUTOSAR ECU integration and generation”; see Section 3.1.4. Corresponding model elements in the test case tool are linked to the elements in the system modelling tool (e.g. DCs).
4. Test cases are generated on DC and possibly integration level. Existing test cases from prior iterations are reused as far as possible.
5. Test cases are linked to requirements, MSCs, MSDs, and/or behavioural models. Furthermore, the full test cases are made available to SystemWeaver. Note that performing the actual tests is not in the scope of this use case.

3.2 Interoperability Challenges

The interoperability challenges in the Volvo use case come in two forms: i) establishing and maintaining data links across tools, and ii) exchanging whole models or parts of models across tools. The first form is necessary to enable traceability and consistency across tools. This means that it should be possible to denote e.g. that two modelling entities in different tools in fact represent the same entity or that they are related somehow. In a central information model these kinds of links are typically already present, e.g. the SystemWeaver meta-model used in the Volvo use case contains many links already. The challenge is to extend these links to also include other tools not using the SystemWeaver meta-model. However, this form

of interoperability assumes that the models in the different tools have been made independently from each other. That is, before the links can be added, individual models must exist in the tools of interest. Typically these models are manually constructed. In many cases it would be more efficient to be able to generate a model skeleton from an already existing model. This would not only speed up the model construction, it would also enforce consistency of the generated model with the existing. Moreover, the generated model could be formed according to well-defined guidelines, which simplifies understanding of the model and automated analysis. Therefore, the Volvo use case considers also the second form of interoperability: model exchange. Due to that a large portion of the system data is already available in the SystemWeaver meta-model and this meta-model is based on an early version of EAST-ADL, the intention is to use EAST-ADL as the exchange format between tools. For lower abstraction levels, AUTOSAR formats will be used.

3.3 Stakeholders & Roles

The stakeholders explicitly mentioned in the described process, together with their roles, are shown in Table 3-1.

Table 3-1: Main stakeholders and their roles.

Stakeholders	**Role**
Product planning	Decides the vehicle features
Function owner	Specifies the vehicle functionality
Application developer	Implements the vehicle functionality in software
System architect	Decides on topology and mapping of functionality to ECUs
ECU integrator	Integrates the SW on one ECU

4 Identification of Engineering Methods

An Engineering Method describes *how* an activity can be conducted using guidelines, tools and languages that interoperate with each other. It can be applied to one or more activities. All identified engineering methods are shown in Table 4-1. These engineering methods denote possibilities of the involved bricks and tools related to this use case, but note that all these engineering methods may not necessarily be fully implemented.

Table 4-1: Overview of the identified engineering methods.

Name	**Purpose**
Requirements Modelling	Model the textual requirements (from SystemWeaver) in Simulink
Requirements Modelling with Modal Sequence Diagrams	Model the textual requirements (from SystemWeaver) using scenarios
Allocation of Design Functions	To allocate the design components to the target where they will run
Network Design	To create a communication matrix
Define End-to-End Function Variability	To define the variability of end-to-end functions
Create AUTOSAR Interface Contract	To Export AUTOSAR Interface Definitions as the "interface contract" towards suppliers
Validate Implementation towards Design	To validate that the implemented Software complies with the Design specification
Design AUTOSAR SW Application Architecture	The application developer wants to create software components from system design in SWC Builder
SW Component Modelling and Verification	The application developer wants to model the external and internal behaviour of the SW components and ensure completeness and correctness.
Generate ASD SW Components	The application developer wants to generate correctly working source code from ASD models and integrate this in the software environment
AUTOSAR ECU Integration - Extract	The ECU integrator creates an ECU Extract from the software components in Extract Builder
AUTOSAR ECU Integration - RTE	The ECU integrator configures an RTE for the ECU in RTE Builder and BSW Builder
AUTOSAR ECU Integration - BSW	The ECU integrator configures the BSW for the ECU in BSW Builder
Timing Analysis	The System Designer wants to check if the system design meets its timing requirements
Test Case Generation	Provide test cases from requirements model for component or system testing

5 Terms, Abbreviations and Definitions

Table 5-1 shows abbreviations and acronyms used throughout the report and Table 5-2 shows the involved tools that are developed by CRYSTAL brick providers.

Table 5-1: Abbreviations and acronyms.

ASL	Adjustable Speed Limiter
BSW	Basic Software
e2e	End-To-End
ECU	Electronic Control Unit
DC	Design Component
MSC	Message Sequence Chart
MSD	Modal Sequence Diagram
RTE	Run-Time Environment
SW	Software
SWC	AUTOSAR Software Component
WCET	Worst Case Execution Time

Table 5-2: Tools, used within the use case, which are developed by CRYSTAL brick providers.

Tool	**Short Description**
Arctic Studio (SWC Builder, Extract Builder, BSW Builder, RTE Builder, etc.)	The Arctic Studio tool chain provides a complete embedded software development environment for automotive embedded software based on the open industry-leading standard AUTOSAR. The tool chain supports all stages in a development project and provides tools for different types of tasks; application development, embedded platform development and system integration.
ASD:Suite	The ASD:Suite is used to define and (automatically) verify discrete event models, and to (automatically) generate fully executable source code from these models. The models specify both structure and behaviour of services and of components that implement and use these services.
DTFSim	The Data Time Flow Simulator DTFSim is a discrete-event simulation environment. Its main application areas are design, simulation and performance analysis of distributed electronic control systems.
MoMuT	The MoMuT family of automated test case generation tools derives test cases from test models. The test case generation approach uses mutations and provides a test suite that achieves requested mutation coverage. Currently, UML and timed automata are available as front end languages; support for formalized requirements and for SCADE is under development.
Orca	Orca is an integrated development environment for modelling and analysing real-time applications. Orca supports the modelling of architecture and relevant requirements for

	the timing use case, as well as triggering state based timing analysis and optimization tools.
Rubus	Rubus is a methodology and concept for designing, perform pre-runtime and post-runtime analysis and synthesize a complete run-time system. The Rubus concept is to support a software engineer to create highly efficient and robust real-time systems using state of the art analysis algorithms, tools and methodologies.
SystemWeaver	SystemWeaver conforms to the ideas of Model Based Development where the actual system building blocks or components act as information carriers. The SystemWeaver platform is an important part, being the infrastructure used for maintaining consistency in design information, distribution of consistent design information, and integration of design processes.

6 References

[1] "AUTOSAR", [Online], Available: http://www.autosar.org

[2] "EAST-ADL", [Online], Available: http://www.east-adl.info/

7 Annex I: Detailed Descriptions of the Engineering Methods

The engineering methods are captured by the Excel file, which is inserted below.

Engineering Method: UC3.1_AUTOSARECUIntegration_BSW_1					
Purpose: The ECU integrator configures the BSW for the ECU in BSWBuilder					
Comments:					
Pre-Condition		**Engineering Activities (made of steps)**		**Post-Condition**	
An ECU Extract with communication matrix Definition of diagnostics parameters Hardware definition and requirements		1. Import Communication Matrix from EcuExtract to get initial communication stack configuration 2. Import ODX file to get initial diagnostics definition 3. Create BSW Template from EcuExtract data 4. Configure all needed BSW Modules based on the hardware requirements and selected hardware 5. Validate configuration for consistency and needs from Extract 6. Generate configuration and service components		A configured BSW in the ECU Configuration model Generated BSW configuration in C-code	
Notes:		Notes:		Notes:	
Artefacts Required as inputs of the Activities		**Artefacts used internally within the Activities (optional)**		**Artefacts Provided as outputs of the Activities**	
Name	Implementation Model	Name		Name	ECU Configuration Model
Generic Type: (Tool or language independend type)	Flat ECUExtract	Generic Type: (Tool or language independend type)		Generic Type: (Tool or language independend type)	BSW Configuration
Required Properties: (Information required in interactions between steps)	Data following Autosar meta-model 4.1.x	Required Properties: (Information required in interactions between steps)		Provided Properties: (Information provided in interactions between steps)	Data following the Autosar meta-model
Description & Interoperability Additional Constraints: Containing Commuincation matrix		Description & Interoperability Additional Constraints:		Description & Interoperability Additional Constraints:	
Name	Diagnostic definition			Name	BSW Configuration Code
Generic Type: (Tool or language independend type)	PID, DTC, FF definitions			Generic Type: (Tool or language independend type)	C-code
Required Properties: (Information required in interactions between steps)	Data following ODX format			Provided Properties: (Information provided in interactions between steps)	
Description & Interoperability Additional Constraints:				Description & Interoperability Additional Constraints:	
Name				Name	
Generic Type: (Tool or language independend type)				Generic Type: (Tool or language independend type)	
Required Properties: (Information required in interactions between steps)				Provided Properties: (Information provided in interactions between steps)	
Description & Interoperability Additional Constraints:				Description & Interoperability Additional Constraints:	

↑ Artefacts considered for Interoperability

↑ Artefacts considered for Interoperability

Engineering Method: UC3.1_AUTOSARECUIntegration_RTE_1					
Purpose: The ECU integrator configures an RTE for the ECU in RTEBuilder and BSWBuilder					
Comments:					
Pre-Condition		**Engineering Activities (made of steps)**		**Post-Condition**	
A complete flat ECU Extract A BSW Configuration		1. Map the runnables in the EcuExtract to Tasks in the OSConfiguration 2. Connect the service ports of the software components to the BSW modules 3. Validate the configuration for consistency 4. Generate the source code of the RTE 5. Compile and link source code		A configured RTE in the ECU Configuration model A generated RTE in source code. A binary of the ECU	
Notes:		Notes:		Notes:	
Artefacts Required as inputs of the Activities		**Artefacts used internally within the Activities (optional)**		**Artefacts Provided as outputs of the Activities**	
Name	Implementation Model	Name		Name	Ecu Configuration Model
Generic Type: (Tool or language independend type)	Flat ECUExtract	Generic Type: (Tool or language independend type)		Generic Type: (Tool or language independend type)	RTE Configuration
Required Properties: (Information required in interactions between steps)	Data following Autosar meta-model 4.1.x	Required Properties: (Information required in interactions between steps)		Provided Properties: (Information provided in interactions between steps)	Data folowing the Autosar meta-model 4.1.x
Description & Interoperability Additional Constraints:		Description & Interoperability Additional Constraints:		Description & Interoperability Additional Constraints:	
Name	ECU Configuration Model			Name	RTE code
Generic Type: (Tool or language independend type)	BSW module configuration			Generic Type: (Tool or language independend type)	C-code
Required Properties: (Information required in interactions between steps)	Data following Autosar meta-model 4.1.x			Provided Properties: (Information provided in interactions between steps)	
Description & Interoperability Additional Constraints:				Description & Interoperability Additional Constraints:	
Name	Service Configuration Model			Name	Binary
Generic Type: (Tool or language independend type)	BSW Service components			Generic Type: (Tool or language independend type)	Elf-file
Required Properties: (Information required in interactions between steps)	Data following Autosar meta-model 4.1.x			Provided Properties: (Information provided in interactions between steps)	
Description & Interoperability Additional Constraints:					

↑
Artefacts considered for Interoperability

↑
Artefacts considered for Interoperability

Engineering Method: UC3.1_TimingAnalysis_1					
Purpose: The System Designer wants to check if the system design meets its timing requirements					
Comments: This is preferably done at design level, before software is available.					
Pre-Condition		**Engineering Activities (made of steps)**		**Post-Condition**	
System model is available in SystemWeaver (includes topology, design components, signals and mapping of components to ECUs) Timing requirements have been formulated in SystemWeaver (includes timing chains, end-to-end latency requirements, and execution periods of design components) **Optional:** Alternative Architectures and safety/costs requirements/constraints (plus already existing preconditions).		1. The engineer presses "Timing Analysis" button in SystemWeaver client. 2. The timing analysis tool is launched. 3. The data needed to perform the timing analysis (i.e. artefacts in pre-condition) is transferred from SystemWeaver to the timing tool. 4. The engineer adds additional timing information in the timing tool e.g. execution times, simulation parameters. 5. The engineer uses the timing analysis tool to check if the timing requirements are met. 6. The added information and verification results are transferred back to SystemWeaver and stored in the system model.		Verification results and additional timing information available in SystemWeaver. **Optional:** Give optimized results regarding architecture design/costs.	
Notes:		Notes: The timing analysis tool could be RubusDesigner or DTFSim e.g. to be chosen in menu. An alternative approach is to keep the information also in the timing tool rather than only in SystemWeaver. **Optional (OFFIS' tool):** Can be employed in the optimization of the design architecture/costs as well as for timing analysis (exact timing analysis).		Notes:	
Artefacts Required as inputs of the Activities		**Artefacts used internally within the Activities (optional)**		**Artefacts Provided as outputs of the Activities**	
Name	System model	Name		Name	Verification result
Generic Type: (Tool or language independent type)	Architecture model, Design components, **Optional**: alternative architectures	Type:		Generic Type: (Tool or language independent type)	Analysis report, Simulation results, **Optional:** cost-optimized deployment of tasks
Required Properties: (Information required in interactions between steps)	Data following the EAST-ADL meta-model	Properties:		Provided Properties: (Information provided in interactions between steps)	Data following the EAST-ADL meta-model
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	
Name	Timing requirement	Name		Name	Timing information
Generic Type: (Tool or language independent type)	Formal timing requirement	Type:		Generic Type: (Tool or language independent type)	Formal timing requirement
Required Properties: (Information required in interactions between steps)	Data following the TADL2 meta-model (included in EAST-ADL)	Properties:		Provided Properties: (Information provided in interactions between steps)	Data following the TADL2 meta-model (included in EAST-ADL)
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	
Name	**Optional:** Cost requirements	Name		Name	**Optional:** Architecture/costs optimization
Generic Type: (Tool or language independent type)	Formal cost objectives	Type:		Generic Type: (Tool or language independent type)	Cost-optimized deployment
Required Properties: (Information required in interactions between steps)	Data following the EAST-ADL meta-model	Properties:		Provided Properties: (Information provided in interactions between steps)	Data following the EAST-ADL meta-model
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	
Name	**Optional:** Safety requirements	Name		Name	**Optional:** Safety information
Generic Type: (Tool or language independent type)	Formal safety requirements	Type:		Generic Type: (Tool or language independent type)	Deployment of tasks which satisfies the safety constraints, analysis report
Required Properties: (Information required in interactions between steps)	Data following the EAST-ADL meta-model	Properties:		Provided Properties: (Information provided in interactions between steps)	Data following the EAST-ADL meta-model
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	
↑ Artefacts considered for Interoperability				↑ Artefacts considered for Interoperability	

Engineering Method: UC3.1_TestCaseGeneration_1					
Purpose: Provide test cases from requirements model for component or system testing					
Comments: Focuses on functional testing					
Pre-Condition		**Engineering Activities (made of steps)**		**Post-Condition**	
1. Test model available 2. Prior test cases available (if they shall be taken into account on generation)		1. The engineer presses "Test Case Generation" button in SystemWeaver client. 2. Test case generation - Job interface is opened in MoMuT. 3. Reference to test model is prefilled 4. Selection of functionality to cover (by DC, test model subset, requirements ID list) 5. Decision if pre-existing test cases shall be taken into account (+ entry of reference to test cases) 6. Start of test case generation job 7. Test cases are generated and uploaded automatically		1. Test cases available in SystemWeaver. 2. Traceability to requirements established 3. Test coverage computed (model coverage) and reported	
Notes: Test model captures the specified behavior		Notes:		Notes:	
Artefacts Required as inputs of the Activities		**Artefacts used internally within the Activities (optional)**		**Artefacts Provided as outputs of the Activities**	
Name	SW behavioral model	Name		Name	Test cases
Generic Type: (Tool or language independend type)	Test model	Type:		Generic Type: (Tool or language independend type)	Test cases
Required Properties: (Information required in interactions between steps)	- ID - language - related DC(s) - relations from requirements to model elements	Properties:		Provided Properties: (Information provided in interactions between steps)	- ID - generation date and time - mapping from test cases to related requirements
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints: Test cases are abstract and need to be mapped to the test environment before execution	
Name		Name		Name	
Generic Type: (Tool or language independend type)		Type:		Generic Type: (Tool or language independend type)	
Required Properties: (Information required in interactions between steps)		Properties:		Provided Properties: (Information provided in interactions between steps)	
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	
Name		Name		Name	
Generic Type: (Tool or language independend type)		Type:		Generic Type: (Tool or language independend type)	
Required Properties: (Information required in interactions between steps)		Properties:		Provided Properties: (Information provided in interactions between steps)	
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	

↑
Artefacts considered for Interoperability

↑
Artefacts considered for Interoperability