Document Title	Methodology
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	068

Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.3.1

Document Change History

Date	Release	Changed by	Description
2017-12-08	4.3.1	AUTOSAR Release Management	• Minor corrections due to the modification of one requirement • Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Support for Data Exchange Points added • Minor corrections / clarifications / editorial changes; For details please refer to the ChangeDocumentation
2015-07-31	4.2.2	AUTOSAR Release Management	• Minor corrections and editorial changes
2014-10-31	4.2.1	AUTOSAR Release Management	• Support for Safety Extensions added • Support for Diagnostic Extract added • Support for Rapid Prototyping added • Support for Sender Receiver Serialization added
2014-03-31	4.1.3	AUTOSAR Release Management	• Alignment of the AUTOSAR Methodology to the System Description categories • Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Harmonization between ECU Configuration specification and AUTOSAR Methodology

2013-03-15	4.1.1	AUTOSAR Administration	• Allow the usage of requirement ID definition and tracing for specification items • Updated chapter 3.6 Ecu Integration and Configuration with support for A2L function • Added chapter 2.14 How to resolve Name Conflicts • Added sections 3.4.1.15 Define Consistency Needs and 3.4.2.17 Consistency Needs • Refine definition of Binding Times
2011-12-22	4.0.3	AUTOSAR Administration	• Simplification of use case diagrams by removing task use and introducing deliverables on use cases level (see Methodology Concept chapter) • Readability improvement by generation of tables with navigable links • Introduction of Variant Handling, E2E support, System Constraints Description • Refinement of Methodology Library, including the extension of deliverables in different use cases
2009-12-18	4.0.1	AUTOSAR Administration	• Changed tool platform for the SPEM model • Publish as pdf file instead of html • Used new table format for the model elements • Added SPEM diagrams • Methodology Concept chapter detailed • Memory Mapping use case added • Reworked and restructured use cases for more readability • Direct references to meta-model elements in figures and tables
2008-08-13	3.1.1	AUTOSAR Administration	• Legal Disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• Subchapter limitations of the current version enhanced • Document meta information extended • Small layout adaptations made

2007-01-24	2.1.15	AUTOSAR Administration	• Updated chapter 5 ECU-Design • Updated chapter 6.1 Relationship with Services • Legal disclaimer revised • Release Notes added • Advice for users revised • Revision Information added
2006-05-16	2.0	AUTOSAR Administration	• Initial release

Table of Contents

Bibliography

[1] Requirements on Methodology
AUTOSAR_RS_Methodology

[2] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

[3] Software Process Engineering Meta-Model Specification
http://www.omg.org/spec/SPEM/2.0/

[4] Integration of Franca IDL Software Component Descriptions
AUTOSAR_TR_FrancaIntegration

[5] Virtual Functional Bus
AUTOSAR_EXP_VFB

[6] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[7] System Template
AUTOSAR_TPS_SystemTemplate

[8] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[9] General Specification on Transformers
AUTOSAR_ASWS_TransformerGeneral

[10] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate

[11] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration

[12] Specification of Memory Mapping
AUTOSAR_SWS_MemoryMapping

[13] Specification of Compiler Abstraction
AUTOSAR_SWS_CompilerAbstraction

[14] Specification of Module E2E Transformer
AUTOSAR_SWS_E2ETransformer

[15] Diagnostic Extract Template
AUTOSAR_TPS_DiagnosticExtractTemplate

[16] Specification of RTE Software
AUTOSAR_SWS_RTE

[17] Specifications of Safety Extensions
AUTOSAR_TPS_SafetyExtensions

[18] ISO 26262 (Part 1-10) – Road vehicles – Functional Safety, First edition
http://www.iso.org

[19] Generic Structure Template
AUTOSAR_TPS_GenericStructureTemplate

[20] Interoperability of AUTOSAR Tools
AUTOSAR_TR_InteroperabilityOfAutosarTools

[21] Interoperability Of Autosar Tools Supplement
AUTOSAR_TR_InteroperabilityOfAutosarToolsSupplement

[22] Specification of ECU Resource Template
AUTOSAR_TPS_ECUResourceTemplate

1 Introduction

1.1 Objective

AUTOSAR requires a common technical approach for some steps of system development. This approach is called the `AUTOSAR methodology`. This document defines and describes this AUTOSAR methodology. It covers all major steps of the development of a system with AUTOSAR: from the definition of the `Virtual Functional Bus` to the generation of an `ECU executable`.

The requirements for the methodology are defined in the document [1].

1.2 Document Conventions

This document follows a list of document conventions, which are described in the following.

Technical terms of AUTOSAR are typeset in mono spaced font, e.g. `ECU`. As a general rule, plural forms of technical terms are created by adding "s" to the singular form, e.g. `ECU`s.

This document contains specification items in textual form that are distinguished from the rest of the text by a unique numerical ID, a headline, and the actual text starting after the ⌈ character and terminated by the ⌋ character. The conventions for requirements traceability follow [TPS_STDT_00080], see Standardization Template ([2]).

1.3 Scope

[TR_METH_01003] Scope of the AUTOSAR methodology ⌈The AUTOSAR methodology is not a complete process description, but rather aggregates the various elements of AUTOSAR and shows how they are brought together to develop a complete system. Sample aggregations are provided as Use Cases in Chapter 2. ⌋*(RS_METH_00006)*

[TR_METH_01004] Support for various stakeholders by the AUTOSAR methodology ⌈ The structure of the methodology was designed to help cover the needs of various AUTOSAR stakeholders:

- Organizations: Methodology is modeled in a modular format to allow organizations to tailor it and combine the Methodology within their own internal processes, while identifying points where they interact with other organizations.
- Engineers: Methodology is scoped to allow engineers of various roles quickly find AUTOSAR information that is relevant to their specific needs.

- Tool Vendors: Methodology provides a common language to share among all AUTOSAR members and a common expectation of what capabilities tools should support.

⌋*(RS_METH_00018, RS_METH_00056, RS_METH_00009)*

[TR_METH_01005] Restrictions of AUTOSAR methodology ⌈ Furthermore, the methodology does not prescribe a precise order in which activities should be carried out. The methodology is a mere work-product flow: it defines the dependencies of activities on work-products. This means that when the information specified in the methodology is available, an activity can be carried out to produce the output work-products. The set of activities is described in Chapter 3.

This restriction implies that the AUTOSAR methodology does not define an overall time-line and does not define how and when iterations are carried out. For example during system and design, the same activity (namely configuring the system) will be carried out repeatedly with various levels of precision. There will be a first "rough" configuration and a final "precise" configuration which might depend on the feedback from the actual configuration or even implementation of `ECU`s. How and when such refinement steps are to be carried out is NOT defined in the methodology. ⌋*(RS_METH_00047)*

1.4 Overview

[TR_METH_01000] Domains of the AUTOSAR methodology ⌈ The AUTOSAR methodology is structured into several domains of development:

- `Virtual Functional Bus`
- `System`
- `Software Component`
- `Basic Software`
- `ECU`

These domains are depicted in the methodology overview workflow Figure 2.9. ⌋*(RS_METH_00018, RS_METH_00032)*

[TR_METH_01001] AUTOSAR methodology assets ⌈ For each domain, relevant `Work Product`, `Task`, `Role`, and `Tool` elements are defined (see chapter 3). In addition, there are elements that are common for all domains (see 3.1). ⌋*(RS_METH_00025, RS_METH_00028, RS_METH_00009)*

[TR_METH_01002] AUTOSAR methodology use cases ⌈ Use cases (see chapter 2) show how these standard reusable elements are applied to support real-world development. The Overall View (see chapter 2.1) provides an end to end view on the typical use cases of all domains. ⌋*(RS_METH_00018, RS_METH_00056, RS_METH_00009)*

1.5 Methodology Concepts

[TR_METH_01006] General AUTOSAR methodology concepts ⌈ The AUTOSAR methodology defines activities[1] performed by roles that create work products as general reusable method patterns. The reusable method pattern elements are described in the methodology library elements section (cf. section 1.5.1). The methodology also describes sample process patterns of typical use cases considered for the creation of AUTOSAR work products. The patterns use process elements that are described in the use case elements section (cf. section 1.5.2).

The definitions and the figures are made according to the Software Process Engineering Meta-Model Specification [3]. The symbols are taken from the Enterprise Architect modeling tool. ⌋*(RS_METH_00018, RS_METH_00021, RS_METH_00047, RS_METH_00048, RS_METH_00025, RS_METH_00061, RS_METH_00028, RS_METH_00056)*

1.5.1 Methodology Library Elements

[TR_METH_01007] `Methodology Library` ⌈ The `Methodology Library` (see chapter 3) defines the `Methodology Library Elements` of every method pattern such as `Roles`, `Tasks`, and `Work Product Definitions`. ⌋*(RS_METH_00018, RS_METH_00021, RS_METH_00025, RS_METH_00028)*

[TR_METH_01008] `Methodology Library Element` ⌈ A `Methodology Library Element` contains a description of the element to define its purpose in the methodology and thus provides the basic contents of the AUTOSAR methodology. The `Methodology Library Elements` are used for the description of the related development processes. These `Methodology Library Elements` can been seen as a standard. ⌋*(RS_METH_00017, RS_METH_00043, RS_METH_00050, RS_METH_00064)*

[TR_METH_01009] Relation of `Methodology Library` and `Methodology Library Element` to the SPEM meta model ⌈ The `Methodology Library` and the `Methodology Library Elements` correspond to the `Method Content` and `Method Content Elements` in the SPEM meta model [3]. ⌋*(RS_METH_00009)*

[TR_METH_01010] Overview of `Methodology Library Elements` ⌈`Methodology Library Elements` comprise:

- `Task Definition` (section 1.5.1.1)
- `Work Product Definition` (section 1.5.1.2)
- `Role Definition` (section 1.5.1.3)

[1] The RS_Methodology document uses the term “Activity” when addressing process elements in general. In this document the atomic process elements are called “Tasks”, whereas an “Activity” is used to organize tasks and to define processes.

- `Tool Definition` (section 1.5.1.4)
- `Guidance` (section 1.5.1.5)[2]

⌋*(RS_METH_00021, RS_METH_00025, RS_METH_00027, RS_METH_00042, RS_METH_00028)*

The element symbols are shown in Figure 1.1.

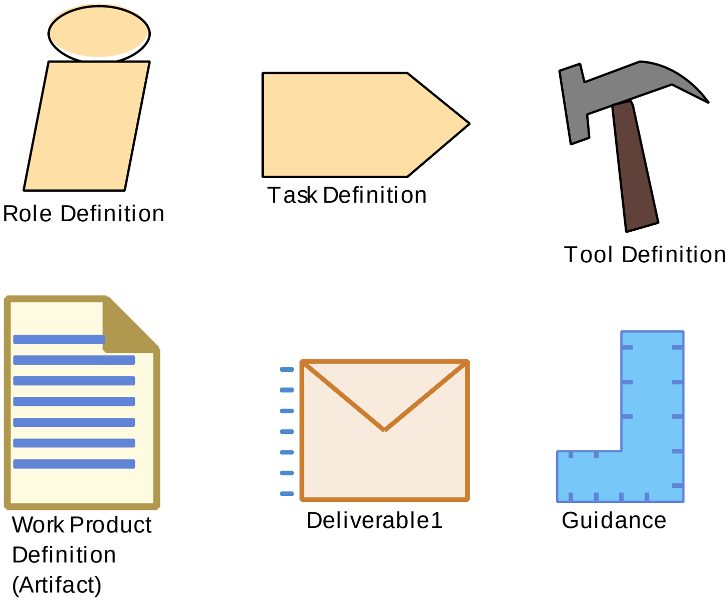

Figure 1.1: Symbols of AUTOSAR Methodology Library Elements

[TR_METH_01028] Usage of tables ⌈ Beside the graphical visualization of the different SPEM diagrams, tables are used to specify and describe the model elements in detail. ⌋*(RS_METH_00050, RS_METH_00064)*

[TR_METH_01113] Usage of hyperlinks ⌈ Beside the conventional references to chapters, figures and sections the AUTOSAR methodology document utilizes hyperlinks to the used SPEM elements. These hyperlinks are used across the text and within the tables. Using the hyperlinks the reader can quickly navigate to the related elements such as `Tasks`, `Activities`, `Roles`, `Work Products` and `Tools`. ⌋*(RS_METH_00067)*

1.5.1.1 Task Definition

[TR_METH_01011] `Task Definition` ⌈ According to the SPEM meta model, a `Task Definition` is an assignable unit of work that is being performed by specific `Roles`. The duration of a task is generally a few hours to a few days. Tasks usually generate one or more work products. Each `Task` is associated to input and output

[2]The Guidance is currently not used in the AUTOSAR Methodology. It may be used in future documents.

Work Products. Inputs are differentiated in *mandatory* and *optional* inputs. A Task is used as one element among others to define a Process. ⌋*(RS_METH_00021)*

[TR_METH_01012] Task semantics ⌈ A Task has a clear purpose in which the performing roles achieve a well defined goal. It provides complete step-by-step explanations of doing all the work that needs to be done to achieve this goal. This description is completely independent of when in a process lifecycle the work would actually be done. It does not describe when what work is being done, but describes all the work that gets done. ⌋*(RS_METH_00021, RS_METH_00056)*

[TR_METH_01013] Task usage ⌈ When a Task will be used in a development process, it provides the information which pieces of the Task will actually be performed at any particular point in time. This assumes that the Task will be performed in the process over and over again, but each time with a slightly different emphasis on different steps or aspects of the task description [3].

For the AUTOSAR Methodology, a Task is a reusable element that is used across multiple methodology use cases. A Task is associated to at least one performing Role and may have several additional performers. Tasks use Tools to achieve their outputs. Optional performers and optional input and outputs to the task are described by the relationship's multiplicity. ⌋*(RS_METH_00021, RS_METH_00042)*

An overview of the Task as it is used in this document is given in Figure 1.2.

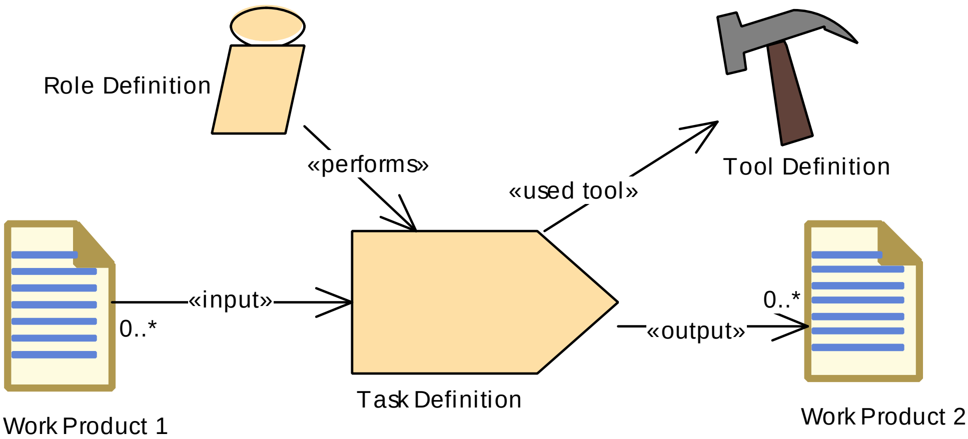

Figure 1.2: Task Definition Overview

1.5.1.1.1 Task Definition Tables

Task Definition	**Task**		
Package	Location in the MetaModel package		
Brief Description	Short description		
Description	Detailed description		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	**Related Element**	**Mul.**	**Note**
Performed by	Which Roles perform the Task	Opt or not	Description of the specific role needed
Consumes	What is consumed by the task	Mult	Explanation on why this Element is needed.
Produces	What is produced by the task	Mult	Explanation on why this Element is needed.
In/out	What is produced and consumed by the task	Mult	Explanation on why this Element is needed.
Used tool	Tool used for that Task	Mult	

Table 1.1: Task

1.5.1.2 Work Product Definition

[TR_METH_01014] `Work Product Definition` ⌈ According to the SPEM meta model, a `Work Product Definition` is used, modified, and produced by `Tasks` (i.e. a task input and output). `Work Products` are in most cases tangible work products consumed, produced, or modified by `Tasks`. They may serve as a basis for defining reusable assets. A `Work Product` can be related to other work products by a kind of nesting relationship, but work products shall not have circular references with other work products. ⌋*(RS_METH_00046, RS_METH_00047, RS_METH_00025, RS_METH_00052, RS_METH_00061, RS_METH_00054)*

[TR_METH_01015] Relationship between `Roles` and `Work Products` ⌈ `Roles` use `Work Products` to perform `Tasks` and produce `Work Products` in the course of performing the `Tasks`. `Work Products` are in the responsibility of the associated `Roles`, thereby also defining a set of skills the performing `Role` should have. Even though one `Role` might own a specific type of `Work Product`, other `Roles` can still use the `Work Product` for their work, and update them [3]. ⌋*(RS_METH_00052, RS_METH_00061)*

A `Work Product` can be of type `Artifact` or `Deliverable`:

- **[TR_METH_01017] `Artifact` Definition** ⌈ `Artifact`: A tangible `Work Product` that is consumed, produced, or modified by one or more `Tasks`. `Artifacts` may be composed of other `Artifacts` and may serve as a basis for defining reusable assets [3]. ⌋*(RS_METH_00052, RS_METH_00061, RS_METH_00054)*

  [TR_METH_01018] Kinds of `Artifacts` ⌈ For the AUTOSAR Methodology, typical kinds of artifacts are:

  - `AUTOSAR XML`
  - `Source Code`
  - `Object Code`

– Executable

– Text

For more details see chapter 3.1.1. ⌋(*RS_METH_00015, RS_METH_00057*)

[TR_METH_01019] Properties of Artifacts ⌈ At a high level, an artifact is represented as a single conceptual file. As a rule of thumb, the AUTOSAR Methodology will distinguish artifacts that have most of the following properties:

– Separate versioning is needed

– A dedicated life cycle has to be cared for

– Different exchange requirements need to be fulfilled

– Change in responsible roles

– Change in multiplicities

– Change in physical representation or format

– One of the products may be a separate deliverable to another party

– Separation of standardized from non-standardized parts

⌋(*RS_METH_00017, RS_METH_00016*)

[TR_METH_01020] Relationship between Artifacts and meta-model elements ⌈ To express a relationship between artifacts of the methodology model and any AUTOSAR meta-model element, a relationship with the stereotype «atpUseMetaModelElement» is used to express this "dependency". For AUTOSAR meta-model elements that are not directly related to methodology elements, there is usually an indirect relationship via a related meta-model element. The methodology can thus focus on the main elements of the meta-model. ⌋(*RS_METH_00051*)

• **[TR_METH_01021] Deliverable Definition** ⌈ Deliverable: Used to predefine typical or recommended content in the form of Work Products that would be packaged for delivery. Deliverables are used to represent an output from a process that has value, material or otherwise, to a client, customer, or other stakeholder. ⌋(*RS_METH_00025, RS_METH_00018, RS_METH_00054*)

[TR_METH_01022] Aggregation of Work Products ⌈ A Deliverable is a Work Product that aggregates other Work Products. The Method Content maintains pre-configured potential Deliverables [3]. For the AUTOSAR Methodology, the aggregation relationship is used to indicate which Work Products are contained in a deliverable. ⌋(*RS_METH_00025, RS_METH_00018, RS_METH_00054*)

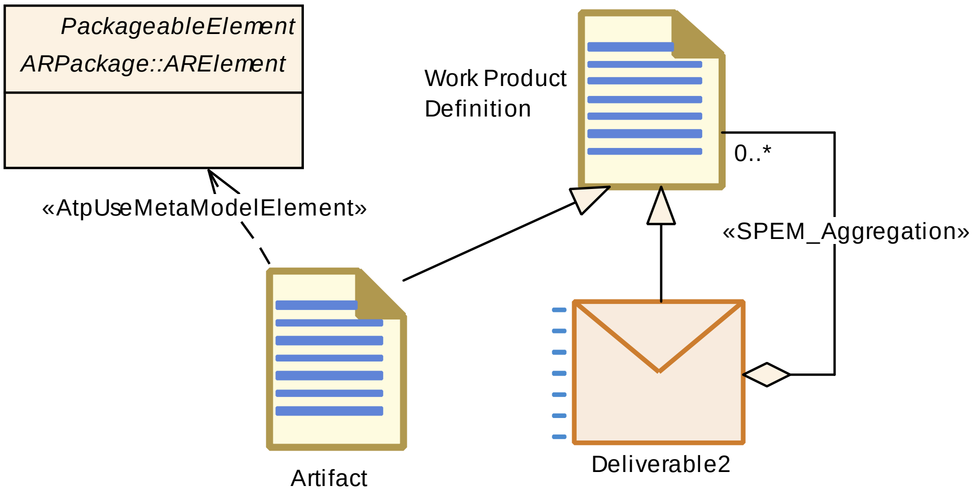

Figure 1.3: Work Product Definition Overview

1.5.1.2.1 Work Product Kind Tables

Category (Work Product Kind)	**Work Product Kind**
Package	Location in the MetaModel package
Brief Description	Short Description
Description	Detailed Description

Table 1.2: Work Product Kind

1.5.1.2.2 Artifact Definition Tables

Artifact	**Artifact name**		
Package	Location in the MetaModel package		
Brief Description	Short Description.		
Description	Detailed Description		
Kind	Work Product Kind		
Extended by	Artifacts which extend this Artifact		
Extends	Artifacts which are extended by this Artifact		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	To which Deliverable is it aggregated	Mult	Description of the context of the Aggregation.
In/out	Which task is producing and consuming the Work Product	Mult	Description of the context of the Work Product production and consumption.
Produced by	Which task is producing the Work Product	Mult	Description of the context of the Work Product production.

Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Which task is consuming the Work Product	Mult	Description of the context of the Work Product consumption.
Use meta model element	MetamodelElement Relationship	Mult	

Table 1.3: Artifact name

1.5.1.2.3 Deliverable Definition Tables

It is the same structure of table as the `Work Product`, only the Aggregation is not the same as it can aggregate other `Work Products` or `Deliverables`.

Deliverable	**Deliverable**		
Package	Location in the MetaModel package		
Brief Description	Short Description.		
Description	Detailed Description		
Kind	Work Product Kind		
Extended by	Deliverables which extend this Deliverable		
Extends	Deliverables which are extended by this Deliverable		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Which Work Products are aggregated to it	Mult	
Aggregated by	To which Deliverable is it aggregated	Mult	Description of the context of the Aggregation.
In/out	Which task is producing and consuming the deliverable	Mult	Description of the Context of production and consumption.
Produced by	Which task is producing the deliverable	Mult	Description of the context of the production.
Consumed by	Which task is consuming the deliverable	Mult	Description of the context of the consumption.
Use meta model element	MetamodelElement Relationship	Mult	

Table 1.4: Deliverable

1.5.1.3 Role Definition

[TR_METH_01023] **`Role Definition`** ⌈ According to the SPEM meta model, `Role Definitions` define responsibilities of an individual or a set of individuals and thereby define a set of related skills, competencies, and qualifications needed to perform a

Task. A Role can be filled by one person or multiple people, one person may fill several Roles. Each Role performs Tasks. ⌋*(RS_METH_00028)*

[TR_METH_01024] Role assignment ⌈ Roles are not individuals or resources. Individual members of the development organization will wear different hats, or perform different Roles. The mapping from individual to Role, usually performed by the project manager when planning and staffing a project, allows different individuals to act as several different Roles, and for a Role to be taken by several individuals [3].

In the AUTOSAR Methodology, a Role also assigns the responsibility of a Task and defines *optional* performers. Performers that are responsible for e.g. a Task have a multiplicity of 1 for the relationship to the Task, optional performers have optional multiplicity assigned. Role Definitions are usually generic and still provide sufficient level of detail for managers to organize a team. Examples of Roles are "System Engineer", "Safety Engineer", or "Software Developer". ⌋*(RS_METH_00028, RS_METH_00056)*

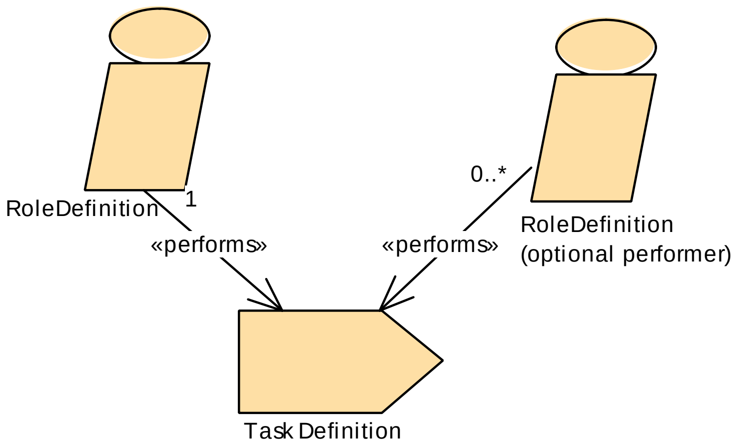

Figure 1.4: Role Definition Overview

1.5.1.3.1 Role Definition Tables

Role	**Role**		
Package	Meta-model Package Name		
Brief Description	Short Description.		
Description	Detailed Description.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performs	In which task the performer is acting	Mult	

Table 1.5: Role

1.5.1.4 Tool Definition

[TR_METH_01025] **`Tool Definition`** ⌈ According to the SPEM meta model, `Tool Definitions` can be used to specify a tool's participation in a `Task`. A `Tool Definition` describes the capabilities of a CASE tool, general purpose tool, or any other automation unit that supports the associated `Roles` in performing the work defined by a `Task`. A `Tool` can identify a resource as *useful*, *recommended*, or *necessary* for a task's completion. A `Tool` can also be used to manage one or more `Work Products` [3].

The AUTOSAR Methodology uses the `Tool Definition` to describe AUTOSAR specific (e.g. Software Component Contract Generator) and other general `Tools` (e.g. Compilers). The relationship of a `Tool` to a `Task` shows which `Tools` a `Role` will need to perform the `Task`. ⌋*(RS_METH_00066, RS_METH_00042)*

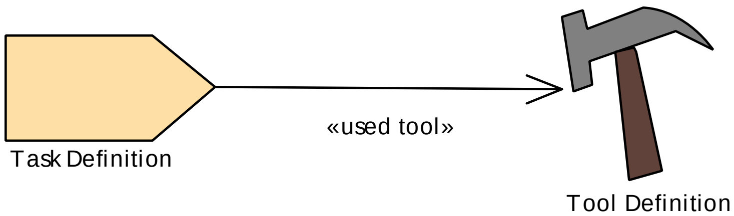

Figure 1.5: Tool Definition Overview

1.5.1.4.1 Tool Definition Tables

Tool	**Tool**		
Package	Meta-model Package name		
Brief Description	Short Description		
Description	Detailed Description		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Used	Task where the tool is used	Mult	

Table 1.6: Tool

1.5.1.5 Guidance

[TR_METH_01026] **`Guidance` definition** ⌈ According to the SPEM meta model, a `Guidance` provides additional information related to e.g. `Roles`, `Work Products`, and `Tasks`. A `Guidance` is classified to indicate a specific type for which perhaps a specific structure and type of content is assumed [3]. ⌋*(RS_METH_00027)*

[TR_METH_01027] **`Guidance` kinds** ⌈ A `Guidance` can be a

- Supporting Material: Supporting Material is a catch-all for other types of guidance not specifically defined elsewhere. It can be related to all kinds of Content Elements, i.e., including other guidance elements. The AUTOSAR Methodology uses the Supporting Material Guidance type to define title pages, change histories, disclaimers etc.
- Tool Mentor: A Tool Mentor shows how to use a specific Tool to accomplish some piece of work either in the context of or independent from a Task or Activity. In the context of the AUTOSAR Methodology, a Tool Mentor is used in the same way as the Tool element.
- White Paper: White Papers are concept guidances that have been externally reviewed or published and can be read and understood in isolation from other Method Content. AUTOSAR documents are examples of White Papers.

Other Guidances such as Checklists, Concepts, Estimates, Guidelines, Practices, Reports, Reusable Assets, Roadmaps, or Templates as defined in [3] are not used within the AUTOSAR Methodology. ⌋*(RS_METH_00027)*

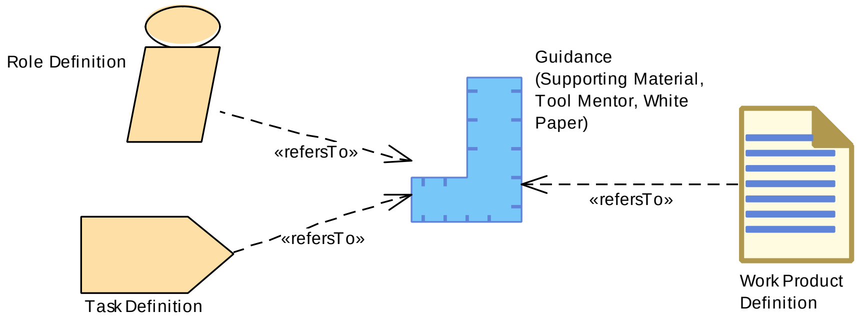

Figure 1.6: Guidance Overview

1.5.2 Use Case Specifications

This section explains how the use cases in chapter 2 are specified. The first two subsections introduce the main constituents of the use cases. Afterwards, it is explained how these elements together with the Methodology Library elements are used for describing the use cases.

[TR_METH_01031] Adaptability of the AUTOSAR methodology ⌈ The main focus of this section is merely to provide a use case process flow that can be supported by an AUTOSAR tool chain rather than to define a complete process description. One reason for doing this is that the AUTOSAR methodology should be adaptable to development processes of different organizations. ⌋*(RS_METH_00056)*

[TR_METH_01032] Use case elements ⌈ This section describes the main elements to build a use case, which are given by the Capability Pattern and the Activity. The element symbols are shown in Figure 1.7. Roles, Work Products, Deliverables and Tasks are used directly to describe the details of an Activity. The SPEM meta model additionally defines the Role Use , the Work Product Use and the Task Use elements, which are not used in the AUTOSAR methodology. Whereas these are important elements when applying SPEM in an organization, the AUTOSAR methodology does not necessarily need these elements since no instantiation of the Enterprise Architect model is intended. ⌋()

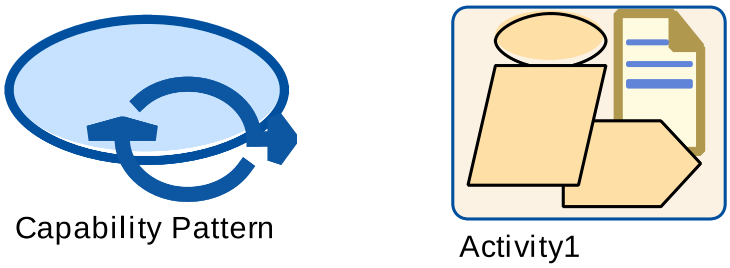

Figure 1.7: Symbols of AUTOSAR Use Case Elements

1.5.2.1 Activity

[TR_METH_01033] Definition of Activities ⌈ In the SPEM meta model, an Activity is the main building block to define a process. An Activity is usually a defined task or work to be done that is commonly executed in one sequence. ⌋ *(RS_METH_00021)*

[TR_METH_01034] Composition of Activities ⌈ Activities can include other Activities and thereby often decompose a flow of work and show which Activity precedes other Activities [3]. At the lowest level, Activities are collections of work breakdown elements which in AUTOSAR methodology are Tasks, Roles, and Work Products. ⌋*(RS_METH_00048, RS_METH_00046, RS_METH_00047, RS_METH_00066)*

[TR_METH_01035] Definition of Processes ⌈ A Process is a special Activity in the SPEM meta model that describes a typical structure of development projects or parts of them. A Process focuses on the lifecycle and the sequencing of work in breakdown structures. Processes contain sequences of Task and Activities and thereby express a lifecycle of the product under development. Processes also define how to get from one milestone to the next by defining sequences of work, operations, or events [3]. ⌋*(RS_METH_00056)*

1.5.2.2 Capability Pattern

The methodology library elements (cf. Section 1.5.1) are referenced in order to describe together with activities the so-called Capability Patterns.

[TR_METH_01029] `Capability Patterns` definition ⌈ A Capability Pattern[3] is a process pattern that contains a reusable set of activities. ⌋ *(RS_METH_00018)*

[TR_METH_01030] Composition of `Capability Patterns` ⌈ Capability Patterns can be assembled to larger Capability Patterns that describe development processes or parts of a development process including typical use cases. ⌋ *(RS_METH_00018, RS_METH_00056)*

1.5.2.3 Description of Use Cases

For the AUTOSAR Methodology, the main Use Cases are described with 3 types of diagrams.

[TR_METH_01036] Description of overall `Use Cases` ⌈ In the first diagram, one Capability Pattern describes the overall Use Case, composing a set of Activities and their main outputs (Deliverables). In these diagrams, the predecessor relationship can be used in order to define a sequence of the Activities. However, the predecessor relationship can be skipped and Deliverables can be extended by other Deliverables (see Figure 1.8). ⌋ *()*

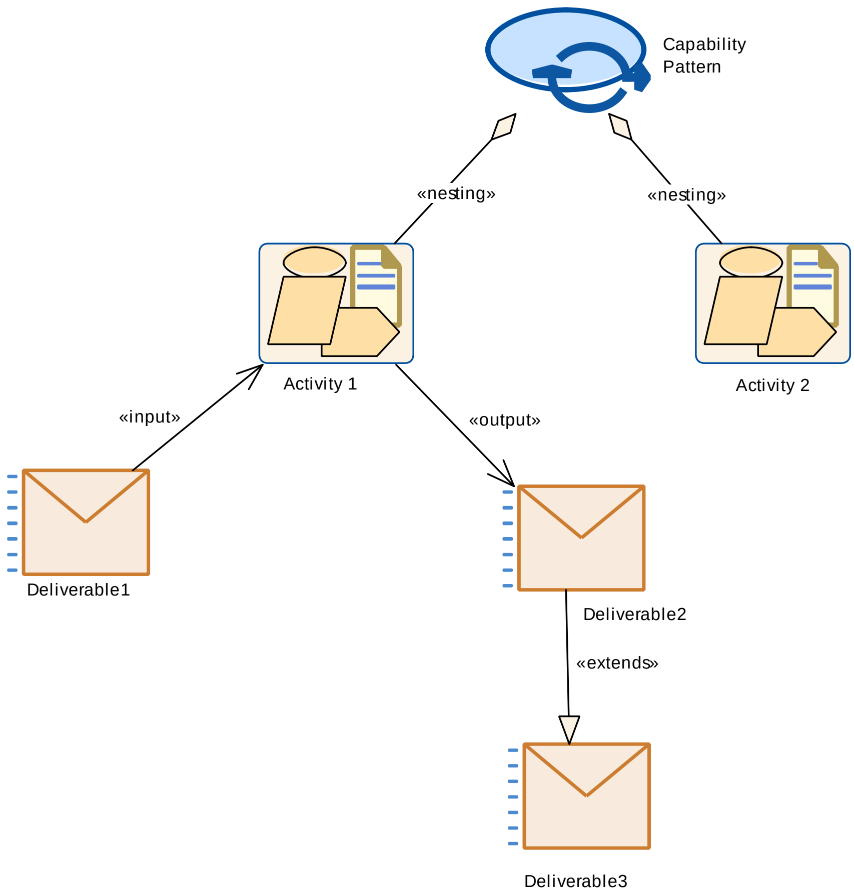

Figure 1.8: Activity Overview

[3] In Enterprise Architect a SPEM “Capability Pattern” is called “Process Pattern”.

The diagram is followed by its corresponding table as detailed hereunder:

Process Pattern	**Capability Pattern**		
Package	Meta-model Package name		
Brief Description	Short Description		
Description	Detailed Description.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Activity nested to the Capability Pattern or to another Activity	Mult	Context explanation
Consumes	Deliverable consumed by the Activity	Mult	Why this Activity needs to consume this Deliverable
Produces	Deliverable produced by the Activity	Mult	Why this Activity is producing this Deliverable

Table 1.7: Capability Pattern

[TR_METH_01037] Precise description of `Use Cases` ⌈ The second type of diagram are `Activities` and `Task Definition` diagrams which precise the main `Tasks` and `Work Products` used for the `Use Cases` but are not as detailed as in the Methodology Library (see Figure 1.9). The task usage in these diagrams can be expressed by the role and in the note of the aggregation. This information will be also visible in the generated table. The Work Products consumed or produced in the use cases will be not integrated in the table for readability. ⌋*()*

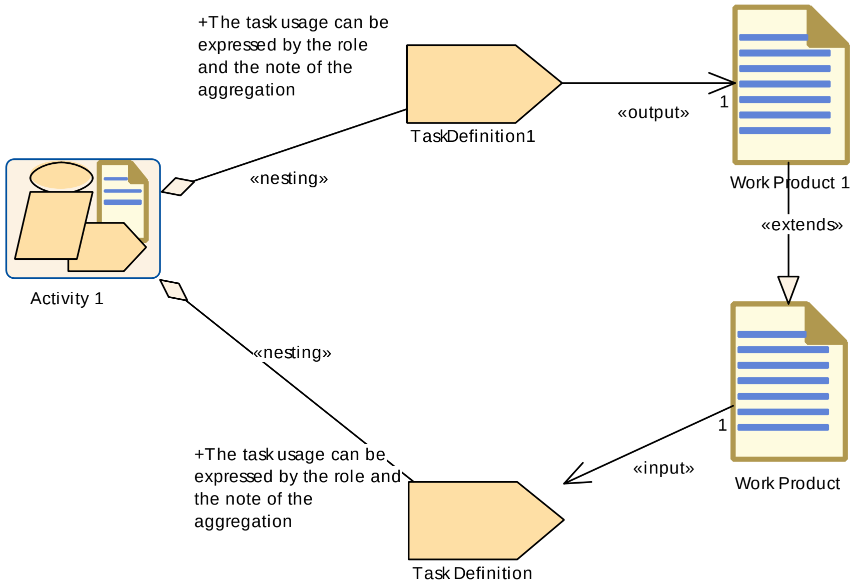

Figure 1.9: Activity and Tasks Overview

The diagram is followed by its corresponding table as detailed hereunder:

Activity	**Activity**		
Package	Meta-model Package Name		
Brief Description	Short Description		
Description	Detailed Description		
Extended by	Activities which extend this Activity		
Extends	Activities which are extended by this Activity		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Nested task definition	Mult	Task usage description if needed
Consumes	What is consumed by the Activity	Mult	Explanation on why this Element is needed.
Produces	What is produced by the activity	Mult	Explanation on why this Element is needed.
In/out	What is produced and consumed by the activity	Mult	Explanation on why this Element is needed.
Predecessor	Predecessor of the activity	Mult	Explanation on why the Predecessor is needed.

Table 1.8: Activity

[TR_METH_01038] Detailed description of the work flow ⌈ The third type of diagram contains the `Tasks` and `Work Products` used by an `Activity` in order to show the detailed work flow but not the structure of `Activities` as seen in Section 1.5.1.1. As an example take Figure 2.16. The table generation is not done for this type of diagram.
⌋*()*

1.6 Requirements Traceability

This section states the response of this specification to the corresponding requirements document[1].

Requirement	**Description**	**Satisfied by**

[RS_METH_00002]	No description	[TR_METH_01044] [TR_METH_01047] [TR_METH_01048] [TR_METH_01050] [TR_METH_01051] [TR_METH_01052] [TR_METH_01053] [TR_METH_01054] [TR_METH_01055] [TR_METH_01056] [TR_METH_01057] [TR_METH_01058] [TR_METH_01059] [TR_METH_01060] [TR_METH_01061] [TR_METH_01065] [TR_METH_01066] [TR_METH_01067] [TR_METH_01068] [TR_METH_01071] [TR_METH_01075] [TR_METH_01076] [TR_METH_01077] [TR_METH_01078] [TR_METH_01079] [TR_METH_01080] [TR_METH_01081] [TR_METH_01082] [TR_METH_01087] [TR_METH_01088] [TR_METH_01090] [TR_METH_01091] [TR_METH_01110] [TR_METH_01112] [TR_METH_01125] [TR_METH_01126] [TR_METH_01127] [TR_METH_01132] [TR_METH_01133] [TR_METH_02000] [TR_METH_02001] [TR_METH_02002] [TR_METH_02005] [TR_METH_03000] [TR_METH_03005] [TR_METH_03006] [TR_METH_03007]

[RS_METH_00003]	No description	[TR_METH_01083] [TR_METH_01084] [TR_METH_01085] [TR_METH_01087] [TR_METH_01088] [TR_METH_01089] [TR_METH_01090] [TR_METH_01091] [TR_METH_01092] [TR_METH_01111] [TR_METH_01112] [TR_METH_01114] [TR_METH_01115] [TR_METH_01117] [TR_METH_02002] [TR_METH_02005] [TR_METH_03000] [TR_METH_03010]
[RS_METH_00004]	No description	[TR_METH_01083] [TR_METH_01086] [TR_METH_01087] [TR_METH_01088] [TR_METH_01089] [TR_METH_01090] [TR_METH_01091] [TR_METH_01092] [TR_METH_01095] [TR_METH_01098] [TR_METH_01103] [TR_METH_01104] [TR_METH_01112] [TR_METH_01114] [TR_METH_01115] [TR_METH_01116] [TR_METH_01117] [TR_METH_01151] [TR_METH_02005] [TR_METH_03000]

[RS_METH_00005]	No description	[TR_METH_01046] [TR_METH_01047] [TR_METH_01048] [TR_METH_01053] [TR_METH_01065] [TR_METH_01066] [TR_METH_01067] [TR_METH_01068] [TR_METH_01070] [TR_METH_01071] [TR_METH_01075] [TR_METH_01076] [TR_METH_01077] [TR_METH_01078] [TR_METH_01079] [TR_METH_01080] [TR_METH_01081] [TR_METH_01082] [TR_METH_01087] [TR_METH_01088] [TR_METH_01090] [TR_METH_01091] [TR_METH_01092] [TR_METH_01109] [TR_METH_01112] [TR_METH_01114] [TR_METH_01125] [TR_METH_01126] [TR_METH_01127] [TR_METH_01130] [TR_METH_01153] [TR_METH_01154] [TR_METH_01155] [TR_METH_01156] [TR_METH_01157] [TR_METH_02003] [TR_METH_02006] [TR_METH_02015] [TR_METH_02016] [TR_METH_02017] [TR_METH_02018] [TR_METH_03000] [TR_METH_03008]

[RS_METH_00006]	The methodology shall explain how to build an AUTOSAR system	[TR_METH_01003] [TR_METH_01039] [TR_METH_01044] [TR_METH_01045] [TR_METH_01046] [TR_METH_01047] [TR_METH_01048] [TR_METH_01049] [TR_METH_01061] [TR_METH_01085] [TR_METH_01087] [TR_METH_01092] [TR_METH_01093] [TR_METH_01109] [TR_METH_01110] [TR_METH_01111] [TR_METH_01112] [TR_METH_01114] [TR_METH_01134] [TR_METH_01135] [TR_METH_03002] [TR_METH_03003] [TR_METH_03004]
[RS_METH_00009]	No description	[TR_METH_01001] [TR_METH_01002] [TR_METH_01004] [TR_METH_01009]
[RS_METH_00010]	No description	[TR_METH_01121]
[RS_METH_00015]	The methodology shall be independent of programming languages	[TR_METH_01018]
[RS_METH_00016]	The methodology shall support building a system of both AUTOSAR and Non-AUTOSAR ECUs	[TR_METH_01019] [TR_METH_01128] [TR_METH_01129]
[RS_METH_00017]	No description	[TR_METH_01008] [TR_METH_01019]
[RS_METH_00018]	The methodology shall be modular	[TR_METH_01000] [TR_METH_01002] [TR_METH_01004] [TR_METH_01006] [TR_METH_01007] [TR_METH_01021] [TR_METH_01022] [TR_METH_01029] [TR_METH_01030] [TR_METH_01084] [TR_METH_01110]
[RS_METH_00020]	The methodology shall support round-trip engineering	[TR_METH_01071] [TR_METH_01089] [TR_METH_02004]

[RS_METH_00021]	No description	[TR_METH_01006] [TR_METH_01007] [TR_METH_01010] [TR_METH_01011] [TR_METH_01012] [TR_METH_01013] [TR_METH_01033]
[RS_METH_00025]	No description	[TR_METH_01001] [TR_METH_01006] [TR_METH_01007] [TR_METH_01010] [TR_METH_01014] [TR_METH_01021] [TR_METH_01022]
[RS_METH_00027]	No description	[TR_METH_01010] [TR_METH_01026] [TR_METH_01027]
[RS_METH_00028]	No description	[TR_METH_01001] [TR_METH_01006] [TR_METH_01007] [TR_METH_01010] [TR_METH_01023] [TR_METH_01024]
[RS_METH_00032]	The methodology shall support different levels of abstractions	[TR_METH_01000] [TR_METH_01040]
[RS_METH_00033]	The methodology should support the VFB concept	[TR_METH_01039] [TR_METH_01045] [TR_METH_01054] [TR_METH_02000]
[RS_METH_00038]	No description	[TR_METH_01060] [TR_METH_01085] [TR_METH_01093] [TR_METH_02005] [TR_METH_03001]
[RS_METH_00041]	The methodology shall support top-down and bottom-up approaches	[TR_METH_01071]
[RS_METH_00042]	The methodology shall incorporate the usage of industry standard tools	[TR_METH_01010] [TR_METH_01013] [TR_METH_01025] [TR_METH_01093]
[RS_METH_00043]	No description	[TR_METH_01008]
[RS_METH_00046]	No description	[TR_METH_01014] [TR_METH_01034]
[RS_METH_00047]	No description	[TR_METH_01005] [TR_METH_01006] [TR_METH_01014] [TR_METH_01034]
[RS_METH_00048]	No description	[TR_METH_01006] [TR_METH_01034]
[RS_METH_00050]	No description	[TR_METH_01008] [TR_METH_01028]
[RS_METH_00051]	No description	[TR_METH_01020]

[RS_METH_00052]	No description	[TR_METH_01014] [TR_METH_01015] [TR_METH_01017]
[RS_METH_00054]	No description	[TR_METH_01014] [TR_METH_01017] [TR_METH_01021] [TR_METH_01022] [TR_METH_01122]
[RS_METH_00056]	The AUTOSAR methodology shall not be bound to a particular life-cycle model	[TR_METH_01002] [TR_METH_01004] [TR_METH_01006] [TR_METH_01012] [TR_METH_01024] [TR_METH_01030] [TR_METH_01031] [TR_METH_01035]
[RS_METH_00057]	No description	[TR_METH_01018] [TR_METH_01123]
[RS_METH_00061]	No description	[TR_METH_01006] [TR_METH_01014] [TR_METH_01015] [TR_METH_01017]
[RS_METH_00062]	The methodology shall support configuration of parameters with different binding time.	[TR_METH_01086] [TR_METH_01095] [TR_METH_01098] [TR_METH_01104] [TR_METH_01108] [TR_METH_01150] [TR_METH_01151]
[RS_METH_00064]	No description	[TR_METH_01008] [TR_METH_01028]
[RS_METH_00066]	The methodology shall allow activities that reference tools	[TR_METH_01025] [TR_METH_01034]
[RS_METH_00067]	No description	[TR_METH_01113]
[RS_METH_00069]	It shall be possible to add precise and human readable documentation to each work product	[TR_METH_01123] [TR_METH_01124]
[RS_METH_00074]	The methodology shall specify binding times	[TR_METH_00001] [TR_METH_00002] [TR_METH_00003] [TR_METH_02011] [TR_METH_02012] [TR_METH_02013] [TR_METH_02014] [TR_METH_02020]
[RS_METH_00075]	The methodology shall specify the tasks of resolving variant	[TR_METH_00001] [TR_METH_02016]
[RS_METH_00076]	The methodology shall specify a work product for values of variant selectors	[TR_METH_02016] [TR_METH_02017]

[RS_METH_00077]	The methodology shall support different views on the SW-C structure by OEMs and suppliers	[TR_METH_01049] [TR_METH_01076] [TR_METH_01079] [TR_METH_01080] [TR_METH_01081] [TR_METH_01082] [TR_METH_01125] [TR_METH_01126] [TR_METH_01127] [TR_METH_01130] [TR_METH_01131]
[RS_METH_00078]	The methodology shall explain the typical usage of different views on the system of the OEM	[TR_METH_01044] [TR_METH_01050] [TR_METH_01068]
[RS_METH_00079]	The methodology shall explain the typical usage of different views on the system of the supplier	[TR_METH_01068] [TR_METH_01079] [TR_METH_01080] [TR_METH_01081] [TR_METH_01082]
[RS_METH_00080]	The AUTOSAR methodology shall support the concept of implicit communication behavior	[TR_METH_01120]
[RS_METH_00081]	No description	[TR_METH_01144] [TR_METH_01145] [TR_METH_01146] [TR_METH_01147] [TR_METH_01148] [TR_METH_01149]
[RS_METH_00082]	No description	[TR_METH_01136] [TR_METH_01137] [TR_METH_01138] [TR_METH_01139] [TR_METH_01140] [TR_METH_01141] [TR_METH_01142] [TR_METH_01143]
[RS_METH_00083]	The AUTOSAR methodology shall explain the description and handling of Data Exchange Points	[TR_METH_01202] [TR_METH_01204] [TR_METH_01205]
[RS_METH_00084]	The AUTOSAR methodology shall relate templates to a distributed development process	[TR_METH_01202] [TR_METH_01204] [TR_METH_01205]

Some input requirements cannot (or not completely) be traced down to single specification items found in this document. They are satisfied by the AUTOSAR methodology in a general way together with other documents as listed in the following:

[TR_METH_01120] Definition of `Consistency Needs` ⌈ The AUTOSAR methodology supports the exchange of implicit communication behavior description. Chapters 3.4.1.14 and 3.4.2.15 depict the task and the artifact which allow to define the corresponding consistency needs. ⌋ *(RS_METH_00080)*

[TR_METH_01121] Building the AUTOSAR methodology document ⌈ All AUTOSAR methodology related model elements (see 1.5) are consumed by an internal AUTOSAR tool that automatically produces the corresponding text, tables, and diagrams. These artifacts are included into a document which is automatically transformed into the final PDF file. ⌋*(RS_METH_00010)*

[TR_METH_01122] Relations between AUTOSAR `Work Products` ⌈ `Work Products` (`Deliverables` and `Artifacts`) are designed in such a way that no circular references with other `Work Products` exist. ⌋*(RS_METH_00054)*

[TR_METH_01123] Traceability to external artifacts ⌈ Artifacts considered in the Methodology model include external artifacts like c-code, libraries, documentation and generated artifacts (see e.g. 3.5.2.22, 3.4.2.4). `General Non Autosar Artifact` is a generic representation of non AUTOSAR artifacts. It is aggregated by the `General Deliverable` and allows linking and tracing of non AUTOSAR artifacts within the AUTOSAR context. Furthermore, several specific artifacts represent non AUTOSAR elements or allow referring to them. The `A2L File` artifact is a representation of the measurement and calibration format that is defined by the ASAM and therefore out of scope of AUTOSAR. The description of the `Atomic Software Component Implementation` artifact explains how external artifacts can be referred from this ARXML artifact. ⌋*(RS_METH_00057, RS_METH_00069)*

[TR_METH_01124] Documentation of `Work Products` ⌈ In order to document design decisions or restrictions during the development process each `Work Product` can aggregate the corresponding documentation which is represented by the `General Documentation` artifact. The `General Documentation` artifact is added to `Work Products` by processing the task `Add General Documentation`. ⌋*(RS_METH_00069)*

2 Use Cases

In the following, the main use cases for building an AUTOSAR system are described. Chapter 2.1 gives an overall brief description of the main development steps. These steps are elaborated in detail in chapter 2.2 to chapter 2.7. In addition, chapter 2.8 to 2.17 decribe general topics of interest.

2.1 Overall View

2.1.1 Purpose

This chapter provides a rough outline of the design steps to build an AUTOSAR system. The main activities are depicted in Figure 2.8. The overall workflow including relevant work products is given in Figure 2.9. A brief description of these main steps is given below in section 2.1.2.2. For a detailed description please refer to the relevant chapters 2.2 to 2.7.

2.1.2 Description

2.1.2.1 Views on the System

During the development of an AUTOSAR system different views on the system can exist. This allows to refine the system step by step as well as to concentrate on the relevant parts during the development.

[TR_METH_01039] Virtual Functional Bus View ⌈ The development of an AUTOSAR System is based on the definition of the `Virtual Functional Bus` (VFB). The VFB is an abstract communication mechanism that allows software components to interact. This view is independent of any ECUs and networks used. Based on the VFB the system is designed. ⌋*(RS_METH_00006, RS_METH_00033)*

[TR_METH_01040] Support of different system views ⌈ The views on the system might further be restricted to e.g. the functionality only, or a subsystem. These views are described explicitly, whereas a mapping mechanism is used to express the relation between them. ⌋*(RS_METH_00032)*

In the following, three different views on the system are distinguished:

- **[TR_METH_01041] Abstract system** ⌈ The abstract system abstracts from the concrete software architecture and describes e.g. the functional view on the system. ⌋*()*
- **[TR_METH_01042] Overall technical system** ⌈ The overall technical system is organized from the software architecture perspective including a topology of ECUs. ⌋*()*

- **[TR_METH_01043] Subsystem** ⌈ The subsystem is a reduced part of the overall technical system and describes relevant aspects for a dedicated subsystem. ⌋*()*

2.1.2.2 Overall Workflow

The main activities in order to develop an AUTOSAR system are described briefly in the following. The first step focuses on the development of an abstract system, followed by the description of the VFB development and finally the activities for refining and developing the system further.

[TR_METH_01044] Development of a functional view on the system ⌈ The overall workflow starts with an optional activity. In this activity, the `Abstract System Description` is developed in advance, which represents the overall system from a functional or abstract view (functional architecture). On the one hand, this `Abstract System Description` might contain VFB-related parts. This information might serve as an input for the development of the VFB later and a mapping between those two views might be established. Please note that during this step the functionality including ports is mapped to software components. Therefore some ports used in the abstract view might not be used in the subsequent development, see Figure 2.1. On the other hand, the `Abstract System Description` might contain information regarding the topology and the mapping to ECUs. This is then the basis for the development of the concrete `System Description`. The development of the `Abstract System Description` is detailed in chapter 2.2. ⌋*(RS_METH_00006, RS_METH_00002, RS_METH_00078)*

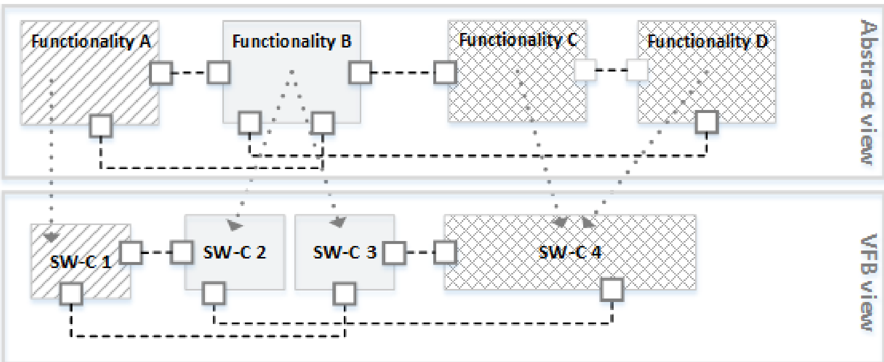

Figure 2.1: Abstract view on the system (top) and exemplary mapping to the SW-Cs of the VFB View (bottom)

[TR_METH_01045] Development of the `Overall VFB System` ⌈ In case of omitting the optional first step, the development directly starts with the definition of the `Overall VFB System`. The VFB is an abstraction of the communication between software components. It provides a dedicated view of all the software components the system contains, see Figure 2.2, independent of any ECUs and networks. See chapter 2.3 for more details. ⌋*(RS_METH_00006, RS_METH_00033)*

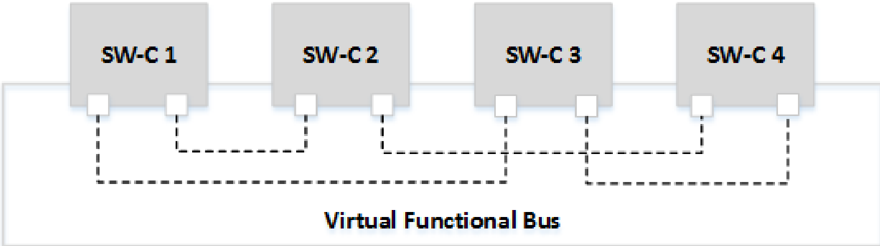

Figure 2.2: VFB View

[TR_METH_01046] Development of the system ⌈ The VFB is refined into a system by defining a topology of ECUs and networks and deploying software components to the ECUs, see Figure 2.3. Additionally, the communication matrices, which are required to interconnect the distributed features, are derived. As a part of the communication development, a custom transformation technology can be specified for transforming the data in case of inter-ECU communication. This transformer specification is the basis for the implementation of the corresponding basic software module. The development of the system can be achieved directly in one phase or in several phases. ⌋ *(RS_METH_00006, RS_METH_00005)*

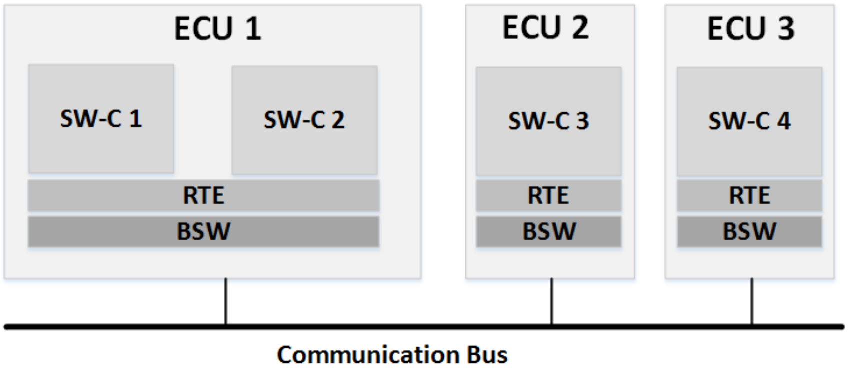

Figure 2.3: Scope of the system

[TR_METH_01047] Two phase development approach ⌈ The two phase approach is used when there is an organizational separation of responsibility, where the primary organization (usually OEM) defines the overall system in the first phase, and several other organizations (usually suppliers) define the subsystems in parallel during the second phase. In this case, the primary organization hands over `System Extract`s, which represent the subsystems of the whole system. These subsystems contain subsystem VFBs, which are parts of the overall VFB. ⌋ *(RS_METH_00006, RS_METH_00002, RS_METH_00005)*

[TR_METH_01048] The overall system ⌈ The overall system defines the major public ECUs and topologies, and the subsystem design contributes by adding private ECUs and networks to the system. Please note that portions defined within a subsystem are not directly visible to any other subsystem or to the overall system. ⌋ *(RS_METH_00006, RS_METH_00005, RS_METH_00002)*

[TR_METH_01049] Interaction between organizations ⌈ Additionally, the software component structure of the System Extracts delivered by the primary organization can be transformed into a different structure for each ECU by the receiving organization (ECU System Description). In this case the System Extract of the primary organization can be considered as a requirement and the subsystem of the receiving organization represented by one or more ECU System Descriptions can be seen as a solution, which has to fulfill the delivered requirements. See Figure 2.4 for the scope of the System Extract and the ECU System Description and sections 2.5.3 to 2.5.5 for details. ⌋*(RS_METH_00006, RS_METH_00077)*

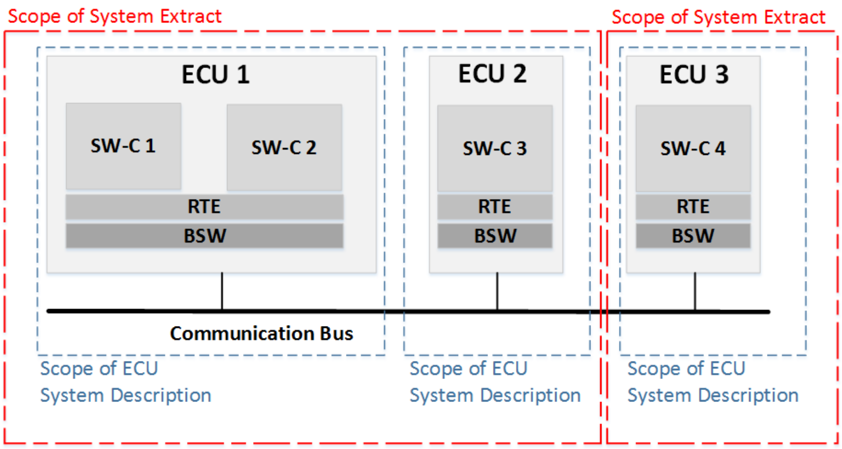

Figure 2.4: Scope of System Extract and ECU System Description

[TR_METH_01109] Producing ECU-specific deliverables ⌈ After the system design is complete, the portions that are related to a specific ECU are extracted producing a deliverable for each ECU, the so-called ECU Extract. Compared to the previous descriptions of the system or the ECU, the ECU Extract is fully decomposed and contains atomic software components only. It is the basis for ECU configuration. The activities for creating this deliverable are elaborated further in section 2.5.6. ⌋ *(RS_METH_00006, RS_METH_00005)*

[TR_METH_01110] Development of Software Components ⌈ In parallel to the system design, the software components (Delivered Atomic Software Components) are implemented according to the definitions required by the abstract VFB, the VFB or the subsystem VFB. Based on the external interfaces defined by the VFB, the internal behavior can be defined and finally the software component can be implemented, see Figure 2.5. The software components are delivered to be integrated in the ECUs, where they are deployed. Please note that the implementation of a software component is to a great extent independent from the configuration of the ECU. This is a key feature of the AUTOSAR methodology. See chapter 2.4 for more details. ⌋ *(RS_METH_00006, RS_METH_00002, RS_METH_00018)*

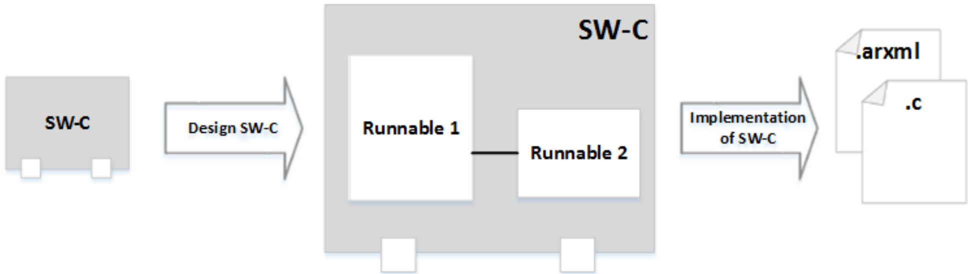

Figure 2.5: Development of a SW-C

[TR_METH_01111] Development of Basic Software modules ⌈ Since the Basic Software modules are independent of the VFB, they can be developed at any time before ECU integration. See Figure 2.6 and chapter 2.6 for more details. ⌋ *(RS_METH_00006, RS_METH_00003)*

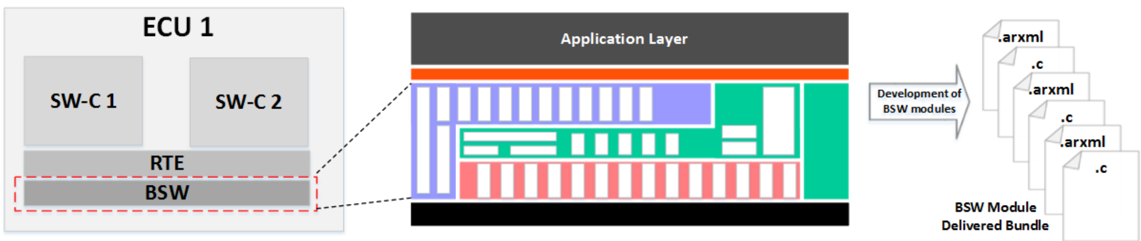

Figure 2.6: Development of BSW

[TR_METH_01112] Integration of AUTOSAR ECUs ⌈ The integration for an AUTOSAR ECU commences when the `BSW Module Delivered Bundle`s, `ECU Extract`, and the implementation of all `Delivered Atomic Software Components` are available. At this stage, the ECU is configured. The execution order is defined by scheduling tasks, and assigning `Software Component Runnables` to these tasks. Finally, the `Basic Software Modules` are configured. After the RTE is generated, the complete code is compiled and linked into an executable, see Figure 2.7. This step is elaborated in detail in chapter 2.7. ⌋*(RS_METH_00006, RS_METH_00002, RS_METH_00003, RS_METH_00004, RS_METH_00005)*

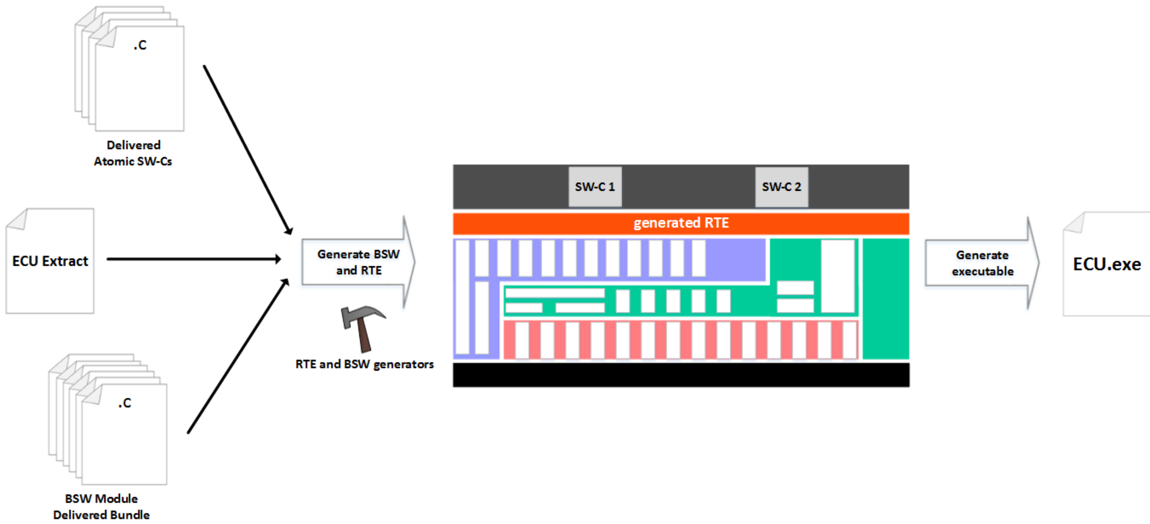

Figure 2.7: Integrate Software for one ECU

2.1.3 Workflow

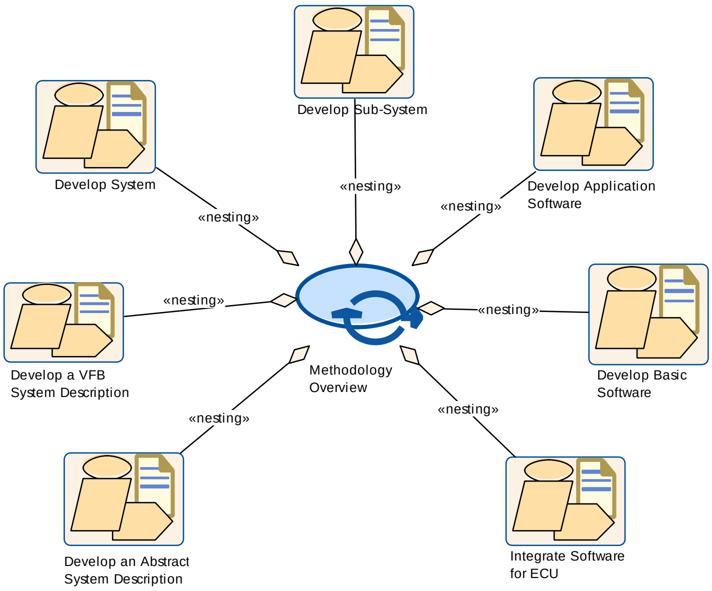

Figure 2.8: Methodology Overview: Overall Structure

Process Pattern	**Methodology Overview**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Methodology Overview		
Brief Description	High level view of the AUTOSAR Methodology		
Description	This Process Patterns contains the typical activities to develop an AUTOSAR system.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Develop Application Software	1	
Aggregates	Develop Basic Software	1	
Aggregates	Develop Sub-System	1	
Aggregates	Develop System	1	
Aggregates	Develop a VFB System Description	1	
Aggregates	Develop an Abstract System Description	1	
Aggregates	Integrate Software for ECU	1	

Table 2.1: Methodology Overview

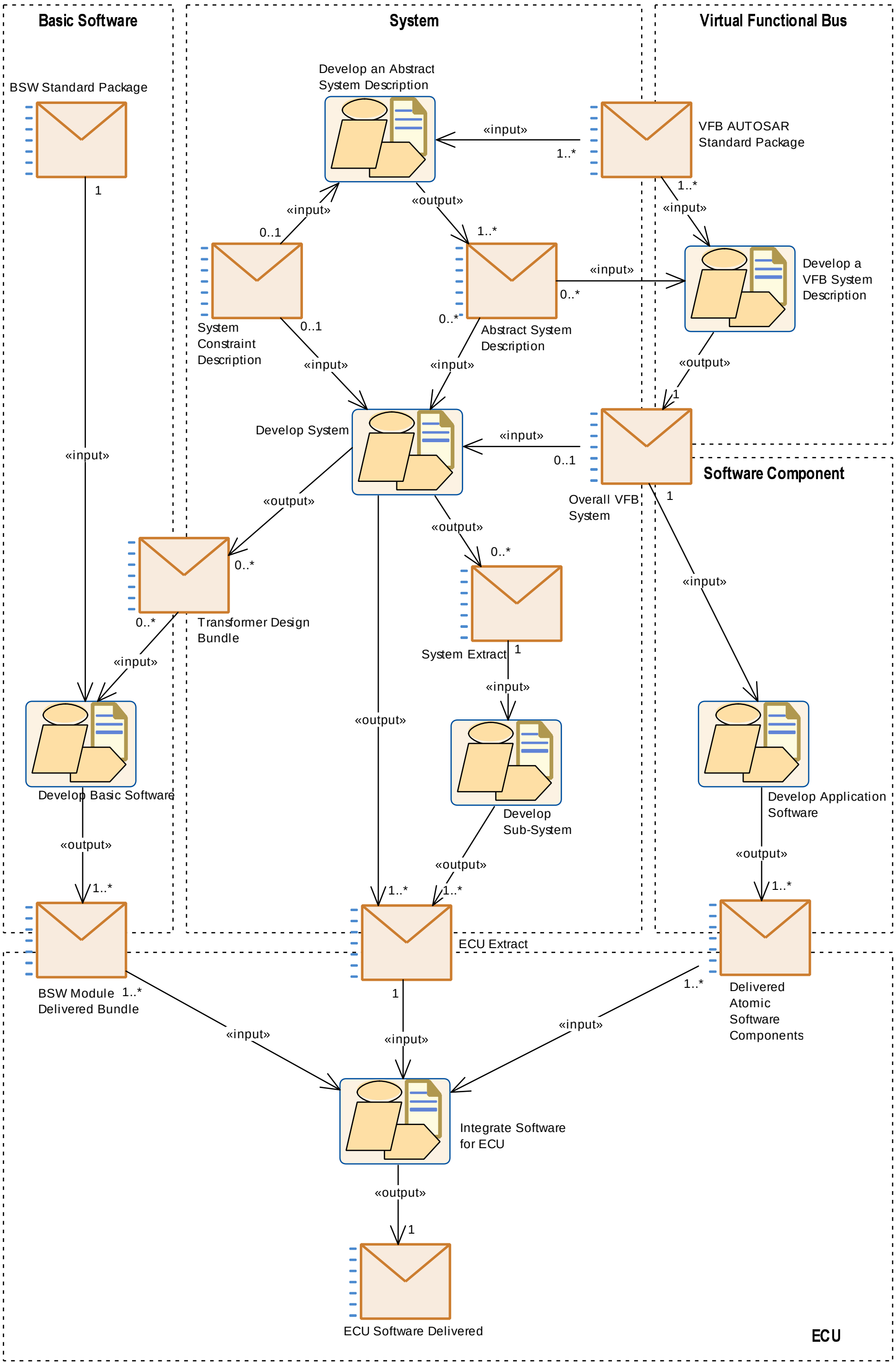

Figure 2.9: Methodology Overview: Workflow

2.2 Develop an Abstract System Description

2.2.1 Purpose

This Activity provides a rough outline of the creation of the Abstract System Description.

2.2.2 Description

[TR_METH_01050] Abstract System Description activity ⌈ Due to the fact that the overall view on vehicle functions can differ from the actual technical definition of the software architectures of individual ECUs, the optional activity Develop an Abstract System Description allows to define a view on the overall system from an abstract or functional perspective. This view describes a dedicated abstract VFB. During the further activities this abstract view is refactored into a technical view of the software architecture. ⌋*(RS_METH_00002, RS_METH_00078)*

For the purpose of this use case, this activity is split into sub-activities and tasks (see Figure 2.10) that are in detail described in Chapter 2.3 and 2.5.2:

- Data Model Development
- Component Model Development
- VFB Timing Development
- Define VFB Top Level
- Define VFB Component Constraints
- Design System
- Integrate Non AUTOSAR System at VFB level

In the Data Model Development activity, the set of VFB Interfaces, VFB Modes, and VFB Types that are used throughout the abstract VFB are defined. Please note, that these objects can be used in later steps by the VFB and the subsystem VFB as well.

[TR_METH_01051] Creation of an overall abstract system ⌈ In the Component Model Development activity, a component model is created which represents the overall system from a functional point of view, e.g. from a customer related perspective of vehicle functions, independent of a concrete vehicle platform design. During this process compositions might be modeled, which are not further refined into Atomic Software Components. However it is also possible to define atomic software components as well in this abstract VFB view. ⌋*(RS_METH_00002)*

[TR_METH_01052] Definition of a constraints in the context of an abstract system ⌈ In the context of the abstract VFB, the task Define VFB Component Constraints defines constraints w.r.t. software components of the abstract VFB. These constraints have to be considered when the abstract VFB is transformed into the concrete, technical VFB. ⌋*(RS_METH_00002)*

[TR_METH_01128] Integration of Non AUTOSAR Systems in the context of an abstract system ⌈ In parallel with the development of the `Abstract System Description` within an AUTOSAR process there may be functions that are developed based on another approach. The functionality of in-vehicle infotainment systems for instance is usually not covered in an AUTOSAR development process. Rather, development methods and platforms such as GENIVI (http://www.genivi.org/) for instance are employed that address the specific needs and conditions of infotainment system development. The integration of these functions into the overall system should be addressed as early as possible. For that purpose first a description of the non-AUTOSAR functionality (`Description of a Non-AUTOSAR System`) is needed, which must be provided by the non-AUTOSAR approach. Within the development of the `Abstract System Description` the functional interaction of the non-AUTOSAR functions and the AUTOSAR functions has to be specified that is based on the given descriptions of both parts. Since the non-AUTOSAR part is typically specified in a non-AUTOSAR format it must be translated to the corresponding AUTOSAR format (task `Translate Non-Autosar Description to Autosar Description`). Moreover, the information on the functional interaction must be incorporated in order to obtain one common view of the integrated system. The "Integration of Franca IDL Software Component Descriptions" document ([4]) defines a format for a VFB `Integration Connector` and a translation of Franca Interface Definitions - that are used in GENIVI - to AUTOSAR. It can be used for the development of an abstract description of an integrated AUTOSAR and GENIVI system. ⌋*(RS_METH_00016)*

[TR_METH_01053] Definition of a `System Description` in the context of an abstract system ⌈ Additionally to the definition of the abstract VFB, parts of the `System Description` can already be defined in the `Design System` activity, e.g. the topology and ECUs where SWCs of the abstract VFB are mapped to. This SW-C mapping from the abstract VFB to ECUs can be used as a methodological step to the definition of the concrete VFB. Please note that not all tasks of the `Design System` activity have to be performed in the context of an abstract system. ⌋*(RS_METH_00002, RS_METH_00005)*

2.2.3 Workflow

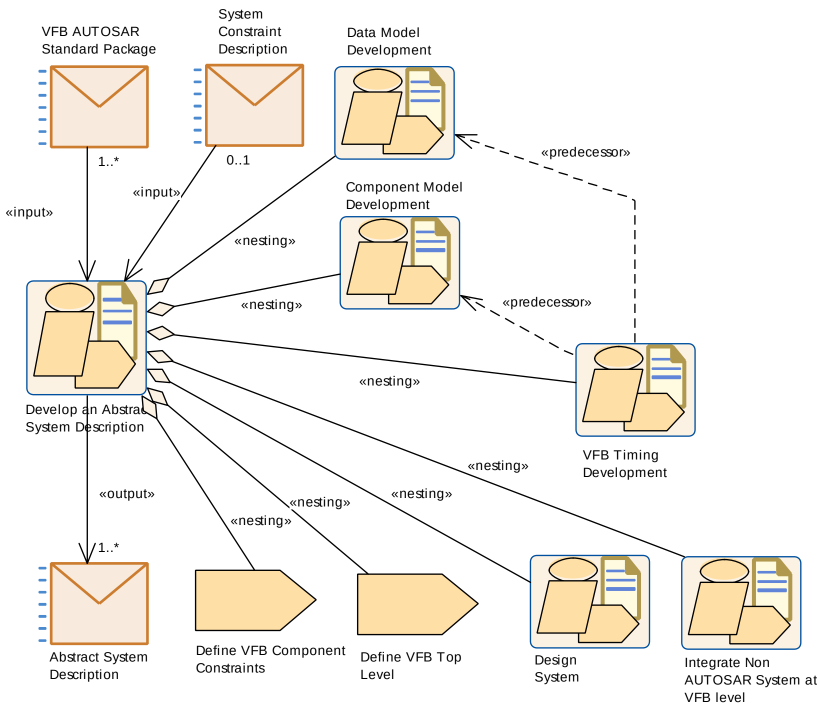

Figure 2.10: Develop an Abstract System Description

Activity	**Develop an Abstract System Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::System::Develop System		
Brief Description	Develop an abstract or functional view on the system.		
Description	This activity defines an abstract view on the overall system from an abstract or functional point of view. This activity is optional.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	System Constraint Description	0..1	In the context of the "Develop an Abstract System Description" activity, the constraints for the abstract or functional view on the system can be provided by the "System Constraint Description".
Consumes	VFB AUTOSAR Standard Package	1..*	
Produces	Abstract System Description	1..*	
Aggregates	Component Model Development	1	
Aggregates	Data Model Development	1	
Aggregates	Define VFB Component Constraints	1	
Aggregates	Define VFB Top Level	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Design System	1	In the context of the "Develop an Abstract System Description" activity, not all tasks have to be performed.
Aggregates	Integrate Non AUTOSAR System at VFB level	1	
Aggregates	VFB Timing Development	1	

Table 2.2: Develop an Abstract System Description

2.3 Develop a VFB System Description

2.3.1 Purpose

This `Activity` provides a rough outline of the creation of a `Virtual Functional Bus` view of a System. [3]

2.3.2 Description

[TR_METH_01054] **`Virtual Functional Bus`** ⌈ The `Virtual Functional Bus` (VFB) view of a System shows how the Systems software functions interact independently of any network topology or deployment of features across multiple ECUs. ⌋*(RS_METH_00033, RS_METH_00002)*

For more information on the VFB concept see [5]. For detailed information on the meta-model parts relevant for the VFB see [6].

For the purpose of this use case, this `Activity` is split into the following sub-activities:

- `Data Model Development`
- `Component Model Development`
- `VFB Timing Development`
- `Integrate Non AUTOSAR System at VFB level`
- `Define VFB Safety Information`

[TR_METH_01055] **`Data Model Development` activity** ⌈ In the `Data Model Development`, the set of `VFB Interfaces`, `VFB Modes`, and `VFB Types` that are used throughout the VFB are defined. Some of these have already been pre-defined by AUTOSAR (so-called “blueprints”), see 3.2.2.7 ⌋*(RS_METH_00002)*

[TR_METH_01056] **Definition of the VFB** ⌈ In the `Component Model Development` activity, the VFB is defined. This can either be done by the use of the abstract VFB as a basis, or is done directly by defining the software components. In case of using the abstract VFB as a basis, a mapping between the abstract and the concrete VFB can be established by performing the tasks `Define System View Mapping` (see Section 3.3.1.14 for more details). ⌋*(RS_METH_00002)*

Two general approaches can be separated:

- **[TR_METH_01057] Top-Down approach** ⌈ Following a Top-Down approach, the highest level `VFB Composition Component`s are created, and these are iteratively broken down to smaller components. At the leaves of the hierarchy the `VFB Atomic Software Component` are defined. Note that the activity can be even finished with empty `VFB Composition Component`s, allowing the detailing of the further structure at a later stage. ⌋*(RS_METH_00002)*
- **[TR_METH_01058] Bottom-Up approach** ⌈ If a Bottom-Up approach is used, then the `VFB Atomic Software Component`s are first defined, and aggregated into `VFB Composition Component`s. ⌋*(RS_METH_00002)*

[TR_METH_01059] Kinds of `VFB Atomic Software Component`s ⌈ Several special kinds of `VFB Atomic Software Component`s can be modeled in this activity:

- `VFB Atomic Application Software Component`s are the core elements. They are used to implement the feature algorithms.
- `VFB Parameter Component` are used to provide characteristic values, such as calibration parameters, to software components.
- `VFB Sensor Actuator Component`s provide the connection between physical sensors/actuators and the `VFB Atomic Application Software Component`s.
- `ECU Abstraction Software Component`s can be modeled at this level as well in oder to model the ECU input and output interfaces which are used by sensors and actuators.
- `Complex Driver Component`s also have to be modeled here, though their implementation is ECU specific, because their ports need to be connected at the VFB level.
- `VFB NvBlock Software Component` can be modeled at this level if application software accesses non-volatile data via ports.
- Empty `VFB Composition Component`s can be provided in case the detailed structure of the desired solution is not in the scope of this activity and will be left open to a later stage in the development.

⌋*(RS_METH_00002)*

[TR_METH_01129] **`Integrate Non AUTOSAR System at VFB level`** **activity** ⌈ In addition to the components that are specified with an AUTOSAR SwComponent Description there may be application components that are specified in other formats because they are developed within another application domain. In-vehicle infotainment components for instance are usually not developed with AUTOSAR means.

Rather, development methods and platforms such as GENIVI (http://www.genivi.org/) are employed that address the specific needs and conditions of infotainment system development. The integration of these components into the overall system should be addressed as early as possible. For that purpose the Description of a Non-AUTOSAR System must be incorporated into the VFB system description (VFB System). Since the non-AUTOSAR components are typically specified in a non-AUTOSAR format their descriptions must be translated to the corresponding AUTOSAR format (Task Translate Non-Autosar Description to Autosar Description). Moreover, the information on the interconnection of the components must be incorporated in order to obtain one common view of the integrated system. The document "Integration of Franca IDL Software Component Descriptions" ([4]) defines a format for a VFB Integration Connector and a translation of Franca Interface Definitions - that are used in GENIVI - to AUTOSAR. It can be used for the development of a VFB description of an integrated AUTOSAR-and-GENIVI system. ⌋ *(RS_METH_00016)*

[TR_METH_01149] Definition of VFB relevant safety information ⌈ In the optional activity Define VFB Safety Information the VFB relevant safety information is defined. Safety requirements and safety measures created at this development stage may be detailed (refined, decomposed, allocated, mapped, etc.) later on in the process (See chapter 2.14). ⌋ *(RS_METH_00081)*

After these activities are completed, the Virtual Functional Bus view of the System is defined. At this point, some VFB Software Component Mapping Constraints may already be known by design, or based on an analysis such as Define VFB Timing. These can be described to provide guidance to the downstream activities.

2.3.3 Workflow

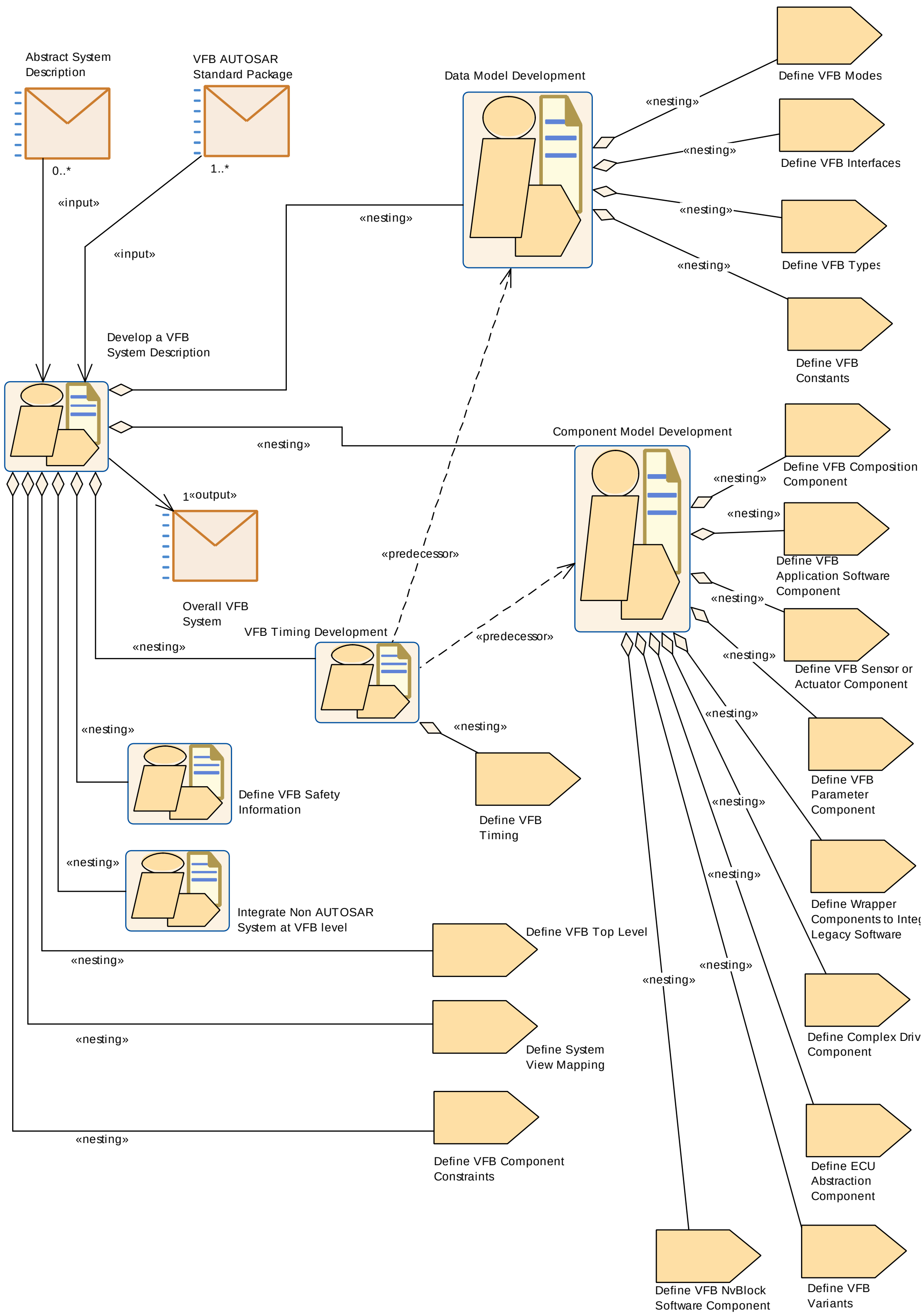

Figure 2.11: Develop a VFB System Description

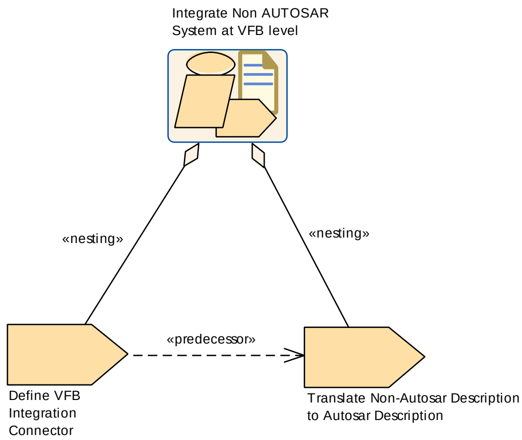

Figure 2.12: Integrate Non AUTOSAR System at VFB level

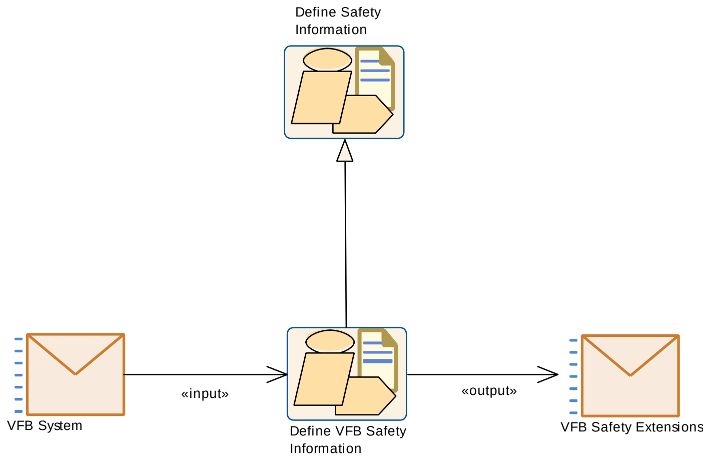

Figure 2.13: Define VFB Safety Information

Activity	**Develop a VFB System Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::VFB::Develop VFB		
Brief Description	This pattern describes the methodology to develop the Virtual Functional Bus view of the System.		
Description	The Virtual Functional Bus (VFB) view of a System shows how the Systems software and hardware functions interact independent of any network topology or deployment of features across muliple ECUs. This Activity is split into three sub-activities: • Data Model Development • Component Model Development • Timing Model Development • Integrate Non AUTOSAR System at VFB level • Define VFB Safety Information.		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	**Related Element**	**Mul.**	**Note**
Consumes	Abstract System Description	0..*	The abstract System Description is an optional input for the activity "Develop a VFB System Description". The VFB-related part of the Abstract System Description can be than refined to the concrete "Overall VFB System". Additionally, a mapping between those two views can be established.
Consumes	VFB AUTOSAR Standard Package	1..*	
Produces	Overall VFB System	1	
Aggregates	Component Model Development	1	
Aggregates	Data Model Development	1	
Aggregates	Define System View Mapping	1	
Aggregates	Define VFB Component Constraints	1	
Aggregates	Define VFB Safety Information	1	
Aggregates	Define VFB Top Level	1	
Aggregates	Integrate Non AUTOSAR System at VFB level	1	
Aggregates	VFB Timing Development	1	

Table 2.3: Develop a VFB System Description

Activity	**Data Model Development**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::VFB::Develop VFB		
Brief Description			
Description			
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregates	Define VFB Constants	1	
Aggregates	Define VFB Interfaces	1	
Aggregates	Define VFB Modes	1	
Aggregates	Define VFB Types	1	

Table 2.4: Data Model Development

Activity	**Component Model Development**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::VFB::Develop VFB		
Brief Description			
Description			
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregates	Define Complex Driver Component	1	
Aggregates	Define ECU Abstraction Component	1	
Aggregates	Define VFB Application Software Component	1	
Aggregates	Define VFB Composition Component	1	
Aggregates	Define VFB Nv Block Software Component	1	
Aggregates	Define VFB Parameter Component	1	
Aggregates	Define VFB Sensor or Actuator Component	1	
Aggregates	Define VFB Variants	1	
Aggregates	Define Wrapper Components to Integrate Legacy Software	1	

Table 2.5: Component Model Development

Activity	**VFB Timing Development**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::VFB::Develop VFB		
Brief Description			
Description			
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregates	Define VFB Timing	1	
Predecessor	Component Model Development	1	
Predecessor	Data Model Development	1	

Table 2.6: VFB Timing Development

Activity	**Integrate Non AUTOSAR System at VFB level**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::VFB::Develop VFB		
Brief Description	Incorporate the description of the non-AUTOSAR system and its connection with the AUTOSAR system into the AUTOSAR methodology activities.		
Description	Based on the description of the non-AUTOSAR system its connection with the AUTOSAR system is defined and specified using the VFB Integration Connector format. This is translated into an AUTOSAR description that becomes part of the VFB system description.		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Define VFB Integration Connector	1	
Aggregates	Translate Non-Autosar Description to Autosar Description	1	

Table 2.7: Integrate Non AUTOSAR System at VFB level

Activity	**Define VFB Safety Information**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::VFB::Develop VFB		
Brief Description	Defines all required safety information at VFB level.		
Description	In this activity, the safety information at VFB level is defined. The safety information can be refined or completed in further development phases.		
Extends	Define Safety Information		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	VFB System	1	
Produces	VFB Safety Extensions	1	

Table 2.8: Define VFB Safety Information

2.4 Develop Software Components

2.4.1 Develop an Atomic Software Component

2.4.1.1 Purpose

This `Activity` provides a rough outline of the creation of an `Atomic Software Component`.

2.4.1.2 Description

[TR_METH_01060] **`Develop an Atomic Software Component`** **activity** ⌈This is the generic `Activity` valid for several kinds of Atomic Software Components. The first step is to create design, including the runnables, events, interrunnable variables, etc. Once this is complete, the contract header files can be created and the software component can be implemented.

Optionally, the safety relevant information for the software component and all contained elements can be defined (See chapter 2.14). If the software component is developed as a SEooC (Safety Element out of Context) and the safety requirements are not fully known at development time, the ASIL attribute can be set to indicate the integrity level the component was developed for, i.e. in the development process all development process related requirements of ISO 26262 for the specified ASIL have been applied. ⌋*(RS_METH_00002, RS_METH_00038)*

Note that the method of implementation, quality, testing, etc. are beyond the scope of this activity.

After the component is implemented and successfully compiled, its resources are measured and stored as part of the software component description for further usage by downstream processes.

The pattern also includes the optional tasks of creating a timing model, binding pre-build-variants and evaluating variants, all in the scope of the atomic software component. Note that the sequence of these optional tasks within the `Activity` is only one possible example.

2.4.1.3 Workflow

Figure 2.14 shows the work breakdown assumed for this use case. The next two figures 2.16 and 2.17 show all the tasks and work products of the method library involved in this use case.

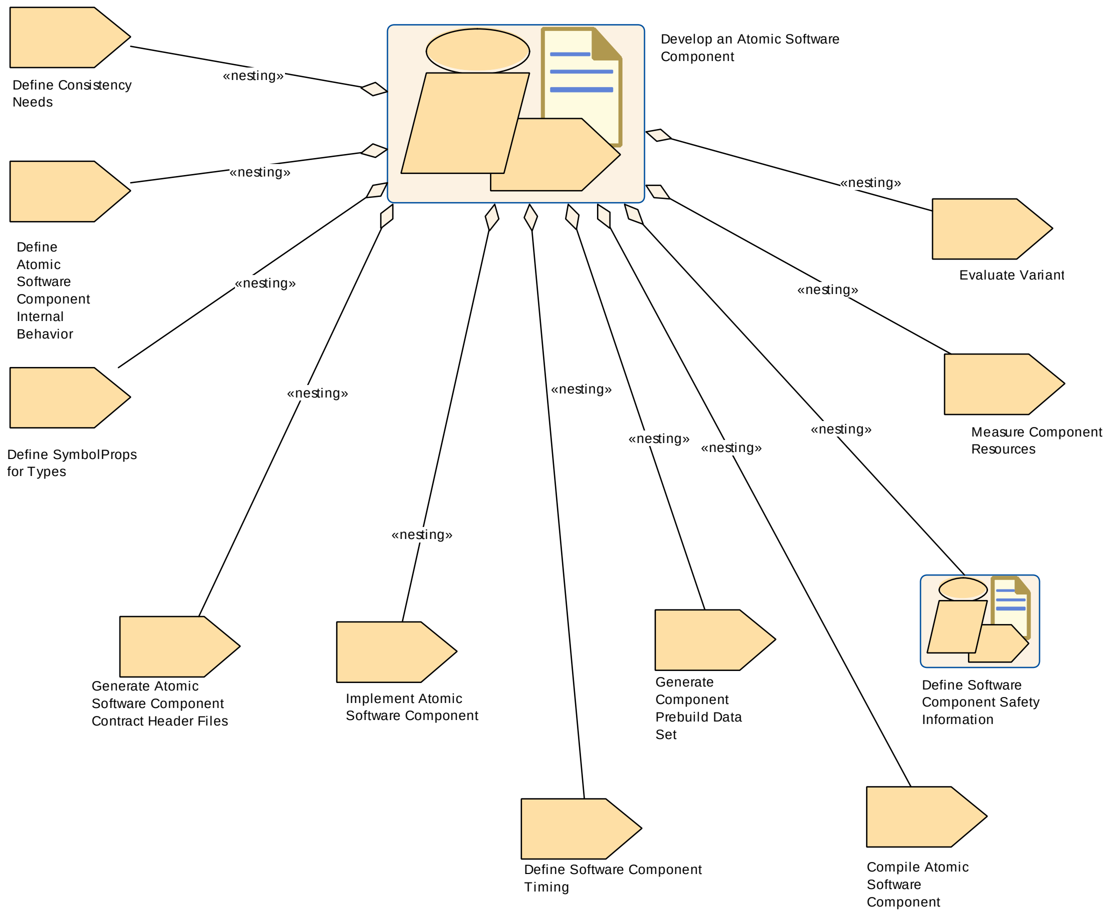

Figure 2.14: Develop an Atomic Software Component

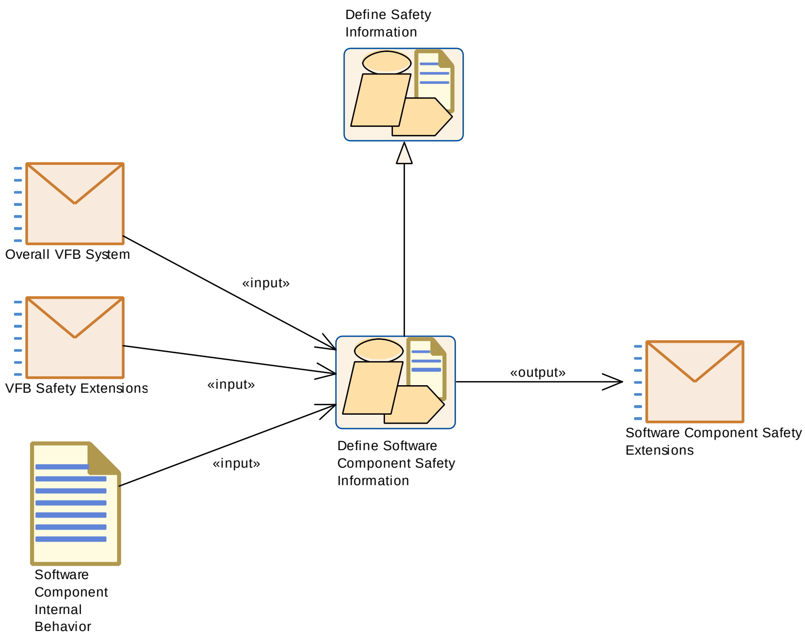

Figure 2.15: Define Software Component Safety Information

Activity	Develop an Atomic Software Component		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Software Component::Develop Atomic SWC		
Brief Description			
Description	This is the generic pattern valid for several kinds of Atomic Software Components. The first step is to create design, including the runnables, events, interrunnable variables, etc. Once this is complete, the contract header files can be created and the software component can be implemented. Note that the method of implementation, quality, testing, etc. are beyond the scope of this capability pattern. After the component is implemented and successfully compiled, its resources are measured and stored as part of the software component for further usage by downstream processes. The pattern also includes the optional tasks of creating a timing model, defining safety relevant information, binding prebuild-variants and evaluating variants, all in the scope of the Atomic Software Component. Note that the sequence of these optional tasks within the capability pattern is only one possible example.		
Extended by	Develop Application Software, Develop a Complex Driver Component, Develop a Sensor Actuator Component, Develop an ECU Abstraction Component, Develop an NvBlock Software Component, Optimize a Software Component for a Specific Target		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Compile Atomic Software Component	1	
Aggregates	Define Atomic Software Component Internal Behavior	1	
Aggregates	Define Consistency Needs	1	Used for defining the consistency relations between a group of RunnableEntitys and a group of DataPrototypes.
Aggregates	Define Software Component Safety Information	1	
Aggregates	Define Software Component Timing	1	
Aggregates	Define Symbol Props for Types	1	Used for solving name conflicts on the level of component or data types.
Aggregates	Evaluate Variant	1	
Aggregates	Generate Atomic Software Component Contract Header Files	1	
Aggregates	Generate Component Prebuild Data Set	1	

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Implement Atomic Software Component	1	
Aggregates	Measure Component Resources	1	

Table 2.9: Develop an Atomic Software Component

Activity	**Define Software Component Safety Information**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Software Component::Develop Atomic SWC		
Brief Description	Defines all required safety information for a software component.		
Description			
Extends	Define Safety Information		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Overall VFB System	1	
Consumes	Software Component Internal Behavior	1	
Consumes	VFB Safety Extensions	1	
Produces	Software Component Safety Extensions	1	

Table 2.10: Define Software Component Safety Information

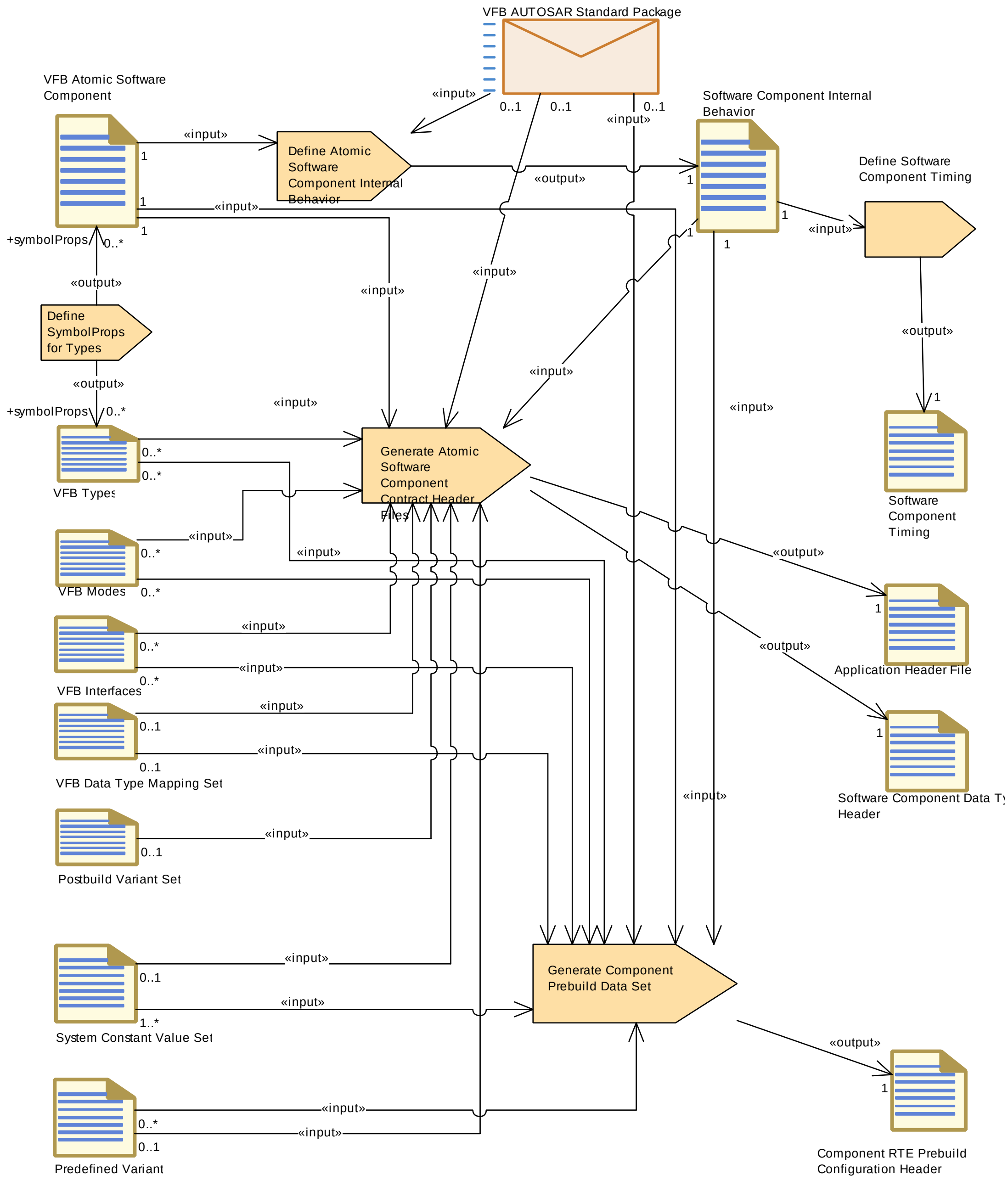

Figure 2.16: Develop an Atomic Software Component - Detailed view with work products (1)

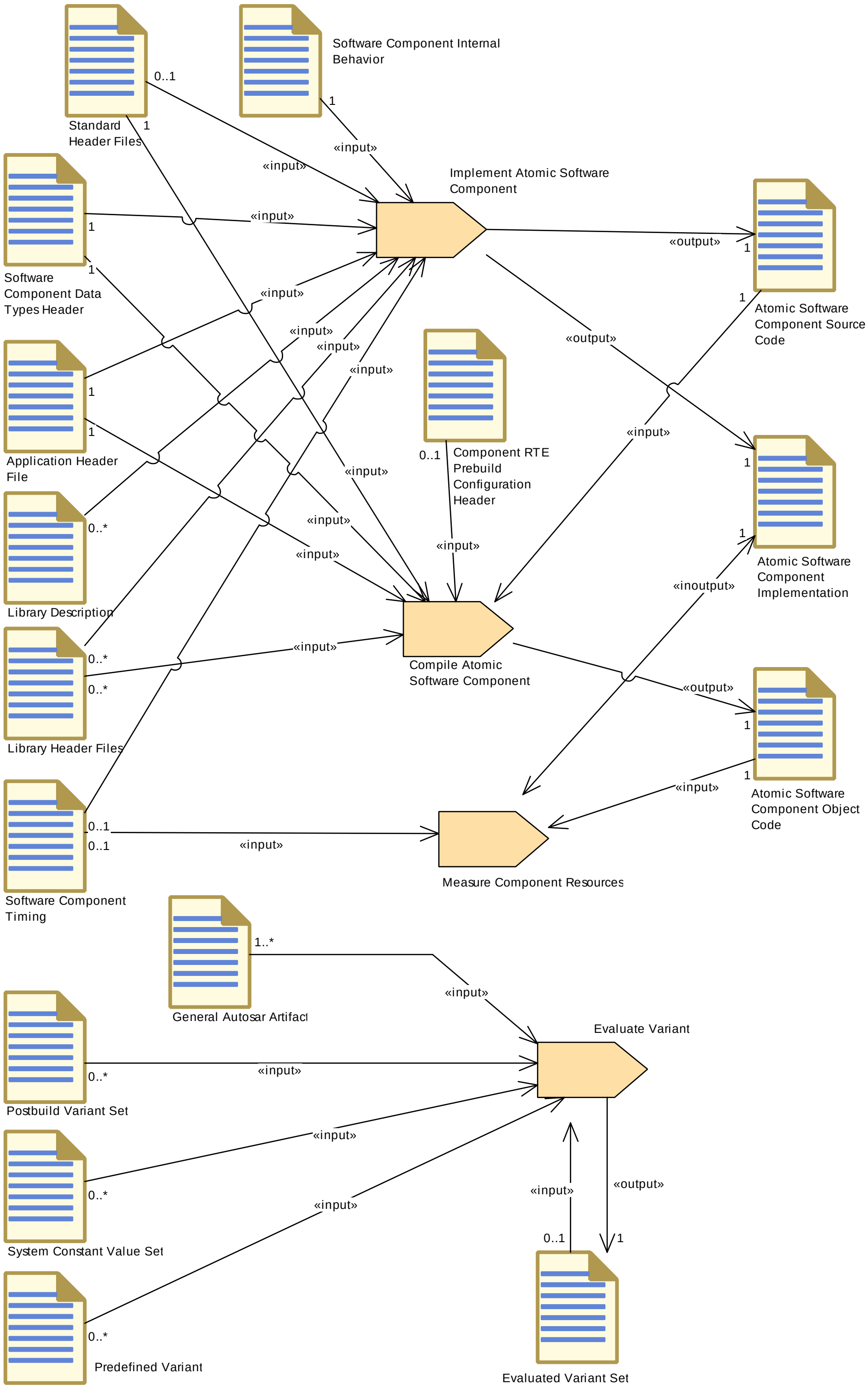

Figure 2.17: Develop an Atomic Software Component - Detailed view with work products (2)

2.4.2 Develop Application Software

2.4.2.1 Purpose

This Activity provides a rough outline of the creation of one or more Application Software Components.

2.4.2.2 Description

[TR_METH_01061] **Develop Application Software activity** ⌈ This Activity describes the work flow and the necessary activities in terms of the AUTOSAR methodology to develop one or more Application Software Components. The work flow shall allow a more or less independent development of the software components core functionality. These activities have to be performed for each Application Software Component. ⌋*(RS_METH_00002, RS_METH_00006)*

2.4.2.3 Workflow

The detailed workflow can be derived from the generic activity Develop an Atomic Software Component.

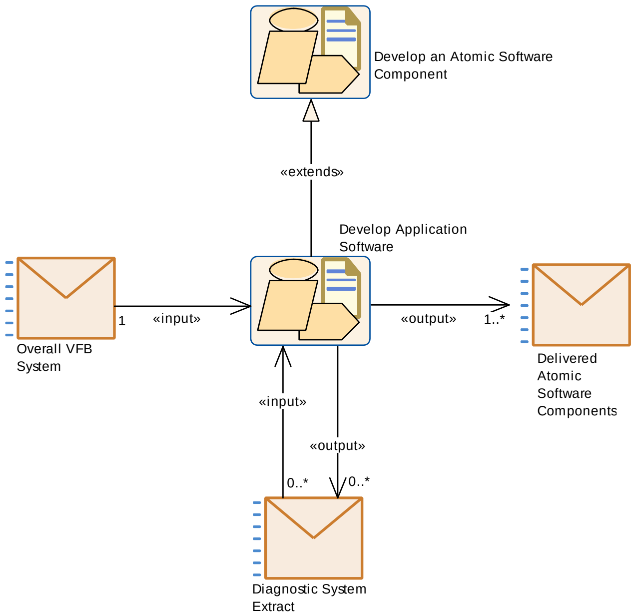

Figure 2.18: Develop Application Software

Activity	**Develop Application Software**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Software Component::Develop Application SWC		
Brief Description			
Description	This pattern describes the workflow and the necessary activities in terms of the AUTOSAR methodology for the development of application software components. The workflow shall allow a more or less independent development of the software component core functionality. These activities have to be performed for every application software component.		
Extends	Develop an Atomic Software Component		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Diagnostic System Extract	0..*	The Diagnostic System Extract contains diagnostic information that serves as a requirement for the software developer.
Consumes	Overall VFB System	1	The application software needs to refer to the relevant elements of the overall VFB system such as Software Component Types, Port Interfaces and Data Types.
Produces	Delivered Atomic Software Components	1..*	Complete description of a set of AtomicSoftwareComponents including implementation (incl. source or object code files)
Produces	Diagnostic System Extract	0..*	Diagnostic information relevant to the SW-Cs is provided as a part of the Diagnostic System Extract and can contain relationships to the SW-C's service needs.

Table 2.11: Develop Application Software

2.4.3 Uses Cases for more Specialized Software Components

2.4.3.1 Purpose

These `Activities` provides a rough outline of the creation of more specialized components and of the ECU specific optimization of a software component.

2.4.3.2 Description

These `Activities` describe the work flow and the necessary activities in terms of the AUTOSAR methodology to develop more specialized components, which could be partially hardware or ECU dependent.

2.4.3.3 Workflow

These work flows are for the most part derived from the generic activity Develop an Atomic Software Component. The diagrams show the required extensions.

Note the development of a Service Component does not fall into this category of use cases, because it is for the most part generated during integration time.

For the development of a VFB Parameter Component refer to the calibration use case 2.9.

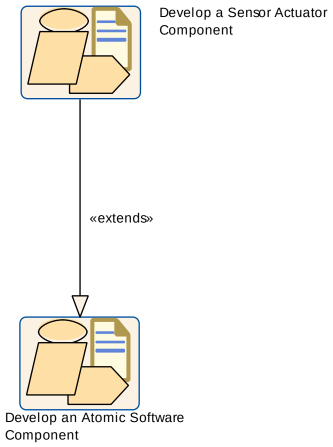

Figure 2.19: Develop a Sensor or Actuator Component

Activity	**Develop a Sensor Actuator Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Software Component::Develop Sensor-Actuator Component		
Brief Description	Show how to develop a Sensor Actuator Component		
Description	Activities to develop a VFB Sensor Actuator Component, i.e. component that represents a physical sensor or actuator.		
Extends	Develop an Atomic Software Component		
Relation Type	***Related Element***	***Mul.***	***Note***

Table 2.12: Develop a Sensor Actuator Component

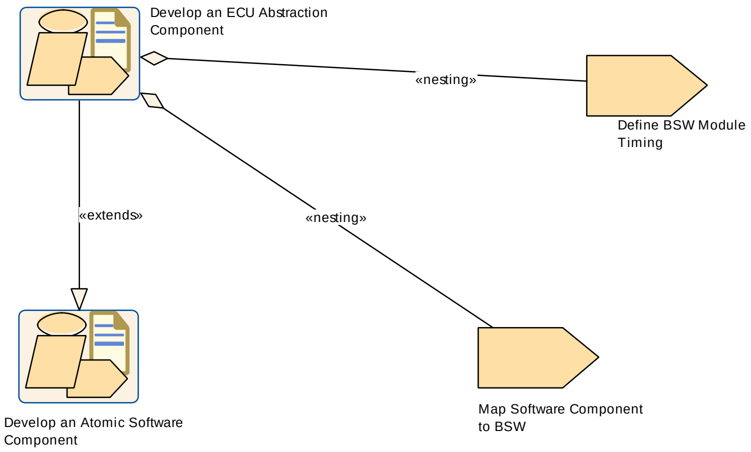

Figure 2.20: Develop an ECU Abstraction Component

Activity	**Develop an ECU Abstraction Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Software Component::Develop Ecuabs Component		
Brief Description	Show how to develop an ECU Abstraction Component.		
Description	Activities to develop an ECU Abstraction Software Component, i.e. a component that implements an ECU Abstraction..		
Extends	Develop an Atomic Software Component		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Define BSW Module Timing	1	
Aggregates	Map Software Component to BSW	1	

Table 2.13: Develop an ECU Abstraction Component

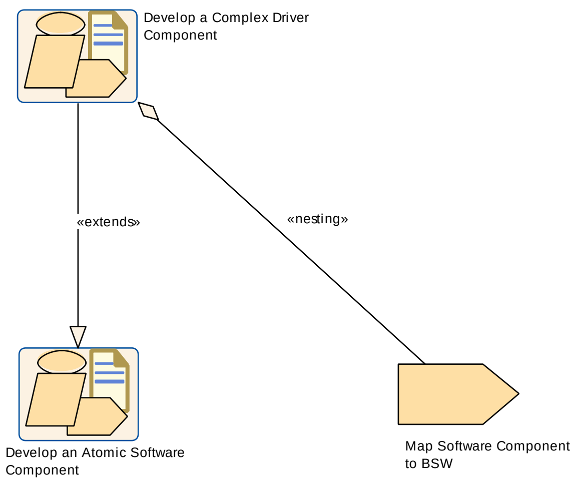

Figure 2.21: Develop a Complex Driver Component

Activity	**Develop a Complex Driver Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Software Component::Develop CDD Component		
Brief Description	Show how to develop a Complex Driver Component		
Description	Show how to develop a Complex Driver Component		
Extends	Develop an Atomic Software Component		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Map Software Component to BSW	1	

Table 2.14: Develop a Complex Driver Component

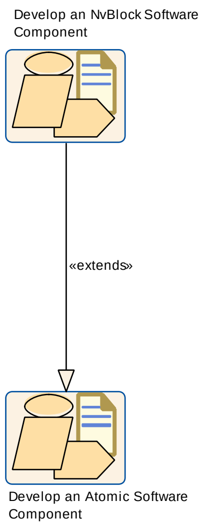

Figure 2.22: Develop an NvBlock Software Component

Activity	**Develop an NvBlock Software Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Software Component::Develop NvBlock Software Component		
Brief Description			
Description	Activities to develop an NvBlock Software Component. An NvBlockSoftwareComponentType (designed as part of activity Component Model Development) allows the application software to access non-volatile data in a convenient way via ports. The NvBlock Software Component takes over the management and buffering of data within blocks including data exchange with the underlying basic software (NvM). Optionally, it implements special writing strategies (e.g. cyclic writing). The development activities are similar to the generic activity Develop an Atomic Software Component with the following differences: • The description of the NvBlockNeeds within a NvBlockSoftwareComponentType is done in response to requirements given by the application software as part of their own NvBlockNeeds. These are part of their Software Component Internal Behavior which means that this level must be available when the NvBlockSoftwareComponentType is finally designed. • The creation of an Software Component Internal Behavior within NvBlockSoftwareComponentType is optional. This artifact is only needed if special writing strategies have to implemented by the RTE or if the application software needs a direct access (via client-server ports) to the NvM. • The source code of an NvBlockSoftwareComponentType will be generated during integration as part of the artifact RTE Source Code. Therefore no source code and no Atomic Software Component Implementation needs to be created during this activity. Note that if non-volatile data are accessed by the application software via an NvBlockSoftwareComponentType, it is not required to define a ServiceComponentType for this use case.		
Extends	Develop an Atomic Software Component		
Relation Type	***Related Element***	***Mul.***	***Note***

Table 2.15: Develop an NvBlock Software Component

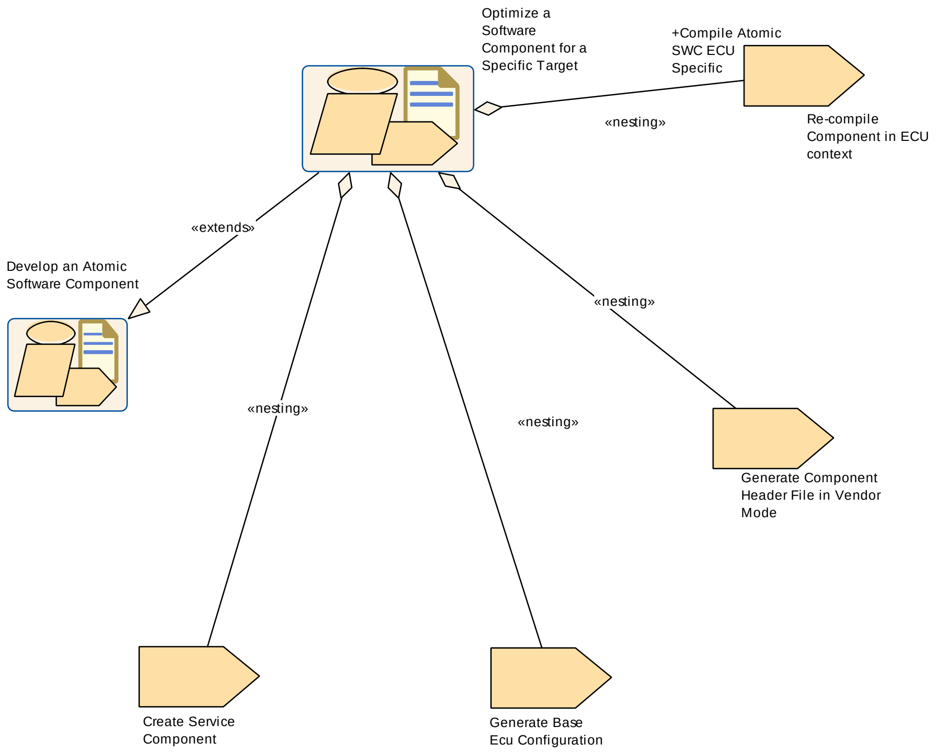

Figure 2.23: Optimize Software Component

Activity	**Optimize a Software Component for a Specific Target**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Software Component::Optimize Software Component		
Brief Description	Show how to optimize a software component for a specific target.		
Description	In practice the integration of an application software component has to consider some optimizations to meet performance or resource requirements. The Component API might be much more efficient, if it will be generated particularly adapted to the concrete ECU configuration, e.g. via using macro definitions instead of function calls for some RTE interaction. In fact this should not change the Component Implementation (i.e. the C-sources). That means now we have a different set of component headers, which include the ECU-configuration-specific optimizations. Note: This use case shows the typical steps needed until the recompilation with the optimized header file can be done. It does not show all the other steps needed for the ECU build.		
Extends	Develop an Atomic Software Component		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Create Service Component	1	
Aggregates	Generate Base Ecu Configuration	1	
Aggregates	Generate Component Header File in Vendor Mode	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Re-compile Component in ECU context	1	Compile Atomic SWC ECU Specific:

Table 2.16: Optimize a Software Component for a Specific Target

2.5 Develop System and Subsystems

2.5.1 Overview

2.5.1.1 Purpose

The Activities to develop the artifacts on the system level include the optional development of the abstract system (see Chapter 2.2), the development of an overall (technical) system and optionally the refinement into one or more subsystems. The reason for this split is, that the latter may be done by another organization, as has already been pointed out in 2.1.2.

2.5.1.2 Description

[TR_METH_01065] Develop System and Develop Sub-System activities ⌈ Figures 2.24 and 2.25 show the main inputs and outputs of these two major activities and how they are refined into sub-activities. Note that the activity Generate ECU Extract and Define System Safety Information can be performed as part of Develop System and Develop Sub-System as well. Optionally a mapping between two different system views represented by different System Descriptions can be added (see Section 3.3.1.14) and a specification of the transformer technology for the communication can be defined. ⌋*(RS_METH_00005, RS_METH_00002)*

[TR_METH_01066] Creation of a System Extract and an ECU Extract ⌈ Depending on the intended work split, the System Configuration Description produced during this activity can be used as a basis

1. to create one or more so-called System Extracts as a basis for further refinement as sub-systems (see 2.5.5)
2. or to generate ECU Extracts which directly contain all relevant information to be integrated on an ECU (see 2.5.6)

In the first case, only an outer system is defined. Based on the outer system, one or more System Extracts can be delivered. The System Extract is not fully decomposed and still needs to be refined before it forms the basis for the ECU configuration. In order to distinguish between the delivered System Extracts and the refined sub-system, one or more ECU System Descriptions are created as a basis for further refinement (See activity Create ECU System Description). Atomic Software Components, additional ECUs, Networks and the resulting communication will be added during the refinement step in the activity Design Sub-System. ⌋*(RS_METH_00005, RS_METH_00002)*

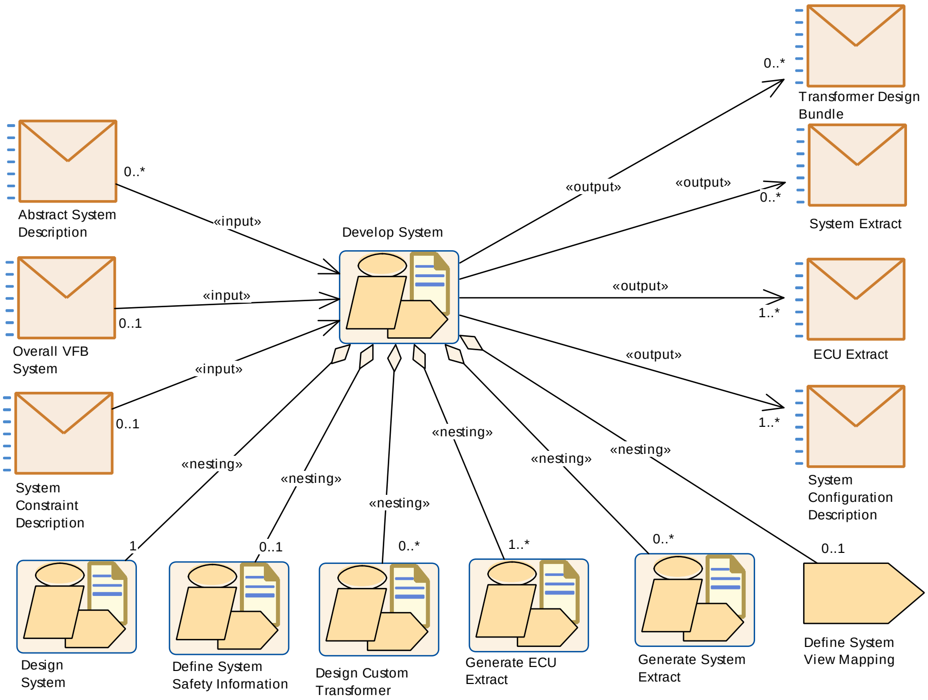

Figure 2.24: Structure of Activity: Develop System

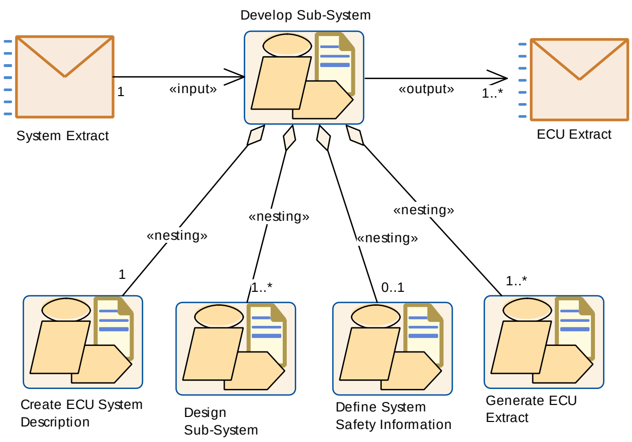

Figure 2.25: Structure of Activity: Develop Subsystem

Figure 2.26 shows how the major deliverables produced during these activities are related and how they refer to artifacts describing the software.

[TR_METH_01067] **Abstract System Description deliverable** ⌈ The Abstract System Description extends the general System Description. The System View Mapping maps the different views on the system together, e.g. different overall VFB systems (e.g. Abstract System Description with System

Configuration Description), or the overall VFB system with the VFB System Extract description. ⌋(*RS_METH_00005, RS_METH_00002*)

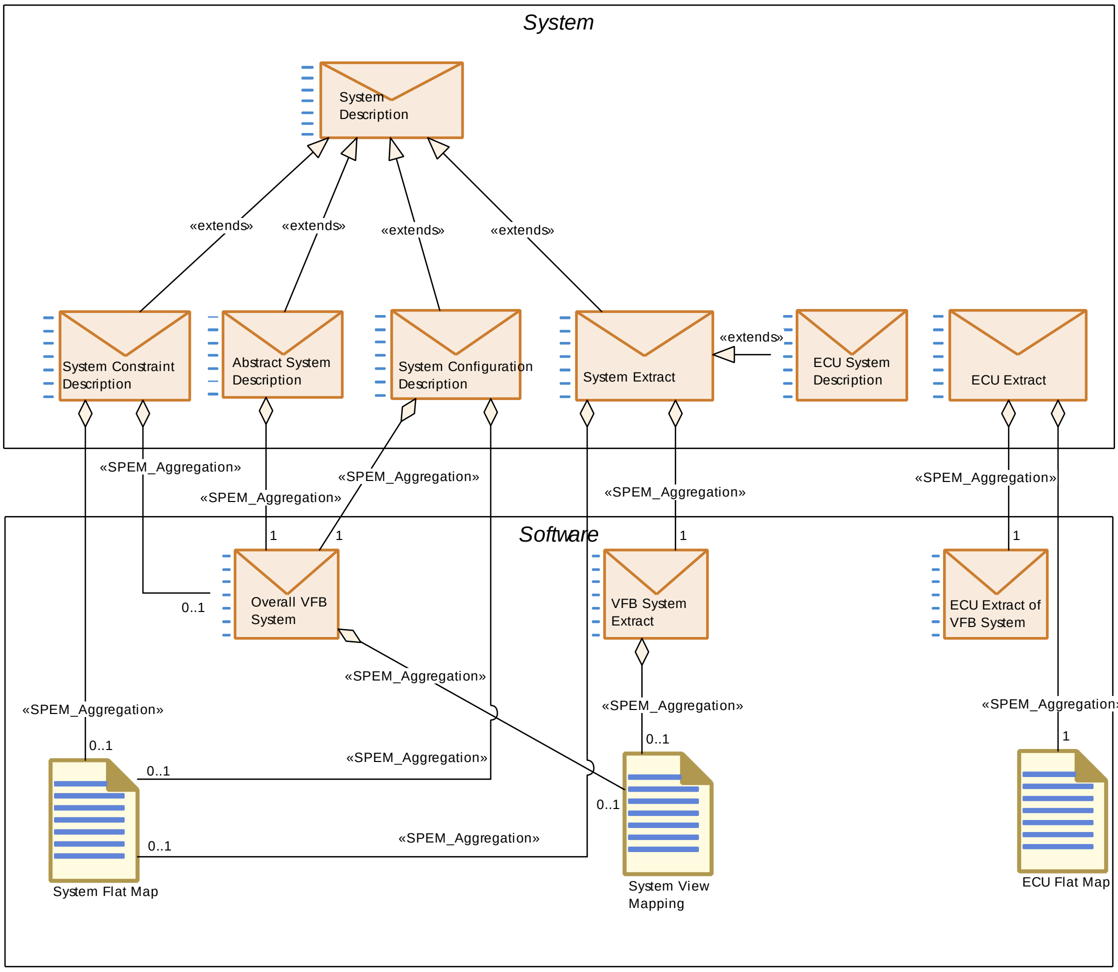

Figure 2.26: Overview on the different roles of deliverables based on System Description

Note that all the deliverables based on the generic deliverable System Description as well as the ECU Extract consist of ARXML files that are using the meta-model element System as the root element, from where the other information can be traced down.

Activity	**Develop System**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::System::Develop System		
Brief Description			
Description	Develop the description of an overall AUTOSAR System as a basis to deliver System and/or ECU extracts.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Abstract System Description	0..*	The abstract System Description is an optional input for the activity "Develop System". Please note, that in this step the Abstract System Description is refined to a System Description.

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Overall VFB System	0..1	Usually the System refers to elements of an overall VFB descriptions. But for the description of a legacy system, this input might be empty.
Consumes	System Constraint Description	0..1	
Produces	ECU Extract	1..*	
Produces	System Configuration Description	1..*	
Produces	System Extract	0..*	
Produces	Transformer Design Bundle	0..*	
Aggregates	Define System Safety Information	0..1	
Aggregates	Define System View Mapping	0..1	
Aggregates	Design Custom Transformer	0..*	
Aggregates	Design System	1	
Aggregates	Generate ECU Extract	1..*	
Aggregates	Generate System Extract	0..*	

Table 2.17: Develop System

Activity	**Develop Sub-System**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::System::Develop System		
Brief Description			
Description	Develop the description of a sub-system based on a given System Extract.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	System Extract	1	
Produces	ECU Extract	1..*	
Aggregates	Create ECU System Description	1	
Aggregates	Define System Safety Information	0..1	
Aggregates	Design Sub-System	1..*	
Aggregates	Generate ECU Extract	1..*	

Table 2.18: Develop Sub-System

2.5.2 Design System

2.5.2.1 Purpose

This Activity provides a rough outline of the design steps leading to an AUTOSAR System Configuration Description and the system-specific part of the Abstract System Description, including its topology, deployment, communication matrix, etc.

2.5.2.2 Description

[TR_METH_01068] Inputs and Output of the Design System activity ⌈ The design of an AUTOSAR System Configuration Description and the system-specific part of the Abstract System Description uses input information from a System Constraint Description and is based on an Overall VFB System for the software part. Optionally, the Abstract System Description that represents the functional view on the system can be used as an input. Please note that the inputs and output are depicted in the top-level activities which aggregates the activity Design System.

The activity involves the creation of a Topology, ECU Resources Descriptions, and the interconnection between ECU instances. ⌋*(RS_METH_00005, RS_METH_00002, RS_METH_00078, RS_METH_00079)*

[TR_METH_01069] Deployment of AUTOSAR Software Components ⌈ The AUTOSAR Software Components defined within the VFB Top Level System Composition are then deployed to the ECU instances. ⌋*()*

[TR_METH_01070] Description of network signals ⌈ The required network signals are identified and a mapping is done to System Signals to implement the VFB. System Signal Groups, are defined to keep certain signals grouped together for consistent transmission. System Signals are then defined and form the initial input to design the Communication. ⌋*(RS_METH_00005)*

[TR_METH_01071] Description of design constraints ⌈ During this stage, design constraints can also be defined Mapping of Software Components to Implementations, Mapping of Software Components to ECUs and Signal Path Constraints. These constraints serve many purposes including the ability for tools to use them to optimization a system, to interface with legacy ECUs, and to "lock" design decision between iterations. ⌋*(RS_METH_00005, RS_METH_00002, RS_METH_00041, RS_METH_00020)*

Note: The mapping of software components to implementations is optional and needed only if those components are specifically required to be used in an ECU.

[TR_METH_01155] Definition of serialization ⌈ There are two approaches possible for defining the serialization. The first approach provides the necessary information based on the network representation, the second approach based on implementation data types. For details of these two approaches, please see [7]. ⌋*(RS_METH_00005)*

[TR_METH_01156] Use case: Serialization based on network representation ⌈ The OEM defines the network representation on network signal (ISignal) level. This network representation is used by the `Serializer Transformer` to create the byte stream. If not provided by the OEM, the Tier1s are free to choose implementation data types for the application software. ⌋*(RS_METH_00005)*

[TR_METH_01157] Use case: Serialization based on implementation data types ⌈ The OEM defines the same implementation data types for the root software composition of communicating Ecu instances. These implementation data types are used by the `Serializer Transformer` to create the byte stream. Tier1s are free to use arbitrary implementation data types for the application SW inside the root software composition. ⌋*(RS_METH_00005)*

2.5.2.3 Workflow

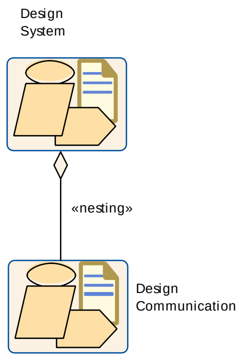

Figure 2.27: Structure overview: Design System

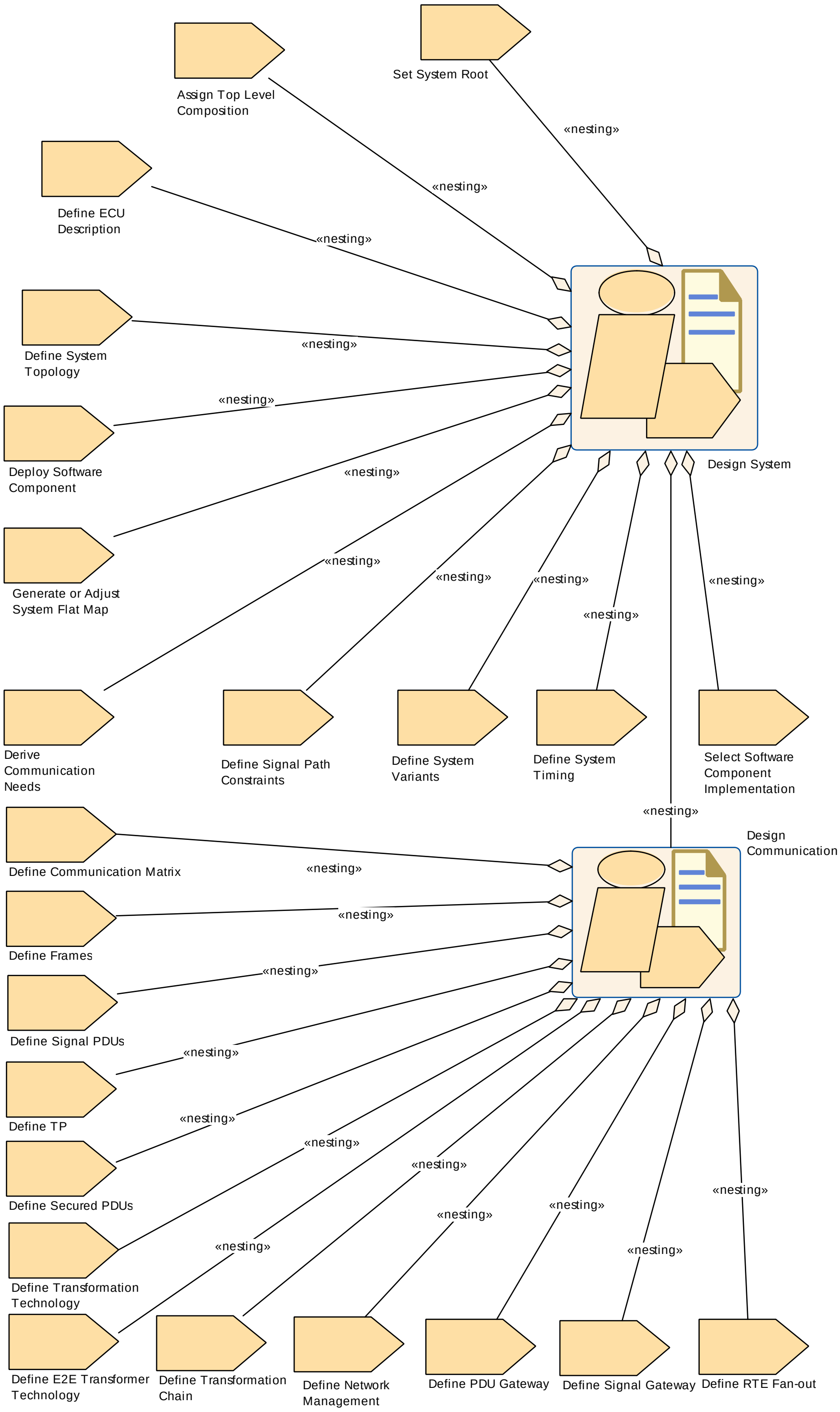

Figure 2.28: Nesting relationship: Design System

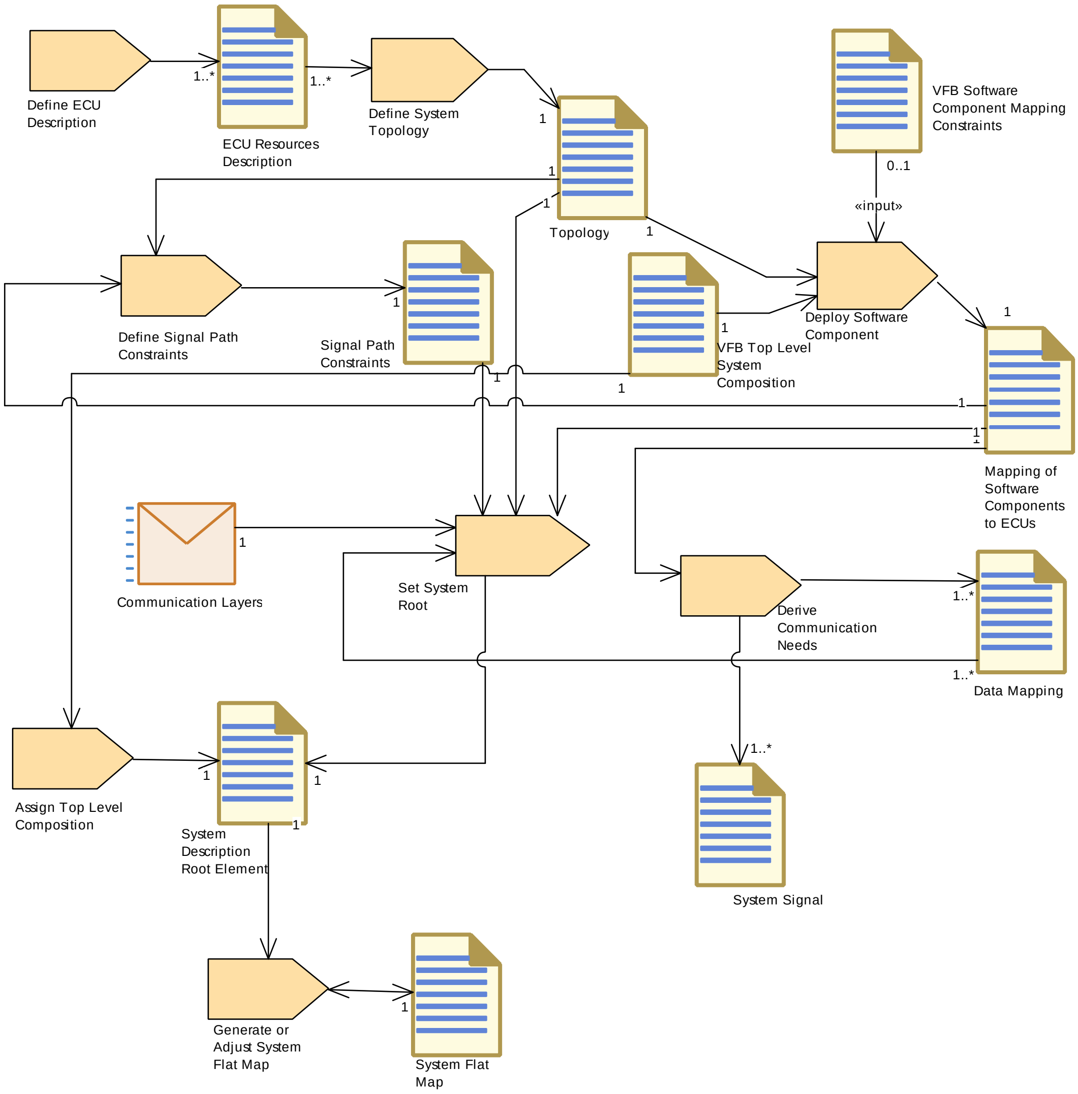

Figure 2.29: Detailed work flow for: Design System

Activity	**Design System**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::System::Design System		
Brief Description	Initial work to create a topology, map a VFB onto that topology and determine the ECU resources each ECU needs.		
Description	The design of an AUTOSAR System involves the creation of a Topology, ECU Resources Descriptions, and the interconnection between ECU instances. The software components defined within the VFB Top Level System Composition are then deployed to the ECU instances. The required network signals are identified and a mapping is done to System Signals to implement the VFB. System Signal Groups, are defined to keep certain signals grouped together for atomic transmission. System Signals are then defined and form the initial input to design the Communication Matrix. During this stage, design constraints can also be defined (Mapping of Software Components to Implementations, Mapping of Software Components to ECUs, Signal Path Constraint). These constraints serve many purposes including the ability for tools to use them to optimization a system, to interface with legacy ECUs, and to "lock" design decision between iterations. Notes: The mapping of software components to implementations is optional and needed only if those components are specifically required to be used in an ECU.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Assign Top Level Composition	1	
Aggregates	Define ECU Description	1	
Aggregates	Define Signal Path Constraints	1	
Aggregates	Define System Timing	1	
Aggregates	Define System Topology	1	
Aggregates	Define System Variants	1	
Aggregates	Deploy Software Component	1	
Aggregates	Derive Communication Needs	1	
Aggregates	Design Communication	1	
Aggregates	Generate or Adjust System Flat Map	1	
Aggregates	Select Software Component Implementation	1	

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Set System Root	1	

Table 2.19: Design System

Activity	**Design Communication**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::System::Design System		
Brief Description			
Description	Describe all communication layers. and define the mapping of the triggering elements within the Physical Channels to the communication connector ports for the individual ECUs. Because the triggering elements are aggregated as splitable elements within the Physical Channels it is possible to define them in an artifact separated from the Topology.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Define Communication Matrix	1	
Aggregates	Define E2E Transformer Technology	1	
Aggregates	Define Frames	1	
Aggregates	Define Network Management	1	
Aggregates	Define PDU Gateway	1	
Aggregates	Define RTE Fan-out	1	
Aggregates	Define Secured PDUs	1	
Aggregates	Define Signal Gateway	1	
Aggregates	Define Signal PDUs	1	
Aggregates	Define TP	1	
Aggregates	Define Transformation Chain	1	
Aggregates	Define Transformation Technology	1	

Table 2.20: Design Communication

2.5.3 Generate System Extract

2.5.3.1 Purpose

This `Activity` provides an extract of the system description for a specific sub-system.

2.5.3.2 Description

Generate a System Extract which is a basis to develop a sub-system.

2.5.3.3 Workflow

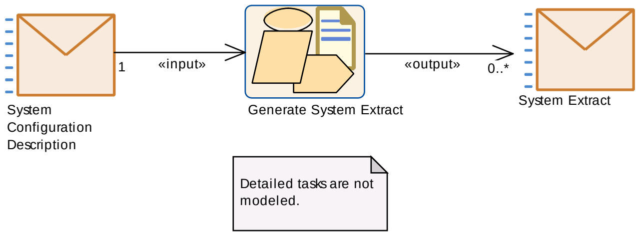

Figure 2.30: Generate the System Extract

The detailed tasks of Generate System Extract are not modeled since they are considered as trivial - it just means to reduce the content of the input description to the subsystem in question.

Activity	**Generate System Extract**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::System::Generate System Extract		
Brief Description			
Description	Generate for further development, a System Extract which represents the description of a part of the system (sub-system). This allows a start of work on ECU's even if the system is not completely described.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	System Configuration Description	1	
Produces	System Extract	0..*	

Table 2.21: Generate System Extract

2.5.4 Create ECU System Description

2.5.4.1 Purpose

Based on a System Extract, this Activity creates ECU System Descriptions which are refined during the design of the sub-system.

2.5.4.2 Description

[TR_METH_01125] Create ECU System Description activity ⌈ Based on the delivered System Extract, the receiving organization creates one or more ECU Descriptions. The ECU Descriptions are used for designing the sub-system artifacts (See activity Design Sub-System). ⌋*(RS_METH_00002, RS_METH_00005, RS_METH_00077)*

From the methodological point of view there are two choices for creating the ECU System Description.

[TR_METH_01126] Using the System Extract as the structural basis for the ECU development ⌈ The System Extract is taken as the structural basis for the ECU development. In this case the System Extract becomes an ECU System Description. ⌋*(RS_METH_00002, RS_METH_00005, RS_METH_00077)*

[TR_METH_01127] Creating a new structure for the ECU development ⌈ A new structure is created as a basis for the ECU development. The newly created ECU System Description is mapped to the initial System Extract. For this purpose the task Define System View Mapping creates the initial System View Mapping artifact which is refined during the sub-system design. ⌋*(RS_METH_00002, RS_METH_00005, RS_METH_00077)*

2.5.4.3 Workflow

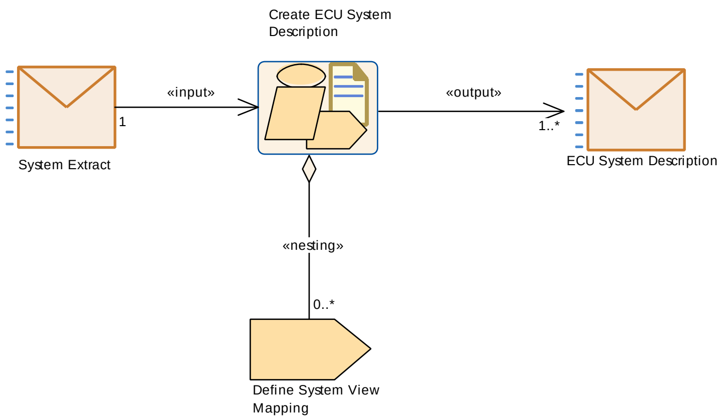

Figure 2.31: Create ECU System Description

Activity	**Create ECU System Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::System::Create ECU System Description		
Brief Description			
Description	During the Develop Sub-System activity the supplier refines the received System Extract so that valid ECU Extracts can be generated. The refinement of the System Extract is done using the ECU System Description. Therefore, this activity creates one or more ECU System Descriptions based on the System Extract. The sub-system artifacts are designed in the ECU System Description during the activity "Design Sub-System". From the methodological point of view there are two choices for creating the ECU System Description. 1) The System Extract is taken as the structural basis for the ECU development. In this case the System Extract becomes an ECU System Description. 2) A new structure is created as a basis for the ECU development. The newly created ECU System Description is mapped to the initial System Extract. For this purpose the task "Define System View Mapping" is performed.		
Relation Type	**Related Element**	**Mul.**	**Note**
Consumes	System Extract	1	
Produces	ECU System Description	1..*	
Aggregates	Define System View Mapping	0..*	

Table 2.22: Create ECU System Description

2.5.5 Design Sub-System

2.5.5.1 Purpose

This `Activity` details a given `ECU System Description` (previously created from the delivered `System Extract`) with additional ECUs and networks.

2.5.5.2 Description

[TR_METH_01075] `Design Sub-System` activity ⌈ Based on the `ECU System Description`, the description of a sub-system is defined. ⌋*(RS_METH_00002, RS_METH_00005)*

[TR_METH_01076] Collaboration between different organizations ⌈ Additionally, the software component structure of the `System Extract`s, delivered by the primary organization can be transformed into a different structure by the receiving organization

(ECU System Description). In this case the System Extract of the primary organization can be considered as a requirement and the sub-system of the receiving organization can be seen as a solution which has to fulfill the delivered requirements. Thus here again a mapping activity can be defined which maps the newly introduced solution sub-system to the provided requirement sub-system from the primary organization. ⌋*(RS_METH_00002, RS_METH_00005, RS_METH_00077)*

[TR_METH_01077] Transformation changes during the Design Sub-System activity ⌈ During this transformation the hierarchical SWC-structure can be changed, some SWCs can be replaced by other SWCs, some can remain in the resulting view. ⌋*(RS_METH_00002, RS_METH_00005)*

[TR_METH_01078] Mapping of different views ⌈ The different views are mapped by the System View Mapping. ⌋*(RS_METH_00002, RS_METH_00005)*

Typical use-cases for this transformation steps are:

- **[TR_METH_01079] Use Case: Substitution of existing components** ⌈ The secondary organization has an existing software architecture. By software sharing some of the existing components are substituted by the delivered software components. ⌋*(RS_METH_00002, RS_METH_00005, RS_METH_00077, RS_METH_00079)*
- **[TR_METH_01080] Use Case: Mapping of requirements to the solution** ⌈ The secondary organization develops one ECU for different primary organizations and therefore has to map the requirements of different primary organizations to its solution. ⌋*(RS_METH_00002, RS_METH_00005, RS_METH_00077, RS_METH_00079)*
- **[TR_METH_01081] Use Case: Reorganization of the software structure** ⌈ The primary organization delivers a sub-system description which defines one ECU. The secondary organization decides to use two ECUs. Therefore the software structure has to be reorganized by the second organization. ⌋*(RS_METH_00002, RS_METH_00005, RS_METH_00077, RS_METH_00079)*
- **[TR_METH_01082] Use Case: Description of changes between different versions of System Descriptions** ⌈ Additionally the mapping can be used to formally describe changes between different versions of System Descriptions. ⌋*(RS_METH_00002, RS_METH_00005, RS_METH_00077, RS_METH_00079)*

Finally all Atomic Software Components in the resulting sub-system scope are included in this sub-system description.

2.5.5.3 Workflow

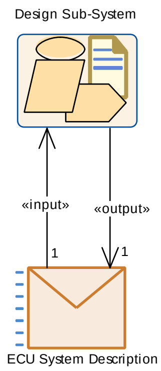

Figure 2.32: Overview: Design Sub-System

Note that the `ECU System Description` appears as input and output of this Activity because it is refined.

As the detailed work flow for this Activity uses the same elements from the methodology library as the one described in 2.5.2.3, the breakdown into tasks is not modeled here.

Activity	**Design Sub-System**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::System::Design Sub-System		
Brief Description			
Description	Design the sub-system artifacts based on an ECU System Description which was previously created from the delivered ECU Extract. It consists of the same tasks as the activity Design System. The description must be completed down to the ECU level, so that valid ECU extracts can be generated.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	ECU System Description	1	System Extract as generated from the outer system.
Produces	ECU System Description	1	System Extract refined during design of the corresponding sub-system with elements needed to generate ECU Extract(s).

Table 2.23: Design Sub-System

2.5.6 Generate ECU Extract

2.5.6.1 Purpose

This `Activity` provides an extract of the System description for setting up an ECU Configuration for specific ECU.

2.5.6.2 Description

Generate an `ECU Extract` basis for setting up the ECU configuration and further development on ECU level.

2.5.6.3 Workflow

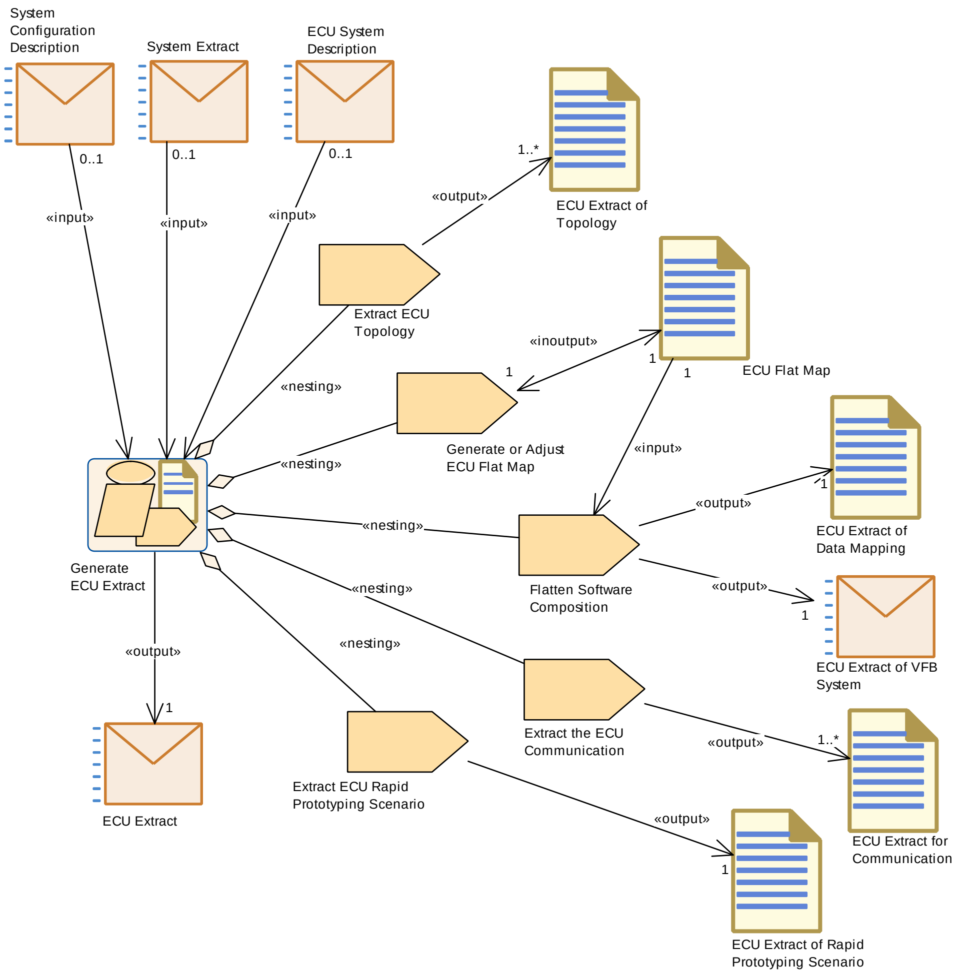

Figure 2.33: Generate the ECU Extract

Activity	**Generate ECU Extract**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::System::Generate Ecu Extract		
Brief Description	Generate the ECU Extract out of the System Description in order to be delivered for integration for further development on ECU level.		
Description	Generate the ECU extract which is a basis for setting up the ECU configuration and further development on ECU level. It can be generated either from a full system (System Configuration Description), a System Extract or a ECU System Description.		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	ECU System Description	0..1	
Consumes	System Configuration Description	0..1	
Consumes	System Extract	0..1	
Produces	ECU Extract	1	
Aggregates	Extract ECU Rapid Prototyping Scenario	1	
Aggregates	Extract ECU Topology	1	
Aggregates	Extract the ECU Communication	1	
Aggregates	Flatten Software Composition	1	
Aggregates	Generate or Adjust ECU Flat Map	1	
Predecessor	Define Rapid Prototyping Scenario	1	

Table 2.24: Generate ECU Extract

2.5.7 Design Custom Transformer

2.5.7.1 Purpose

This `Activity` specifies the functional aspects of a transformation technology used for the serialization of selected system signals.

2.5.7.2 Description

Transformer enable AUTOSAR systems to use a data transformation mechanism to linearize and transform data. They can be concatenated to transformer chains and are executed by the RTE for inter-ECU communication which is configured to be transformed.

The transformation technology (which transformer should be used for which communication) is defined in the context of the `Design Communication` activity (task `Define Transformation Technology`). For the transformation of communication data standardized transformers (e.g. SOME/IP transformer) or custom transformers can be used.

[TR_METH_01130] **`Design Custom Transformer`** **activity** ⌈ In case of custom transformers the `Design Custom Transformer` activity has to be performed to define the functional specification of the custom transformation mechanism (`Custom Transformer Specification`) and the corresponding configuration parameters (BSW Module Vendor- Specific Configuration Parameter Definition). The Design Custom Transformer activity is done during the Develop System activity because it produces a definition what a transformer does and therefore significantly affects the corresponding communication. ⌋*(RS_METH_00005, RS_METH_00077)*

The specified transformer is then implemented (Develop Basic Software) and can be used in the Design Communication activity. There, inter-ECU communication can be marked for being transformed.

[TR_METH_01131] Output of Design Custom Transformer activity ⌈ The Design Custom Transformer activity shall result in a set of complete and unambiguous written Custom Transformer Specifications and the corresponding BSW Module Vendor- Specific Configuration Parameter Definition. A specification of a specific transformer shall adhere to [8, SWS BSW General] and [9, ASWS Transformer General].

A specification of a transformer shall contain:

- Functional specification of the transformer. See [9, ASWS Transformer General] for details. The most important issue are:
  - Specification of the transformers output
  - Transformer class
  - Transformer errors
- Definition of Development Errors, Production Errors and Extended Production Errors.
- Transformer APIs
- Extension of the transformer EcuC if necessary for the specific transformer

⌋*(RS_METH_00077)*

2.5.7.3 Workflow

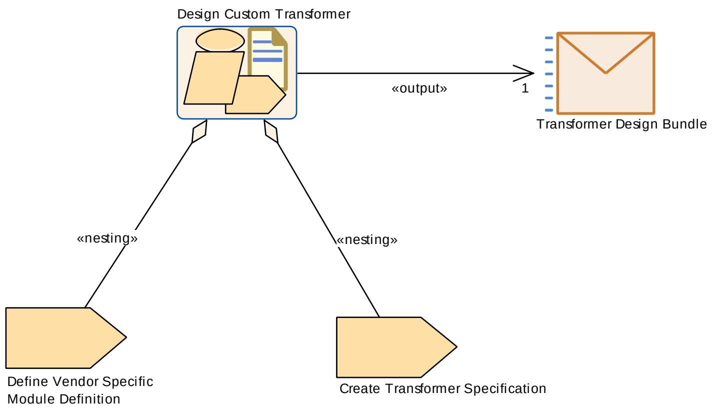

Figure 2.34: Design Custom Transformer activity

Activity	**Design Custom Transformer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::System::Design Custom Transformer		
Brief Description			
Description	In this activity the functional specification of the custom transformer module is created and the corresponding parameter definition is specified. The creation of the functional specification of the Transformer can be seen as a part of the communication design. This activity is performed only if a custom transformer for the communication is required.		
Relation Type	***Related Element***	***Mul.***	***Note***
Produces	Transformer Design Bundle	1	
Aggregates	Create Transformer Specification	1	
Aggregates	Define Vendor Specific Module Definition	1	

Table 2.25: Design Custom Transformer

2.5.8 Define System Safety Information

2.5.8.1 Purpose

This Activity allows specifying safety information at system level.

2.5.8.2 Description

In this activity, the safety information at system or sub-system level is defined. Obviously, the safety information defined in previous development stages is detailed. (For detailed tasks see chapter 2.14).

2.5.8.3 Workflow

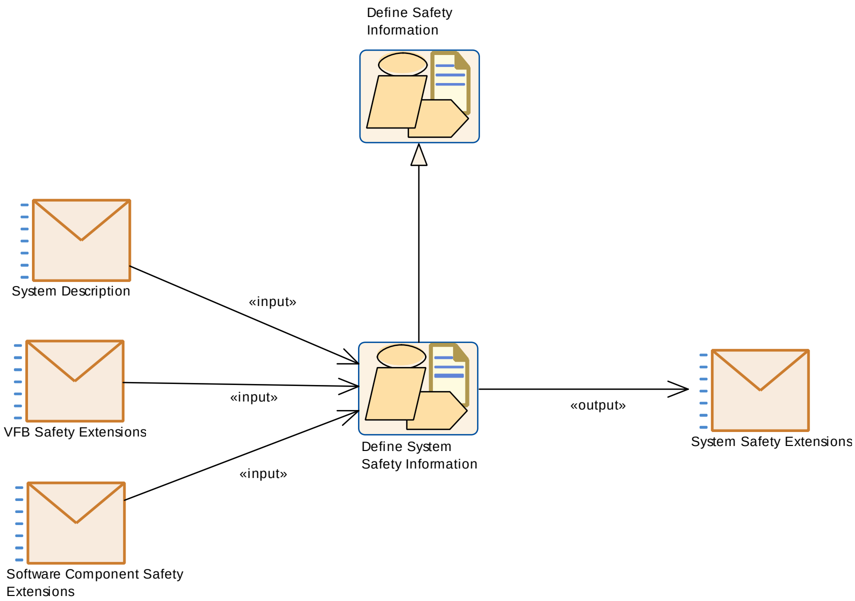

Figure 2.35: Define System Safety Information

Activity	**Define System Safety Information**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::System::Develop System		
Brief Description	Defines all required safety information at system level.		
Description	In this activity, the safety information at system level is defined. The safety information can be refined or completed in further development phases.		
Extends	Define Safety Information		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Software Component Safety Extensions	1	
Consumes	System Description	1	
Consumes	VFB Safety Extensions	1	
Produces	System Safety Extensions	1	

Table 2.26: Define System Safety Information

2.6 Develop Basic Software

2.6.1 Overview

2.6.1.1 Purpose

This `Activity` provides an overall use case how to the develop AUTOSAR Basic Software.

2.6.1.2 Description

2.6.1.3 Workflow

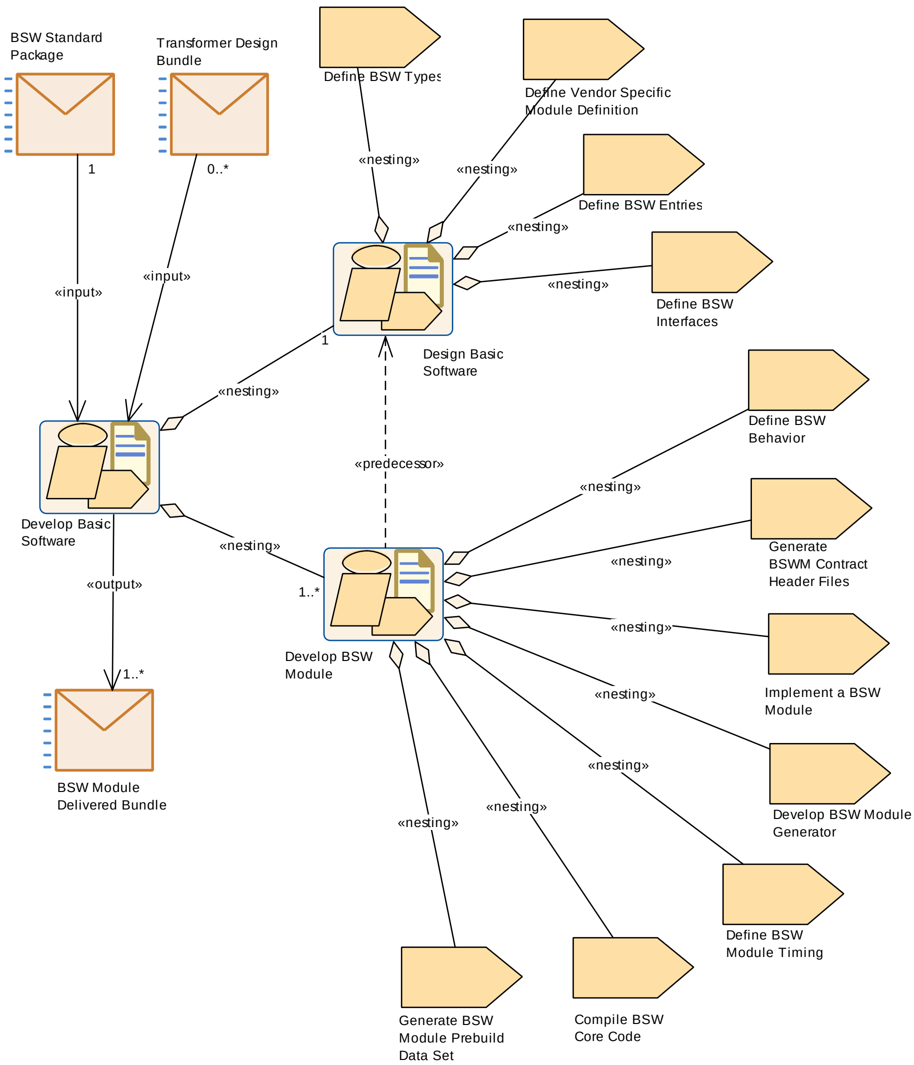

Figure 2.36: Nesting relationship: Develop Basic Software

Activity	**Develop Basic Software**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::BSW::develop_bsw		
Brief Description			
Description	Describes the overall activities to develop Basic Software, starting from the design down to delivery of modules. In case of custom transformer module development, the Transformer Design Bundle containing the functional specification and the parameter definition is taken as a basis for all required activities.		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	BSW Standard Package	1	
Consumes	Diagnostic System Extract	0..*	
Consumes	Transformer Design Bundle	0..*	
Produces	BSW Module Delivered Bundle	1..*	
Produces	Diagnostic System Extract	0..*	
Aggregates	Design Basic Software	1	
Aggregates	Develop BSW Module	1..*	

Table 2.27: Develop Basic Software

It consists of two parts:

- `Design Basic Software`
- `Develop BSW Module`

2.6.2 Design BSW

2.6.2.1 Purpose

This `Activity` provides a rough outline for the Basic Software design for an ECU or a set of ECUs.

2.6.2.2 Description

[TR_METH_01083] **`Design Basic Software`** **activity** ⌈ Design the Basic Software for an ECU or a set of ECUs. This shall result in a set of complete and unambiguous `Basic Software Module Description`s. ⌋*(RS_METH_00003, RS_METH_00004)*

Note that existing descriptions, especially standardized ones, can be reused, eventually setting only optional elements or user specific extension.

[TR_METH_01084] Separation of design and development of basic software ⌈ This `Activity` is conceptually separated from `Develop BSW Module`, because it might be performed by a `Basic Software Designer` responsible for the complete Basic Software Design on a given ECU, which may be different in general from the Basic Software Module Developer who develops or delivers the single modules. ⌋ *(RS_METH_00003, RS_METH_00018)*

2.6.2.3 Workflow

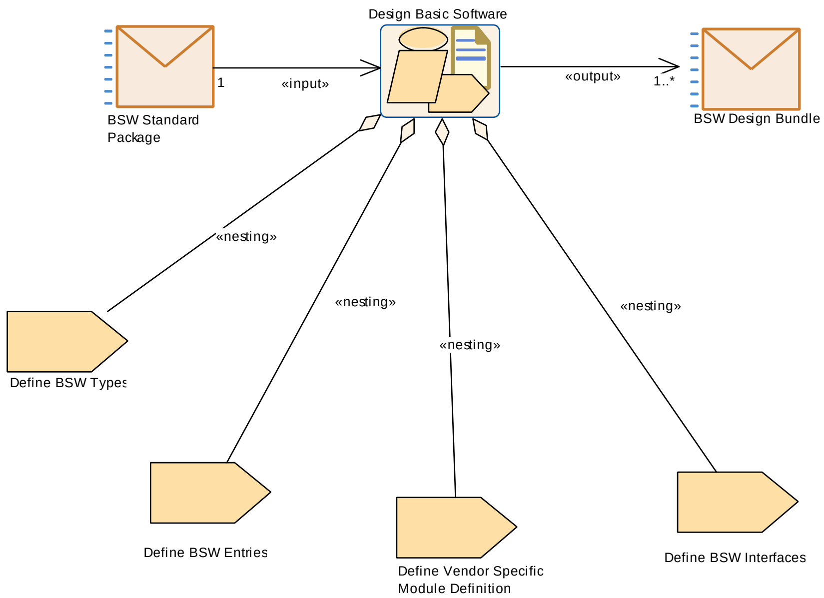

Figure 2.37: Nesting Relationship : Design Basic Software

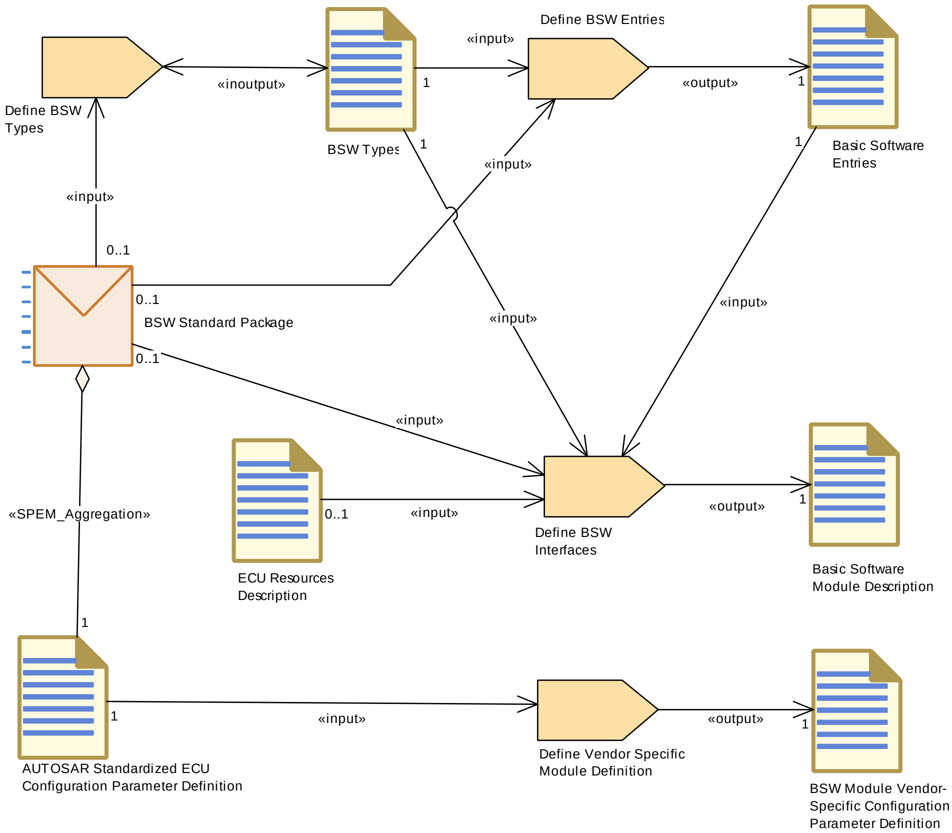

Figure 2.38: Design Basic Software

Activity	**Design Basic Software**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::BSW::develop_bsw		
Brief Description	Design the Basic Software for an ECU or a set of ECUs.		
Description	Design the Basic Software for an ECU or a set of ECUs. This shall result in a set of complete and unambiguous Basic Software Module Description. Note that existing descriptions, especially standardized ones, can be reused, eventually setting only optional elements or user specific extension. This activity is conceptually separated from the activity Develop Basic Software Module, because it might be performed by a Basic Software Designer responsible for the complete Basic Software Design on a given ECU, which may be different (in general) from the Basic Software Module Developer who develops and/or delivers the single modules.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	BSW Standard Package	1	
Produces	BSW Design Bundle	1..*	
Aggregates	Define BSW Entries	1	
Aggregates	Define BSW Interfaces	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Define BSW Types	1	
Aggregates	Define Vendor Specific Module Definition	1	

Table 2.28: Design Basic Software

2.6.3 Develop BSW Module

2.6.3.1 Purpose

This `Activity` provides a rough outline for a single Basic Software module or cluster development prior to an ECU integration.

2.6.3.2 Description

[TR_METH_01085] **`Develop BSW Module`** **activity** ⌈ To develop the core code (i.e. the code not generated during integration) of a single BSW module or cluster prior to ECU integration. This `Activity` focuses on the tasks which are common for most BSW modules. It is not valid for those modules (RTE, BSW Scheduler) which are completely generated at integration time. ⌋*(RS_METH_00003, RS_METH_00006, RS_METH_00038)*

2.6.3.3 Workflow

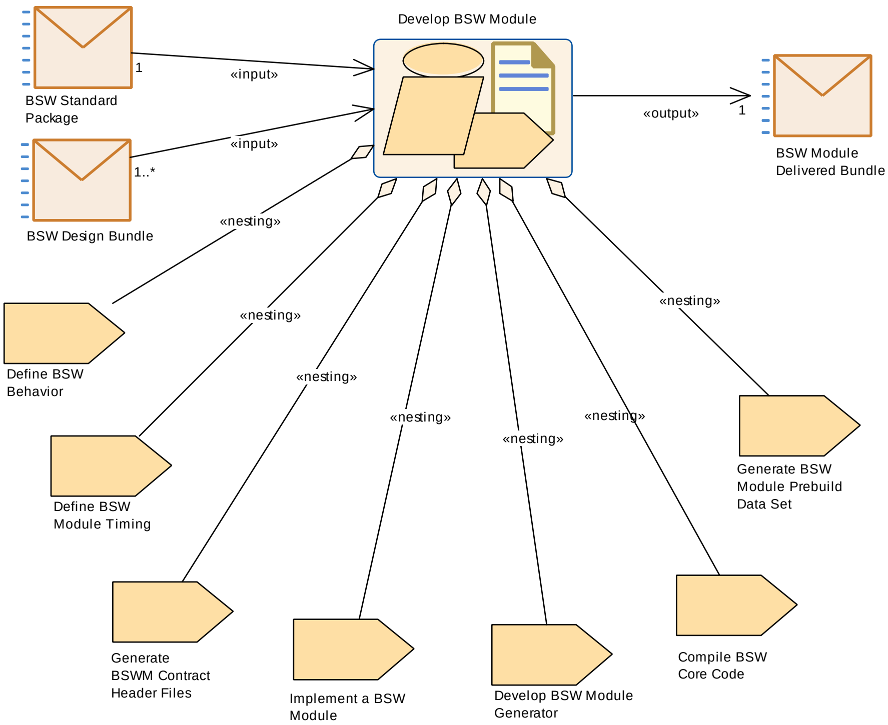

Figure 2.39: Nesting relationship : Develop Basic Software Module

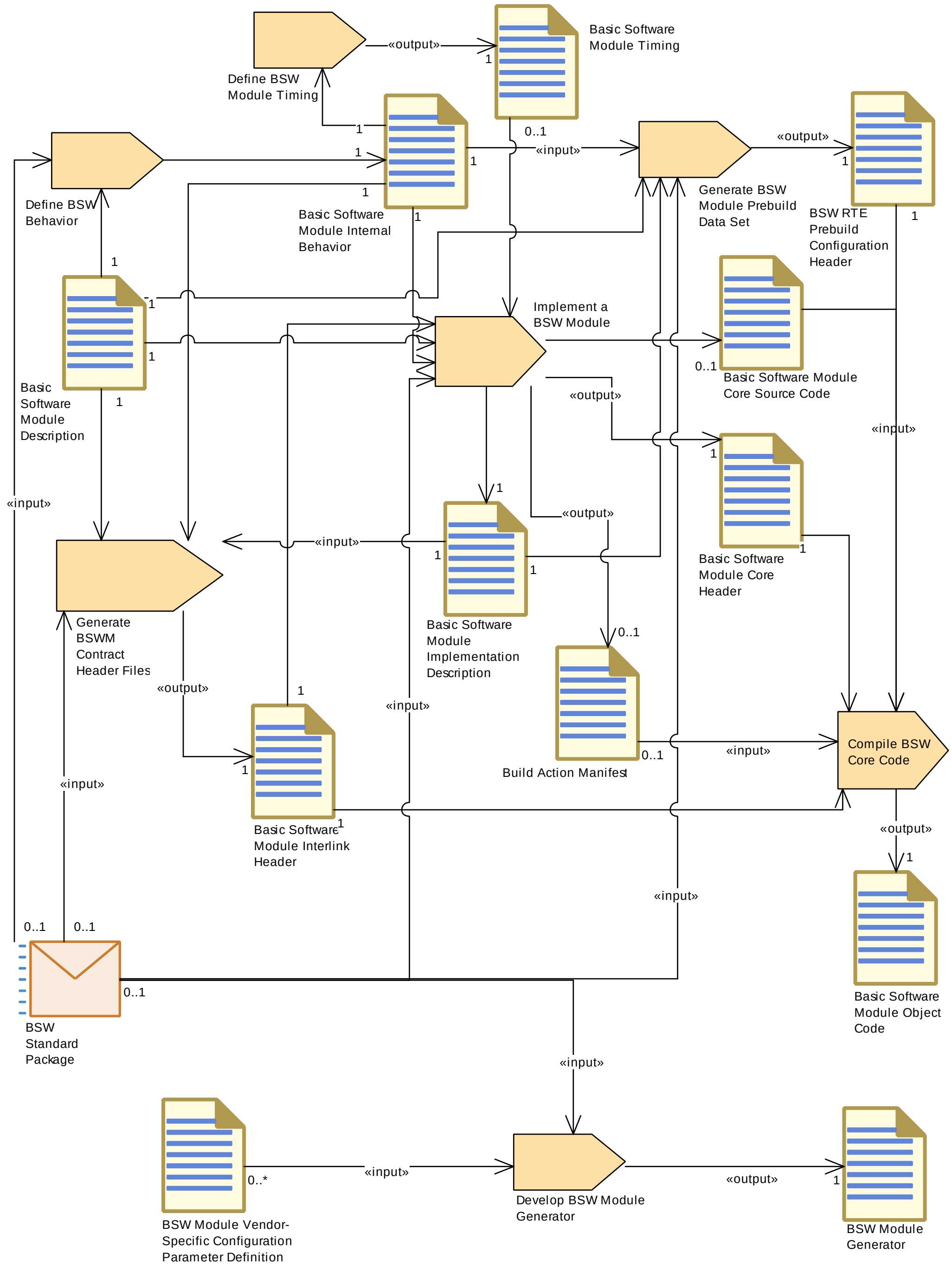

Figure 2.40: Develop Basic Software Module

Activity	**Develop BSW Module**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::BSW::develop_bsw		
Brief Description	Develop a single BSW module or cluster prior to ECU integration.		
Description	Develop a single BSW module or cluster prior to ECU integration. To develop the core code (i.e. the code not generated during integration) of a single BSW module or cluster prior to ECU integration including vendor specific configuration parameters and module generators. This activity focuses on the tasks which are common for most BSW modules. It is not valid for those modules (RTE, BSW Scheduler) which are completely generated at integration time.		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	BSW Design Bundle	1..*	
Consumes	BSW Standard Package	1	
Produces	BSW Module Delivered Bundle	1	
Aggregates	Compile BSW Core Code	1	
Aggregates	Define BSW Behavior	1	
Aggregates	Define BSW Module Timing	1	
Aggregates	Develop BSW Module Generator	1	
Aggregates	Generate BSW Module Prebuild Data Set	1	
Aggregates	Generate BSWM Contract Header Files	1	
Aggregates	Implement a BSW Module	1	
Predecessor	Design Basic Software	1	
Predecessor	Design Basic Software	1	

Table 2.29: Develop BSW Module

2.7 Integrate Software for ECU

2.7.1 Description

In this chapter, the integration for an AUTOSAR ECU is described. In the AUTOSAR sense an ECU means a microcontroller plus peripherals and the according software/-configuration. Therefore, each microcontroller requires its own ECU Configuration.

[TR_METH_01086] Integrate Software for ECU activity ⌈ The main activities include configuring and/or generating the BSW modules (including the RTE) and building the executable. The BSW configuration can be done during different steps of development. The detailed use cases for these different ways of configuration are introduced later in the chapter, thanks to the Configuration Classes definition :

- Pre-compile time
- Link time
- Post-build time

⌋*(RS_METH_00004, RS_METH_00062)*

2.7.2 Overview

2.7.2.1 Purpose

This Activity is showing the high level view how to integrate AUTOSAR Software for an ECU.

2.7.2.2 Description

[TR_METH_01087] Scope of Integrate Software for ECU activity ⌈ The development of an AUTOSAR ECU consists of four main activities:

- Prepare ECU Configuration
- Configure BSW and RTE
- Generate BSW and RTE
- Build Executable

In addition, the optional activity Model ECU Timing is shown. The ECU timing model depends on ECU configuration details (BSW and RTE), but the results shall help to optimize the configuration in an iterative approach. ⌋*(RS_METH_00005, RS_METH_00003, RS_METH_00004, RS_METH_00002, RS_METH_00006)*

The ECU configuration plays a significant role during the integration of the software for an ECU. The relevant workflow is depicted in figure 2.42[1]. All three activities (Prepare ECU Configuration, Configure BSW and RTE, Generate BSW and RTE) use the work product ECU Configuration Values which contains (i.e. references) all the configuration information for all BSW modules on the ECU. In order to better understand the three different activities an introduction to configuration classes is given in chapter 2.7.9.

[1]In order to be more comprehensible, this figure hides some outputs of the activity Generate BSW and RTE. For more details see the outputs of all aggregated tasks.

One can measure resources used by the various BSW modules and applications and save that information within the Basic Software Module Implementation Description or Atomic Software Component Implementation.

One can also generate an A2L File processing the Generate A2L task at this point.

2.7.2.2.1 Inputs to ECU Configuration

[TR_METH_01114] Input sources for ECU Configuration ⌈ ECU Configuration has two input sources (see figure 2.42). First of all, all configuration that must be agreed across ECUs is defined in the System Configuration, which results in a System Configuration Description (and the resulting ECU Extract for the individual ECUs).

Secondly, the ECU BSW is built using BSW modules. The specifics of these module implementation are defined in the BSW Module descriptions covered by the BSW Module Delivered Bundle. ⌋*(RS_METH_00003, RS_METH_00004, RS_METH_00005, RS_METH_00006)*

The latter is described in [10] in more detail. The concept of the ECU Extract is depicted below:

ECU Extract

ECU Configuration can only be started once a plausible System Configuration Description and the corresponding ECU Extract has been generated (see figure 2.42). Details on the System Configuration Description can be found in [7].

The System Configuration Description contains all relevant system-wide configuration, such as

- ECUs present in the system
- Communication systems interconnecting those ECUs and their configuration
- Communication matrices (frames sent and received) for those communication systems
- Definition of Software Components with their ports and interfaces and connections (defined in the SWC Description and referenced in the System Configuration Description).
- Mapping of SWCs to ECUs

The ECU Extract is a description in the same format as the System Configuration Description, but with only those elements included that are relevant for the configuration of one specific ECU.

2.7.2.2.2 ECU Configuration Value description

The ECU Extract only defines the configuration elements that must be agreed between ECUs. In order to generate a working executable that runs on the ECU, much more configuration information must be provided.

The remaining part of the configuration is about configuring all BSW modules within the ECU. Typical BSW modules within an ECU can be: RTE, Com, Can, OS, NVRAM etc. There are also dependencies between BSW modules to consider when configuring the ECU.

When the configuration is done, the generation of configuration data takes place. I.e. there are both configuration editors and configuration generators involved in the process.

In order to obtain consistency within the overall configuration of the ECU, AUTOSAR has defined a single format, the ECU Configuration Value description to be used for all BSW modules within an ECU. Both configuration editors and configuration generators are working toward ECU Configuration Value descriptions. In the AUTOSAR Methodology the ECU Configuration Value descriptions is represented by the artifact ECU Configuration Values.

[TR_METH_01116] ECU Configuration Value description contains the configuration of all BSW modules in a single ECU ⌈ This one description (ECU Configuration Values) collects the complete configuration of BSW modules in a single ECU. Each module generator may then extract the subset of configuration data it needs from that single format. ⌋*(RS_METH_00004)*

2.7.2.3 Workflow

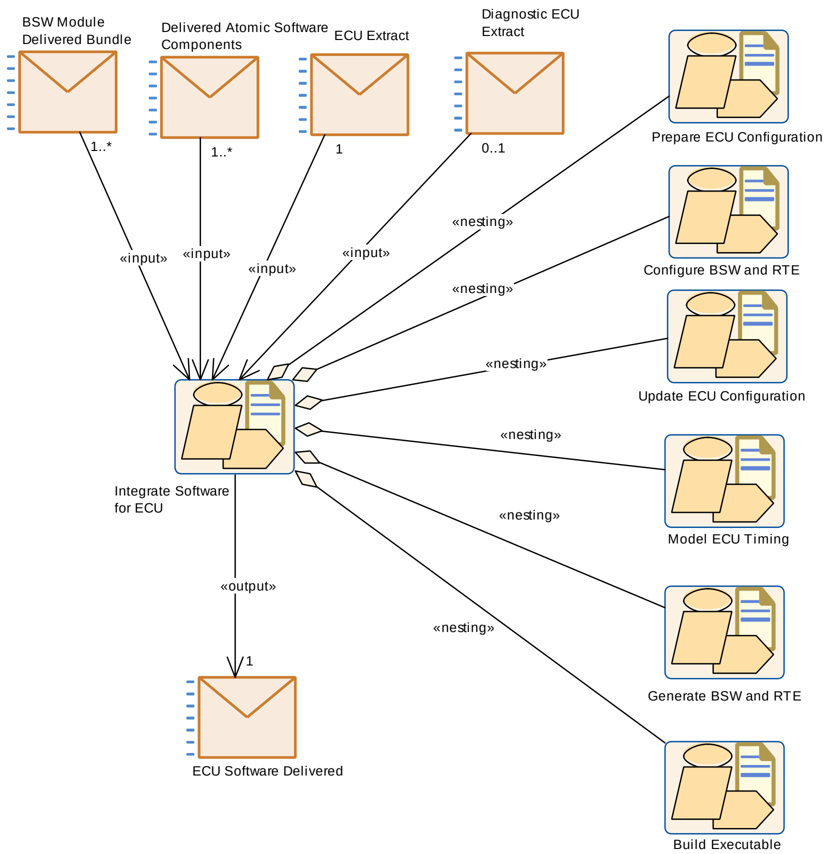

Figure 2.41: Integrate Software for ECU Overview

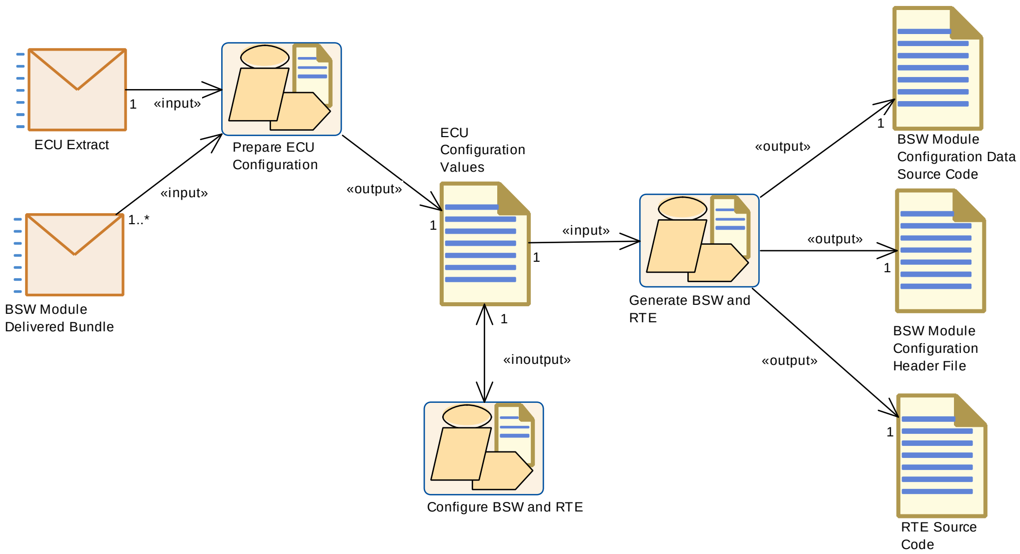

Figure 2.42: ECU Configuration Overview

Activity	**Integrate Software for ECU**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::ECU:: Integrate Software for ECU		
Brief Description			
Description	This activity contains all typical sub-activities required to integrate the software components and modules on an AUTOSAR ECU. ECU in this context means processor, so if an electronic control unit consists of several processors, one "ECU Delivered" will be needed for each processor.		
Relation Type	**Related Element**	**Mul.**	**Note**
Consumes	BSW Module Delivered Bundle	1..*	
Consumes	Delivered Atomic Software Components	1..*	
Consumes	Diagnostic ECU Extract	0..1	complete DE:
Consumes	ECU Extract	1	
Produces	ECU Software Delivered	1	
Aggregates	Build Executable	1	
Aggregates	Configure BSW and RTE	1	
Aggregates	Generate BSW and RTE	1	
Aggregates	Model ECU Timing	1	
Aggregates	Prepare ECU Configuration	1	
Aggregates	Update ECU Configuration	1	

Table 2.30: Integrate Software for ECU

2.7.3 Prepare ECU Configuration

2.7.3.1 Description

[TR_METH_01088] **`Prepare ECU Configuration`** **activity** ⌈ During the `Prepare ECU Configuration` activity, the information available in `ECU Extract` for the specific ECU is extended by implementing the `Service Needs` required by the Software Components and BSW Modules and by including their initial configurations as provided in the `BSW Module Preconfigured Configuration` or `BSW Module Recommended Configuration`. The result of this activity is the base `ECU Configuration`.

In addition, the `BSW Module Vendor- Specific Configuration Parameter Definition`, which defines all possible configuration parameters and their structure, is incorporated into the `ECU Configuration`. This is necessary because the output ECU Configuration has a flexible structure which does not define a fixed number of configuration parameters a priori. ⌋*(RS_METH_00005, RS_METH_00003, RS_METH_00004, RS_METH_00002)*

[TR_METH_01117] BSW implementation shall be chosen for each BSW module that is present in the ECU ⌈ For each BSW module that shall be present in the ECU, the implementation must be chosen. This is done by referencing the BSW Module description delivered with the BSW module (BSW Module Delivered Bundle). ⌋ *(RS_METH_00003, RS_METH_00004)*

The rules that must be followed when building the base ECU Configuration Value description are available in [11] chapter 4.2.

2.7.3.2 Workflow

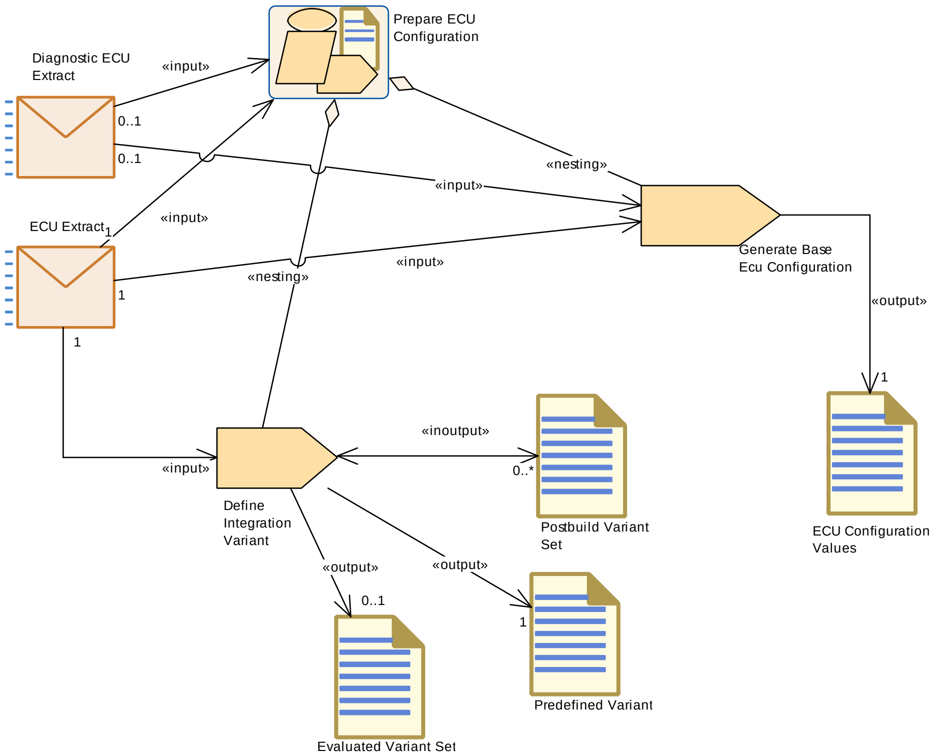

Figure 2.43: Prepare ECU Configuration

Activity	**Prepare ECU Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::ECU::Integrate Software for ECU		
Brief Description			
Description	Initial actions required to create the initial ECU Configuration.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	BSW Module Delivered Bundle	1..*	

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Diagnostic ECU Extract	0..1	
Consumes	ECU Extract	1	
Produces	ECU Configuration Values	1	
Aggregates	Define Integration Variant	1	
Aggregates	Generate Base Ecu Configuration	1	
Predecessor	Refine Rapid Prototyping Scenario	1	

Table 2.31: Prepare ECU Configuration

2.7.4 Configure BSW and RTE

2.7.4.1 Description

[TR_METH_01089] **`Configure BSW and RTE` activity** ⌈ Once there is a base `ECU Configuration`, the complete configuration can be performed. This is mainly editing work on the `ECU Configuration` which is typically supported by an editing tool. In practice this will require iterations and/or parallel work to configure the RTE and all participating BSW modules. ⌋*(RS_METH_00003, RS_METH_00004, RS_METH_00020)*

The methodology does not prescribe a certain order of these configuration steps. The `ECU Configuration` description (e.g. `ECU Configuration Values`) which was produced by one activity can be read by another activity (e.g. `Configure RTE` generates a description and `Configure Com` reads this). Usually the configuration activities for the BSW modules (e.g. COM and OS) read and write the `ECU Configuration`.

[TR_METH_01090] **`Configure RTE` task** ⌈ The `Configure RTE` task is more complex as this additionally needs all the `Atomic Software Component Implementation`s required for that ECU. Whenever these change, e.g. because software components have been moved to or from other ECUs, or simply another implementation of a software component has been selected, the `Configure RTE` task must be repeated as well. ⌋*(RS_METH_00005, RS_METH_00003, RS_METH_00004, RS_METH_00002)*

[TR_METH_01091] **`Configure Debug` task** ⌈ Finally the `Configure Debug` task can be completed. Since this configuration depends on previous configuration results, it should be completed last. ⌋*(RS_METH_00005, RS_METH_00003, RS_METH_00004, RS_METH_00002)*

2.7.4.2 Workflow

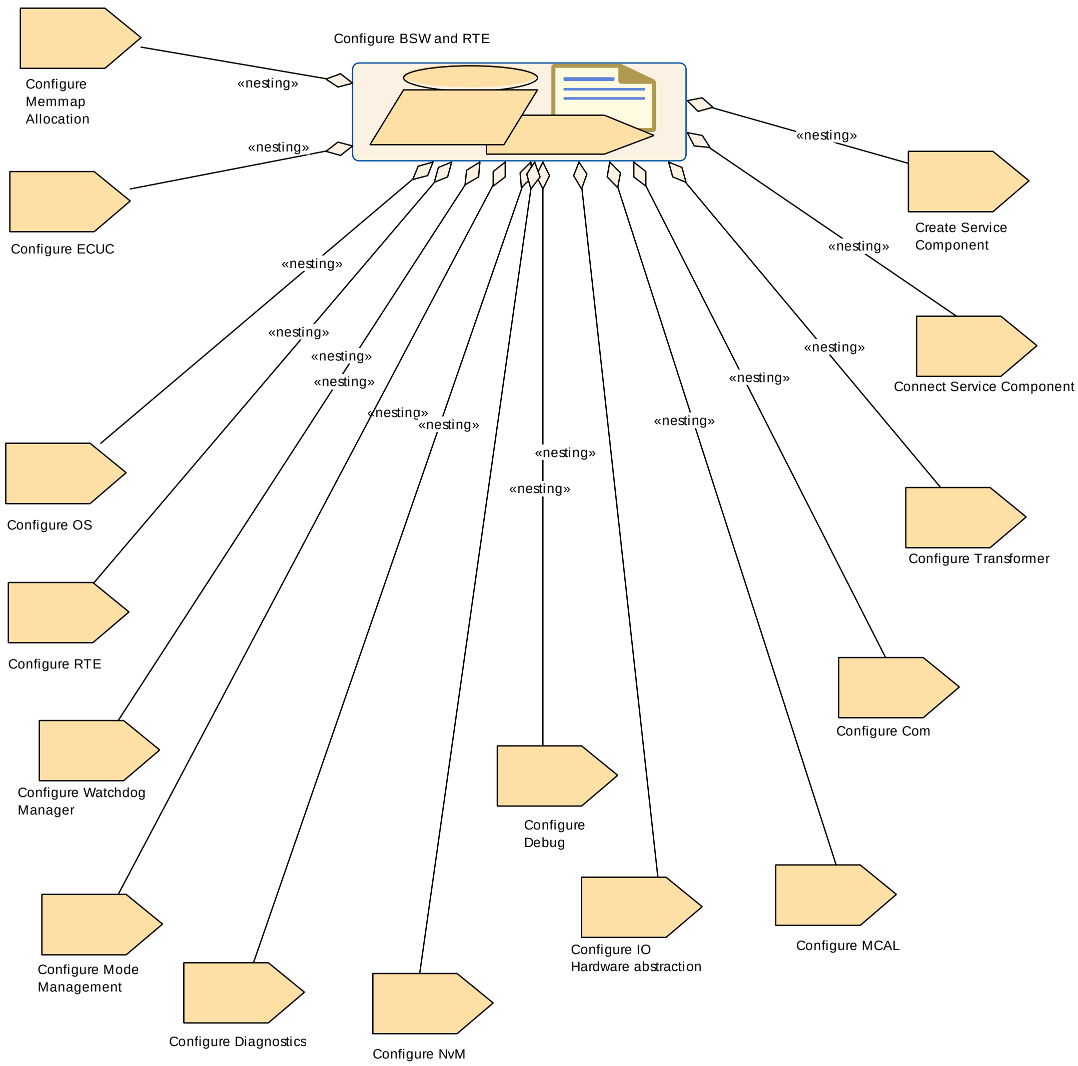

Figure 2.44: Configure BSW and RTE

Activity	**Configure BSW and RTE**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::ECU::Integrate Software for ECU		
Brief Description			
Description	All the tasks used to configure the Basic Software Modules on an ECU.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Configure Com	1	
Aggregates	Configure Debug	1	
Aggregates	Configure Diagnostics	1	
Aggregates	Configure ECUC	1	
Aggregates	Configure IO Hardware abstraction	1	
Aggregates	Configure MCAL	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Configure Memmap Allocation	1	
Aggregates	Configure Mode Management	1	
Aggregates	Configure NvM	1	Since the configuration of the DEM usually has impact on the data to be stored in NvM, the task Configure Diagnostics is assumed to precede the task Configure NvM.
Aggregates	Configure OS	1	
Aggregates	Configure RTE	1	
Aggregates	Configure Transformer	1	
Aggregates	Configure Watchdog Manager	1	
Aggregates	Connect Service Component	1	
Aggregates	Create Service Component	1	
Predecessor	Prepare ECU Configuration	1	
In/out	ECU Configuration Values	1	

Table 2.32: Configure BSW and RTE

2.7.5 Update ECU Configuration

2.7.5.1 Description

In a post-build scenario, there are two loadable files generated in the end - one of them containing the application software, basic software and the pre-compile and link time configuration of the basic software (referred to as `ECU Executable`) and the other one containing only the post-build time configuration of the basic software (`BSW Module Configuration Data Loadable to ECU Memory`). These two loadable files represent the initial configuration. This initial configuration can be updated in post-build time by generating two new loadable files. In this update, the `ECU Executable` is not modified.

[TR_METH_01151] **`Update ECU Configuration`** **activity** ⌈ The update of the `BSW Module Configuration Data Loadable to ECU Memory` is usually done by importing the updated EcuExtract containing the needed post-build updates to the ECU configuration tool which already contains the initial ECU configuration. Based on these updates in the EcuExtract and everything else from the initial ECU configuration, an updated ECU configuration shall be created (therefore we have both input and output relations between the `ECU Configuration Values` and the `Update ECU Configuration` activity). ⌋*(RS_METH_00004, RS_METH_00062)*

2.7.5.2 Workflow

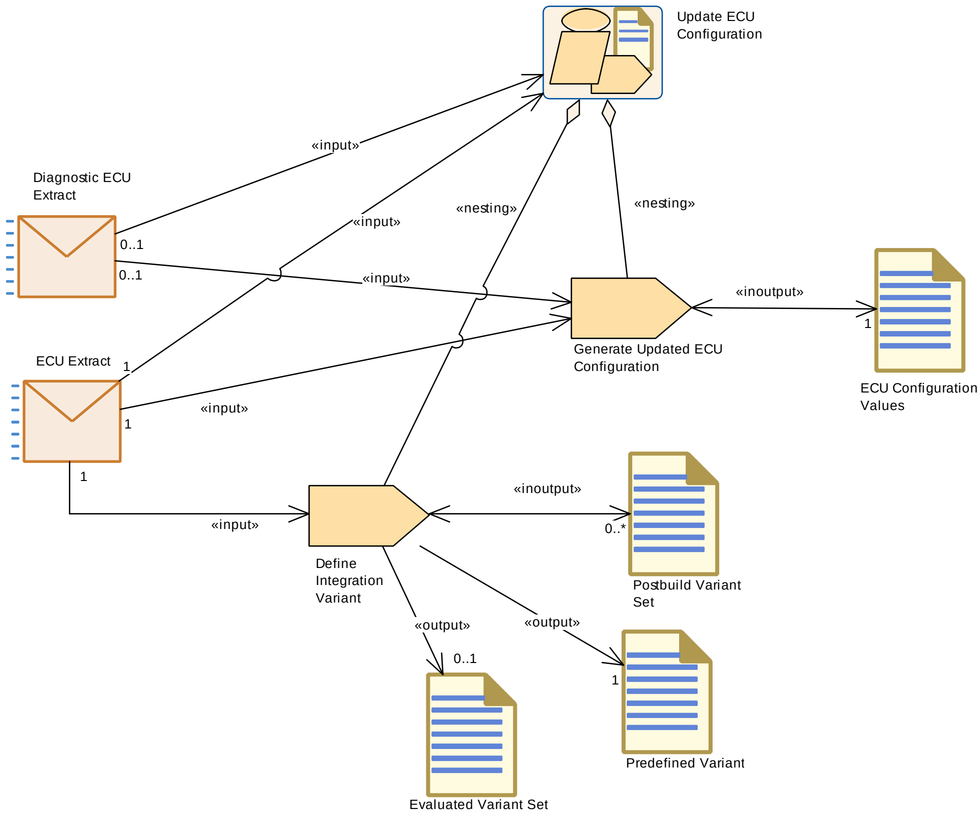

Figure 2.45: Update ECU Configuration

Activity	**Update ECU Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::ECU::Integrate Software for ECU		
Brief Description	Tasks required to create the updated ECU Configuration.		
Description	Tasks required to create the updated ECU Configuration.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Diagnostic ECU Extract	0..1	
Consumes	ECU Extract	1	
Aggregates	Define Integration Variant	1	
Aggregates	Generate Updated ECU Configuration	1	

Table 2.33: Update ECU Configuration

2.7.6 Model ECU Timing

2.7.6.1 Workflow

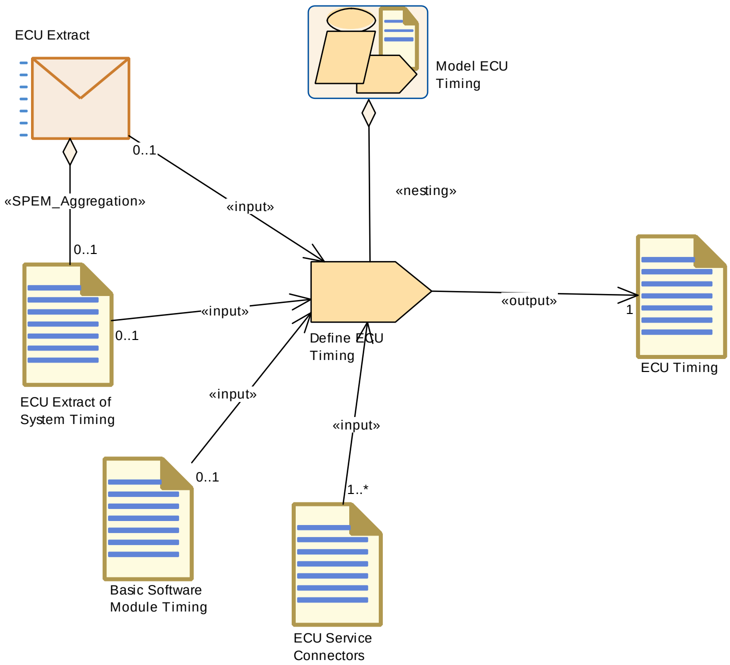

Figure 2.46: Model ECU Timing

Activity	**Model ECU Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::ECU:: Integrate Software for ECU		
Brief Description			
Description	ECU timing model depends on ECU configuration data (BSW and RTE) but the result of the ECU timing model shall help to optimize ECU configuration. The relation between "Configure BSW and RTE" and "Model ECU Timing" must be seen as an iterative work.		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Define ECU Timing	1	
Predecessor	Configure BSW and RTE	1	

Table 2.34: Model ECU Timing

2.7.7 Generate BSW and RTE

2.7.7.1 Description

[TR_METH_01092] Generating BSW modules, RTE, and OS source files ⌈ After the `ECU Configuration` is completed, the BSW modules, RTE, and OS source files are generated. ⌋*(RS_METH_00005, RS_METH_00003, RS_METH_00004, RS_METH_00006)*

Generation is the process of applying the tailored `ECU Configuration Value description` to the software modules. This can be performed in different ways, and is dependent on the configuration classes chosen for the different modules (see 2.7.9), and on implementers choices.

For each BSW module, a generator reads the relevant parameters from the `ECU Configuration Value description` and creates code that implements the specified configuration.

In this generation step, the abstract parameters of the `ECU Configuration Value description` are translated to hardware and implementation-specific data structures that fit to the implementation of the corresponding software module. The AUTOSAR Methodology specification does not specify the generator tools in detail.

It is assumed however that generators perform error, consistency and completeness checks on the part of the configuration they require for generation.

There are some alternative approaches when it comes to generation of configuration data. See chapter A.1.2 in [11] for more details.

2.7.7.2 Workflow

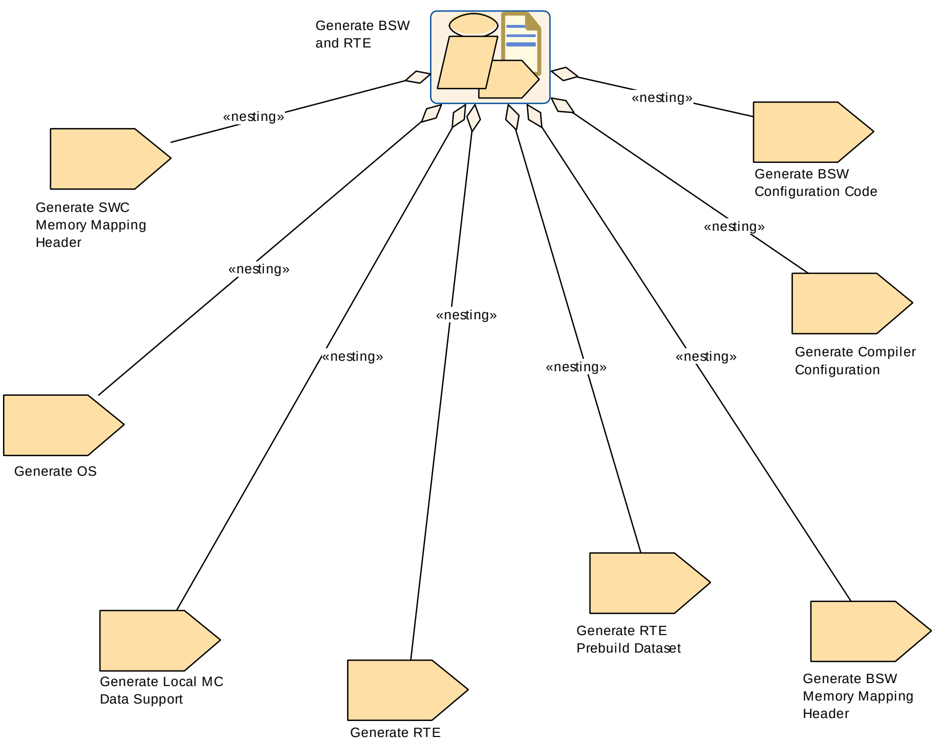

Figure 2.47: Generate BSW and RTE

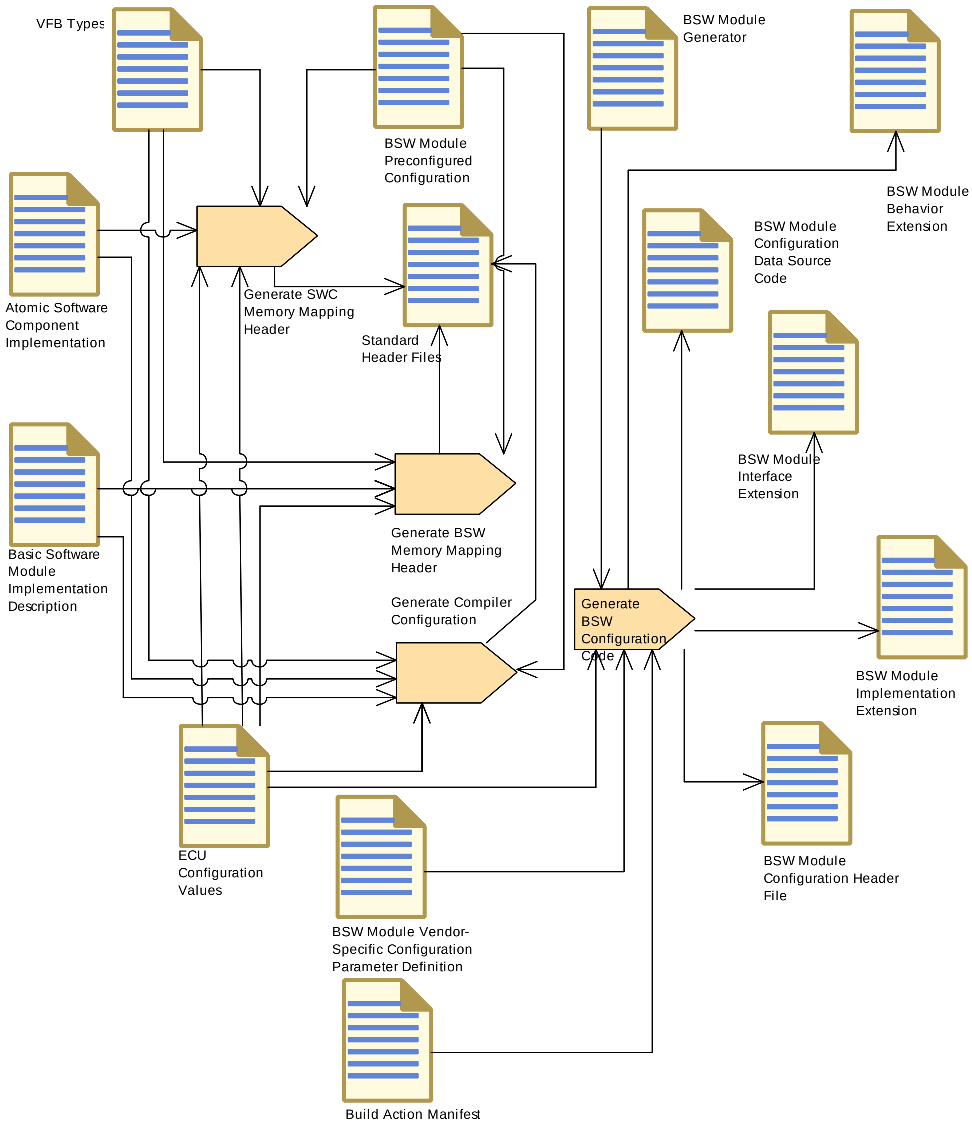

Figure 2.48: Generate BSW and RTE (Part 1)

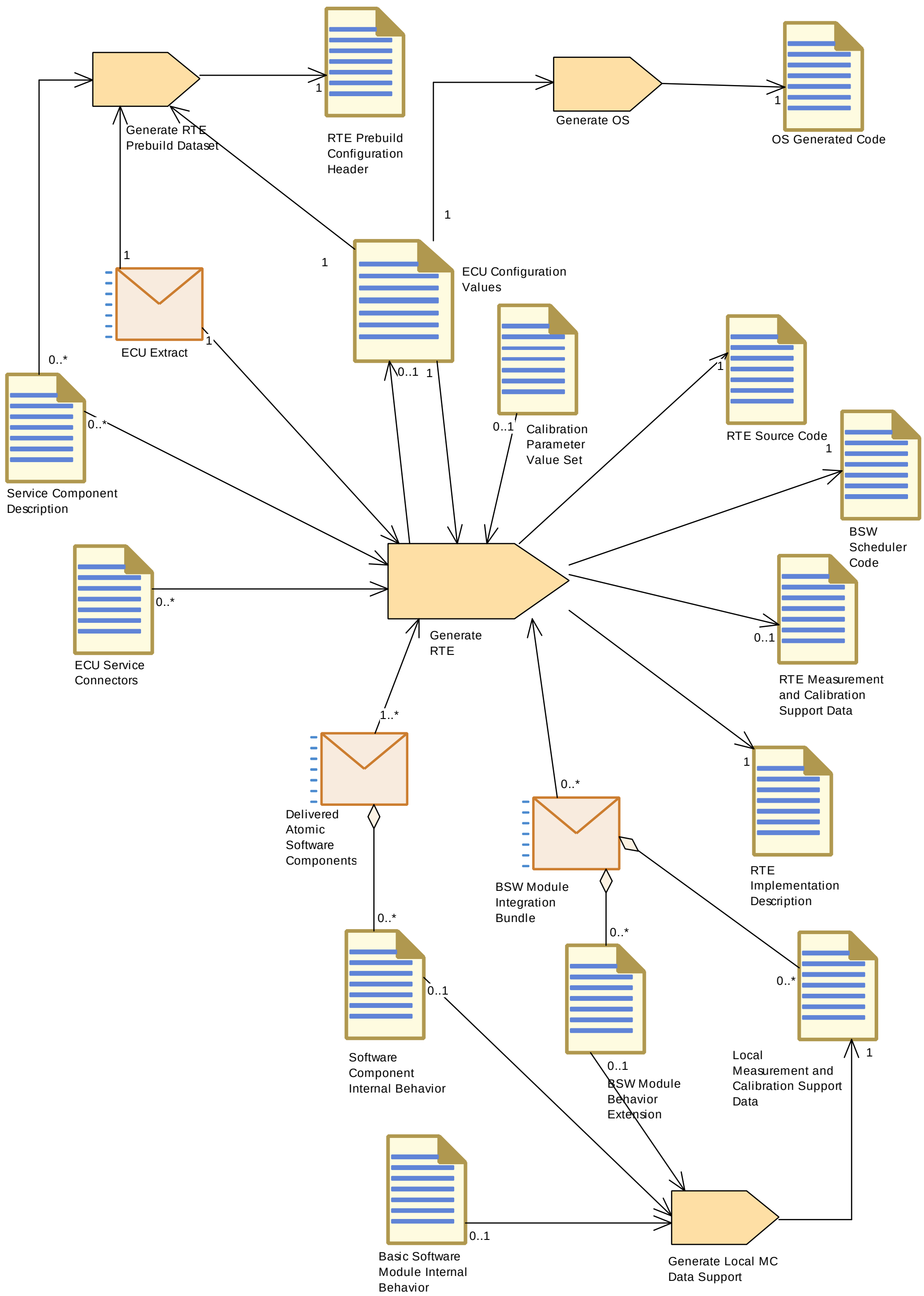

Figure 2.49: Generate BSW and RTE(Part 2)

Activity	**Generate BSW and RTE**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::ECU::Integrate Software for ECU		
Brief Description	High Level view showing how to build an AUTOSAR ECU software.		
Description	There are many possibilities how to run the configuration of the different modules in detail (see the detailed use cases for the configuration classes). This overall use case shows the generation of RTE, OS and Memory Mapping explicitly, for the other modules it shows as an example the generic task required for link time configuration of the modules plus the generic task to generate local calibration support data.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	ECU Configuration Values	1	
Produces	BSW Module Configuration Data Source Code	1	
Produces	BSW Module Configuration Header File	1	
Produces	RTE Source Code	1	
Aggregates	Generate BSW Configuration Code	1	
Aggregates	Generate BSW Memory Mapping Header	1	
Aggregates	Generate Compiler Configuration	1	
Aggregates	Generate Local MC Data Support	1	
Aggregates	Generate OS	1	
Aggregates	Generate RTE	1	
Aggregates	Generate RTE Prebuild Dataset	1	
Aggregates	Generate SWC Memory Mapping Header	1	
Predecessor	Configure BSW and RTE	1	

Table 2.35: Generate BSW and RTE

2.7.8 Build Executable

2.7.8.1 Description

[TR_METH_01093] Building `ECU Executable` ⌈ After BSW and RTE have been generated, all the source code is compiled and linked along with all the applications, libraries, object code etc. to build the ECU Executable. The details of the various compiling and linking options are explained in the chapters 2.7.9.1, 2.7.9.2, 2.7.9.3 and 2.7.9.4. ⌋*(RS_METH_00006, RS_METH_00042, RS_METH_00038)*

2.7.8.2 Workflow

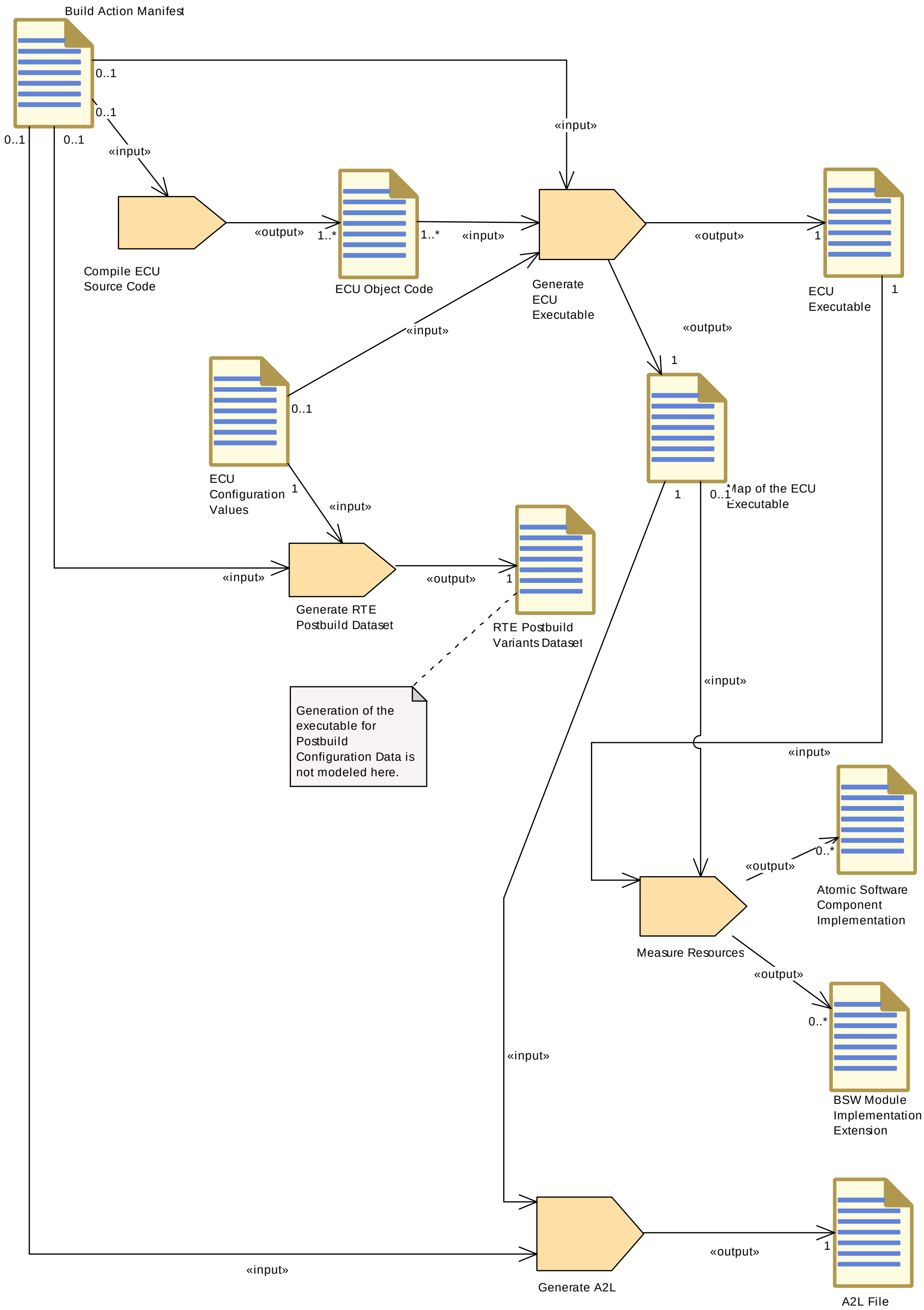

Figure 2.50: Build Executable

Activity	**Build Executable**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::ECU::Integrate Software for ECU		
Brief Description			
Description	Describes how to build one executable and related artifacts (A2L file) starting from the source code (and delivered object code).		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Compile ECU Source Code	1	
Aggregates	Generate A2L	1	
Aggregates	Generate ECU Executable	1	
Aggregates	Generate RTE Postbuild Dataset	1	
Aggregates	Measure Resources	1	
Predecessor	Generate BSW and RTE	1	

Table 2.36: Build Executable

2.7.9 Configuration Classes

The development of BSW modules involve the following development cycles: compiling, linking and downloading of the executable to ECU memory. Configuration of parameters can be done in any of these process-steps: pre-compile time, link time or even post-build time.

According to the process-step that does the configuration of parameters, the configuration classes are categorized as below

- pre-compile time
- link time
- post-build time

The configuration in different process-steps has some consequences for the handling of ECU configuration parameters. If a configuration parameter is defined as pre-compile time, after compilation this configuration parameter can not be changed any more.

Or if a configuration parameter is defined at post-build time the configuration parameter has to be stored at a known memory location. Also, the format in which the BSW module is delivered determines in what way parameters are changeable. A source code delivery or an object code delivery of a BSW module has different degrees of freedom regarding the configuration.

The configuration class of a parameter depends on the chosen implementation variants of the BSW module it belongs to. However once the module is implemented, the configuration class for each of the parameters is fixed. Choosing the right implementation variant for a module depends on the type of application and the design decisions taken by the module implementer.

Different configuration classes can be combined within one module. For example, for post-build time configurable BSW implementations only a subset of the parameters might be configurable post-build time. Some parameters might be configured as pre-compile time or link time.

File formats used for describing the configuration classes:

- `.arxml` (An xml file standardized by AUTOSAR.)
- `.exe` (An executable that can be downloaded to an ECU.)
- `.hex` (A binary file that can be downloaded to an ECU , but it can not execute by its own.)
- `.c` (A C-source file containing either source code or configuration data.)
- `.h` (A header file for either source code or configuration data.)
- `.obj` (A object file for either source code or configuration data.)

[TR_METH_01115] A mix of parameters with different configuration classes within a BSW module is allowed ⌈ In a real implementation of a BSW module all configuration parameters are most likely not in the same configuration class. I.e it will be a mix of parameters with different configuration classes within a BSW module. ⌋ *(RS_METH_00003, RS_METH_00004)*

2.7.9.1 Configuration Class: Pre-compile Time

[TR_METH_01095] Configuration Class: Pre-compile Time ⌈ ([TPS_ECUC_01031], see [11]) This type of configuration is a standalone configuration done before compiling the source code. That means parameter values for those configurable elements are selected before compiling and will be effective after compilation time. The value of the configurable parameter is decided in earlier stage of software development process and any changes in the parameter value calls for a re-compilation. The contents of pre-compile time parameters can not be changed at the subsequent development steps like link time or post-build time. ⌋ *(RS_METH_00004, RS_METH_00062)*

2.7.9.1.1 Description

The work breakdown structure shows two approaches:

[TR_METH_01096] Generating header files only ⌈ The first approach is to generate a `BSW Module Configuration Header File`, then compile the module core code using this header file. In this case the module core code is not touched by the BSW Configuration Generator. ⌋()

[TR_METH_01097] Generating header and source files ⌈ An alternative approach, in which the BSW Configuration Generator generates the complete, configuration-specific BSW Module Configuration Header Files plus BSW Module Completely Generated Source Code. In this case, no core code exist. ⌋()

Both approaches are equally valid.

Whenever the decision of parameter value must be taken before the selection of other dependable parameters, pre-compile time configuration is the right choice. For example, the algorithm choice for CRC initial checksum parameter is based on the selection of CRC type (CRC16 or CRC32). When CRC16 is selected, there will be increase in processing time but reduction in memory usage. Whereas when CRC32 is selected, there will be decrease in processing time but increase in memory usage. The correct choice should be made by the implementer before compilation of source code based on the requirement and resource availability.

Sample cases where pre-compile time configuration can be adopted are:

- Configure the number of memory tables and block descriptor table of NVRAM manager.
- Enable the macro for the development error tracing of the software modules.

2.7.9.1.2 Workflow

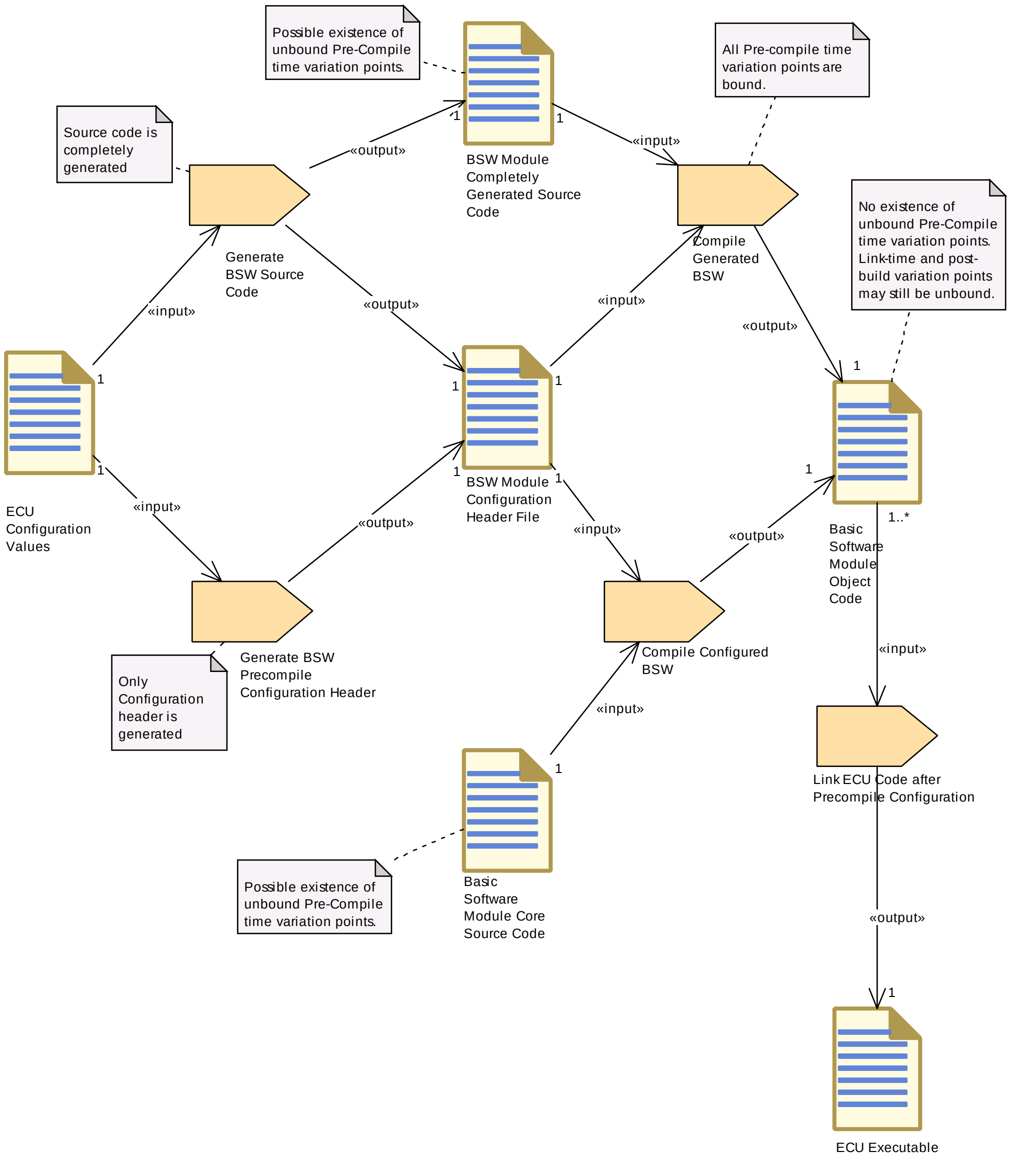

Figure 2.51: Pre-compile time configuration overview

Further description of the PreCompile binding time can be found in Section 2.16.3.6.

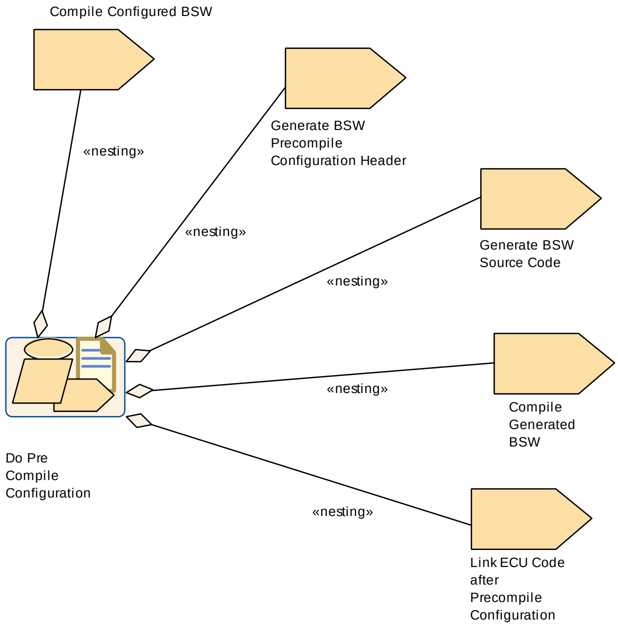

Figure 2.52: Pre compile time configuration activities

Activity	**Do Pre Compile Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::ECU:: Pre Compile Conf		
Brief Description			
Description	[from ecuc sws 1031] This type of configuration is a standalone configuration done before compiling the source code. That means parameter values for those configurable elements are defined before compiling and will be effective after compilation time. The value of the configurable parameter is decided in an earlier stage of software development process and any changes in the parameter value calls for a re-compilation. The contents of pre-compile time parameters cannot be changed at the subsequent development steps like link time or post-build time.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Compile Configured BSW	1	
Aggregates	Compile Generated BSW	1	
Aggregates	Generate BSW Precompile Configuration Header	1	
Aggregates	Generate BSW Source Code	1	
Aggregates	Link ECU Code after Precompile Configuration	1	

Table 2.37: Do Pre Compile Configuration

2.7.9.2 Configuration Class: Link Time

[TR_METH_01098] Configuration Class: Link Time ⌈ ([TPS_ECUC_01032], see [11]) This type of configuration is done for the BSW module during link time. That means the object code of the BSW module receives parts of its configuration from another object code file or it is defined by linker options. Link time parameters are typically used when delivering object code to the integrator. ⌋*(RS_METH_00004, RS_METH_00062)*

2.7.9.2.1 Description

This configuration class provides a modular approach to the configuration process. A separate module will handle the configuration details and those parameter values will be made available to the other modules during the linking process.

[TR_METH_01099] Generation and compilation of BSW Configuration Code ⌈ The first step is to `Generate BSW Configuration Code`, which produces the `BSW Module Configuration Data Source Code` and the `BSW Module Configuration Header File`. These are compiled along with the `Basic Software Module Core Header` into the `BSW Module Configuration Data Object Code`. ⌋*()*

[TR_METH_01100] Definition of configuration data ⌈ The configuration parameter data is defined in a common header file `Basic Software Module Core Header` and included by both `Basic Software Module Core Source Code` and `BSW Module Configuration Data Source Code`. The module source file needs this header file to resolve the references and module configuration source file will need it in order to cross check the declaration of data type against the definition. ⌋*()*

[TR_METH_01101] Separate compilation of module source and configuration file ⌈ Both module source file and module configuration source file are compiled separately to generate `Basic Software Module Object Code` and `BSW Module Configuration Data Object Code` respectively. ⌋*()*

[TR_METH_01102] Linking process ⌈ During the linking process, the configuration data will be available to `Basic Software Module Object Code` by resolving the external references. ⌋*()*

[TR_METH_01103] Re-generation in case of configuration value changes ⌈ When the values of configuration parameters change the `Basic Software Module Object Code` needs to be re-generated. ⌋*(RS_METH_00004)*

Sample cases where Link time configuration can be adopted are:

- Initial value and invalid value of signal
- Unique channel identifier configured for the respective instance of the Network Management.

- Logical handle of CAN network.
- Identifier and type of Hardware Reception Handle and Hardware Transmission
- Handle for CAN interface.
- Definition of ComFilterAlgorithm.
- COM callback function to indicate RTE about the reception of an invalidated signal.

2.7.9.2.2 Workflow

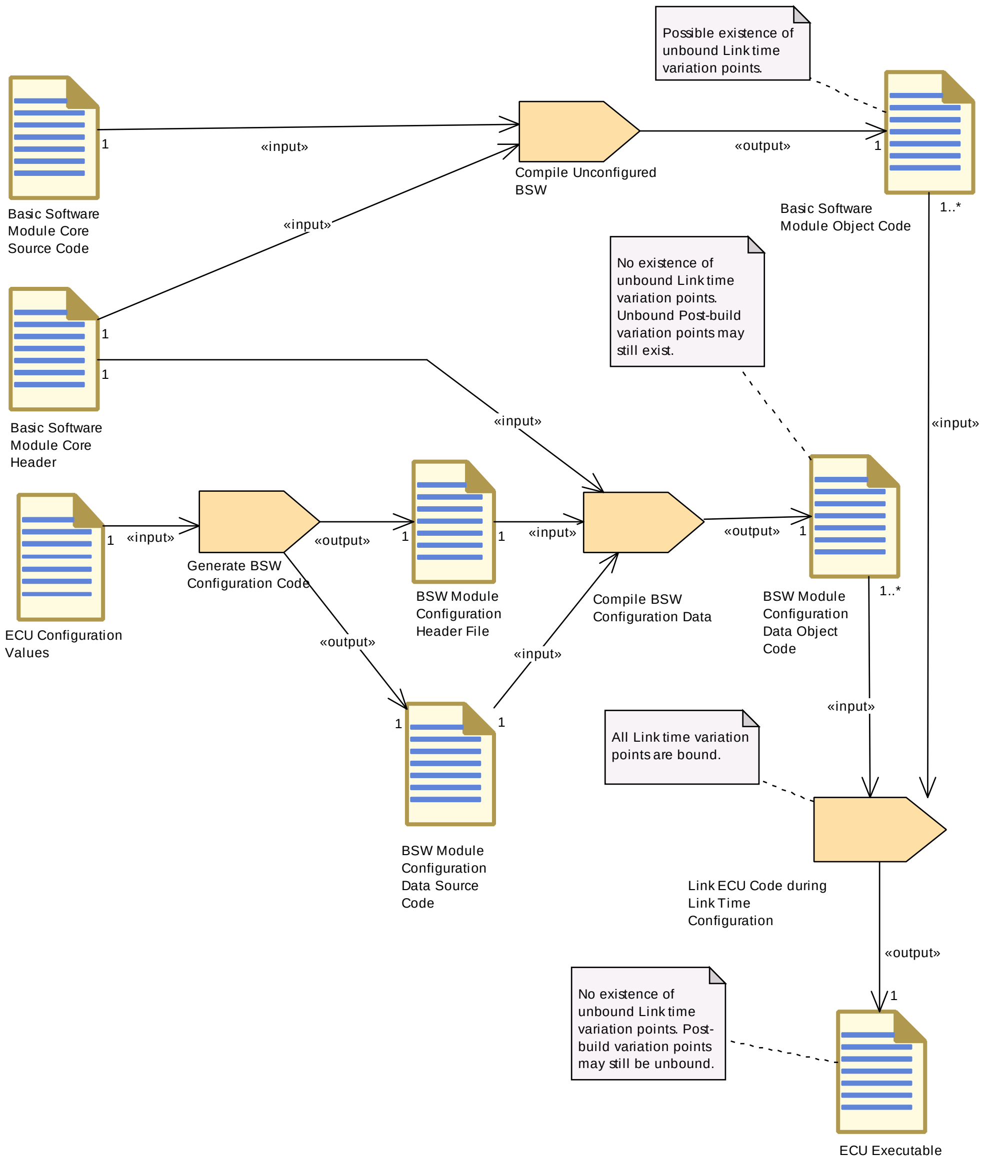

Figure 2.53: Overview Link Time Configuration

Further description of the LinkTime binding time can be found in Section 2.16.3.8.

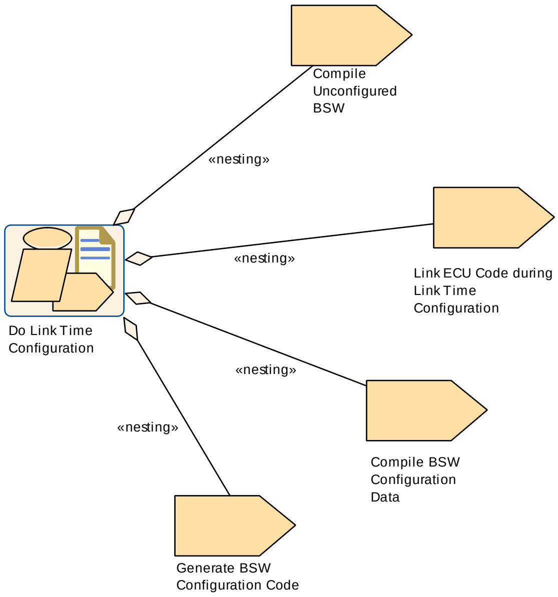

Figure 2.54: Link time configuration

Activity	**Do Link Time Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::ECU::Link Time Conf		
Brief Description			
Description	[from ecuc sws 1032] This type of configuration is done for the BSW module during link time. That means the object code of the BSW module receives parts of its configuration from another object code file or it is defined by linker options. Link time parameters are typically used when delivering object code to the integrator.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Compile BSW Configuration Data	1	
Aggregates	Compile Unconfig-ured BSW	1	
Aggregates	Generate BSW Configuration Code	1	
Aggregates	Link ECU Code during Link Time Configuration	1	

Table 2.38: Do Link Time Configuration

2.7.9.3 Configuration Class: Post-build Time

[TR_METH_01104] Configuration Class: Post-build Time ⌈ ([TPS_ECUC_04006], see [11]) This type of configuration is possible after building the BSW module or the ECU software. The BSW module gets the parameters of its configuration by downloading a separate file to the ECU memory, avoiding a re-compilation and re-build of the BSW module. ⌋*(RS_METH_00004, RS_METH_00062)*

2.7.9.3.1 Description

[TR_METH_01105] Generate BSW Postbuild Configuration Code ⌈ In order to make the post-build time re-configuration possible, the re-configurable parameters shall be stored at a known memory location of the ECU memory. In this approach the `Basic Software Module Core Source Code` is compiled and linked independently of its configuration data. The `BSW Configuration Generator` generates the configuration data as `BSW Module Configuration Data Source Code` that is compiled and linked independently of the core source code. ⌋*()*

The generation of the post-build configuration is a process that can be done multiple times. The first time it is done during the creation of the initial ECU configuration which includes the generation of both `ECU Executable` and `BSW Module Configuration Data Loadable to ECU Memory` binary files. This approach is shown in Figure 2.55. After this, the post-build configuration may be updated (the updates usually originate from the `ECU Extract`) separately from the `ECU Executable` as many times as needed according to the process shown in Figure 2.56.

Sample cases where post-build time configuration can be adopted are:

- Identifiers of the CAN frames
- CAN driver baudrate and propagation delay
- COM transmission mode, transmission mode time offset and time period
- Enabling/disabling signal transmission
- Frame packing
- Signal gateway
- LIN/FlexRay schedule

2.7.9.3.2 Workflow

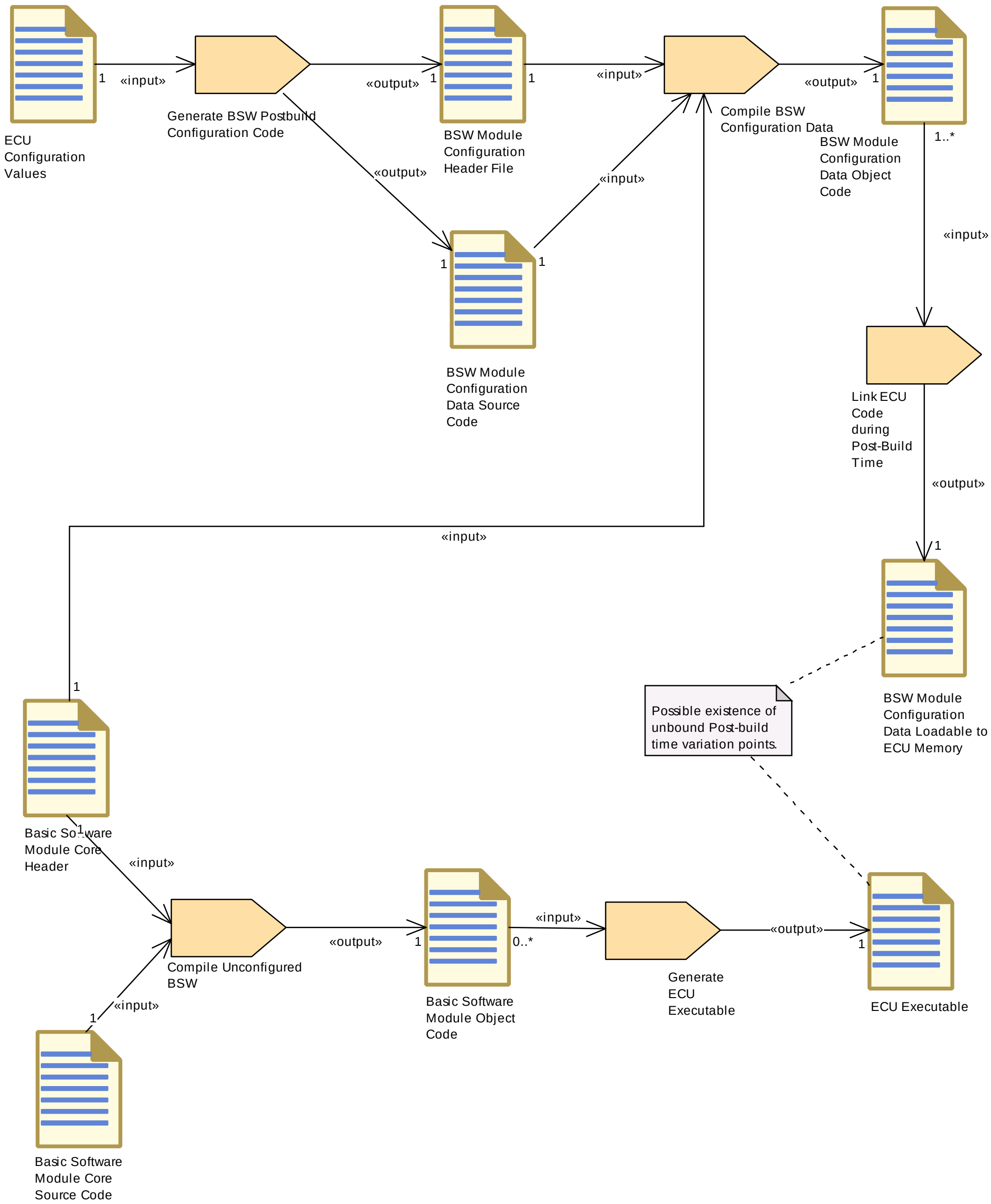

Figure 2.55: Overview of initial Post-Build Configuration

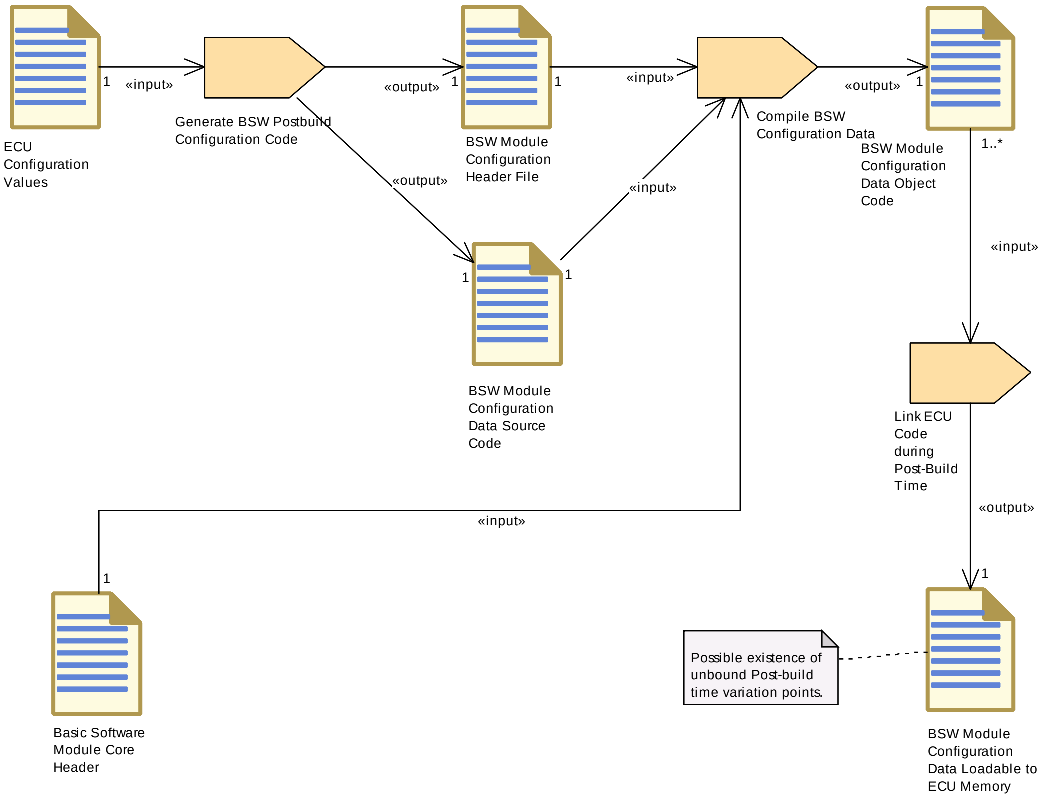

Figure 2.56: Update of the Post-Build Configuration

Further description of the PostBuild binding time can be found in Section 2.16.3.9.

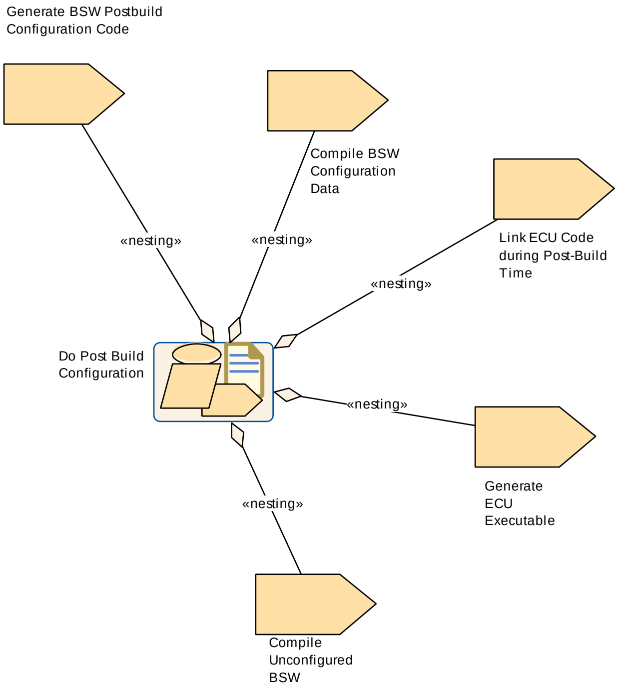

Figure 2.57: Work Flow for Post-Build Configuration

Activity	**Do Post Build Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::ECU::Post Build Conf		
Brief Description			
Description	[from ecuc sws 4006] This type of configuration is possible after building the BSW module or the ECU software. The BSW module gets the parameters of its configuration by downloading a separate file to the ECU memory, avoiding a re-compilation and re-build of the BSW module.		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Compile BSW Configuration Data	1	
Aggregates	Compile Unconfigured BSW	1	
Aggregates	Generate BSW Postbuild Configuration Code	1	
Aggregates	Generate ECU Executable	1	
Aggregates	Link ECU Code during Post-Build Time	1	

Table 2.39: Do Post Build Configuration

2.7.9.4 Handling of different post-build variants in configuration classes

2.7.9.4.1 Description

[TR_METH_01108] Generating multiple post-build configuration variants ⌈ In this use case, the `BSW Configuration Generator` generates two or more variants of configuration parameters within `BSW Module Configuration Header File`s and `BSW Module Configuration Data Source Code`. The configuration data is compiled and linked together with the `Basic Software Module Core Source Code`. The resulting ECU Executable includes all configuration variants as well as the source code of the BSW module. I.e. it is not possible to exchange the configuration data without re-building the entire executable. ⌋*(RS_METH_00062)*

[TR_METH_01150] Including different post-build variants ⌈ Different post-build variants are included in the configuration by specifying different variation points which shall be bound at post-build time. This can be done regardless of the configuration class, as shown in the notes of 2.51, Figure 2.53 and Figure 2.55. ⌋*(RS_METH_00062)*

2.8 Components and Services

2.8.1 Purpose

This use case focuses on the activities required to use and configure AUTOSAR Services. It is therefore a subset of the overall use case (see 2.1).

2.8.2 Description

[TR_METH_02000] Use of AUTOSAR Services ⌈Atomic Software Components can use AUTOSAR Services. In order to do so, two things have to be defined on the VFB and Software Component level:

- The ports which are to be connected to the Service during ECU integration (this is a sub-task of `Define VFB Application Software Component`). The port interfaces used for service ports should be standardized.
- The needs to configure the Service (for example NvM blocks or symbolic names for diagnostic events) from the perspective of the single Software Component (this is a sub-task of `Define Atomic Software Component Internal Behavior`.)

⌋*(RS_METH_00002, RS_METH_00033)*

The service ports have impact on the component API just like any other port, so there is no difference between service ports and "normal" ports with respect to API generation.

When the Application Software Components are mapped to an ECU their description is put into the corresponding `ECU Extract`. These activities belong to the System domain (see 2.5.6) and are not explicitly shown in this use case.

As part of the ECU integration, additional artifacts are generated to connect the service ports over the RTE: `Service Component Description`s, including their mapping to the Basic Software Modules, and the connectors between their ports and the service ports of the `Application Software Components`.

The use case shows also the creation of ECU configuration of the corresponding Basic Software Module (e.g. DEM, DCM, Watchdog Manager etc.). This must be done with respect to the service ports and the `Service Needs` of all `Application Software Components` connected to the corresponding Service Component (the diagram shows only the configuration activity of diagnostics as an example).

2.8.3 Workflow

Figure 2.58 shows the work sequence assumed for this use case. The next two figures 2.59 and 2.60 show the tasks and work products of the method library involved in the activities on the VFB and Component resp. the ECU level.

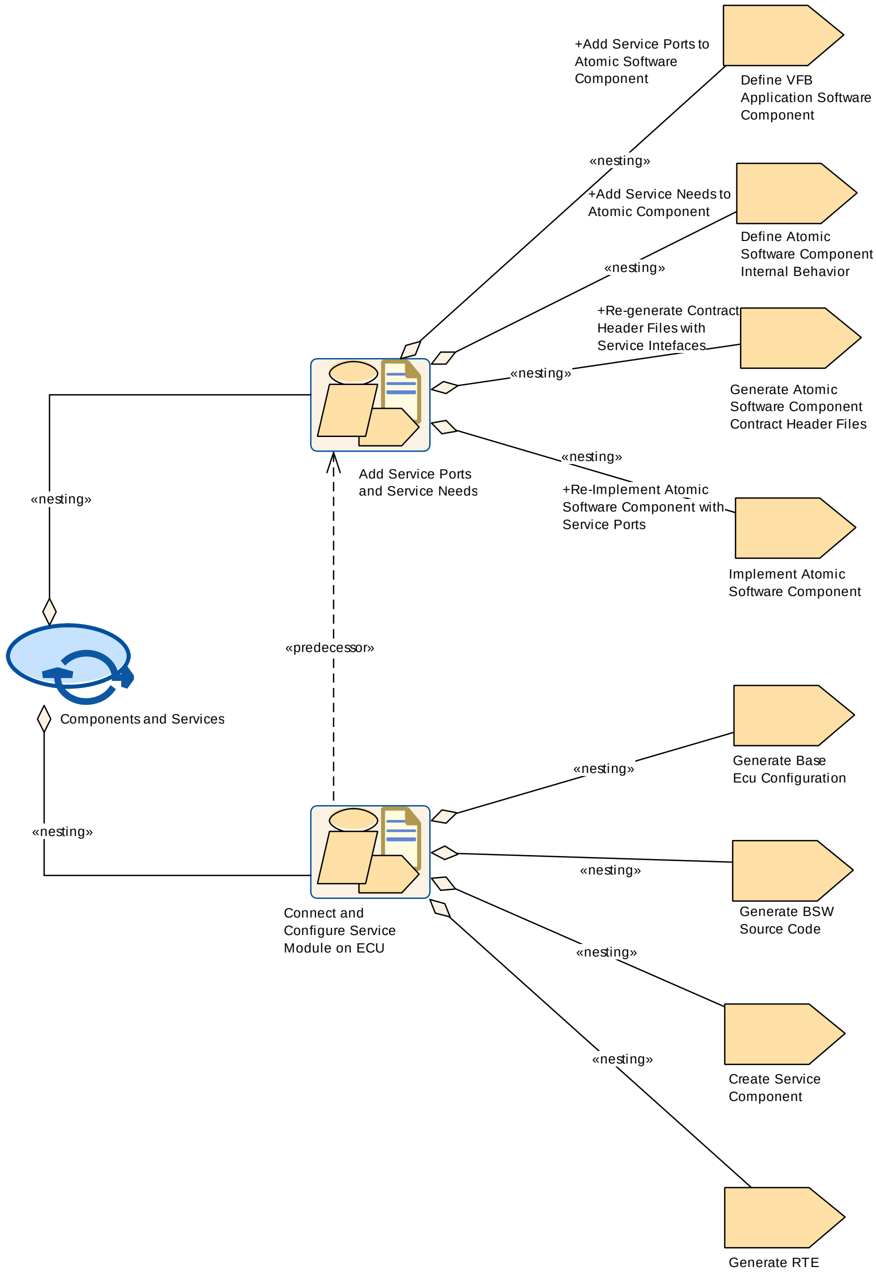

Figure 2.58: Use Case: Components and Services

Process Pattern	**Components and Services**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Components and Services		
Brief Description	This use case focuses on the activities required to use and configure AUTOSAR Services. It is therefore a subset of the overall use case (Methodology Overview).		
Description	Atomic Software Components can use AUTOSAR Services. In order to do so, two things have to be defined: The ports which are to be connected to the Service during ECU integration and in addition the needs to configure the Service (for example NvM blocks or symbolic names for diagnostic events) from the perspecive of the single Software Component. The service ports have impact on the component API just like any other port, so there is no difference between service ports and "normal" ports with respect to API generation. Afterwards the Application Software Components are mapped to an ECU and their description is put into the corresponding ECU extract (deliverable Complete ECU Description). These activities belong to the system domain and are not explictly shown in this use case (see Methodology Overview). As part of the ECU integration, additional artifacts are generated to connect the service ports over the RTE: Service Component Descriptions, including their mapping to the Basic Software Modules, and the connectors between their ports and the service ports of the Appplication Software Components. The ECU configuration of the Basic Software Module (e.g. DEM, DCM, Watchdog Manager etc.) is then created with respect to the service ports and the SeviceNeeds of the Application Software Components connected to that Service.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Add Service Ports and Service Needs	1	
Aggregates	Connect and Configure Service Module on ECU	1	

Table 2.40: Components and Services

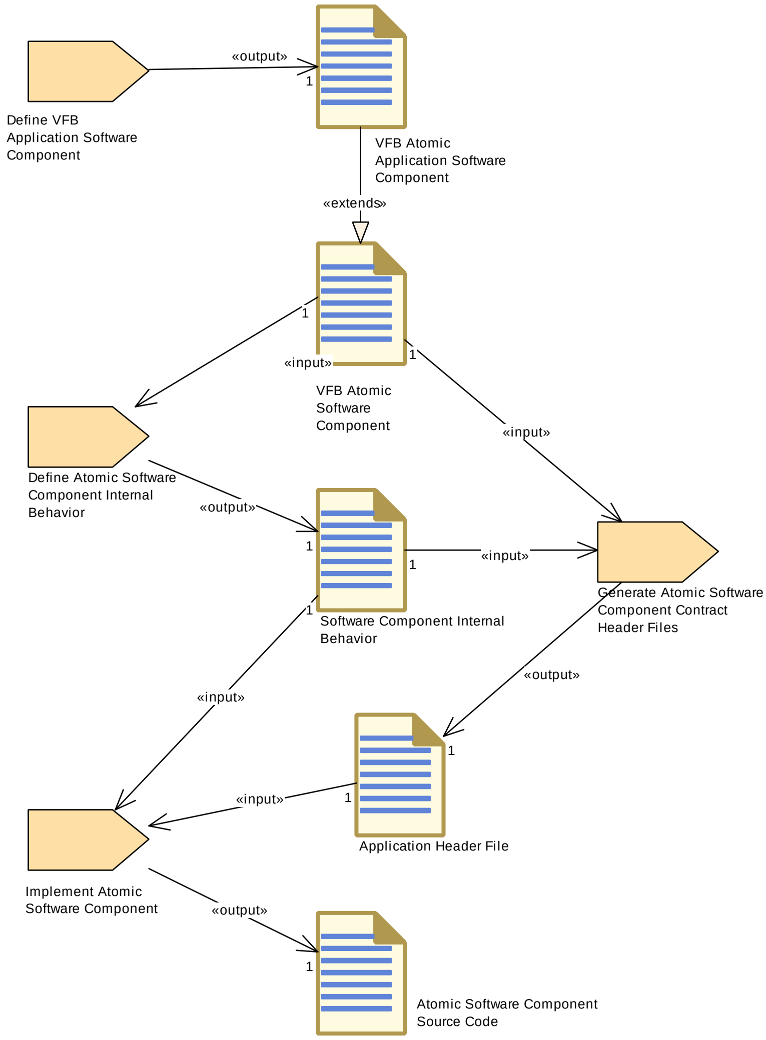

Figure 2.59: Add Service Ports and Service Needs - Detailed view with work products

Activity	**Add Service Ports and Service Needs**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Components and Services		
Brief Description			
Description	Atomic Software Components can use AUTOSAR Services. In order to do so, two things have to be defined: • The ports which are to be connected to the Service during ECU integration (this is a sub-task of Define VFB Application Software Component). The port interfaces used for service ports should be standardized. • The needs to configure the Service (for example NvM blocks or symbolic names for diagnostic events) from the perspecive of the single Software Component (this is a sub-task of Define Atomic Software Component Internal Behavior) The service ports have impact on the component API just like any other port, so there is no difference between service ports and "normal" ports with respect to API generation.		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Define Atomic Software Component Internal Behavior	1	Add Service Needs to Atomic Component:
Aggregates	Define VFB Application Software Component	1	Add Service Ports to Atomic Software Component:
Aggregates	Generate Atomic Software Component Contract Header Files	1	Re-generate Contract Header Files with Service Intefaces:
Aggregates	Implement Atomic Software Component	1	Re-Implement Atomic Software Component with Service Ports:

Table 2.41: Add Service Ports and Service Needs

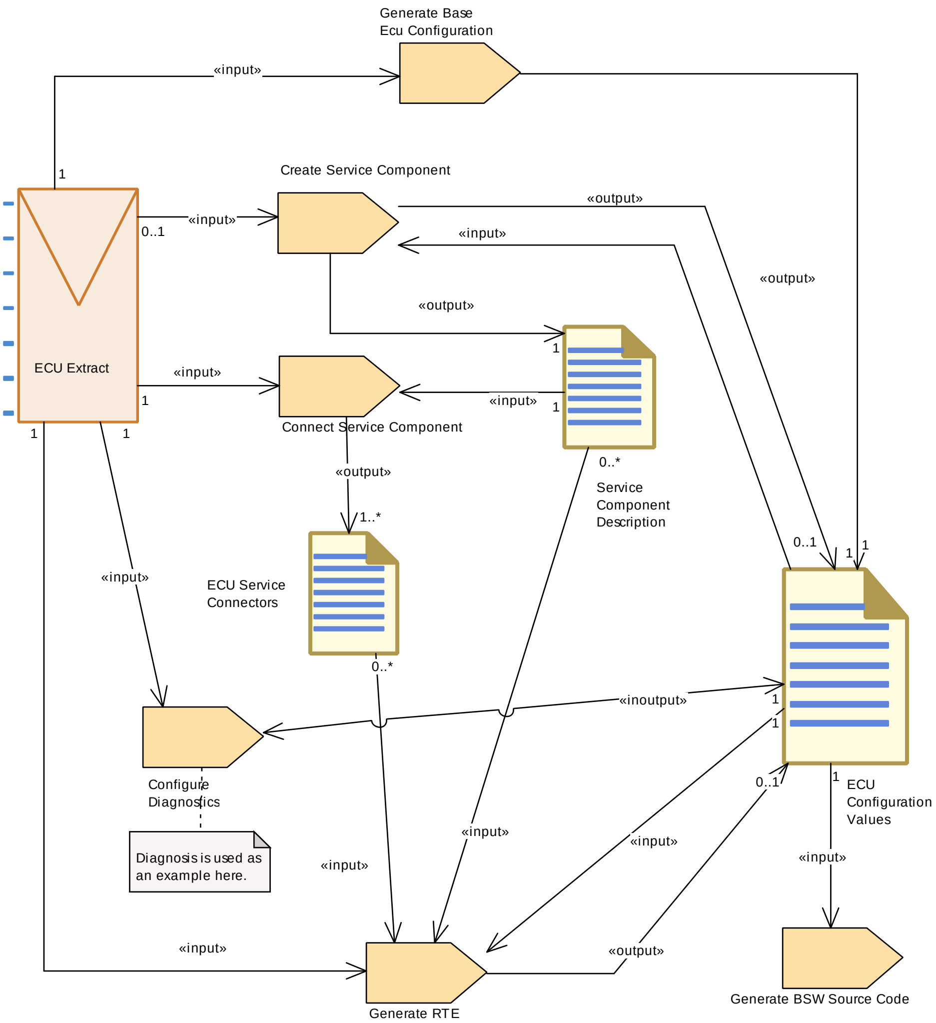

Figure 2.60: Connect and Configure Service Module on ECU - Detailed view with work products

Activity	**Connect and Configure Service Module on ECU**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Components and Services		
Brief Description			
Description	As part of the ECU integration, additional artifacts are generated to connect the service ports over the RTE: Service Component Descriptions, including their mapping to the Basic Software Modules, and the connectors between their ports and the service ports of the Appplication Software Components. The ECU configuration of the Basic Software Module (e.g. DEM, DCM, Watchdog Manager etc.) is then created with respect to the service ports and the SeviceNeeds of the Application Software Components connected to that Service (the diagram shows only the configuration activity of diagnostics as an example). The code gneration of the service module (e.g. DEM, DCM) and of the RTE is shown for completeness.		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Create Service Component	1	
Aggregates	Generate BSW Source Code	1	
Aggregates	Generate Base Ecu Configuration	1	
Aggregates	Generate RTE	1	
Predecessor	Add Service Ports and Service Needs	1	

Table 2.42: Connect and Configure Service Module on ECU

2.9 Calibration Overview

2.9.1 Purpose

This use case describes the typical activities required from the creation or update of calibration parameters down to the creation or update of the A2L Files.

2.9.2 Description

The use cases assumes, that calibration parameters are changed in an already existing system, thus the tasks required to define and build a new system are omitted, only the calibration relevant steps are shown.

In addition, the use case includes the (optional) task of updating a set of calibration parameter values as input for the RTE.

As far as AUTOSAR artifacts are involved, this use case can be divided into four major activities:

[TR_METH_02001] Define Cross-component Calibration Parameters activity ⌈ Define Cross-component Calibration Parameters: Contains the tasks used to define or update cross-component calibration parameters. These parameters have to be provided via ports by Parameter Components. ⌋ *(RS_METH_00002)*

[TR_METH_02002] Define Local Calibration Parameters activity ⌈ Define Local Calibration Parameters: Contains the tasks used to define or update component-local calibration parameters or calibration parameters defined within a BSW module. These parameters are declared within the Internal Behavior of the component (or the BSW module) which uses them. ⌋*(RS_METH_00002, RS_METH_00003)*

[TR_METH_02003] Provide Unique Parameter Names activity ⌈ Provide Unique Parameter Names: Contains the tasks used to provide unique names for calibration parameters. A Flat Map is used to provide unique names for MCD tools. An Alias Name Set can be provided additionally in cases, where this is not sufficient. ⌋*(RS_METH_00005)*

[TR_METH_02004] Re-generate RTE and Calibration Support activity ⌈ Re-generate RTE and Calibration Support: Contains the tasks used to re-generate relevant artifacts during ECU integration (before the final build) after an update of calibration parameters. ⌋*(RS_METH_00020)*

2.9.3 Workflow

Figure 2.61 shows the work sequence assumed for this use case.

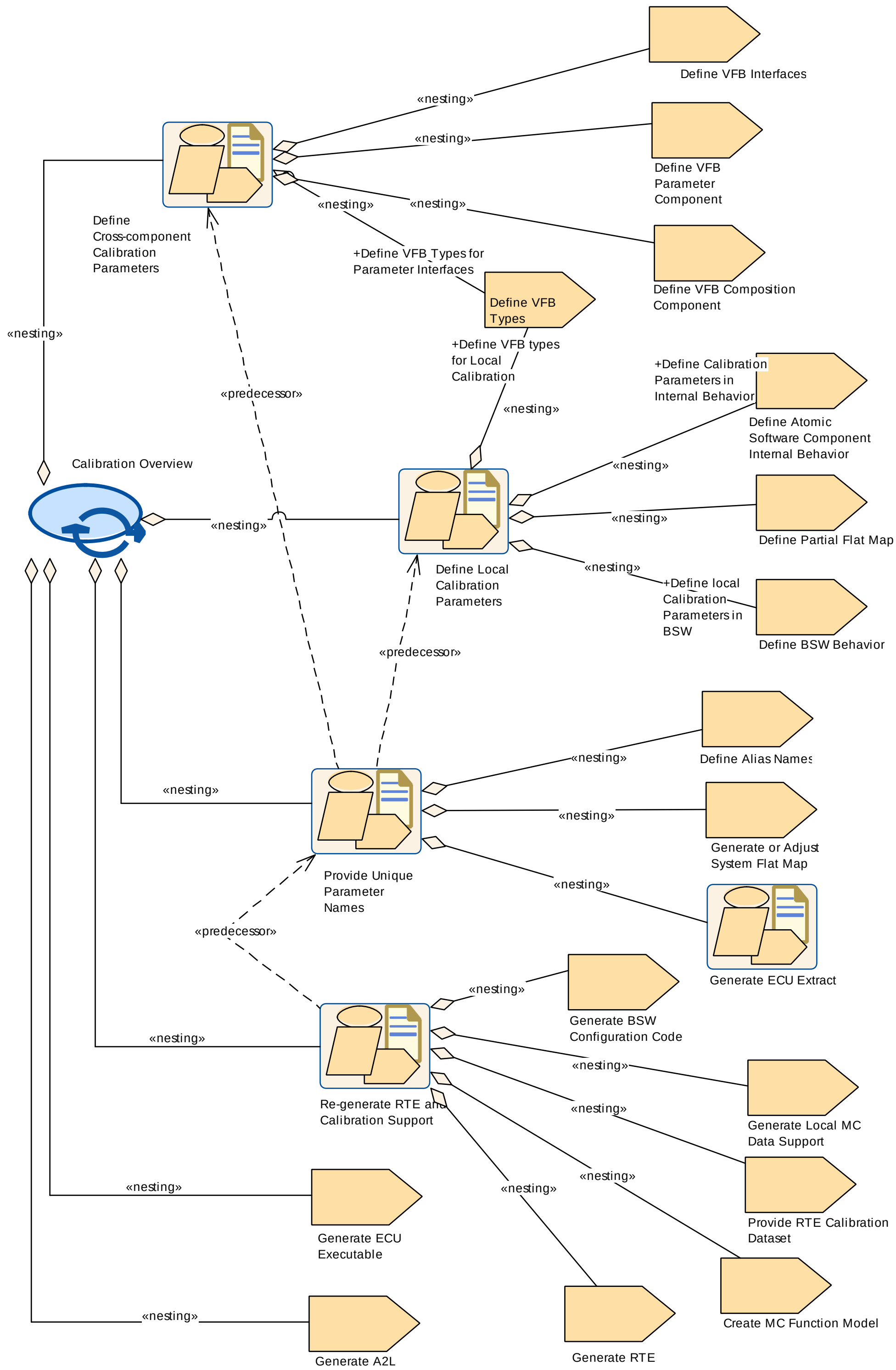

Figure 2.61: Use Case: Calibration Overview

Process Pattern	**Calibration Overview**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Calibration Overview		
Brief Description	Describe the required steps to update the calibrations data down to an update of the A2L files.		
Description	This use case shows the typical steps required from an updated design of calibration data down to an update of the A2L file. The use cases assumes, that calibration parameters are changed in an already existing system, thus the steps required to define and build a new system are omitted, only the calibration relevant steps are shown. In addition, the use case includes the (optional) task of updating a set of calibration parameter values as input for the RTE.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Define Cross-component Calibration Parameters	1	
Aggregates	Define Local Calibration Parameters	1	
Aggregates	Generate A2L	1	
Aggregates	Generate ECU Executable	1	
Aggregates	Provide Unique Parameter Names	1	
Aggregates	Re-generate RTE and Calibration Support	1	

Table 2.43: Calibration Overview

Activity	**Define Cross-component Calibration Parameters**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Calibration Overview		
Brief Description			
Description	Contains the tasks used to define or update cross-component calibration parameters. These parameters are provided by Parameter Components.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Define VFB Composition Component	1	
Aggregates	Define VFB Interfaces	1	
Aggregates	Define VFB Parameter Component	1	
Aggregates	Define VFB Types	1	Define VFB Types for Parameter Interfaces: Use this task to define VFB Types for Parameter Interfaces

Table 2.44: Define Cross-component Calibration Parameters

Activity	Define Local Calibration Parameters		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Calibration Overview		
Brief Description			
Description	Contains the tasks used to define or update component-local (or module-local) calibration parameters. These parameters are declared within the Internal Behavior of the component (or BSW module) which uses them.		
Relation Type	Related Element	Mul.	Note
Aggregates	Define Atomic Software Component Internal Behavior	1	Define Calibration Parameters in Internal Behavior: Use this task to define local calibration parameters as part of the Internal Behavior of a software component.
Aggregates	Define BSW Behavior	1	Define local Calibration Parameters in BSW: Use this task to define local calibration parameters as part of the Internal Behavior of a BSW module.
Aggregates	Define Partial Flat Map	1	Define (optionally) a Partial Flat Map for one or more delivered components.
Aggregates	Define VFB Types	1	Define VFB types for Local Calibration: Use this task to define VFB types for Local Calibration.

Table 2.45: Define Local Calibration Parameters

Activity	Provide Unique Parameter Names		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Calibration Overview		
Brief Description			
Description	Contains the tasks used to provide unique names for calibration parameters. A Flat Map is used to provide unique names for MCD tools. An Alias Name Set can be provided in cases, where this is not sufficient.		
Relation Type	Related Element	Mul.	Note
Aggregates	Define Alias Names	1	
Aggregates	Generate ECU Extract	1	Use this activity to update the ECU Extract. This includes updating the ECU Flat Map if parameter names on ECU level have changed.
Aggregates	Generate or Adjust System Flat Map	1	Use this task if parameter names are defined on system level.
Predecessor	Define Cross-component Calibration Parameters	1	
Predecessor	Define Local Calibration Parameters	1	

Table 2.46: Provide Unique Parameter Names

Activity	Re-generate RTE and Calibration Support		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Calibration Overview		
Brief Description			
Description	Contains the tasks used to re-generate relevant artifacts during ECU integration (before the final build) after an update of calibration parameters.		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregates	Create MC Function Model	1	This use case shows the creation of an MC Function Model as part of the activity that generates also the RTE and calibration support data. This is only one possibility. It is also possible to create an MC Function Model earlier in the process (as part of the design activities) or later (shortly before the A2L is generated).
Aggregates	Generate BSW Configuration Code	1	Use this task to generate the description of calibration parameters in BSW that are a result of ECU configuration. Such parameters will be described within the artifact BSW Module Behavior Extension.
Aggregates	Generate Local MC Data Support	1	Use this task to generate support for calibration data that are not handled via the RTE.
Aggregates	Generate RTE	1	Use this task to generate support for calibration data that are handled over the RTE. This includes cross-component calibration as well as local calibration (in SWC and BSW) that needs emulation support by the RTE.
Aggregates	Provide RTE Calibration Dataset	1	
Predecessor	Provide Unique Parameter Names	1	

Table 2.47: Re-generate RTE and Calibration Support

2.10 Memory Mapping

2.10.1 Purpose

This use case gives a comprehensive view on the tasks required to define, configure and generate header files for memory mapping and for the compiler abstraction related to memory aspects. The underlying concepts are specified in [12] and [13].

2.10.2 Description

[TR_METH_02005] Memory sections for data and code ⌈ AUTOSAR basic software as well as application software use a standardized preprocessor mechanism in order to define memory sections for their data and code as well as compiler memory classes[2] defined globally or per section. The goal of this mechanism is to maintain the compiler specific statements and the ECU specific mappings separately from the main code. ⌋ *(RS_METH_00002, RS_METH_00003, RS_METH_00004, RS_METH_00038)*

With AUTOSAR it is possible to derive (i.e. generate) the content of these header files from XML artifacts. This use case shows how the required artifacts and tasks are related.

2.10.3 Workflow

Figure 2.62 shows the work sequence assumed for this use case. The next figures 2.63 and 2.64 show the involved tasks and work products of the method library.

Note that this use case ends with compilation of the code. The assignment of memory sections to the actual hardware (which is typically done by the configuration of the linker) is currently not considered to be part of the AUTOSAR methodology.

[2] This determines far and near addressing on certain platforms.

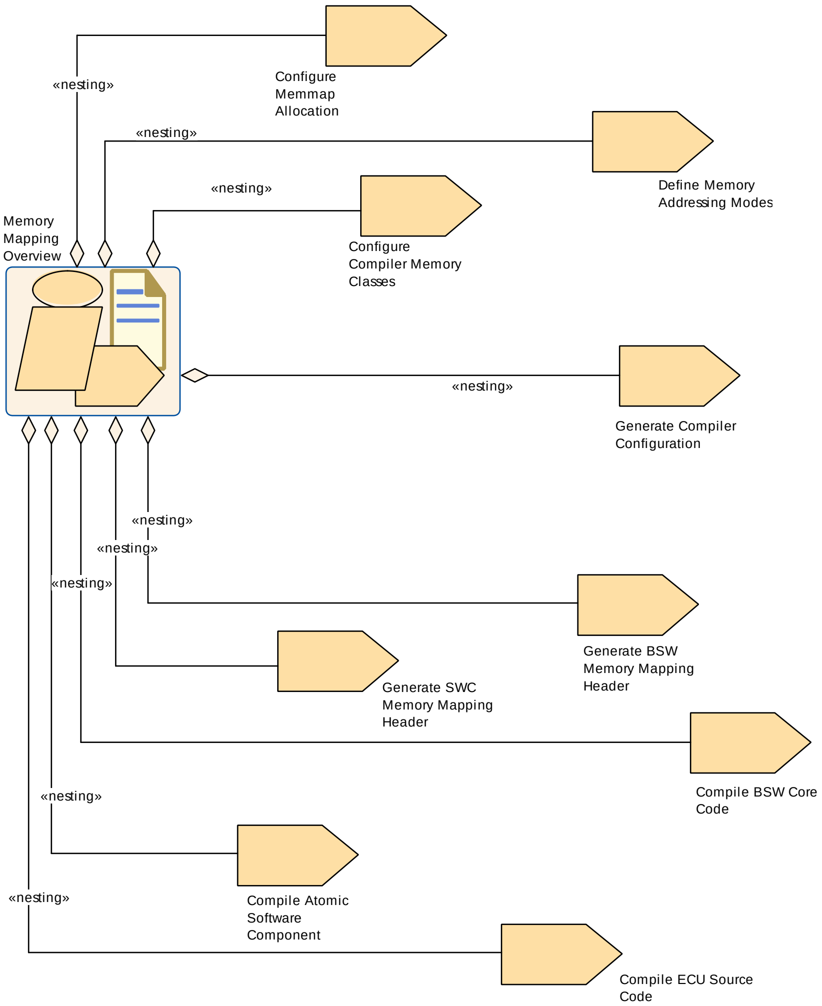

Figure 2.62: Use Case: Memory Mapping

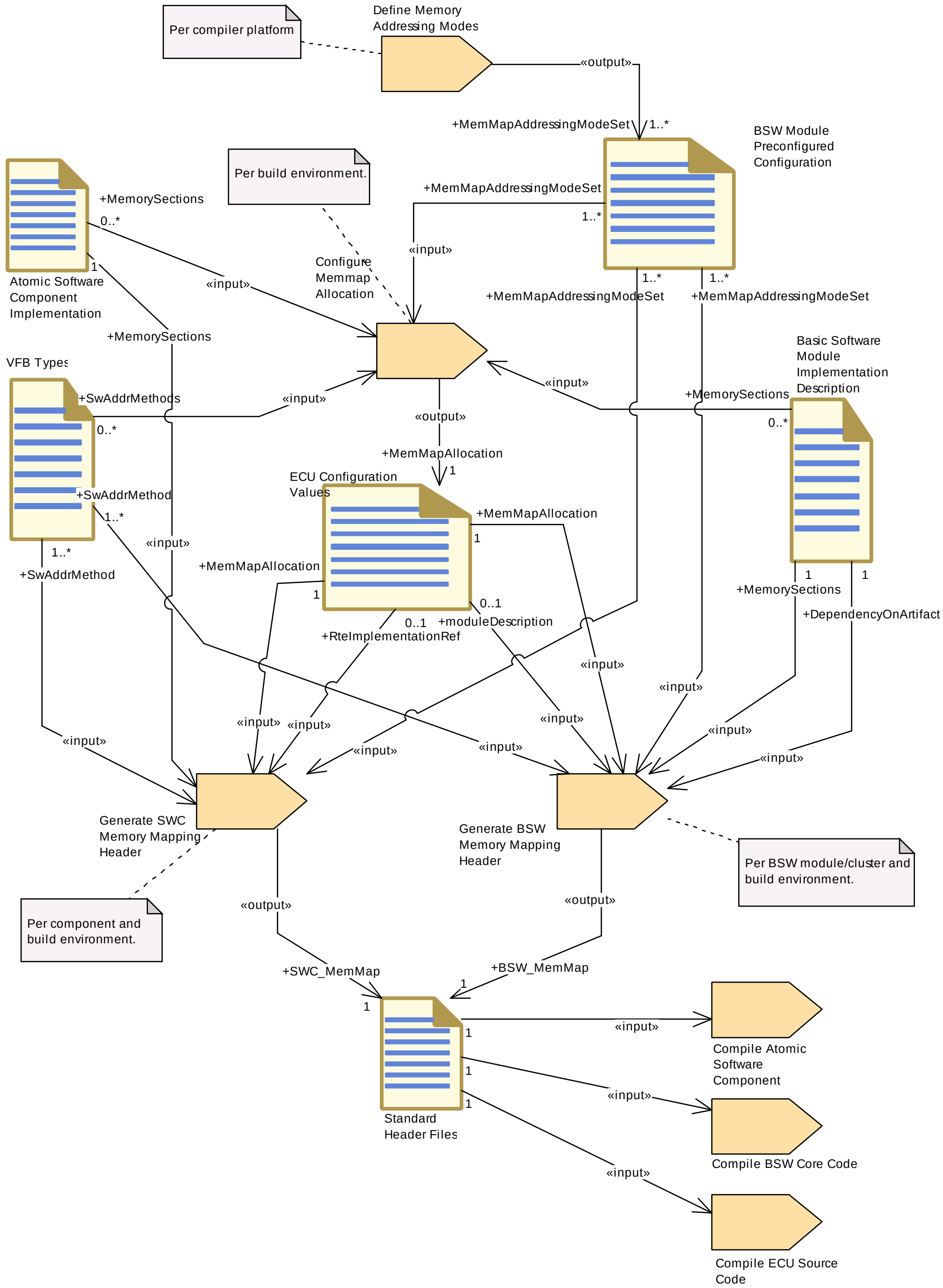

Figure 2.63: Memory Mapping - Detailed view with work products

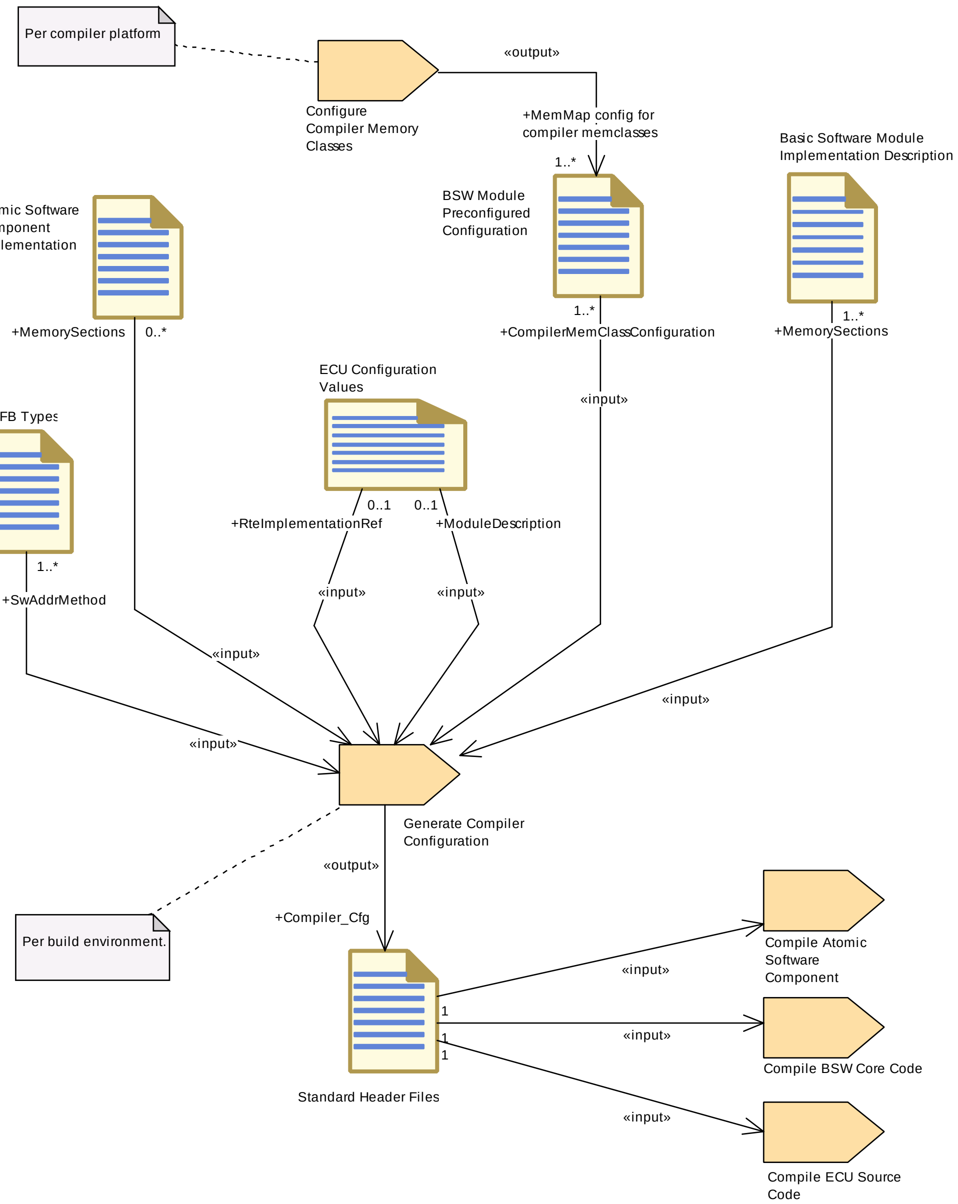

Figure 2.64: Compiler Configuration - Detailed view with work products

Activity	**Memory Mapping Overview**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Memory Mapping Overview		
Brief Description			
Description	Overview of the work sequence for defining and configuration of memory sections.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Compile Atomic Software Component	1	
Aggregates	Compile BSW Core Code	1	

Relation Type	Related Element	Mul.	Note
Aggregates	Compile ECU Source Code	1	
Aggregates	Configure Compiler Memory Classes	1	
Aggregates	Configure Memmap Allocation	1	
Aggregates	Define Memory Addressing Modes	1	
Aggregates	Generate BSW Memory Mapping Header	1	
Aggregates	Generate Compiler Configuration	1	
Aggregates	Generate SWC Memory Mapping Header	1	

Table 2.48: Memory Mapping Overview

2.11 E2E Protection

2.11.1 Purpose

This `Activity` provides a rough outline of the creation of `E2E Protection` to secure communication flow in an AUTOSAR Architecture. [14]

2.11.2 Description

`E2E Protection` mechanisms are needed when safety related data exchanges need to be protected at runtime against communication link faults.

[TR_METH_02006] E2E Protection ⌈ The `E2E Protection` in AUTOSAR is realized as an E2E Transformer Module [14] which is invoked by the RTE. First of all, the Serializer Transformer serializes the data and then the RTE invokes E2E Transformer to protect the communication. The software component communicates through RTE using the plain RTE API. ⌋*(RS_METH_00005)*

[TR_METH_01153] Configuration and Generation of the E2E Transformer ⌈ According to the generic transformer approach, the E2E Transformer can be configured at the system level (Inter-ECU communication). The generation of the E2E Transformer module is done based on the System Description. No ECU configuration is needed. ⌋*(RS_METH_00005)*

[TR_METH_01154] **Define E2E Transformer Technology Task** ⌈ The task Define E2E Transformer Technology is needed to define all information required for the generation of the E2E transformer module like pre-defined Profiles and state machine configuration. ⌋ *(RS_METH_00005)*

2.11.3 Workflow

Figure 2.65 shows the Define E2E Transformer Technology task which is mainly processed in the activity Design Communication.

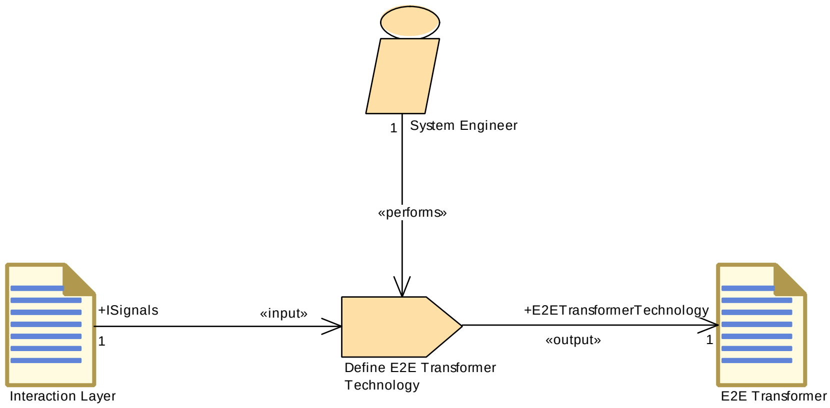

Figure 2.65: Task Define E2E Transformer Technology

2.12 Diagnostic Extract

2.12.1 Purpose

This use case provides a rough outline of the diagnostics configuration using the Diagnostic Extract Template [15]. The involved activities and deliverables will be refined based on the experience in the field in next AUTOSAR releases.

2.12.2 Description

The distributed nature of an AUTOSAR ECU development requires an optimized capturing of information. In particular, diagnostic information (i.e. DEM and DCM configuration) shall be captured only once by the person with the best knowledge and therefore being able to take responsibility better than one centralized individual. ECU configuration is not suitable to be exchanged between partners in an ECU development project. Therefore, AUTOSAR defines the Diagnostic Extract Template that represents a standardized exchange format on diagnostic functionality. The Diagnostic Extract Template allows the decentralized configuration of diagnostic aspects. The basic usage of the Diagnostic Extract Template is the exchange of diagnostic data between the different parties involved in the diagnostic development process to allow the configuration of the DCM and the DEM and to provide the description of corresponding application interfaces to implement diagnostic services and fault handling. In the AUTOSAR Methodology the Diagnostic Extract is represented by the deliverable Diagnostic Extract and its sub-deliverables.

[TR_METH_01136] Content of Diagnostic Extract ⌈ The deliverable Diagnostic Extract contains all relevant diagnostics aspects.

- Diagnostic Services (e.g. IOControl, MemoryByAddress)
- Diagnostic Event Handling (e.g. events, trouble codes, conditions)
- Mappings (Service Mappings, Diagnostic Mappings, etc.)

⌋*(RS_METH_00082)*

[TR_METH_01137] Diagnostic Extract category ⌈ Depending on the phase in the process, the Diagnostic Extract can have several categories that are represented as specialized deliverables:

- Diagnostic Abstract System Description: This deliverable represents a high-level definition that can be taken as a template for creating concrete Diagnostic System Extracts or Diagnostic ECU Extracts.
- Diagnostic System Extract: This deliverable represents the diagnostic aspects for several ECUs.
- Diagnostic ECU Extract: This deliverable represents the diagnostic aspects for a single ECUs.

⌋*(RS_METH_00082)*

[TR_METH_01138] Decentralized configuration ⌈ The timing and frequency of exchanges and the content in each of these exchanged files is highly dependent on the individual project setup and situation. The Diagnostic Extract Template has been designed to support the decentralized and independent definition of diagnostic requirements that can be linked together at a late point during the development process. The approach of decentralized configuration is met in the Diagnostic Extract Template mainly in two ways:

- Separation of elements over several physical files: Most elements of the Diagnostic Extract template can be split over several physical files. Therefore, parts of these elements (e.g. certain attributes) can be defined by, for example, an OEM and other parts of these elements by, for example, an ECU supplier.
- Usage of self-contained mappings: Many diagnostic requirements are established by mappings between diagnostic elements (e.g., DTC to DemEvent mapping). However, the "‘decentralized configuration"’ approach requires that these mappings can be flexibly defined at almost any time within the ECU development process and by any of the involved companies respectively roles. Therefore, the Diagnostic Extract Template defines self-contained mapping elements that have

references to two (or potentially more) diagnostic elements to define a mapping. The usage of the Diagnostic Extract Template will be restricted by the appropriate application of the "‘roles and rights"’ concepts in next AUTOSAR releases.

⌋*(RS_METH_00082)*

[TR_METH_01139] Roles ⌈ The relevant activities of the Diagnostic Extract use case are logically grouped to the following roles (see diagram 2.67): Diagnostic Requester, Software Developer and Diagnostic Integrator. Obviously, the OEM acts as a diagnostic requester and the ECU supplier as the diagnostic integrator. Nevertheless, in several situations (e.g. in-house development of application software components), the OEM may act as the diagnostic integrator and performs collecting and merging tasks. ⌋*(RS_METH_00082)*

[TR_METH_01140] `Develop Diagnostic Abstract System Description` activity ⌈ The basic workflow for the configuration of the diagnostic aspects may start with the optional activity `Develop Diagnostic Abstract System Description`. This activity defines diagnostic requirements at abstract level. The resulting `Diagnostic Abstract System Description` may be used by the following activity as a basis for the `Diagnostic System Extract` or the `Diagnostic ECU Extract`. ⌋*(RS_METH_00082)*

[TR_METH_01141] Development of diagnostic requirements ⌈ In the activity `Develop Diagnostic Requirements` the requester of diagnostic data defines the diagnostic interfaces of one or multiple ECUs. The following tasks may be performed:

- Define the values of the DTCs
- Define the UDS services and sub-services supported by the ECUs
- Define the required events needed by a specific composition implemented by an Application Developer

During this activity, several `Develop Diagnostic Requirements` from different parties may be collected and merged. ⌋*(RS_METH_00082)*

[TR_METH_01142] Diagnostic information in the context of SW-C development ⌈ The purpose of the `Diagnostic Extract` during the development of software components is basically twofold: On the one side the `Diagnostic System Extract` may serve as a requirement for the software developer. The diagnostic requester can specify e.g. the following issues:

- Definition of the content of a specific ReadDataByIdentifier which has to be implemented by a specific SW-C
- Definition of the events needed for a certain SW-C

On the other side the application developer has the possibility to provide diagnostic information relevant to the SW-Cs as a part of the `Diagnostic System Extract` and/or using Service Needs. The Service Needs within the SW-C Description are still to be used along with the `Diagnostic System Extract` in order to annotate the

SW-C ports which are relevant for further mapping and handling as defined by the Diagnostic System Extract. ⌋*(RS_METH_00082)*

[TR_METH_01143] Integration of diagnostic information ⌈ In activity Integrate Diagnostic Information, the integrator receives one or several Diagnostic System Extracts (or Diagnostic ECU Extracts) from the diagnostic requester and from multiple application software or basic software developers. The main goal of the integration activity is to integrate and merge all delivered Diagnostic Extracts so that the configuration of the corresponding basic software modules (DCM, DEM) can be generated (activity Integrate Software for ECU).

Since the AUTOSAR Methodology does not restrict the definition of elements like DIDs, parameters of a UDS service, Events, Sessions, etc. in activity Integrate Diagnostic Information the integrator has to ensure that the complete information is still valid after merging it. Usually, the following task may be performed:

- Mapping of DTCs (Diagnostic Trouble Code) to events
- Merge of events
- Mapping of services

During the integration activity the following issues and conflicts may be considered:

- Some DTCs may already be mapped to events - especially in cases where both come from the same party. But if the DTCs are defined by the OEM and the software components are implemented by other supplier acting as an application developer the integrator has to ensure that both are mapped together.
- In some cases, an diagnostic event may be defined multiple times. An diagnostic requester defines the events which shall be implemented by an application developer. A supplier implements a software component which will be used in multiple projects and which also detects this type of error and also defines this same event. Both events may have different naming but the same meaning. The integrator has to detect this redundancy during the integration and merge them together.
- The diagnostic requester requires a specific ReadDataByIdentifier and an application developer implements it. If the implementation is performed for one specific project only, the application developer may map the DID from the diagnostic requester to the already defined job in their software component. In other cases in which the application developer implements a generic diagnostic job, it will be a task of the diagnostic integrator to merge this information and to map the jobs to the corresponding DID.

⌋*(RS_METH_00082)*

After all issues and conflicts are resolved and the inputs are merged, the final complete Diagnostic ECU Extract is produced. Based on this deliverable, the initial configuration of the relevant basic software modules is generated (activity Integrate Software for ECU).

2.12.3 Workflow

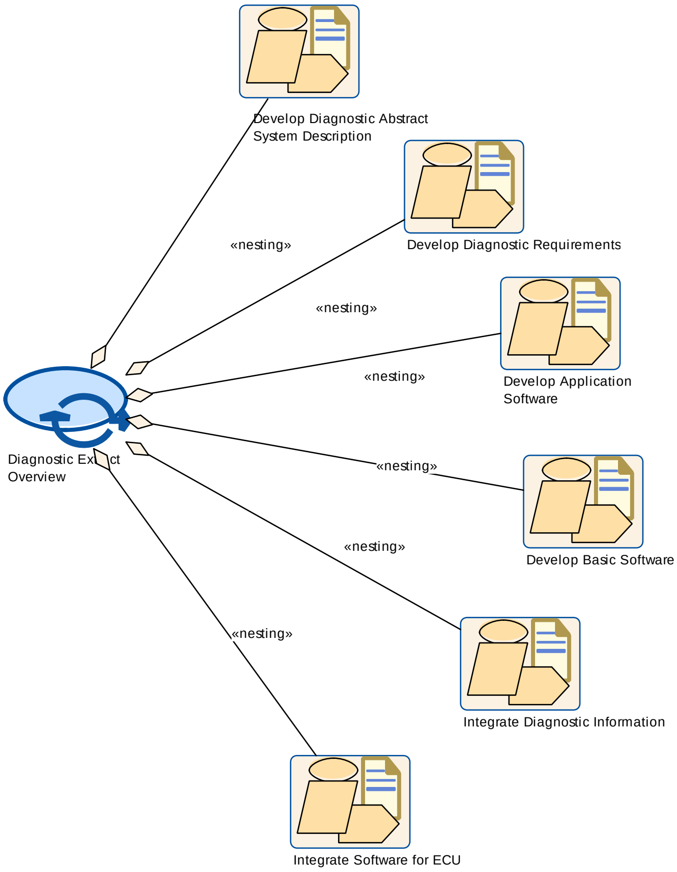

Figure 2.66: Diagnostic Extract Overview

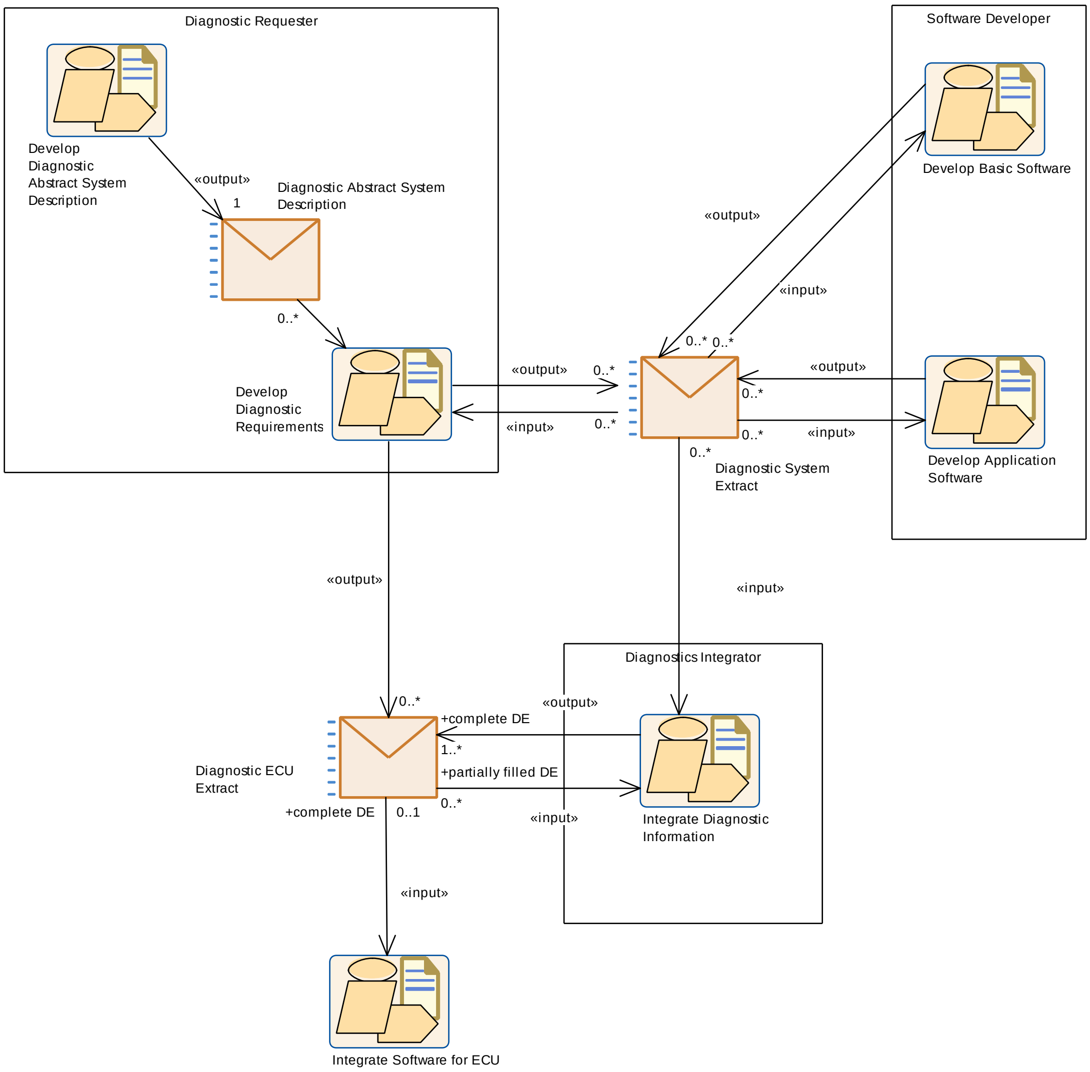

Figure 2.67: Diagnostic Extract Workflow

Process Pattern	**Diagnostic Extract Overview**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Diagnostic Extract Overview		
Brief Description			
Description			
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Develop Application Software	1	
Aggregates	Develop Basic Software	1	
Aggregates	Develop Diagnostic Abstract System Description	1	
Aggregates	Develop Diagnostic Requirements	1	
Aggregates	Integrate Diagnostic Information	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Integrate Software for ECU	1	

Table 2.49: Diagnostic Extract Overview

Activity	**Develop Diagnostic Abstract System Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Diagnostic Extract Overview		
Brief Description			
Description	This activity defines diagnostic requirements at functional/abstract level. The resulting Diagnostic Abstract System Description may be used by the following activity as a basis for the Diagnostic System Extract or the Diagnostic ECU Extract.		
Relation Type	*Related Element*	*Mul.*	*Note*
Produces	Diagnostic Abstract System Description	1	

Table 2.50: Develop Diagnostic Abstract System Description

Activity	**Develop Diagnostic Requirements**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Diagnostic Extract Overview		
Brief Description			
Description	In this activity the OEM or diagnostic requirer defines the diagnostic interfaces of one or multiple ECUs. It may also define some InternalBehaviors as requirements for the ECU-Supplier or application developer. The following tasks may be relevant: • Define the values of the DTCs • Define the UDS services and sub-services supported by the ECUs • Define the required events needed by a specific composition Additionally, the OEM may also collect Diagnostic Extracts from different departments as well as from SW-C developers and merge the information into one Diagnostic Extract.		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Diagnostic System Extract	0..*	
Produces	Diagnostic ECU Extract	0..*	
Produces	Diagnostic System Extract	0..*	

Relation Type	Related Element	Mul.	Note
	Diagnostic Abstract System Description	0..*	

Table 2.51: Develop Diagnostic Requirements

Activity	Integrate Diagnostic Information		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Diagnostic Extract Overview		
Brief Description			
Description	The main goal of this activity is to integrate all parts of the Diagnostic Description received from the OEM and from the application developer. Based on the complete Diagnostic Extract the initial ECUC can be generated.		
Relation Type	**Related Element**	**Mul.**	**Note**
Consumes	Diagnostic ECU Extract	0..*	partially filled DE:
Consumes	Diagnostic System Extract	0..*	
Produces	Diagnostic ECU Extract	1..*	complete DE:

Table 2.52: Integrate Diagnostic Information

2.13 Rapid Prototyping

2.13.1 Purpose

This use case describes usual activities to enable rapid prototyping in AUTOSAR.

2.13.2 Description

Rapid prototyping can be used during electronic control unit development to evaluate and test new software control algorithms for various functions.

With Fullpass technology the original ECU is totally replaced by a Rapid Prototyping Unit (RPU). With Bypass technology the original ECU and software stays in the control loop to supports the majority of the control algorithms and interface with sensors, actuators and communication buses: only the specific control algorithm that shall be prototyped is deported into the RPU (external bypass) or even directly executed in the original ECU (internal bypass). Bypass mainly consists in replacing at run time inputs and/or outputs of the original software algorithms by value computed by the prototype algorithm under test.

[TR_METH_01132] Definition of a Rapid Prototyping Scenario ⌈ In order to enable rapid prototyping, first of all the initial Rapid Prototyping Scenario is defined (task Define Rapid Prototyping Scenario). After the generation of the ECU Extract the ECU Extract of Rapid Prototyping Scenario should be refined to achieve a complete rapid prototyping scenario (task Refine Rapid Prototyping Scenario). ⌋*(RS_METH_00002)*

[TR_METH_01133] Content of Rapid Prototyping Scenario artifact ⌈ A RPT Scenario consist out of two main aspects: The description of the bypass points and the relation to a hook. A bypass point describes the required preparation of the host ECU. At a bypass point the host ECU shall be capable to communicate with a RPT system in order to support the execution of the rapid prototyping algorithms with the original data calculated by the host system and to replace dedicated results of the host system by the results of the rapid prototyping algorithm. The hook represents the link between a bypass point and the rapid prototyping algorithm.

Obviously, the bypass point and the hook reference aspects like parameterAccess (dataWriteAccess, dataReadAccess, dataSendPoint, dataReceivePointByValue, dataReceivePointByArgument, writtenLocalVariable, readLocalVariable). For more details see SW-C Template [6] (constr_2055). ⌋*(RS_METH_00002)*

Currently, AUTOSAR supports two approaches for Rapid Prototyping: Component wrapper method and direct buffer access method.

[TR_METH_01134] Component wrapper method ⌈ The component wrapper method consists in wrapping the original software component implementation with an integration code (Rapid Prototyping Wrapper Header File and Rapid Prototyping Wrapper Source Code) that implements the bypass. With this method the integration code is able to take the control of the AUTOSAR interfaces of the software component because there is no more direct call between RTE and the SW-C but everything go through the integration code.

In order to use this method, the RTE has to be configured properly (task Configure RTE, for configuration details see AUTOSAR_SWS_RTE [16], section 4.9.2). Furthermore, based on the complete ECU Extract of Rapid Prototyping Scenario artifact the corresponding wrapper code has to be generated and compiled (activity Encapsulate SW-C). Depending on the development strategy the wrapper code generation may be processed in different stages of the development process.

The RTE supports the component wrapper method by generating the SW-C interfaces with a c-namespace including an additional [Byps_] infix for the bypassed SW-C (task Generate RTE, for details see AUTOSAR_SWS_RTE [16], section 4.9.2). ⌋*(RS_METH_00006)*

[TR_METH_01135] Direct buffer access method ⌈ The direct buffer access method provides runtime direct read and write access to the RTE buffers that implement the ECU communication infrastructure. If the direct buffer access method for bypass support is enabled for a software component type, the Generate RTE task produces RTE Measurement and Calibration Support Data with mcDataAccessDetails for each preemption area specific buffer that implements the implicit communication for this software component type (For details see AUTOSAR_SWS_RTE [16], section 4.9.3). For this method no wrapper code has to be generated. ⌋*(RS_METH_00006)*

2.13.3 Workflow

Figure 2.68 shows the work sequence for this use case.

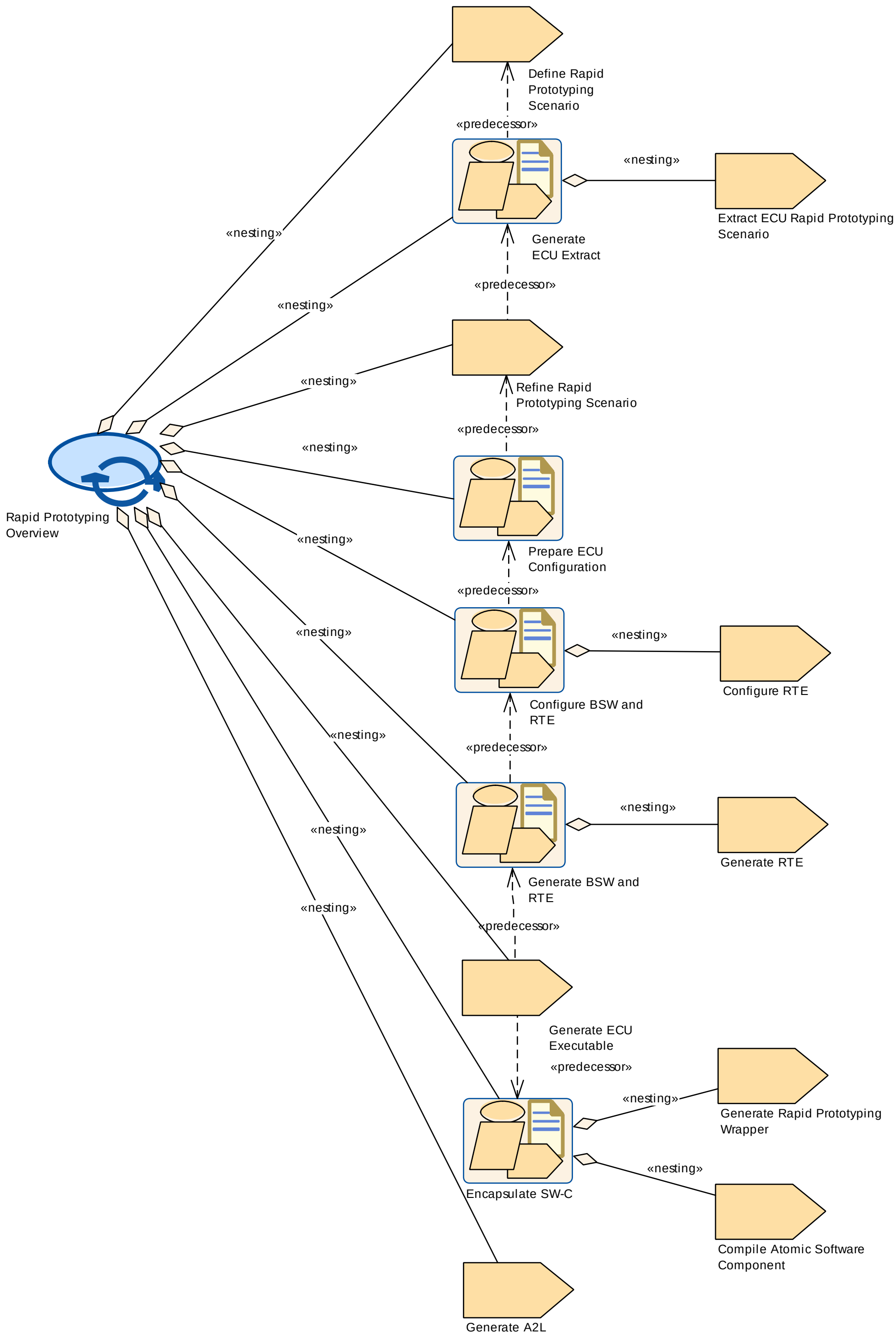

Figure 2.68: Rapid Prototyping Overview

Process Pattern	**Rapid Prototyping Overview**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Rapid Prototyping Overview		
Brief Description			
Description	This use case shows the typical steps required from an updated rapid prototyping scenario down to an update of the generated RTE and the produced A2L file. The use cases assumes, that rapid prototyping scenario is changed in an already existing system, thus the steps required to define and build a new system are omitted, only the calibration relevant steps are shown. In addition, the use case includes the (optional) task of updating a set of calibration parameter values as input for the RTE.		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregates	Configure BSW and RTE	1	
Aggregates	Define Rapid Prototyping Scenario	1	
Aggregates	Encapsulate SW-C	1	
Aggregates	Generate A2L	1	
Aggregates	Generate BSW and RTE	1	
Aggregates	Generate ECU Executable	1	
Aggregates	Generate ECU Extract	1	
Aggregates	Prepare ECU Configuration	1	
Aggregates	Refine Rapid Prototyping Scenario	1	

Table 2.53: Rapid Prototyping Overview

Activity	**Encapsulate SW-C**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Rapid Prototyping Overview		
Brief Description			
Description	Encapsulate the software component to enable rapid prototyping. During this activity the wrapper code is generated based on the Rapid Prototyping Scenario and the software component is compiled and linked with the generated wrapper.		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregates	Compile Atomic Software Component	1	
Aggregates	Generate Rapid Prototyping Wrapper	1	

Table 2.54: Encapsulate SW-C

2.14 Safety Extensions

2.14.1 Purpose

This use case provides an overview of the usage of the Safety Extensions (see [17]).

2.14.2 Description

ISO 26262 [18] is the applicable standard for functional safety of electronic and software based systems in road vehicles which impacts almost all development activities, including software specifications, design and implementation. The Safety Extensions enable a standardized exchange of the safety information in an AUTOSAR context and provide the basis for consistent management as required by ISO 26262. The additional safety related information can be used e.g. for generation of the documentation or the checking of ASIL constraints (w.r.t. allocation, mapping, decomposition and hierarchy), which are prescribed by the ISO 26262. The AUTOSAR Methodology focuses on the creation and refinement of the information. The corresponding analysis is out of scope of this document.

According to the ISO 26262, the Safety Extensions provide the following means to express safety information (for more details see TPS_SafetyExtension [17]):

- Safety Requirements (Artifact `Safety Requirement`)
- Safety Measures (Artifact `Safety Measure`)
- Safety integrity levels: attribute of `Safety Requirement`, `Safety Measure` and any AUTOSAR element
- Decomposition of Safety Requirements: reference between the original and the decomposed requirement (Task `Decompose Safety Requirement`)
- Refinement of Safety Requirements: reference between the original and the refined requirement (Task `Refine Safety Requirement`)
- Allocation of Safety Requirements: reference between of Safety Requirement and an AUTOSAR element (Task `Allocate Safety Requirement`)
- Allocation of Safety Measures: reference between Safety Measure and an AUTOSAR element (Task `Allocate Safety Measure`)
- Mapping between Safety Requirements and Safety Measures (Task `Map Safety Requirement to Safety Measure`)
- Independence relation between Safety Requirements (Task `Add Independence Relation`)

The safety relevant information can be exchanged independently and are therefore consolidated in a separate deliverable `Safety Extensions`.

[TR_METH_01144] Activity Define Safety Information ⌈ The activity Define Safety Information (see Figure 2.69) represents a generic pattern for defining safety relevant information. The safety extensions are not restricted to specific AUTOSAR elements so that safety relevant information can be added and modified in several stages of the AUTOSAR Methodology in an iterative way. Thus, the AUTOSAR elements consumed by some of the nested tasks are modeled using the General Autosar Artifact. The AUTOSAR Methodology does not prescribe an explicit execution order of the tasks. The only restrictions with respect to the execution order are given by the input and output relations (E.g. obviously, before a Safety Requirement can be decomposed, it has to be defined). ⌋*(RS_METH_00081)*

[TR_METH_01145] Creation of Safety Requirements ⌈ Naturally, the process starts with the task Define Safety Requirement. This task creates a Safety Requirement and assigns the required attributes such as ASIL. The top level Safety Requirement is a safety goal and obviously results from the hazard analysis and risk assessment. If Safety Requirements are not detailed enough to allocate them directly to appropriate AUTOSAR elements, it is necessary to refine them first (task Refine Safety Requirement). The refinement will add new Safety Requirements which are in a hierarchy relation to existing Safety Requirements. The ASIL is maintained as attribute at each safety goal and inherited consistently through the subsequent levels of functional safety requirements (as part of the Functional Safety Concept) and technical safety requirements (as part of the Technical Safety Concept). The latter will be refined into SW and HW safety requirements. ⌋*(RS_METH_00081)*

[TR_METH_01146] Allocation of Safety Requirements ⌈ Each Safety Requirement must be allocated properly to an element of the system architecture, i.e. component, HW, SW or both (HW and SW). Hence, an AUTOSAR element might receive an ASIL which indicates that it is in the scope of an ISO 26262 development. The allocation is done by task Allocate Safety Requirement. If safety requirements are not available or will not be exchanged together with a specification, the AUTOSAR implementation must at least be aware that the element is used in a safety context. Hence, the task Define ASIL For AUTOSAR Element directly assigns the ASIL attribute to an AUTOSAR element (independent of an allocation). Especially in cases of a SEooC (Safety Element out of Context) development, where the safety requirements are not fully known at development time, the ASIL attribute supports the integration and verification of such parts in a later stage of development by matching the assumptions against the finalized safety requirements. ⌋*(RS_METH_00081)*

[TR_METH_01147] Decomposition of Safety Requirements ⌈ In order to tailor the ASIL of Safety Requirements, ASIL decomposition may be applied. The decomposition is done by task Decompose Safety Requirement. According to the ISO 26262 a requirement can be decomposed into two requirements. In the context of ASIL decomposition the independence (freedom of interference) for the resulting requirements has to be demonstrated (Task Add Independence Relation). ⌋*(RS_METH_00081)*

[TR_METH_01148] Definition of Safety Measures ⌈ Safety of a system is achieved by means of safety measures that are applied at various stages of the development process and safety mechanisms which are implemented in a number of technologies into the system. Safety measures and safety mechanisms are represented by the artifact Safety Measure which is created by the task Define Safety Measure. In task Allocate Safety Measure the Safety Measures which are safety mechanisms realized in AUTOSAR are allocated to AUTOSAR elements in order to describe what elements are involved in the provision of a safety measure. The task Map Safety Requirement to Safety Measure creates a mapping between the Safety Measure and the Safety Requirement. ⌋*(RS_METH_00081)*

The following specialized activities demonstrate the usage of the Safety Extensions in different development stages and are integrated into the corresponding use cases:

- Define VFB Safety Information
- Define Software Component Safety Information
- Define System Safety Information

2.14.3 Workflow

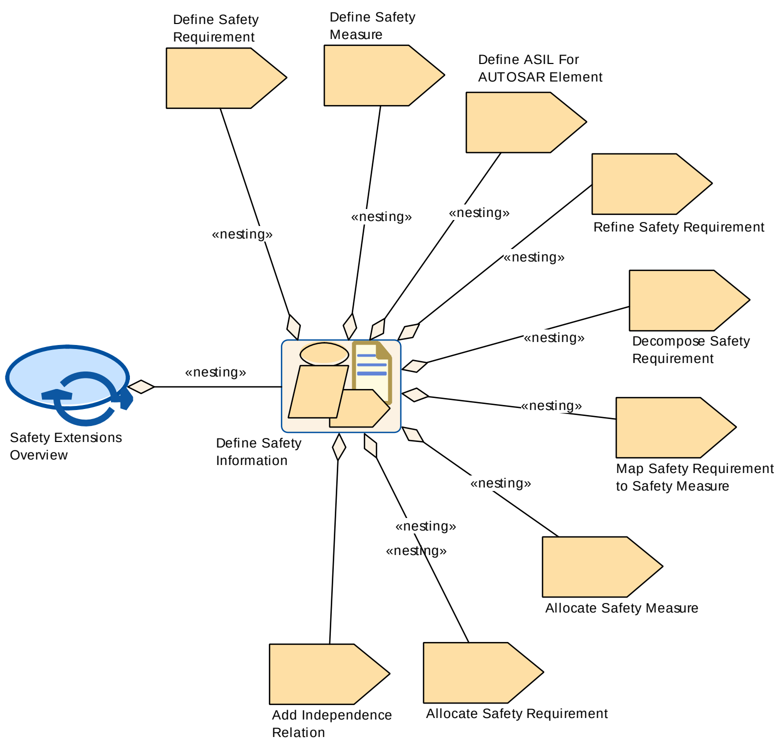

Figure 2.69: Safety Extensions Overview

Process Pattern	**Safety Extensions Overview**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Safety Extensions Overview		
Brief Description			
Description			
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Define Safety Information	1	

Table 2.55: Safety Extensions Overview

Activity	**Define Safety Information**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Safety Extensions Overview		
Brief Description	Defines all required safety information.		
Description	This activity represents a generic pattern for defining safety relevant information. The safety extensions are not restricted to specific AUTOSAR elements so that safety relevant information can be added and modified in several stages of the AUTOSAR Methodology. Thus, the AUTOSAR elements consumed by some of the nested tasks are modeled using the "General Autosar Artifact".		
Extended by	Define Software Component Safety Information, Define System Safety Information, Define VFB Safety Information		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Add Independence Relation	1	
Aggregates	Allocate Safety Measure	1	
Aggregates	Allocate Safety Requirement	1	
Aggregates	Decompose Safety Requirement	1	
Aggregates	Define ASIL For A UTOSAR Element	1	
Aggregates	Define Safety Measure	1	
Aggregates	Define Safety Requirement	1	
Aggregates	Map Safety Requirement to Safety Measure	1	
Aggregates	Refine Safety Requirement	1	

Table 2.56: Define Safety Information

2.15 Variant Handling

2.15.1 Overview

[TR_METH_02009] Variation points in Variant Handling ⌈ Variant Handling for AUTOSAR is defined in the Generic Structure Template Template [19]. First, this concept defines means to designate certain locations in the AUTOSAR meta-model as *variation points*. A point roughly consists of a condition (under which conditions is this variation active?) and a binding time (when should this variation be resolved?). ⌋*()*

Second, there are *predefined variants*.

[TR_METH_02010] Predefined variants in Variant Handling ⌈ A typical AUTOSAR model may contain a large number of variation points. However, usually only a relatively small number of variants (i.e., combinations of "active" variation points) is actively used. Each predefined variant describes such a variant. ⌋*()*

2.15.2 Binding Times

[TR_METH_02011] Types of binding times ⌈ The AUTOSAR variant handling defines two kinds of binding times for AUTOSAR: the *latest binding time* and the *actual binding time*. They have the same kinds of values[3], but are used in different contexts. ⌋*(RS_METH_00074)*

AUTOSAR defines the following binding times (presented here in chronological order):

- `BlueprintDerivationTime`
- `SystemDesignTime`
- `CodeGenerationTime`
- `PreCompileTime`
- `LinkTime`
- `PostBuild`

The Generic Structure Template mentions two more binding times. First, there is `FunctionDesignTime`, which comes before `SystemDesignTime`, but is independent of `BluePrintDerivationTime`. Second, there is `Runtime`, which comes after `PostBuild`. These binding times are not covered by AUTOSAR and mentioned here only for completeness.

[TR_METH_02012] Definition of a binding time ⌈ It should also be noted that a binding "time" is not really a point in time, but rather denotes a phase in the development of an AUTOSAR system. ⌋*(RS_METH_00074)*

[3] `BlueprintDerivationTime` and `PostBuild` are not part of the actual enum that is used in the meta-model, but they are implied by the structure of the variation point. See chapter 7 in the Generic Structure Template Template [19] are more details.

2.15.2.1 Latest Binding Time

[TR_METH_02013] Latest Binding Time ⌈ In the AUTOSAR meta model, every variation point has a latest binding time, which is implemented by the tag `Vh.LatestBindingTime`. As the name suggests, the latest binding time of a particular variation point puts an upper limit on *when* this point can be bound. A variation may be bound earlier than this time, but not later. ⌋*(RS_METH_00074)*

For example, the latest binding time for a software component which is part of a composition is `PostBuild`. In other words, an ECU can be configured to decide at startup whether a software component is active or not.

However, it is not always possible to bind a variant at the latest *possible* time. To continue the above example, making all software components `PostBuild` means that an executable always contains code and other resources for all software components, regardless whether it gets activated or not. Because of this, it may happen that the executable becomes too large to fit onto its designated ECU. If this is the case, the software component needs to be bound earlier, typically at PreCompileTime or even at SystemDesignTime.

This is not the only scenario that leads to this decision. For example, a software component might contain two or more subcomponents each of which is specific to a certain vendor. In this case, before delivering the software component to a specific vendor, it is custom to remove the subcomponents that are targeted at the other vendor(s). This can obviously be done at `PrecompileTime` the latest.

There are also cases where there is an implicit (i.e., not stated of the meta-model) lower limit for the binding time of a variation point. For example, if a variant in software component A uses a variant in software component B, then the binding times need to be coordinated. Component A cannot be `SystemDesignTime` if component B is `PostBuild`, but makes use of software component A.

2.15.2.2 Actual Binding Time

[TR_METH_02014] Actual Binding Time ⌈ This brings us to the actual binding time of a variation point, which is stored in an attribute[4] of the variation point. Again, it is not mandatory that the variation point is bound exactly at this stage; it rather states that the variation point must not be bound at a later stage.

This binding time may be earlier than the latest binding time. ⌋*(RS_METH_00074)*

As explained in the previous section, composition of software components can be bound at `PostBuild`, but it is not always desirable or even feasible to do so. In such a case. `bindingTime` should state an earlier binding time.

[4]The attribute is named `bindingTime` and is located at the `ConditionByformula` element of a variation point. For an `AttributeValueVariationPoint`, it is contained in the attribute `binding-Time`.

Also, unlike the latest binding time, which is a *meta model* element and is stated on M2 level, this binding time is a *model* element associated with a variation point and is stated on M1 level.

That is, the binding time of a variation point limits the point at which a *particular* variation point has to be bound, but this binding time is again constrained by the *latest binding time*.

2.15.3 Defining Variants

[TR_METH_02015] Definition of variants ⌈ A variant is almost always more than a single variant point or a single system constant. Typically, a variant is a list of value assignments to system constants or postbuild variant conditions. In an AUTOSAR model, such a list is represented by an instance of the meta-class `PredefinedVariant`, see definition of artifact `Predefined Variant`. ⌋*(RS_METH_00005)*

[TR_METH_02016] `Evaluated Variant Set` ⌈ Similarly, an instance of the meta-class `EvaluatedVariantSet` is a set of `PredefinedVariant`s that are known to work (or not to work) for a certain element of the meta-model, for example a specific software component. Evaluated variants may be used to exchange information about known variants between different vendors, for example to document which variants of a software component have been tested and are known to work.

In the Methodology SPEM model, the variant selectors are represented by the `Evaluated Variant Set` artifact which is created by the `Evaluate Variant` task. ⌋ *(RS_METH_00005, RS_METH_00075, RS_METH_00076)*

This information is necessary because there is a extremely high number of *possible* variants, but only a very small subset of them are feasible.

[TR_METH_02017] Use of `Predefined Variant` ⌈ The set of system constants that are contained in an instance of `PredefinedVariant` usually affect a number of variation points, which are at different locations in the model and have different binding times.

Hence, a predefined variant cannot be directly associated with a specific location in the meta-model, or a certain binding time. On the contrary, a `PredefinedVariant` is used for several meta-model elements and at different binding times. ⌋ *(RS_METH_00005, RS_METH_00076)*

2.15.4 Choosing Variants

Whether a variation point is included in a system or not is determined by one or more variables. If the binding time of a variation point is anywhere from `SystemDesignTime` to `LinkTime`, then the variation point contains an expression that is based on system constants (see artifact `System Constant Value Set`). If this expression evaluates to true, then the variation point is included in the system. `PostBuild` uses a simplified scheme that allows only a single comparison with a PostBuildVariantCriterion (technically, an ARElement).

[TR_METH_02018] Choosing variants ⌈ So, a variant is *chosen* as soon as the values for the respective system constants or postbuild variant conditions have been determined. This is usually done by selecting a PredefinedVariant, which contains the respective values. This selection must obviously happen before a variation point is bound. But, it does not need to happen *immediately* before a variation point is bound. ⌋*(RS_METH_00005)*

For example, the system constants that determine a PreCompileTime variation point may already have been chosen at SystemDesignTime, but the actual binding has to be delayed to PreCompileTime because of a dependency on another software components that have the binding time PreCompileTime, as described in Section 2.15.2.2.

Furthermore, since PredefinedVariant spans several variation points, which may have different binding times, some might have a binding time (latest or even actual) immediately after the PredefinedVariant has been chosen, and the others might have a later binding time.

Finally, the decision to go for a particular variant is often tied to vendor specific processes that follow their own timeline.

Hence, the time at which a particular variant is chosen is often not the same as the time when the associated variation points are bound. In summary, a variant must be chosen some time before it is bound, but the actual time when this is happening is not determined by AUTOSAR, and is also quite vendor specific.

2.16 Definition of Binding Times

2.16.1 Overview

A binding time is not (as the name probably suggests) a precise point in time, but rather a classification of processing steps. For example, the binding time CodeGenerationTime refers to a transformation step from an *AUTOSAR model* in ARXML format to *code*.

In this section, we define binding times for artifacts and tasks in the methodology.

[TR_METH_00001] Definition of Binding Time for Tasks ⌈ A task has binding time X if it binds variation points of binding time X.

This means in particular:

- Any task that works on the model *may* bind variation points that have the binding time SystemDesignTime.

- Any task that *generates* code needs to bind open variation points that have the binding time CodeGenerationTime. All variation points with earlier binding times must have been bound by then.
- Similarly, any task that *compiles* code needs to bind open variation points that have the binding time PreCompileTime.[5] All variation points with earlier binding times must have been bound by then.

  At this time, the *values* for PostBuildVariantConditions of variation points must also be bound. These values have a latest binding time of PreCompile-Time[6].

In all these cases, the system constants that are needed by the condition of a variation point obviously must be defined before the variation point is bound.

In the Methodology library, the binding time of a task is indicated by a value of the tag Meth.bindingTime for those tasks which *always* can be associated with this binding time. It is *not* indicated for tasks that only optionally bind variations. This typically is the case for all tasks that only work on the ARXML model, for example, it is up to the concrete process whether a task like Extract ECU Topology shall bind any variations. ⌋*(RS_METH_00074, RS_METH_00075)*

[TR_METH_00002] Definition of Binding Time for Artifacts ⌈ In an artifact with binding time X, all variation points up to binding time X shall be bound.

We do not denote such a binding time for artifacts in the Methodology library, because their binding time typically depends on the context. However, this definition could be used to assign a binding time to an artifact as part of a specific use case. ⌋ *(RS_METH_00074)*

[TR_METH_00003] Definition of Binding Time for Artifacts in the context of particular tasks ⌈ If an artifact of binding time X is used as input or output of a particular task, then all variation points *related to that task* with binding time up to X shall be bound.

This in particular means that if the artifact is input to the task, then binding time variation points X shall be bound and the task relies on this.

If the artifact is output to the task, it is granted that the such created artifact has all variation points of binding time X bound.

In the Methodology library, this is indicated by a value of the tag Meth.bindingTime attached to a Consumes/ConsumedBy resp. Produces/ProducedBy relationship.

[5] Note that in case of the RTE code, the technical step of binding PreCompileTime variants is partially done by a preparatory task which runs before the actual compilation, see Generate RTE Prebuild Dataset. That means in particular, the relevant system constants must be defined before executing this preparatory task. The binding time of actual compilation task Compile ECU Source Code is indicated as CompileTime in this case.

[6] The variation point is still PostBuild: the PostBuildVariantCondition is fixed at PreCompile-Time, but the comparison with the associated PostBuildVariantCriterion occurs at PostBuild-VariantCriterion. See the Generic Structure Template [19] for details

Note that the tag `Meth.bindingTime` is not applicable to `inout` relationships, as the binding time values according to the above definition are usually different for the inputs and outputs of a particular task. If it is important to express these binding times, the `inout` relation must be split into an input (i.e. `ConsumedBy`) and output (i.e. `Produces`) relation. ⌋*(RS_METH_00074)*

Figure 2.70 presents an overview of binding times as used in the AUTOSAR methodology. Boxed elements in this figure correspond to binding times, and the connections between them characterize artifacts.

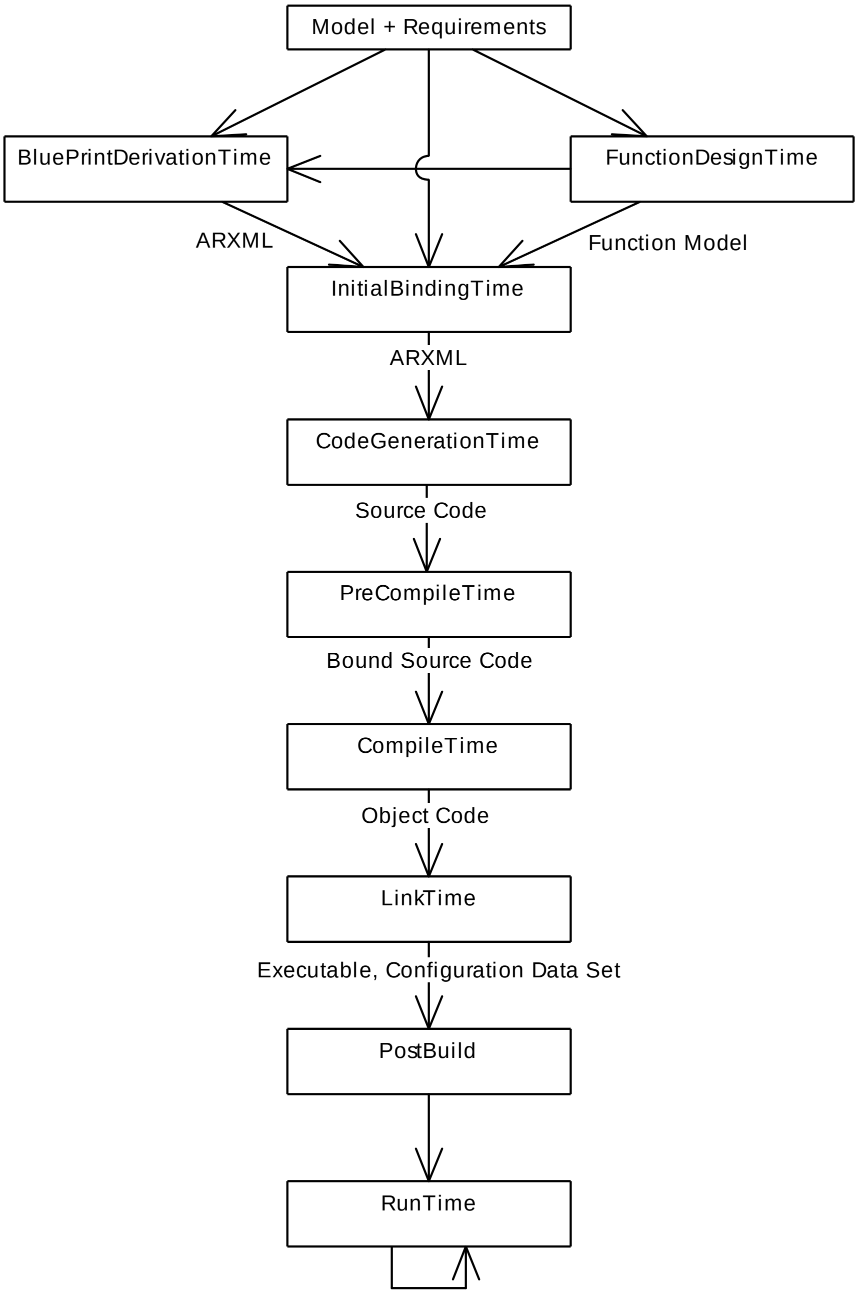

Figure 2.70: Overview of Binding Times

2.16.2 A Classification of Artifacts with respect to Binding Times

Model, Requirements, Functional Model These refer to models that are not an *AUTOSAR Model*. For example, a *Model* may be a Matlab/Simulink model or a requirements document.

ARXML An *ARXML* artifact is a XML document that conforms to the AUTOSAR XML schema.

Source Code A *Source Code* artifact is text written using the syntax of a programming language, for example such as C or C++.

Source Code may be generated by hand, or may be the output of a code generator.

Bound Source Code A *Bound Source Code* artifact contains source code without any unbound precompile variation points.

Object Code An *Object Code* is the output of a compiler. Object code is typically machine code, but may also include descriptive information in a format such as XML.

Executable An *Executable* is an artifact that can run on an ECU. It is often similar to *Object Code*; the difference between the two is that the former does not provide means for execution on an ECU.

Configuration Data Set A *Configuration Data Set* is a set of assignments to `PostBuildVariantCriterion`.

2.16.3 Classification of Binding Times

Table 2.57 presents an overview of the binding times in AUTOSAR Variant Handling.

Binding Time	AUTOSAR Metamodel	AUTOSAR Methodology
`BlueprintDerivationTime`	partially	yes
`FunctionDesignTime`	out of scope	out of scope
`InitialBindingTime`	no	yes
`SystemDesignTime`	yes	yes
`CodeGenerationTime`	yes	yes
`PreCompileTime`	yes	yes
`CompileTime`	unused	yes
`LinkTime`	yes	yes
`PostBuild`	yes	yes
`Runtime`	out of scope	out of scope

Table 2.57: Binding Times in Meta Model and Methodology

Variant handling in the AUTOSAR meta model supports the following binding times:

- `BlueprintDerivationTime`

- SystemDesignTime
- CodeGenerationTime
- PreCompileTime
- LinkTime
- PostBuild

[TR_METH_02020] Definition of latest Binding Time for a variation point in the meta-model ⌈ All these binding times may be used in the tag ≪Vh.latestBindingTime≫, which is used to define the latest binding time for a variation point in the meta model.

The actual binding time of a variation point is stored in the attribute bindingTime of the ConditionByFormula of a VariationPoint, and can only use the values SystemDesignTime, CodeGenerationTime, PreCompileTime, LinkTime. ⌋*(RS_METH_00074)*

The AUTOSAR methodology utilizes two more binding times, InitialBindingTimes to characterize artifacts where no variation points are bound, and CompileTime to distinguish between preprocessing and compiling of code. Finally, FunctionDesignTime and Runtime are not in the scope of AUTOSAR variant handling but mentioned here for completeness.

2.16.3.1 BlueprintDerivationTime

At BlueprintDerivationTime, a model is derived from Blueprints. For example, a function design tool provides the option to derive objects from a predefined set of blueprints. See [2] for more details. This is different from the variant handling defined in this chapter, but it uses the same meta model features (see [19]).

BlueprintDerivationTime is out of the scope of this document, but mentioned here for completeness.

Input Artifacts: Model, Requirements

Output Artifacts: ARXML

2.16.3.2 FunctionDesignTime

At FunctionDesignTime, a software architecture independent model for (control) systems is developed. Typical tools used at this stage are *Matlab/Simulink*, or *ASCET-MD*.

If a function design tool supports variant handling according to AUTOSAR it has no other choice than using CodeGenerationTime or later as binding time in the generated AUTOSAR artifacts.

`FunctionDesignTime` is out of the scope of this document (as long as it does not affect calibration measurements), but mentioned here for completeness.

Input Artifacts: Model, Requirements

Output Artifacts: Function model

2.16.3.3 InitialBindingTime

At InitialBindingTime, no variation points are bound. This binding time is needed to express a state where no SystemDesignTime points are bound in artifact

Input Artifacts: Model, Requirements, Function model, AUTOSAR models from blueprints in `ARXML` format.

Output Artifacts: `ARXML`.

2.16.3.4 SystemDesignTime

`SystemDesignTime` is characterized by the following tasks:

- Designing the VFB
- Software Component types (Interfaces)
- SWC Prototypes and the Connections between SWCprototypes
- Designing the Topology
- ECUs and interconnecting Networks
- Designing the Communication Matrix and Data Mapping

Input Artifacts: Function model, Requirements, AUTOSAR models from blueprints in `ARXML` format.

Output Artifacts: `ARXML`.

2.16.3.5 CodeGenerationTime

At this step, code is generated. This may be done either by hand, or using a tool, or a mixture of both.

Handwritten code is typically based on a requirements document, whereas generated code is usually created from a model that was designed at `FunctionDesignTime` or `SystemDesignTime`.

Both the requirements and the model may contain variants, but code is only generated for those variants that have been selected, or which need to be resolved later.

Input Artifacts: ARXML.

Output Artifacts: Source Code.

2.16.3.6 PreCompileTime

At PreCompileTime, a preprocessor (e.g., the C preprocessor) is used to further customize the code and exclude parts of the code from the compilation process.

There are several reasons for such an exclusion: code is not required for the selected variant(s), code is incompatible with the selected variant(s), or code requires resources that are not present in the selected variant(s). The code that is excluded at this stage code will not be available at later stages.

PreCompileTime is typically used for handwritten code (for which SystemDesignTime and CodeGenerationTime obviously cannot not take effect) or when a system constant needs to be bound after code generation.

Input Artifacts: Source Code.

Output Artifacts: Bound Source Code.

2.16.3.7 CompileTime

At CompileTime, source code that has already been processed by a macro processor such as the C preprocessor and stripped of all PreCompileTime variation points is transformed into object code. The compiler might eliminate further variants by removing unused code paths.

CompileTime is not used in the AUTOSAR meta model, but is used in the AUTOSAR methodology to discriminate between a preprocessor and a compiler.

Input Artifacts: Bound Source Code.

Output Artifacts: Object code.

2.16.3.8 LinkTime

The configuration at this stage determines which modules are included in the resulting object code (executable), and which ones are omitted based on the selected variants.

Input Artifacts: Object code.

Output Artifacts: Executable program.

2.16.3.9 PostBuild

PostBuild is the binding time which is bound latest at startup of the ECU. In other words this is everything between creation of the executable program and startup of the ECU.

The startup of the ECU is the PostBuild binding since and obviously cannot be resolved in the model.

Input Artifacts: Executable program, Configuration data set.

Output Artifacts: –

2.16.3.10 Runtime

Everything after startup and initialization is RunTime. Variant Handling at RunTime is out of the scope of this document, but mentioned here for completeness.

2.17 How to resolve Name Conflicts

2.17.1 Reasons for Name Conflicts

In the highly distributed development of an AUTOSAR system, there is a certain risk that symbolic names used in different development artifacts are not unique so that name conflicts may occur when applying software tools.

[TR_METH_03000] Name spaces via ARPackages ⌈ In the "upstream" specification of an AUTOSAR system, a software component, a basic software module or configuration parameters via AUTOSAR XML artifacts, such a risk can be widely avoided through the proper usage of ARPackages because they set up name spaces and may be nested (see also General Autosar Artifact). Here it is recommended to follow similar rules as AUTOSAR is using for its own published artifacts, see [19]: [TPS_GST_00081], [TPS_GST_00083], [TPS_GST_00086]. ⌋*(RS_METH_00002, RS_METH_00003, RS_METH_00004, RS_METH_00005)*

However, certain symbols specified in the AUTOSAR XML artifacts need to be transferred to other development artifacts in later process steps ("downstream") and will appear e.g. as symbols in C-code, as file names, as names displayed by calibration tools or in textual documents. Here we have in general two reasons for naming conflicts (which may also occur in combination):

[TR_METH_03001] Reasons for name conflicts in "downstream" artifacts ⌈

- **Uncoordinated co-development**

  Due to the global name space of the C-language within one compilation unit, the risk of name conflicts is rather high if pieces of source code are integrated that were developed by different parties without coordinating the definition of symbols.

The same can happen with names of header files or with symbols visible by the linker.

In AUTOSAR, the programming language interfaces between software components and (to some extend) between basic software modules are restricted to certain patterns and are generated from ARXML, so the coordination effort is restricted to the proper definition of the relevant symbols in ARXML.

In several cases the `shortName` of an `ARElement` corresponds to an identifier in the code (or to a part of such an identifier), sometimes also to a file name or a part of it. Since `shortName`s are also used in the links between ARXML elements, it is hard to change such a name without impact on the overall design. This is for example the case for the names of the `AtomicSwComponentType`s.

- **Multiple instantiation**

  The AUTOSAR Runtime Environment (RTE) supports multiple instantiation of software components. This means, in a system and even on one ECU there can be several instances of a given `AtomicSwComponentType`. Each instance possesses its own data (managed by the RTE), but there is only one artifact (`VFB Atomic Software Component`) describing the whole type. If one needs a symbol identifying a particular component instance or particular data belonging to that instance (for example for display in a calibration tool), a conflict arises.

  A similar thing happens with data elements or operation arguments in a `PortInterface` or in a composite data type, if the enclosing element is reused in more than one context.

  A different kind of "multiple instantiation" can occur in the basic software, if several driver modules implement the same interface (only distinguished by an instance identifier). In this case, we actually have different implementations of code, the modules only share the upper levels of description (artifacts `Basic Software Module Description` and `Basic Software Module Internal Behavior`).

⌋(*RS_METH_00038*)

2.17.2 Points in the Methodology where Name Conflicts are resolved

On the other hand we have multiple points in the methodology where to resolve those conflicts.

In general we can distinguish between the development phase in which a name conflict is resolved and the phase in which it occurs (or would occur). Because a conflict usually prevents a certain task from being completed (e.g. compilation), it must be resolved in the same or an earlier phase than the phase in which it would occur.

- **[TR_METH_03002] Conflict solution at system design time** ⌈
  This is mentioned mainly for completeness. Of course, a proper system design

can avoid conflicts in the first place and if a name conflict still arises in a later phase, it is in principle possible to iterate over the system design. But in this chapter we focus on solutions that allow to resolve name conflicts in later process phases which usually causes less effort.⌋(*RS_METH_00006*)

- **[TR_METH_03003] Conflict solution at coding time ⌈**
  Conflicts occurring at compile time or link time must be resolved (latest) at the time a developer is producing the code and/or the ARXML descriptions leading to the generation of code. In other words, this has to happen within the activities `Develop an Atomic Software Component` or `Develop BSW Module`. Note that in the worst case, such a conflict is detected not before integration time (during activity `Build Executable`) which means that some kind of iteration of the activities is required.⌋(*RS_METH_00006*)

- **[TR_METH_03004] Conflict solution at ECU integration time ⌈**
  During ECU integration time (latest) it is still possible to resolve name conflicts that would occur in tasks after the software build, e.g. during generation of A2L files.⌋(*RS_METH_00006*)

2.17.3 Mechanisms for resolving Name Conflicts

The mechanisms to resolve the name conflicts are:

- **[TR_METH_03005] Conflict solution via `SymbolProps` ⌈**

  This mechanism allows to redefine a name in cases where the `shortName` by default is used to generate RTE relevant code. This avoids to change the overall design in the ARXML model.

  This mechanism can be applied at coding time (activity `Develop an Atomic Software Component`, task `Define SymbolProps for Types`) and solves conflicts caused by uncoordinated development. Such changes - even if they do not influence the overall design of the software - should be agreed upon by the involved parties.

  This mechanism is provided for the following meta-model elements:

  `AtomicSwComponentType.symbolProps`
  Allows to redefine the software component type name that the RTE is using in its code. This resolves name clashes among different software component types designed accidentally with the same `shortName`.[7]

  `ImplementationDataType.symbolProps`
  Allows to redefine the implementation data type name used in the code of the

[7]Note that this mechanism is not applicable for the prefixes used in the preprocessor code of memory sections and compiler memory classes. Conflicts among these preprocessor symbols due to duplicate component type names are not visible to the linker. However conflicts might occur when compiling the header file Compiler_Cfg.h and must be resolved manually.

RTE and/or the components. This resolves name clashes among different implementation data types designed accidentally with the same `shortName`.

For more information on the meta-model refer to [TPS_SWCT_01194] and [TPS_SWCT_01110] in [6].⌋*(RS_METH_00002)*

- **[TR_METH_03006] Conflict solution via literal prefixes** ⌈

  This mechanisms is similar to the one described before. It allows to define a prefix for preprocessor literals (e.g. for enumeration types or upper/lower limits) created by the RTE generator contract phase. Also this mechanism solves conflicts caused by uncoordinated development and must be applied at coding time (part of task `Define Atomic Software Component Internal Behavior`).

  The model element to be manipulated is:
  `SwcInternalBehavior.includedDataTypeSet.literalPrefix`

  For more information refer to [TPS_SWCT_01157] in [6].⌋*(RS_METH_00002)*

- **[TR_METH_03007] Conflict solution in names of runnable entities** ⌈

  In case of a `RunnableEntity` the symbol used in the code is already independent from the `shortName` - it is always modeled via the attribute `RunnableEntity.symbol`. However, since these symbols need to be unique in the scope of one RTE instance (see [constr_2025] in [6]), also here a name conflict can occur at integration time if the definition of the symbols was not coordinated before.

  Similar to the cases discussed before, this conflict must be solved at coding time simply be changing the symbol. Note that such a change would not influence the overall design and can be done locally on one component (whose runnable shall be renamed) since the runnable symbol is hidden to other component by the RTE. Despite of that, the definition of unique runnable symbols still might need some human coordination.⌋*(RS_METH_00002)*

- **[TR_METH_03008] Conflict solution via `FlatMap`** ⌈

  This mechanism allows to assign identifiers to instances of model elements (e.g. software component instances or data element instances) so that they are unique in a certain scope, e.g. a system or an ECU. Thereby name conflicts are avoided, which would occur if simply the `shortName`s of the ARXML elements would be used. In other words, this mechanisms solves the name conflicts arising from multiple instantiation of types in the ARXML model.

  The identifiers defined in this way are typically not used within the code, since AUTOSAR components do not rely on global variables. The main purpose is the usage within other artifacts which need to handle symbols out of the package context of the ARXML model, for example citation in documents (e.g. in artifact `Software Component Documentation`) or input for measurement and calibration tools (e.g. in artifact `RTE Measurement and Calibration Support Data`). A special use case of the `ECU Flat Map` is the the model transformation from the System to ECU Extract, where it is used to define additional names of component prototypes.

The point in the methodology where this mechanisms is applied depends of course on the use case. The typical tasks in the methodology library for defining a Flat Map are normally performed before integration time: Generate or Adjust System Flat Map, Define Partial Flat Map and Generate or Adjust ECU Flat Map. But since identifiers in a FlatMap are independent of the code, it can in principle be adjusted even at integration time in case a conflict occurs.

For more information see artifacts System Flat Map, Partial Flat Map and ECU Flat Map, for the underlying meta-model parts refer to refer to [7].⌋*(RS_METH_00005)*

- **[TR_METH_03009] Conflict solution via AliasNameSet** ⌈

  This mechanism is similar to FlatMap. It allows to define additional names for model elements, either on top of an entry in a FlatMap or standalone. The usage is also similar, but there are no standardized use cases in connection with the AUTOSAR RTE. It can be used in cases where the format of the FlatMap is too restrictive.

  For more information refer to the artifact Alias Name Set and task Define Alias Names. For the meta-model of AliasNameSet refer to [7]. The document [7] also gives recommendations on how to transfer certain attributes below AliasNameSet into an ASAM ASAP2 (“A2L”) specification.⌋*()*

- **[TR_METH_03010] Conflict solution via API Infixes** ⌈

  If several “instances” of a basic software module (with different implementation but identical interface definition) are linked together, name conflicts have to be solved by defining “infixes”. These are small pieces of strings denoting the module vendor and the instance role. They are used to extend globally visible C symbols and certain header file names. The mechanism is also relevant for the basic software scheduler APIs generated in task Generate BSWM Contract Header Files.

  Though this mechanism solves a conflict of a certain kind of multiple instantiation, it is relevant to the code and thus must be applied at coding time. The description of the infixes has to be put into the artifact Basic Software Module Implementation Description.

  For more information refer to [TPS_BSWMDT_04031] in [10] and to [SWS_BSW_00102] in [8].⌋*(RS_METH_00003)*

2.18 Data Exchange Points

2.18.1 Purpose

Profile of Data Exchange Points intend to improve the interoperability between tools by describing which data is expected for a given activity or task in the methodology. This use case describes the main steps to create these profiles as well as how they are used in the further development.

2.18.2 Description

Complex supply chains with several different organizations, partners and tools are involved in the overall automotive system development. Real life projects have shown that the exchange of data between stakeholders and tools often doesn't work properly. Interoperability issues are often detected in late development phases and typically require analysis and adaptations of tools in each individual project.

The most severe issues are related to insufficient definition and harmonization of data exchange points: It is often unclear, which data is required and which constraints apply at what time in the overall methodology.
To overcome this difficulty, AUTOSAR introduces the detailed description of data exchange points as one step of the development process. It is important to note that this step is not modeled explicitly in the overall view use case in chapter 2.1, but can rather be executed whenever an exchange of data may result in incompatibilities. Major exchange points are e.g. given by the domain boundaries as seen in Figure 2.9.
See [20] for more information on interoperability of AUTOSAR tools. The detailed approach of data exchange points with corresponsing meta classes is described in [2]. In [21] an example for the ECU System Description as data exchange point is given.

[TR_METH_01202] Create a Profile of Data Exchange Point ⌈ There are several approaches of creating Profile of Data Exchange Points, each of them is based on a Baseline Profile of Data Exchange Points, which defines default values, e.g.:

- Analyze Autosar Specifications or General Non Autosar Artifacts such as tool documentation, documentation of previously used exchange formats (e.g. FIBEX, DBC, LDF) or company-specific documents describing contents for exchange artifacts in order to identify which parts of the Autosar standard and the Autosar data exchange format are relevant at the specific Data Exchange Point. This step may be started "from scratch" or may be based on an existing Profile of Data Exchange Point. Autosar could provide predefined Profile of Data Exchange Points for typical Data Exchange Points which may be refined if necessary in individual projects.
- Analyze existing General Autosar Artifacts (Autosar models) that are typically exchanged at a specific Data Exchange Point. This can help to figure out, which parts of the meta model are currently used. E.g. if the General

Autosar Artifact contains FlexrayClusters the derived Profile of Data Exchange Point might consider FlexrayClusters as relevant information. Note that this analysis can only deal as a first hint. Engineering know-how is required in order to identify if information is missing or if information was in the model that is not relevant for the specific Data Exchange Point.

⌋*(RS_METH_00083, RS_METH_00084)*

[TR_METH_01204] Agreement on a profile for data exchange points ⌈ An important step in the development process is to agree on a specific profile. Several Profile of Data Exchange Points might serve as an input. For example, profiles of different tools need to be agreed on in order to reduce the risk of tool interoperability issues. Therefore, common parts of the Profile of Data Exchange Points as well as potential interoperability issues need to be identified and a compromise need to be found. If an agreement on a profile exists, this profile can be used for validation purposes. ⌋*(RS_METH_00083, RS_METH_00084)*

[TR_METH_01205] Validation based on an Agreed Profile of Data Exchange Point ⌈ Once there is an agreement on a profile, this can be used to continuously validate if the General Autosar Artifacts (Autosar models)s still conform with the Agreed Profile of Data Exchange Point. This can e.g. help to unveil unintended modifications of the Autosar models because of changed tool configurations or tool updates. ⌋*(RS_METH_00083, RS_METH_00084)*

2.18.3 Workflow

Figure 2.71 shows the work sequence for this use case.

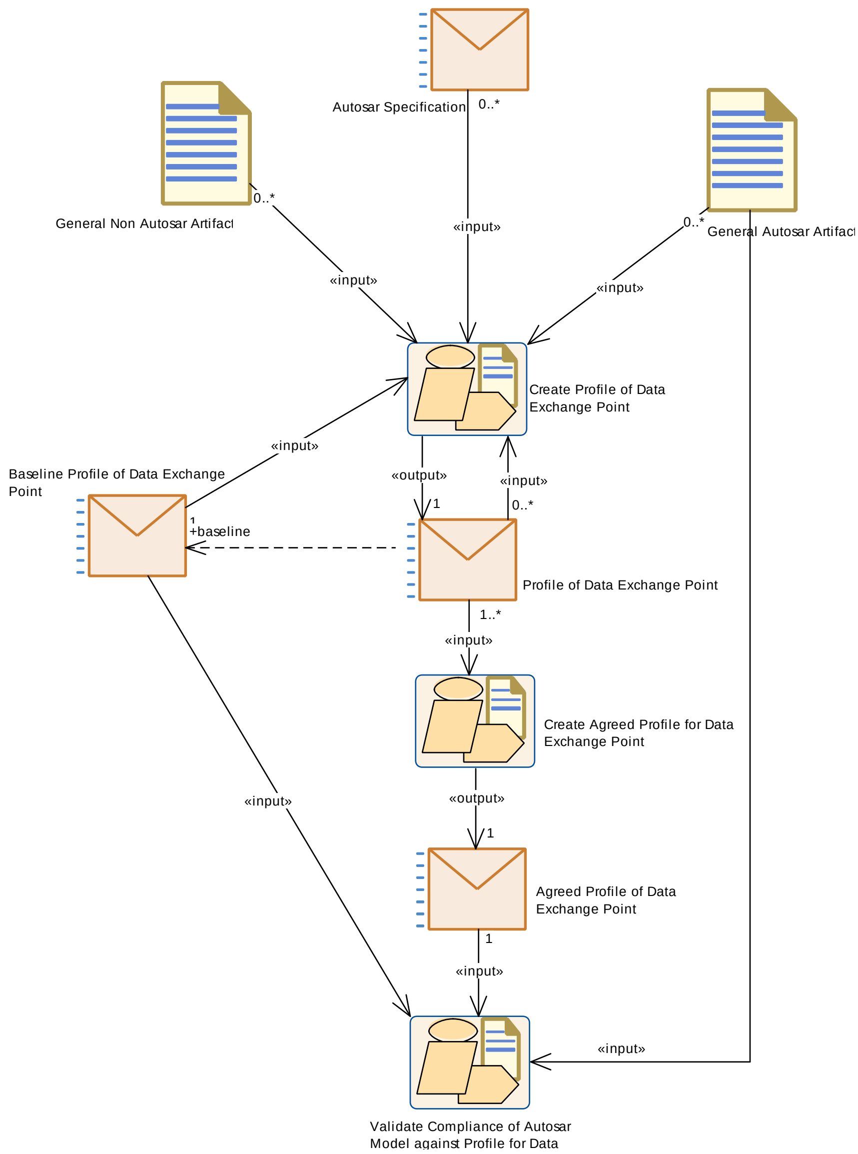

Figure 2.71: Data Exchange Point Workflow

Activity	**Create Profile of Data Exchange Point**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Data Exchange Point Overview		
Brief Description	Create a Profile of Data Exchange Point		
Description	There are several approaches of creating Profiles of Data Exchange Points, each of them is based on a Baseline Profile of Data Exchange Points, which defines default values, e.g.: (1) Analyze Autosar specifications or General Non-Autosar Artifacts such as tool documentation, documentation of previously used exchange formats (e.g. FIBEX, DBC, LDF) or company-specific documents describing contents for exchange artifacts in order to identify which parts of the Autosar specifications and Autosar templates are relevant at the specific Data Exchange Point. This step could be started "from Scratch" or based on an existing Profile of Data Exchange Point. Autosar could provide predefined Profiles of Data Exchange Point for typical exchange scenarios which could be refined if necessary in individual projects. (2) Analyze existing General Autosar Artifacts (Autosar models) that are typically exchanged at a specific Data Exchange Point. This can help to figure out, which parts of the meta model are currently used. E.g. if the General Autosar Artifact contains FlexrayClusters the derived Profile of Data Exchange Point should consider FlexrayClusters as relevant information. Note, that this analysis can only deal as a first hint. Engineer Know How is required in order to identify if information is missing or of information was in the model that is not relevant for the specific Data Exchange Point. In order to reduce the size of a Profile of Data Exchange Point it should only describe information that is not already provided by referenced Autosar standardized Baseline Profile of Data Exchange Point.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Autosar Specification	0..*	The Autosar specifications that are analyzed in order to figure out which data is required for the given data exchange point.
Consumes	Baseline Profile of Data Exchange Point	1	The Baseline Profile of Data Exchange Point defines the default values of the Profile.
Consumes	General Autosar Artifact	0..*	The analysis of existing Autosar models that are typically exchanged at a specific Data Exchange Point can help during the development of a Profile of Data Exchange Point.
Consumes	General Non Autosar Artifact	0..*	Custom specification for data exchange (textual or non-AUTOSAR exchange format (e.g. FIBEX, DBC, LDF))
Consumes	Profile of Data Exchange Point	0..*	An existing Profile of Data Exchange Point that is used as a starting point for a new Profile of Data Exchange Point. E.g. start with a profile of an old tool version and tailor it so that new features are covered.

Relation Type	Related Element	Mul.	Note
Produces	Profile of Data Exchange Point	1	The new Profile of Data Exchange Point

Table 2.58: Create Profile of Data Exchange Point

Activity	**Create Agreed Profile for Data Exchange Point**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Data Exchange Point Overview		
Brief Description	Analysis of input Profiles of Data Exchange Point with respect to tool interoperability issues and documentation of the results as an Agreed Profile of Data Exchange Point.		
Description	In early phases of a project the Profiles of Data Exchange Point describe the requirements and assertions of the involved tools. These Profiles are then analyzed. Aspects that are common on all Profiles have low risk of interoperability issues and can directly be copied into the Agreed Profile of Data Exchange Point. If the Profiles are different, potential interoperability issues need to be investigated and an agreement needs to be developed. This is a highly manual step that requires negotiation between the stakeholders.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Profile of Data Exchange Point	1..*	The Profiles of Data Exchange Point that are analyzed with respect to interoperability issues.
Produces	Agreed Profile of Data Exchange Point	1	The Agreed Profile of Data Exchange Point

Table 2.59: Create Agreed Profile for Data Exchange Point

Activity	**Validate Compliance of Autosar Model against Profile for Data Exchange Point**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::High Level::Data Exchange Point Overview		
Brief Description	Validate the compliance of AUTOSAR models against a Profile for Data Exchange Point.		
Description	Once there is an agreement on a profile, this can be used to continuously validate if the exchanged General Autosar Artifgacts (Autosar models) still conform with the Agreed Profile of Data Exchange Point. This can e.g. help to unveil unintended modifications of the Autosar models because of changed tool configurations or tool updates.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Agreed Profile of Data Exchange Point	1	The DataFormatTailoring and Baseline sections of the Agreed Profile of Data Exchange Point can be used to configure the validation engine of Autosar Validation Tools (e.g. provide rules for existence of elements or switch on/off individual semantic constraints)

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Baseline Profile of Data Exchange Point	1	The Baseline Profile of Data Exchange Point provides default values foe parameters that are not specified in the Agreed Profile of Data Exchange Point.
Consumes	General Autosar Artifact	1	An Autosar model can be validated against the Agreed Profile for Data Exchange Point. Default values of attributes that are not defined in the Agreed Profile of Data Exchange Point are retrieved from the Baseline Profile of Data Exchange Point.

Table 2.60: Validate Compliance of Autosar Model against Profile for Data Exchange Point

3 Methodology Library

3.1 Common Elements

This chapter contains the definition of work products and tasks used in several areas of AUTOSAR development. For the definition of the relevant meta-model elements refer to [19].

3.1.1 Work Product Kinds

Category (Work Product Kind)	**AUTOSAR XML**
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Product Kinds
Brief Description	
Description	An artifact that conforms to the AUTOSAR XML schema.

Table 3.1: AUTOSAR XML

Category (Work Product Kind)	**Source Code**
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Product Kinds
Brief Description	
Description	A human readable artifact that conforms to a defined programming language syntax, such as C or Java.

Table 3.2: Source Code

Category (Work Product Kind)	**Bound Source Code**
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Product Kinds
Brief Description	
Description	A Bound Source Code artifact contains source code without any unbound precompile variation points.

Table 3.3: Bound Source Code

Category (Work Product Kind)	**Object Code**
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Product Kinds
Brief Description	

Description	An Object Code is the output of a compiler. Object code is typically machine code, but may also include descriptive information in a format such as XML.

Table 3.4: Object Code

Category (Work Product Kind)	**Configuration Data Set**
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Product Kinds
Brief Description	
Description	This is a special kind of binary code containing configuration that can be loaded separately from the main ECU code.

Table 3.5: Configuration Data Set

Category (Work Product Kind)	**Executable**
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Product Kinds
Brief Description	
Description	An Executable is an artifact that can run on an ECU. It is often similar to Object Code; the difference between the two is that the former does not provide means for execution on an ECU.

Table 3.6: Executable

Category (Work Product Kind)	**Text**
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Product Kinds
Brief Description	
Description	A human readable artifact that is stored as plain text, rich text, PDF, etc.

Table 3.7: Text

Category (Work Product Kind)	**Custom**
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Product Kinds
Brief Description	
Description	A custom artifact format which is not further specified in the AUTOSAR Methodology.

Table 3.8: Custom

Category (Work Product Kind)	**Delivered**
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Product Kinds
Brief Description	
Description	These are collections of delivered work products. They form the basis of exchange between organizations.

Table 3.9: Delivered

3.1.2 Tasks

3.1.2.1 Add General Documentation

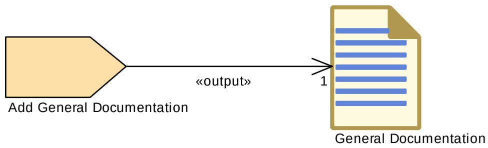

Figure 3.1: Add General Documentation

Task Definition	**Add General Documentation**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Tasks		
Brief Description			
Description	Add General Documentation to work products (AR_MET_REQ069)		
Relation Type	***Related Element***	***Mul.***	***Note***
Produces	General Documentation	1	

Table 3.10: Add General Documentation

3.1.2.2 Define Admin Data

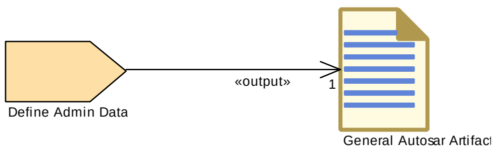

Figure 3.2: Define Admin Data

Task Definition	**Define Admin Data**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Tasks		
Brief Description	Generic task to define admin data of an Identifiable within an AUTOSAR artifact.		
Description	Generic task to define administration data (metamodel element AdminData) of an Identifiable within an AUTOSAR artifact. Note that administration data can be defined on several levels, namely for the top-level package of a General Autosar Artifact, but also for sub-packages and for other Identifiables within the XML description. Admininistration data include versioning information of the model element via the meta-class DocRevision, and the aggretation of user specific data via so-called special data groups, meta-class Sdg. For more details on the administration data content see AUTOSAR_TPS_GenericStructureTemplate.pdf.		
Relation Type	*Related Element*	*Mul.*	*Note*
Produces	General Autosar Artifact	1	

Table 3.11: Define Admin Data

3.1.2.3 Define Alias Names

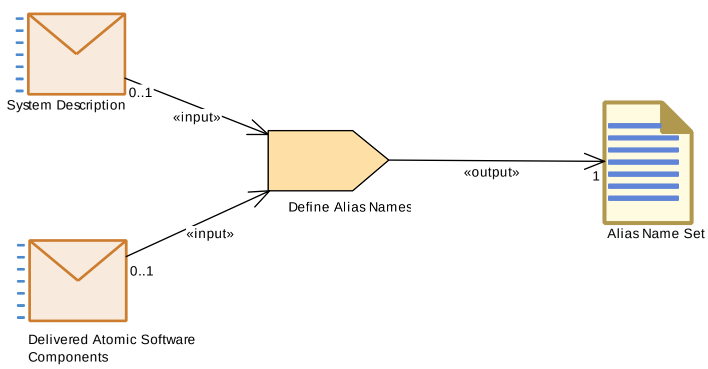

Figure 3.3: Define Alias Names

Task Definition	**Define Alias Names**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Tasks		
Brief Description	Define a set of alias names for AUTOSAR model elements.		
Description	The usual mechanism for defining global names for nested elements within an AUTOSAR XML model is the Flat Map. However in the cooperation with non-AUTOSAR tools, there are uses cases which require additional alias names which can be defined by this task. It can be applied on System and on ECU level as well. Possible use cases are for example: • The names defined by an ECU Flat Map, System Flat Map or Partial Flat Map shall be superseded when used by an external tool (e.g. in order to use a more general string format). • Resolve name conflicts for elements which cannot be referred in the context of a Flat Map (e.g. for elements directly defined in the scope of ARPackages, like System Constants to be displayed by A2L tools).		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Delivered Atomic Software Components	0..1	Needed for definition of alias names in the scope of delivered software components.
Consumes	System Description	0..1	Needed for definition of alias names with system, system extract or ECU scope, depending of the role of the System Description.
Produces	Alias Name Set	1	

Table 3.12: Define Alias Names

3.1.2.4 Evaluate Variant

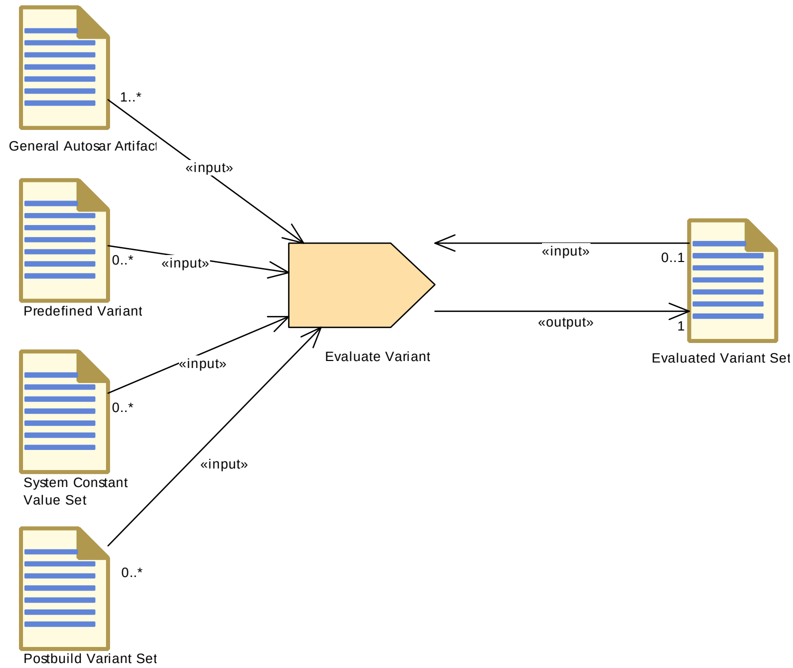

Figure 3.4: Evaluate Variant

Task Definition	**Evaluate Variant**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Tasks		
Brief Description	Document the evaluation of variants in the software description.		
Description	Create or modify an Evaluated Variant Set in order to document the outcome of an evaluation of particular variants. This namely means setting the "approval status" in relation to a given set of PredefinedVariants and a given set of model elements (e.g. a particular Software Component) which were evaluated. This is a general task which can be applied on different levels, therefore the input is modeled as General Autosar Artifact.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	General Autosar Artifact	1..*	
Consumes	Evaluated Variant Set	0..1	
Consumes	Postbuild Variant Set	0..*	
Consumes	Predefined Variant	0..*	
Consumes	System Constant Value Set	0..*	
Produces	Evaluated Variant Set	1	

Table 3.13: Evaluate Variant

3.1.2.5 Define Memory Addressing Modes

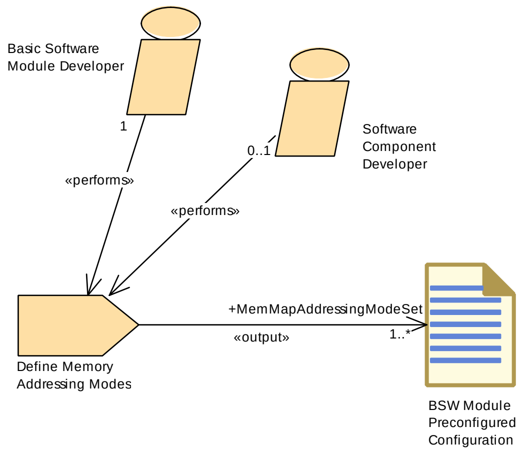

Figure 3.5: Define Memory Addressing Modes

Task Definition	**Define Memory Addressing Modes**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Tasks		
Brief Description			
Description	Define the compiler specific configuration used in a later task to generate the "pragmas" in memory mapping header files. The output (container MemMapAddressingModeSet) is treated as pre-configured configuration values for the "module" MemMap, because it can be prepared independently from the configuration for a specific integration project. Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Basic Software Module Developer	1	
Performed by	Software Component Developer	0..1	
Produces	BSW Module Preconfigured Configuration	1..*	MemMapAddressingModeSet: Meth.bindingTime = SystemDesignTime

Table 3.14: Define Memory Addressing Modes

3.1.2.6 Configure Memmap Allocation

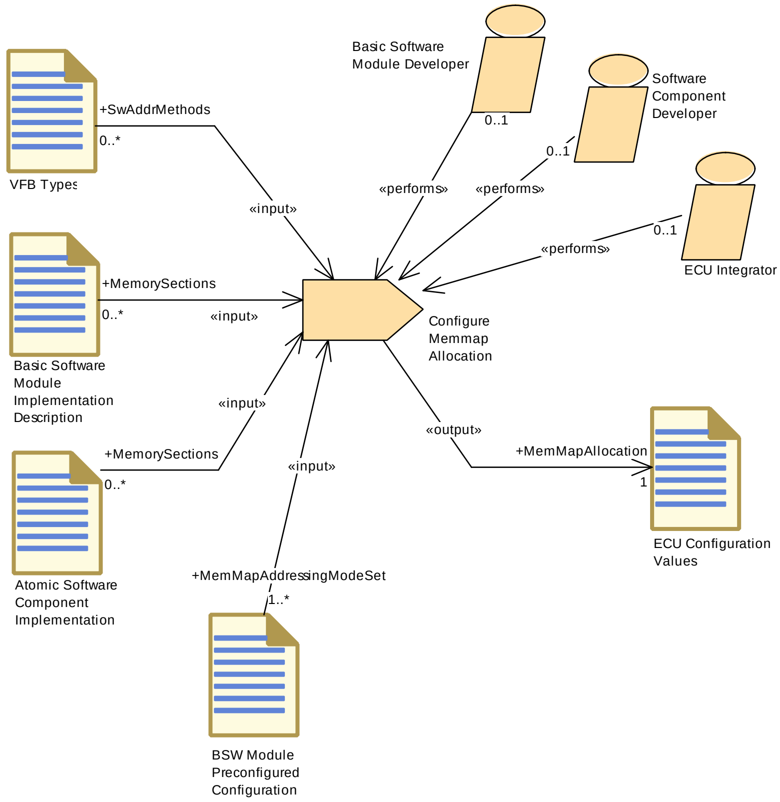

Figure 3.6: Configure Memmap Allocation

Task Definition	Configure Memmap Allocation		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Tasks		
Brief Description			
Description	Configure the ECU Configuration part MemMapAllocation for module "MemMap". The output is to be used for generating memory mapping headers during ECU integration as well as for BSW and SWC compiling/linking in local environments. MemMapAllocation defines a mapping between abstract memory sections used in BSW or SWC code and compiler specific configuration elements. The abstract sections are identified via links to SwAddrmethods (generic mapping) resp. MemorySections of the XML input files. The compiler specific configuration is given as a pre-configured configuration for module "MemMap" via the container MemMapAddressingModeSet. For more information refer to document ID 128: SWS_MemoryMapping. Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Basic Software Module Developer	0..1	
Performed by	ECU Integrator	0..1	
Performed by	Software Component Developer	0..1	
Consumes	BSW Module Preconfigured Configuration	1..*	MemMapAddressingModeSet: Collection of compiler specific configuration elements for memory allocation and addressing modes.
Consumes	Atomic Software Component Implementation	0..*	MemorySections:
Consumes	Basic Software Module Implementation Description	0..*	MemorySections:
Consumes	VFB Types	0..*	SwAddrMethods: SwAddrMethods used for the generic mapping. Note that one SwAddrmethod can represent several memory sections.
Produces	ECU Configuration Values	1	MemMapAllocation: Meth.bindingTime = SystemDesignTime

Table 3.15: Configure Memmap Allocation

3.1.2.7 Generate BSW Memory Mapping Header

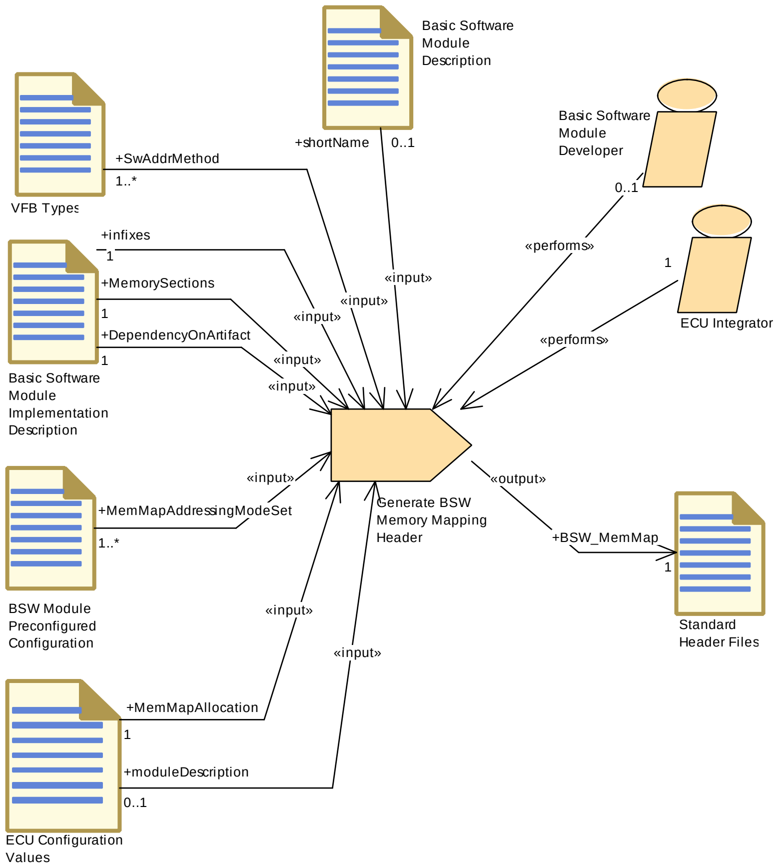

Figure 3.7: Generate BSW Memory Mapping Header

<table>
<tr><td>Task Definition</td><td colspan="3">Generate BSW Memory Mapping Header</td></tr>
<tr><td>Package</td><td colspan="3">AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Tasks</td></tr>
<tr><td>Brief Description</td><td colspan="3"></td></tr>
<tr><td>Description</td><td colspan="3">Generate a memory mapping header to be used for one BSW module (the default case) or a group of BSW modules (e.g. an ICC2 cluster). Note that the usage of one MemMap.h for the complete BSW of one build environment is possible, but deprecated.

This task can be used in ECU scope or with preliminary scope to test BSW modules. Note that the content of the generated file is compiler specific (#pragma statements).

Inputs are:
From Basic Software Module Description: The shortName is used (in the default case) as the first part of the generated file name.
From VFB Types: Properties of abstract sections given by SwAddrmethods, which in turn are referred by MemorySection as well as by MemMapAllocation.
From Basic Software Module Implementation Description: Names of the individual abstract sections (preprocessor macros) used in the code (including optional prefixes overriding the default rule); optional infixes for the file name (if the default rule is used); optional declaration of file name (element DependencyOnArtifact) overriding the default rule.
From Preconfigured Configuration for module "MemMap": Collection of compiler specific configuration elements.
From ECU Configuration for module "MemMap" : MemMapAllocation - this is the concrete mapping for this environment.
From ECU Configuration: Find the list of used BSW modules in case the task is done for the whole BSW (EcucValueCollection.ecucValue.moduleDescription).
Meth.bindingTime = CodeGenerationTime</td></tr>
<tr><td>Relation Type</td><td>Related Element</td><td>Mul.</td><td>Note</td></tr>
<tr><td>Performed by</td><td>ECU Integrator</td><td>1</td><td></td></tr>
<tr><td>Performed by</td><td>Basic Software Module Developer</td><td>0..1</td><td></td></tr>
<tr><td>Consumes</td><td>Basic Software Module Implementation Description</td><td>1</td><td>DependencyOnArtifact: Can be used to override the default name of the memory mapping header file.
Meth.bindingTime = SystemDesignTime</td></tr>
<tr><td>Consumes</td><td>Basic Software Module Implementation Description</td><td>1</td><td>MemorySections: MemorySections defined for a BSW module. This input includes optional prefixes for memory sections overriding the default rule.
Meth.bindingTime = SystemDesignTime</td></tr>
</table>

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Basic Software Module Implementation Description	1	infixes: Optional infixes (denoting instance and vendor ID) to be used within the created header file name. Meth.bindingTime = SystemDesignTime
Consumes	ECU Configuration Values	1	MemMapAllocation: Mapping of the abstract sections (SwAddressMethods for generic mapping resp. MemorySection Elements for specific mapping) to the compiler specific MemMapAddressingModes. Meth.bindingTime = SystemDesignTime
Consumes	BSW Module Preconfigured Configuration	1..*	MemMapAddressingModeSet: Collection of compiler specific configuration elements for memory allocation. Meth.bindingTime = SystemDesignTime
Consumes	VFB Types	1..*	SwAddrMethod: Referred SwAddrMethods Meth.bindingTime = SystemDesignTime
Consumes	Basic Software Module Description	0..1	shortName: The BSW module's shortName is used as the first part of the generated file name, in case the default rule applies. Meth.bindingTime = SystemDesignTime
Consumes	ECU Configuration Values	0..1	moduleDescription: List of used BSW modules (EcucValueCollection.ecucValue.moduleDescription) Meth.bindingTime = SystemDesignTime
Produces	Standard Header Files	1	BSW_MemMap: The memory mapping header file to be used for one or more BSW modules in a given build environment. The file name has in the standardized case a form like {Mip}_MemMap.h in which the prefixes {Mip} are determined by the module (or cluster) name and optional infixes. However, it is also possible to create a completely different filename via explicit declaration in the BSW Module Implementation. For more detailed rules on the name of the generated file refer to AUTOSAR_SWS_MemoryMapping. Meth.bindingTime = CodeGenerationTime

Table 3.16: Generate BSW Memory Mapping Header

3.1.2.8 Generate SWC Memory Mapping Header

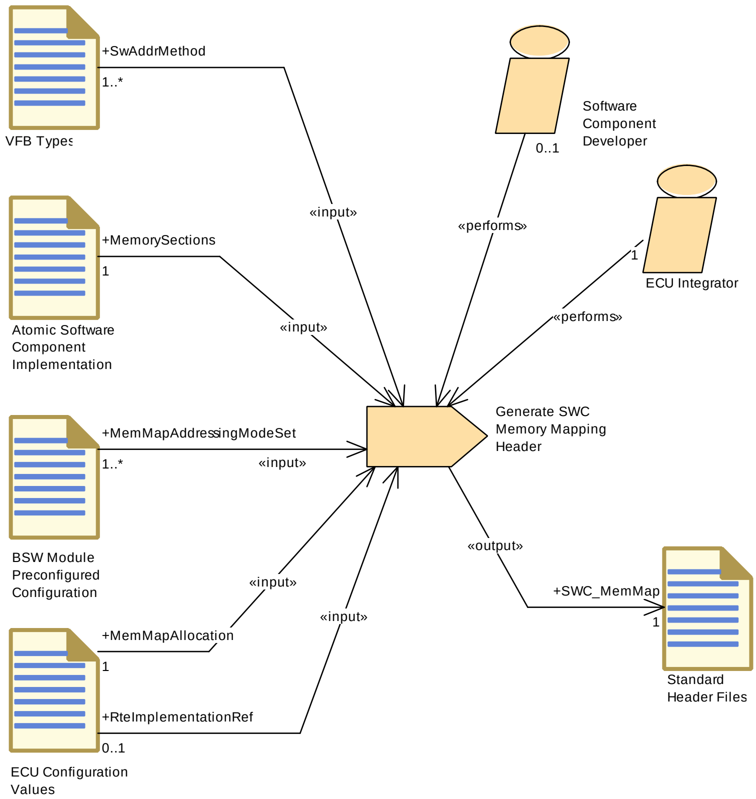

Figure 3.8: Generate SWC Memory Mapping Header

Task Definition	**Generate SWC Memory Mapping Header**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Tasks		
Brief Description			
Description	Generate the memory mapping header file for one build environment and one Atomic Software Component. This task can be used in ECU scope or with preliminary scope to test software component. Note that the generated header file is compiler specific (#pragma statements). Inputs are: • From VFB Types: Properties of abstract sections given by SwAddrmethods, which in turn are referred by MemorySection as well as by MemMapAllocation • From Software Component Implementation, element MemorySection: Names of the individual abstract sections (preprocessor macros) used in the code. • From Preconfigured Configuration for module "MemMap": Collection of compiler specific configuration elements. • From ECU Configuration for module "MemMap" : MemMapAllocation - This is the concrete mapping for this environment. • From ECU Configuration: Find (optionally) the list of used software component implementations by usage of the RTE ECU Configuration "RteSwComponentType.RteImplementationRef" Meth.bindingTime = CodeGenerationTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Performed by	Software Component Developer	0..1	
Consumes	Atomic Software Component Implementation	1	MemorySections: MemorySections defined for an Atomic Software Component. Meth.bindingTime = SystemDesignTime
Consumes	ECU Configuration Values	1	MemMapAllocation: Mapipng of the abstract sections (SwAddressMethods for generic mapping resp. MemorySection Elements for specific mapping) to the compiler specific MemMapAddressingModes. Meth.bindingTime = SystemDesignTime
Consumes	BSW Module Preconfigured Configuration	1..*	MemMapAddressingModeSet: Collection of compiler specific configuration elements for memory allocation. Meth.bindingTime = SystemDesignTime
Consumes	VFB Types	1..*	SwAddrMethod: Referred SwAddrMethods Meth.bindingTime = SystemDesignTime

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	ECU Configuration Values	0..1	RteImplementationRef: Existence of SWCs could be identified by usage of the RTE ECU Configuration "RteSwComponentType.RteImplementationRef" Meth.bindingTime = SystemDesignTime
Produces	Standard Header Files	1	SWC_MemMap: One header per software component type for a given build environment. The file name follows the pattern {componentTypeName}_MemMap.h in which the prefix componentTypeName is determined by the software component type name. For more detailed rules on the name of the generated file refer to AUTOSAR_SWS_MemoryMapping. Meth.bindingTime = CodeGenerationTime

Table 3.17: Generate SWC Memory Mapping Header

3.1.2.9 Configure Compiler Memory Classes

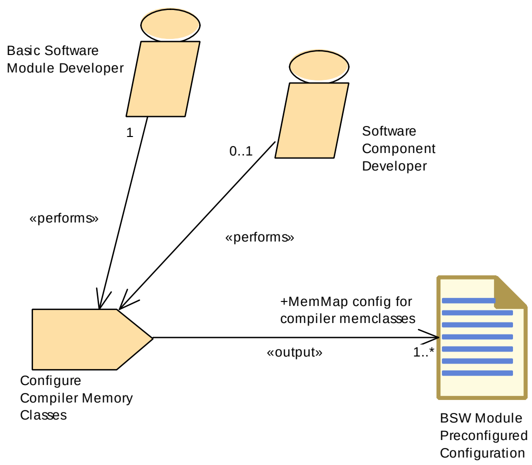

Figure 3.9: Define Compiler Memory Classes

Task Definition	**Configure Compiler Memory Classes**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Tasks		
Brief Description			
Description	Define the compiler specific configuration for "memory classes" used in a later task to generate the preprocessor code of the compiler configuration header file (Compiler_Cfg.h). The output is treated as pre-configured configuration values for the "module" MemMap, because it can be prepared independently from the configuration for a specific integration project. Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Basic Software Module Developer	1	
Performed by	Software Component Developer	0..1	
Produces	BSW Module Pre-configured Configuration	1..*	MemMap config for compiler memclasses: Set the parameter values that define generic MemClassSymbols (i.e. those not defined by modules or SWCs.). Set the parameter values that define the implementation behind all kind of MemClassSymbols (generic and local ones). Meth.bindingTime = SystemDesignTime

Table 3.18: Configure Compiler Memory Classes

3.1.2.10 Generate Compiler Configuration

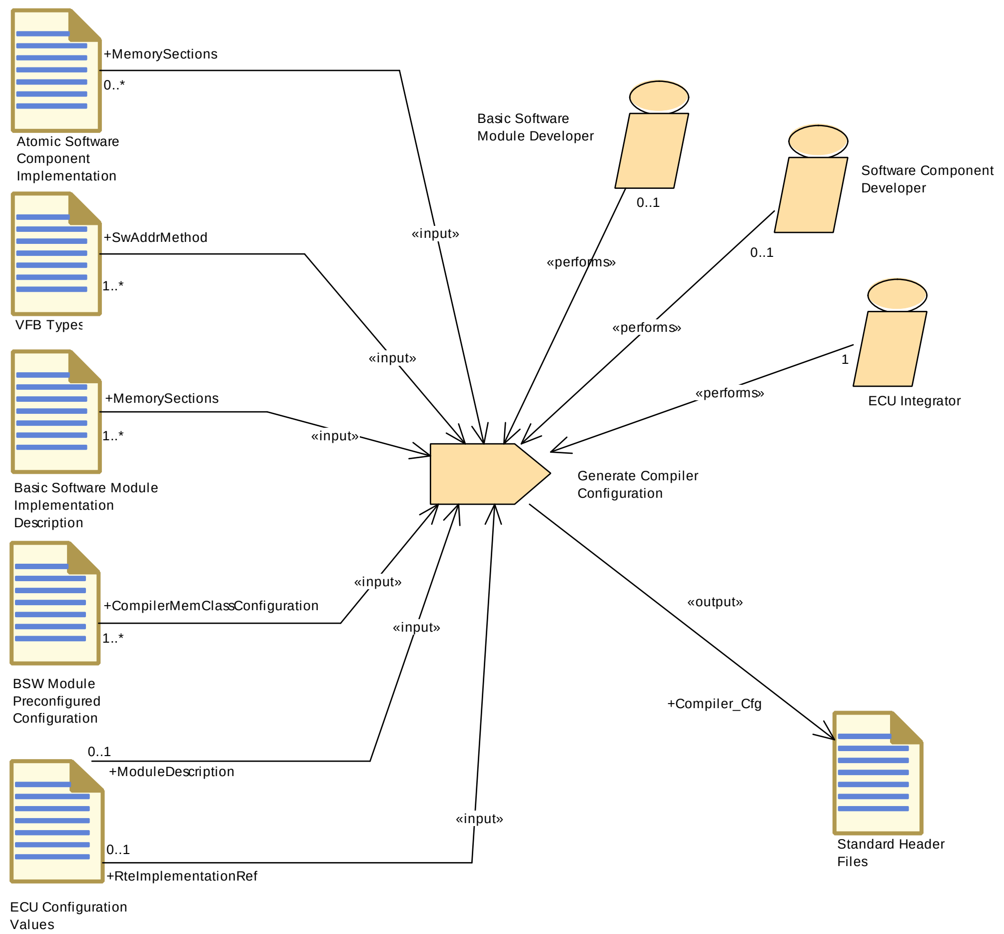

Figure 3.10: Generate Compiler Configuration

Task Definition	**Generate Compiler Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Tasks		
Brief Description			
Description	This task generates a compiler configuration header (Compiler_cfg.h) for one build environment to be used for all BSW modules and software components. Meth.bindingTime = CodeGenerationTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Performed by	Basic Software Module Developer	0..1	
Performed by	Software Component Developer	0..1	

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	BSW Module Pre-configured Configuration	1..*	CompilerMemClassConfiguration: The parameters "MemMapCompilerMemClassSymbolImpl" and "MemMapGenericCompilerMemClassSymbolImpl" define the implementation behind a MemClassSymbol. Meth.bindingTime = SystemDesignTime
Consumes	Basic Software Module Implementation Description	1..*	MemorySections: Find referred SwAddrMethods or specific memClassSymbols in the MemorySections defined for BSW modules. Meth.bindingTime = SystemDesignTime
Consumes	VFB Types	1..*	SwAddrMethod: Referred SwAddrMethods. They provide the default names for the compiler memory classes. Meth.bindingTime = SystemDesignTime
Consumes	ECU Configuration Values	0..1	ModuleDescription: List of used BSW modules (EcucValueCollection.ecucValue.moduleDescription) Meth.bindingTime = SystemDesignTime
Consumes	ECU Configuration Values	0..1	RteImplementationRef: Existence of SWCs could be identified by usage of the RTE ECU Configuration "RteSwComponentType.RteImplementationRef" Meth.bindingTime = SystemDesignTime
Consumes	Atomic Software Component Implementation	0..*	MemorySections: Find referred SwAddrMethods or specific memClassSymbols in the MemorySections defined for Atomic Software Components. Meth.bindingTime = SystemDesignTime
Produces	Standard Header Files	1	Compiler_Cfg: The output file "Compiler_Cfg.h" configures the abstraction of compiler specifics. Meth.bindingTime = CodeGenerationTime

Table 3.19: Generate Compiler Configuration

3.1.3 Work Products

3.1.3.1 General Documentation

Artifact	**General Documentation**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Products		
Brief Description			
Description	General documentation link to a given work product		
Kind	Custom		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	General Deliverable	0..*	
Produced by	Add General Documentation	1	

Table 3.20: General Documentation

3.1.3.2 Alias Name Set

Artifact	**Alias Name Set**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Products		
Brief Description	Set of alias names for AUTOSAR model elements for usage outside of AUTOSAR.		
Description	Set of alias names, each consisting of the name (string) itself and the reference to the model element it renames. Each reference to a model element is either a reference to an Identifiable or to an entry in an ECU Flat Map or System Flat Map. For an explanation of uses cases see task Define Alias Names.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Delivered Atomic Software Components	0..1	Alias names valid in the context of the delivered components.
Aggregated by	System Description	0..*	
Produced by	Define Alias Names	1	
Consumed by	Add Documentation to the Software Component	0..*	Optional input in order to refer to unique names defined in an Alias Name Set (e.g. System Constants).
Consumed by	Generate A2L	0..*	
Use meta model element	AliasNameSet	1	

Table 3.21: Alias Name Set

3.1.3.3 Evaluated Variant Set

Artifact	**Evaluated Variant Set**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Products		
Brief Description	A set of evaluated variants		
Description	This artifact represents a table defining which ArElements or ArPackages (referrred as "evaluatedElements") are able to support one or more particular variant. It can thus be used to document which variants are support by a certain delivery, e.g. of a software component or of a system. In other words, for a given set of evaluatedElements this element represents a table of evaluated variants, where each PredefinedVariant represents one column. In this column each descendant swSystemConstantValue (part of System Constant Value Set) resp. postbuildVariantCriterionValue (part of Postbuid Variant Set) represents one entry. In a graphical representation each swSystemConstantValueSet / postBuildVariantCriterionValueSet could be used as an intermediate headline in the table column. The Evaluated Variant Set comes with an attribute "approvalStatus". If this is set to "APPROVED" it expresses that the evaluatedElements are known be valid for the given evaluated variants. Note that an evaluatedElement could be another Evaluated Variant Set. This allows to establish a hierarchy of EvaluatedVariantSets.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Delivered Atomic Software Components	0..1	
Aggregated by	ECU Extract of System Variant Model	0..*	
Aggregated by	System Description	0..*	
Aggregated by	VFB System	0..*	
Produced by	Define System Variants	1	
Produced by	Evaluate Variant	1	
Produced by	Define Integration Variant	0..1	Meth.bindingTime = SystemDesignTime
Produced by	Define VFB Variants	0..*	
Consumed by	Evaluate Variant	0..1	
Consumed by	Extract ECU System Variant Model	0..*	
Use meta model element	EvaluatedVariant Set	1	

Table 3.22: Evaluated Variant Set

3.1.3.4 Autosar Specification

Deliverable	**Autosar Specification**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Products		
Brief Description			
Description	An Autosar specification that is part of the Autosar standard. E.g. Software Component Template, Main Requirements, Autosar Model Constraints, Specification of Communication, etc.		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Create Profile of Data Exchange Point	0..*	The Autosar specifications that are analyzed in order to figure out which data is required for the given data exchange point.

Table 3.23: Autosar Specification

3.1.3.5 General Autosar Artifact

Artifact	**General Autosar Artifact**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Products		
Brief Description	Describes the meta data for an AUTOSAR artifact.		
Description	This artifact represents the data which are common to all AUTOSAR XML artifacts. Each file starts with the root element AUTOSAR. The content of such an artifact below this root element is organized by packages using the element ARPackage. Packages can be nested. It is important to understand, that the hierarchy defined via packages and other aggregated elements can (in general) span over several XML files, i.e. over several artifacts. That means, if an aggregation is "split" between several files, each file is considered as a separate artifact by the methodology, even if the elements are formally aggregated within the same package. All elements derived from meta-class Identifiable can carry documentation and administrative description based on the element AdminData. Note that ARPackage is itself derived from Identifiable, so there can be AdminData for the top-level package, for sub-packages and for more specific elements (derived from Identifiable) as well. The AdminData among other things contain revision information (including the artifact version) based on the metamodel element DocRevision .		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	General Deliverable	0..*	
Produced by	Define ASIL For A UTOSAR Element	1	

Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define Admin Data	1	
Produced by	Allocate Safety Measure	0..*	Allocated Elements:
Produced by	Allocate Safety Requirement	0..*	Allocated Elements:
Consumed by	Define ASIL For AUTOSAR Element	1	
Consumed by	Validate Compliance of Autosar Model against Profile for Data Exchange Point	1	An Autosar model can be validated against the Agreed Profile for Data Exchange Point. Default values of attributes that are not defined in the Agreed Profile of Data Exchange Point are retrieved from the Baseline Profile of Data Exchange Point.
Consumed by	Allocate Safety Measure	1..*	
Consumed by	Allocate Safety Requirement	1..*	
Consumed by	Evaluate Variant	1..*	
Consumed by	Create Profile of Data Exchange Point	0..*	The analysis of existing Autosar models that are typically exchanged at a specific Data Exchange Point can help during the development of a Profile of Data Exchange Point.
Consumed by	Define Safety Measure	0..*	
Consumed by	Define Safety Requirement	0..*	
Use meta model element	ARPackage	1	
Use meta model element	AUTOSAR	1	

Table 3.24: General Autosar Artifact

3.1.3.6 General Deliverable

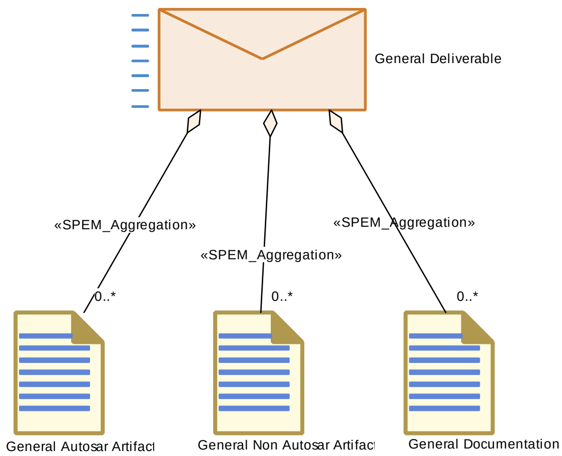

Figure 3.11: General Deliverable

Deliverable	**General Deliverable**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Products		
Brief Description	General data for an XML based deliverable within AUTOSAR.		
Description	General data for an XML based deliverable within AUTOSAR : Especially it contains a catalog of all included artifacts. These can be AUTOSAR artifacts (see General Autosar Artifact) or non-AUTOSAR artifacts (see General Non AUTOSAR Artifact). An AUTOSAR XML artifact which is contained in the catalog may refer to an non AUTOSAR Artifact whithin the catalog via the metamodel element AutosarEngineeringObject (see AUTOSAR_TPS_GenericStructureTemplate.pdf for further description).		
Kind	Delivered		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	General Autosar Artifact	0..*	
Aggregates	General Documentation	0..*	
Aggregates	General Non Autosar Artifact	0..*	

Table 3.25: General Deliverable

3.1.3.7 General Non-Autosar Artifact

Artifact	**General Non Autosar Artifact**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Products		
Brief Description	Describes the data for a non AUTOSAR artifact.		
Description	Describes the data for a non AUTOSAR artifact.		
Kind	Custom		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	General Deliverable	0..*	
Consumed by	Provide RTE Calibration Dataset	1..*	input from calibration process
Consumed by	Create Profile of Data Exchange Point	0..*	Custom specification for data exchange (textual or non-AUTOSAR exchange format (e.g. FIBEX, DBC, LDF))

Table 3.26: General Non Autosar Artifact

3.1.3.8 Postbuild Variant Set

Artifact	**Postbuild Variant Set**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Products		
Brief Description	Set of Postbuild Variant Criterion Values used to define post-build variants of the software.		
Description	Set of Postbuild Variant Criterion Values used to define post-build variants of the software. Such a set does not necessarily define a variant which is actually used. To define a meaningful variant in the production process, such a set is to be used via reference by artifact PredefinedVariant.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	Delivered Atomic Software Components	0..1	
Aggregated by	ECU Extract of System Variant Model	0..*	
Aggregated by	System Description	0..*	
Aggregated by	VFB System	0..*	
In/out	Define System Variants	1	
In/out	Define Integration Variant	0..*	
In/out	Define VFB Variants	0..*	
Consumed by	Generate RTE Postbuild Dataset	1	
Consumed by	Generate Atomic Software Component Contract Header Files	0..1	
Consumed by	Generate RTE Prebuild Dataset	0..1	
Consumed by	Evaluate Variant	0..*	
Consumed by	Extract ECU System Variant Model	0..*	
Use meta model element	PostBuildVariant CriterionValueSet	1	

Table 3.27: Postbuild Variant Set

3.1.3.9 Predefined Variant

Artifact	**Predefined Variant**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Products		
Brief Description	Defines a variant predefined for usage in subsequent process steps.		
Description	Defines one variant of a software description for delivery and/or usage in subsequent process steps. The actual definition of all settings which make up this variant is given by attached System Constant Value Set (all settings which are resolved prior to post-build) and/or Postbuid Variant Set (all settings which are resolved after software build). These sets may be part of the same artifact or may be separated artifacts. Via these settings, the actual values which make up a particular variant, are selected.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	Delivered Atomic Software Components	0..*	
Aggregated by	ECU Extract of System Variant Model	0..*	
Aggregated by	System Description	0..*	
Aggregated by	VFB System	0..*	
Produced by	Define Integration Variant	1	Meth.bindingTime = SystemDesignTime
Produced by	Define System Variants	1	
Produced by	Define VFB Variants	0..*	
Consumed by	Generate BSW Module Prebuild Data Set	1	
Consumed by	Generate RTE Postbuild Dataset	1	
Consumed by	Generate RTE Prebuild Dataset	1	
Consumed by	Generate Atomic Software Component Contract Header Files	0..1	
Consumed by	Evaluate Variant	0..*	
Consumed by	Extract ECU System Variant Model	0..*	
Consumed by	Generate Component Prebuild Data Set	0..*	
Use meta model element	PredefinedVariant	1	

Table 3.28: Predefined Variant

3.1.3.10 Standard Header Files

Artifact	**Standard Header Files**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Products		
Brief Description	Overall header files to be included by each standardized BSW module, optionally also by Software Component code.		
Description	Overall header files to be included by each standardized BSW module, optionally also by Software Component code. For simplicity of the methodology, these are modeled as one artifact though in practice these are several different files: • (<prefixes>_)MemMap.h - defines a common set of macros in order to define abstract memory sections for code and data in the source code . The prefixes indicates whether the scope is limited to a component, module or some other source code area (e.g. an ICC2 cluster). Note that the usage of one MemMap.h for the complete BSW is possible, but deprecated. It is also possible to use a completely different filename via explicit declaration in the BSW Module Implementation Description. • Std_Types.h - defines a common set of C data types for usage within the basic software, this header includes the following two headers: • Compiler.h (in turn including Compiler_Cfg.h) - for abstraction of compiler specifics, in which the second header is the part that is subject to configuration • Platform_Types.h - for abstraction of platform specific types		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Generate BSW Memory Mapping Header	1	BSW_MemMap: The memory mapping header file to be used for one or more BSW modules in a given build environment. The file name has in the standardized case a form like {Mip}_MemMap.h in which the prefixes {Mip} are determined by the module (or cluster) name and optional infixes. However, it is also possible to create a completely different filename via explicit declaration in the BSW Module Implementation. For more detailed rules on the name of the generated file refer to AUTOSAR_SWS_MemoryMapping. Meth.bindingTime = CodeGenerationTime

Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Generate Compiler Configuration	1	Compiler_Cfg: The output file "Compiler_Cfg.h" configures the abstraction of compiler specifics. Meth.bindingTime = CodeGenerationTime
Produced by	Generate SWC Memory Mapping Header	1	SWC_MemMap: One header per software component type for a given build environment. The file name follows the pattern {componentTypeName}_MemMap.h in which the prefix componentTypeName is determined by the software component type name. For more detailed rules on the name of the generated file refer to AUTOSAR_SWS_MemoryMapping. Meth.bindingTime = CodeGenerationTime
Consumed by	Compile Atomic Software Component	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile BSW Core Code	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile ECU Source Code	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Implement a BSW Module	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Re-compile Component in ECU context	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Implement Atomic Software Component	0..1	Meth.bindingTime = CodeGenerationTime

Table 3.29: Standard Header Files

3.1.3.11 System Constant Value Set

Artifact	**System Constant Value Set**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Work Products		
Brief Description	Set of System Constant Values used to handle variants.		
Description	Set of System Constant Values used to define pre-build variants of the software. Such a set does not necessarily define a variant which is actually used. To define a meaningful variant in the production process, such a set is to be used via reference by artifact PredefinedVariant.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	Delivered Atomic Software Components	0..*	
Aggregated by	ECU Extract of System Variant Model	0..*	
Aggregated by	System Description	0..*	
Aggregated by	VFB System	0..*	
In/out	Define System Variants	1	
In/out	Define Integration Variant	0..*	
In/out	Define VFB Variants	0..*	
Consumed by	Generate BSW Module Prebuild Data Set	1	
Consumed by	Generate RTE Prebuild Dataset	1	
Consumed by	Generate Component Prebuild Data Set	1..*	Meth.bindingTime = CodeGenerationTime
Consumed by	Generate Atomic Software Component Contract Header Files	0..1	Meth.bindingTime = SystemDesignTime
Consumed by	Evaluate Variant	0..*	
Consumed by	Extract ECU System Variant Model	0..*	
Use meta model element	SwSystemconstantValueSet	1	

Table 3.30: System Constant Value Set

3.1.4 Roles

Role	AUTOSAR Partnership		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Roles		
Brief Description	The AUTOSAR Partnership development defines standard artifacts.		
Description			
Relation Type	Related Element	Mul.	Note

Table 3.31: AUTOSAR Partnership

Role	Basic Software Designer		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Roles		
Brief Description	Role responsible for the overall design of the Basic Software.		
Description	Role responsible for the overall design of the Basic Software. In contrast to the Basic Software Module Developer he is responsible for the consistency of interfaces and data types between modules.		
Relation Type	Related Element	Mul.	Note
Performs	Define BSW Behavior	1	
Performs	Define BSW Entries	1	
Performs	Define BSW Interfaces	1	
Performs	Define BSW Types	1	
Performs	Create Transformer Specification	0..1	
Performs	Define VFB Nv Block Software Component	0..1	
Performs	Define Vendor Specific Module Definition	0..1	

Table 3.32: Basic Software Designer

Role	Basic Software Module Developer		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Roles		
Brief Description	Role responsible to develop and deliver a Basic Software Module.		
Description			
Relation Type	Related Element	Mul.	Note
Performs	Compile BSW Core Code	1	
Performs	Configure Compiler Memory Classes	1	
Performs	Create Library	1	

Relation Type	***Related Element***	***Mul.***	***Note***
Performs	Define BSW Entries	1	
Performs	Define BSW Interfaces	1	
Performs	Define BSW Module Timing	1	
Performs	Define BSW Types	1	
Performs	Define Memory Addressing Modes	1	
Performs	Develop BSW Module Generator	1	
Performs	Generate BSW Module Prebuild Data Set	1	
Performs	Generate BSWM Contract Header Files	1	
Performs	Implement a BSW Module	1	
Performs	Configure Memmap Allocation	0..1	
Performs	Define Vendor Specific Module Definition	0..1	
Performs	Generate BSW Memory Mapping Header	0..1	
Performs	Generate Compiler Configuration	0..1	
Performs	Measure Component Resources	0..1	

Table 3.33: Basic Software Module Developer

Role	**Calibration Engineer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Roles		
Brief Description	The calibration engieer determines the calibration parameters of an ECU.		
Description			
Relation Type	***Related Element***	***Mul.***	***Note***
Performs	Define VFB Parameter Component	1	
Performs	Generate A2L	1	
Performs	Create MC Function Model	0..1	

Relation Type	**Related Element**	**Mul.**	**Note**
Performs	Define VFB Constants	0..1	
Performs	Provide RTE Calibration Dataset	0..1	

Table 3.34: Calibration Engineer

Role	**Certification Agency**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Roles		
Brief Description	The certification agency verifies the conformance of artifacts with respect to the standard artifacts defined by the autosar consortium.		
Description			
Relation Type	**Related Element**	**Mul.**	**Note**

Table 3.35: Certification Agency

Role	**ECU Integrator**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Roles		
Brief Description	Integrates the complete software on an ECU.		
Description	Integrates the complete software on an ECU, which includes generating necessary code and completing the configuration of all software components and basic software modules.		
Relation Type	**Related Element**	**Mul.**	**Note**
Performs	Compile ECU Source Code	1	
Performs	Configure Com	1	
Performs	Configure Debug	1	
Performs	Configure Diagnostics	1	
Performs	Configure ECUC	1	
Performs	Configure IO Hardware abstraction	1	
Performs	Configure MCAL	1	
Performs	Configure Mode Management	1	
Performs	Configure NvM	1	
Performs	Configure OS	1	
Performs	Configure RTE	1	
Performs	Configure Transformer	1	
Performs	Configure Watchdog Manager	1	
Performs	Connect Service Component	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Performs	Create Library	1	
Performs	Create Service Component	1	
Performs	Define ECU Timing	1	
Performs	Define Integration Variant	1	
Performs	Extract the ECU Communication	1	
Performs	Generate BSW Configuration Code	1	
Performs	Generate BSW Memory Mapping Header	1	
Performs	Generate Base Ecu Configuration	1	
Performs	Generate Compiler Configuration	1	
Performs	Generate ECU Executable	1	
Performs	Generate Local MC Data Support	1	
Performs	Generate OS	1	
Performs	Generate RTE	1	
Performs	Generate RTE Postbuild Dataset	1	
Performs	Generate RTE Prebuild Dataset	1	
Performs	Generate SWC Memory Mapping Header	1	
Performs	Generate Scheduler	1	
Performs	Generate Updated ECU Configuration	1	
Performs	Measure Resources	1	
Performs	Provide RTE Calibration Dataset	1	
Performs	Configure Memmap Allocation	0..1	
Performs	Create MC Function Model	0..1	
Performs	Define VFB Nv Block Software Component	0..1	
Performs	Extend Topology	0..1	

Relation Type	**Related Element**	**Mul.**	**Note**
Performs	Extract ECU Rapid Prototyping Scenario	0..1	
Performs	Extract ECU System Timing	0..1	
Performs	Extract ECU System Variant Model	0..1	
Performs	Extract ECU Topology	0..1	
Performs	Flatten Software Composition	0..1	
Performs	Generate Component Header File in Vendor Mode	0..1	
Performs	Generate or Adjust ECU Flat Map	0..1	
Performs	Map Software Component to BSW	0..1	
Performs	Measure Component Resources	0..1	

Table 3.36: ECU Integrator

Role	**Software Component Designer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Roles		
Brief Description	Designer of software components and VFB systems.		
Description			
Relation Type	**Related Element**	**Mul.**	**Note**
Performs	Add Documentation to the Software Component	1	
Performs	Define Atomic Software Component Internal Behavior	1	
Performs	Define Complex Driver Component	1	
Performs	Define Consistency Needs	1	
Performs	Define ECU Abstraction Component	1	
Performs	Define VFB Application Software Component	1	
Performs	Define VFB Composition Component	1	

Relation Type	***Related Element***	***Mul.***	***Note***
Performs	Define VFB Constants	1	
Performs	Define VFB Interfaces	1	
Performs	Define VFB Modes	1	
Performs	Define VFB Sensor or Actuator Component	1	
Performs	Define VFB Timing	1	
Performs	Define VFB Types	1	
Performs	Define VFB Variants	1	
Performs	Define Wrapper Components to Integrate Legacy Software	1	
Performs	Map Software Component to BS W	1	
Performs	Define Partial Flat Map	0..1	
Performs	Define VFB Component Constraints	0..1	
Performs	Define VFB Nv Block Software Component	0..1	
Performs	Define VFB Top Level	0..1	

Table 3.37: Software Component Designer

Role	**Software Component Developer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Roles		
Brief Description	Developer of the software component code.		
Description			
Relation Type	***Related Element***	***Mul.***	***Note***
Performs	Define Consistency Needs	1	
Performs	Define Software Component Timing	1	
Performs	Define Symbol Props for Types	1	
Performs	Generate Atomic Software Component Contract Header Files	1	

Relation Type	Related Element	Mul.	Note
Performs	Generate Component Header File in Vendor Mode	1	
Performs	Generate Component Prebuild Data Set	1	
Performs	Implement Atomic Software Component	1	
Performs	Measure Component Resources	1	
Performs	Re-compile Component in ECU context	1	
Performs	Add Documentation to the Software Component	0..1	
Performs	Compile Atomic Software Component	0..1	
Performs	Configure Compiler Memory Classes	0..1	
Performs	Configure Memmap Allocation	0..1	
Performs	Define Atomic Software Component Internal Behavior	0..1	
Performs	Define Memory Addressing Modes	0..1	
Performs	Define Partial Flat Map	0..1	
Performs	Generate Compiler Configuration	0..1	
Performs	Generate SWC Memory Mapping Header	0..1	

Table 3.38: Software Component Developer

Role	**System Engineer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Roles		
Brief Description	Creation, management, developement and integration of systems within the vehicle		
Description			
Relation Type	**Related Element**	**Mul.**	**Note**

Relation Type	***Related Element***	***Mul.***	***Note***
Performs	Assign Top Level Composition	1	
Performs	Create Transformer Specification	1	
Performs	Define Communication Matrix	1	
Performs	Define E2E Transformer Technology	1	
Performs	Define ECU Description	1	
Performs	Define Frames	1	
Performs	Define Network Management	1	
Performs	Define PDU Gateway	1	
Performs	Define RTE Fanout	1	
Performs	Define Secured PDUs	1	
Performs	Define Signal Gateway	1	
Performs	Define Signal PDUs	1	
Performs	Define Signal Path Constraints	1	
Performs	Define System Timing	1	
Performs	Define System Topology	1	
Performs	Define System Variants	1	
Performs	Define System View Mapping	1	
Performs	Define TP	1	
Performs	Define Transformation Chain	1	
Performs	Define Transformation Technology	1	
Performs	Deploy Software Component	1	
Performs	Derive Communication Needs	1	
Performs	Extend Composition	1	
Performs	Extract the ECU Communication	1	
Performs	Flatten Software Composition	1	

Relation Type	***Related Element***	***Mul.***	***Note***
Performs	Generate or Adjust System Flat Map	1	
Performs	Select Design Time Variant	1	
Performs	Select Software Component Implementation	1	
Performs	Set System Root	1	
Performs	Define VFB Component Constraints	0..1	
Performs	Define VFB Composition Component	0..1	
Performs	Define VFB Constants	0..1	
Performs	Define VFB Top Level	0..1	
Performs	Extend Topology	0..1	
Performs	Extract ECU Rapid Prototyping Scenario	0..1	
Performs	Extract ECU System Timing	0..1	
Performs	Extract ECU System Variant Model	0..1	
Performs	Extract ECU Topology	0..1	
Performs	Generate or Adjust ECU Flat Map	0..1	

Table 3.39: System Engineer

Role	**Non-AUTOSAR System Integrator**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Roles		
Brief Description	Responsibility for the quality of the description of the non-AUTOSAR system and its integration into the AUTOSAR process.		
Description	The non-AUTOSAR System Integrator is responsible for the quality of the Description of the non-AUTOSAR System, the correct definition of the VFB Integration Connector, and the integration of the non-AUTOSAR system into the AUTOSAR process via the translation of the non-AUTOSAR artifacts.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performs	Define VFB Integration Connector	1	
Performs	Translate Non-Autosar Description to Autosar Description	1	

Relation Type	*Related Element*	*Mul.*	*Note*

Table 3.40: Non-AUTOSAR System Integrator

Role	**Rapid Prototyping Engineer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Roles		
Brief Description			
Description			
Relation Type	*Related Element*	*Mul.*	*Note*
Performs	Define Rapid Prototyping Scenario	1	
Performs	Generate Rapid Prototyping Wrapper	1	
Performs	Refine Rapid Prototyping Scenario	1	
Performs	Compile Atomic Software Component	0..1	

Table 3.41: Rapid Prototyping Engineer

Role	**Safety Engineer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Roles		
Brief Description			
Description	Responsibility for the safety relevant steps in the AUTOSAR development process		
Relation Type	*Related Element*	*Mul.*	*Note*
Performs	Add Independence Relation	1	
Performs	Allocate Safety Measure	1	
Performs	Allocate Safety Requirement	1	
Performs	Decompose Safety Requirement	1	
Performs	Define ASIL For AUTOSAR Element	1	
Performs	Define Safety Measure	1	
Performs	Define Safety Requirement	1	
Performs	Map Safety Requirement to Safety Measure	1	
Performs	Refine Safety Requirement	1	

Relation Type	Related Element	Mul.	Note

Table 3.42: Safety Engineer

3.1.5 Tools

3.1.5.1 Compiler

Tool	Compiler		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Guidance		
Brief Description			
Description			
Kind			
Relation Type	Related Element	Mul.	Note
Used	Compile Atomic Software Component	1	
Used	Compile BSW Configuration Data	1	
Used	Compile BSW Core Code	1	
Used	Compile Configured BSW	1	
Used	Compile ECU Source Code	1	
Used	Compile Generated BSW	1	
Used	Compile Unconfigured BSW	1	
Used	Re-compile Component in ECU context	1	

Table 3.43: Compiler

3.1.5.2 Linker

Tool	Linker		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Guidance		
Brief Description			
Description			
Kind			
Relation Type	Related Element	Mul.	Note
Used	Generate ECU Executable	1	

Relation Type	Related Element	Mul.	Note
Used	Link ECU Code after Precompile Configuration	1	
Used	Link ECU Code during Link Time Configuration	1	
Used	Link ECU Code during Post-Build Time	1	

Table 3.44: Linker

3.1.6 Diagnostics

3.1.6.1 Work Products

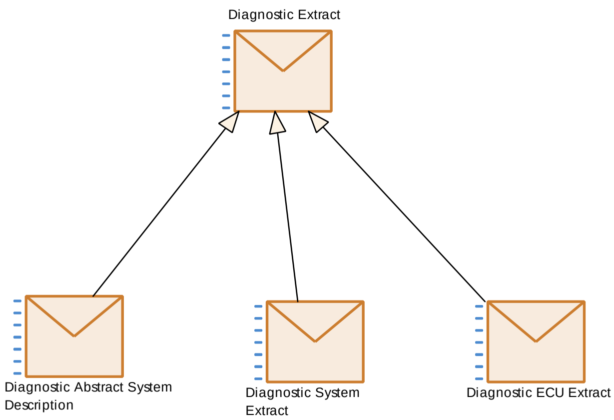

Figure 3.12: Diagnostic Extract Deliverables

Deliverable	**Diagnostic Extract**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Diagnostics::Work Products		
Brief Description			
Description	Generic deliverable for defining diagnostic information. It is used in different roles (Diagnostic Extract categories). In each role, this deliverable may contain variation points in its ARXML artifacts which need to be bound in later steps. If such variation points are present, the Diagnostic Description may optionally include PredefinedVariants in order to predefine variants for later selection and an Evaluated Variant Set.		
Kind			
Extended by	Diagnostic Abstract System Description, Diagnostic ECU Extract, Diagnostic System Extract		
Relation Type	*Related Element*	*Mul.*	*Note*

Table 3.45: Diagnostic Extract

Deliverable	**Diagnostic Abstract System Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Diagnostics::Work Products		
Brief Description			
Description	This deliverable represents a more or less high-level definition of diagnostic information that can be taken as a template for creating Diagnostic System Extract or Diagnostic ECU Extract. It corresponds to an Diagnostic Extract with DiagnosticContributionSet of category DIAGNOSTICS_ABSTRACT_SYSTEM_DESCRIPTION.		
Kind			
Extends	Diagnostic Extract		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Develop Diagnostic Abstract System Description	1	
	Develop Diagnostic Requirements	0..*	

Table 3.46: Diagnostic Abstract System Description

Deliverable	**Diagnostic System Extract**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Diagnostics::Work Products		
Brief Description			
Description	This deliverable represents concrete diagnostic information for several ECUs. It corresponds to an Diagnostic Extract with DiagnosticContributionSet of category DIAGNOSTICS_SYSTEM_EXTRACT.		
Kind			
Extends	Diagnostic Extract		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Develop Application Software	0..*	Diagnostic information relevant to the SW-Cs is provided as a part of the Diagnostic System Extract and can contain relationships to the SW-C's service needs.
Produced by	Develop Basic Software	0..*	
Produced by	Develop Diagnostic Requirements	0..*	
Consumed by	Develop Application Software	0..*	The Diagnostic System Extract contains diagnostic information that serves as a requirement for the software developer.
Consumed by	Develop Basic Software	0..*	
Consumed by	Develop Diagnostic Requirements	0..*	
Consumed by	Integrate Diagnostic Information	0..*	

Table 3.47: Diagnostic System Extract

Deliverable	**Diagnostic ECU Extract**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Diagnostics::Work Products		
Brief Description			
Description	This deliverable represents concrete diagnostic information for a single ECUs. It corresponds to an Diagnostic Extract with DiagnosticContributionSet of category DIAGNOSTICS_ECU_EXTRACT.		
Kind			
Extends	Diagnostic Extract		
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Integrate Diagnostic Information	1..*	complete DE:
Produced by	Develop Diagnostic Requirements	0..*	
Consumed by	Generate Base Ecu Configuration	0..1	
Consumed by	Generate Updated ECU Configuration	0..1	
Consumed by	Integrate Software for ECU	0..1	complete DE:
Consumed by	Prepare ECU Configuration	0..1	
Consumed by	Update ECU Configuration	0..1	
Consumed by	Integrate Diagnostic Information	0..*	partially filled DE:

Table 3.48: Diagnostic ECU Extract

3.1.7 Safety

3.1.7.1 Tasks

3.1.7.1.1 Define Safety Requirement

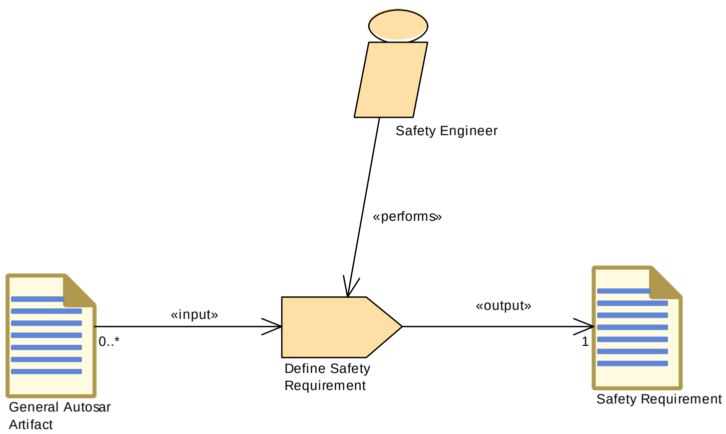

Figure 3.13: Define Safety Requirement

Task Definition	**Define Safety Requirement**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Tasks		
Brief Description	Add Safety Requirements to work products.		
Description	This task creates a safety requirement and sets the corresponding attributes such as ASIL. The allocation to an AUTOSAR element and the mapping to a safety measure are not part of this task.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Safety Engineer	1	
Consumes	General Autosar Artifact	0..*	
Produces	Safety Requirement	1	

Table 3.49: Define Safety Requirement

3.1.7.1.2 Define Safety Measure

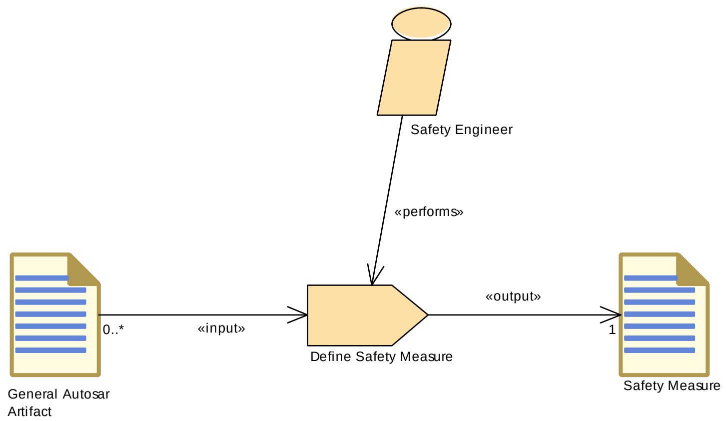

Figure 3.14: Define Safety Measure

Task Definition	**Define Safety Measure**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Tasks		
Brief Description	Add Safety Measures to work products.		
Description	This task creates a safety measure and sets the corresponding attributes such as ASIL. The allocation to an AUTOSAR element and the mapping to a safety requirement are not part of this task.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Safety Engineer	1	
Consumes	General Autosar Artifact	0..*	
Produces	Safety Measure	1	

Table 3.50: Define Safety Measure

3.1.7.1.3 Define ASIL For AUTOSAR Element

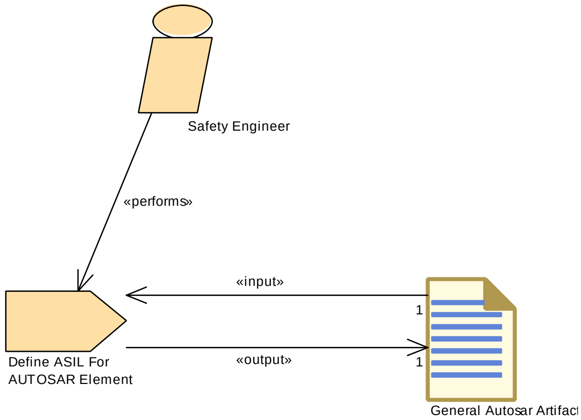

Figure 3.15: Define ASIL For AUTOSAR Element

Task Definition	**Define ASIL For AUTOSAR Element**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Tasks		
Brief Description	Provide ASIL attribute for AUTOSAR element.		
Description	According to the safety extensions, AUTOSAR elements can carry ASIL attributes if they are safety relevant. This task assigns the ASIL attribute to an AUTOSAR element. The assignment of the ASIL attribute can also be done for safety requirements and safety measures. This is covered by the tasks "Define Safety Requirement" and "Define Safety Measure".		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	Safety Engineer	1	
Consumes	General Autosar Artifact	1	
Produces	General Autosar Artifact	1	

Table 3.51: Define ASIL For AUTOSAR Element

3.1.7.1.4 Refine Safety Requirement

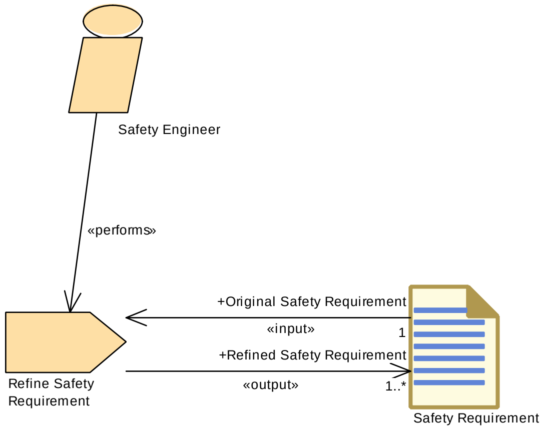

Figure 3.16: Refine Safety Requirement

Task Definition	**Refine Safety Requirement**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Tasks		
Brief Description	Refine existing Safety Requirements by adding more detailed safety requirements and organize them in an appropriate hierarchy.		
Description	If safety requirements are not detailed enough to allocate them directly to appropriate AUTOSAR elements, it is necessary to refine them first. The refinement will add new safety requirements which are in a hierarchy relation to existing safety requirements. This task adds the corresponding "REFINEMENT" relation between the original requirement and the newly created requirements. This task can be done on different levels, depending on the level of details of the safety requirements.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Safety Engineer	1	
Consumes	Safety Requirement	1	Original Safety Requirement:
Produces	Safety Requirement	1..*	Refined Safety Requirement:

Table 3.52: Refine Safety Requirement

3.1.7.1.5 Decompose Safety Requirement

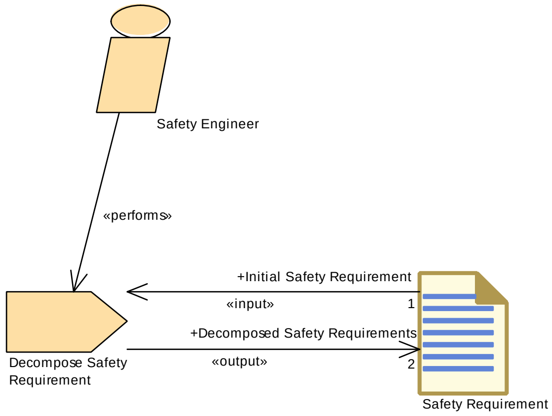

Figure 3.17: Decompose Safety Requirement

Task Definition	**Decompose Safety Requirement**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Tasks		
Brief Description	Decompose existing Safety Requirements into independent Safety Requirements to tailor the ASIL.		
Description	By ASIL decomposition it is possible to decompose a safety requirement into two new safety requirements with potentially lower ASILs. This can be done, if the independence (freedom of interference) for the resulting requirements can be demonstrated. The modeling of the corresponding INDEPENDENCE relation is covered by task "Add Independence Relation". This task adds the corresponding "DECOMPOSITION" reference.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Safety Engineer	1	
Consumes	Safety Requirement	1	Initial Safety Requirement:
Produces	Safety Requirement	2	Decomposed Safety Requirements:

Table 3.53: Decompose Safety Requirement

3.1.7.1.6 Allocate Safety Measure

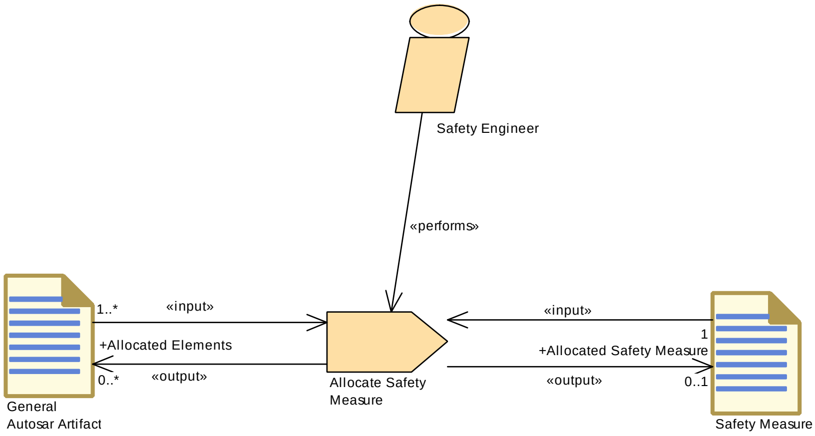

Figure 3.18: Allocate Safety Measure

Task Definition	**Allocate Safety Measure**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Tasks		
Brief Description	Allocate Safety Measure to AUTOSAR elements.		
Description	Safety measures which are safety mechanisms realized in AUTOSAR are allocated to AUTOSAR elements in order to describe what elements are involved in the provision of a safety measure. This task adds the corresponding "ALLOCATION" reference. The reference can be contained by the AUTOSAR element or by the safety measure. The allocation can be done on different levels, depending on the granularity of the safety measures and the availability of the appropriate elements in the model.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Safety Engineer	1	
Consumes	Safety Measure	1	
Consumes	General Autosar Artifact	1..*	
Produces	Safety Measure	0..1	Allocated Safety Measure:
Produces	General Autosar Artifact	0..*	Allocated Elements:

Table 3.54: Allocate Safety Measure

3.1.7.1.7 Allocate Safety Requirement

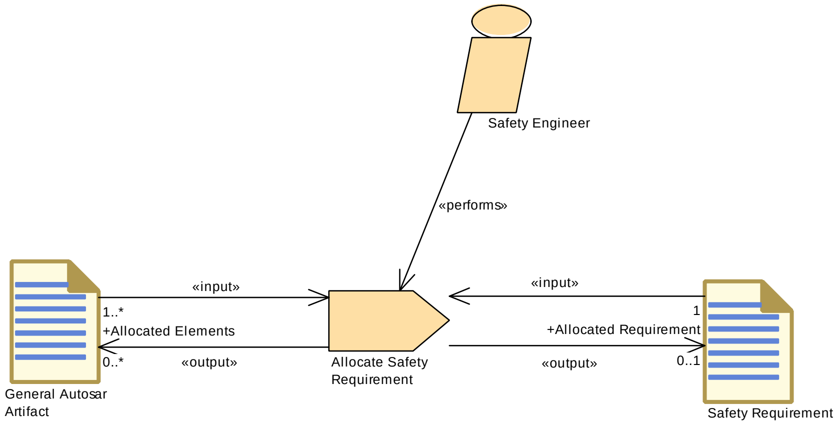

Figure 3.19: Allocate Safety Requirement

Task Definition	**Allocate Safety Requirement**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Tasks		
Brief Description	Allocate Safety Requirement to AUTOSAR elements.		
Description	Safety requirements are allocated to AUTOSAR elements in order to fulfill the needs of ISO 26262. By this allocation, AUTOSAR elements obtain their ASIL attribute (if not defined e.g. during previous development of the element). This task adds the corresponding allocation reference to the AUTOSAR element. The reference can be contained by the AUTOSAR element or by the safety requirement. The allocation can be done on different levels, depending on the granularity of the safety requirements and the availability of the appropriate elements in the model.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Safety Engineer	1	
Consumes	Safety Requirement	1	
Consumes	General Autosar Artifact	1..*	
Produces	Safety Requirement	0..1	Allocated Requirement:
Produces	General Autosar Artifact	0..*	Allocated Elements:

Table 3.55: Allocate Safety Requirement

3.1.7.1.8 Map Safety Requirement to Safety Measure

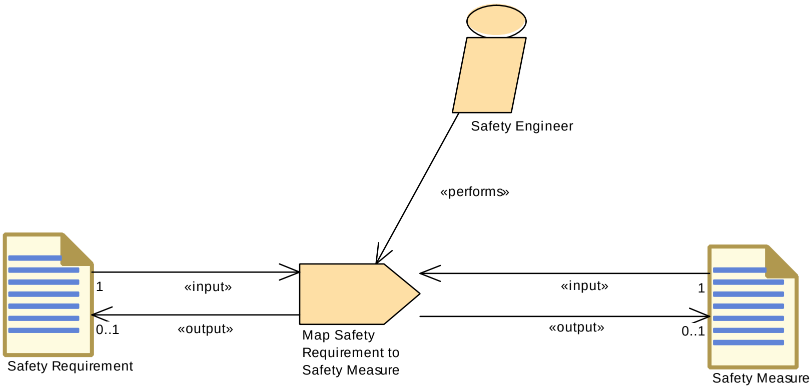

Figure 3.20: Map Safety Requirement to Safety Measure

Task Definition	**Map Safety Requirement to Safety Measure**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Tasks		
Brief Description	Map Safety Requirements to Safety Measures		
Description	The mapping relates safety requirements with safety measures. This task creates the corresponding MAPS_TO relation. The mapping relation can either be contained by the safety requirement or by the safety measure. The mapping can be done on different levels, depending on the granularity of the safety requirements and the safety measures.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Safety Engineer	1	
Consumes	Safety Measure	1	
Consumes	Safety Requirement	1	
Produces	Safety Measure	0..1	
Produces	Safety Requirement	0..1	

Table 3.56: Map Safety Requirement to Safety Measure

3.1.7.1.9 Add Independence Relation

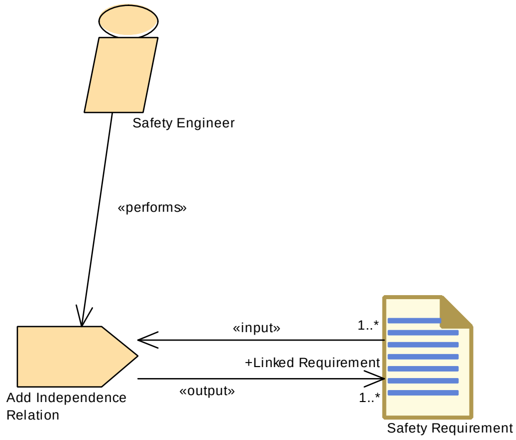

Figure 3.21: Add Independence Relation

Task Definition	**Add Independence Relation**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Tasks		
Brief Description	Add Independence relation to decomposed requirements.		
Description	This task establishes the INDEPENDENCE relation between requirements. The relation is established between a decomposed requirement and a requirement which express a means to achieve freedom from interference for the two requirements into which the decomposed requirement is decomposed by the task Decompose Safety Requirement. Obviously, this task is processed in the context of the decomposition of safety requirements.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Safety Engineer	1	
Consumes	Safety Requirement	1..*	
Produces	Safety Requirement	1..*	Linked Requirement:

Table 3.57: Add Independence Relation

3.1.7.2 Work Products

3.1.7.2.1 Safety Extensions

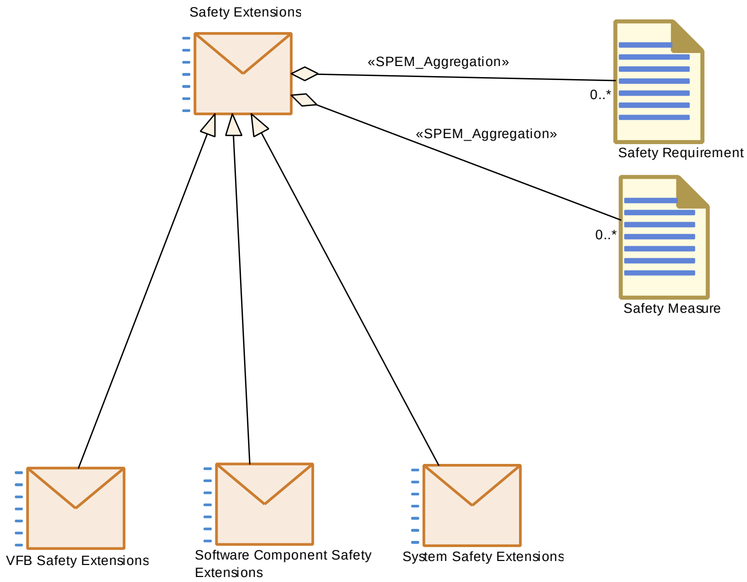

Figure 3.22: Safety Extensions

Deliverable	**Safety Extensions**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Work Products		
Brief Description	Safety Extensions		
Description	This element represents an abstract deliverable containing all safety relevant artifacts. Several specializations of this deliverable are used to demonstrate the handling of safety extensions in specific development activities. The explicit separation of the safety information from the AUTOSAR models allows an independent exchange and processing of them.		
Kind	Delivered		
Extended by	Software Component Safety Extensions, System Safety Extensions, VFB Safety Extensions		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Safety Measure	0..*	
Aggregates	Safety Requirement	0..*	

Table 3.58: Safety Extensions

Deliverable	**VFB Safety Extensions**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Work Products		
Brief Description	Vfb Safety Extensions		
Description	This deliverable contains all safety information related to VFB elements.		
Kind	Delivered		
Extends	Safety Extensions		
Relation Type	**Related Element**	**Mul.**	**Note**
Produced by	Define VFB Safety Information	1	
Consumed by	Define Software Component Safety Information	1	
Consumed by	Define System Safety Information	1	

Table 3.59: VFB Safety Extensions

Deliverable	**Software Component Safety Extensions**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Work Products		
Brief Description	Software Component Safety Extensions		
Description	This deliverable contains all safety information related to software components.		
Kind	Delivered		
Extends	Safety Extensions		
Relation Type	**Related Element**	**Mul.**	**Note**
Produced by	Define Software Component Safety Information	1	
Consumed by	Define System Safety Information	1	

Table 3.60: Software Component Safety Extensions

Deliverable	**System Safety Extensions**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Work Products		
Brief Description	System Safety Extensions		
Description	This deliverable contains all safety information related to system elements (see Deliverable "System Description" for more details).		
Kind	Delivered		
Extends	Safety Extensions		
Relation Type	**Related Element**	**Mul.**	**Note**
Produced by	Define System Safety Information	1	

Table 3.61: System Safety Extensions

3.1.7.2.2 Safety Requirement

Artifact	**Safety Requirement**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Work Products		
Brief Description	Safety Requirement		
Description	This artifact represents a safety requirement and the corresponding ASIL attribute. ISO 26262 defines a hierarchy of safety requirements: safety goals, technical, hardware and software. Furthermore, it might be the case that safety requirements are specified outside the AUTOSAR model (external) and are only referenced. Thus, the safety requirement can have one of the following categories: • SAFETY_GOAL • SAFETY_FUNCTIONAL • SAFETY_TECHNICAL • SAFETY_SOFTWARE • SAFETY_HARDWARE • SAFETY_EXTERNAL For details refer to ISO 26262-3, 4, 9 and TPS_SafetyExtensions document.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	Safety Extensions	0..*	
Produced by	Decompose Safety Requirement	2	Decomposed Safety Requirements:
Produced by	Define Safety Requirement	1	
Produced by	Add Independence Relation	1..*	Linked Requirement:
Produced by	Refine Safety Requirement	1..*	Refined Safety Requirement:
Produced by	Allocate Safety Requirement	0..1	Allocated Requirement:
Produced by	Map Safety Requirement to Safety Measure	0..1	
Consumed by	Allocate Safety Requirement	1	
Consumed by	Decompose Safety Requirement	1	Initial Safety Requirement:
Consumed by	Map Safety Requirement to Safety Measure	1	
Consumed by	Refine Safety Requirement	1	Original Safety Requirement:

Relation Type	Related Element	Mul.	Note
Consumed by	Add Independence Relation	1..*	
Use meta model element	StructuredReq	1	

Table 3.62: Safety Requirement

3.1.7.2.3 Safety Measure

Artifact	**Safety Measure**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Safety::Work Products		
Brief Description	Safety Measure		
Description	This artifact represents a safety measure. A safety measure is an activity or solution to avoid systematic failures and to detect random hardware failures or control failures (see ISO 26262). The safety measure can have one of the following categories: • SAFETY_MEASURE • SAFETY_MECHANISM For further details refer to TPS_SafetyExtensions document.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Safety Extensions	0..*	
Produced by	Define Safety Measure	1	
Produced by	Allocate Safety Measure	0..1	Allocated Safety Measure:
Produced by	Map Safety Requirement to Safety Measure	0..1	
Consumed by	Allocate Safety Measure	1	
Consumed by	Map Safety Requirement to Safety Measure	1	
Use meta model element	TraceableText	1	

Table 3.63: Safety Measure

3.1.8 Data Exchange Points

3.1.8.1 Work Products

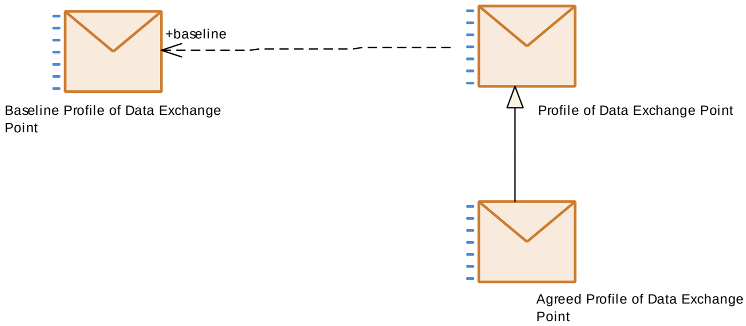

Figure 3.23: Data Exchange Point Work Products

Deliverable	**Baseline Profile of Data Exchange Point**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Data Exchange Point::Work Products		
Brief Description	The Baseline Profile of Data Exchange Point specifies default values that are used when processing Profile of Data Exchange Points that do not specify all values.		
Description	The Baseline Profile of Data Exchange Point specifies default values that are used when processing Profile of Data Exchange Points that do not specify all values. Actual Profiles of Data Exchange Point only have to specify information that deviates from the default values provided in the Baseline Profile of Data Exchange Point. This Baseline Profile of Data Exchange Point is provided and standardized by AUTOSAR for each commonly used revision of the standard.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Create Profile of Data Exchange Point	1	The Baseline Profile of Data Exchange Point defines the default values of the Profile.
Consumed by	Validate Compliance of Autosar Model against Profile for Data Exchange Point	1	The Baseline Profile of Data Exchange Point provides default values foe parameters that are not specified in the Agreed Profile of Data Exchange Point.

Table 3.64: Baseline Profile of Data Exchange Point

Deliverable	**Profile of Data Exchange Point**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Data Exchange Point::Work Products		
Brief Description	A Profile of Data Exchange Point specifies which parts of the Autosar standard are relevant for a specific data exchange scenario.		
Description	A Profile of Data Exchange Point specifies which parts of the AUTOSAR standard are relevant for a specific data exchange scenario. Each Profile of Data Exchange Point shall specify the revision of the AUTOSAR standard that is used as the baseline. Additionally, it should contain a high-level description of the data exchange point by specifying the deliverable and the intended use as well a detailed specification of the relevant part of the meta model and the applicable constraints. A Profile of Data Exchange Point that describes an individual Data Exchange Point is based on a Baseline Profile of Data Exchange Point that is provided by AUTOSAR. The Profile of Data Exchange Point only needs to describe the parameters that differ from its Baseline Profile of Data Exchange Point.		
Kind	AUTOSAR XML		
Extended by	Agreed Profile of Data Exchange Point		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Create Profile of Data Exchange Point	1	The new Profile of Data Exchange Point
Consumed by	Create Agreed Profile for Data Exchange Point	1..*	The Profiles of Data Exchange Point that are analyzed with respect to interoperability issues.
Consumed by	Create Profile of Data Exchange Point	0..*	An existing Profile of Data Exchange Point that is used as a starting point for a new Profile of Data Exchange Point. E.g. start with a profile of an old tool version and tailor it so that new features are covered.

Table 3.65: Profile of Data Exchange Point

Deliverable	**Agreed Profile of Data Exchange Point**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Common Elements::Data Exchange Point::Work Products		
Brief Description	The Agreed Profile of Data Exchange Point is a Profile of Data Exchange Point that documents the agreed Data Exchange Point between partners		
Description	The Agreed Profile of Data Exchange Point is a Profile of Data Exchange Point that documents the agreed Data Exchange Point between partners. It can be used during the project to continuously evaluate if the exchanged AUTOSAR models still conform with the agreed contract.		
Kind	AUTOSAR XML		
Extends	Profile of Data Exchange Point		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Create Agreed Profile for Data Exchange Point	1	The Agreed Profile of Data Exchange Point
Consumed by	Validate Compliance of Autosar Model against Profile for Data Exchange Point	1	The DataFormatTailoring and Baseline sections of the Agreed Profile of Data Exchange Point can be used to configure the validation engine of Autosar Validation Tools (e.g. provide rules for existence of elements or switch on/off individual semantic constraints)

Table 3.66: Agreed Profile of Data Exchange Point

3.2 Virtual Functional Bus

This chapter contains the definition of work products and tasks used for the development of a VFB system. For the definition of the relevant meta-model elements refer to [6], for the VFB concepts refer to [5].

3.2.1 Tasks

3.2.1.1 Define VFB Top Level

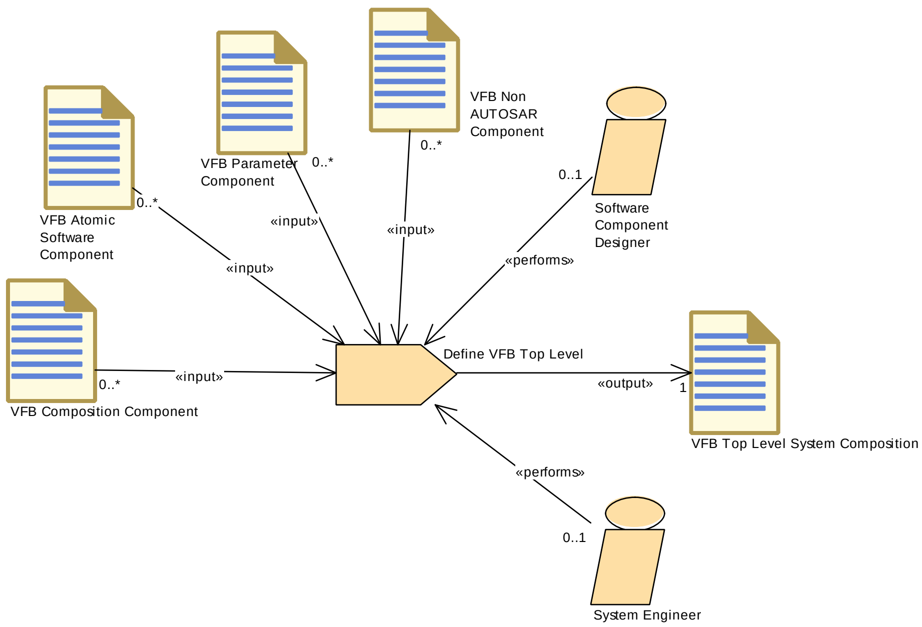

Figure 3.24: Task Define VFB Top Level

Task Definition	**Define VFB Top Level**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define the top level VFB composition of a concrete system.		
Description	Define the top level composition of a VFB system.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Designer	0..1	
Performed by	System Engineer	0..1	
Consumes	VFB Interfaces	1..*	
Consumes	VFB Types	1..*	
Consumes	VFB Atomic Software Component	0..*	
Consumes	VFB Composition Component	0..*	
Consumes	VFB Modes	0..*	
Consumes	VFB Non AUTOSA R Component	0..*	
Consumes	VFB Parameter Component	0..*	
Produces	VFB Top Level System Composition	1	

Table 3.67: Define VFB Top Level

3.2.1.2 Define VFB Composition Component

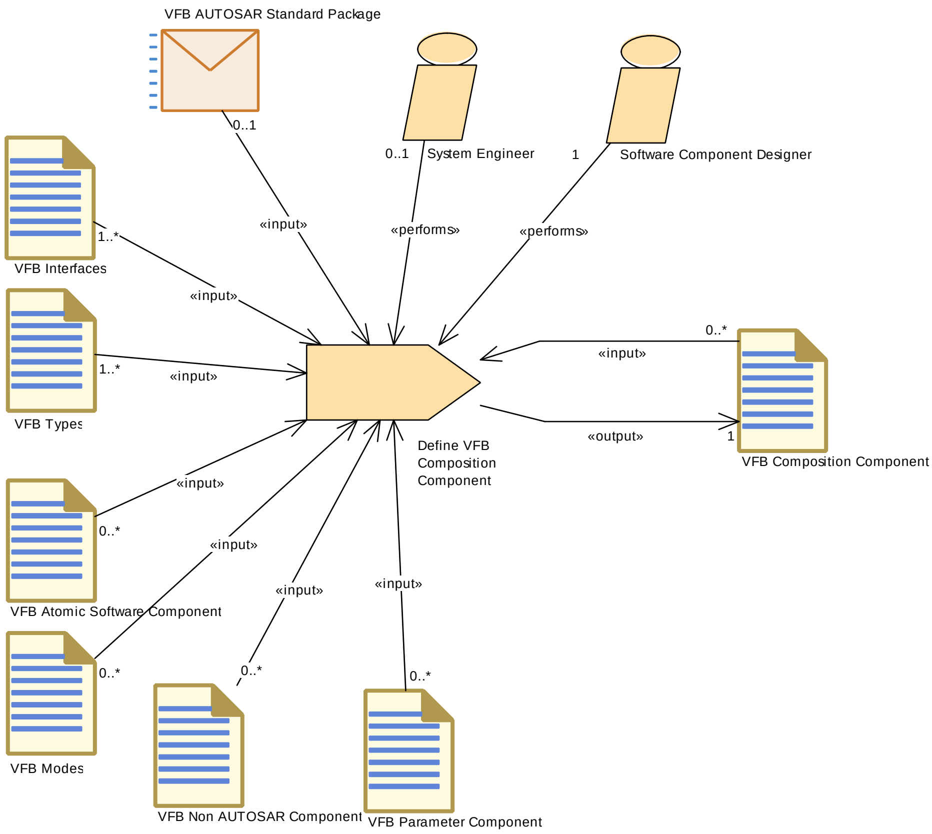

Figure 3.25: Task Define VFB Composition Component

Task Definition	**Define VFB Composition Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define a Composition of VFB Software Components, i.e. a ComponentTypes which contains other Component Types.		
Description	Define a Composition of VFB Software Components, i.e. a ComponentType which contains other Component Types. Iteration of this task can create a complete VFB system without the Atomic Software Components itself.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	Software Component Designer	1	
Performed by	System Engineer	0..1	
Consumes	VFB Interfaces	1..*	
Consumes	VFB Types	1..*	
Consumes	VFB AUTOSAR Standard Package	0..1	Use port blueprints in order to create ports with standardized application interfaces.
Consumes	VFB Atomic Software Component	0..*	
Consumes	VFB Composition Component	0..*	

Relation Type	Related Element	Mul.	Note
Consumes	VFB Modes	0..*	
Consumes	VFB Non AUTOSAR Component	0..*	
Consumes	VFB Parameter Component	0..*	
Produces	VFB Composition Component	1	

Table 3.68: Define VFB Composition Component

3.2.1.3 Extend Composition

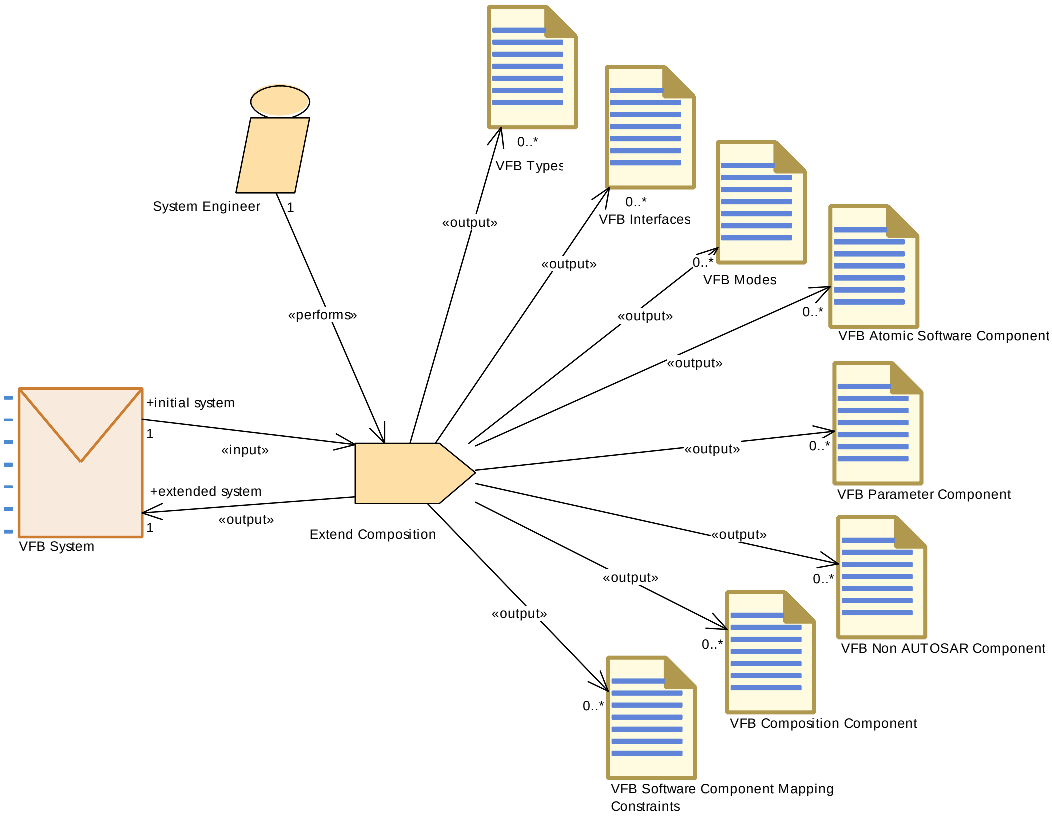

Figure 3.26: Task Extend Composition

Task Definition	**Extend Composition**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Extend a software composistion with further compositions and atomic software components.		
Description	This tasks describes the refinement of a delivered VFB System by extending an existing composition with further sub-elements, which could be software components (Atomic Software Components as well as Compositions), connectors or port groups, plus the related interfaces, data types and modes. The main use case is the refinement of the VFB description of a sub-system: New elements are added but the original delivery is not changed.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	VFB System	1	initial system:
Produces	VFB System	1	extended system:
Produces	VFB Atomic Software Component	0..*	
Produces	VFB Composition Component	0..*	
Produces	VFB Interfaces	0..*	
Produces	VFB Modes	0..*	
Produces	VFB Non AUTOSAR Component	0..*	
Produces	VFB Parameter Component	0..*	
Produces	VFB Software Component Mapping Constraints	0..*	
Produces	VFB Types	0..*	

Table 3.69: Extend Composition

3.2.1.4 Define VFB Component Constraints

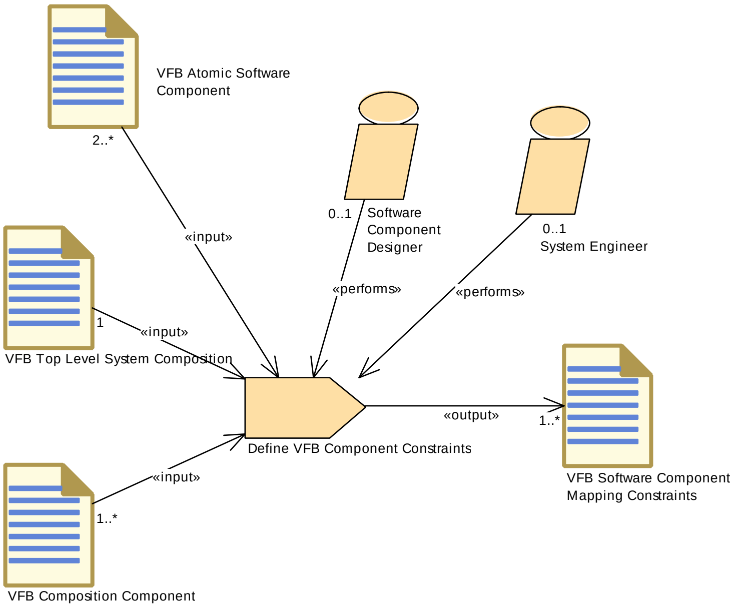

Figure 3.27: Task Define VFB Component Constraints

Task Definition	**Define VFB Component Constraints**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define which components need to be deployed together, and which need to be deployed separately.		
Description	In this task constraints for software components are defined. These constraints can on the one hand describe which SW-Cs should be mapped together to a single ECU, and which must be mapped to separate ECUs, without regard to any particular ECU or topology. This can be done by using the meta-model ComponentClustering and ComponentSeparation constraint. In fact, before the mapping process begins, it can be useful to impose the allocation of a predefined set of SW components onto the same ECU, especially if such a set is tightly linked from a functional point of view. In the same way, two critical SW components, performing some kind of redundancy, may be not suitable to run both on the same ECU. Thus, we call these two kinds of mapping constraints, respectively, ComponentClustering and ComponentSeparation. The ComponentClustering constraint (also, clustering) is to be used for expressing that a certain set of SW components (atomic or not) shall be mapped (allocated) onto the same ECU. This is some kind of "execute together on same ECU" constraint. The ComponentSeparation constraint (also, separation) is to be used for expressing that two SW components (atomic or not) shall not be mapped (allocated) onto the same ECU. This is some kind of "do not execute together on same ECU" constraint.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Designer	0..1	
Performed by	System Engineer	0..1	

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	VFB Atomic Software Component	2..*	
Consumes	VFB Top Level System Composition	1	
Consumes	VFB Composition Component	1..*	
Produces	VFB Software Component Mapping Constraints	1..*	

Table 3.70: Define VFB Component Constraints

3.2.1.5 Define VFB Application Software Component

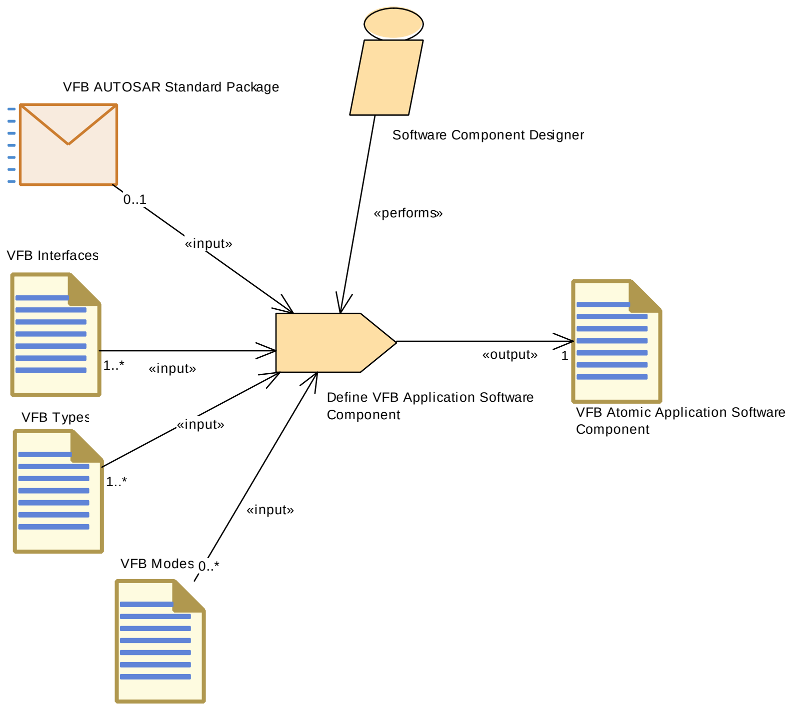

Figure 3.28: Task `Define VFB Application Software Component`

Task Definition	**Define VFB Application Software Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define an ApplicationSoftwareComponentType on VFB level		
Description	Define an ApplicationSwComponentType on VFB level. (i.e. without Internal Behavior and Implementation).		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Designer	1	
Consumes	VFB Interfaces	1..*	
Consumes	VFB Types	1..*	

Relation Type	Related Element	Mul.	Note
Consumes	VFB AUTOSAR Standard Package	0..1	Use port blueprints in order to create ports with standardized application interfaces.
Consumes	VFB Modes	0..*	
Produces	VFB Atomic Application Software Component	1	

Table 3.71: Define VFB Application Software Component

3.2.1.6 Define VFB Sensor or Actuator Component

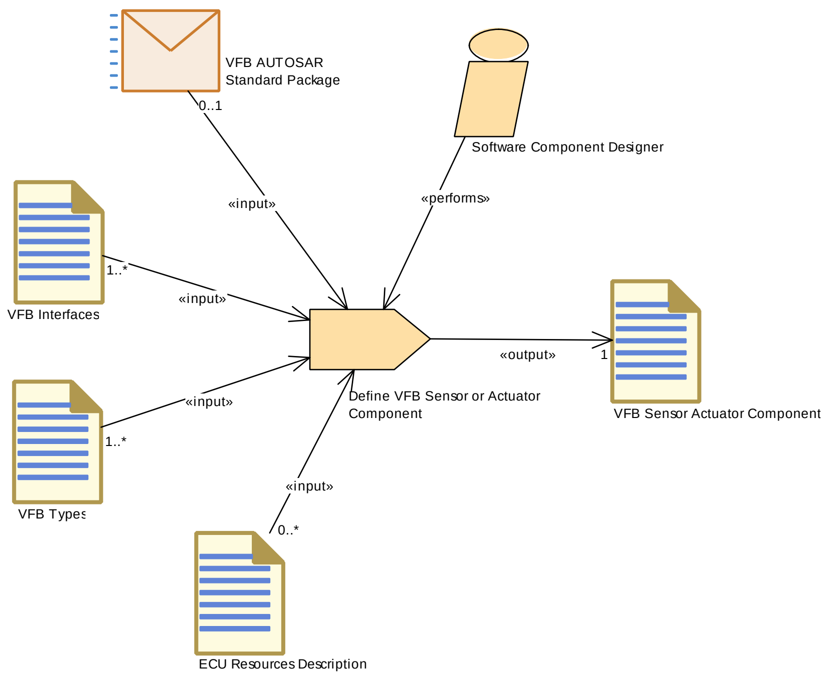

Figure 3.29: Task Define VFB Sensor or Actuator Component

Task Definition	Define VFB Sensor or Actuator Component		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define a VFB Sensor or Actuator Comnponent.		
Description	Define a SensorActuatorSwComponentType on VFB level. (i.e. without Internal Behavior and Implementation). In addition to defining the ports, references to the required sensor/actuator hardrware shall be specified.		
Relation Type	**Related Element**	**Mul.**	**Note**
Performed by	Software Component Designer	1	
Consumes	VFB Interfaces	1..*	
Consumes	VFB Types	1..*	

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	VFB AUTOSAR Standard Package	0..1	Use port blueprints in order to create ports with standardized application interfaces.
Consumes	ECU Resources Description	0..*	
Produces	VFB Sensor Actuator Component	1	

Table 3.72: Define VFB Sensor or Actuator Component

3.2.1.7 Define VFB Parameter Component

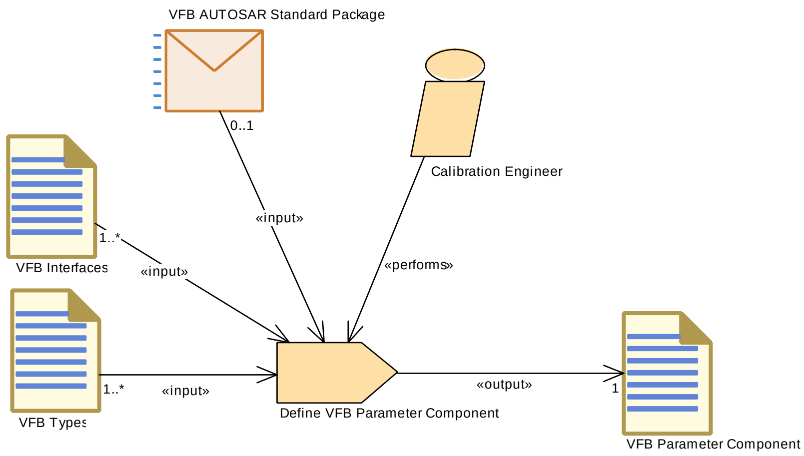

Figure 3.30: Task Define VFB Parameter Component

Task Definition	**Define VFB Parameter Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define a VFB Parameter Component.		
Description	Define a VFB Parameter Component.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	Calibration Engineer	1	
Consumes	VFB Interfaces	1..*	
Consumes	VFB Types	1..*	
Consumes	VFB AUTOSAR Standard Package	0..1	Use port blueprints in order to create ports with standardized application interfaces.
Produces	VFB Parameter Component	1	

Table 3.73: Define VFB Parameter Component

3.2.1.8 Define ECU Abstraction Component

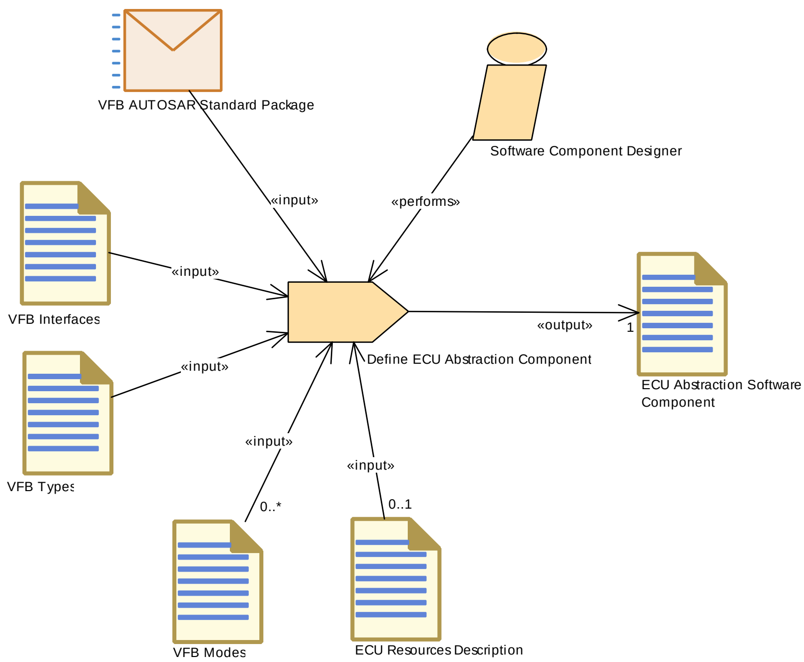

Figure 3.31: Task `Define ECU Abstraction Component`

Task Definition	**Define ECU Abstraction Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define an EcuAbstractionSoftwareComponentType on VFB level.		
Description	Define a EcuAbstractionSwComponentType on VFB level. (i.e. without Internal Behavior and Implementation). In addition to the defining the ports, references to required ECU or processor hardware elements shall be specified.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Designer	1	
Consumes	VFB AUTOSAR Standard Package	1	Use port blueprints in order to create ports with standardized application interfaces.
Consumes	VFB Interfaces	1	
Consumes	VFB Types	1	
Consumes	ECU Resources Description	0..1	
Consumes	VFB Modes	0..*	
Produces	ECU Abstraction Software Component	1	

Table 3.74: Define ECU Abstraction Component

3.2.1.9 Define Complex Driver Component

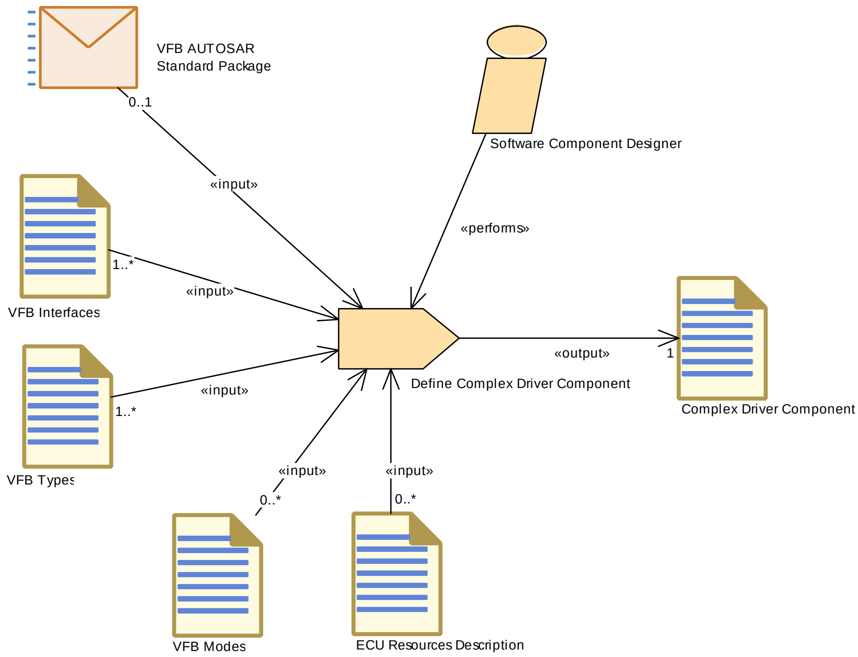

Figure 3.32: Task Define Complex Driver Component

Task Definition	**Define Complex Driver Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define a ComplexDeviceDriverSwComponentType on VFB level.		
Description	Define a ComplexDeviceDriverSwComponentType on VFB level. (i.e. without Internal Behavior and Implementation). In addition to the defining the ports, references to the required ECU or processor hardware elements shall be specified.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	Software Component Designer	1	
Consumes	VFB Interfaces	1..*	
Consumes	VFB Types	1..*	
Consumes	VFB AUTOSAR Standard Package	0..1	Use port blueprints in order to create ports with standardized application interfaces.
Consumes	ECU Resources Description	0..*	
Consumes	VFB Modes	0..*	
Produces	Complex Driver Component	1	

Table 3.75: Define Complex Driver Component

3.2.1.10 Define VFB NvBlock Software Component

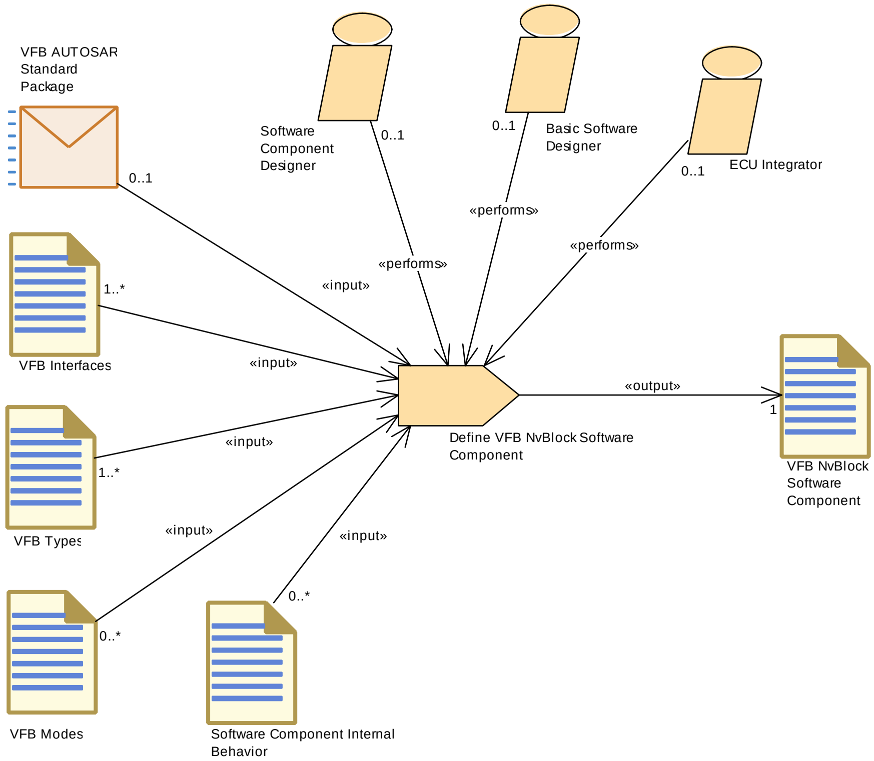

Figure 3.33: Task Define VFB NvBlock Software Component

Task Definition	**Define VFB NvBlock Software Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description			
Description	Define an NvBlockSwComponentType on VFB level. The NvBlockSwComponentType defines non volatile data which can be shared between SwComponentPrototypes. The non volatile data of the NvBlockSwComponentType are accessible via provided and required ports.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Basic Software Designer	0..1	
Performed by	ECU Integrator	0..1	
Performed by	Software Component Designer	0..1	
Consumes	VFB Interfaces	1..*	
Consumes	VFB Types	1..*	
Consumes	VFB AUTOSAR Standard Package	0..1	
Consumes	Software Component Internal Behavior	0..*	This input is required to collect the requirements for the NvBlockNeeds from the using application software.
Consumes	VFB Modes	0..*	

Relation Type	*Related Element*	*Mul.*	*Note*
Produces	VFB NvBlock Software Component	1	

Table 3.76: Define VFB NvBlock Software Component

3.2.1.11 Define Wrapper Components to Integrate Legacy Software

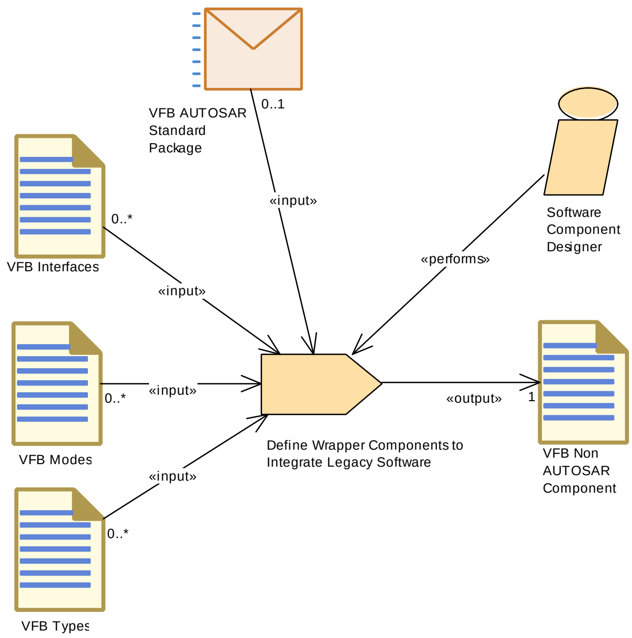

Figure 3.34: Task `Define Wrapper Components to Integrate Legacy Software`

Task Definition	**Define Wrapper Components to Integrate Legacy Software**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define a wrapper component used to represent legacy software that is integrated into an AUTOSAR system.		
Description	Define a wrapper component used to represent legacy software that is integrated into an AUTOSAR system. For the VFB system, this mainly means to define the corresponding port interfaces and data elements.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	Software Component Designer	1	
Consumes	VFB AUTOSAR Standard Package	0..1	Use port blueprints in order to create ports with standardized application interfaces.
Consumes	VFB Interfaces	0..*	
Consumes	VFB Modes	0..*	
Consumes	VFB Types	0..*	
Produces	VFB Non AUTOSAR Component	1	

Table 3.77: Define Wrapper Components to Integrate Legacy Software

3.2.1.12 Define VFB Interfaces

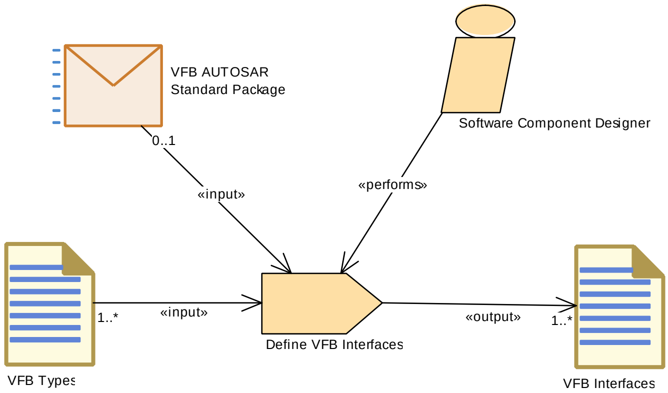

Figure 3.35: Task `Define VFB Interfaces`

Task Definition	**Define VFB Interfaces**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define a set of Port Interface required by a system.		
Description	Define a set of Port Interfaces required by a VFB system, to describe the communication of data via SWC ports.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Designer	1	
Consumes	VFB Types	1..*	
Consumes	VFB AUTOSAR Standard Package	0..1	Use standardized Port Interfaces as blueprints (as far as applicable) to create the corresponding elements of the actual project.
Produces	VFB Interfaces	1..*	

Table 3.78: Define VFB Interfaces

3.2.1.13 Define VFB Types

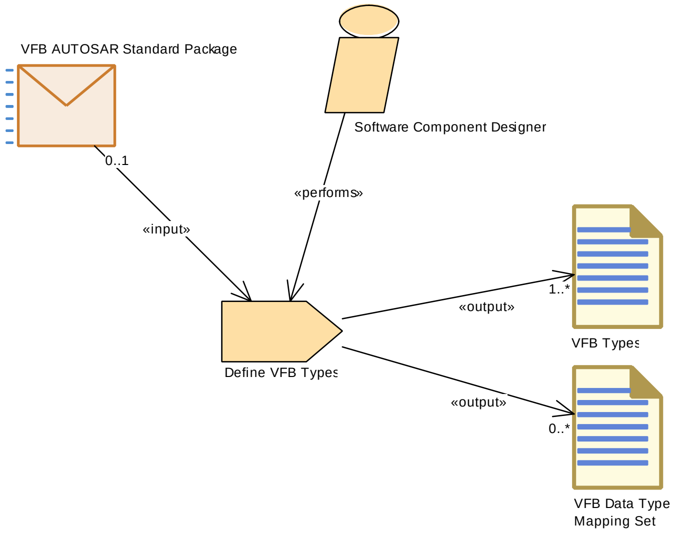

Figure 3.36: Task `Define VFB Types`

Task Definition	**Define VFB Types**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define a set of data types required by a system, but not already defined by AUTOSAR.		
Description	Define a set of Autosar Data Types and related elements as far as visible on the VFB. Standardized types can be used as input in order to copy and refine them. The VFB Types will be used for specifying types of DataElements in Sender-Receiver PortInterfaces and argument/return values of Client-Server PortInterfaces. This task inludes (optionally) also the creation of a VFB Data Type mapping Set between application and implementation data types.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Designer	1	
Consumes	VFB AUTOSAR Standard Package	0..1	Use standardized elements (e.g. Data Types, Compu Methods) as blueprints (as far as applicable) to create the corresponding elements of the actual project.
Produces	VFB Types	1..*	
Produces	VFB Data Type Mapping Set	0..*	

Table 3.79: Define VFB Types

3.2.1.14 Define VFB Modes

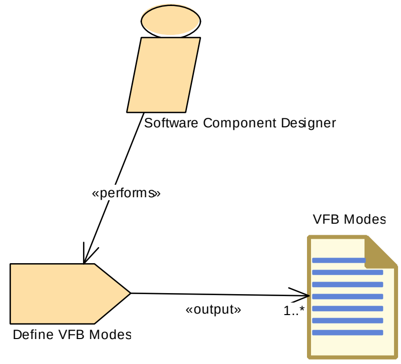

Figure 3.37: Task Define VFB Modes

Task Definition	**Define VFB Modes**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define modes that are used by the VFB components.		
Description	Define modes (mode groups and the modes they contain) that are used by the VFB components.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Designer	1	
Produces	VFB Modes	1..*	

Table 3.80: Define VFB Modes

3.2.1.15 Define VFB Constants

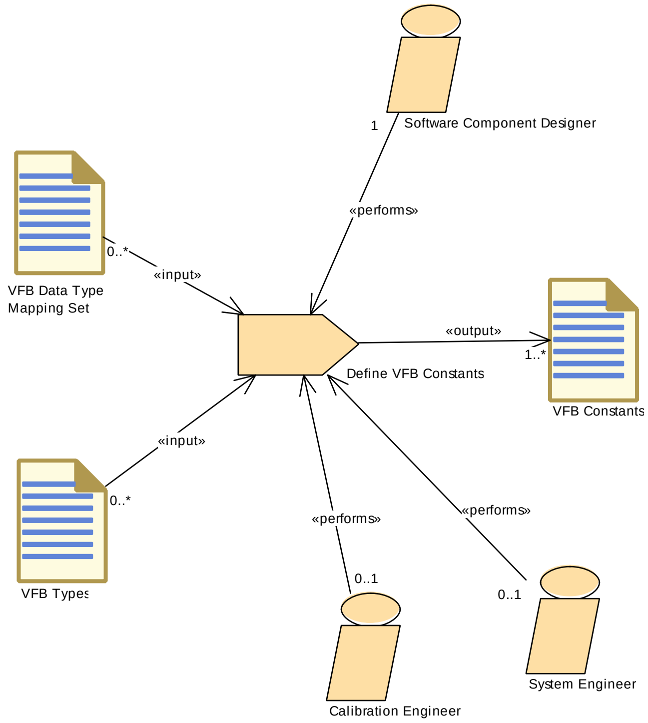

Figure 3.38: Task Define VFB Constants

Task Definition	**Define VFB Constants**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define one or more VFB Constants.		
Description	Define one or more VFB Constants as standalone artifact. Such constants can be referred in the specification of inital values at several places in the VFB descrption, such as port interfaces or declaration of local parameters or variables.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Designer	1	
Performed by	Calibration Engineer	0..1	
Performed by	System Engineer	0..1	
Consumes	VFB Data Type Mapping Set	0..*	
Consumes	VFB Types	0..*	
Produces	VFB Constants	1..*	

Table 3.81: Define VFB Constants

3.2.1.16 Define VFB Timing

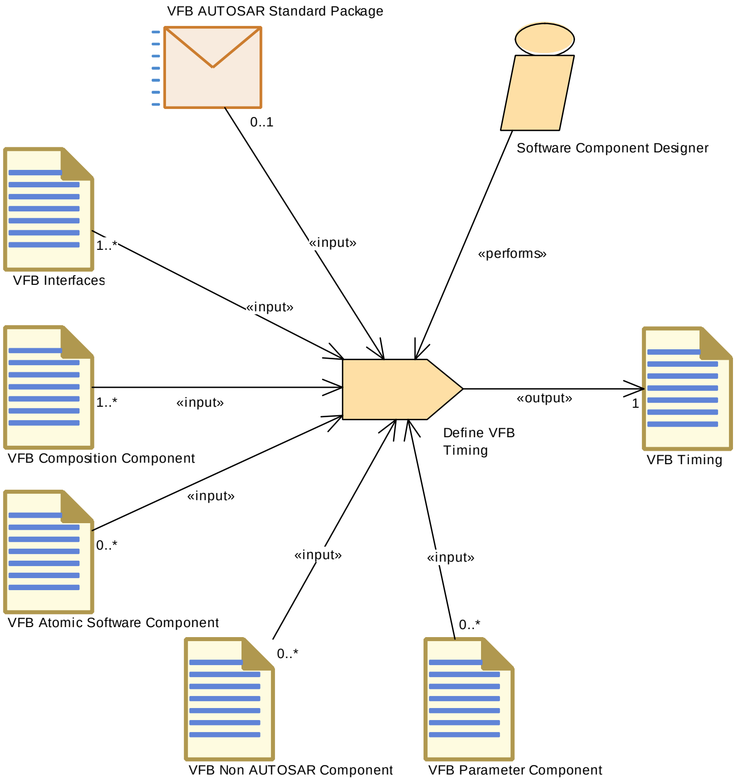

Figure 3.39: Task Define VFB Timing

Task Definition	**Define VFB Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define VFB Timing (TimingDescription and TimingConstraints) for an Atomic Software Component or a Composition Component		
Description	Define VFB Timing (TimingDescription and TimingConstraints) for an Atomic Software Component or a Composition Component		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Designer	1	
Consumes	VFB Composition Component	1..*	
Consumes	VFB Interfaces	1..*	
Consumes	VFB AUTOSAR Standard Package	0..1	
Consumes	VFB Atomic Software Component	0..*	
Consumes	VFB Non AUTOSAR Component	0..*	
Consumes	VFB Parameter Component	0..*	
Produces	VFB Timing	1	

Table 3.82: Define VFB Timing

3.2.1.17 Define VFB Variants

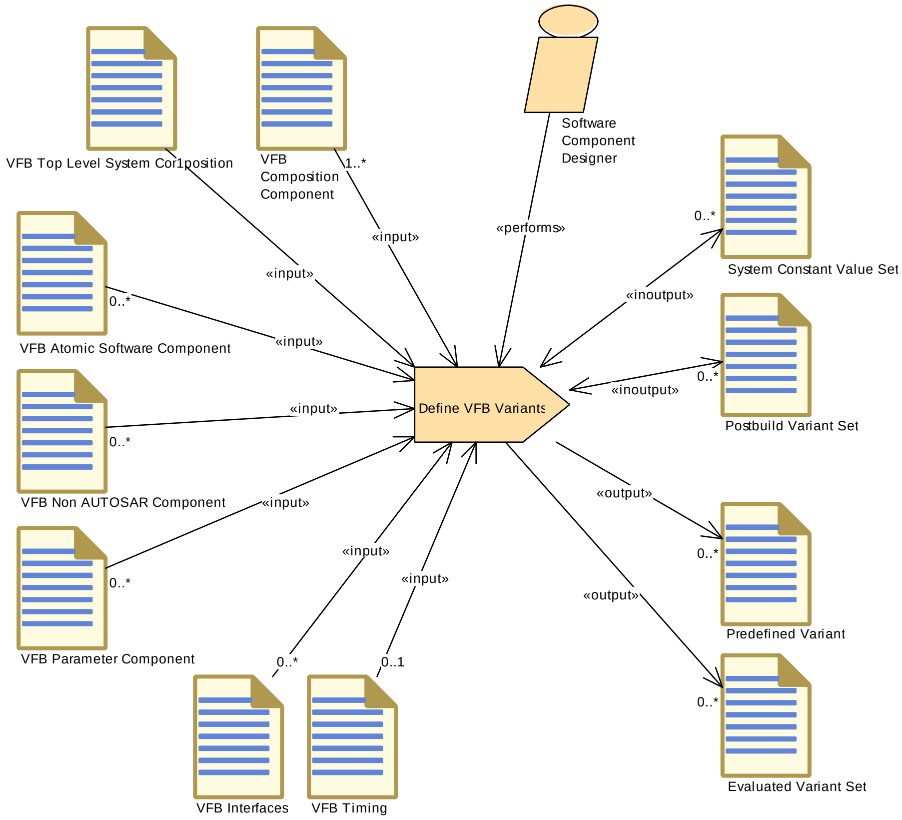

Figure 3.40: Task **Define VFB Variants**

Task Definition	**Define VFB Variants**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define variants for the artifacts of a VFB system.		
Description	Define one or more variants for the artifacts of a VFB system. Defining one variant means creating a Predefined Variant related to the settings used by the VFB elements in scope. To do so, this task can make use of existing System Constant Value Sets and/or Postbuid Variant Sets or define new ones. Several Predefined Variants can be combined to one Evaluated Variant Set.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Designer	1	
Consumes	VFB Top Level System Composition	1	
Consumes	VFB Composition Component	1..*	
Consumes	VFB Timing	0..1	
Consumes	VFB Atomic Software Component	0..*	

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	VFB Interfaces	0..*	
Consumes	VFB Non AUTOSAR Component	0..*	
Consumes	VFB Parameter Component	0..*	
In/out	Postbuild Variant Set	0..*	
In/out	System Constant Value Set	0..*	
Produces	Evaluated Variant Set	0..*	
Produces	Predefined Variant	0..*	

Table 3.83: Define VFB Variants

3.2.1.18 Define VFB Integration Connector

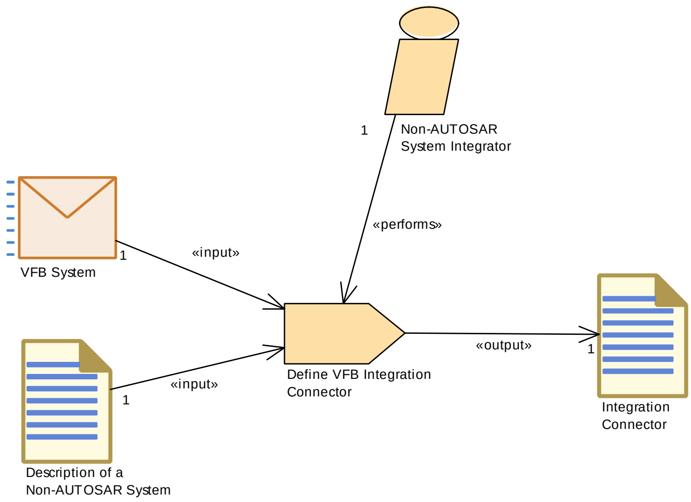

Figure 3.41: Task Define VFB Integration Connector

Task Definition	**Define VFB Integration Connector**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Define how the non-AUTOSAR system shall be connected to the AUTOSAR system.		
Description	The VFB Integration Connector is used to represent the connection of the non-AUTOSAR system and the AUTOSAR system. Its contents and format depend on the way in which the non-AUTOSAR system is defined. To define the VFB Integration Connector the requirements on the connection are brought into the format of the Integration Connector. When the requirements are defined in a proprietary format the have to be translated to the format of the Integration Connector. When they are only informally defined or are even more tangible the format of the Integration Connector can be used to elicit, formalize, and analyze the connection requirements.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Non-AUTOSAR System Integrator	1	
Consumes	Description of a Non-AUTOSAR System	1	
Consumes	VFB System	1	
Produces	Integration Connector	1	
Predecessor	Translate Non-Autosar Description to Autosar Description	1	

Table 3.84: Define VFB Integration Connector

3.2.1.19 Translate Non-AUTOSAR Description to AUTOSAR Description

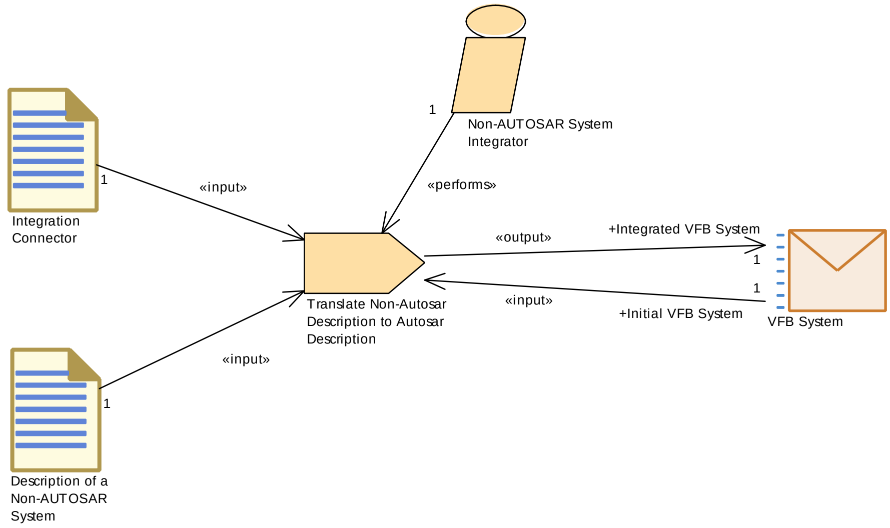

Figure 3.42: Task `Translate Non-AUTOSAR Description to AUTOSAR Description`

Task Definition	**Translate Non-Autosar Description to Autosar Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Tasks		
Brief Description	Translate the description of the non-AUTOSAR system into a semantically equivalent AUTOSAR description (template).		
Description	In order to incorporate the development of the non-AUTOSAR system into the AUTOSAR process the Description of the non-AUTOSAR system must be translated into an AUTOSAR format. Typically this will be achieved by a translation tool, although in principle it might also be done manually.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Non-AUTOSAR System Integrator	1	
Consumes	Description of a Non-AUTOSAR System	1	
Consumes	Integration Connector	1	
Consumes	VFB System	1	Initial VFB System:
Produces	VFB System	1	Integrated VFB System:

Table 3.85: Translate Non-Autosar Description to Autosar Description

3.2.2 Work Products

3.2.2.1 VFB System

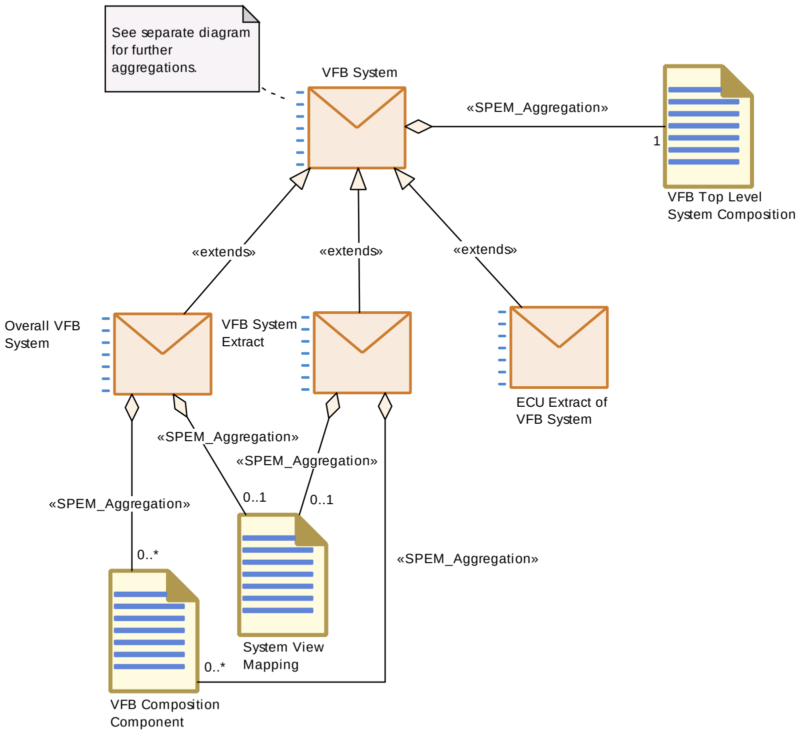

Figure 3.43: Overview on the different roles of Deliverables based on `VFB System`

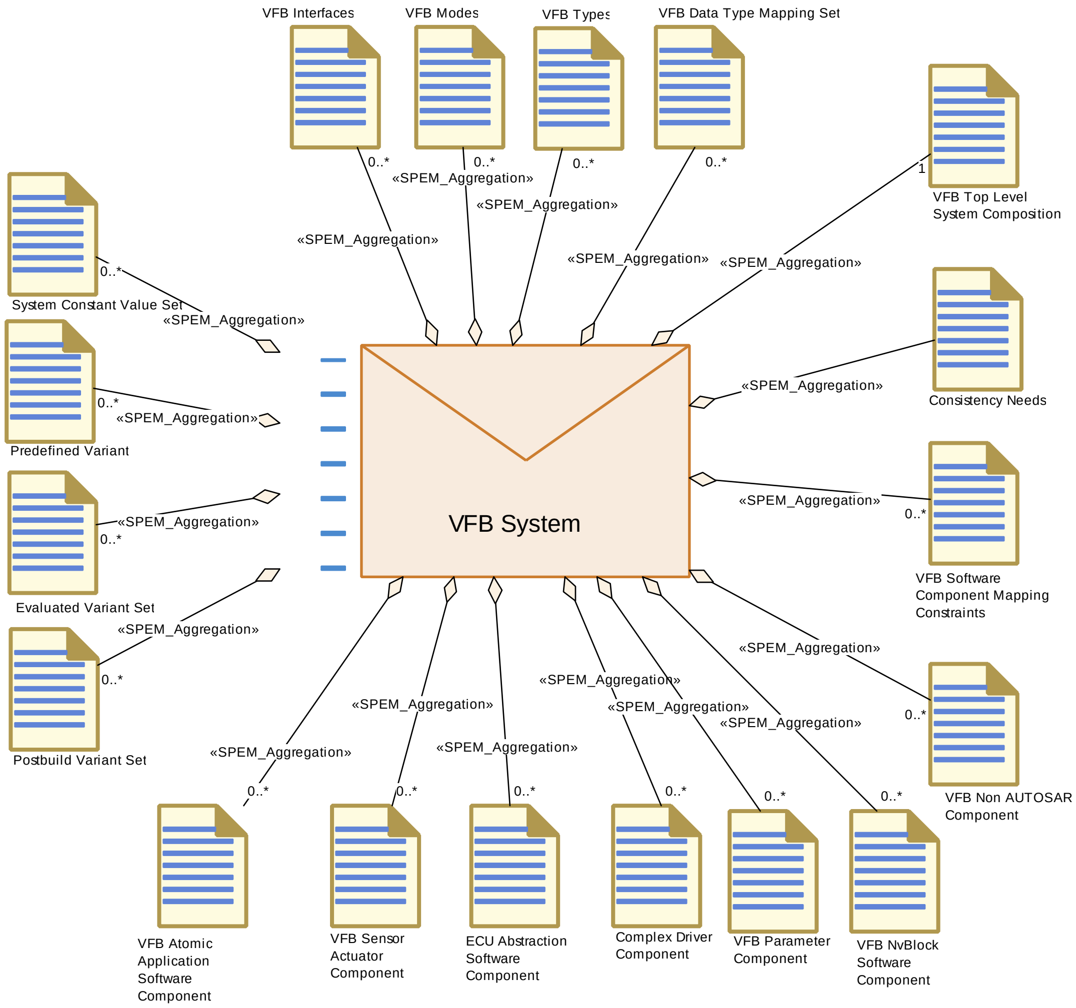

Figure 3.44: Structure of Deliverable VFB System

Deliverable	**VFB System**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Complete VFB view of a concrete system.		
Description	Delivery of a VFB view of a concrete system. i.e. the top level composition and all nested compositions and components. This element is the basis for several extensions according to the scope of the VFB which can be an Overall System, a System Extract or an ECU Extract. This deliverable may contain variation points in its XML artifacts which need to be bound in later steps of the methodology. If such variation points are present, the delivered VFB system may optionally include PredefinedVariants in order to predefine variants for later selection and an Evaluated Variant Set.		
Kind	Delivered		
Extended by	ECU Extract of VFB System, Overall VFB System, VFB System Extract		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Consistency Needs	1	Correlation between a group of RunnableEntitys and a group of DataPrototypes.
Aggregates	VFB Top Level System Composition	1	
Aggregates	Complex Driver Component	0..*	
Aggregates	ECU Abstraction Software Component	0..*	
Aggregates	Evaluated Variant Set	0..*	
Aggregates	Postbuild Variant Set	0..*	
Aggregates	Predefined Variant	0..*	
Aggregates	System Constant Value Set	0..*	
Aggregates	VFB Atomic Application Software Component	0..*	
Aggregates	VFB Data Type Mapping Set	0..*	
Aggregates	VFB Interfaces	0..*	
Aggregates	VFB Modes	0..*	
Aggregates	VFB Non AUTOSAR Component	0..*	
Aggregates	VFB NvBlock Software Component	0..*	
Aggregates	VFB Parameter Component	0..*	
Aggregates	VFB Sensor Actuator Component	0..*	
Aggregates	VFB Software Component Mapping Constraints	0..*	
Aggregates	VFB Types	0..*	
Produced by	Extend Composition	1	extended system:
Produced by	Translate Non-Autosar Description to Autosar Description	1	Integrated VFB System:

Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Define Partial Flat Map	1	Various parts of a given VFB system will be used as input: • Refer to parameters and variables in port interfaces and their data types. • In order to define unique names, also other the component definitions not in the scope of the partial flat map might be checked. • Set a link to the context of the Flat Map, e.g. a VFB Composition.
Consumed by	Define VFB Integration Connector	1	
Consumed by	Define VFB Safety Information	1	
Consumed by	Extend Composition	1	initial system:
Consumed by	Extract the ECU Communication	1	Need as input in order to set up the Data Mapping.
Consumed by	Generate or Adjust System Flat Map	1	
Consumed by	Translate Non-Autosar Description to Autosar Description	1	Initial VFB System:

Table 3.86: VFB System

3.2.2.2 Overall VFB System

Deliverable	**Overall VFB System**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description			
Description	Deliverable containing an overall VFB description. It must contain the VFB Top Level System Composition of the complete system.		
Kind	Delivered		
Extends	VFB System		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Abstract System Description	1	
Aggregated by	System Configuration Description	1	
Aggregated by	System Constraint Description	0..1	

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	System View Mapping	0..1	The Overall VFB System aggregates a potential mapping to the abstract or functional view of the system.
Aggregates	VFB Composition Component	0..*	Further compositions below the top level composition.
Produced by	Develop a VFB System Description	1	
Consumed by	Define Software Component Safety Information	1	
Consumed by	Develop Application Software	1	The application software needs to refer to the relevant elements of the overall VFB system such as Software Component Types, Port Interfaces and Data Types.
Consumed by	Develop System	0..1	Usually the System refers to elements of an overall VFB descriptions. But for the description of a legacy system, this input might be empty.
Consumed by	Flatten Software Composition	0..1	Read relevant elements starting from VFB Top Level System Composition in case transformation starts with the full system.
Consumed by	Generate or Adjust ECU Flat Map	0..1	Used to set the upstream references in case one starts from a complete system.

Table 3.87: Overall VFB System

3.2.2.3 VFB System Extract

Deliverable	**VFB System Extract**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	The VFB description for the partial system.		
Description	The VFB description for a sub-system. It contains only those software components which belong to this sub-system. It should contain a VFB Top Level System Composition which has unconnected ports reflecting the connection points to the outer system.		
Kind	Delivered		
Extends	VFB System		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	System Extract	1	
Aggregates	System View Mapping	0..1	The VFB System Extract aggregates a potential mapping to the abstract or functional view of the system.
Aggregates	VFB Composition Component	0..*	Further compositions below the top level composition.

Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Flatten Software Composition	0..1	Read relevant elements starting from VFB Top Level System Composition in case transformation starts from the system extract.
Consumed by	Generate or Adjust ECU Flat Map	0..1	Used to set the upstream references in case one starts from a system extract.

Table 3.88: VFB System Extract

3.2.2.4 VFB Top Level System Composition

Artifact	**VFB Top Level System Composition**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Highest Level Composition consisting of all components that make up the Virtual Functional Bus.		
Description	Highest Level Composition consisting of all components and their connectors that make up the VFB System Deliverable. This composition is not allowed to have ports if it represents the top level composition of an Overall VFB System, but it may have unconnected ports (and port groups) if it is at the top of a System Extract or ECU Extract.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	VFB System	1	
Produced by	Define VFB Top Level	1	
Consumed by	Assign Top Level Composition	1	
Consumed by	Define VFB Component Constraints	1	
Consumed by	Define VFB Variants	1	
Consumed by	Deploy Software Component	1	
Use meta model element	CompositionSwComponentType	1	

Table 3.89: VFB Top Level System Composition

3.2.2.5 VFB Composition Component

Artifact	**VFB Composition Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Describes a set of VFB CompositionTypes.		
Description	Describes a set of CompositionComponentTypes, which may be nested. A VFB composition aggregates component types to encapsulate and abstract subsystem functionality. Compositions contain instances of components (other compositions and atomic components), as well as the connectors between them.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	Delivered Atomic Software Components	0..*	In case the delivered atomic components make up one or more VFB Compositions, the composition description(s) shall be included in the delivery.
Aggregated by	Overall VFB System	0..*	Further compositions below the top level composition.
Aggregated by	VFB System Extract	0..*	Further compositions below the top level composition.
Produced by	Define VFB Composition Component	1	
Produced by	Extend Composition	0..*	
Consumed by	Set System Root	1	Only the reference to the artifact is needed
Consumed by	Define VFB Component Constraints	1..*	
Consumed by	Define VFB Timing	1..*	
Consumed by	Define VFB Variants	1..*	
Consumed by	Define VFB Composition Component	0..*	
Consumed by	Define VFB Top Level	0..*	
Use meta model element	CompositionSw ComponentType	1	
Use meta model element	SwComponent Type	1	

Table 3.90: VFB Composition Component

3.2.2.6 VFB AUTOSAR Standard Package

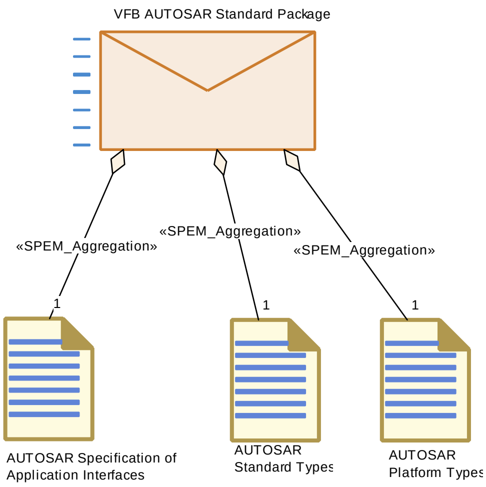

Figure 3.45: Structure of Deliverable VFB AUTOSAR Standard Package

Deliverable	**VFB AUTOSAR Standard Package**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Package with standardized AUTOSAR DataTypes, PortInterfaces, ComponentTypes (may include compositions), etc. on VFB level.		
Description	Package with standardized AUTOSAR elements needed on VFB level. This deliverable is released by AUTOSAR and is readonly within the methodology.		
Kind	Delivered		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	AUTOSAR Platform Types	1	
Aggregates	AUTOSAR Specification of Application Interfaces	1	
Aggregates	AUTOSAR Standard Types	1	
Consumed by	Define ECU Abstraction Component	1	Use port blueprints in order to create ports with standardized application interfaces.
Consumed by	Develop a VFB System Description	1..*	
Consumed by	Develop an Abstract System Description	1..*	
Consumed by	Define Atomic Software Component Internal Behavior	0..1	Use standardized elements (e.g. Data Types) as blueprints (as far as applicable) to create the corresponding elements of the actual project.

Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Define Complex Driver Component	0..1	Use port blueprints in order to create ports with standardized application interfaces.
Consumed by	Define VFB Application Software Component	0..1	Use port blueprints in order to create ports with standardized application interfaces.
Consumed by	Define VFB Composition Component	0..1	Use port blueprints in order to create ports with standardized application interfaces.
Consumed by	Define VFB Interfaces	0..1	Use standardized Port Interfaces as blueprints (as far as applicable) to create the corresponding elements of the actual project.
Consumed by	Define VFB Nv Block Software Component	0..1	
Consumed by	Define VFB Parameter Component	0..1	Use port blueprints in order to create ports with standardized application interfaces.
Consumed by	Define VFB Sensor or Actuator Component	0..1	Use port blueprints in order to create ports with standardized application interfaces.
Consumed by	Define VFB Timing	0..1	
Consumed by	Define VFB Types	0..1	Use standardized elements (e.g. Data Types, Compu Methods) as blueprints (as far as applicable) to create the corresponding elements of the actual project.
Consumed by	Define Wrapper Components to Integrate Legacy Software	0..1	Use port blueprints in order to create ports with standardized application interfaces.
Consumed by	Generate Atomic Software Component Contract Header Files	0..1	
Consumed by	Generate Component Header File in Vendor Mode	0..1	
Consumed by	Generate Component Prebuild Data Set	0..1	

Table 3.91: VFB AUTOSAR Standard Package

3.2.2.7 AUTOSAR Specification of Application Interfaces

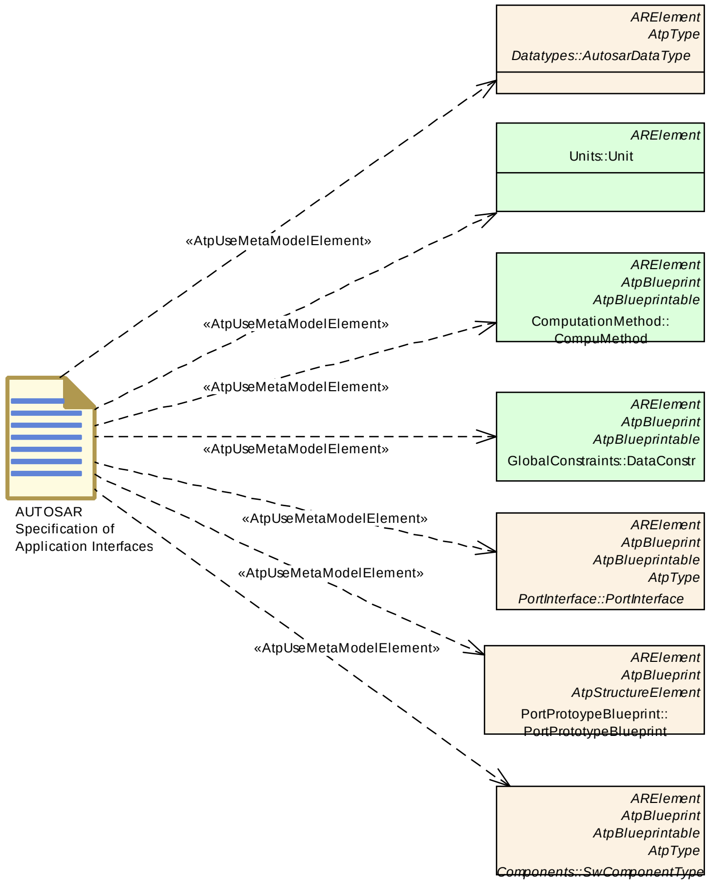

Figure 3.46: The AUTOSAR Specification of Application Interfaces

Artifact	**AUTOSAR Specification of Application Interfaces**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Definitions of the AUTOSAR standard appliction interfaces.		
Description	This includes standardized data types, port interfaces, units, port blueprints and example component types (including compositions) for the design of Application Software Components. Note that most of the content is not meant as direct input for defining a VFB system but as so-called blueprints: Blueprints need to be completed with company or project specific elements (e.g. a component type defined as blueprint may need additional ports or a data type defined as blueprint may need additional properties).		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	VFB AUTOSAR Standard Package	1	
Use meta model element	AutosarDataType	1	
Use meta model element	CompuMethod	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Use meta model element	DataConstr	1	
Use meta model element	PortInterface	1	
Use meta model element	PortPrototype Blueprint	1	
Use meta model element	SwComponent Type	1	
Use meta model element	Unit	1	

Table 3.92: AUTOSAR Specification of Application Interfaces

3.2.2.8 VFB Atomic Software Component

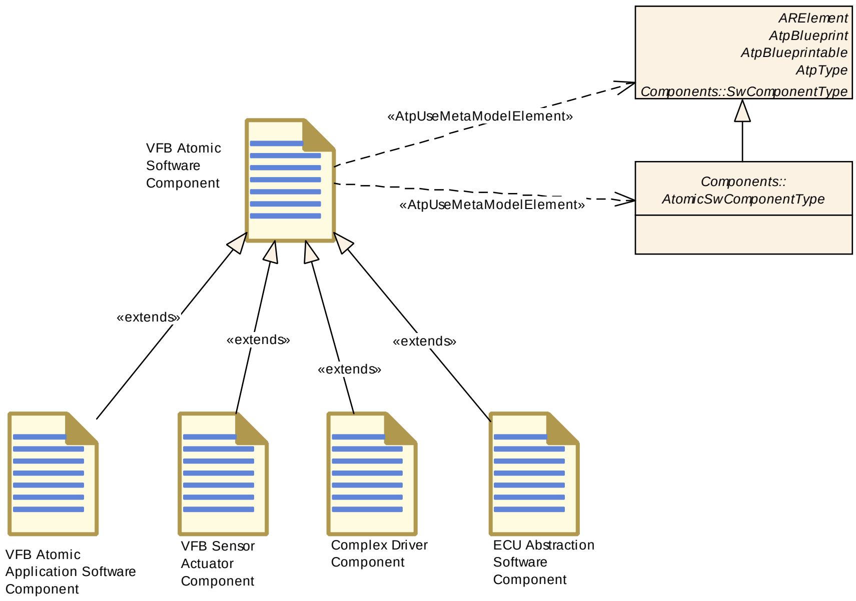

Figure 3.47: The Generic Work Product VFB Atomic Software Component

Artifact	**VFB Atomic Software Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Description of an Atomic VFB Component.		
Description	The description of an Atomic Software Component Type without Internal Behavior. Note that there are more specific artifacts extending this one. This artifact is used to describe general use cases which are valid for all kind of Atomic Software Components.		
Kind	AUTOSAR XML		
Extended by	Complex Driver Component, ECU Abstraction Software Component, VFB Atomic Application Software Component, VFB Sensor Actuator Component		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	Delivered Atomic Software Components	1..*	

Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define Symbol Props for Types	0..*	symbolProps: The symbolProps attribute redefines the software component type name used in the code of the RTE. This resolves name clashes among different software component types designed accidentally with the same shortName. Note that this output is a splitable element, so it can be added later without changing the VFB model.
Produced by	Extend Composition	0..*	
Consumed by	Define VFB Component Constraints	2..*	
Consumed by	Define Atomic Software Component Internal Behavior	1	
Consumed by	Generate Atomic Software Component Contract Header Files	1	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Component Header File in Vendor Mode	1	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Component Prebuild Data Set	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Select Software Component Implementation	1..*	
Consumed by	Define Consistency Needs	0..*	The description of an AtomicSoftwareComponentType without InternalBehavior.
Consumed by	Define VFB Composition Component	0..*	
Consumed by	Define VFB Timing	0..*	
Consumed by	Define VFB Top Level	0..*	
Consumed by	Define VFB Variants	0..*	
Use meta model element	AtomicSwComponentType	1	
Use meta model element	SwComponent Type	1	

Table 3.93: VFB Atomic Software Component

3.2.2.9 VFB Atomic Application Software Component

Artifact	**VFB Atomic Application Software Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Description of an Atomic VFB Component.		
Description	The description of an Application Software Component Type. It is used to represent the ECU-independent application software.		
Kind	AUTOSAR XML		
Extends	VFB Atomic Software Component		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	VFB System	0..*	
Produced by	Define VFB Application Software Component	1	
Use meta model element	ApplicationSw ComponentType	1	

Table 3.94: VFB Atomic Application Software Component

3.2.2.10 Complex Driver Component

Artifact	**Complex Driver Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	VFB Description of a Complex Driver Component.		
Description	The Complex Driver Component is a special VFB Atomic Software Component that has direct access to hardware on an ECU and which is therefore linked to a specific ECU or specific hardware. It uses the meta-model element ComplexDeviceDriverSwComponentType which introduces the possibility to link from the software representation to its hardware description provided by the ECU Resource Template. It provides (non-standardized) AUTOSAR Interfaces via ports on VFB level.		
Kind	AUTOSAR XML		
Extends	VFB Atomic Software Component		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	VFB System	0..*	
Produced by	Define Complex Driver Component	1	
Consumed by	Configure Debug	0..1	
Consumed by	Map Software Component to BSW	0..1	

Relation Type	Related Element	Mul.	Note
Use meta model element	ComplexDeviceDriverSwComponentType	1	

Table 3.95: Complex Driver Component

3.2.2.11 ECU Abstraction Software Component

Artifact	**ECU Abstraction Software Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	VFB Description of an ECU Abstraction Software Component.		
Description	The ECU Abstraction Software Component is a special Atomic Software Component that sits between a component that wants to access ECU periphery (typically a Sensor Actuator Component) and the Microcontroller Abstraction. It provides (non-standardized) AUTOSAR Interfaces via ports which represent the ECU periphery. The EcuAbstractionSwComponentType introduces the possibility to link from the software representation to its hardware description provided by the ECU Resource Template. During integration, an ECU Abstraction Software Component will be mapped to a BSW module which implements it and which will directly (without RTE) be connected to the Microcontroller Abstraction.		
Kind	AUTOSAR XML		
Extends	VFB Atomic Software Component		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	VFB System	0..*	
Produced by	Define ECU Abstraction Component	1	
Consumed by	Map Software Component to BSW	0..1	
Use meta model element	EcuAbstractionSwComponentType	1	

Table 3.96: ECU Abstraction Software Component

3.2.2.12 VFB Parameter Component

Artifact	**VFB Parameter Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	A ParameterComponentType defines parameters and characteristic values accessible via provided Ports.		
Description	A ParameterSwComponentType defines parameters and characteristic values accessible via Provide Ports. The provided values are the same for all connected Component Prototypes. This is as opposed to private parameters which are only available within the scope of an Atomic Software Component		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	VFB System	0..*	
Produced by	Define VFB Parameter Component	1	
Produced by	Extend Composition	0..*	
Consumed by	Define VFB Composition Component	0..*	
Consumed by	Define VFB Timing	0..*	
Consumed by	Define VFB Top Level	0..*	
Consumed by	Define VFB Variants	0..*	
Use meta model element	ParameterSw ComponentType	1	

Table 3.97: VFB Parameter Component

3.2.2.13 VFB Sensor Actuator Component

Artifact	**VFB Sensor Actuator Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Describes a sensor or actuator component that exist at the VFB Level and represents the physical interface of an actual sensor or actuator hardware element.		
Description	A Sensor Actuator Software Component is an Atomic Software Component that makes the functionality of a sensor or actuator usable for other software components. That means that the Sensor Actuator Software Component provides to the application software components an interface for the physical values of the sensors and actuators. It is written for a concrete sensor or actuator and uses the ECU Abstraction interface. It references the description of the associated hardware elements.		
Kind	AUTOSAR XML		
Extends	VFB Atomic Software Component		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Complete ECU Description	0..*	
Aggregated by	VFB System	0..*	
Produced by	Define VFB Sensor or Actuator Component	1	
Use meta model element	SensorActuatorSwComponentType	1	

Table 3.98: VFB Sensor Actuator Component

3.2.2.14 VFB NvBlock Software Component

Artifact	**VFB NvBlock Software Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description			
Description	The VFB NvBlock Software Component defines non volatile data which can be shared between SwComponentPrototypes. The non volatile data of the VFB NvBlock Software Component are accessible via provided and required ports.		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	VFB System	0..*	
Produced by	Define VFB Nv Block Software Component	1	
Use meta model element	NvBlockSwComponentType	1	

Table 3.99: VFB NvBlock Software Component

3.2.2.15 VFB Non AUTOSAR Component

Artifact	**VFB Non AUTOSAR Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	A Component used to describe the non-autosar entities that exist at the VFB level.		
Description	A Component used to describe the non-AUTOSAR entities that exist at the VFB level.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	VFB System	0..*	
Produced by	Define Wrapper Components to Integrate Legacy Software	1	
Produced by	Extend Composition	0..*	
Consumed by	Define VFB Composition Component	0..*	
Consumed by	Define VFB Timing	0..*	
Consumed by	Define VFB Top Level	0..*	
Consumed by	Define VFB Variants	0..*	
Use meta model element	SwComponent Type	1	

Table 3.100: VFB Non AUTOSAR Component

3.2.2.16 VFB Interfaces

Artifact	**VFB Interfaces**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Interfaces and related elements that form part of the VFB, but are not standardized by AUTOSAR.		
Description	Interfaces and related elements that form part of the VFB, but are not standardized by AUTOSAR.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Delivered Atomic Software Components	0..*	
Aggregated by	VFB System	0..*	
Produced by	Define VFB Interfaces	1..*	

Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Extend Composition	0..*	
Consumed by	Define ECU Abstraction Component	1	
Consumed by	Define Complex Driver Component	1..*	
Consumed by	Define VFB Application Software Component	1..*	
Consumed by	Define VFB Composition Component	1..*	
Consumed by	Define VFB Nv Block Software Component	1..*	
Consumed by	Define VFB Parameter Component	1..*	
Consumed by	Define VFB Sensor or Actuator Component	1..*	
Consumed by	Define VFB Timing	1..*	
Consumed by	Define VFB Top Level	1..*	
Consumed by	Define Consistency Needs	0..*	Interfaces which are relevant for the consistency definition.
Consumed by	Define VFB Variants	0..*	
Consumed by	Define Wrapper Components to Integrate Legacy Software	0..*	
Consumed by	Generate Atomic Software Component Contract Header Files	0..*	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Component Header File in Vendor Mode	0..*	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Component Prebuild Data Set	0..*	Meth.bindingTime = CodeGenerationTime
Use meta model element	AutosarDataType	1	
Use meta model element	ModeDeclarationGroup	1	
Use meta model element	PortInterface	1	

Table 3.101: VFB Interfaces

3.2.2.17 VFB Types

Artifact	**VFB Types**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Data types and related elements that form part of the VFB, but are not standardized by AUTOSAR.		
Description	Description of AutosarDataTypes and related elements (e.g. units, computation methods, etc.) that form part of the VFB, but are not standardized by AUTOSAR. This may also include copies of standardized elements which have been completed with project specific information (e.g. with calibration access information or computation methods). A VFB system can contain several different instances of this artifact, which may fulfill different roles. AutosarDataTypes can come as so-called ApplicationDatatypes or ImplementationDataTypes. This package can contain both kinds but they can also be split into separate artifacts. However, since it is also possible to generate ImplementationDataTypes from ApplicationDataTypes, a VFB system can be completely defined with ApplicationDatatypes only. Note that this work product is meant for use cases, in which a set of data types is maintained as a separate artifact. It is also possible to define particular AutosarDataTypes as part of another artifact, e.g. of VFB Interfaces if the types are closely related to certain port interfaces. In the methodology this artifact stands not only for data type definitions, but also for related elements like addressing methods, units, computation methods, constraints. etc. This is done for simplicity, because these elements are often consumed by the same tasks. Of course these can be treated as separate artifacts in real projects.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Delivered Atomic Software Components	0..*	
Aggregated by	VFB System	0..*	
Produced by	Define VFB Types	1..*	
Produced by	Define Symbol Props for Types	0..*	symbolProps: The symbolProps attribute redefines the implementation data type name used in the code of the RTE and/or the component. This resolves name clashes among different implementation data types designed accidentally with the same shortName. Note that this output is a splitable element, so it can be added later without changing the VFB model.
Produced by	Extend Composition	0..*	

Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Define ECU Abstraction Component	1	
Consumed by	Define Complex Driver Component	1..*	
Consumed by	Define VFB Application Software Component	1..*	
Consumed by	Define VFB Composition Component	1..*	
Consumed by	Define VFB Interfaces	1..*	
Consumed by	Define VFB Nv Block Software Component	1..*	
Consumed by	Define VFB Parameter Component	1..*	
Consumed by	Define VFB Sensor or Actuator Component	1..*	
Consumed by	Define VFB Top Level	1..*	
Consumed by	Generate BSW Memory Mapping Header	1..*	SwAddrMethod: Referred SwAddrMethods Meth.bindingTime = SystemDesignTime
Consumed by	Generate Compiler Configuration	1..*	SwAddrMethod: Referred SwAddrMethods. They provide the default names for the compiler memory classes. Meth.bindingTime = SystemDesignTime
Consumed by	Generate SWC Memory Mapping Header	1..*	SwAddrMethod: Referred SwAddrMethods Meth.bindingTime = SystemDesignTime
Consumed by	Configure Memmap Allocation	0..*	SwAddrMethods: SwAddrMethods used for the generic mapping. Note that one SwAddrmethod can represent several memory sections.
Consumed by	Define Consistency Needs	0..*	Data types which are relevant for the consistency definition.
Consumed by	Define VFB Constants	0..*	
Consumed by	Define Wrapper Components to Integrate Legacy Software	0..*	
Consumed by	Generate Atomic Software Component Contract Header Files	0..*	Meth.bindingTime = SystemDesignTime

Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Generate Component Header File in Vendor Mode	0..*	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Component Prebuild Data Set	0..*	Meth.bindingTime = CodeGenerationTime
Use meta model element	ApplicationDataType	1	
Use meta model element	AutosarDataType	1	
Use meta model element	CompuMethod	1	
Use meta model element	DataConstr	1	
Use meta model element	ImplementationDataType	1	
Use meta model element	SwAddrMethod	1	
Use meta model element	Unit	1	

Table 3.102: VFB Types

3.2.2.18 VFB Data Type Mapping Set

Artifact	**VFB Data Type Mapping Set**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Mapping Set between Application and Implementation Data Types.		
Description	Mapping Set between Application and Implementation Data Types.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Delivered Atomic Software Components	0..*	
Aggregated by	VFB System	0..*	
Produced by	Define VFB Types	0..*	
Consumed by	Generate Atomic Software Component Contract Header Files	0..1	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Component Header File in Vendor Mode	0..1	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Component Prebuild Data Set	0..1	Meth.bindingTime = CodeGenerationTime
Consumed by	Define VFB Constants	0..*	
Use meta model element	DataTypeMappingSet	1	

Table 3.103: VFB Data Type Mapping Set

3.2.2.19 VFB Modes

Artifact	**VFB Modes**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Modes declared here are non-AUTOSAR standard. They are modes that are managed by a software component acting as a application mode manager.		
Description	Desclaration of mode groups and of the modes they contain. Modes declared here are non-AUTOSAR standard. They are modes that are managed by an application software component acting as a mode manager.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Delivered Atomic Software Components	0..*	
Aggregated by	VFB System	0..*	
Produced by	Define VFB Modes	1..*	
Produced by	Extend Composition	0..*	
Consumed by	Define Complex Driver Component	0..*	
Consumed by	Define ECU Abstraction Component	0..*	
Consumed by	Define VFB Application Software Component	0..*	
Consumed by	Define VFB Composition Component	0..*	
Consumed by	Define VFB Nv Block Software Component	0..*	
Consumed by	Define VFB Top Level	0..*	
Consumed by	Define Wrapper Components to Integrate Legacy Software	0..*	
Consumed by	Generate Atomic Software Component Contract Header Files	0..*	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Component Header File in Vendor Mode	0..*	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Component Prebuild Data Set	0..*	Meth.bindingTime = CodeGenerationTime

Relation Type	**Related Element**	**Mul.**	**Note**
Use meta model element	ModeDeclaration Group	1	

Table 3.104: VFB Modes

3.2.2.20 VFB Constants

Artifact	**VFB Constants**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Specification of constant data for usage as initial values by other artifacts.		
Description	Specification of constant data for usage as initial values by other artifacts, e.g. initial values for calibration parameters or variable data elements provided in ports. By using the ConstantSpecification meta-class, such data can be standalone artifacts and thus be maintained independently of the components or interfaces to which they apply.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Produced by	Define VFB Constants	1..*	
Use meta model element	ConstantSpecification	1	

Table 3.105: VFB Constants

3.2.2.21 VFB Software Component Mapping Constraints

Artifact	**VFB Software Component Mapping Constraints**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	A defined constraint on how certain components must be mapped (clustered or separated) to ECUs.		
Description	These MappingConstraints define constraints describing which components need to be mapped to a single ECU, and which must be mapped to separate ECUs, without regard to any particular ECU or topology. The ComponentClustering constraint (also, clustering) is to be used for expressing that a certain set of SW components (atomic or not) shall be mapped (allocated) onto the same ECU. This is some kind of "execute together on same ECU" constraint. The semantic of the clustering constraint is straightforward if all concerned SW components are atomic. Otherwise, it shall be interpreted as follows: all of the atomic SW components making up the composition shall be mapped together onto the same ECU together with all other SW components (atomic or not) affected by the constraint. This also means that a clustering constraint can also refer to only a single composition. The ComponentSeparation constraint (also, separation) is to be used for expressing that two SW components (atomic or not) shall not be mapped (allocated) onto the same ECU. This is some kind of "do not execute together on same ECU" constraint. The semantic of the separation constraint is straightforward if one or both SW components are atomic. Otherwise, it shall be interpreted as follows: any of the atomic SW components making up the first composition, shall not be mapped onto the same ECU with any atomic SW component from the second composition. As a consequence, and to preserve consistency, an atomic SW component instance cannot be part of two compositions concerned by the same separation constraint, i.e. the two compositions have to be disjoint with regards to component instances.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	VFB System	0..*	
Produced by	Define VFB Component Constraints	1..*	
Produced by	Extend Composition	0..*	
Consumed by	Deploy Software Component	0..1	
Use meta model element	MappingConstraint	1	
Use meta model element	SystemMapping	1	The splitable element SystemMapping is the root for this artifact.

Table 3.106: VFB Software Component Mapping Constraints

3.2.2.22 VFB Timing

Artifact	**VFB Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Atomic Software Component or Composition Component TimingDescription and TimingConstraints		
Description	TimingDescription and TimingConstraints defined for an Atomic Software Component or a Composition Component		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Define VFB Timing	1	
Consumed by	Define Software Component Timing	0..1	
Consumed by	Define System Timing	0..1	
Consumed by	Define VFB Variants	0..1	
Use meta model element	VfbTiming	1	

Table 3.107: VFB Timing

3.2.2.23 Description of a Non-AUTOSAR System

Artifact	**Description of a Non-AUTOSAR System**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	View of the non-AUTOSAR system that contains the relevant information for its integration with the AUTOSAR system at VFB level		
Description	This artifact describes the elements of the non-AUTOSAR system that are relevant for its integration with an AUTOSAR system at the VFB level. The format of the description depends on the methodology or platform that is employed for the development of the non-AUTOSAR system. It may not be assumed that the description of the non-AUTOSAR system comes in an AUTOSAR format. Also the contents of the description may differ both in its scope and in its details from an AUTOSAR description that also addresses the VFB level, i.e. a SwComponent Description. The interfaces of infotainment system components developed on the basis of the GENIVI platform for instance are specified with the Franca Interface Definition Language. A Franca IDL description contains interfaces that define data types, methods, attributes, and broadcasts. It does neither define the components that implement these interfaces nor their connections. In addition, the granularity of the data type description is much coarser than a data type description with the SwComponent Template.		
Kind	Custom		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumed by	Define VFB Integration Connector	1	

Relation Type	Related Element	Mul.	Note
Consumed by	Translate Non-Autosar Description to Autosar Description	1	

Table 3.108: Description of a Non-AUTOSAR System

3.2.2.24 Integration Connector

Artifact	**Integration Connector**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::VFB::Work Products		
Brief Description	Specification of the connections of the elements of the non-AUTOSAR system with the elements of the AUTOSAR system		
Description	This artifact specifies which elements of the non-AUTOSAR system are to be connected with which elements of the AUTOSAR system. If for instance the Description of the non-AUTOSAR system contains elements corresponding to port instances, the integration connector would define how these ports are connected with the port instances contained in the AUTOSAR SwComponent Description. In addition, the Integration Connector may specify information that is necessary for the integration but not yet contained in the Description of the non-AUTOSAR system. If for instance the Description of the non-AUTOSAR system contains only very coarse grained data type descriptions the Integration Connector will be used to add sufficient information such that the compatibility of the data types with the ones defined in the AUTOSAR SwComponent Description can be checked.		
Kind	Custom		
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Define VFB Integration Connector	1	
Consumed by	Translate Non-Autosar Description to Autosar Description	1	

Table 3.109: Integration Connector

3.3 System

This chapter contains the definition of work products and tasks used for the development of systems and sub-systems. For the definition of the relevant meta-model elements refer to [7] and [22].

3.3.1 Tasks

3.3.1.1 Set System Root

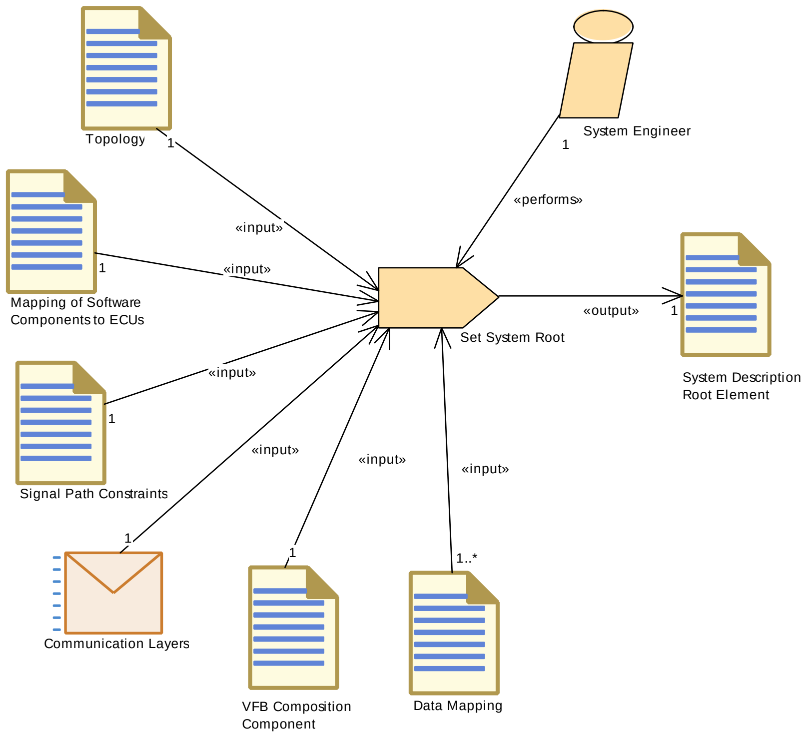

Figure 3.48: Set System Root

Task Definition	**Set System Root**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description			
Description	Set up the root element of a system description.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Communication Layers	1	Only the reference to the artifact is needed
Consumes	Mapping of Software Components to ECUs	1	Only the reference to the artifact is needed
Consumes	Signal Path Constraints	1	Only the reference to the artifact is needed
Consumes	Topology	1	Only the reference to the artifact is needed
Consumes	VFB Composition Component	1	Only the reference to the artifact is needed
Consumes	Data Mapping	1..*	Only the reference to the artifact is needed
Produces	System Description Root Element	1	Set up the root element, and the links to other artifacts

Relation Type	***Related Element***	***Mul.***	***Note***

Table 3.110: Set System Root

3.3.1.2 Assign Top Level Composition

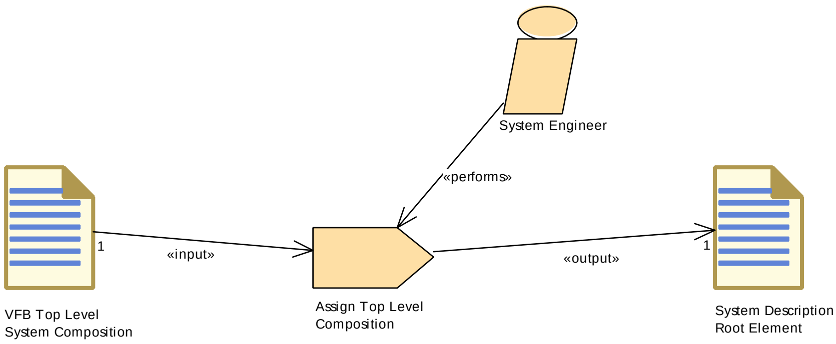

Figure 3.49: Assign Top Level Composition

Task Definition	**Assign Top Level Composition**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description			
Description	Assign a VFB Top Level Composition to the System Root		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	VFB Top Level System Composition	1	
Produces	System Description Root Element	1	

Table 3.111: Assign Top Level Composition

3.3.1.3 Define ECU Description

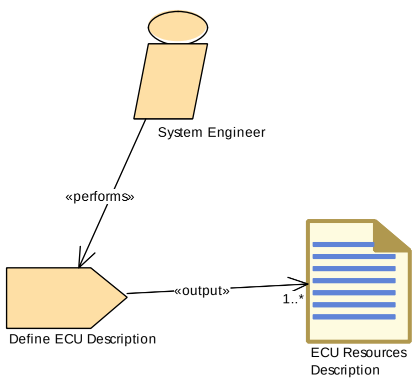

Figure 3.50: Define ECU description

Task Definition	**Define ECU Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description	Define a particular ECU's resources.		
Description	Define a particular ECU's resources by describing Hardware Elements, pins, connections.The HW Elements are the main describing elements of an ECU,e.g processing units, memory, peripherals, sensors and actuators. HW Elements have a unique name and can be identified within the ECU description. HW Elements do not necessarily have to be described on the level of an ECU. It is possible to describe HW Elements as parts of other HW Elements. By this means, a hierarchical description of HW Elements can be created. HW Elements provide HW PinGroups and HW Pins for being interconnected among each others. HW PinGroups allow a rough description of how certain groups of HWPins are arranged. The detailed description can be done using the HW Pins.HW Connections are used to describe connection on several levels:connections between HW Elements, connections between HW PinGroups, connections between HW Pins.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Produces	ECU Resources Description	1..*	

Table 3.112: Define ECU Description

3.3.1.4 Define System Topology

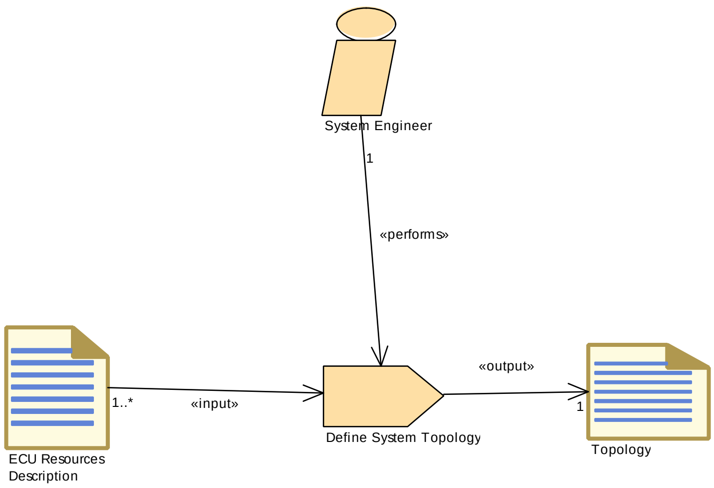

Figure 3.51: Define System Topology

Task Definition	**Define System Topology**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description	Select the ECUs and how the they are interconnected by networks.		
Description	Define how the ECUs of a system are interconnected by networks.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	ECU Resources Description	1..*	
Produces	Topology	1	

Table 3.113: Define System Topology

3.3.1.5 Deploy Software Component

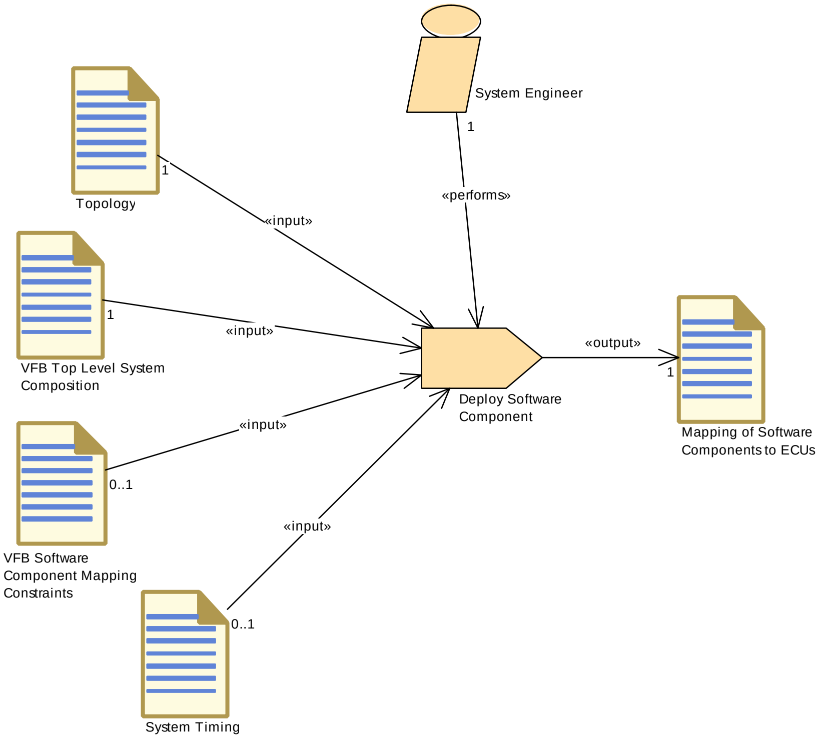

Figure 3.52: Deploy Software Component

Task Definition	**Deploy Software Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description	Deploy VFB Software Components to an ECU		
Description	Deploy each VFB Software Component to an ECU that will execute the component.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Topology	1	

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	VFB Top Level System Composition	1	
Consumes	System Timing	0..1	
Consumes	VFB Software Component Mapping Constraints	0..1	
Produces	Mapping of Software Components to ECUs	1	

Table 3.114: Deploy Software Component

3.3.1.6 Generate or Adjust System Flat Map

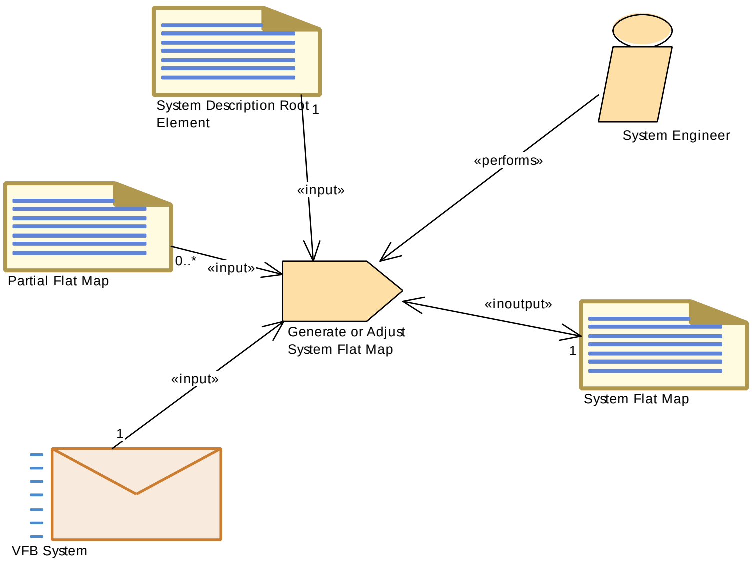

Figure 3.53: Generate or Adjust System Flat Map

Task Definition	**Generate or Adjust System Flat Map**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description	Generates and/or adjust the unique names of component prototypes and MCD display data in the scope of system.		
Description	Generates and/or adjust the unique names of component prototypes and MCD display data in the scope of a System or System Extract.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	System Description Root Element	1	
Consumes	VFB System	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Partial Flat Map	0..*	If Partial Flat Maps were delivered along with software components, they must be integrated into the System Flat Map: • The instance refs used in a partial flat map must be taken over and adjusted to the context of the System or System Extract. • Name conflicts have to be resolved if several partial flat maps are merged.
In/out	System Flat Map	1	

Table 3.115: Generate or Adjust System Flat Map

3.3.1.7 Derive Communication Needs

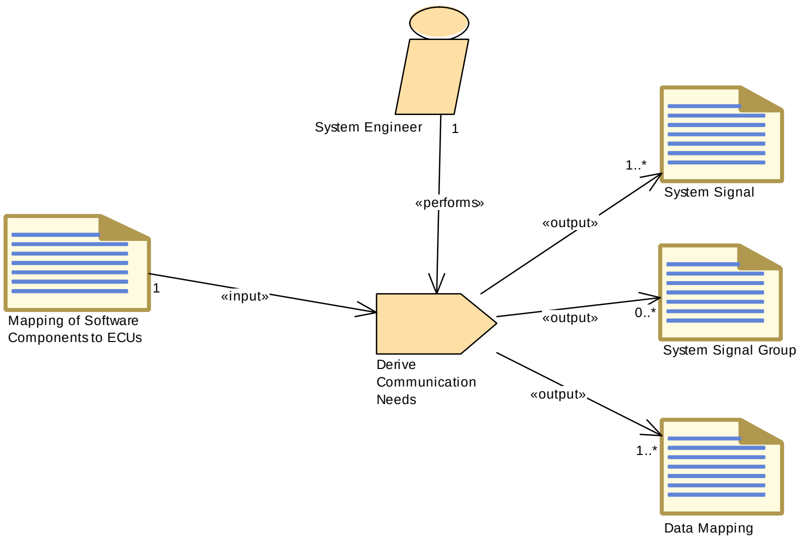

Figure 3.54: Derive Communication Needs

Task Definition	**Derive Communication Needs**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description	Define the signals used to exchange data & operations needed by software components over a network.		
Description	Define the signals used to exchange data & operations needed by software components over a network.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Mapping of Software Components to ECUs	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Produces	Data Mapping	1..*	
Produces	System Signal	1..*	
Produces	System Signal Group	0..*	

Table 3.116: Derive Communication Needs

3.3.1.8 Define Signal Path Constraints

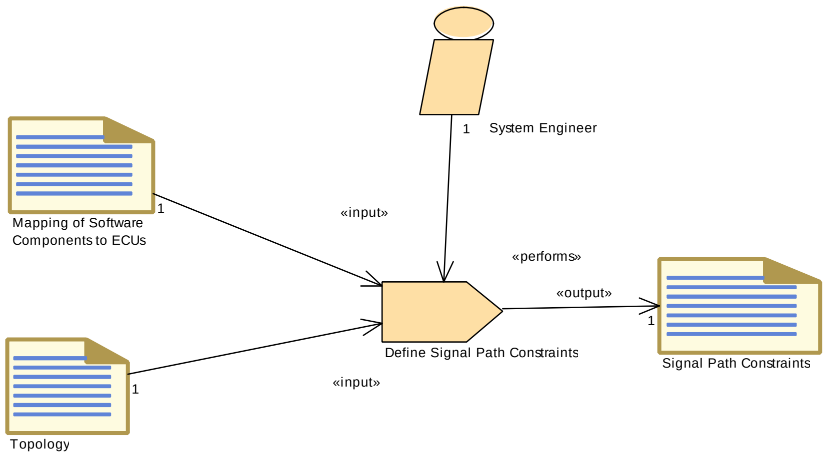

Figure 3.55: Define Signal Path Constraints

Task Definition	**Define Signal Path Constraints**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description	Additional guidelines for the System Generator, which specific way a signal between two Software Components should take in the network without defining in which frame and with which timing it is transmitted.		
Description	Define additional guidelines for the System Generator, which specific way a signal between two Software Components should take in the network without defining in which frame and with which timing it is transmitted.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	System Engineer	1	
Consumes	Mapping of Software Components to ECUs	1	
Consumes	Topology	1	
Produces	Signal Path Constraints	1	

Table 3.117: Define Signal Path Constraints

3.3.1.9 Define System Variants

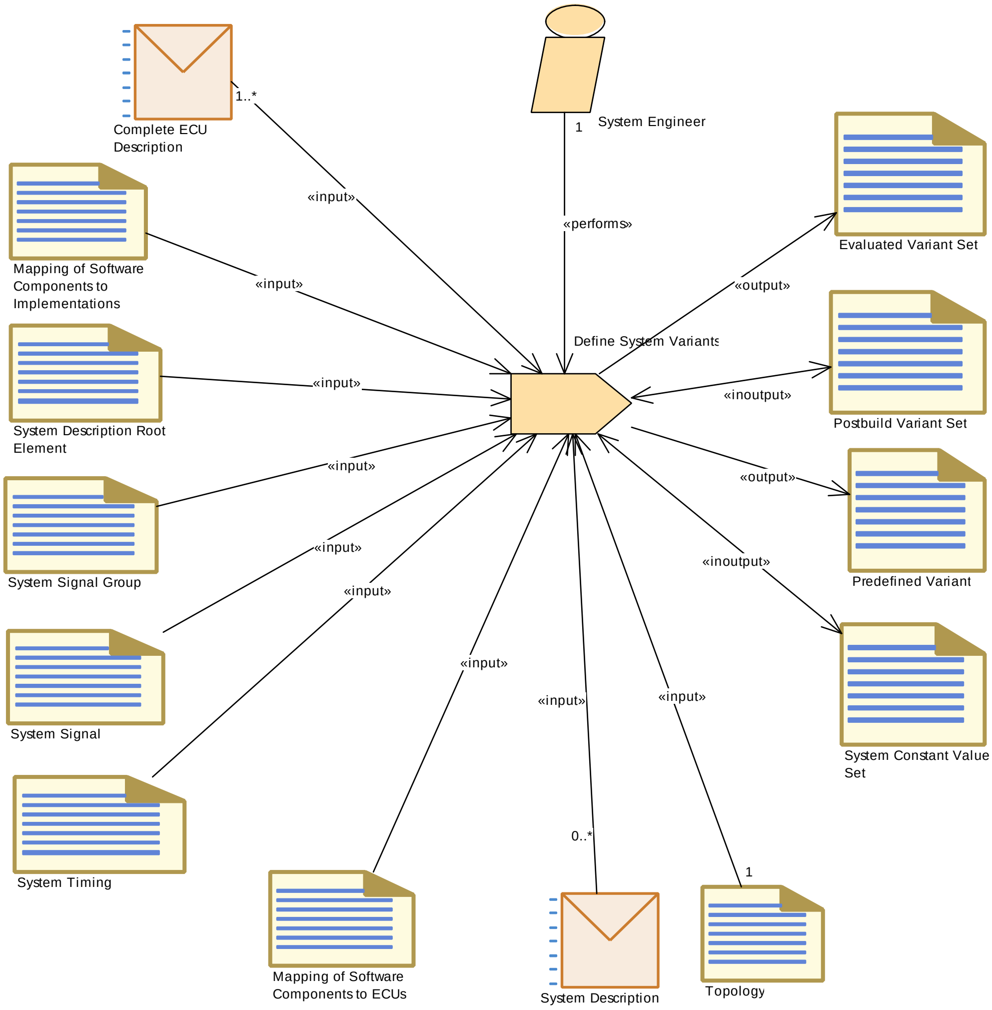

Figure 3.56: Define System Variants

Task Definition	**Define System Variants**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description	Define variants for the artifacts of a System Description.		
Description	Define variants for the artifacts of a System Description. Definition of a variant means in general to define its conditions and its latest binding time. Therefore one has to create a PredefinedVariant referring to the settings which are used by the system elements in scope. To do so, this task can make use of existing System Constant Value Set s and/or Postbuid Variant Set s or define new ones. Several PredefinedVariant s can be combined to one Evaluated Variant Set . This task can also be applied when designing a subsystem, therefore the System Extract is an optional input.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Mapping of Software Components to ECUs	1	
Consumes	Mapping of Software Components to Implementations	1	
Consumes	System Description Root Element	1	
Consumes	System Signal	1	
Consumes	System Signal Group	1	
Consumes	System Timing	1	
Consumes	Topology	1	
Consumes	Complete ECU Description	1..*	
Consumes	System Description	0..*	
In/out	Postbuild Variant Set	1	
In/out	System Constant Value Set	1	
Produces	Evaluated Variant Set	1	
Produces	Predefined Variant	1	

Table 3.118: Define System Variants

3.3.1.10 Define System Timing

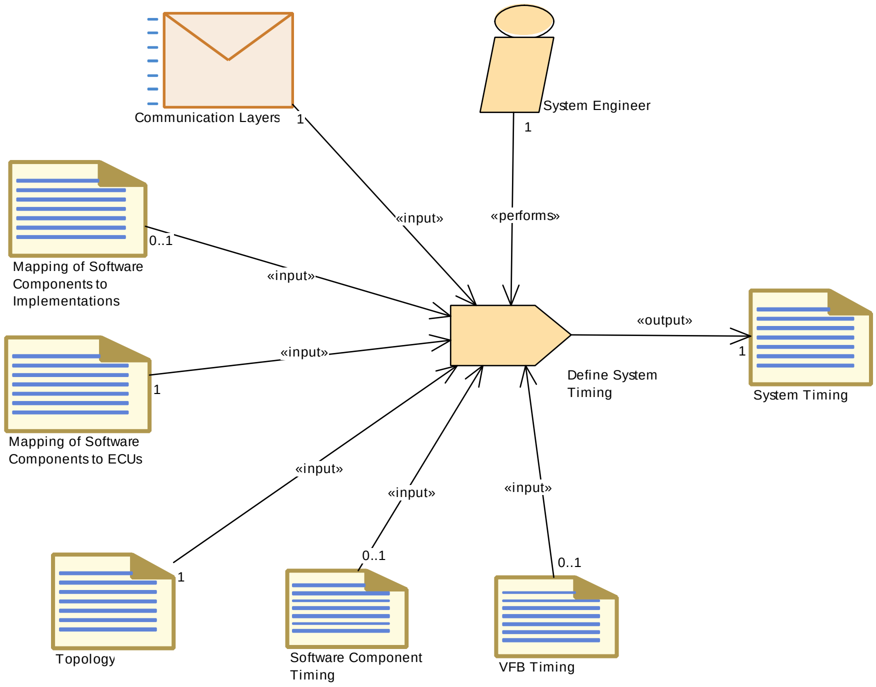

Figure 3.57: Define System Timing

Task Definition	**Define System Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description	Define SystemTiming for a concrete system taking the mapping of software components to ECUs and their implementation into account		
Description	Define SystemTiming (TimingDescription and TimingConstraints) for a concrete system taking the mapping of software components to ECUs and their implementation into account. This means that the resulting Communication Matrix (and its implication to the communication stack) can also be referenced by the timing specification to refine remote communication timing behavior.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Communication Layers	1	
Consumes	Mapping of Software Components to ECUs	1	
Consumes	Topology	1	
Consumes	Mapping of Software Components to Implementations	0..1	
Consumes	Software Component Timing	0..1	
Consumes	VFB Timing	0..1	
Produces	System Timing	1	

Relation Type	***Related Element***	***Mul.***	***Note***

Table 3.119: Define System Timing

3.3.1.11 Extend Topology

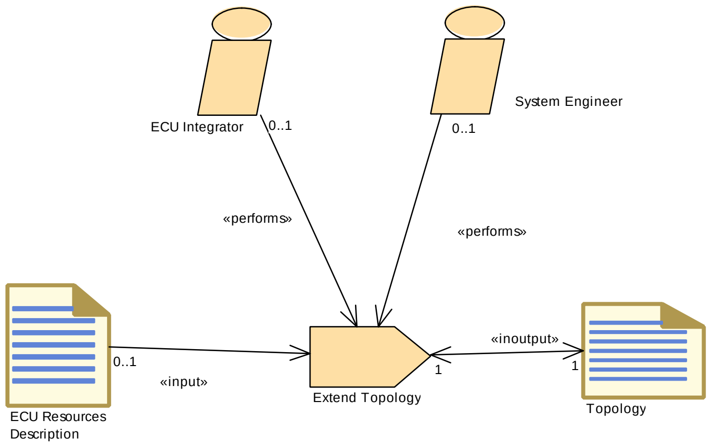

Figure 3.58: Extend Topology

Task Definition	**Extend Topology**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description	Extend the existing System Topology		
Description	Extend the existing System Topology by describing how new ECUs will be connected to the existing one through the current network		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	0..1	
Performed by	System Engineer	0..1	
Consumes	ECU Resources Description	0..1	
In/out	Topology	1	

Table 3.120: Extend Topology

3.3.1.12 Select Software Component Implementation

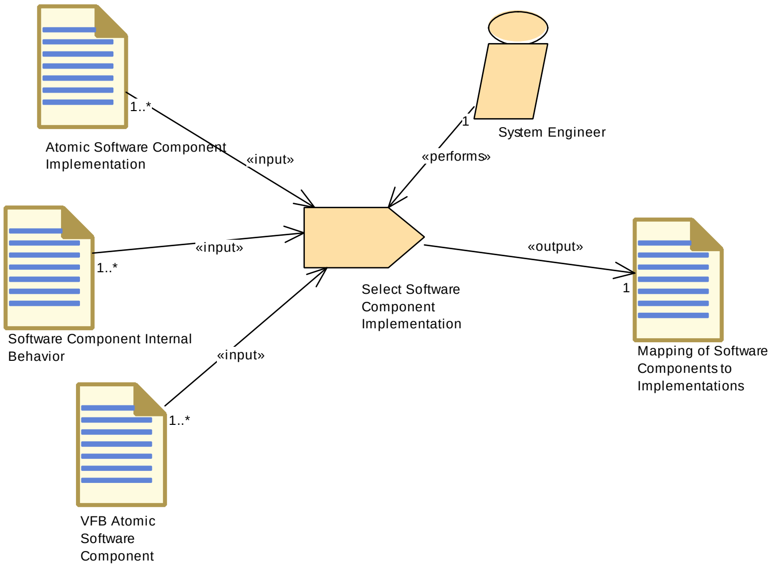

Figure 3.59: Select Software Component Implementation

Task Definition	**Select Software Component Implementation**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description	Select implementation for an Atomic Software Component.		
Description	The system engineer selects an Atomic Software Component Implementation for each defined VFB Atomic Software Component		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Atomic Software Component Implementation	1..*	
Consumes	Software Component Internal Behavior	1..*	
Consumes	VFB Atomic Software Component	1..*	
Produces	Mapping of Software Components to Implementations	1	

Table 3.121: Select Software Component Implementation

3.3.1.13 Select Design Time Variant

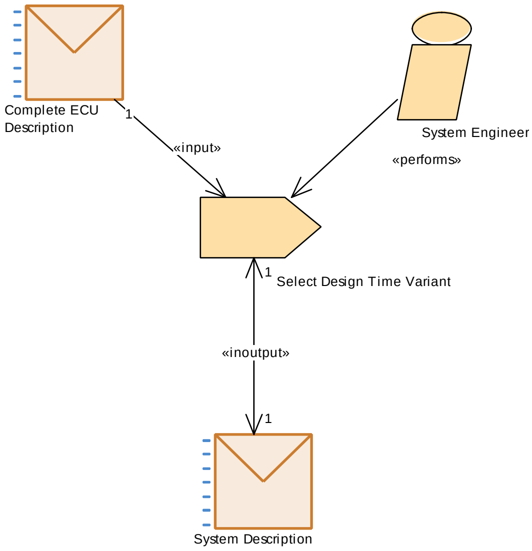

Figure 3.60: Select Design Time Variant

Task Definition	**Select Design Time Variant**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description	Select a system variant at system design time.		
Description	Select a system variant at system design time. This could be done in different ways: Replace a model, which contains the variation points contributing to this particular variant and all the possible settings/elements, by a model, which does no more contain these variation points and which contains only the particular settings/elements selected for this variant. In order to document the selection for further process steps, it is also possible to keep the information about the selected variant and the variation points in the model by introducing a PredefinedVariant along with appropriate fixed settings of system constant values. In constrast to variant selection in later process steps, no code generation or compilation is involved at system design time, thus this task is just a transformation of one XML model into another one. This task can be applied to a complete system description, represented by a System Extract Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Complete ECU Description	1	
In/out	System Description	1	

Table 3.122: Select Design Time Variant

3.3.1.14 Define System View Mapping

The task Define System View Mapping (see Figure 3.61) creates the System View Mapping between two System Descriptions. Different cases can be separated:

- Mapping of different overall VFB systems - the Abstract System Description and the System Configuration Description.
- Mapping of different structured System Extracts, e.g. System Extract delivered by a primary organization and the different structure (ECU System Description) of the secondary organization (see 2.5.4, 2.5.5).

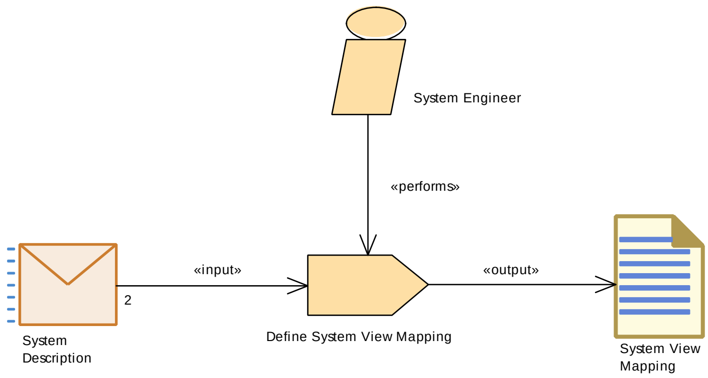

Figure 3.61: Define System View Mapping

Task Definition	**Define System View Mapping**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description	Map elements from different views on the system.		
Description	This task creates the System View Mapping between two System Descriptions (Mapping of different structured system descriptions, e.g. system extract delivered by a primary organization and the different structure of the secondary organisation).		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	System Description	2	
Produces	System View Mapping	1	

Table 3.123: Define System View Mapping

3.3.1.15 Create Transformer Specification

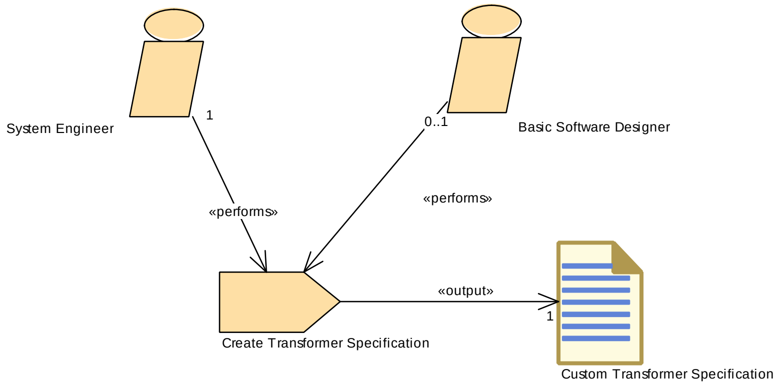

Figure 3.62: Create Transformer Specification

Task Definition	**Create Transformer Specification**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description			
Description	In this task the specification of a transformer module is created. Since the specification is created as a part of the communication design, the System Engineer has to perform this task. Optionally a Basic Software Designer can support the creation of the specification.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Performed by	Basic Software Designer	0..1	
Produces	Custom Transformer Specification	1	

Table 3.124: Create Transformer Specification

3.3.1.16 Define Rapid Prototyping Scenario

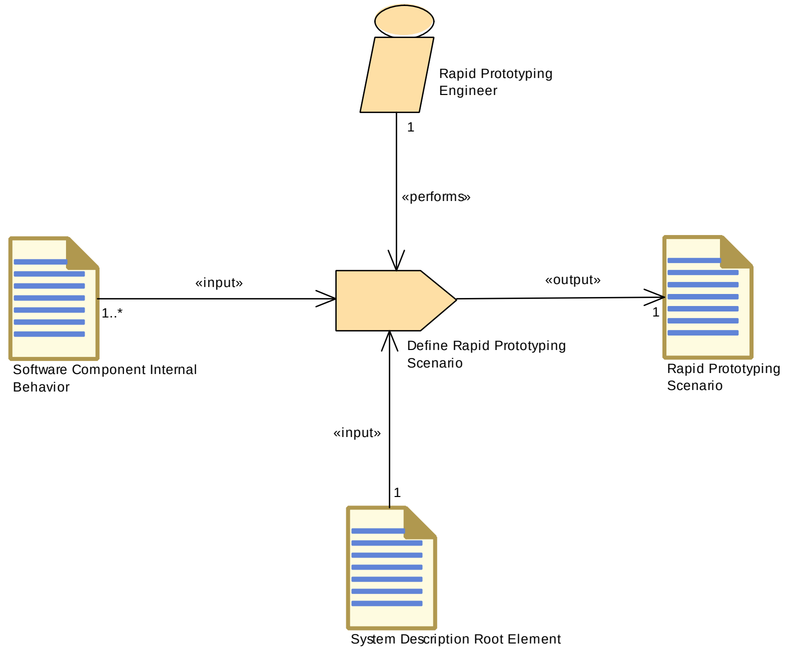

Figure 3.63: Define Rapid Prototyping Scenario

Task Definition	**Define Rapid Prototyping Scenario**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
Brief Description			
Description	Defines the rapid prototyping scenario.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Rapid Prototyping Engineer	1	
Consumes	System Description Root Element	1	
Consumes	Software Component Internal Behavior	1..*	
Produces	Rapid Prototyping Scenario	1	

Table 3.125: Define Rapid Prototyping Scenario

3.3.2 Work Products

3.3.2.1 System Description

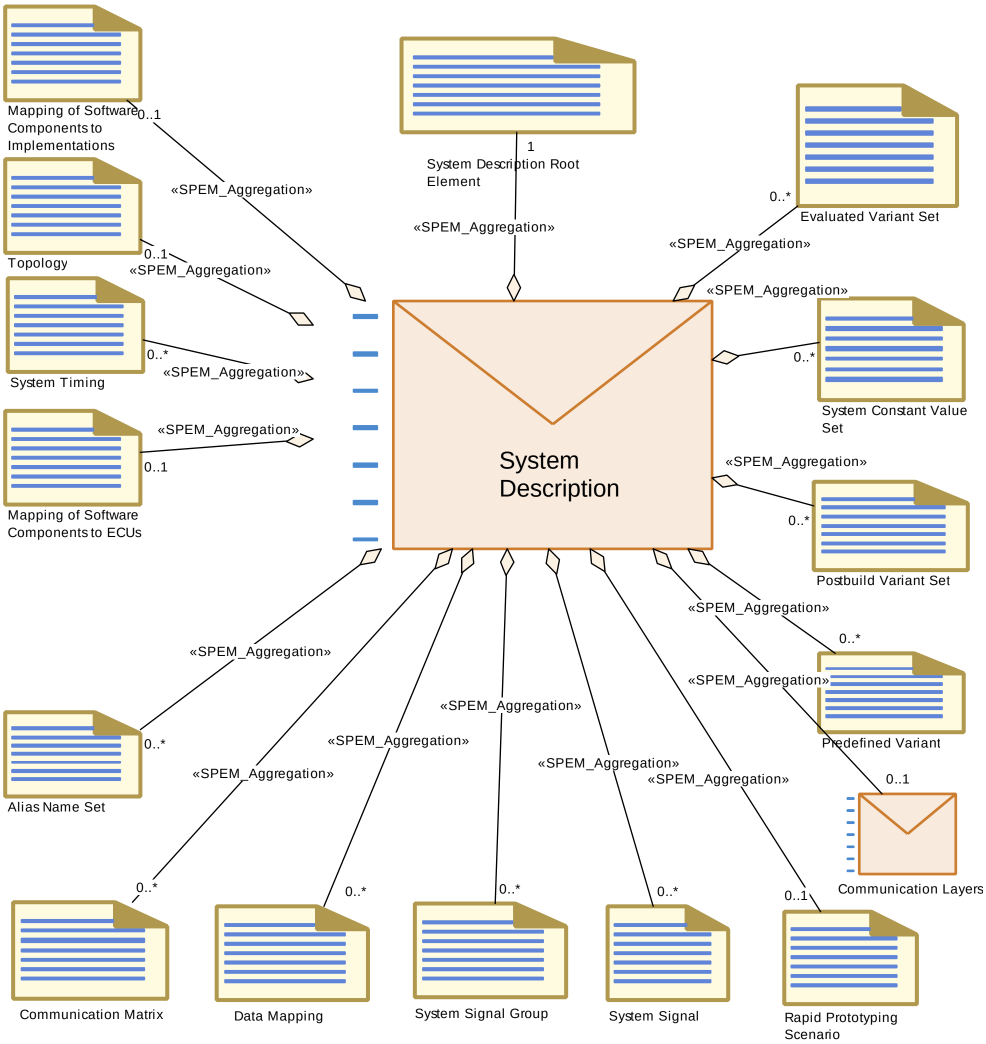

Figure 3.64: Structure of generic deliverable System Description

Deliverable	**System Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description	Partial Extract of a System		
Description	Generic deliverable for defining a System. It is used in different roles within the methodology. In each role, this deliverable may contain variation points in its ARXML artifacts which need to be bound in later steps, e.g. when defining a subsystem from a complete system or later for the single ECUs. If such variation points are present, the System Description may optionally include PredefinedVariants in order to predefine variants for later selection and an Evaluated Variant Set. Please note that this generic deliverable does not correspond to the system description with the system category "SYSTEM_DESCRIPTION" (see [TPS_SYST_01003]). The system description with the category "SYSTEM_DESCRIPTION" is represented by the deliverable "System Configuration Description". This deliverable is equivalent to a description of a system with any category. In the System Template Specification "system description" is the most frequently used term for this kind of artifact.		
Kind	Delivered		
Extended by	Abstract System Description, System Configuration Description, System Constraint Description, System Extract		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregates	System Description Root Element	1	
Aggregates	Communication Layers	0..1	
Aggregates	Mapping of Software Components to ECUs	0..1	
Aggregates	Mapping of Software Components to Implementations	0..1	
Aggregates	Rapid Prototyping Scenario	0..1	
Aggregates	Topology	0..1	
Aggregates	Alias Name Set	0..*	
Aggregates	Communication Matrix	0..*	
Aggregates	Data Mapping	0..*	
Aggregates	Evaluated Variant Set	0..*	
Aggregates	Postbuild Variant Set	0..*	
Aggregates	Predefined Variant	0..*	
Aggregates	System Constant Value Set	0..*	

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	System Signal	0..*	
Aggregates	System Signal Group	0..*	
Aggregates	System Timing	0..*	
In/out	Select Design Time Variant	1	
Consumed by	Define System View Mapping	2	
Consumed by	Define System Safety Information	1	
Consumed by	Design signal oriented communication between Classic and Adaptive Platform	1	The System Description based on the System Template on the AUTOSAR classic platform is used; it contains a communication matrix description with Pdus and ISignals
Consumed by	Define Alias Names	0..1	Needed for definition of alias names with system, system extract or ECU scope, depending of the role of the System Description.
Consumed by	Define System Variants	0..*	

Table 3.126: System Description

Deliverable	**System Constraint Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description			
Description	Contains the artifacts that describe System Constraints. It serves as an input for setting up the complete Abstract System Description and/or System Configuration Description. This deliverable corresponds to the system description with the system category "SYSTEM_CONSTRAINTS" (see [TPS_SYST_01003]).		
Kind	Delivered		
Extends	System Description		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Overall VFB System	0..1	
Aggregates	System Flat Map	0..1	
Consumed by	Develop System	0..1	
Consumed by	Develop an Abstract System Description	0..1	In the context of the "Develop an Abstract System Description" activity, the constraints for the abstract or functional view on the system can be provided by the "System Constraint Description".

Table 3.127: System Constraint Description

Deliverable	**System Configuration Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description			
Description	Contains the artifacts that describe a complete AUTOSAR System. It is the basis for extracting descriptions for sub-systems or ECUs. Note that System Extracts may be refined by details which are not present in the System Configuration. This deliverable corresponds to the system description with the system category "SYSTEM_DESCRIPTION" (see [TPS_SYST_01003]).		
Kind	Delivered		
Extends	System Description		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregates	Overall VFB System	1	
Aggregates	System Flat Map	0..1	
Produced by	Develop System	1..*	
Consumed by	Generate System Extract	1	
Consumed by	Generate ECU Extract	0..1	

Table 3.128: System Configuration Description

Deliverable	**System Extract**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description			
Description	Contains the artifacts that describe a subsystem specific view on the complete System Description. Initially, the System Extract is not fully decomposed and still contains compositions. It is the basis for designing subsystems, e.g. by adding further ECUs within the given constraints. This deliverable corresponds to the system description with the system category "SYSTEM_EXTRACT" (see [TPS_SYST_01003]).		
Kind	Delivered		
Extended by	ECU System Description		
Extends	System Description		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregates	VFB System Extract	1	
Aggregates	System Flat Map	0..1	
Produced by	Develop System	0..*	
Produced by	Generate System Extract	0..*	
Consumed by	Create ECU System Description	1	

Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Develop Sub-System	1	
Consumed by	Generate ECU Extract	0..1	

Table 3.129: System Extract

Deliverable	**ECU System Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description			
Description	This System Description is used to describe the closed view on one ECU (note that an AUTOSAR ECU is defined being one microprocessor running one AUTOSAR Stack). It can be derived from a System Extract or it can be designed independently and mapped to a System Extract. The ECU System Description is not fully decomposed and still may contain compositions. It is refined during the activity Design Sub-System. This deliverable corresponds to the system description with the system category "ECU_SYSTEM_DESCRIPTION" (see [TPS_SYST_01003]).		
Kind			
Extends	System Extract		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Design Sub-System	1	System Extract refined during design of the corresponding sub-system with elements needed to generate ECU Extract(s).
Produced by	Create ECU System Description	1..*	
Consumed by	Design Sub-System	1	System Extract as generated from the outer system.
Consumed by	Configure Mode Management	0..1	Input in case ECU Extract is not available (atomic software components not available)
Consumed by	Generate ECU Extract	0..1	

Table 3.130: ECU System Description

3.3.2.2 Abstract System Description

Deliverable	**Abstract System Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description	Provides an abstract or functional view on the system		
Description	The Abstract System Description extends the general System Description and provides an abstract or functional view on the system to be developed. This deliverable corresponds to the system description with the system category "ABSTRACT_SYSTEM_DESCRIPTION" (see [TPS_SYST_01003]).		
Kind	Delivered		
Extends	System Description		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Overall VFB System	1	
Produced by	Develop an Abstract System Description	1..*	
Consumed by	Develop System	0..*	The abstract System Description is an optional input for the activity "Develop System". Please note, that in this step the Abstract System Description is refined to a System Description.
Consumed by	Develop a VFB System Description	0..*	The abstract System Description is an optional input for the activity "Develop a VFB System Description". The VFB-related part of the Abstract System Description can be than refined to the concrete "Overall VFB System". Additionally, a mapping between those two views can be established.

Table 3.131: Abstract System Description

3.3.2.3 Complete ECU Description

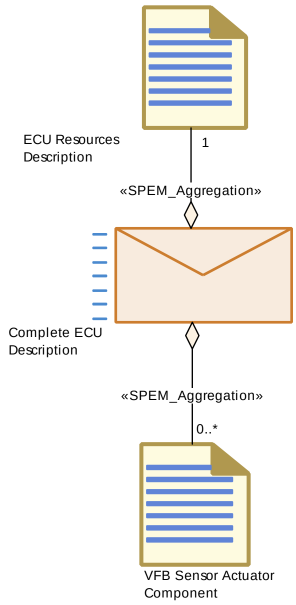

Figure 3.65: Complete ECU Description

Deliverable	**Complete ECU Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description	An ECU Description includes the resources it has available along with its corresponding ECU-specific software components.		
Description	An ECU Description includes the resources it has available along with its corresponding ECU-specific software components.		
Kind	Delivered		
Relation Type	**_Related Element_**	**_Mul._**	**_Note_**
Aggregates	ECU Resources Description	1	
Aggregates	VFB Sensor Actuator Component	0..*	
Consumed by	Select Design Time Variant	1	
Consumed by	Define System Variants	1..*	

Table 3.132: Complete ECU Description

3.3.2.4 System Description Root Element

Artifact	**System Description Root Element**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description	A System Description root element.		
Description	The System description defines the following major elements: • Topology : description of the Topology of the System. • Software : description of the root software composition containing all software components in the System in a hierarchical structure. • Communication : description of all Communication elements used in the System. • Mapping and Mapping Constraints : description of all mapping aspects (mapping of SW components to ECUs, mapping of data elements to signals, and mapping constraints). The root element can be the basis for a System extract as well as for the whole System depending on which elements are aggregated.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	System Description	1	
Produced by	Assign Top Level Composition	1	
Produced by	Set System Root	1	Set up the root element, and the links to other artifacts
Consumed by	Define Rapid Prototyping Scenario	1	
Consumed by	Define System Variants	1	
Consumed by	Flatten Software Composition	1	find the top level composition
Consumed by	Generate or Adjust System Flat Map	1	
Use meta model element	System	1	

Table 3.133: System Description Root Element

3.3.2.5 System Mapping Overview

There are various artifacts which correspond to the mappings collected under the meta-model element `SystemMapping`. Figure 3.66 shows an overview. The details will be explained in the following sub-chapters.

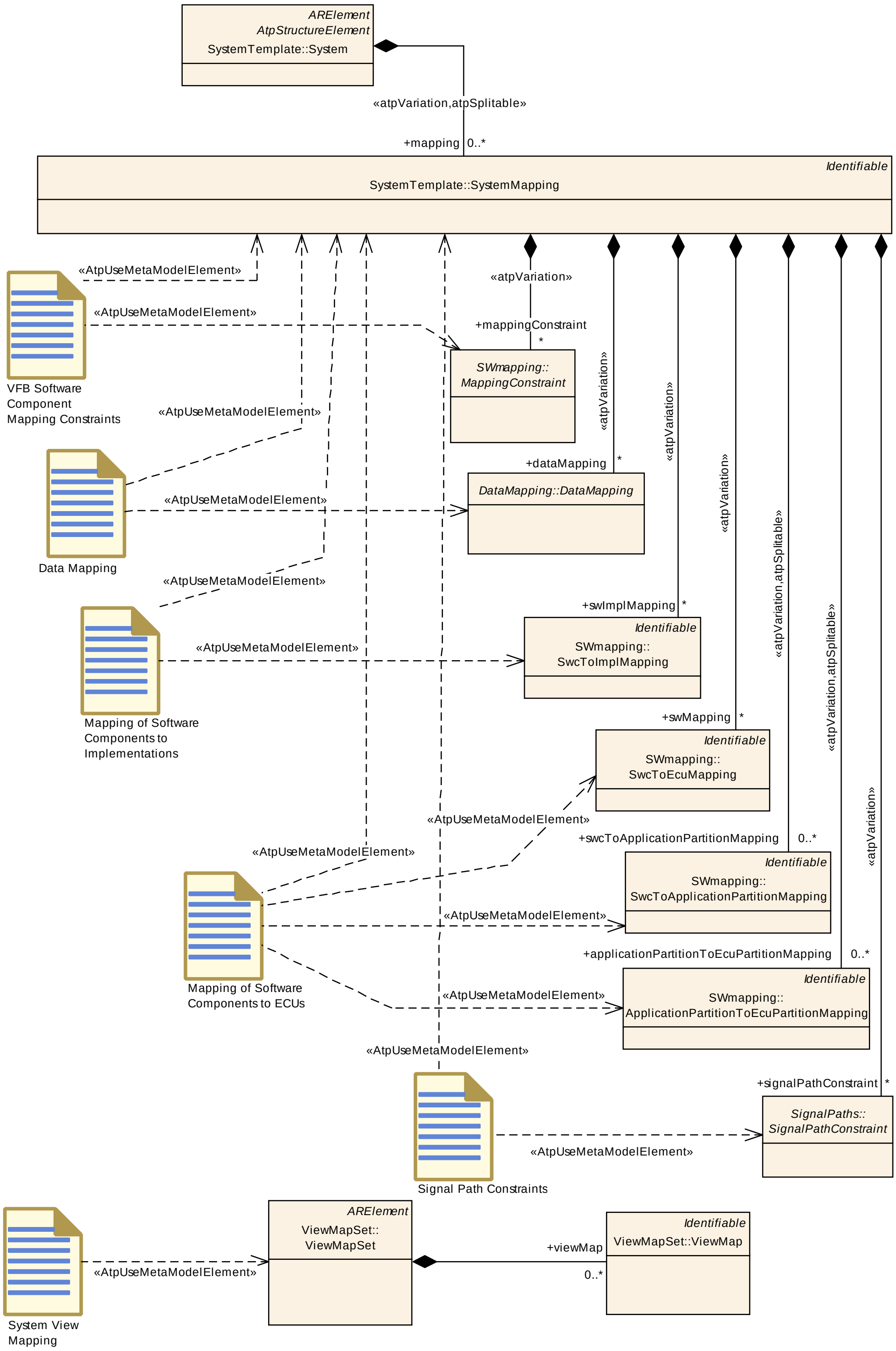

Figure 3.66: Overview on the various artifacts for System Mapping

3.3.2.6 Data Mapping

Artifact	**Data Mapping**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description			
Description	Mapping of data prototypes from the VFB description to System signals.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	System Description	0..*	
Produced by	Derive Communication Needs	1..*	
Consumed by	Define Signal PDUs	1	
Consumed by	Flatten Software Composition	1..*	
Consumed by	Set System Root	1..*	Only the reference to the artifact is needed
Use meta model element	DataMapping	1	
Use meta model element	SystemMapping	1	The splitable element SystemMapping is the root for this artifact.

Table 3.134: Data Mapping

3.3.2.7 Mapping of Software Components to ECUs

Artifact	**Mapping of Software Components to ECUs**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description	Describes the mapping of Software Components to the ECUs that are defined in the VFB context.		
Description	The VFB shows all software components independently of their deployment on individual ECUs. This work product defines for each software component the corresponding ECU on which the software component will be deployed and executed. This artifact may contain a mapping of software components to application partitions by a SwcToApplicationPartitionMapping. With an ApplicationPartitionToEcuPartitionMapping the application partitions are assigned to ECU partitions. This can substitute the direct mapping of software components to ECUs via SwcToEcuMapping.ecuInstance.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	System Description	0..1	
Produced by	Deploy Software Component	1	
Consumed by	Define Signal PDUs	1	

Relation Type	Related Element	Mul.	Note
Consumed by	Define Signal Path Constraints	1	
Consumed by	Define System Timing	1	
Consumed by	Define System Variants	1	
Consumed by	Derive Communication Needs	1	
Consumed by	Extract the ECU Communication	1	
Consumed by	Flatten Software Composition	1	
Consumed by	Set System Root	1	Only the reference to the artifact is needed
Use meta model element	ApplicationPartitionToEcuPartition Mapping	1	
Use meta model element	SwcToApplication PartitionMapping	1	
Use meta model element	SwcToEcuMapping	1	
Use meta model element	SystemMapping	1	The splitable element SystemMapping is the root for this artifact.

Table 3.135: Mapping of Software Components to ECUs

3.3.2.8 Mapping of Software Components to Implementations

Artifact	**Mapping of Software Components to Implementations**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description			
Description	Specifies the selection of software implementations for the atomic component prototypes. Because component prototypes can be located on different ECUs, it is possible to have different Implementations of two prototypes of the same AtomicComponentType in the system.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	System Description	0..1	
Produced by	Select Software Component Implementation	1	
Consumed by	Define System Variants	1	
Consumed by	Define System Timing	0..1	
Use meta model element	SwcToImplMapping	1	

Relation Type	Related Element	Mul.	Note
Use meta model element	SystemMapping	1	The splitable element SystemMapping is the root for this artifact..

Table 3.136: Mapping of Software Components to Implementations

3.3.2.9 Signal Path Constraints

Artifact	**Signal Path Constraints**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description	Constraints on the Path that should be used or not by Signals		
Description	One of the tasks of the System Generator is actually to calculate automatically the communication (signals) between the RTEs and define the needed frames for that communication. These definitions of the frames include implicitly the definition of the paths the AUTOSAR-Signals are transmitted through the system. Thereby the System Generator often has the choice between alternative ways through the system. There exist four different constraints for signals regarding the signal path: • The CommonSignalPath describes that two signals must take the same way (Signal Path) in the topology. • 'The ForbiddenSignalPath describes the way (Signal Path) that a signal must not take in the topology, e.g. in case of safety critical transmission. • The PermissibleSignalPath describes the way (Signal Path) a signal can take in the topology. If more than one PermissibleSignalPath is defined for the same signal/operation attributes, any of them can be chosen. • The SeparateSignalPath describes that two or more signals must not take the same way (Signal Path) in the topology e.g. in case of redundant transmission. It is also possible that the same signal is aggregated two times by the SeparateSignalPath element to indicate that this signal should be transmitted redundantly over two different paths.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Produced by	Define Signal Path Constraints	1	
Consumed by	Set System Root	1	Only the reference to the artifact is needed
Use meta model element	SignalPathConstraint	1	
Use meta model element	SystemMapping	1	The splitable element SystemMapping is the root for this artifact.

Table 3.137: Signal Path Constraints

3.3.2.10 Topology

Artifact	**Topology**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description	The system topology, which may be reused in different systems.		
Description	Describes the topology of the system : A topology is formed by a number of EcuInstances that are interconnected to each other in order to form ensembles of ECUs and CommunicationClusters.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	System Description	0..1	
Produced by	Define System Topology	1	
In/out	Extend Topology	1	
Consumed by	Define Communication Matrix	1	
Consumed by	Define Network Management	1	
Consumed by	Define Signal PDUs	1	
Consumed by	Define Signal Path Constraints	1	
Consumed by	Define System Timing	1	
Consumed by	Define System Variants	1	
Consumed by	Define TP	1	
Consumed by	Deploy Software Component	1	
Consumed by	Extract ECU Topology	1	
Consumed by	Set System Root	1	Only the reference to the artifact is needed
Consumed by	Define Secured PDUs	0..1	
Use meta model element	Communication Cluster	1	
Use meta model element	EcuInstance	1	

Table 3.138: Topology

3.3.2.11 Ecu Resources Description

Artifact	**ECU Resources Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description	Definition of the resources available on an ECU.		
Description	Definition of the resources available on an ECU. It mainly contains a description of hardware elements (like physical memory sections or peripherals, pins, hardware connections) which need to be referred by a software component or a basic software description. The focus is to describe an already engineered piece of hardware, its content and structure. It is not in the focus of the ECU Resource Description to support the design of electronics hardware itself. In the XML it is represented as a set of HwDescriptionEntity -s		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Complete ECU Description	1	
Produced by	Define ECU Description	1..*	
Consumed by	Define and configure machine	1	All resources which are available for the ECU
Consumed by	Describe Available HW Resources	1	Definition of available HW resources for the machine based on the description of the ECU
Consumed by	Define System Topology	1..*	
Consumed by	Define BSW Interfaces	0..1	
Consumed by	Define ECU Abstraction Component	0..1	
Consumed by	Extend Topology	0..1	
Consumed by	Generate ECU Executable	0..1	may be used to set up build environment Meth.bindingTime = CompileTime
Consumed by	Implement a BSW Module	0..1	Meth.bindingTime = SystemDesignTime
Consumed by	Measure Component Resources	0..1	
Consumed by	Measure Resources	0..1	
Consumed by	Define Complex Driver Component	0..*	
Consumed by	Define VFB Sensor or Actuator Component	0..*	
Use meta model element	HwElement	1	

Table 3.139: ECU Resources Description

3.3.2.12 System Signal

Artifact	**System Signal**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description			
Description	The system signals allow to represent this communication view in a flattened structure, with (at least) one system signal defined for each data element sent or received by a SW component instance. If data has to be sent over gateways, there is still only one system signal representing this data. The representation of the data on the individual communication systems is done by the cluster signals.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	System Description	0..*	
Produced by	Derive Communication Needs	1..*	
Consumed by	Define Signal PDUs	1	
Consumed by	Define System Variants	1	
Consumed by	Define RTE Fan-out	1..*	
Consumed by	Extract the ECU Communication	0..*	
Use meta model element	SystemSignal	1	

Table 3.140: System Signal

3.3.2.13 System Signal Group

Artifact	**System Signal Group**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System:: Work products		
Brief Description	A signal group refers to a set of signals that must always be kept together. A signal group is used to guarantee the atomic transfer of AUTOSAR composite data types.		
Description	The System Signal Group is representing a set of Signals that must be kept together. A signal group is to guarantee the transfer of AUTOSAR composite data types for sender receiver communication.The RTE is required to treat AUTOSAR signals transmitted using sender-receiver communication atomically. To achieve this, the "signal group" mechanisms shall be utilized.It is not possible to map a Variable Data Prototype with a composite datatype directly to a System Signal . The complex data type must be decomposed into single signals. As this set of single signals has to be treated as atomic, it is placed in a "signal group". It is also used in client server communication when the RTE maps a response to a corresponding operation request. The arguments, application errors, client identifier and sequence counter of an operation are mapped to System Signal of two dedicated SystemSignalGroup elements;one for the request and one for the response. The RTE Client Server Protocol is used to provide a specific semantics to each of these SystemSignalGroups and System Signal , also those which are introduced only to support the protocol.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	System Description	0..*	
Produced by	Derive Communication Needs	0..*	
Consumed by	Define System Variants	1	
Consumed by	Extract the ECU Communication	0..*	
Use meta model element	SystemSignal Group	1	

Table 3.141: System Signal Group

3.3.2.14 System Flat Map

Artifact	**System Flat Map**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description	Mapping of instance names to nested model elements. Use cases: Resolve name conflicts when flattening VFB software compositions; provide unique names and unique model references for measurement and calibration data.		
Description	The flat map is a list of elements, each element represents exactly one node (e.g. a component instance or data element) of the instance tree of a software system. The purpose of this element is to map the various nested representations of this instance to a flat representation and assign a unique name to it. The name will be unique in the scope to which this Flat Map belongs (which could be a whole System or a System Extract). Use case: The System Flat Map is defined in the context of a System or System Extract. It serves as a basis for generating an ECU Flat Map (or a Flat Map of a "child" System Extract). In the ECU Flat Map, the names will be used as display names for MCD tools or as names for component prototypes in a flattened software composition. For further information refer to the description of artifact ECU Flat Map.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	System Configuration Description	0..1	
Aggregated by	System Constraint Description	0..1	
Aggregated by	System Extract	0..1	
In/out	Generate or Adjust System Flat Map	1	
Consumed by	Add Documentation to the Software Component	0..1	Optional input in order to refer to unique names defined in system context.
Consumed by	Generate or Adjust ECU Flat Map	0..1	Take over definitions of unique names from system level to ECU level.
Use meta model element	FlatMap	1	

Table 3.142: System Flat Map

3.3.2.15 System Timing

Artifact	**System Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description	Concrete system's TimingDescription and TimingConstraints		
Description	TimingDescription and TimingConstraints defined for a concrete system taking the mapping of software components to ECUs and their implementation into account. This means that the resulting Communication Matrix (and its implication to the communication stack) can also be referenced by the timing specification to refine remote communication timing behavior.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	System Description	0..*	
Produced by	Define System Timing	1	
Consumed by	Define System Variants	1	
Consumed by	Extract ECU System Timing	1	
Consumed by	Deploy Software Component	0..1	
Use meta model element	SystemTiming	1	

Table 3.143: System Timing

3.3.2.16 System View Mapping

Artifact	**System View Mapping**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description	The System View Mapping provide an mapping between different views on the system.		
Description	This artifact contains a set of system view mappings and provides an mapping between different views on the system, e.g. different overall VFB systems (e.g. abstract system description with system configuration description), or the overall VFB system with the VFB System Extract description.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Overall VFB System	0..1	The Overall VFB System aggregates a potential mapping to the abstract or functional view of the system.
Aggregated by	VFB System Extract	0..1	The VFB System Extract aggregates a potential mapping to the abstract or functional view of the system.
Produced by	Define System View Mapping	1	
Use meta model element	ViewMapSet	1	

Table 3.144: System View Mapping

3.3.2.17 Transformer Design Bundle

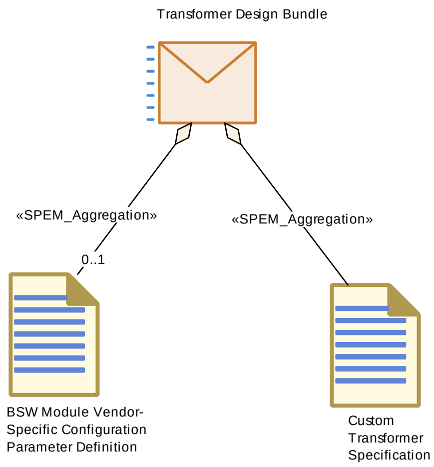

Figure 3.67: Structure of deliverable Transformer Design Bundle

Deliverable	**Transformer Design Bundle**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
Brief Description			
Description	This deliverable contains a specification of the transformer technology to be implemented by the BSWM developer. Furthermore it contains the Vendor specific parameter definition for the corresponding transformer.		
Kind	Delivered		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Custom Transformer Specification	1	
Aggregates	BSW Module Vendor-Specific Configuration Parameter Definition	0..1	
Produced by	Design Custom Transformer	1	
Produced by	Develop System	0..*	
Consumed by	Develop Basic Software	0..*	

Table 3.145: Transformer Design Bundle

3.3.2.18 Custom Transformer Specification

Artifact	**Custom Transformer Specification**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System:: Work products		
Brief Description			
Description	This artifact represents the functional specification of the Transformer to be implemented. The AUTOSAR methodology does not prescribe the format of this artifact.		
Kind	Custom		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	Transformer Design Bundle	1	
Produced by	Create Transformer Specification	1	

Table 3.146: Custom Transformer Specification

3.3.2.19 Rapid Prototyping Scenario

Artifact	**Rapid Prototyping Scenario**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System:: Work products		
Brief Description	Description of the (required) bypass points and the hooks in the system.		
Description	Description of the (required) bypass points and the in the system and the corresponding hooks. This artifact contains the RptContainers with bypass points referencing things like parameterAccess (dataWriteAccess, dataReadAccess, dataSendPoint, dataReceivePointByValue, dataReceivePointByArgument, writtenLocalVariable, readLocalVariable, etc.) The hooks describe the link between the bypass points and the rapid prototyping algorithm.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	System Description	0..1	
Produced by	Define Rapid Prototyping Scenario	1	
Consumed by	Extract ECU Rapid Prototyping Scenario	1	
Use meta model element	RapidPrototyping Scenario	1	

Table 3.147: Rapid Prototyping Scenario

3.3.3 Communication Matrix and Communication Layers

This section contains the tasks and work products to set up the communication matrix and the communication layers as part of a system description.

3.3.3.1 Tasks

3.3.3.1.1 Define Communication Matrix

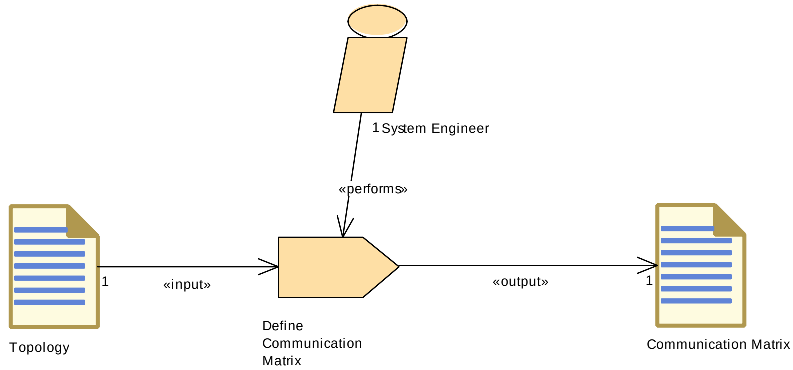

Figure 3.68: Define Communication Matrix

Task Definition	**Define Communication Matrix**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Tasks		
Brief Description	The communication matrix contents are created or extended by adding communication definitions.		
Description	Define or extend Communication Matrix. Define the triggering of the Physical Channels and the mapping to the communication connector ports. In case of extension the original communication matrix contents (which were delivered as part of a system extract) are extended by adding communication definitions. The main use case is the extension of the communication matrix when refining a sub-system.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Topology	1	
Produces	Communication Matrix	1	

Table 3.148: Define Communication Matrix

3.3.3.1.2 Define Frames

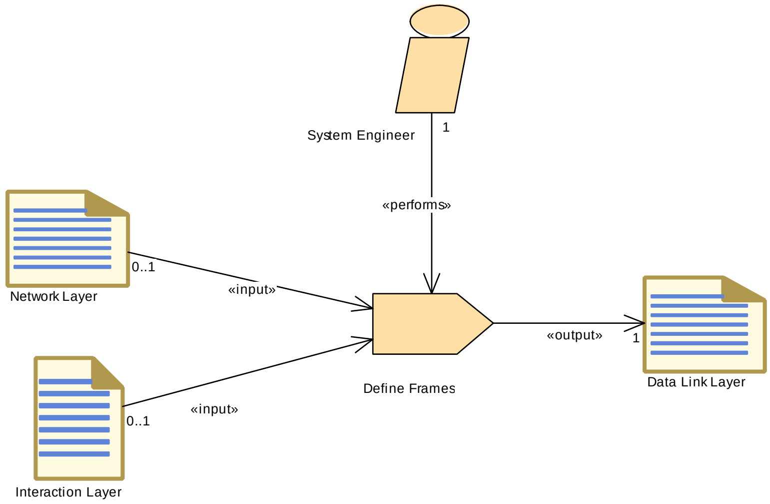

Figure 3.69: Define Frames

Task Definition	**Define Frames**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Tasks		
Brief Description	Define Data Link Layer		
Description	Define the Frame and assign it to a physical channel of a communication cluster. Determine the number, the type, the length and the timing of Frames that are sent or received by the ECUs. Describe the mapping of Pdus (I-Pdus, N-Pdus or NmPdus) into the frame. Define the triggering and the identification of a frame on the physical channel, on which it is sent.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Interaction Layer	0..1	
Consumes	Network Layer	0..1	
Produces	Data Link Layer	1	

Table 3.149: Define Frames

3.3.3.1.3 Define Signal PDUs

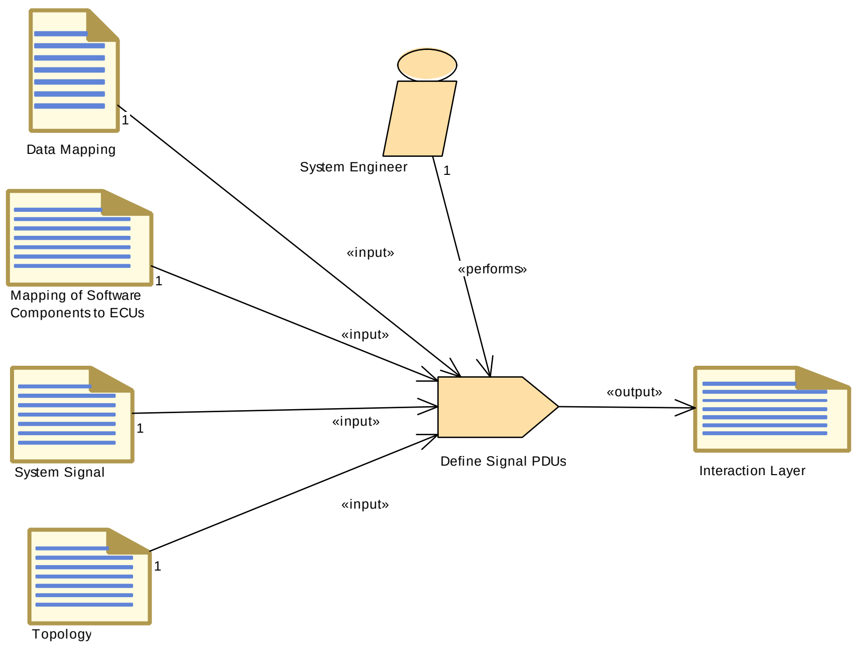

Figure 3.70: Define Signal PDUs

Task Definition	**Define Signal PDUs**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Tasks		
Brief Description	Define the I-PDU and their ISignals		
Description	Define the Signal Pdu that is handled by AUTOSAR COM and assign it to a physical channel of a communication cluster. Determine the length and the timing and describe the mapping of Signals into the Signal Pdu..		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Data Mapping	1	
Consumes	Mapping of Software Components to ECUs	1	
Consumes	System Signal	1	
Consumes	Topology	1	
Produces	Interaction Layer	1	ISignals

Table 3.150: Define Signal PDUs

3.3.3.1.4 Define Secured PDUs

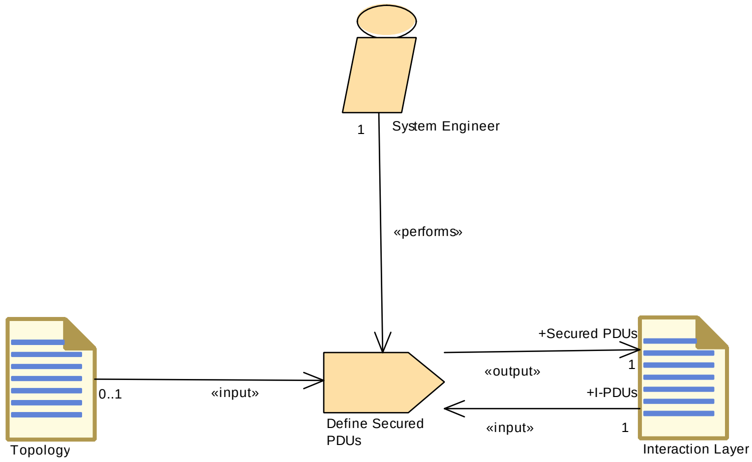

Figure 3.71: Define Secured PDUs

Task Definition	**Define Secured PDUs**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Tasks		
Brief Description	Define Secured PDUs		
Description	If a secured communication of a PDU over network is required, SecuredIPDUs are defined. A secured communication can be established for IPDUs from the Interaction Layer. In addition to the SecuredPDUs corresponding SecureCommunicationProperties are specified that describe how the PDU is secured (e.g. authentication algorithm).		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Interaction Layer	1	I-PDUs: Authentic IPdu that will be secured against manipulation and replay attacks.
Consumes	Topology	0..1	
Produces	Interaction Layer	1	Secured PDUs: Secured IPdu that contains payload of an Authentic IPdu supplemented by additional Authentication Information.

Table 3.151: Define Secured PDUs

3.3.3.1.5 Define TP

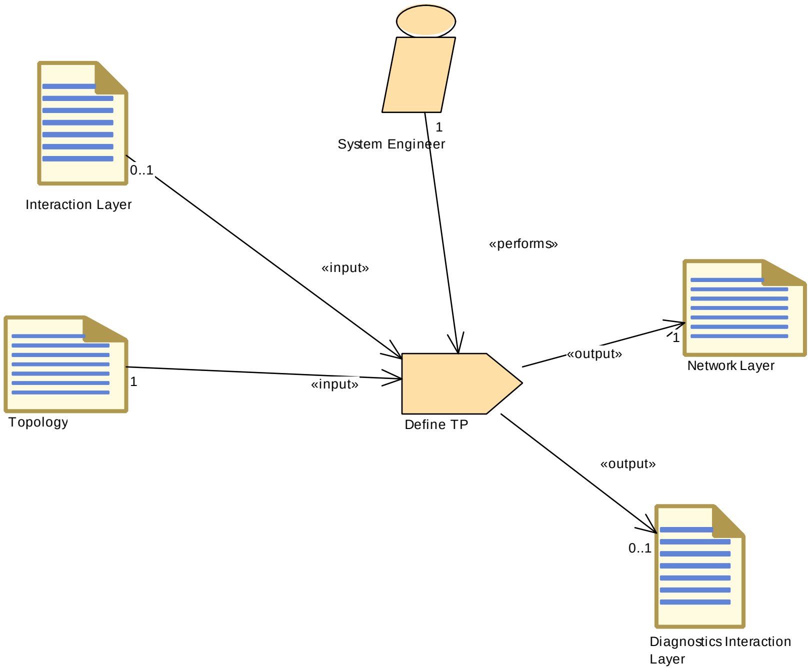

Figure 3.72: Define TP

Task Definition	**Define TP**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Tasks		
Brief Description	Define the Network management and the N-PDUs		
Description	Define the N-PDU - Network Layer Protocol Data Unit (assembled and disassembled in a Transport Protocol module). If an I-PDU does not fit into one frame, a segmentation is needed and will be done through several N-PDUs by the Transport Protocol module. If large COM PDUs are transported by TP, the Interaction Layer should be the Input to the Define TP task. If Diagnostic is used then the Diagnostics Interaction Layer should be an output of Task Define TP.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Topology	1	
Consumes	Interaction Layer	0..1	
Produces	Network Layer	1	
Produces	Diagnostics Interaction Layer	0..1	

Table 3.152: Define TP

3.3.3.1.6 Define Network Management

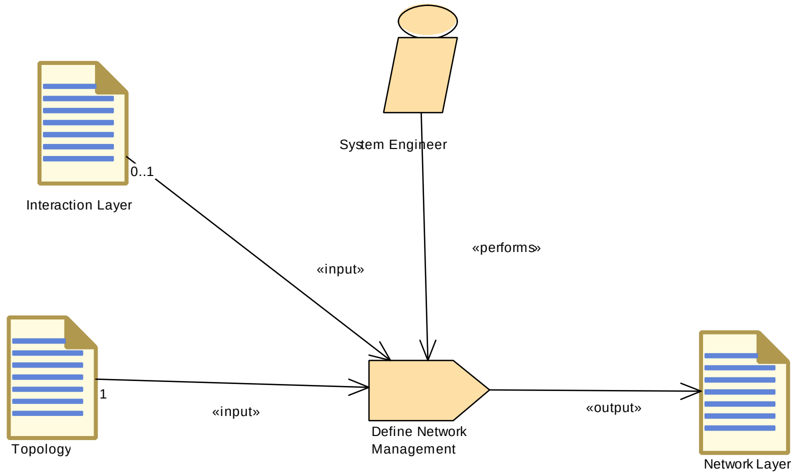

Figure 3.73: Define Network Management

Task Definition	**Define Network Management**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Tasks		
Brief Description			
Description	Define the Network Management that is responsible for the cluster wide coordinated switching of ECUs between operational modes (Network Mode, Bus-sleep Mode). Describe the Nm Pdus and configure the Nm Coordinator, the Nm Clusters and Nm Nodes.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Topology	1	
Consumes	Interaction Layer	0..1	
Produces	Network Layer	1	

Table 3.153: Define Network Management

3.3.3.1.7 Define PDU Gateway

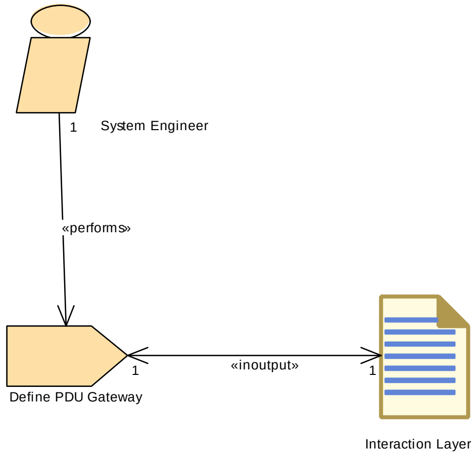

Figure 3.74: Define PDU Gateway

Task Definition	**Define PDU Gateway**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Tasks		
Brief Description	Define the gateway for IPDUs		
Description	Define the gateways that are transferring the I-Pdus from one channel to the other in pairs. Each pair consist of a source and a target referencing to a IPduTriggering. In the case that a Pdu is being gatewayed to more than one channel of the same cluster, all of this gateway relationships shall be specified. Therefore, all affected IpduTriggerings must be described as gateway mappings.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	System Engineer	1	
In/out	Interaction Layer	1	

Table 3.154: Define PDU Gateway

3.3.3.1.8 Define Signal Gateway

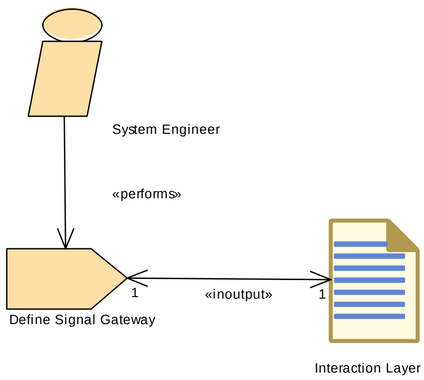

Figure 3.75: Define Signal Gateway

Task Definition	**Define Signal Gateway**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System:: Communication Matrix::Tasks		
Brief Description			
Description	Define the Signal Gateway to describe the routing of signals and signal groups from one Physical Channel to another Physical Channel.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	System Engineer	1	
In/out	Interaction Layer	1	

Table 3.155: Define Signal Gateway

3.3.3.1.9 Define RTE Fan-out

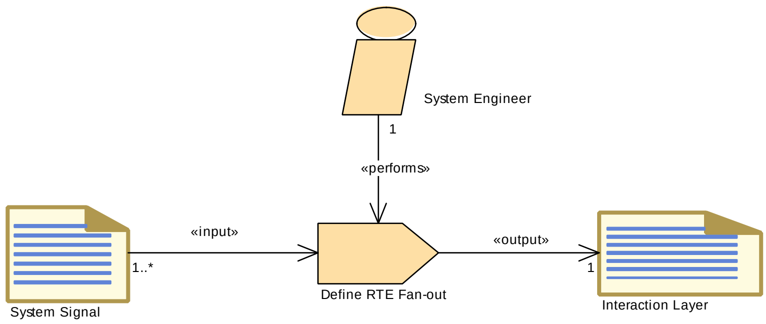

Figure 3.76: Define RTE Fan-out

Task Definition	**Define RTE Fan-out**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System:: Communication Matrix::Tasks		
Brief Description	Define RTE fan-out which are the relation between ISignals and System Signal		
Description	The RTE supports a "signal fan-out" where the same signal (System Signal) is sent in different IPdus to multiple receivers. The Pdu Router supports the "PDU fan-out" where the same IPdu is sent to multiple destinations.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	System Engineer	1	
Consumes	System Signal	1..*	
Produces	Interaction Layer	1	Link of ISignals to System Signals

Table 3.156: Define RTE Fan-out

3.3.3.1.10 Define Transformation Technology

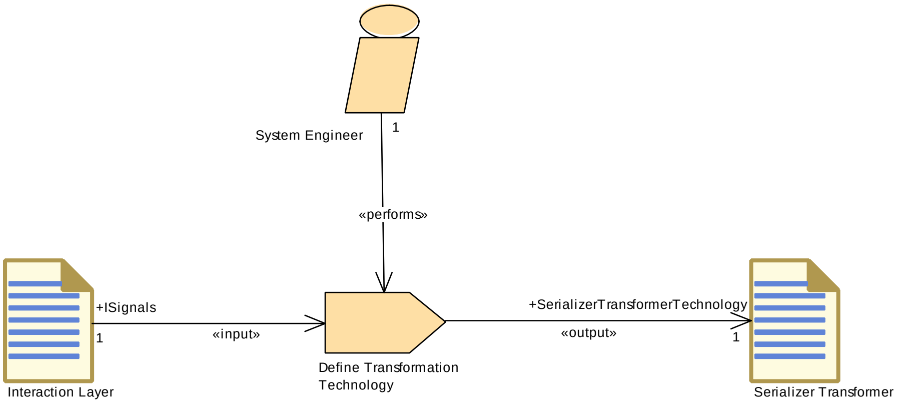

Figure 3.77: Define Transformation Technology

Task Definition	**Define Transformation Technology**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Tasks		
Brief Description	Define the transformer for serialization.		
Description	This task defines the transformer for serialization. In general, there are two possibilities: serialization based on network representation and serialization based on Implementation data types.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	System Engineer	1	
Consumes	Interaction Layer	1	ISignals:
Produces	Serializer Transformer	1	SerializerTransformerTechnology:

Table 3.157: Define Transformation Technology

3.3.3.1.11 Define E2E Transformer Technology

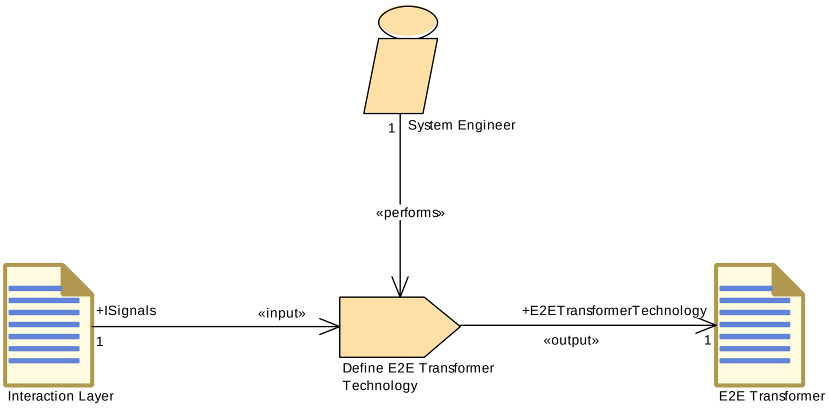

Figure 3.78: Define E2E Transformer Technology

Task Definition	**Define E2E Transformer Technology**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Tasks		
Brief Description	Define the E2E transformer technology.		
Description	This task defines the E2E transformer technology.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Interaction Layer	1	ISignals:
Produces	E2E Transformer	1	E2ETransformerTechnology:

Table 3.158: Define E2E Transformer Technology

3.3.3.1.12 Define Transformation Chain

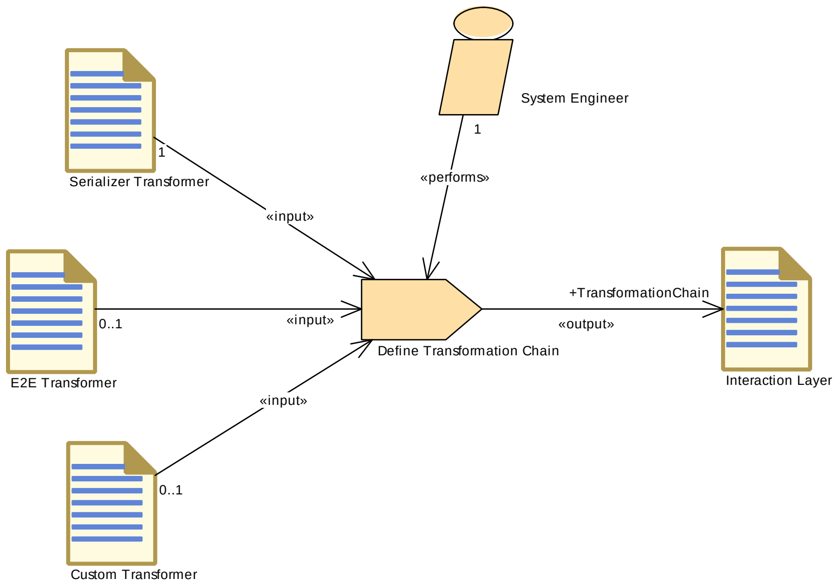

Figure 3.79: Define Transformation Chain

Task Definition	**Define Transformation Chain**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Tasks		
Brief Description	Concatenate several transformers to a transformer chain.		
Description	In this task the several Transformers are concatenated to a Transformer chain producing a set of DataTransformationSets.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Consumes	Serializer Transformer	1	
Consumes	Custom Transformer	0..1	
Consumes	E2E Transformer	0..1	
Produces	Interaction Layer	1	TransformationChain:

Relation Type	Related Element	Mul.	Note

Table 3.159: Define Transformation Chain

3.3.3.2 Work Products

3.3.3.2.1 Communication Layers

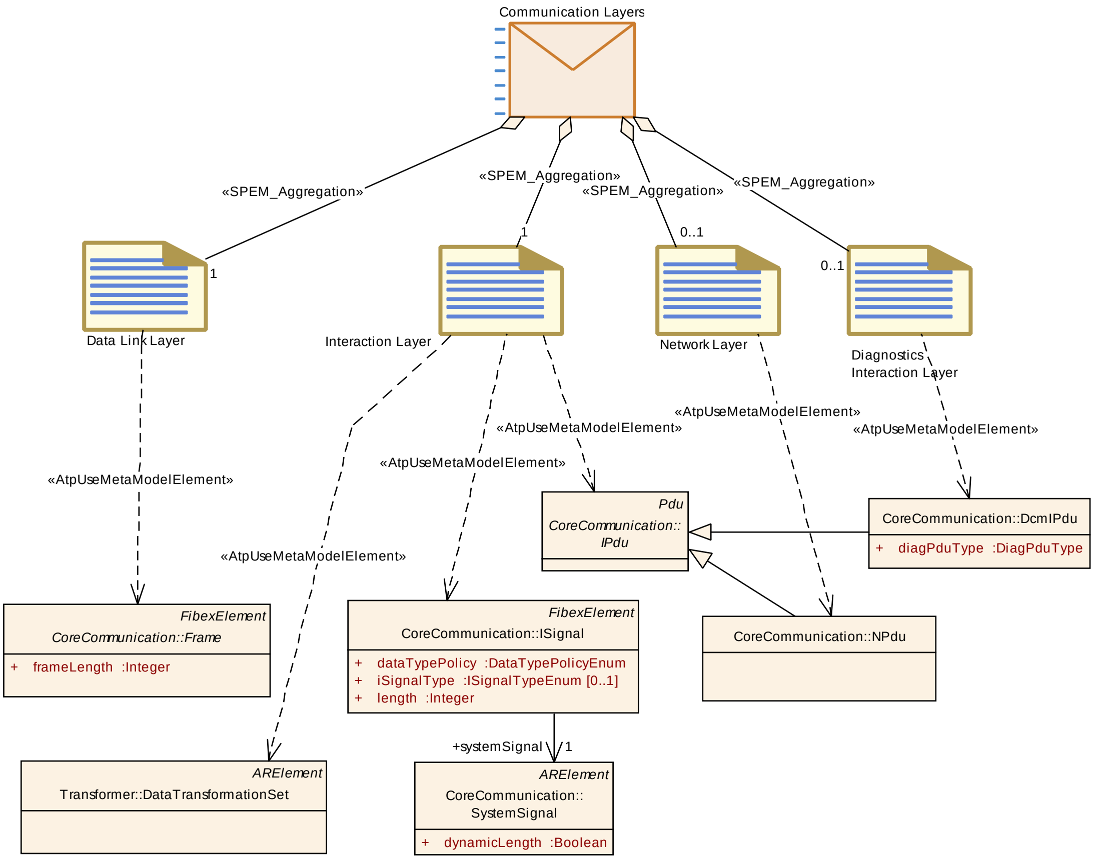

Figure 3.80: Communication Layers

Deliverable	**Communication Layers**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Work products		
Brief Description	Communication Matrix		
Description	It's a container for the description elements of the communication layers		
Kind	Delivered		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	System Description	0..1	
Aggregates	Data Link Layer	1	
Aggregates	Interaction Layer	1	

Relation Type	Related Element	Mul.	Note
Aggregates	Diagnostics Interaction Layer	0..1	
Aggregates	Network Layer	0..1	
Consumed by	Define System Timing	1	
Consumed by	Extract the ECU Communication	1	
Consumed by	Set System Root	1	Only the reference to the artifact is needed

Table 3.160: Communication Layers

3.3.3.2.2 Communication Matrix

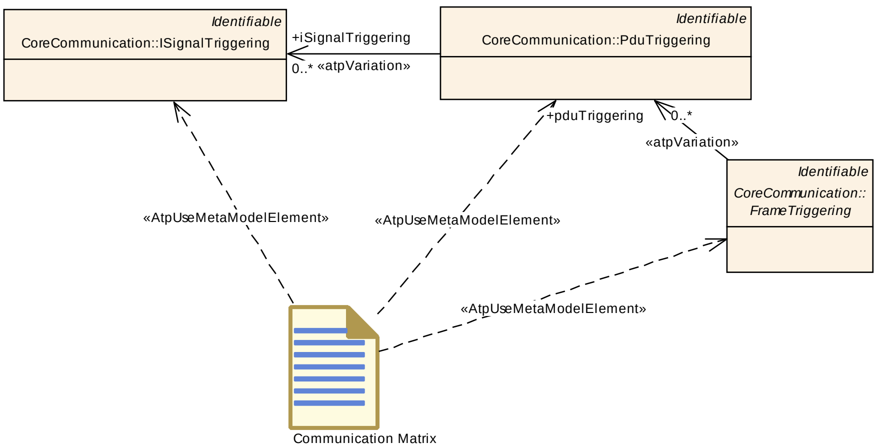

Figure 3.81: Communication Matrix

Artifact	**Communication Matrix**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Work products		
Brief Description			
Description	Define the mapping of the triggering elements within the Physical Channels to the communication connector ports for the individual ECUs. Because the triggering elements are aggregated as splitable elements within the Physical Channels it is possible to define them in an artifact separated from the Topology.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	System Description	0..*	
Produced by	Define Communication Matrix	1	
Use meta model element	FrameTriggering	1	

Relation Type	Related Element	Mul.	Note
Use meta model element	ISignalTriggering	1	
Use meta model element	PduTriggering	1	

Table 3.161: Communication Matrix

3.3.3.2.3 Data Link Layer

Artifact	**Data Link Layer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Work products		
Brief Description	Describes the frames that are used in the Data Link Layer		
Description	Describes the layout of frames to be sent over communication channels. This definition belongs to the Data Link Layer. The Data Link Layer provides the functional and procedural means to transfer data between network entities. This layer is used to transmit data passed by an upper layer (PduR, Tp) between adjacent network nodes. In AUTOSAR the Drivers (FrDrv, CanDrv, LinDrv) and Interfaces (FrIf, CanIf, LinIf) belong to the Data Link Layer.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	Communication Layers	1	
Produced by	Define Frames	1	
Use meta model element	Frame	1	

Table 3.162: Data Link Layer

3.3.3.2.4 Interaction Layer

Artifact	**Interaction Layer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Work products		
Brief Description	Describes the Signals of the Interaction Layer.		
Description	Describes the Signals of the Interaction Layer covering the COM Signals. The Interaction Layer packs one or more signals into assigned COM I-Pdus and passes them to the underlying layer for transfer between nodes in a network.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	Communication Layers	1	
Produced by	Define RTE Fan-out	1	Link of ISignals to System Signals
Produced by	Define Secured PDUs	1	Secured PDUs: Secured IPdu that contains payload of an Authentic IPdu supplemented by additional Authentication Information.

Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define Signal PDUs	1	ISignals
Produced by	Define Transformation Chain	1	TransformationChain:
In/out	Define PDU Gateway	1	
In/out	Define Signal Gateway	1	
Consumed by	Define E2E Transformer Technology	1	ISignals:
Consumed by	Define Secured PDUs	1	I-PDUs: Authentic IPdu that will be secured against manipulation and replay attacks.
Consumed by	Define Transformation Technology	1	ISignals:
Consumed by	Define Frames	0..1	
Consumed by	Define Network Management	0..1	
Consumed by	Define TP	0..1	
Use meta model element	DataTransformationSet	1	
Use meta model element	IPdu	1	
Use meta model element	ISignal	1	

Table 3.163: Interaction Layer

3.3.3.2.5 Diagnostics Interaction Layer

Artifact	**Diagnostics Interaction Layer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Work products		
Brief Description			
Description	Collection of DCM IPDUs.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Communication Layers	0..1	
Produced by	Define TP	0..1	
Use meta model element	DcmIPdu	1	

Table 3.164: Diagnostics Interaction Layer

3.3.3.2.6 Network Layer

Artifact	**Network Layer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Work products		
Brief Description	Describes the PDUs of the Network Layer.		
Description	Describes the PDUs of the Network Layer (N-PDUs and NM-PDUs). The Network Layer's main purposes are : • the segmentation and reassembly of I-PDUs and DCM I-PDUs that do not fit in one of the assigned N-PDUs • the definition of NM-PDUs		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Communication Layers	0..1	
Produced by	Define Network Management	1	
Produced by	Define TP	1	
Consumed by	Define Frames	0..1	
Use meta model element	NPdu	1	

Table 3.165: Network Layer

3.3.3.2.7 Serializer Transformer

Artifact	**Serializer Transformer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Work products		
Brief Description	Serialization of the input data		
Description	This transformer performs the serialization of the input data. It is the first transformer in the transformer chain.		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define Transformation Technology	1	SerializerTransformerTechnology:
Consumed by	Define Transformation Chain	1	

Table 3.166: Serializer Transformer

3.3.3.2.8 E2E Transformer

Artifact	**E2E Transformer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::Communication Matrix::Work products		
Brief Description	E2E protection transformation		
Description	This transformer adds E2E protection related information to the data stream.		
Kind			
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Define E2E Transformer Technology	1	E2ETransformerTechnology:
Consumed by	Define Transformation Chain	0..1	

Table 3.167: E2E Transformer

3.3.4 ECU Extract

3.3.4.1 Tasks

3.3.4.1.1 Extract ECU Topology

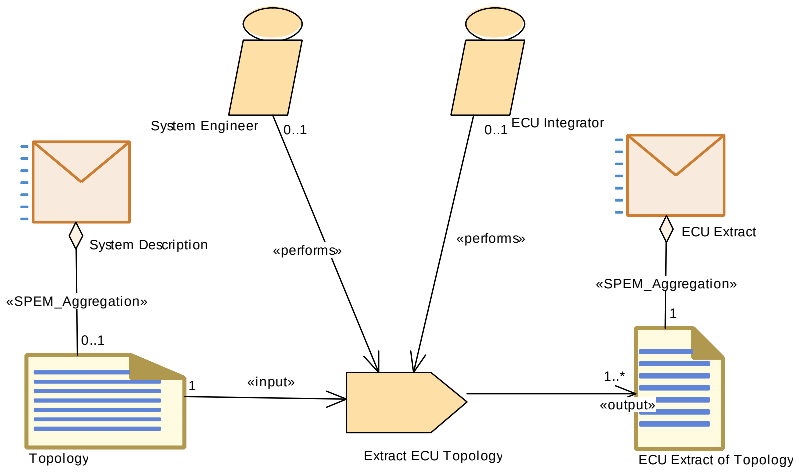

Figure 3.82: Extract ECU Topology

Task Definition	**Extract ECU Topology**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Tasks		
Brief Description	Extract the topology for a single ECU from the System Topology		
Description	From the System or System Extract Topology, extract the topology for a single ECU.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	ECU Integrator	0..1	
Performed by	System Engineer	0..1	
Consumes	Topology	1	
Produces	ECU Extract of Topology	1..*	

Relation Type	***Related Element***	***Mul.***	***Note***

Table 3.168: Extract ECU Topology

3.3.4.1.2 Generate or Adjust ECU Flat Map

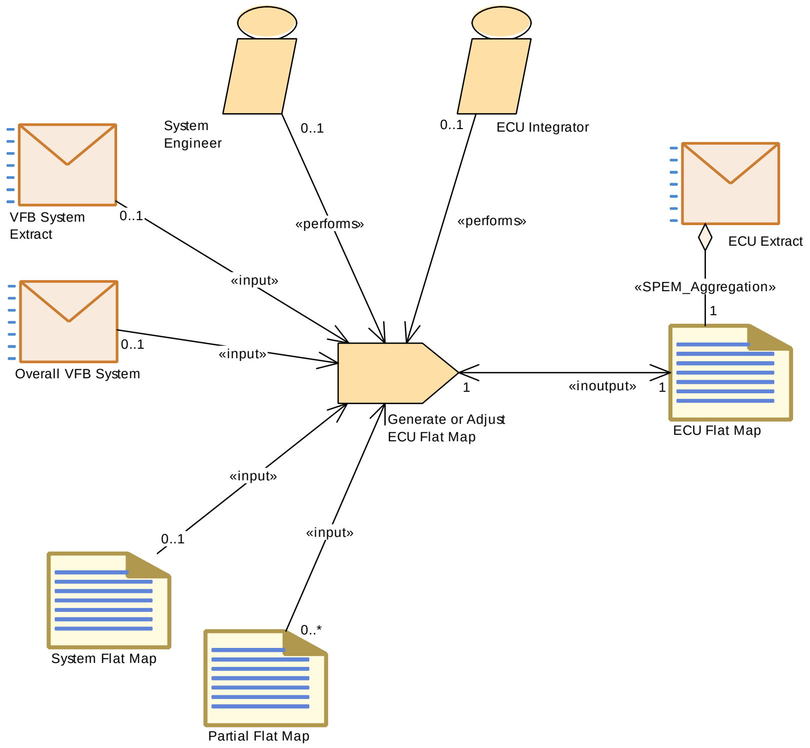

Figure 3.83: Generate or Adjust ECU Flat Map

Task Definition	**Generate or Adjust ECU Flat Map**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Tasks		
Brief Description	Generates and/or adjust the unique names of component prototypes and MCD display data in the scope of a single ECU.		
Description	Generates and/or adjust the unique names of component prototypes and MCD display data in the scope of a single ECU. This information is kept in the so-called ECU Flat Map. The names can be generated according to some rules (e.g. from model elements of the VFB system), taken over from the System Flat Map, from partial Flat Maps, or be manually defined. The task shall always result in an ECU Flat Map with unique names.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	0..1	
Performed by	System Engineer	0..1	
Consumes	Overall VFB System	0..1	Used to set the upstream references in case one starts from a complete system.

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	System Flat Map	0..1	Take over definitions of unique names from system level to ECU level.
Consumes	VFB System Extract	0..1	Used to set the upstream references in case one starts from a system extract.
Consumes	Partial Flat Map	0..*	If Partial Flat Maps were delivered along with software components referring only to ECU internal information, they may be integrated into the ECU Flat Map directly, i.e. without needing the System Flat Map. • The instance refs used in a partial flat map must be taken over and adjusted to the context ECU Extract. • Name conflicts have to be resolved if several partial flat maps are merged.
In/out	ECU Flat Map	1	

Table 3.169: Generate or Adjust ECU Flat Map

3.3.4.1.3 Flatten Software Composition

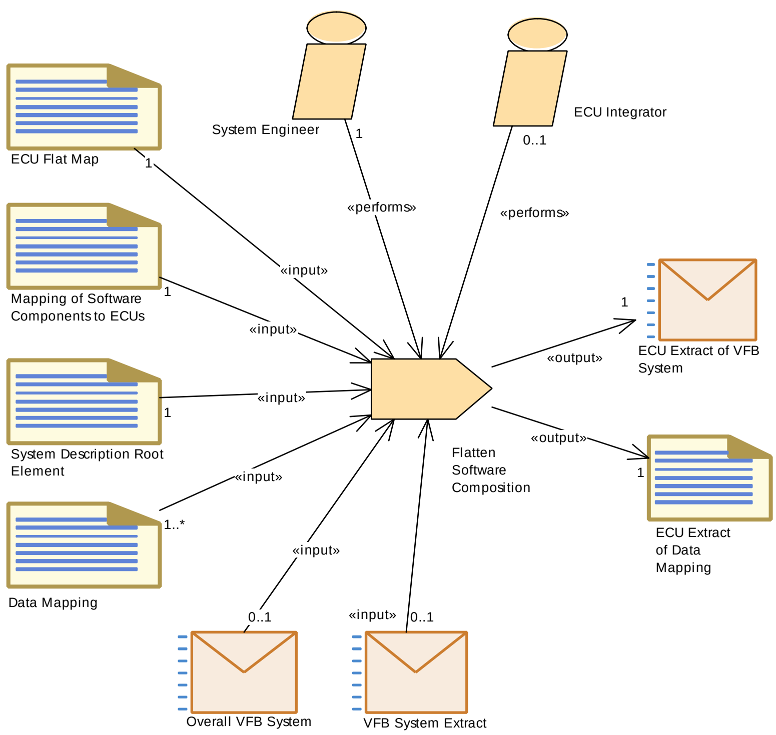

Figure 3.84: Flatten Software Composition

Task Definition	**Flatten Software Composition**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Tasks		
Brief Description	Extract and flatten the ECU Software Composition.		
Description	Generate the complete software composition in an ECU by copying ComponentPrototypes from the VFB description into a flat representation (still without service components). Flat representation means, that all compositions are removed and a "flat" set of ComponetPrototypes is generated. Due to the replication of ComponentPrototypes new names have to be generated for those. These can be predefined in the FlatMap which is an input to this task. The ECU Extract of Data Mapping is also created by this task, as the references to the Data Prototypes need to be created with respect to the new component structure.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	System Engineer	1	
Performed by	ECU Integrator	0..1	
Consumes	ECU Flat Map	1	
Consumes	Mapping of Software Components to ECUs	1	
Consumes	System Description Root Element	1	find the top level composition
Consumes	Data Mapping	1..*	
Consumes	Overall VFB System	0..1	Read relevant elements starting from VFB Top Level System Composition in case transformation starts with the full system.
Consumes	VFB System Extract	0..1	Read relevant elements starting from VFB Top Level System Composition in case transformation starts from the system extract.
Produces	ECU Extract of Data Mapping	1	
Produces	ECU Extract of VFB System	1	

Table 3.170: Flatten Software Composition

3.3.4.1.4 Extract the ECU Communication

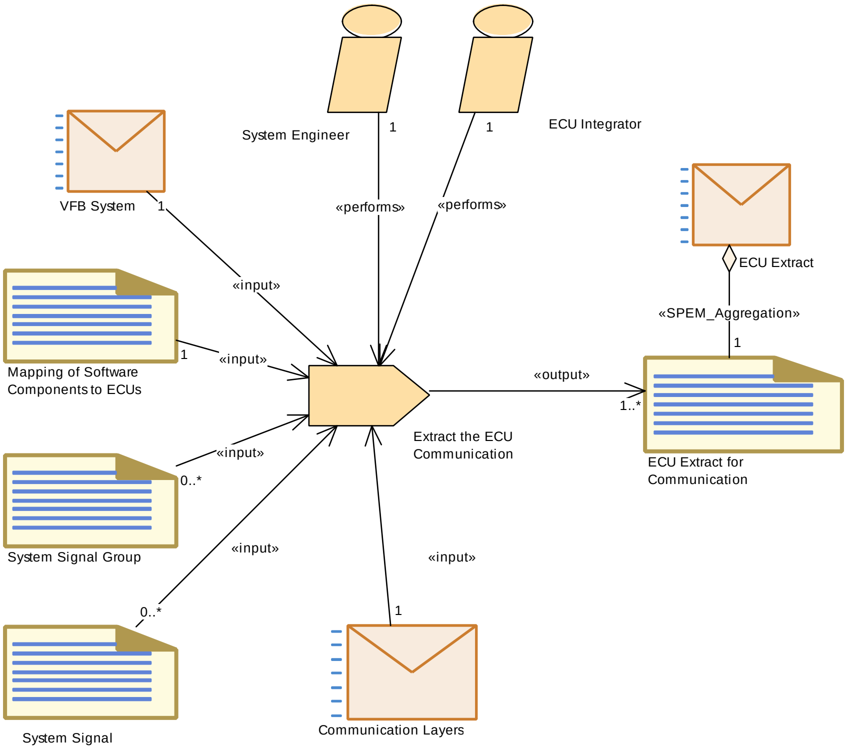

Figure 3.85: Extract the ECU Communication

Task Definition	**Extract the ECU Communication**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Tasks		
Brief Description	The limited-scope communication matrices for an ECU to communicate on all networks on which it is directly connected.		
Description	The limited-scope communication matrices for an ECU to communicate on all networks on which it is directly connected.		
Relation Type	**_Related Element_**	**_Mul._**	**_Note_**
Performed by	ECU Integrator	1	
Performed by	System Engineer	1	
Consumes	Communication Layers	1	
Consumes	Mapping of Software Components to ECUs	1	
Consumes	VFB System	1	Need as input in order to set up the Data Mapping.
Consumes	System Signal	0..*	
Consumes	System Signal Group	0..*	
Produces	ECU Extract for Communication	1..*	

Table 3.171: Extract the ECU Communication

3.3.4.1.5 Extract the ECU Timing Model

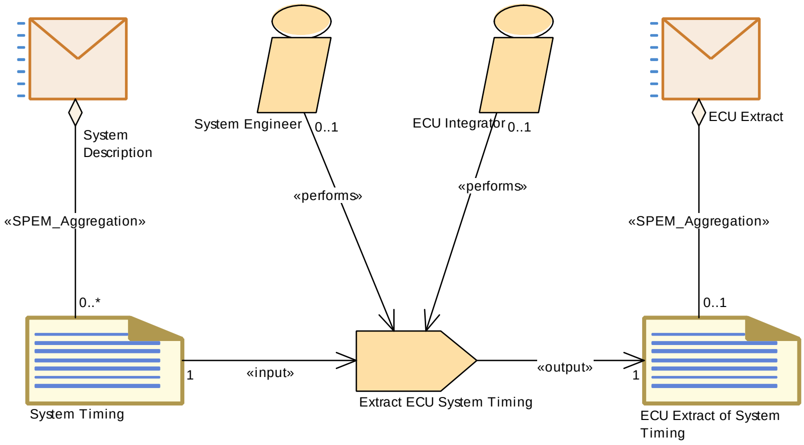

Figure 3.86: Extract the ECU System Timing Model

Task Definition	**Extract ECU System Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Tasks		
Brief Description			
Description	Extract the System Timing Model for a particular ECU from the model for a complete system or system extract.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	0..1	
Performed by	System Engineer	0..1	
Consumes	System Timing	1	
Produces	ECU Extract of System Timing	1	

Table 3.172: Extract ECU System Timing

3.3.4.1.6 Extract the ECU System Variant Model

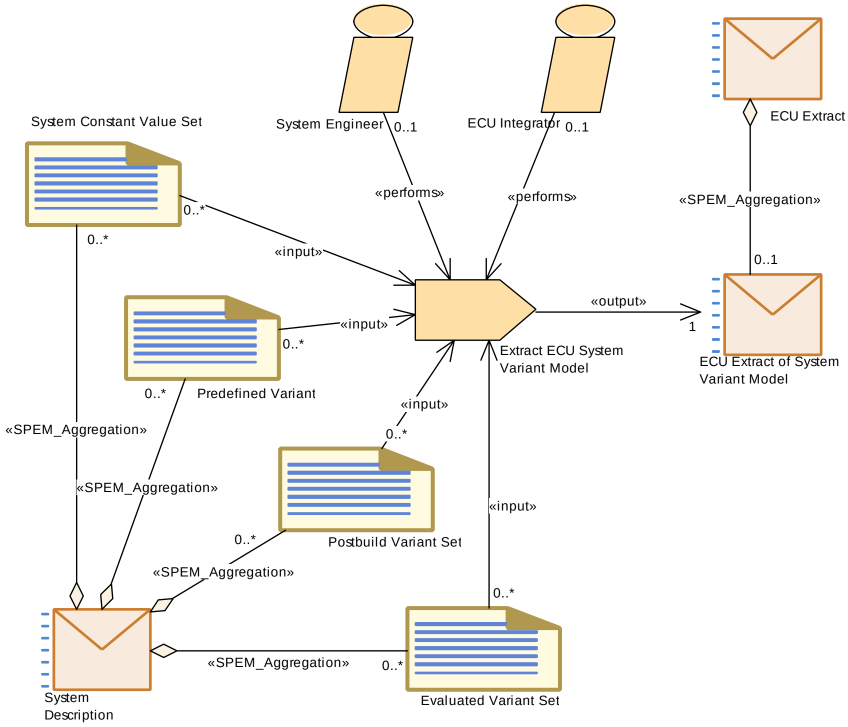

Figure 3.87: Extract the ECU System Variant Model

Task Definition	**Extract ECU System Variant Model**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Tasks		
Brief Description			
Description	Extract the global model elements (ARElements) that are used to describe variants from system or system extract scope to a particular ECU scope. This applies to: • System Constant Value Set • Postbuild Variant Set • Predefined Variant • Evaluated Variant Set They are transformed as far as they are needed into the ECU Extract.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	0..1	
Performed by	System Engineer	0..1	
Consumes	Evaluated Variant Set	0..*	
Consumes	Postbuild Variant Set	0..*	
Consumes	Predefined Variant	0..*	

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	System Constant Value Set	0..*	
Produces	ECU Extract of System Variant Model	1	

Table 3.173: Extract ECU System Variant Model

3.3.4.1.7 Extract ECU Rapid Prototyping Scenario

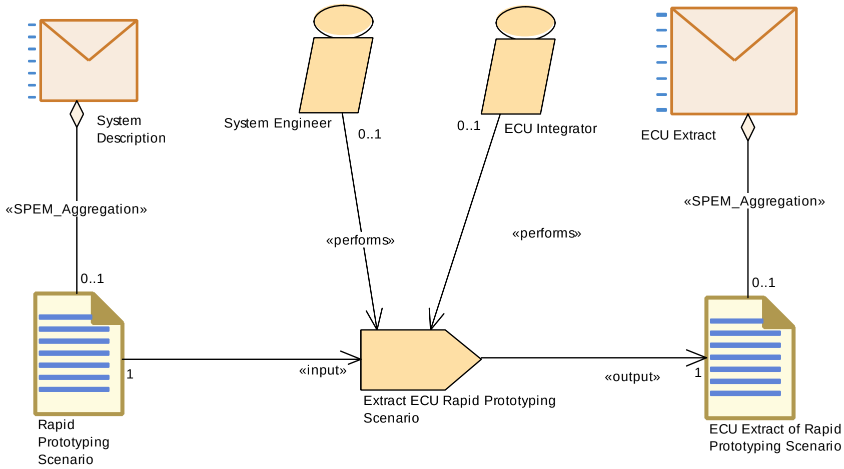

Figure 3.88: Extract ECU Rapid Prototyping Scenario

Task Definition	**Extract ECU Rapid Prototyping Scenario**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Tasks		
Brief Description	Extracts the ECU Rapid Prototyping Scenario		
Description	From the System Rapid Prototyping Scenario extract the entities relevant for the single ECU.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	0..1	
Performed by	System Engineer	0..1	
Consumes	Rapid Prototyping Scenario	1	
Produces	ECU Extract of Rapid Prototyping Scenario	1	

Table 3.174: Extract ECU Rapid Prototyping Scenario

3.3.4.2 Work Products

3.3.4.2.1 ECU Extract

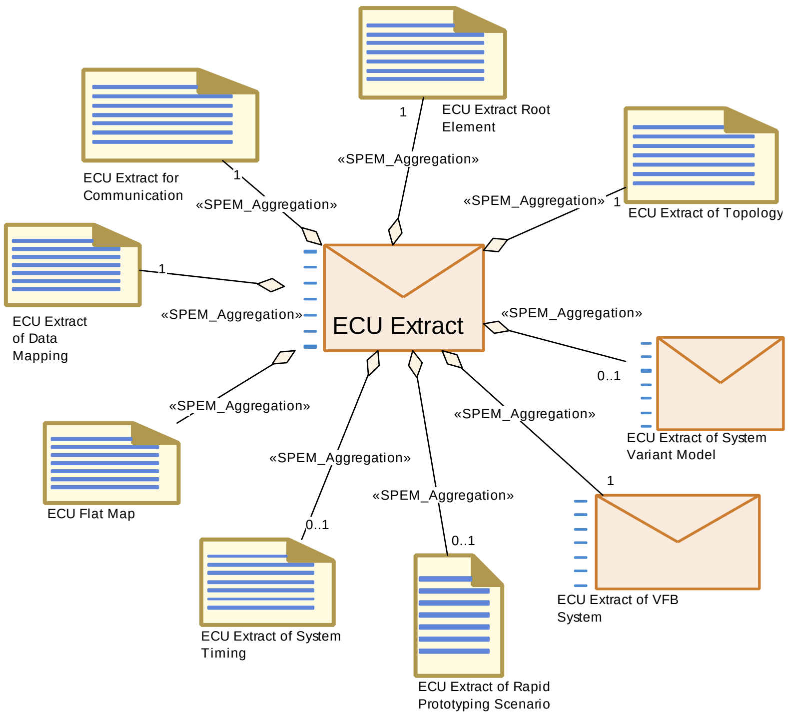

Figure 3.89: ECU Extract

Deliverable	**ECU Extract**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Work products		
Brief Description	A version of the System Description, with information pertaining to a single ECU.		
Description	A deliverable used to describe the ECU specific view on the System Description. The ECU Extract is fully decomposed and contains only Atomic Software Components.It is the basis for setting up the ECU Configuration. A timing model is optionally included. This deliverable may contain variation points in its XML artifacts which need to be bound for the ECU. If such variation points are present, the ECU extract may optionally include Predefined Variants in order to predefine variants for later selection and an Evaluated Variant Set (this is expressed by artifact ECU Extract of System Variant Model). This deliverable corresponds to the system description with the system category "ECU_EXTRACT" (see [TPS_SYST_01003]).		
Kind	Delivered		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	ECU Extract Root Element	1	

Relation Type	Related Element	Mul.	Note
Aggregates	ECU Extract for Communication	1	
Aggregates	ECU Extract of Data Mapping	1	
Aggregates	ECU Extract of Topology	1	
Aggregates	ECU Extract of VFB System	1	
Aggregates	ECU Flat Map	1	
Aggregates	ECU Extract of Rapid Prototyping Scenario	0..1	
Aggregates	ECU Extract of System Timing	0..1	
Aggregates	ECU Extract of System Variant Model	0..1	
Produced by	Generate ECU Extract	1	
Produced by	Develop Sub-System	1..*	
Produced by	Develop System	1..*	
Consumed by	Configure Com	1	
Consumed by	Configure Debug	1	
Consumed by	Configure Diagnostics	1	Application software requirements for diagnostics, especially SwcServiceDependency and ServiceNeeds.
Consumed by	Configure ECUC	1	
Consumed by	Configure NvM	1	Application software requirements for NvM, especially SwcServiceDependency and ServiceNeeds.
Consumed by	Configure RTE	1	Elements of the System Description and VFB Description are referred by the RTE configuration. Optional Input: ECU Extract of System Timing, e.g. execution order constraints.
Consumed by	Configure Watchdog Manager	1	Application software requirements for WdgM, especially SwcServiceDependency and ServiceNeeds.
Consumed by	Connect Service Component	1	Find the ports on the application side to be connected to the Service Component.
Consumed by	Define Integration Variant	1	
Consumed by	Generate Base Ecu Configuration	1	

Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Generate RTE	1	Find the VFB description of all Atomic Software Components on this ECU and the relevant parts of the system description. The ECU Flat Map is also an input. Meth.bindingTime = SystemDesignTime
Consumed by	Generate RTE Postbuild Dataset	1	Meth.bindingTime = LinkTime
Consumed by	Generate RTE Prebuild Dataset	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Generate Updated ECU Configuration	1	
Consumed by	Integrate Software for ECU	1	
Consumed by	Prepare ECU Configuration	1	
Consumed by	Update ECU Configuration	1	
Consumed by	Configure Mode Management	0..1	Application software requirements for NvM, especially SwcServiceDependency and ServiceNeeds. Input in case atomic software components are available.
Consumed by	Create MC Function Model	0..1	The ECU Flat Map can be used to define references to variables and parameters which are later visible in A2L. Furthermore, the ECU Extract can be used to find the relevant software components.
Consumed by	Create Service Component	0..1	Input information about the Service Ports and Service Dependencies of the software components.
Consumed by	Define ECU Timing	0..1	Needed to set up links to the elements of the ECU extract.
	Configure Transformer	1	

Table 3.175: ECU Extract

3.3.4.2.2 ECU Extract Root Element

Artifact	**ECU Extract Root Element**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Work products		
Brief Description			
Description	Extract of the System root element for a specific ECU.		
Kind	AUTOSAR XML		
Extends	System		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	ECU Extract	1	
Consumed by	Generate Rapid Prototyping Wrapper	1	
Use meta model element	System	1	

Table 3.176: ECU Extract Root Element

3.3.4.2.3 ECU Extract of VFB System

Deliverable	**ECU Extract of VFB System**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Work products		
Brief Description	Contains the complete software composition in an ECU, copied from the VFB description into a flat representation, it is still without service components.		
Description	Contains the complete software composition in an ECU, copied from the VFB description into a flat representation, that means it is still without service components. Flat representation means, that all compositions have been removed and a "flat" set of ComponentPrototypes was generated (including their connectors) which are put into the top level composition of the ECU.		
Kind	Delivered		
Extends	VFB System		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	ECU Extract	1	
Produced by	Flatten Software Composition	1	
Consumed by	Generate Rapid Prototyping Wrapper	1	
Use meta model element	RootSwCompositionPrototype	1	

Table 3.177: ECU Extract of VFB System

3.3.4.2.4 ECU Extract of Data Mapping

Artifact	**ECU Extract of Data Mapping**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Work products		
Brief Description			
Description	ECU extract of the mapping of data prototypes from the (flattened) VFB description to System Signals.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	ECU Extract	1	
Produced by	Flatten Software Composition	1	
Use meta model element	DataMapping	1	

Table 3.178: ECU Extract of Data Mapping

3.3.4.2.5 ECU Extract of Topology

Artifact	**ECU Extract of Topology**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Work products		
Brief Description	A view of the topology centered around a single ECU.		
Description	A view of the topology centered around a single ECU.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	ECU Extract	1	
Produced by	Extract ECU Topology	1..*	
Use meta model element	Communication Cluster	1	
Use meta model element	EcuInstance	1	

Table 3.179: ECU Extract of Topology

3.3.4.2.6 ECU Extract for Communication

Artifact	**ECU Extract for Communication**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Work products		
Brief Description	A version of the System Communication Matrix work product, with information pertaining to a single ECU.		
Description	This artifact represents an extract of the System Description elements for communication with respect to a single ECU. It provides all information needed to let the ECU communicate on all networks on which it is directly connected. It is extracted from these system artifacts: • Communication Matrix • Communication Layers • System Signal(s) • System Signal Group(s)		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	ECU Extract	1	
Produced by	Extract the ECU Communication	1..*	
Use meta model element	FibexElement	1	

Table 3.180: ECU Extract for Communication

3.3.4.2.7 ECU Extract of System Timing

Artifact	**ECU Extract of System Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Work products		
Brief Description			
Description	The extract of the System Timing for a particular ECU.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	ECU Extract	0..1	
Produced by	Extract ECU System Timing	1	
Consumed by	Define ECU Timing	0..1	
Use meta model element	SystemTiming	1	

Table 3.181: ECU Extract of System Timing

3.3.4.2.8 ECU Extract of System Variant Model

Deliverable	**ECU Extract of System Variant Model**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Work products		
Brief Description			
Description	An extract of the System artifacts • System Constant Value Set • Postbuld Variant Set • Predefined Variant • Evaluated Variant Set It contains only the elements relevant for a particular ECU.		
Kind	Delivered		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	ECU Extract	0..1	
Aggregates	Evaluated Variant Set	0..*	
Aggregates	Postbuild Variant Set	0..*	
Aggregates	Predefined Variant	0..*	
Aggregates	System Constant Value Set	0..*	
Produced by	Extract ECU System Variant Model	1	
Consumed by	Generate Rapid Prototyping Wrapper	0..1	

Table 3.182: ECU Extract of System Variant Model

3.3.4.2.9 ECU Flat Map

Artifact	**ECU Flat Map**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Work products		
Brief Description	Mapping of instance names to nested model elements. Use cases: Resolve name conflicts when flattening VFB software compositions; provide unique names for measurement and calibration data.		
Description	The flat map is a list of elements, each element represents exactly one node (e.g. a component instance or data element) of the instance tree of a software system. The purpose of this element is to map the various nested representations of this instance to a flat representation and assign a unique name to it. The name will be unique in the scope of a single ECU. (Note that additional alias names can be defined via artifact Alias Name Set.) Use cases: • Specify the display name of a data object for measurement and calibration. This serves as an input for the calibration support which is produced by the RTE generator. The RTE generator needs to find the attributes assigned to these data via the attached references. • Specify a unique name for an instance of a component prototype in the ECU extract of the system description. This information is needed to set up the ECU extract. • Assign initial values to calibration parameters as input for the RTE generator.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	ECU Extract	1	
In/out	Generate or Adjust ECU Flat Map	1	
Consumed by	Flatten Software Composition	1	
Consumed by	Generate Local MC Data Support	1	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Rapid Prototyping Wrapper	1	
Consumed by	Provide RTE Calibration Dataset	1	
Consumed by	Generate A2L	0..1	The ECU Flat Map is needed in case the A2L generator has to process an MC Function Model that relates to data in the ECU Flat Map.
Use meta model element	FlatInstanceDescriptor	1	

Table 3.183: ECU Flat Map

3.3.4.2.10 ECU Extract of Rapid Prototyping Scenario

Artifact	**ECU Extract of Rapid Prototyping Scenario**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System::ECU Extract::Work products		
Brief Description	Description of the (required) bypass points in the ECU.		
Description	Description of the (required) bypass points in the ECU.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	ECU Extract	0..1	
Produced by	Extract ECU Rapid Prototyping Scenario	1	
In/out	Refine Rapid Prototyping Scenario	1	
Consumed by	Generate Rapid Prototyping Wrapper	1	

Table 3.184: ECU Extract of Rapid Prototyping Scenario

3.4 Software Component

This chapter contains the definition of work products and tasks used for the development of a single software component against a given VFB description. For the definition of the relevant meta-model elements refer to [6].

3.4.1 Tasks

3.4.1.1 Define Software Component Internal Behavior

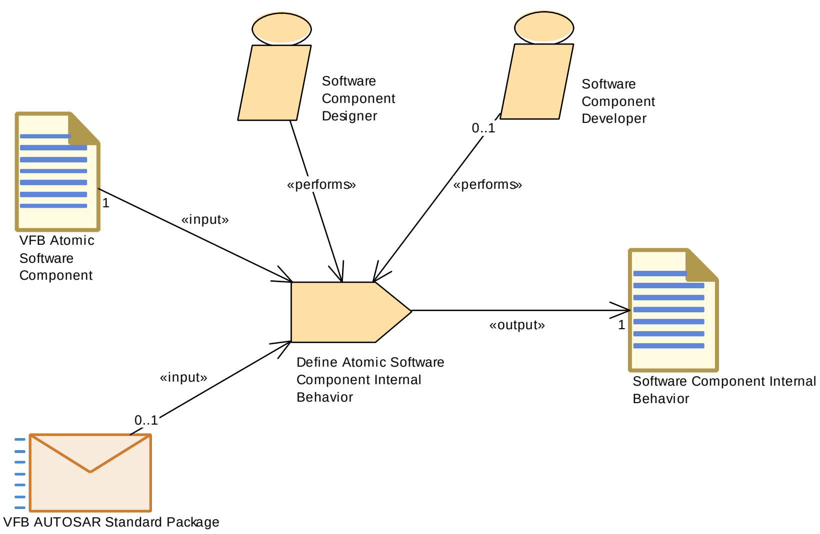

Figure 3.90: Define Software Component Internal Behavior

Task Definition	**Define Atomic Software Component Internal Behavior**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description	Define the InternalBehavior in relation to a given AtomicSoftwareComponentType		
Description	Define the InternalBehavior in relation to a given AtomicSoftwareComponentType so that an RTE API can be generated. This includes the definition of Runnables, RTE Events, Inter-Runnable variables, etc.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Designer	1	
Performed by	Software Component Developer	0..1	
Consumes	VFB Atomic Software Component	1	
Consumes	VFB AUTOSAR Standard Package	0..1	Use standardized elements (e.g. Data Types) as blueprints (as far as applicable) to create the corresponding elements of the actual project.
Produces	Software Component Internal Behavior	1	

Table 3.185: Define Atomic Software Component Internal Behavior

3.4.1.2 Define Partial Flat Map

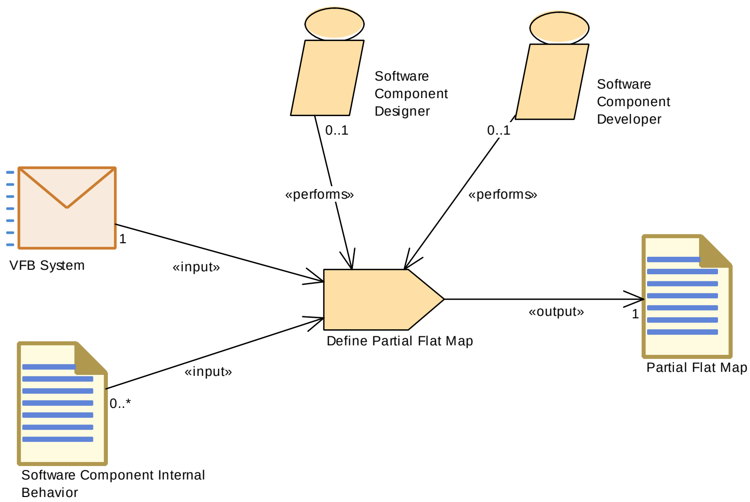

Figure 3.91: Define Partial Flat Map

Task Definition	**Define Partial Flat Map**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description			
Description	Define a Partial Flat Map for an intended delivery of Atomic Software Components.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Designer	0..1	
Performed by	Software Component Developer	0..1	
Consumes	VFB System	1	Various parts of a given VFB system will be used as input: • Refer to parameters and variables in port interfaces and their data types. • In order to define unique names, also other the component definitions not in the scope of the partial flat map might be checked. • Set a link to the context of the Flat Map, e.g. a VFB Composition.
Consumes	Software Component Internal Behavior	0..*	Refer to parameter and variables defined in the Internal Behavior of one or more Atomic Software Components.
Produces	Partial Flat Map	1	

Table 3.186: Define Partial Flat Map

3.4.1.3 Define Software Component Timing

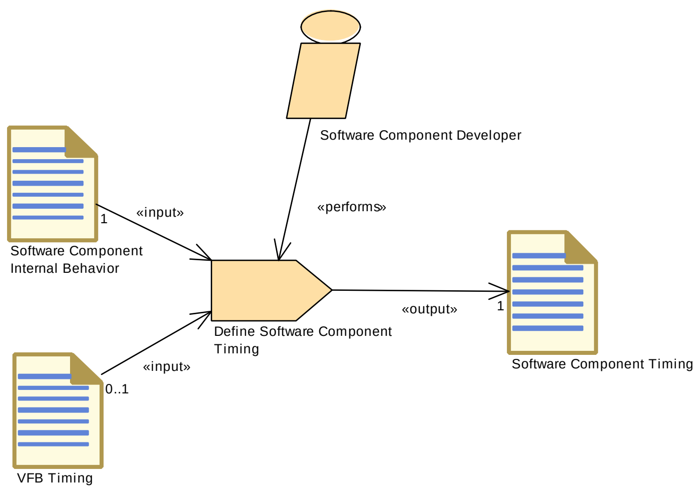

Figure 3.92: Define Software Component Timing

Task Definition	**Define Software Component Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description	Define SWCTiming (TimingDescription and TimingConstraints) for the Internal Behavior (RunnableEntities) of a Software Component		
Description	Define SWCTiming (TimingDescription and TimingConstraints) of a software component. A software component can either be of type AtomicSWComponentType or CompositionSWComponentType. In the former case, the task allows to describe timing description and constraints for the InternalBehavior of the AtomicSWComponentType. In the latter case, timing descriptions and constraints can be defined for all Atomic Software Components in the CompositionSWComponentType.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Developer	1	
Consumes	Software Component Internal Behavior	1	
Consumes	VFB Timing	0..1	
Produces	Software Component Timing	1	

Table 3.187: Define Software Component Timing

3.4.1.4 Define SymbolProps for Types

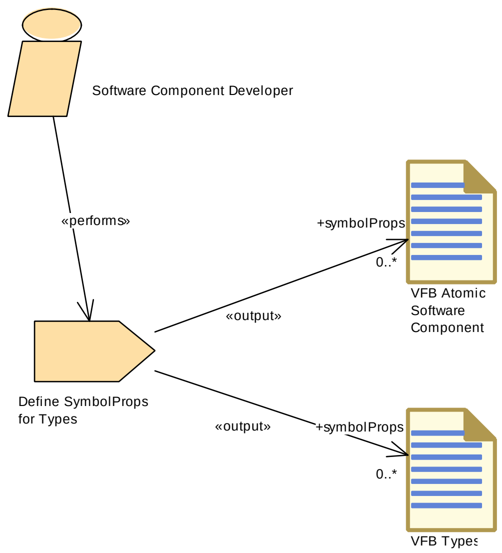

Figure 3.93: Define SymbolProps for Types

Task Definition	**Define SymbolProps for Types**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description	Define SymbolProps for types in order to resolve name conflicts in the code.		
Description	Redefines the symbols used by the RTE contract for the names of software component types and/or implementation data types (in the code as well as in certain header file names). This task is used to resolve name conflicts between different software components without changing the VFB model.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Developer	1	
Produces	VFB Atomic Software Component	0..*	symbolProps: The symbolProps attribute redefines the software component type name used in the code of the RTE. This resolves name clashes among different software component types designed accidentally with the same shortName. Note that this output is a splitable element, so it can be added later without changing the VFB model.
Produces	VFB Types	0..*	symbolProps: The symbolProps attribute redefines the implementation data type name used in the code of the RTE and/or the component. This resolves name clashes among different implementation data types designed accidentally with the same shortName. Note that this output is a splitable element, so it can be added later without changing the VFB model.

Table 3.188: Define SymbolProps for Types

3.4.1.5 Add Documentation to the Software Component

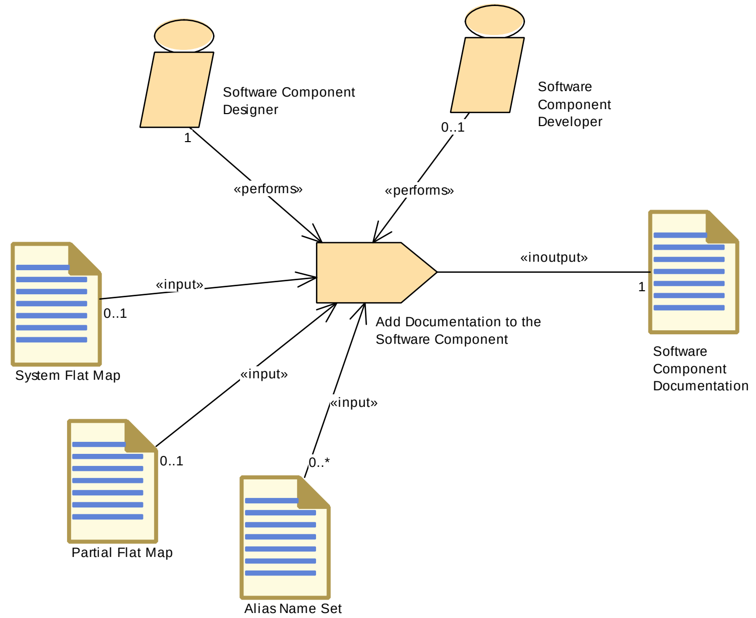

Figure 3.94: Add Documentation to the Software Component

Task Definition	**Add Documentation to the Software Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description	Add documentation to the Software Component		
Description	Add documentation to the Software Component describing the functionality, how to test it, the calibration uses, the maintenance and diagnosis issues.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Designer	1	
Performed by	Software Component Developer	0..1	
Consumes	Partial Flat Map	0..1	Optional input in order to refer to unique names defined in component or composition context.
Consumes	System Flat Map	0..1	Optional input in order to refer to unique names defined in system context.
Consumes	Alias Name Set	0..*	Optional input in order to refer to unique names defined in an Alias Name Set (e.g. System Constants).
In/out	Software Component Documentation	1	

Table 3.189: Add Documentation to the Software Component

3.4.1.6 Generate Atomic Software Component Contract Header Files

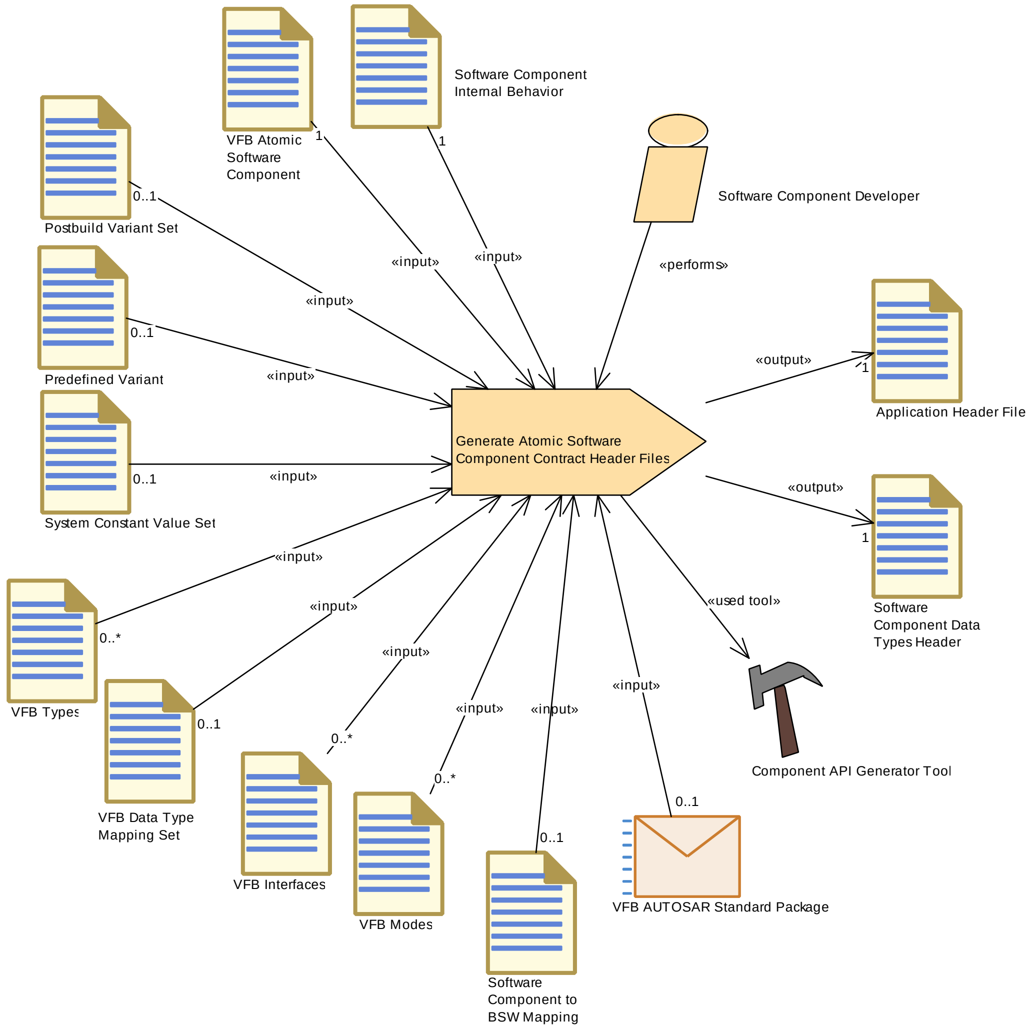

Figure 3.95: Generate Atomic Software Component Contract Header Files

Task Definition	**Generate Atomic Software Component Contract Header Files**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description	Generate the component contract header files.		
Description	Generate the component header files as part of the so-called "contract phase". These headers will allow to link the component lateron with the RTE. The header can still contain variants with later binding time, therefore the information about these variants is contained in the input to this task. Meth.bindingTime = CodeGenerationTime		
Relation Type	**Related Element**	**Mul.**	**Note**
Performed by	Software Component Developer	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Software Component Internal Behavior	1	Meth.bindingTime = SystemDesignTime
Consumes	VFB Atomic Software Component	1	Meth.bindingTime = SystemDesignTime
Consumes	Postbuild Variant Set	0..1	
Consumes	Predefined Variant	0..1	
Consumes	Software Component to BSW Mapping	0..1	If a Software Component is mapped to a BSW module description, this input is optionally needed already in the contract phase in order to ensure that the generated prototypes for runnables are consistent with the definitions in Software Component and BSW. Meth.bindingTime = SystemDesignTime
Consumes	System Constant Value Set	0..1	Meth.bindingTime = SystemDesignTime
Consumes	VFB AUTOSAR Standard Package	0..1	
Consumes	VFB Data Type Mapping Set	0..1	Meth.bindingTime = SystemDesignTime
Consumes	VFB Interfaces	0..*	Meth.bindingTime = SystemDesignTime
Consumes	VFB Modes	0..*	Meth.bindingTime = SystemDesignTime
Consumes	VFB Types	0..*	Meth.bindingTime = SystemDesignTime
Produces	Application Header File	1	Meth.bindingTime = CodeGenerationTime
Produces	Software Component Data Types Header	1	Meth.bindingTime = CodeGenerationTime
Used tool	Component API Generator Tool	1	

Table 3.190: Generate Atomic Software Component Contract Header Files

3.4.1.7 Generate Component Header File in Vendor Mode

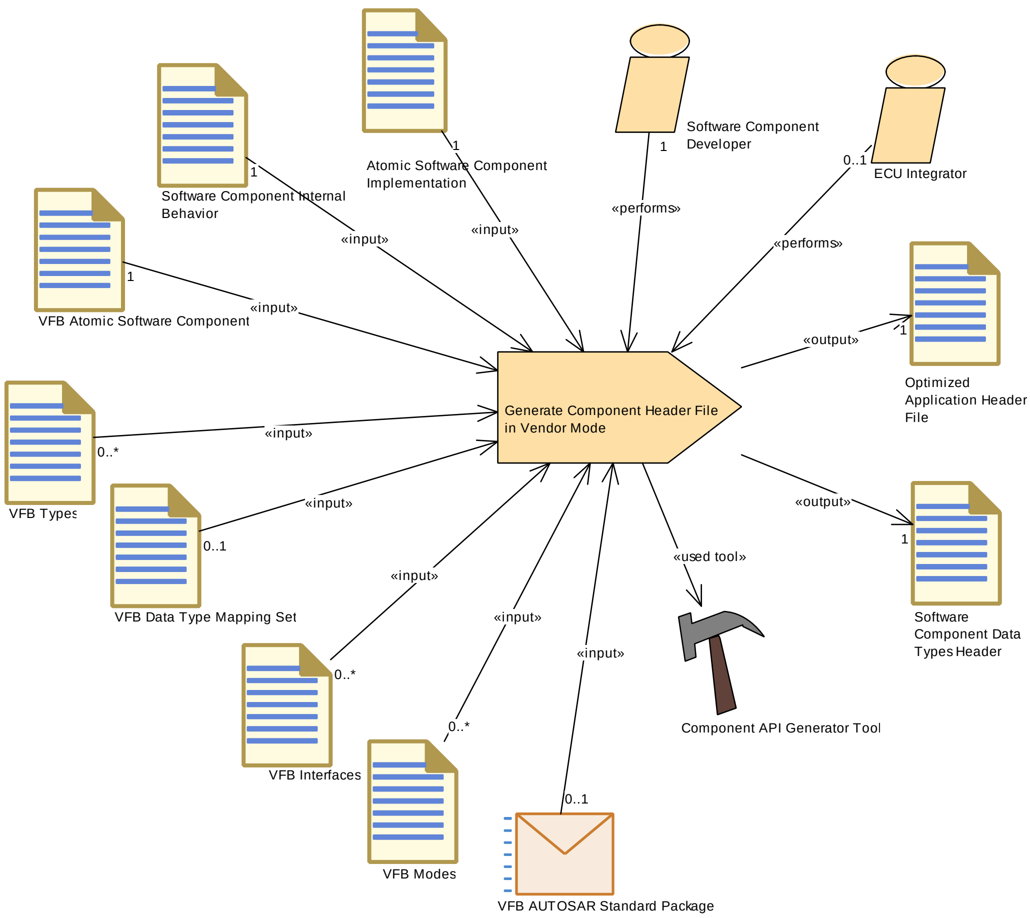

Figure 3.96: Generate Component Header File in Vendor Mode

Task Definition	**Generate Component Header File in Vendor Mode**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description	Generate an optimized component header file. This is achieved by using the RTE's vendor mode.		
Description	Generate an optimized component header file. This is achieved by using the RTE's vendor mode. Meth.bindingTime = CodeGenerationTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Developer	1	
Performed by	ECU Integrator	0..1	
Consumes	Atomic Software Component Implementation	1	Meth.bindingTime = SystemDesignTime
Consumes	Software Component Internal Behavior	1	Meth.bindingTime = SystemDesignTime

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	VFB Atomic Software Component	1	Meth.bindingTime = SystemDesignTime
Consumes	VFB AUTOSAR Standard Package	0..1	
Consumes	VFB Data Type Mapping Set	0..1	Meth.bindingTime = SystemDesignTime
Consumes	VFB Interfaces	0..*	Meth.bindingTime = SystemDesignTime
Consumes	VFB Modes	0..*	Meth.bindingTime = SystemDesignTime
Consumes	VFB Types	0..*	Meth.bindingTime = SystemDesignTime
Produces	Optimized Application Header File	1	Meth.bindingTime = CodeGenerationTime
Produces	Software Component Data Types Header	1	Meth.bindingTime = CodeGenerationTime
Used tool	Component API Generator Tool	1	

Table 3.191: Generate Component Header File in Vendor Mode

3.4.1.8 Generate Component Prebuild Data Set

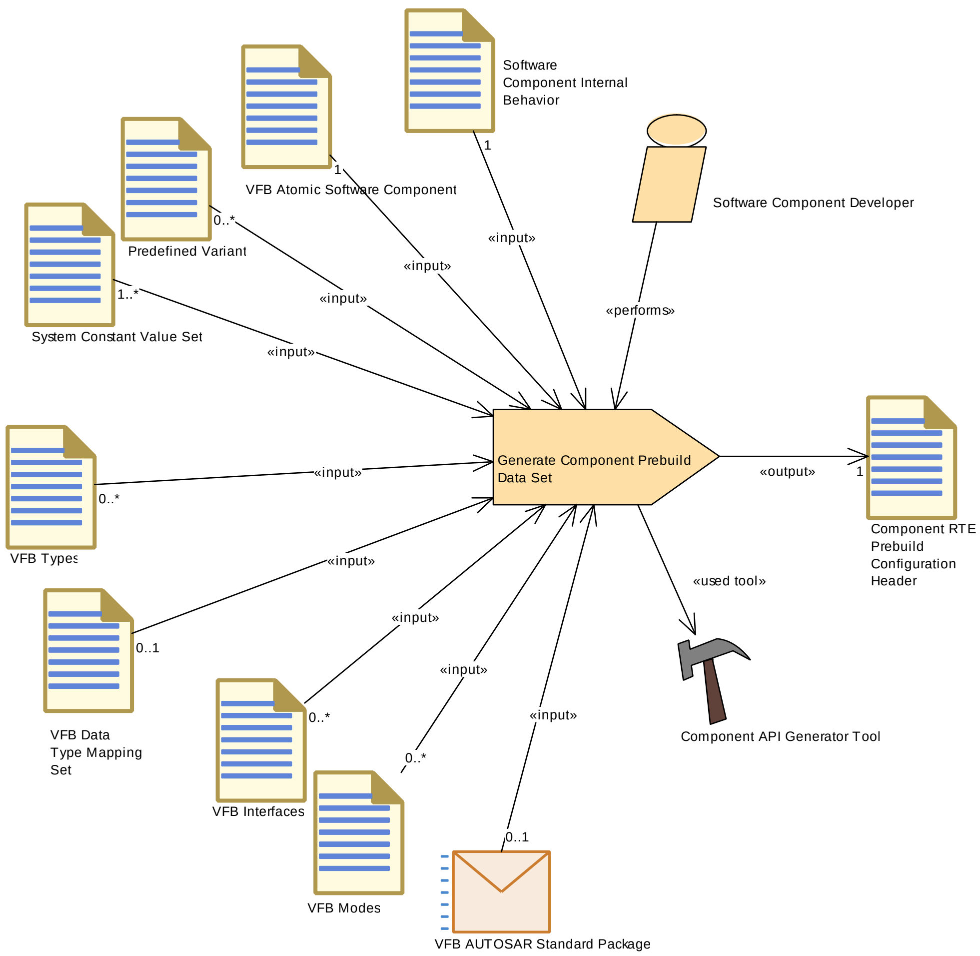

Figure 3.97: Generate Component Prebuild Data Set

Task Definition	**Generate Component Prebuild Data Set**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description	Prebuild Data Set Generation Phase for a software component: It binds all variations which need to be set after generation of the RTE contract header but before compilation of the component.		
Description	Prebuild Data Set Generation Phase for a software component: It binds all variations which need to be set after generation of the RTE contract header but before compilation of the component. The output is a configuration header which is used when compiling the component and the RTE as well. Meth.bindingTime = PreCompileTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Developer	1	
Consumes	Software Component Internal Behavior	1	Meth.bindingTime = CodeGenerationTime

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	VFB Atomic Software Component	1	Meth.bindingTime = CodeGenerationTime
Consumes	System Constant Value Set	1..*	Meth.bindingTime = CodeGenerationTime
Consumes	VFB AUTOSAR Standard Package	0..1	
Consumes	VFB Data Type Mapping Set	0..1	Meth.bindingTime = CodeGenerationTime
Consumes	Predefined Variant	0..*	
Consumes	VFB Interfaces	0..*	Meth.bindingTime = CodeGenerationTime
Consumes	VFB Modes	0..*	Meth.bindingTime = CodeGenerationTime
Consumes	VFB Types	0..*	Meth.bindingTime = CodeGenerationTime
Produces	Component RTE Prebuild Configuration Header	1	Meth.bindingTime = PreCompileTime
Used tool	Component API Generator Tool	1	

Table 3.192: Generate Component Prebuild Data Set

3.4.1.9 Implement Atomic Software Component

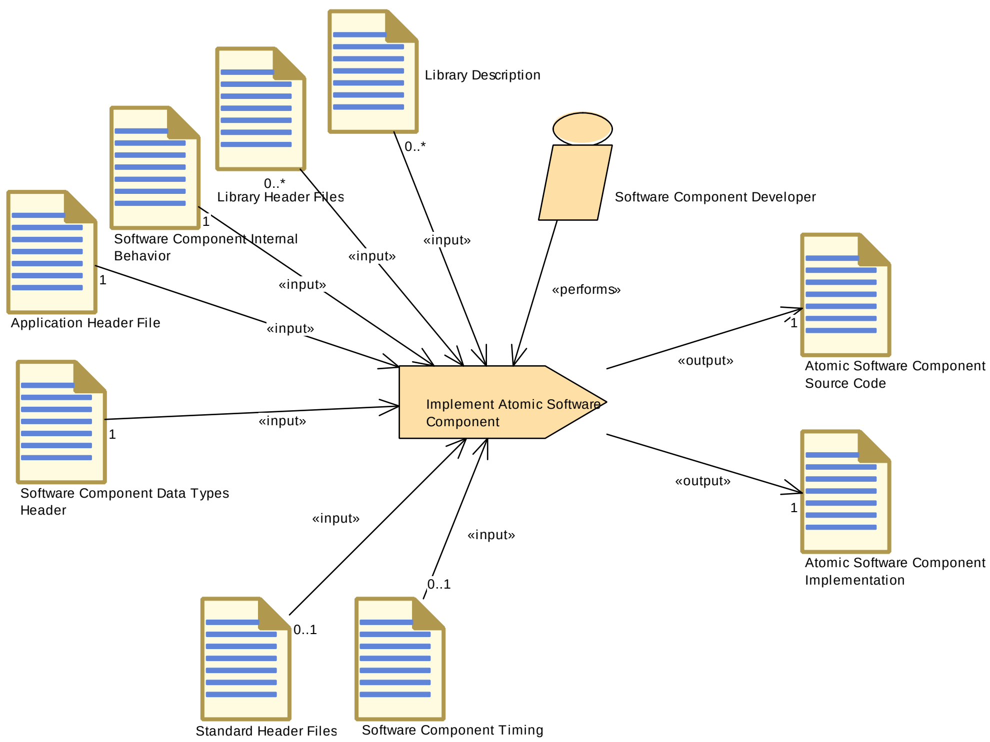

Figure 3.98: Implement Atomic Software Component

Task Definition	**Implement Atomic Software Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description	Implement the code of the AtomicSoftwareComponent and decribe the Implementation.		
Description	Implement the code of the AtomicSoftwareComponent against the generated component contract header. Document the basic information in the Implementation Description. Meth.bindingTime = CodeGenerationTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Developer	1	
Consumes	Application Header File	1	Meth.bindingTime = SystemDesignTime
Consumes	Software Component Data Types Header	1	Meth.bindingTime = SystemDesignTime
Consumes	Software Component Internal Behavior	1	Meth.bindingTime = SystemDesignTime
Consumes	Software Component Timing	0..1	Meth.bindingTime = SystemDesignTime
Consumes	Standard Header Files	0..1	Meth.bindingTime = CodeGenerationTime
Consumes	Library Description	0..*	Meth.bindingTime = CodeGenerationTime
Consumes	Library Header Files	0..*	Meth.bindingTime = CodeGenerationTime
Produces	Atomic Software Component Implementation	1	Meth.bindingTime = CodeGenerationTime
Produces	Atomic Software Component Source Code	1	Meth.bindingTime = CodeGenerationTime

Table 3.193: Implement Atomic Software Component

3.4.1.10 Compile Atomic Software Component

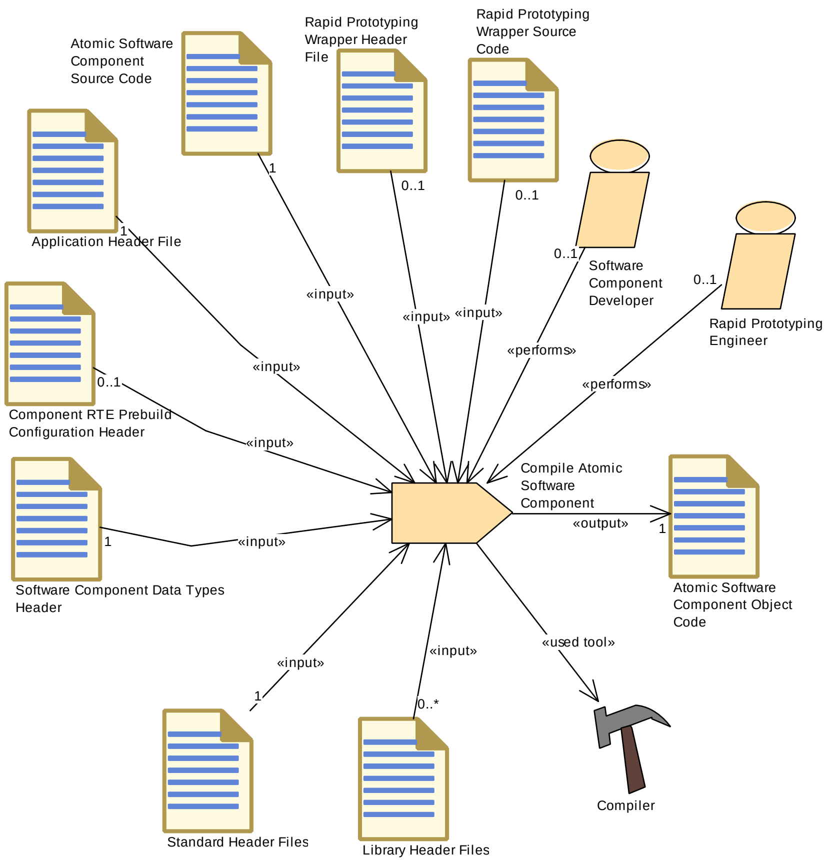

Figure 3.99: Compile Atomic Software Component

Task Definition	**Compile Atomic Software Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description	Compile the AtomicSoftwareComponent independently of an ECU.		
Description	Compile the Atomic Software Component independently of an ECU. In the context of Rapid Prototyping Wrapper compilation the task is performed by the Rapid Prototyping Engineer. Meth.bindingTime = CompileTime		
Relation Type	**_Related Element_**	**_Mul._**	**_Note_**
Performed by	Rapid Prototyping Engineer	0..1	
Performed by	Software Component Developer	0..1	
Consumes	Application Header File	1	Meth.bindingTime = CodeGenerationTime
Consumes	Atomic Software Component Source Code	1	Meth.bindingTime = CodeGenerationTime

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Software Component Data Types Header	1	Meth.bindingTime = CodeGenerationTime
Consumes	Standard Header Files	1	Meth.bindingTime = CodeGenerationTime
Consumes	Component RTE Prebuild Configuration Header	0..1	Meth.bindingTime = PreCompileTime
Consumes	Rapid Prototyping Wrapper Header File	0..1	
Consumes	Rapid Prototyping Wrapper Source Code	0..1	
Consumes	Library Header Files	0..*	Meth.bindingTime = CodeGenerationTime
Produces	Atomic Software Component Object Code	1	The object file should include both code of the SWC and the E2E Protection Wrapper code (if present as an input). Meth.bindingTime = CompileTime
Used tool	Compiler	1	

Table 3.194: Compile Atomic Software Component

3.4.1.11 Map Software Component to BSW

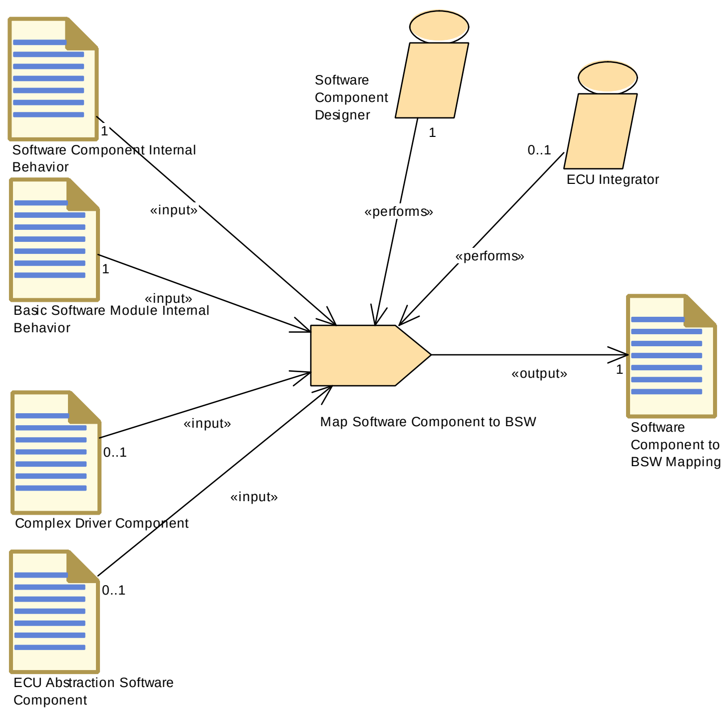

Figure 3.100: Map Software Component to BSW

Task Definition	**Map Software Component to BSW**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description	Define the mapping between a Software Component and a BSW Module.		
Description	Define the mapping between a Software Component and a BSW Module. Required only for Complex Drivers and ECU Abstraction Components. Note that for Service Components, this mapping will be generated in the ECU integration phase, so the latter is not considered as a task in the responsibility of the BSW developer.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	Software Component Designer	1	
Performed by	ECU Integrator	0..1	
Consumes	Basic Software Module Internal Behavior	1	
Consumes	Software Component Internal Behavior	1	
Consumes	Complex Driver Component	0..1	
Consumes	ECU Abstraction Software Component	0..1	
Produces	Software Component to BSW Mapping	1	

Table 3.195: Map Software Component to BSW

3.4.1.12 Measure Component Resources

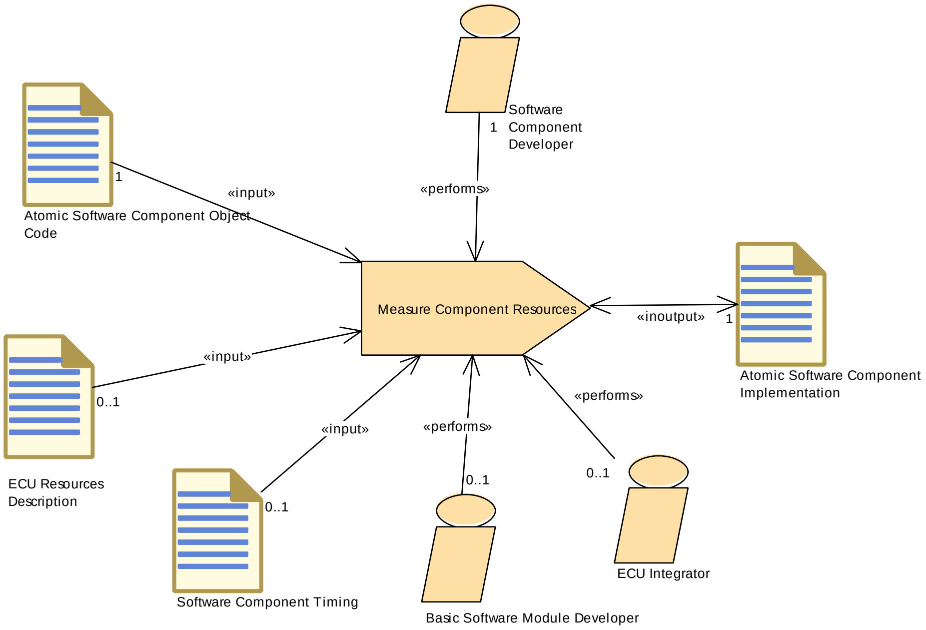

Figure 3.101: Measure Component Resources

Task Definition	**Measure Component Resources**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description	Measure the resource consumption of an Atomic Software Component		
Description	Determine the resource consumption (memory, execution time) for a specific implementation of an Atomic Software Component in a certain context (ECU or test environment) and document the results in the Implementation description targeted at this specific platform. The ECU Resources Description is an optional input, because some results should be documented in relation to the hardware elements.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Developer	1	
Performed by	Basic Software Module Developer	0..1	
Performed by	ECU Integrator	0..1	
Consumes	Atomic Software Component Object Code	1	
Consumes	ECU Resources Description	0..1	
Consumes	Software Component Timing	0..1	
In/out	Atomic Software Component Implementation	1	

Relation Type	***Related Element***	***Mul.***	***Note***

Table 3.196: Measure Component Resources

3.4.1.13 Recompile Component in ECU Context

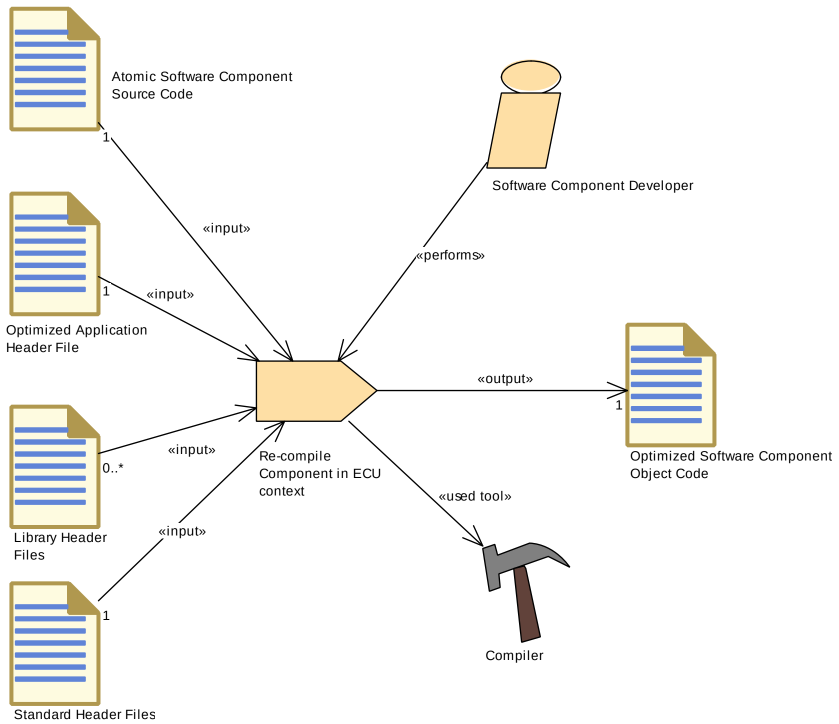

Figure 3.102: Recompile Component in ECU Context

Task Definition	**Re-compile Component in ECU context**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description	Re-compile Component with ECU-Configuration specific optimizations.		
Description	Re-compile Component with optimizations made by the RTE in the context of an ECU (so-called RTE implementation phase). Meth.bindingTime = CompileTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Software Component Developer	1	
Consumes	Atomic Software Component Source Code	1	Meth.bindingTime = CodeGenerationTime
Consumes	Optimized Application Header File	1	Meth.bindingTime = CodeGenerationTime
Consumes	Standard Header Files	1	Meth.bindingTime = CodeGenerationTime
Consumes	Library Header Files	0..*	Meth.bindingTime = CodeGenerationTime

Relation Type	*Related Element*	*Mul.*	*Note*
Produces	Optimized Software Component Object Code	1	Meth.bindingTime = CompileTime
Used tool	Compiler	1	

Table 3.197: Re-compile Component in ECU context

3.4.1.14 Define Consistency Needs

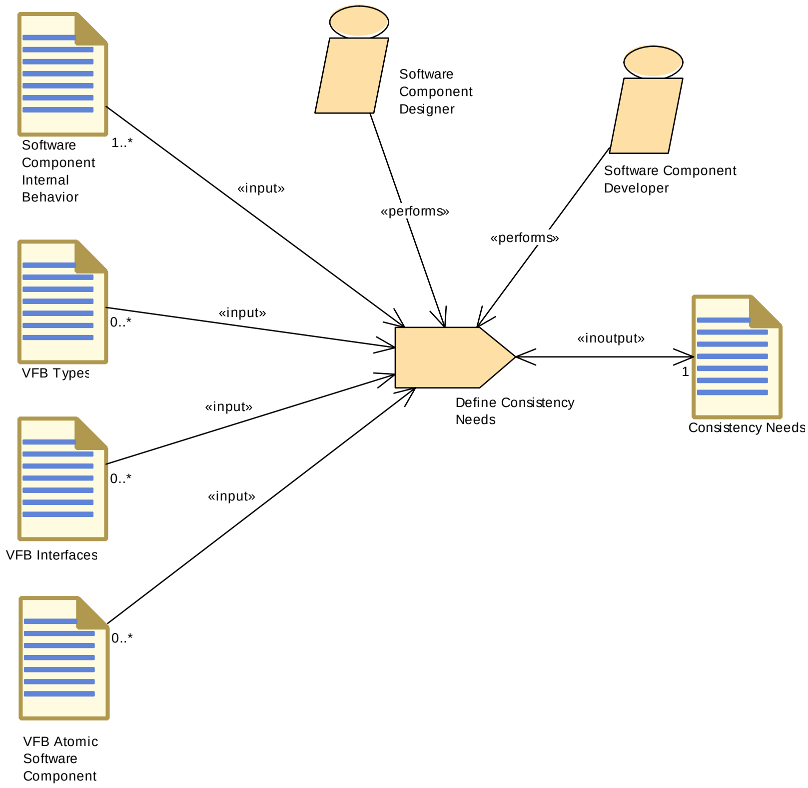

Figure 3.103: Define Consistency Needs

Task Definition	**Define Consistency Needs**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description			
Description	Defines the consistency relations between a group of RunnableEntitys and a group of DataPrototypes. The consistency relations can be defined first time at the design of an Atomic Software Component but can be added as well if Compositions are created.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	Software Component Designer	1	
Performed by	Software Component Developer	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Software Component Internal Behavior	1..*	Runnables the consistency is defined for.
Consumes	VFB Atomic Software Component	0..*	The description of an AtomicSoftwareComponentType without InternalBehavior.
Consumes	VFB Interfaces	0..*	Interfaces which are relevant for the consistency definition.
Consumes	VFB Types	0..*	Data types which are relevant for the consistency definition.
In/out	Consistency Needs	1	The description of the correlation between a group of RunnableEntitys and a group of DataPrototypes. In order to allow incremental development and refinement the Consistency Needs artifact is also used as an input.

Table 3.198: Define Consistency Needs

3.4.1.15 Generate Rapid Prototyping Wrapper

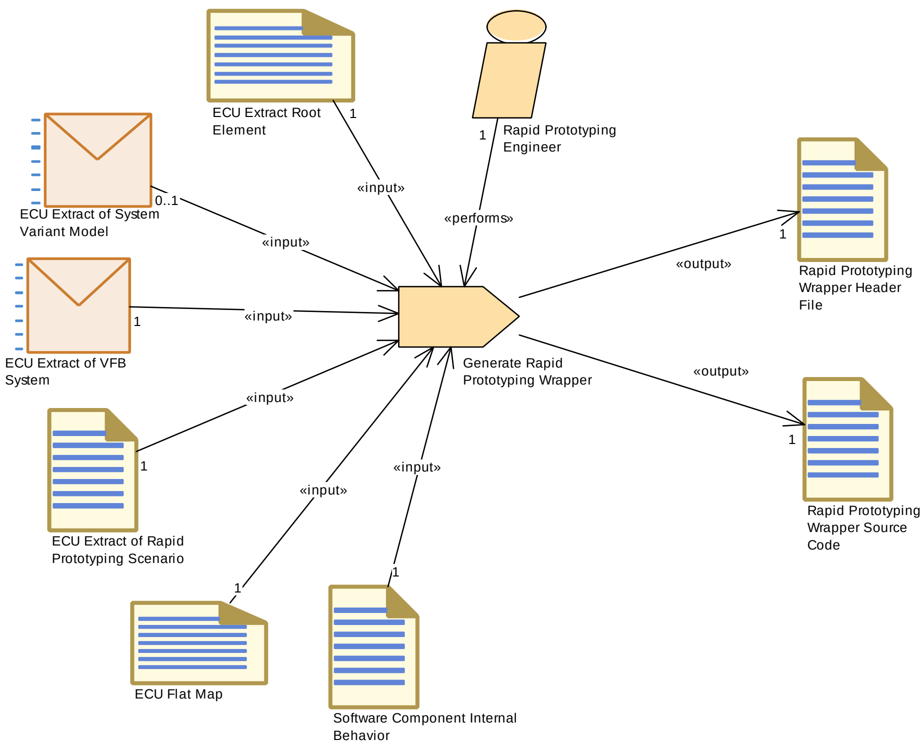

Figure 3.104: Generate Rapid Prototyping Wrapper

Task Definition	**Generate Rapid Prototyping Wrapper**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Tasks		
Brief Description	Generate Rapid Prototyping Wrapper code.		
Description	Generate Rapid Prototyping Wrapper code. The header and source code are generated based on the Rapid Prototyping Scenario describing the bypass points and the RPT hooks.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Rapid Prototyping Engineer	1	
Consumes	ECU Extract Root Element	1	
Consumes	ECU Extract of Rapid Prototyping Scenario	1	
Consumes	ECU Extract of VFB System	1	
Consumes	ECU Flat Map	1	
Consumes	Software Component Internal Behavior	1	
Consumes	ECU Extract of System Variant Model	0..1	
Produces	Rapid Prototyping Wrapper Header File	1	
Produces	Rapid Prototyping Wrapper Source Code	1	

Table 3.199: Generate Rapid Prototyping Wrapper

3.4.2 Work Products

3.4.2.1 Delivered Atomic Software Components

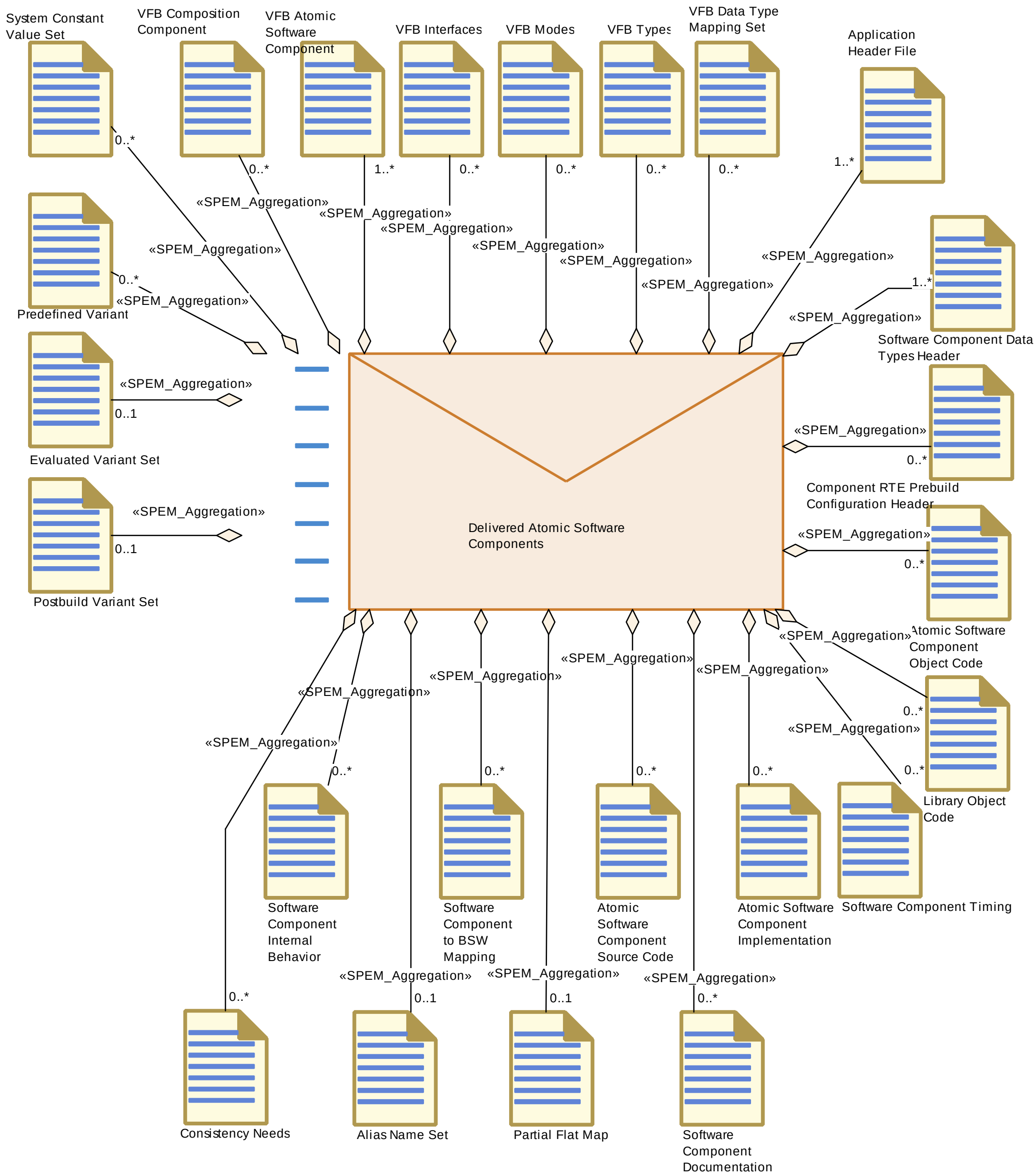

Figure 3.105: Delivered Atomic Software Components

Deliverable	**Delivered Atomic Software Components**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description	Delivery of a set of AtomicSoftwareComponents including their Implementation.		
Description	Complete description of a set of AtomicSoftwareComponents including Implementation (still standalone, not yet mapped to a specific ECU). The source or object code files are referred by the Implementation Description. The Atomic Software Components that make up the delivery may or may not form a composition (in the sense of the VFB). Note that the VFB descriptions of the components, compositions and the used interfaces are part of the deliverable too in order to describe the delivered components completely. However, depending on the use case, these parts could have been predefined and were treated as "readonly" during the component development. The same holds (optionally) for the Internal Behavior(s). In case of RTE generation a mapping set between Application and Implementation Data Types shall be included if Application Data Types are used. A Timing Model is included optionally. The delivery can optionally also contain variants (an Evaluated Variant Set and the related artifacts).		
Kind	Delivered		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Application Header File	1..*	
Aggregates	Software Component Data Types Header	1..*	
Aggregates	VFB Atomic Software Component	1..*	
Aggregates	Alias Name Set	0..1	Alias names valid in the context of the delivered components.
Aggregates	Evaluated Variant Set	0..1	
Aggregates	Partial Flat Map	0..1	
Aggregates	Postbuild Variant Set	0..1	
Aggregates	Atomic Software Component Implementation	0..*	If the delivery contains only VFB NvBlock Software Components, no implementation is contained as the code is generated as part of the RTE.
Aggregates	Atomic Software Component Object Code	0..*	
Aggregates	Atomic Software Component Source Code	0..*	

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Component RTE Prebuild Configuration Header	0..*	
Aggregates	Consistency Needs	0..*	Correlation between a group of RunnableEntitys and a group of DataPrototypes.
Aggregates	Library Object Code	0..*	
Aggregates	Predefined Variant	0..*	
Aggregates	Software Component Documentation	0..*	
Aggregates	Software Component Internal Behavior	0..*	If the delivery contains only VFB NvBlock Software Components, the Internal Behavior is optional since it is needed only in special cases.
Aggregates	Software Component Timing	0..*	
Aggregates	Software Component to BSW Mapping	0..*	
Aggregates	System Constant Value Set	0..*	
Aggregates	VFB Composition Component	0..*	In case the delivered atomic components make up one or more VFB Compositions, the composition description(s) shall be included in the delivery.
Aggregates	VFB Data Type Mapping Set	0..*	
Aggregates	VFB Interfaces	0..*	
Aggregates	VFB Modes	0..*	
Aggregates	VFB Types	0..*	
Produced by	Develop Application Software	1..*	Complete description of a set of AtomicSoftwareComponents including implementation (incl. source or object code files)
Consumed by	Configure RTE	1..*	Required input: • References to all component implementation descriptions on this ECU • SwcInternalBehavior (for example to map the runnables to tasks) which was used in the contract phase of the software components on this ECU

Relation Type	**Related Element**	**Mul.**	**Note**
Consumed by	Generate RTE	1..*	Required input: • References to all component implementation descriptions on this ECU • SwcInternalBehavior which was used in the contract phase of the software components on this ECU • (optional) Software Component to BSW Mapping Meth.bindingTime = SystemDesignTime
Consumed by	Integrate Software for ECU	1..*	
Consumed by	Define Alias Names	0..1	Needed for definition of alias names in the scope of delivered software components.
Consumed by	Create MC Function Model	0..*	The component model may be used to derive an MC Function Model.

Table 3.200: Delivered Atomic Software Components

3.4.2.2 Software Component Internal Behavior

Artifact	**Software Component Internal Behavior**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description	Description of the InternalBehavor: It describes the RTE relevant aspects of a component, for example the runnable entities and the events they respond to.		
Description	Description of the Internal Behavor. The Internal Behavior of an Atomic Software Component describes the RTE relevant aspects of a component, i.e. the runnable entities and the events they respond to. It is used to generate the RTE but also as input for parts of the basic software generation (AUTOSAR Services). The Internal Behavior (i.e. the XML description) can only be used together with an Atomic Software Component Type to which it is related.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	Delivered Atomic Software Components	0..*	If the delivery contains only VFB NvBlock Software Components, the Internal Behavior is optional since it is needed only in special cases.
Produced by	Define Atomic Software Component Internal Behavior	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Consumed by	Define Software Component Safety Information	1	
Consumed by	Define Software Component Timing	1	
Consumed by	Generate Atomic Software Component Contract Header Files	1	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Component Header File in Vendor Mode	1	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Component Prebuild Data Set	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Generate Rapid Prototyping Wrapper	1	
Consumed by	Implement Atomic Software Component	1	Meth.bindingTime = SystemDesignTime
Consumed by	Map Software Component to BSW	1	
Consumed by	Refine Rapid Prototyping Scenario	1	
Consumed by	Define Consistency Needs	1..*	Runnables the consistency is defined for.
Consumed by	Define Rapid Prototyping Scenario	1..*	
Consumed by	Select Software Component Implementation	1..*	
Consumed by	Generate Local MC Data Support	0..1	Meth.bindingTime = SystemDesignTime
Consumed by	Define Partial Flat Map	0..*	Refer to parameter and variables defined in the Internal Behavior of one or more Atomic Software Components.
Consumed by	Define VFB Nv Block Software Component	0..*	This input is required to collect the requirements for the NvBlockNeeds from the using application software.
Use meta model element	SwcInternalBehavior	1	

Table 3.201: Software Component Internal Behavior

3.4.2.3 Atomic Software Component Implementation

Artifact	**Atomic Software Component Implementation**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description	Description of an implementation for a single Atomic Software Component.		
Description	Description of an implementation for a single Atomic Software Component. It is possible to have several different implementations for the same Software Component Internal Behavior, but only one implementation can be mapped to a particular ECU. In general, this XML artifact relates to one particular version of the code. It contains the version information as defined by the vendor. An implementation description may depend on several non-AUTOSAR artifacts, especially its own code files (source or object) but also required libraries, generator tools etc. These dependencies are not described by direct references to files (because this might be ambiguous), but by referring entries in the container catalog of the General Deliverable which contains the implementation artifacts. Such a reference is described via the metamodel element AutosarEngineeringObject (see AUTOSAR_TPS_GenericStructureTemplate.pdf for further description). This allows among other things to refer to a particular version of an artifact. For more information on the content of the implmementation description refer to AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Delivered Atomic Software Components	0..*	If the delivery contains only VFB NvBlock Software Components, no implementation is contained as the code is generated as part of the RTE.
Produced by	Create Service Component	1	In order to generate the RTE, one needs to create a kind of dummy Implementation element for the Service Component, however this should not be filled with descriptive elements, e.g. resource consumption, as these are already defined by the Basic Software Module Implementation Description. Meth.bindingTime = SystemDesignTime
Produced by	Implement Atomic Software Component	1	Meth.bindingTime = CodeGenerationTime
Produced by	Measure Resources	0..*	Add extensions to the Implementation Description. Meth.bindingTime = PostBuild
In/out	Measure Component Resources	1	
Consumed by	Generate Component Header File in Vendor Mode	1	Meth.bindingTime = SystemDesignTime

Relation Type	Related Element	Mul.	Note
Consumed by	Generate SWC Memory Mapping Header	1	MemorySections: MemorySections defined for an Atomic Software Component. Meth.bindingTime = SystemDesignTime
Consumed by	Select Software Component Implementation	1..*	
Consumed by	Configure Memmap Allocation	0..*	MemorySections:
Consumed by	Generate Compiler Configuration	0..*	MemorySections: Find referred SwAddrMethods or specific memClassSymbols in the MemorySections defined for Atomic Software Components. Meth.bindingTime = SystemDesignTime
Use meta model element	Implementation	1	

Table 3.202: Atomic Software Component Implementation

3.4.2.4 Software Component Documentation

Artifact	**Software Component Documentation**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description	Documentation dedicated to a Software Component.		
Description	Documentation of a dedicated Software Component. This documentation is following the ASAM FSX standard. In this documentation, you will find the SW Feature definition and description which define the physical functionality of the Swc, the SW test description which will contains suggestions and hints for the test of the software functionality of the Swc, the SW calibration notes which will give calibration instructions and hints for a calibration engineer, some maintenance, diagnosis and CARB notes which will bring general information, on the maintenance diagnosis and CARB issues on the Swc. For other description not listed previously, some notes (chapters) are left free for that. This artifact may also contain standalone documentation (meta-class Documentation) not aggregeted by a specific software component.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Delivered Atomic Software Components	0..*	
In/out	Add Documentation to the Software Component	1	
Use meta model element	Documentation	1	
Use meta model element	SwComponent Documentation	1	

Relation Type	***Related Element***	***Mul.***	***Note***

Table 3.203: Software Component Documentation

3.4.2.5 Software Component Timing

Artifact	**Software Component Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description	Software Component's TimingDescription and TimingConstraints		
Description	TimingDescription and TimingConstraints of a software component. A software component can either be of type AtomicSWComponentType or CompositionSWComponentType. In the former case, the SwcTiming allows to describe timing description and constraints for the InternalBehavior of the AtomicSWComponentType. In the latter case, timing descriptions and constraints can be defined for all Atomic Software Components in the CompositionSWComponentType.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Delivered Atomic Software Components	0..*	
Produced by	Define Software Component Timing	1	
Consumed by	Define System Timing	0..1	
Consumed by	Implement Atomic Software Component	0..1	Meth.bindingTime = SystemDesignTime
Consumed by	Measure Component Resources	0..1	
Use meta model element	SwcTiming	1	

Table 3.204: Software Component Timing

3.4.2.6 Software Component to BSW Mapping

Artifact	**Software Component to BSW Mapping**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description	Desribes how to map a software component to basic software elements (required in special cases only).		
Description	Maps an SwcInternalBehavior to an BswInternalBehavior. This is required to coordinate the API generation and the scheduling for AUTOSAR Service Components, ECU Abstraction Components and Complex Driver Components by the RTE and the BSW scheduling mechanisms.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	Delivered Atomic Software Components	0..*	
Produced by	Map Software Component to BSW	1	
Produced by	Create Service Component	0..1	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Atomic Software Component Contract Header Files	0..1	If a Software Component is mapped to a BSW module description, this input is optionally needed already in the contract phase in order to ensure that the generated prototypes for runnables are consistent with the definitions in Software Component and BSW. Meth.bindingTime = SystemDesignTime
Consumed by	Generate RTE	0..*	This input is explicitly stated because the mapping may be created during ECU integration and thus is not necessarily part of the Delivered Atomic Software Components. Meth.bindingTime = SystemDesignTime
Use meta model element	SwcBswMapping	1	

Table 3.205: Software Component to BSW Mapping

3.4.2.7 Partial Flat Map

Artifact	**Partial Flat Map**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description			
Description	The Partial Flat Map pre-defines Flat Map entries in the context of delivered software components. This allows the component developer to specify names of data instances for measurement and calibration. It has to be integrated into the System Flat Map. For more information on the Flat Map concept refer to artifact System Flat Map in the system domain.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Delivered Atomic Software Components	0..1	
Produced by	Define Partial Flat Map	1	
Consumed by	Add Documentation to the Software Component	0..1	Optional input in order to refer to unique names defined in component or composition context.
Consumed by	Generate or Adjust ECU Flat Map	0..*	If Partial Flat Maps were delivered along with software components referring only to ECU internal information, they may be integrated into the ECU Flat Map directly, i.e. without needing the System Flat Map. • The instance refs used in a partial flat map must be taken over and adjusted to the context ECU Extract. • Name conflicts have to be resolved if several partial flat maps are merged.
Consumed by	Generate or Adjust System Flat Map	0..*	If Partial Flat Maps were delivered along with software components, they must be integrated into the System Flat Map: • The instance refs used in a partial flat map must be taken over and adjusted to the context of the System or System Extract. • Name conflicts have to be resolved if several partial flat maps are merged.
Use meta model element	FlatMap	1	

Table 3.206: Partial Flat Map

3.4.2.8 Application Header File

Artifact	**Application Header File**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description	Header generated for an AtomicSoftwareComponentType in the RTE contract phase.		
Description	Header generated for an AtomicSoftwareComponentType in the RTE contract phase. It represents the complete source-code interface between the component code and RTE (calls into the RTE as well as prototypes called by the RTE). All communication of the component code with other components is routed through this header.		
Kind	Source Code		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	Delivered Atomic Software Components	1..*	
Produced by	Generate Atomic Software Component Contract Header Files	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile Atomic Software Component	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Implement Atomic Software Component	1	Meth.bindingTime = SystemDesignTime
Consumed by	Compile ECU Source Code	1..*	Meth.bindingTime = CodeGenerationTime

Table 3.207: Application Header File

3.4.2.9 Software Component Data Types Header

Artifact	**Software Component Data Types Header**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description	Software Component Data Types Header provided by the RTE in the contract phase.		
Description	Software Component Data Types Header provided by the RTE in the contract phase. This includes data types, which were declared as part of the SWC description but not used in any ports or data elements.		
Kind	Source Code		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	Delivered Atomic Software Components	1..*	

Relation Type	Related Element	Mul.	Note
Produced by	Generate Atomic Software Component Contract Header Files	1	Meth.bindingTime = CodeGenerationTime
Produced by	Generate Component Header File in Vendor Mode	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile Atomic Software Component	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Implement Atomic Software Component	1	Meth.bindingTime = SystemDesignTime
Consumed by	Compile ECU Source Code	0..*	Meth.bindingTime = CodeGenerationTime

Table 3.208: Software Component Data Types Header

3.4.2.10 Component RTE Prebuild Configuration Header

Artifact	**Component RTE Prebuild Configuration Header**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description	Generated header file used to resolve the prebuild variants in the prebuild RTE contract phase for an SWC.		
Description	Generated header file used to resolve the prebuild variants of a software component in the prebuild RTE contract phase. Contains macros which resolve the variants when compiled with the module and the generated RTE.		
Kind	Bound Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Delivered Atomic Software Components	0..*	
Produced by	Generate Component Prebuild Data Set	1	Meth.bindingTime = PreCompileTime
Consumed by	Compile Atomic Software Component	0..1	Meth.bindingTime = PreCompileTime
Consumed by	Compile ECU Source Code	0..*	Meth.bindingTime = CodeGenerationTime

Table 3.209: Component RTE Prebuild Configuration Header

3.4.2.11 Atomic Software Component Source Code

Artifact	**Atomic Software Component Source Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description	Source code implementing an Atomic Software Component Type		
Description	Source code implementing an Atomic Software Component Type. In general it is independent from an ECU.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Delivered Atomic Software Components	0..*	
Produced by	Implement Atomic Software Component	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile Atomic Software Component	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Re-compile Component in ECU context	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile ECU Source Code	0..*	Meth.bindingTime = CodeGenerationTime

Table 3.210: Atomic Software Component Source Code

3.4.2.12 Atomic Software Component Object Code

Artifact	**Atomic Software Component Object Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description			
Description	Object Code of an Atomic Software Component.		
Kind	Object Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Delivered Atomic Software Components	0..*	
Produced by	Compile Atomic Software Component	1	The object file should include both code of the SWC and the E2E Protection Wrapper code (if present as an input). Meth.bindingTime = CompileTime
Consumed by	Measure Component Resources	1	
Consumed by	Generate ECU Executable	0..*	Meth.bindingTime = CompileTime

Table 3.211: Atomic Software Component Object Code

3.4.2.13 Optimized Application Header File

Artifact	**Optimized Application Header File**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description	Optimized application header file for a software component.		
Description	Application header file for a software component optimized by the RTE in vendor mode.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Generate Component Header File in Vendor Mode	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Re-compile Component in ECU context	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile ECU Source Code	0..*	Meth.bindingTime = CodeGenerationTime

Table 3.212: Optimized Application Header File

3.4.2.14 Optimized Software Component Object Code

Artifact	**Optimized Software Component Object Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description	The object code of a software component compiled with ECU specific optimizations.		
Description	The object code of a software component compiled with ECU specific optimizations.		
Kind	Object Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Re-compile Component in ECU context	1	Meth.bindingTime = CompileTime

Table 3.213: Optimized Software Component Object Code

3.4.2.15 Consistency Needs

Artifact	**Consistency Needs**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description			
Description	A ConsistencyNeed describes the correlation between a group of RunnableEntitys and a group of DataPrototypes with the intended purpose to describe the need for • Stable data during the execution of a group of RunnableEntitys. • Coherent data consumption and propagation for a group of DataPrototypes. The information can be defined first time at the design of an Atomic Software Component but can be added as well if Compositions are created. In order to allow incremental development the groups of Runnables and DataPrototypes can be distributed over several artifacts.		
Kind			
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	VFB System	1	Correlation between a group of RunnableEntitys and a group of DataPrototypes.
Aggregated by	Delivered Atomic Software Components	0..*	Correlation between a group of RunnableEntitys and a group of DataPrototypes.
In/out	Define Consistency Needs	1	The description of the correlation between a group of RunnableEntitys and a group of DataPrototypes. In order to allow incremental development and refinement the Consistency Needs artifact is also used as an input.
Use meta model element	ConsistencyNeeds	1	

Table 3.214: Consistency Needs

3.4.2.16 Rapid Prototyping Wrapper Header File

Artifact	**Rapid Prototyping Wrapper Header File**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description			
Description	This header replaces the RTE API in order to allow to read and modify inputs and outputs of the original SWC as well as to control execution of the original (and prototype) runnable.		
Kind	Source Code		
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Generate Rapid Prototyping Wrapper	1	

Relation Type	Related Element	Mul.	Note
Consumed by	Compile Atomic Software Component	0..1	

Table 3.215: Rapid Prototyping Wrapper Header File

3.4.2.17 Rapid Prototyping Wrapper Source Code

Artifact	**Rapid Prototyping Wrapper Source Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Work Products		
Brief Description			
Description	A piece of code that is placed between software components and the RTE in order to provide rapid prototyping functionality. This code allows to encapsulate the SWC to bypass into the rapid prototyping component and may be implemented ad as a complex device driver and/or integration code.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Generate Rapid Prototyping Wrapper	1	
Consumed by	Compile Atomic Software Component	0..1	

Table 3.216: Rapid Prototyping Wrapper Source Code

3.4.3 Tools

3.4.3.1 Component API Generator Tool

Tool	**Component API Generator Tool**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Component::Guidance		
Brief Description	Generates the software component contract header used to connect the software component to the RTE layer.		
Description	This guidance represents the so-called contract phase of the RTE generation process. • SWC Contract phase - a limited set of information about a component, principally the AUTOSAR Interface definitions and the internal behavior, is used to create an application header file for a component type. The application header file defines the "contract" between component and RTE. • BSW Contract phase - a similar use case for a BSW module in order to generate the module interlink header files, which are used to interface between the module and the BSW Scheduler. • Additional phases - for SWS and BSW as well - are used to bind pre-build variants in the contract headers of a single Software Component or BSW module.		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Used	Generate Atomic Software Component Contract Header Files	1	
Used	Generate BSW Module Prebuild Data Set	1	
Used	Generate BSWM Contract Header Files	1	
Used	Generate Component Header File in Vendor Mode	1	
Used	Generate Component Prebuild Data Set	1	

Table 3.217: Component API Generator Tool

3.5 Basic Software

This chapter contains the definition of work products and tasks used for the development of Basic Software modules. For the definition of the relevant meta-model elements refer to [10].

3.5.1 Tasks

3.5.1.1 Define BSW Types

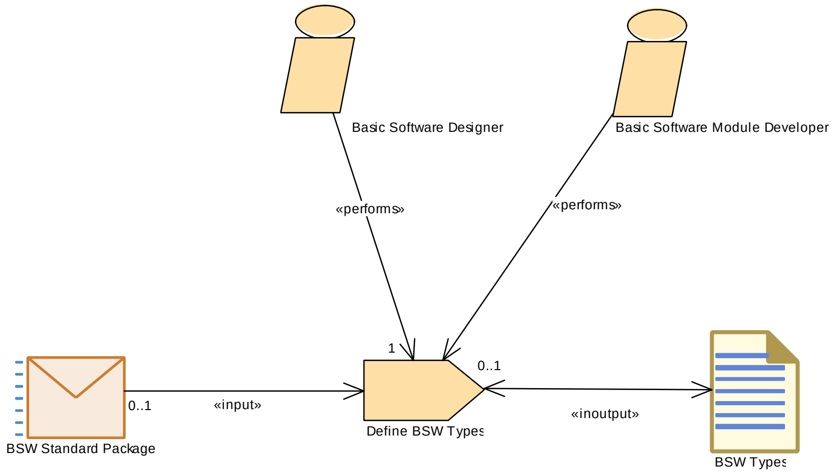

Figure 3.106: Define BSW Types

Task Definition	**Define BSW Types**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Tasks		
Brief Description	Define data types for usage within the Basic Software.		
Description	A data type is typically based on elements standardized by AUTOSAR, therefore BSW Standard Package appears as a mandatory input.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Basic Software Designer	1	
Performed by	Basic Software Module Developer	1	
Consumes	BSW Standard Package	0..1	
In/out	BSW Types	1	

Table 3.218: Define BSW Types

3.5.1.2 Define BSW Entries

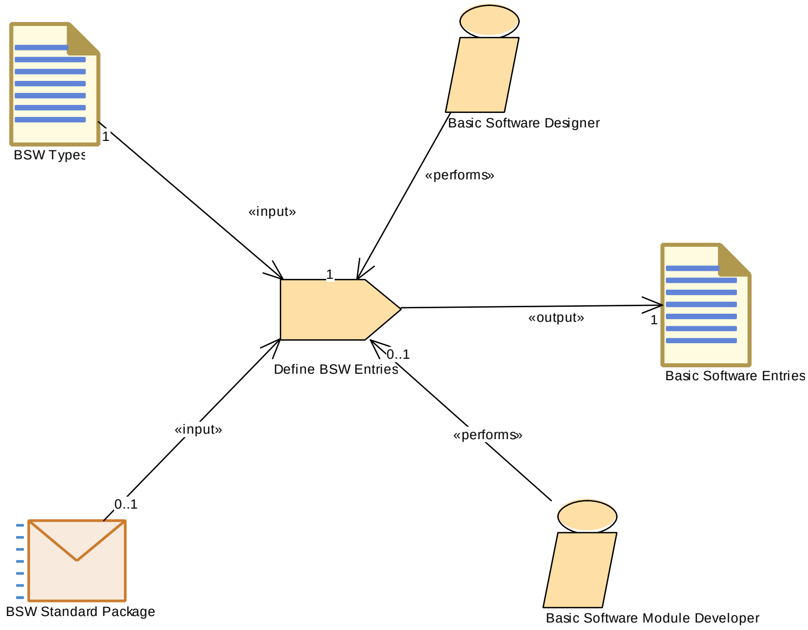

Figure 3.107: Define BSW Entries

Task Definition	**Define BSW Entries**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Tasks		
Brief Description	Define BswEntries (= function signatures) for usage within the Basic Software.		
Description			
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Basic Software Designer	1	
Performed by	Basic Software Module Developer	1	
Consumes	BSW Types	1	
Consumes	BSW Standard Package	0..1	
Produces	Basic Software Entries	1	

Table 3.219: Define BSW Entries

3.5.1.3 Define BSW Interfaces

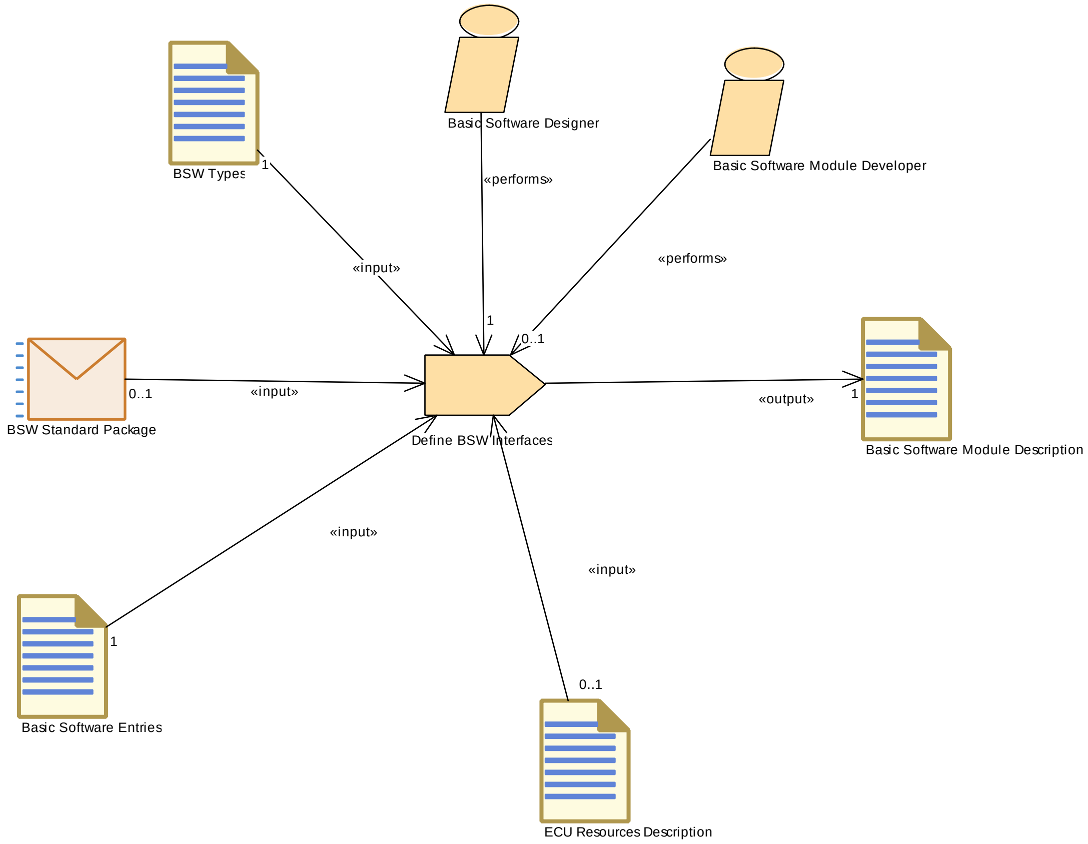

Figure 3.108: Define BSW Interfaces

Task Definition	**Define BSW Interfaces**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Tasks		
Brief Description	Define the interfaces for a single BSW Module.		
Description	Define the interfaces for a particular BSW Module or cluster as part of the BSW Module Description. This includes an abstraction of the required and provided C-functions, as well as triggers and modes. Note that this task also exists for modules standardized by AUTOSAR, as it may be required to decide on optional or alternative elements and to add allowed project specific extensions.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Basic Software Designer	1	
Performed by	Basic Software Module Developer	1	
Consumes	BSW Types	1	
Consumes	Basic Software Entries	1	
Consumes	BSW Standard Package	0..1	
Consumes	ECU Resources Description	0..1	

Relation Type	*Related Element*	*Mul.*	*Note*
Produces	Basic Software Module Description	1	

Table 3.220: Define BSW Interfaces

3.5.1.4 Define Vendor Specific Module Definition

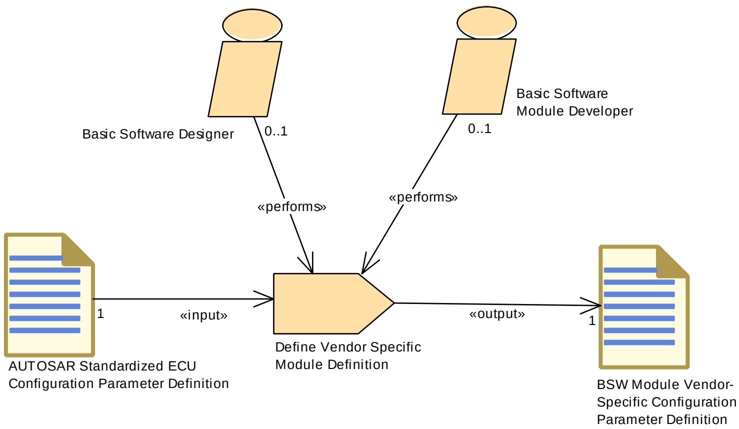

Figure 3.109: Define Vendor Specific Module Definition

Task Definition	**Define Vendor Specific Module Definition**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Tasks		
Brief Description			
Description	Define the Vendor Specific Module Definition (=Configuration Parameters).		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	Basic Software Designer	0..1	
Performed by	Basic Software Module Developer	0..1	
Consumes	AUTOSAR Standardized ECU Configuration Parameter Definition	1	
Produces	BSW Module Vendor-Specific Configuration Parameter Definition	1	

Table 3.221: Define Vendor Specific Module Definition

3.5.1.5 Define BSW Behavior

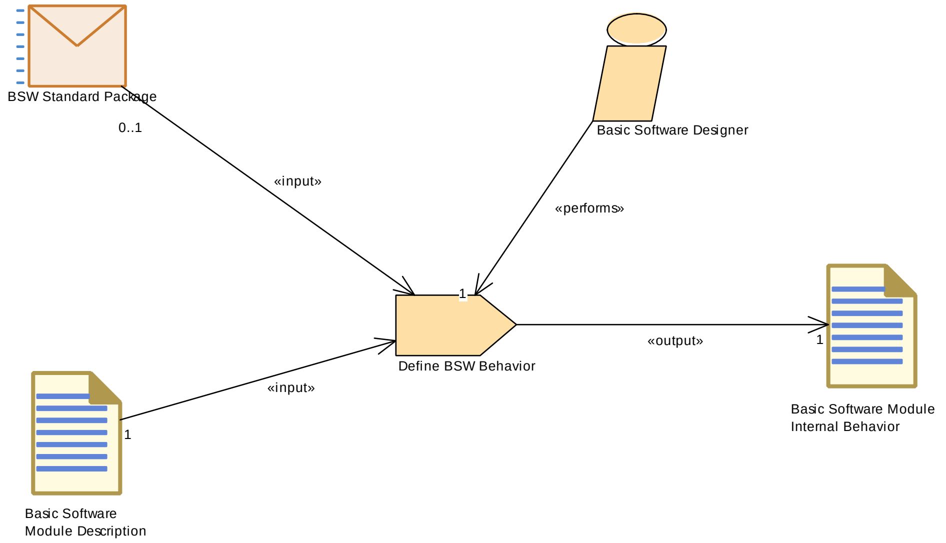

Figure 3.110: Define BSW Behavior

Task Definition	**Define BSW Behavior**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Tasks		
Brief Description	Define the BSW Behavior related to a BSW Module Description.		
Description	Define the BSW Behavior related to a BSW Module Description. This task is required during BSW module development in order to be able to generate the API to the BSW Scheduler. In addition, local data (variables or parameters) may be defined during this task in order to use the AUTOSAR data type system for module local data and to generate measurement & calibration support.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Basic Software Designer	1	
Consumes	Basic Software Module Description	1	
Consumes	BSW Standard Package	0..1	
Produces	Basic Software Module Internal Behavior	1	

Table 3.222: Define BSW Behavior

3.5.1.6 Define BSW Module Timing

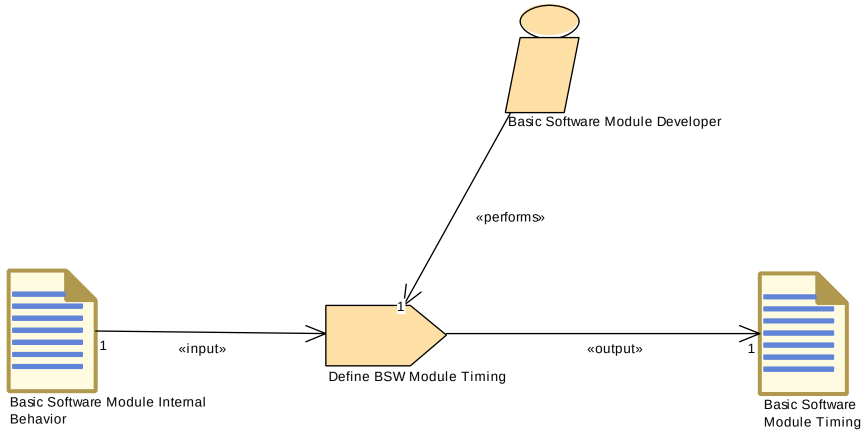

Figure 3.111: Define BSW Module Timing

Task Definition	**Define BSW Module Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Tasks		
Brief Description	Define BSWModuleTiming (TimingDescription and TimingConstraints) for the Internal Behavior (BSWModuleEntities) of a BSW module		
Description	Define BSWModuleTiming (TimingDescription and TimingConstraints) for the Internal Behavior (BSWModuleEntities) of a BSW module		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	Basic Software Module Developer	1	
Consumes	Basic Software Module Internal Behavior	1	
Produces	Basic Software Module Timing	1	

Table 3.223: Define BSW Module Timing

3.5.1.7 Generate BSW Contract Header Files

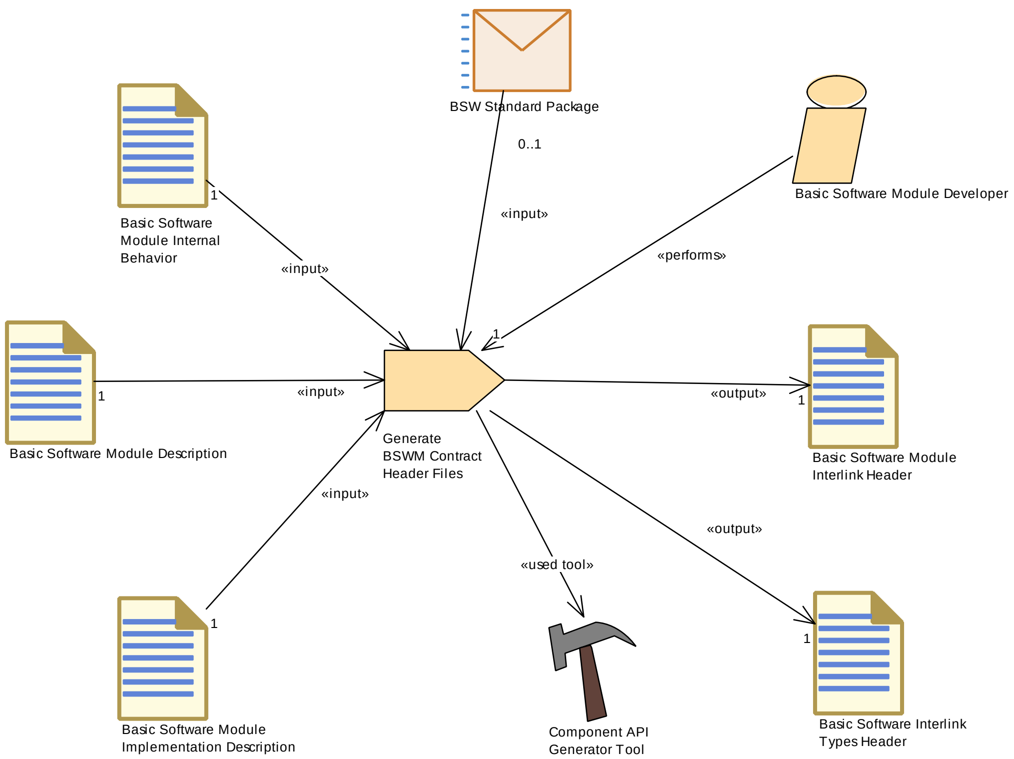

Figure 3.112: Generate BSW Contract Header Files

Task Definition	**Generate BSWM Contract Header Files**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Tasks		
Brief Description	Generate Basic Softwaree Module Contract Header Files		
Description	Generate the header files needed for a BSW module as part of the so-called "contract phase". These headers will allow to link the module lateron with the RTE (namely the BSW Scheduler). Meth.bindingTime = CodeGenerationTime		
Relation Type	**_Related Element_**	**_Mul._**	**_Note_**
Performed by	Basic Software Module Developer	1	
Consumes	Basic Software Module Description	1	Meth.bindingTime = SystemDesignTime
Consumes	Basic Software Module Implementation Description	1	Meth.bindingTime = SystemDesignTime
Consumes	Basic Software Module Internal Behavior	1	Meth.bindingTime = SystemDesignTime
Consumes	BSW Standard Package	0..1	

Relation Type	*Related Element*	*Mul.*	*Note*
Produces	Basic Software Interlink Types Header	1	Meth.bindingTime = CodeGenerationTime
Produces	Basic Software Module Interlink Header	1	Meth.bindingTime = CodeGenerationTime
Used tool	Component API Generator Tool	1	

Table 3.224: Generate BSWM Contract Header Files

3.5.1.8 Implement a BSW Module

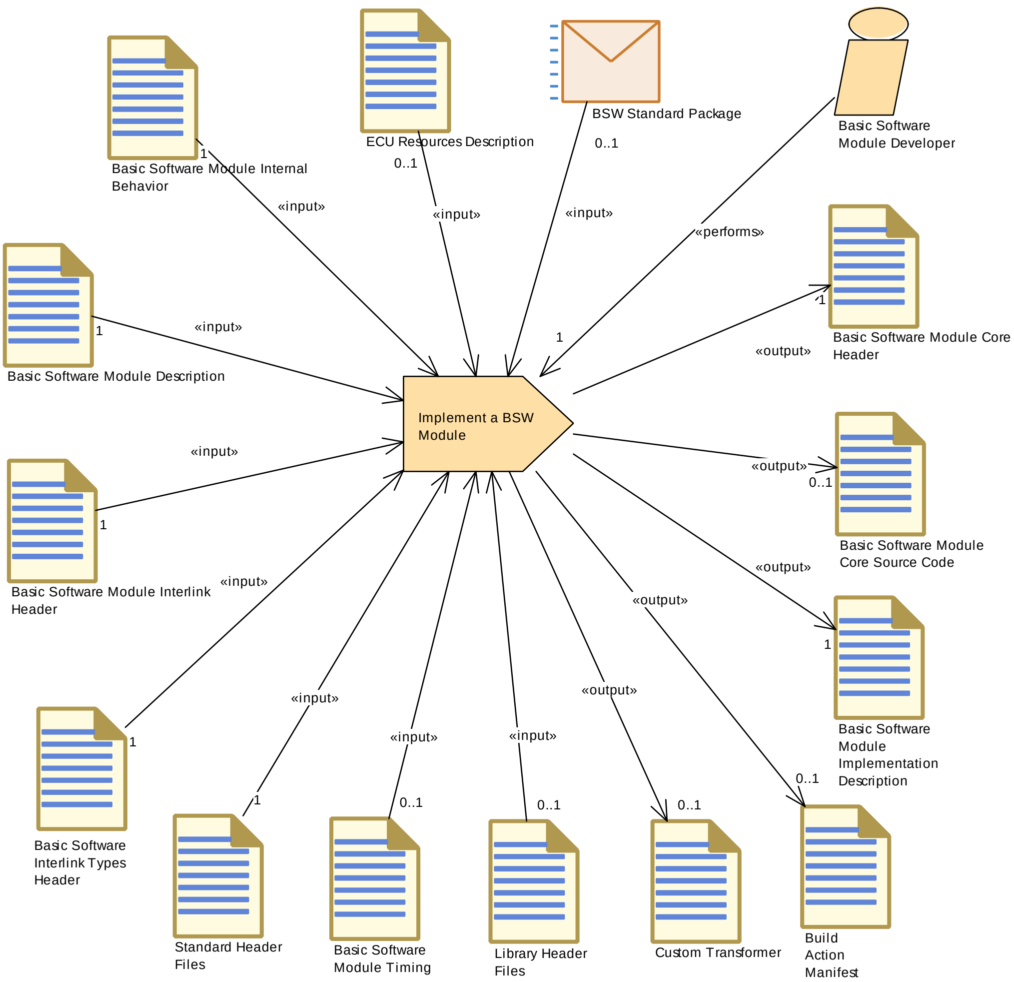

Figure 3.113: Implement a BSW Module

Task Definition	**Implement a BSW Module**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Tasks		
Brief Description	Implement the source code of a BSW module.		
Description	Implement the source code of a BSW module. This task is not described by AUTOSAR completely, but included for completeness of the AUTOSAR use cases. Note that specification of an AUTOSAR standard module imposes several requirements, e.g. the inclusion of certain header files, onto this task. In addition to the code, this task also produces the necessary XML descriptions. Optionally, a build action manifest may be created or modified in order to be used for code generation or further processing of the code. Meth.bindingTime = CodeGenerationTime		
Relation Type	**Related Element**	**Mul.**	**Note**
Performed by	Basic Software Module Developer	1	
Consumes	Basic Software Interlink Types Header	1	Meth.bindingTime = SystemDesignTime
Consumes	Basic Software Module Description	1	Meth.bindingTime = SystemDesignTime
Consumes	Basic Software Module Interlink Header	1	Meth.bindingTime = SystemDesignTime
Consumes	Basic Software Module Internal Behavior	1	Meth.bindingTime = SystemDesignTime
Consumes	Standard Header Files	1	Meth.bindingTime = CodeGenerationTime
Consumes	BSW Standard Package	0..1	
Consumes	Basic Software Module Timing	0..1	Meth.bindingTime = SystemDesignTime
Consumes	ECU Resources Description	0..1	Meth.bindingTime = SystemDesignTime
Consumes	Library Header Files	0..1	Meth.bindingTime = CodeGenerationTime
Produces	Basic Software Module Core Header	1	Meth.bindingTime = CodeGenerationTime
Produces	Basic Software Module Implementation Description	1	Meth.bindingTime = CodeGenerationTime

Relation Type	*Related Element*	*Mul.*	*Note*
Produces	Basic Software Module Core Source Code	0..1	The creation of source code is optional, since it might be generated completely in a later step based on the Build Action Manifest. Meth.bindingTime = CodeGenerationTime
Produces	Build Action Manifest	0..1	
Produces	Custom Transformer	0..1	

Table 3.225: Implement a BSW Module

3.5.1.9 Develop BSW Module Generator

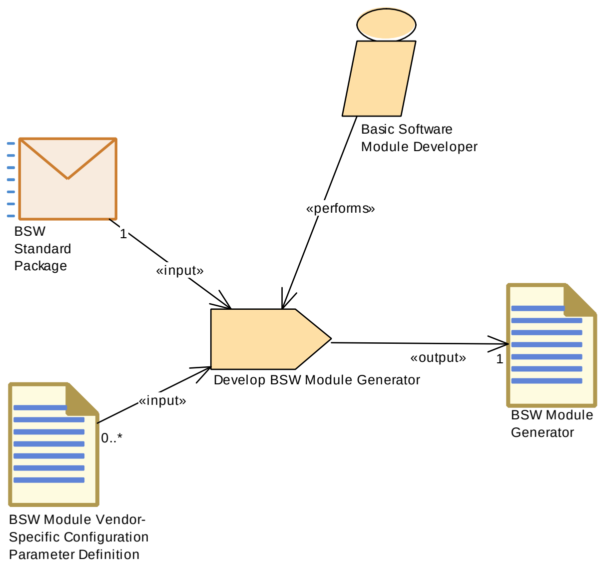

Figure 3.114: Develop BSW Module Generator

Task Definition	**Develop BSW Module Generator**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Tasks		
Brief Description			
Description	Develop a generator for one or more BSW modules.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	Basic Software Module Developer	1	
Consumes	BSW Standard Package	1	
Consumes	BSW Module Vendor-Specific Configuration Parameter Definition	0..*	

Relation Type	Related Element	Mul.	Note
Produces	BSW Module Generator	1	

Table 3.226: Develop BSW Module Generator

3.5.1.10 Create Library

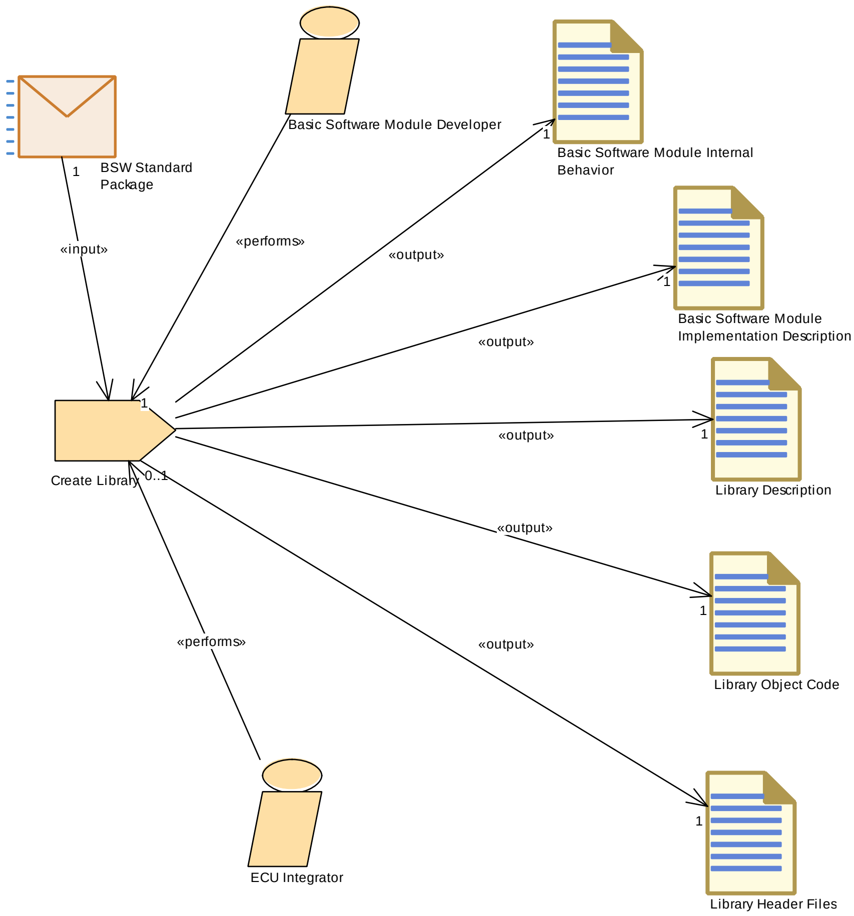

Figure 3.115: Create Library

Task Definition	**Create Library**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Tasks		
Brief Description	Create a library to be used within an Autosar ECU.		
Description	Create a non-standardized library to be used within an Autosar ECU. The task is the same for the basic software and application level, but it is considered as a basic software task because no VFB resp. RTE abstraction is used. The output includes source code, header file and XML descriptions of the interfaces and of the implementation. A "dummy" BSW Behavior must be created too in order to be able to link the other two XML artifacts. Meth.bindingTime = CodeGenerationTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Basic Software Module Developer	1	
Performed by	ECU Integrator	1	
Consumes	BSW Standard Package	1	Used for standard types and specifications.
Produces	Basic Software Module Implementation Description	1	Meth.bindingTime = CodeGenerationTime
Produces	Basic Software Module Internal Behavior	1	Meth.bindingTime = CodeGenerationTime
Produces	Library Description	1	Meth.bindingTime = CodeGenerationTime
Produces	Library Header Files	1	Meth.bindingTime = CodeGenerationTime
Produces	Library Object Code	1	Meth.bindingTime = CodeGenerationTime

Table 3.227: Create Library

3.5.1.11 Compile BSW Core Code

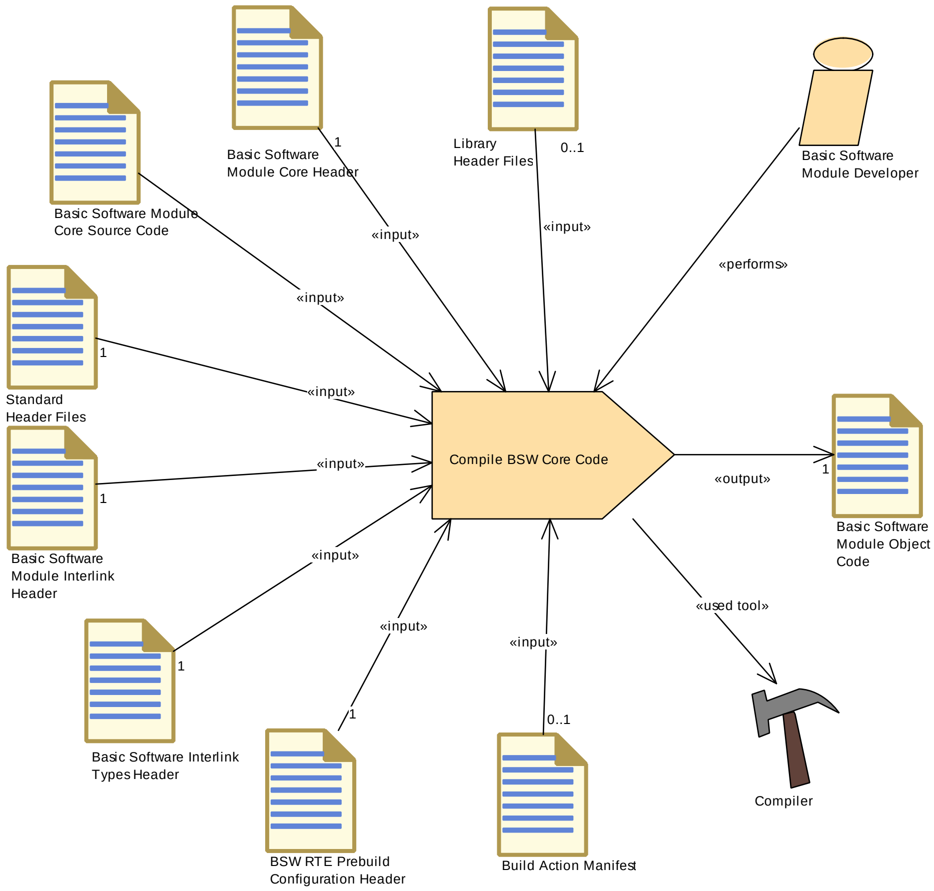

Figure 3.116: Compile BSW Core Code

Task Definition	**Compile BSW Core Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Tasks		
Brief Description	Compile the source code of a BSW modue without ECU specific configurations.		
Description	Compile the source code of a BSW modue without ECU specific configurations. This task is mainly used to describe the use cases of BSW development for object code delivery. The output will only represent the "core code". During ECU integration, additional generated code may be added per module in response to ECU configuration. Meth.bindingTime = CompileTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Basic Software Module Developer	1	
Consumes	BSW RTE Pre-build Configuration Header	1	Meth.bindingTime = PreCompileTime
Consumes	BSW Types	1	Meth.bindingTime = CodeGenerationTime

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Basic Software Interlink Types Header	1	Meth.bindingTime = CodeGenerationTime
Consumes	Basic Software Module Core Header	1	Meth.bindingTime = CodeGenerationTime
Consumes	Basic Software Module Core Source Code	1	Meth.bindingTime = CodeGenerationTime
Consumes	Basic Software Module Interlink Header	1	Meth.bindingTime = CodeGenerationTime
Consumes	Standard Header Files	1	Meth.bindingTime = CodeGenerationTime
Consumes	Build Action Manifest	0..1	The compilation can optionally be controlled by a Build Action Manifest.
Consumes	Library Header Files	0..1	Meth.bindingTime = CodeGenerationTime
Produces	Basic Software Module Object Code	1	Meth.bindingTime = CompileTime
Used tool	Compiler	1	

Table 3.228: Compile BSW Core Code

3.5.1.12 Generate BSW Module Prebuild Dataset

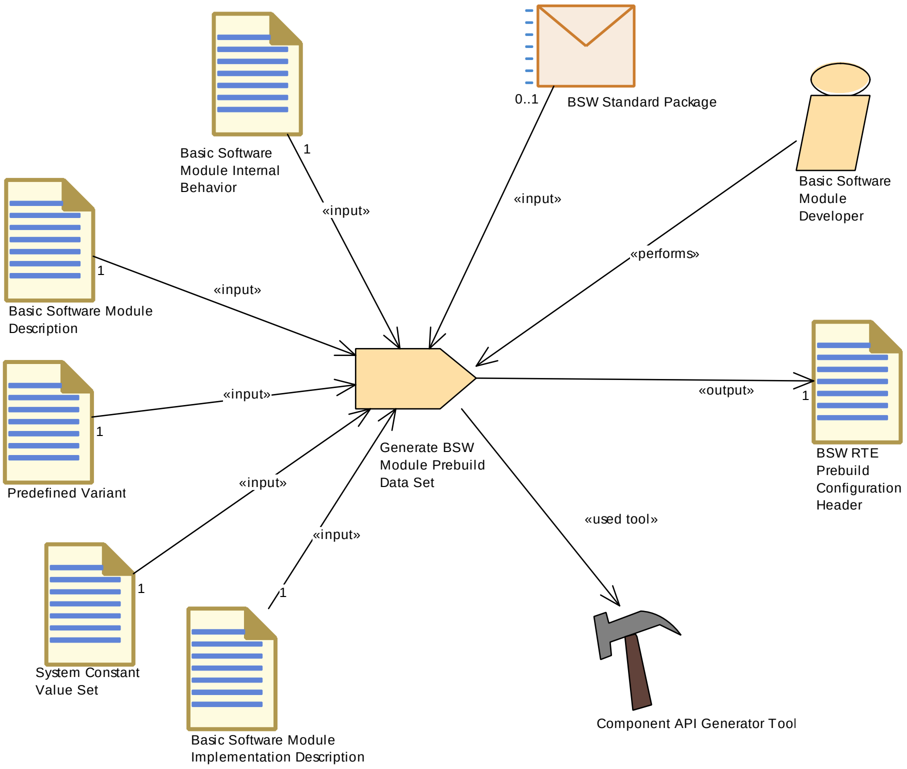

Figure 3.117: Generate BSW Module Prebuild Dataset

Task Definition	**Generate BSW Module Prebuild Data Set**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Tasks		
Brief Description	Prebuild Data Set Generation Phase for a BSW module: It binds all variations which need to be set after generation of the RTE contract header but before compilation of the module.		
Description	Prebuild Data Set Generation Phase for a basic software module: It binds all variations which need to be set after generation of the RTE contract header but before compilation of the module. The variant settings must be defined by the PredefinedVariant given as input. The output is a BSW Module RTE Prebuild Configuration Header which is included by the corresponding BSW Module Interlink Header, thereby resolving the variation points when compiled. Note that link time variants are not allowed here. Meth.bindingTime = PreCompileTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Basic Software Module Developer	1	
Consumes	Basic Software Module Description	1	Meth.bindingTime = CodeGenerationTime

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Basic Software Module Implementation Description	1	Meth.bindingTime = CodeGenerationTime
Consumes	Basic Software Module Internal Behavior	1	Meth.bindingTime = CodeGenerationTime
Consumes	Predefined Variant	1	
Consumes	System Constant Value Set	1	
Consumes	BSW Standard Package	0..1	
Produces	BSW RTE Pre-build Configuration Header	1	Meth.bindingTime = PreCompileTime
Used tool	Component API Generator Tool	1	

Table 3.229: Generate BSW Module Prebuild Data Set

3.5.2 Work Products

3.5.2.1 BSW Standard Package

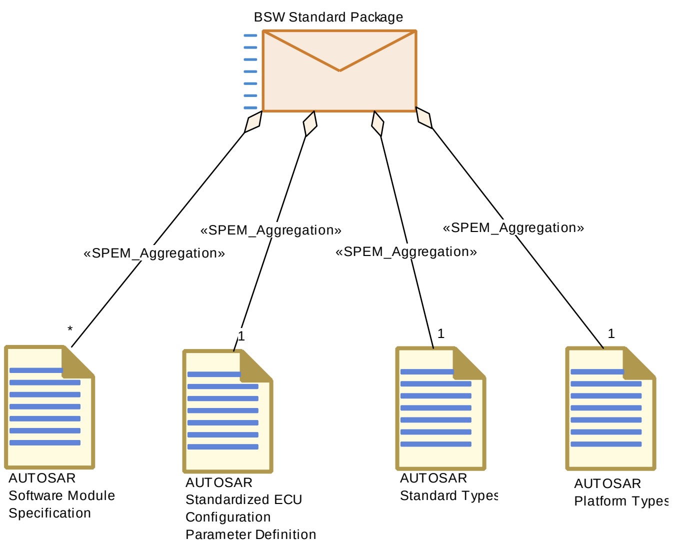

Figure 3.118: BSW Standard Package

Deliverable	**BSW Standard Package**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Package containing standard artifacts for BSW.		
Description	Contains the standard specifications and standard ARXML artifacts to be used within the AUTOSAR basic software and for the generation of the RTE. This deliverable is released by AUTOSAR and is readonly within the methodology.		
Kind	Delivered		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	AUTOSAR Platform Types	1	
Aggregates	AUTOSAR Standard Types	1	
Aggregates	AUTOSAR Standardized ECU Configuration Parameter Definition	1	
Aggregates	AUTOSAR Software Module Specification	0..*	
Consumed by	Create Library	1	Used for standard types and specifications.
Consumed by	Design Basic Software	1	
Consumed by	Develop BSW Module	1	
Consumed by	Develop BSW Module Generator	1	
Consumed by	Develop Basic Software	1	
Consumed by	Define BSW Behavior	0..1	
Consumed by	Define BSW Entries	0..1	
Consumed by	Define BSW Interfaces	0..1	
Consumed by	Define BSW Types	0..1	
Consumed by	Generate BSW Module Prebuild Data Set	0..1	
Consumed by	Generate BSWM Contract Header Files	0..1	
Consumed by	Implement a BSW Module	0..1	

Table 3.230: BSW Standard Package

3.5.2.2 BSW Module Bundle

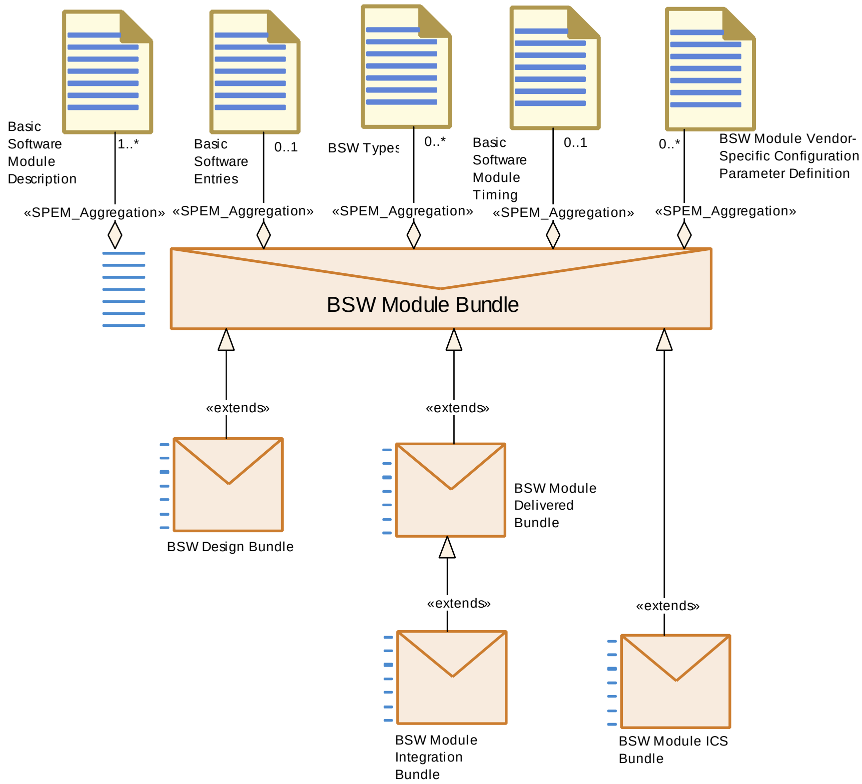

Figure 3.119: BSW Module Bundle

Deliverable	**BSW Module Bundle**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description			
Description	Generic deliverable representing a bundle of one or more BSW modules. It is used as a basis for extended deliverables. The deliverable aggregates the ARXML definitions on the interface level including vendor specific configuration parameter definition. According to the role of the extended deliverable, these elements maybe blueprints completely or partially. .		
Kind	Delivered		
Extended by	BSW Design Bundle, BSW Module Delivered Bundle, BSW Module ICS Bundle		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Basic Software Module Description	1..*	
Aggregates	Basic Software Entries	0..1	
Aggregates	Basic Software Module Timing	0..1	

Relation Type	Related Element	Mul.	Note
Aggregates	BSW Module Vendor-Specific Configuration Parameter Definition	0..*	The configuration parameter definitions of the modules under test - needed for static check against the standardized configuration parameters.
Aggregates	BSW Types	0..*	

Table 3.231: BSW Module Bundle

3.5.2.3 BSW Design Bundle

Deliverable	**BSW Design Bundle**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description			
Description	A bundle of one or more BSW modules used in the design phase. It contains only definitions on the interface level. These elements maybe blueprints completely or partially.		
Kind	Delivered		
Extends	BSW Module Bundle		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Design Basic Software	1..*	
Consumed by	Develop BSW Module	1..*	

Table 3.232: BSW Design Bundle

3.5.2.4 BSW Module ICS Bundle

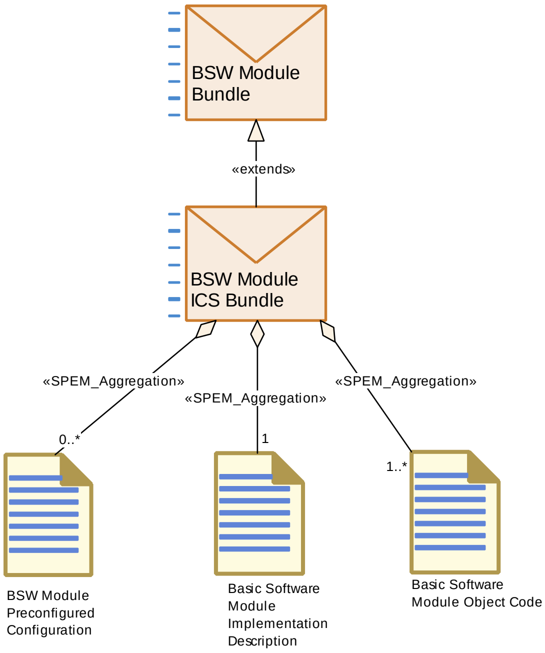

Figure 3.120: BSW Module ICS Bundle

Deliverable	**BSW Module ICS Bundle**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description			
Description	Deliverable containing the Implementation Conformance Statement (ICS) for one or more BSW modules.		
Kind	Delivered		
Extends	BSW Module Bundle		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Basic Software Module Implementation Description	1	The administrative elements (e.g. version info) of the Implementation model needed for the conformance test.
Aggregates	Basic Software Module Object Code	1..*	
Aggregates	BSW Module Preconfigured Configuration	0..*	The predefined configurations implemented by the modules under test. The modules under test are completely configured.

Table 3.233: BSW Module ICS Bundle

3.5.2.5 BSW Module Delivered Bundle

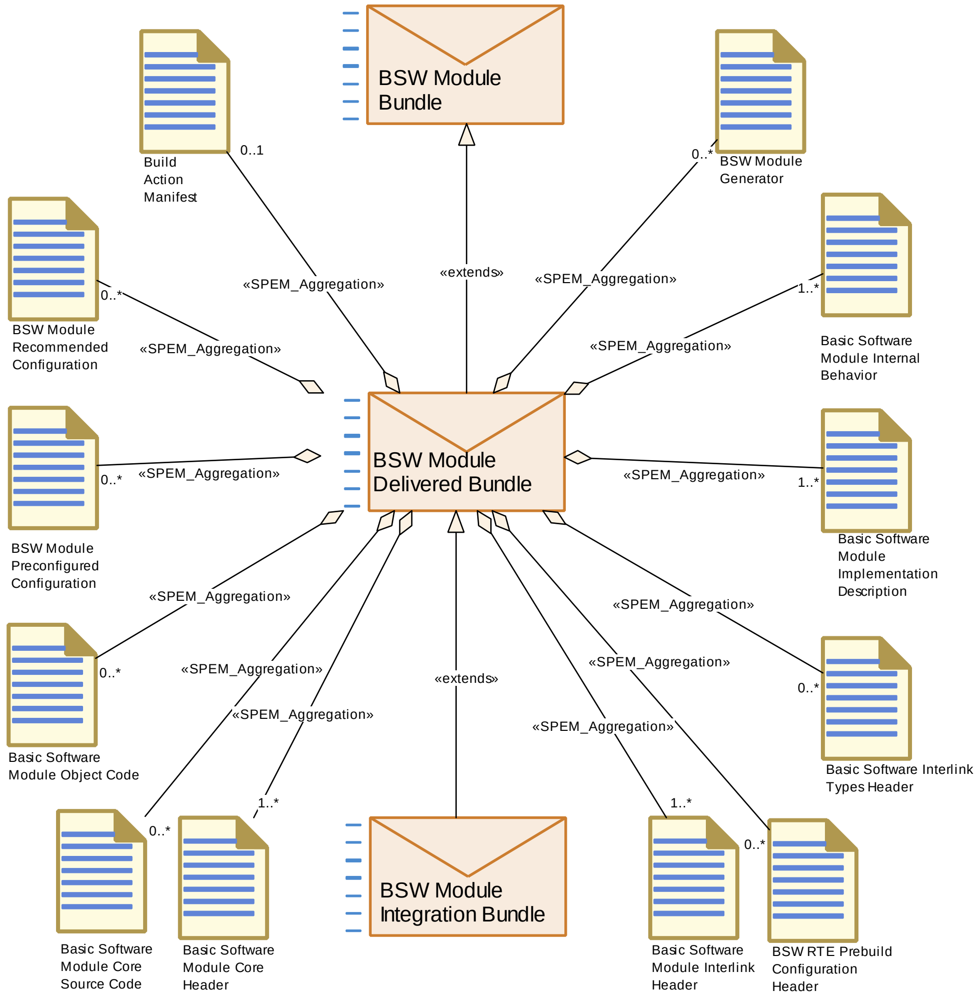

Figure 3.121: BSW Module Delivered Bundle

Deliverable	**BSW Module Delivered Bundle**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description			
Description	Deliverable containing one or more BSW modules delivered for integration (code and ARXML descriptions). It can still contain blueprints for some of the elements which need to be extended during ECU integration.		
Kind	Delivered		
Extended by	BSW Module Integration Bundle		
Extends	BSW Module Bundle		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Basic Software Module Core Header	1..*	

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Basic Software Module Implementation Description	1..*	
Aggregates	Basic Software Module Interlink Header	1..*	
Aggregates	Basic Software Module Internal Behavior	1..*	
Aggregates	Build Action Manifest	0..1	The build action manifest to be used for the delivered basic software.
Aggregates	BSW Module Generator	0..*	
Aggregates	BSW Module Pre-configured Configuration	0..*	
Aggregates	BSW Module Recommended Configuration	0..*	
Aggregates	BSW RTE Pre-build Configuration Header	0..*	
Aggregates	Basic Software Interlink Types Header	0..*	
Aggregates	Basic Software Module Core Source Code	0..*	
Aggregates	Basic Software Module Object Code	0..*	
Produced by	Develop BSW Module	1	
Produced by	Develop Basic Software	1..*	
Consumed by	Define Integration Variant	1..*	
Consumed by	Generate Base Ecu Configuration	1..*	Need vendor specific configuration parameters and their recommended or pre-configured values.
Consumed by	Generate Updated ECU Configuration	1..*	
Consumed by	Integrate Software for ECU	1..*	
Consumed by	Prepare ECU Configuration	1..*	
Consumed by	Configure Com	0..1	

Relation Type	Related Element	Mul.	Note
Consumed by	Configure Diagnostics	0..1	Predefined or recommended configuration values, vendor specific parameters, ServiceNeeds defined by BSW.
Consumed by	Configure MCAL	0..1	
Consumed by	Configure Mode Management	0..1	Predefined or recommended configuration values, vendor specific parameters, ServiceNeeds defined by BSW.
Consumed by	Configure NvM	0..1	Predefined or recommended configuration values, vendor specific parameters, ServiceNeeds defined by BSW.
Consumed by	Configure Watchdog Manager	0..1	Predefined or recommended configuration values, vendor specific parameters, ServiceNeeds defined by BSW.
Consumed by	Create Service Component	0..1	Required in order to define a mapping between SWC and BSW. In addition, the Build Action Manifest may be used.
Consumed by	Configure Debug	0..*	
Consumed by	Configure ECUC	0..*	
Consumed by	Configure IO Hardware abstraction	0..*	
Consumed by	Configure OS	0..*	OS Resources required by Basic Software. Optional Input: Basic Software Module Timing, e.g. execution order constraints.
Consumed by	Configure RTE	0..*	Input from the BSW Module Description is needed related to Scheduling, Exclusive Areas, Triggers and Modes. Optional Input: Basic Software Module Timing, e.g. execution order constraints.
	Configure Transformer	0..1	

Table 3.234: BSW Module Delivered Bundle

3.5.2.6 AUTOSAR Software Module Specification

Artifact	**AUTOSAR Software Module Specification**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	The standard sofware module specification.		
Description	Specification of a standardized Basic Software Module (SWS). It is published as a textual specification, but can be seen as a Basic Software Design bundle in the methodology, consisting mainly of blueprints. It may be published as ARXML in future releases of AUTOSAR.		
Kind	Text		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	BSW Standard Package	0..*	

Table 3.235: AUTOSAR Software Module Specification

3.5.2.7 AUTOSAR Standard Types

Artifact	**AUTOSAR Standard Types**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Contains all the standardized module definition parameters.		
Description	ARXML description of the AUTOSAR standard types (e.g. Std_ReturnType).		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	BSW Standard Package	1	
Aggregated by	VFB AUTOSAR Standard Package	1	
Use meta model element	Implementation DataType	1	

Table 3.236: AUTOSAR Standard Types

3.5.2.8 AUTOSAR Platform Types

Artifact	**AUTOSAR Platform Types**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Contains all the standardized module definition parameters.		
Description	ARXML description of the standardized part of the AUTOSAR platform types. It consists of • ImplementationDataTypes for the platform types - this part is still platform independent. • Blueprints of the underlying BaseTypes. These have to be refined for each processor platform.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	BSW Standard Package	1	
Aggregated by	VFB AUTOSAR Standard Package	1	
Use meta model element	Implementation DataType	1	
Use meta model element	SwBaseType	1	

Table 3.237: AUTOSAR Platform Types

3.5.2.9 BSW Module Generator

Artifact	**BSW Module Generator**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description			
Description	A generator that comes as part of one or more delivered BSW modules. It can be put into a framework to let it generate a module's configuration code.		
Kind	Custom		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	BSW Module Delivered Bundle	0..*	
Produced by	Develop BSW Module Generator	1	
Consumed by	Generate BSW Configuration Code	0..1	This is an input in case a generator framework is used which has to run some module specific generator code.

Table 3.238: BSW Module Generator

3.5.2.10 AUTOSAR Standardized ECU Configuration Parameter Definition

Artifact	**AUTOSAR Standardized ECU Configuration Parameter Definition**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Contains all the standardized module definition parameters.		
Description	Contains all the standardized module definition parameters. These parameters must be referred by the vendor specific configuration of a specific module.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	BSW Standard Package	1	
Consumed by	Configure Debug	1	
Consumed by	Define Vendor Specific Module Definition	1	
Consumed by	Configure Com	0..1	
Consumed by	Configure Diagnostics	0..1	
Consumed by	Configure ECUC	0..1	
Consumed by	Configure IO Hardware abstraction	0..1	
Consumed by	Configure MCAL	0..1	
Consumed by	Configure Mode Management	0..1	
Consumed by	Configure NvM	0..1	
Consumed by	Configure OS	0..1	
Use meta model element	EcucModuleDef	1	
	Configure Transformer	0..1	

Table 3.239: AUTOSAR Standardized ECU Configuration Parameter Definition

3.5.2.11 BSW Module Preconfigured Configuration

Artifact	**BSW Module Preconfigured Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Configuration parameter values that are fixed to the object code and cannot be changed without recompilation.		
Description	Configuration parameter values that are pre-configured in the delivered code. They cannot be changed during the ECU integration of the code. Pre-configuration is possible for object and source code as well.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	BSW Module Delivered Bundle	0..*	

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	BSW Module ICS Bundle	0..*	The predefined configurations implemented by the modules under test. The modules under test are completely configured.
Produced by	Configure Compiler Memory Classes	1..*	MemMap config for compiler memclasses: Set the parameter values that define generic MemClassSymbols (i.e. those not defined by modules or SWCs.). Set the parameter values that define the implementation behind all kind of MemClassSymbols (generic and local ones). Meth.bindingTime = SystemDesignTime
Produced by	Define Memory Addressing Modes	1..*	MemMapAddressingModeSet: Meth.bindingTime = SystemDesignTime
Consumed by	Configure Memmap Allocation	1..*	MemMapAddressingModeSet: Collection of compiler specific configuration elements for memory allocation and addressing modes.
Consumed by	Generate BSW Memory Mapping Header	1..*	MemMapAddressingModeSet: Collection of compiler specific configuration elements for memory allocation. Meth.bindingTime = SystemDesignTime
Consumed by	Generate Compiler Configuration	1..*	CompilerMemClassConfiguration: The parameters "MemMapCompilerMemClassSymbolImpl" and "MemMapGenericCompilerMemClassSymbolImpl" define the implementation behind a MemClassSymbol. Meth.bindingTime = SystemDesignTime
Consumed by	Generate SWC Memory Mapping Header	1..*	MemMapAddressingModeSet: Collection of compiler specific configuration elements for memory allocation. Meth.bindingTime = SystemDesignTime
Use meta model element	EcucModuleConfigurationValues	1	

Table 3.240: BSW Module Preconfigured Configuration

3.5.2.12 BSW Module Recommended Configuration

Artifact	**BSW Module Recommended Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Recommended "default" configuration parameter values.		
Description	Set of configuration parameter values, which are recommended by the module vendor as a default, but are not mandatory for the integration. There can be more than one such set in order to allow for variable usage of the module. This artifact does not include values of so-called published parameters. These must always be given as Basic Software Module Preconfigured Configuration.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	BSW Module Delivered Bundle	0..*	
Use meta model element	EcucModuleConfigurationValues	1	

Table 3.241: BSW Module Recommended Configuration

3.5.2.13 BSW Module Vendor Specific Configuration Parameter Definition

Artifact	**BSW Module Vendor- Specific Configuration Parameter Definition**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Vendor specific parameter definition for a module. This defines the format of the parameters, not its values.		
Description	Vendor specific parameter definition for a module. This defines the format of the parameters, not its values. In case of a standardized module, it redefines the existing standardized configuration parameter format (ModuleDef).		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Transformer Design Bundle	0..1	
Aggregated by	BSW Module Bundle	0..*	The configuration parameter definitions of the modules under test - needed for static check against the standardized configuration parameters.
Produced by	Define Vendor Specific Module Definition	1	
Consumed by	Configure RTE	1	The definitions for the module RTE
Consumed by	Develop BSW Module Generator	0..*	
Consumed by	Generate BSW Configuration Code	0..*	
Use meta model element	EcucModuleDef	1	

Table 3.242: BSW Module Vendor- Specific Configuration Parameter Definition

3.5.2.14 BSW Types

Artifact	**BSW Types**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Set of data types for usage within the Basic Software.		
Description	Set of data types (arxml descriptions) for usage by Basic Software Modules. They will be referred by the Basic Software Module Description		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	BSW Module Bundle	0..*	
In/out	Define BSW Types	1	
Consumed by	Compile BSW Core Code	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Define BSW Entries	1	
Consumed by	Define BSW Interfaces	1	
Use meta model element	AutosarDataType	1	

Table 3.243: BSW Types

3.5.2.15 Basic Software Entries

Artifact	**Basic Software Entries**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Set of signatures for calls between BSW modules.		
Description	Set of signatures for calls between BSW modules. Defining such a set as a separate artifact allows for a better reuse by several BSW modules.They are decribed in terms of the meta-model element BswModuleEntry which represents a C-function signature and associated properties.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	BSW Module Bundle	0..1	
Produced by	Define BSW Entries	1	
Consumed by	Define BSW Interfaces	1	
Use meta model element	BswModuleEntry	1	

Table 3.244: Basic Software Entries

3.5.2.16 Basic Software Module Description

Artifact	**Basic Software Module Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Description of a single BSW module or a module cluster in terms of its interfaces, dependencies and module Id.		
Description	Description of all interfaces (ingoing and outgoing C-function calls, triggers and modes) and other dependencies of a single BSW module or a module cluster. In addition, this artifacts defines the so-called module Id, which indicates the role of the module within the architecture (only mandatory for standardized modules). Note that the description of the function signatures (so-called BswModuleEntry and their ImplementationDataType can be factored out into separate artifacts BSW Entries and BSW Types in order to improve their reuse.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	BSW Module Bundle	1..*	
Produced by	Define BSW Interfaces	1	
Consumed by	Define BSW Behavior	1	
Consumed by	Generate BSW Module Prebuild Data Set	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Generate BSWM Contract Header Files	1	Meth.bindingTime = SystemDesignTime
Consumed by	Implement a BSW Module	1	Meth.bindingTime = SystemDesignTime
Consumed by	Generate BSW Memory Mapping Header	0..1	shortName: The BSW module's shortName is used as the first part of the generated file name, in case the default rule applies. Meth.bindingTime = SystemDesignTime
Use meta model element	BswModuleDescription	1	

Table 3.245: Basic Software Module Description

3.5.2.17 Basic Software Module Internal Behavior

Artifact	**Basic Software Module Internal Behavior**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Specifies the InternalBehavior of a BSW module or a BSW cluster, especially the scheduling aspect.		
Description	Specifies the behavior of a BSW module or a BSW cluster w.r.t. the code entities visible by the BSW Scheduler. It is possible to have several different BswInternalBehaviors referring to the same BswModuleDescription, but only one of them can be integrated on one CPU.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	BSW Module Delivered Bundle	1..*	
Produced by	Create Library	1	Meth.bindingTime = CodeGenerationTime
Produced by	Define BSW Behavior	1	
Consumed by	Define BSW Module Timing	1	
Consumed by	Generate BSW Module Prebuild Data Set	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Generate BSWM Contract Header Files	1	Meth.bindingTime = SystemDesignTime
Consumed by	Implement a BSW Module	1	Meth.bindingTime = SystemDesignTime
Consumed by	Map Software Component to BSW	1	
Consumed by	Generate Local MC Data Support	0..1	Meth.bindingTime = SystemDesignTime
Use meta model element	BswInternalBehavior	1	

Table 3.246: Basic Software Module Internal Behavior

3.5.2.18 Basic Software Module Implementation Description

Artifact	**Basic Software Module Implementation Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Contains the implementation specific information of a module.		
Description	Contains the implementation specific information of a module in addition to the generic specification given in Basic Software Module Description and Basic Software Module Internal Behavior.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	BSW Module ICS Bundle	1	The administrative elements (e.g. version info) of the Implementation model needed for the conformance test.
Aggregated by	BSW Module Delivered Bundle	1..*	
Produced by	Create Library	1	Meth.bindingTime = CodeGenerationTime
Produced by	Implement a BSW Module	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Generate BSW Memory Mapping Header	1	DependencyOnArtifact: Can be used to override the default name of the memory mapping header file. Meth.bindingTime = SystemDesignTime
Consumed by	Generate BSW Memory Mapping Header	1	MemorySections: MemorySections defined for a BSW module. This input includes optional prefixes for memory sections overriding the default rule. Meth.bindingTime = SystemDesignTime
Consumed by	Generate BSW Memory Mapping Header	1	infixes: Optional infixes (denoting instance and vendor ID) to be used within the created header file name. Meth.bindingTime = SystemDesignTime
Consumed by	Generate BSW Module Prebuild Data Set	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Generate BSWM Contract Header Files	1	Meth.bindingTime = SystemDesignTime
Consumed by	Generate Compiler Configuration	1..*	MemorySections: Find referred SwAddrMethods or specific memClassSymbols in the MemorySections defined for BSW modules. Meth.bindingTime = SystemDesignTime
Consumed by	Configure Memmap Allocation	0..*	MemorySections:
Use meta model element	BswImplementation	1	

Table 3.247: Basic Software Module Implementation Description

3.5.2.19 Build Action Manifest

Artifact	**Build Action Manifest**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Describes the actions used to build certain artifacts from other artifacts.		
Description	Describes the actions used to build certain artifacts from other artifacts (generate, compile, link...). Note: A build action manifest can include the actions for processing of basic software as well as of application software artifacts. The manifest itself is however considered as a product of basic software development.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	BSW Module Delivered Bundle	0..1	The build action manifest to be used for the delivered basic software.
Produced by	Implement a BSW Module	0..1	
Consumed by	Compile BSW Core Code	0..1	The compilation can optionally be controlled by a Build Action Manifest.
Consumed by	Compile ECU Source Code	0..1	The task may be controlled by a Build Action Manifest.
Consumed by	Connect Service Component	0..1	The task may be controlled by a Build Action Manifest.
Consumed by	Generate A2L	0..1	The task may be controlled by a Build Action Manifest.
Consumed by	Generate BSW Configuration Code	0..1	The task may be controlled by a Build Action Manifest.
Consumed by	Generate ECU Executable	0..1	The task may be controlled by a Build Action Manifest.
Consumed by	Generate OS	0..1	The task may be controlled by a Build Action Manifest.
Consumed by	Generate RTE Postbuild Dataset	0..1	The task may be controlled by a Build Action Manifest.
Consumed by	Generate RTE Prebuild Dataset	0..1	The task may be controlled by a Build Action Manifest.
Use meta model element	BuildActionManifest	1	

Table 3.248: Build Action Manifest

3.5.2.20 Basic Software Module Timing

Artifact	**Basic Software Module Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	BSW module's TimingDescription and TimingConstraints		
Description	TimingDescription and TimingConstraints defined for the Internal Behavior of a BSW module (BSWModuleEntities)		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	BSW Module Bundle	0..1	
Produced by	Define BSW Module Timing	1	
Consumed by	Define ECU Timing	0..1	
Consumed by	Implement a BSW Module	0..1	Meth.bindingTime = SystemDesignTime
Use meta model element	BswModuleTiming	1	

Table 3.249: Basic Software Module Timing

3.5.2.21 Basic Software Module Core Header

Artifact	**Basic Software Module Core Header**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	C-header files delivered with a BSW module.		
Description	C-header file delivered with a BSW module. It may have to be included by other modules.		
Kind	Source Code		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	BSW Module Delivered Bundle	1..*	
Produced by	Implement a BSW Module	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile BSW Configuration Data	1	
Consumed by	Compile BSW Core Code	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile Configured BSW	1	
Consumed by	Compile Unconfigured BSW	1	
Consumed by	Compile ECU Source Code	0..*	Meth.bindingTime = CodeGenerationTime

Table 3.250: Basic Software Module Core Header

3.5.2.22 Basic Software Module Core Source Code

Artifact	**Basic Software Module Core Source Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	The core source code of a module provided by the vendor.		
Description	The core source code of a module provided by the vendor. "Core" means, that it does not include addtional source code, which may be generated during the configuration process.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	BSW Module Delivered Bundle	0..*	
Produced by	Implement a BSW Module	0..1	The creation of source code is optional, since it might be generated completely in a later step based on the Build Action Manifest. Meth.bindingTime = CodeGenerationTime
Consumed by	Compile BSW Core Code	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile Configured BSW	1	
Consumed by	Compile Unconfigured BSW	1	
Consumed by	Compile ECU Source Code	0..*	Meth.bindingTime = CodeGenerationTime

Table 3.251: Basic Software Module Core Source Code

3.5.2.23 Basic Software Interlink Header

Artifact	**Basic Software Module Interlink Header**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Generated Header file used to link a BSW module with the BSW Scheduler.		
Description	Generated Header file used to link a BSW module with the BSW Scheduler during Contract phase.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	BSW Module Delivered Bundle	1..*	
Produced by	Generate BSWM Contract Header Files	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile BSW Core Code	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Implement a BSW Module	1	Meth.bindingTime = SystemDesignTime
Consumed by	Compile ECU Source Code	1..*	Meth.bindingTime = CodeGenerationTime

Relation Type	Related Element	Mul.	Note

Table 3.252: Basic Software Module Interlink Header

3.5.2.24 Basic Software Interlink Types Header

Artifact	Basic Software Interlink Types Header		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Generated Header file with data types used to link a BSW module with the BSW Scheduler		
Description	Generated Header file with data types used to link a BSW module with the BSW Scheduler.		
Kind	Source Code		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	BSW Module Delivered Bundle	0..*	
Produced by	Generate BSWM Contract Header Files	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile BSW Core Code	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Implement a BSW Module	1	Meth.bindingTime = SystemDesignTime
Consumed by	Compile ECU Source Code	0..*	Meth.bindingTime = CodeGenerationTime

Table 3.253: Basic Software Interlink Types Header

3.5.2.25 BSW RTE Prebuild Configuration Header

Artifact	BSW RTE Prebuild Configuration Header		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Generated header file used to resolve the prebuild variants in the prebuild RTE contract phase for the BSW.		
Description	Generated header file used to resolve the prebuild variants of a basic software module in the prebuild RTE contract phase. Contains macros which resolve the variants when compiled with the module.		
Kind	Bound Source Code		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	BSW Module Delivered Bundle	0..*	
Produced by	Generate BSW Module Prebuild Data Set	1	Meth.bindingTime = PreCompileTime
Consumed by	Compile BSW Core Code	1	Meth.bindingTime = PreCompileTime

Relation Type	**Related Element**	**Mul.**	**Note**
Consumed by	Compile ECU Source Code	0..*	Meth.bindingTime = PreCompileTime

Table 3.254: BSW RTE Prebuild Configuration Header

3.5.2.26 Basic Software Module Object Code

Artifact	**Basic Software Module Object Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Object code of a BSW module.		
Description	Object code of a BSW module.		
Kind	Object Code		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregated by	BSW Module ICS Bundle	1..*	
Aggregated by	BSW Module Delivered Bundle	0..*	
Produced by	Compile BSW Core Code	1	Meth.bindingTime = CompileTime
Produced by	Compile Configured BSW	1	
Produced by	Compile Generated BSW	1	
Produced by	Compile Unconfigured BSW	1	
Consumed by	Link ECU Code after Precompile Configuration	1..*	
Consumed by	Link ECU Code during Link Time Configuration	1..*	
Consumed by	Generate ECU Executable	0..*	for object code delivery Meth.bindingTime = CompileTime

Table 3.255: Basic Software Module Object Code

3.5.2.27 Library Description

Artifact	**Library Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	Description of a library in Autosar XML.		
Description	Description of a library in Autosar XML. This uses the same template as for describing Basic Software Modules, but with restricted content. Main purpose is to describe the C-interfaces of the library.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Create Library	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Implement Atomic Software Component	0..*	Meth.bindingTime = CodeGenerationTime
Use meta model element	BswModuleDescription	1	

Table 3.256: Library Description

3.5.2.28 Library Header Files

Artifact	**Library Header Files**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	These additional headers are typically needed for libraries that a component uses.		
Description	These additional headers are typically needed for libraries that a component or a module uses (e.g. a "math-libary").		
Kind	Source Code		
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Create Library	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile BSW Core Code	0..1	Meth.bindingTime = CodeGenerationTime
Consumed by	Implement a BSW Module	0..1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile Atomic Software Component	0..*	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile ECU Source Code	0..*	Meth.bindingTime = CodeGenerationTime
Consumed by	Implement Atomic Software Component	0..*	Meth.bindingTime = CodeGenerationTime
Consumed by	Re-compile Component in ECU context	0..*	Meth.bindingTime = CodeGenerationTime

Table 3.257: Library Header Files

3.5.2.29 Library Object Code

Artifact	**Library Object Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Bsw::Work products		
Brief Description	The object code of a lbrary.		
Description	The object code of a library, to be linked with other object code during a build of the ECU executable.		
Kind	Object Code		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	Delivered Atomic Software Components	0..*	
Produced by	Create Library	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Generate ECU Executable	0..*	for object code delivery Meth.bindingTime = CompileTime

Table 3.258: Library Object Code

3.5.2.30 Custom Transformer

Artifact	**Custom Transformer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System:: Work products		
Brief Description	Custom transformation		
Description	This is a user defined transformer that is not standardized in AUTOSAR.		
Kind			
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Implement a BSW Module	0..1	
Consumed by	Define Transformation Chain	0..1	

Table 3.259: Custom Transformer

3.6 ECU Integration and Configuration

This chapter contains the definition of work products and tasks used for the integration and configuration of AUTOSAR software on an ECU. For the definition of the relevant meta-model elements refer to [11].

3.6.1 Tasks

3.6.1.1 Provide RTE Calibration Dataset

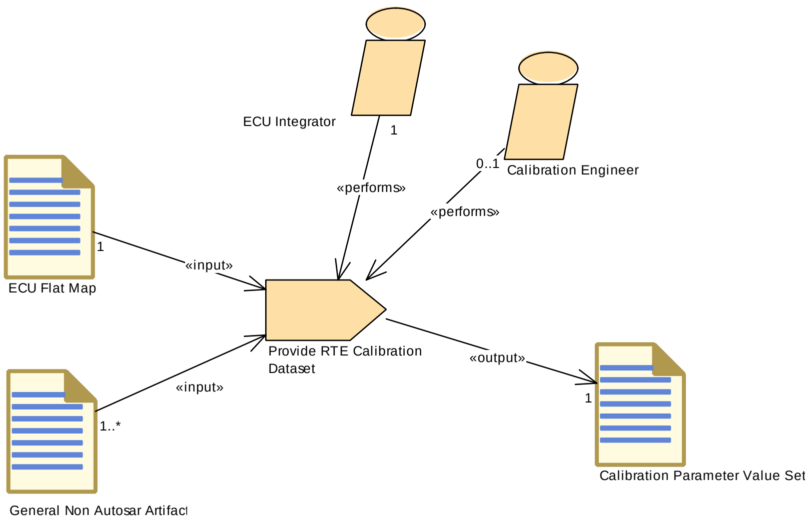

Figure 3.122: Provide RTE Calibration Dataset

Task Definition	**Provide RTE Calibration Dataset**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Provide a data set defining initial values for calibration parameters in the RTE code.		
Description	Since a model of the "downstream" calibration process of an ECU is not part of the AUTOSAR methodology, the input data are only shown as a General Non AUTOSAR Artifact. The output of this task is a set of calibration values in AUTOSAR format, which can be further processed within AUTOSAR, namely by the RTE generator. The calibration values have to be associated to the corresponding parameter specification via a reference to the ECU Flat Map.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Performed by	Calibration Engineer	0..1	
Consumes	ECU Flat Map	1	
Consumes	General Non Autosar Artifact	1..*	input from calibration process
Produces	Calibration Parameter Value Set	1	

Table 3.260: Provide RTE Calibration Dataset

3.6.1.2 Define Integration Variant

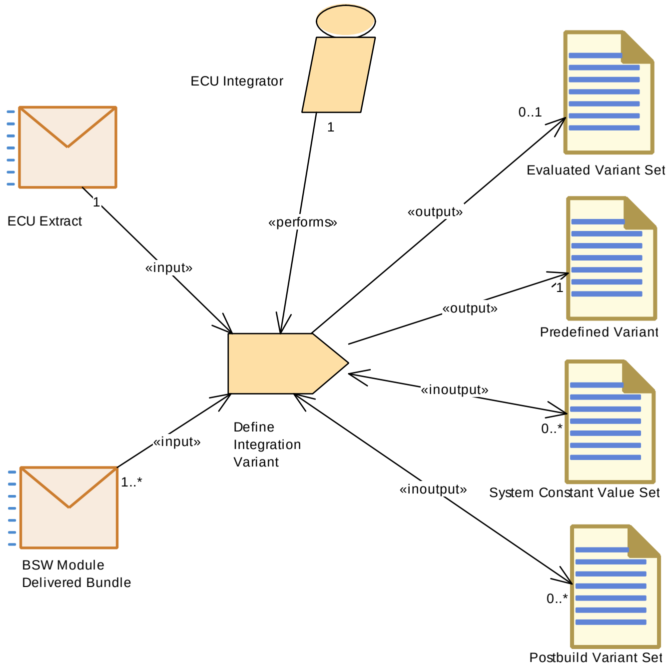

Figure 3.123: Define Integration Variant

Task Definition	**Define Integration Variant**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Define a variant for the artifacts integrated on an ECU.		
Description	Define a variant for the artifacts integrated on an ECU, this means adding a PredefinedVariant related to the ECU extract and the BSW modules in scope. To do so, this task can make use of existing System Constant Value Set and/or Postbuid Variant Sets or define new ones. Several PredefinedVariants can be combined to one Evaluated Variant Set. It is up to particular process definition to decide, which variants are allowed to be set at integration time. Technically, since this task is part of ECU integration, it can only resolve variation points which have not yet been resolved in the delivered ECU extract or BSW modules. Especially, variation points which have to be bound at system design time, should have been already resolved before. Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Extract	1	
Consumes	BSW Module Delivered Bundle	1..*	
In/out	Postbuild Variant Set	0..*	

Relation Type	*Related Element*	*Mul.*	*Note*
In/out	System Constant Value Set	0..*	
Produces	Predefined Variant	1	Meth.bindingTime = SystemDesignTime
Produces	Evaluated Variant Set	0..1	Meth.bindingTime = SystemDesignTime

Table 3.261: Define Integration Variant

3.6.1.3 Generate Base ECU Configuration

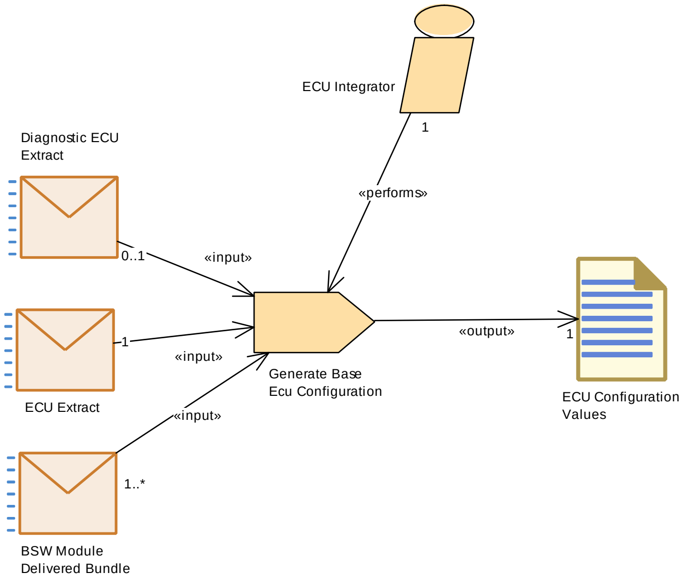

Figure 3.124: Generate Base ECU Configuration

Task Definition	**Generate Base Ecu Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Generate an initial set of ECU configuration values based on the delivered ECU extract.		
Description	Create the ECU configuration module structure including an initial set of ECU configuration values. This is based on the delivered ECU extract and on the vendor specific configuration parameters and their recommended or pre-configured values provided with the delivered BSW modules. Furthermore the diagnostic extract is used to create the initial configuration for diagnostic related modules, such as DCM and DEM. Meth.bindingTime = SystemDesignTime		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	ECU Integrator	1	
Consumes	ECU Extract	1	

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	BSW Module Delivered Bundle	1..*	Need vendor specific configuration parameters and their recommended or pre-configured values.
Consumes	Diagnostic ECU Extract	0..1	
Produces	ECU Configuration Values	1	Meth.bindingTime = SystemDesignTime

Table 3.262: Generate Base Ecu Configuration

3.6.1.4 Generate Updated ECU Configuration

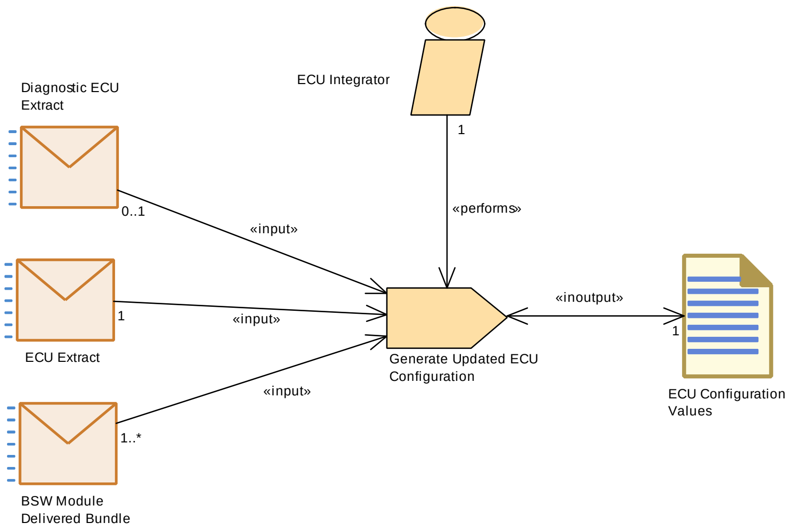

Figure 3.125: Generate Updated ECU Configuration

Task Definition	**Generate Updated ECU Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Generates the updated ECU configuration.		
Description	This task generates the updated ECU configuration based on the initial ECU configuration, the updated ECU Extract and optionally the Diagnostic Extract. Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Extract	1	
Consumes	BSW Module Delivered Bundle	1..*	
Consumes	Diagnostic ECU Extract	0..1	

Relation Type	***Related Element***	***Mul.***	***Note***
In/out	ECU Configuration Values	1	The task "Generate Updated ECU Configuration" consumes the initial ECU configuration values and produces the updated ECU configuration values.

Table 3.263: Generate Updated ECU Configuration

3.6.1.5 Define ECU Timing

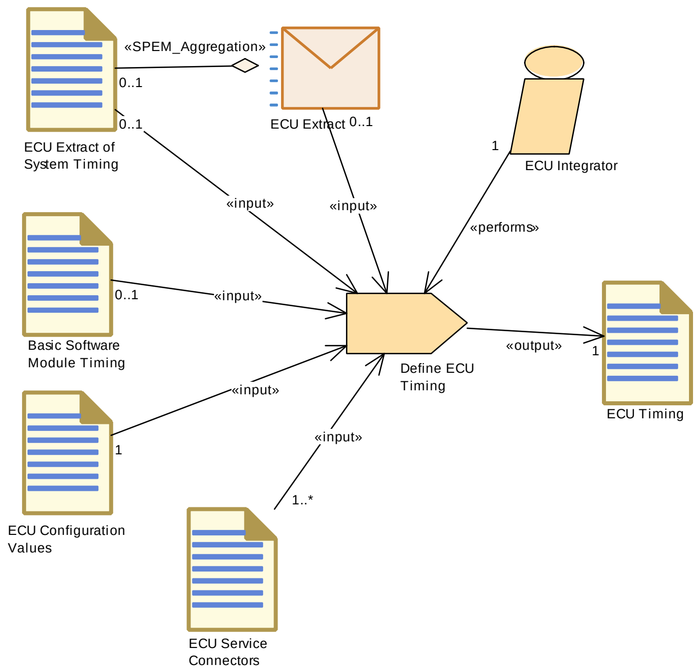

Figure 3.126: Define ECU Timing

Task Definition	**Define ECU Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Define ECUTiming (TimingDescription and TimingConstraints) for a concrete ECU taking the ECU configuration and the ECU Software Composition (including their implementation) into account.		
Description	Define ECUTiming (TimingDescription and TimingConstraints) for a concrete ECU taking the ECU configuration and the ECU Software Composition (including their implementation) into account. Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Configuration Values	1	
Consumes	ECU Service Connectors	1..*	

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Basic Software Module Timing	0..1	
Consumes	ECU Extract	0..1	Needed to set up links to the elements of the ECU extract.
Consumes	ECU Extract of System Timing	0..1	
Produces	ECU Timing	1	Meth.bindingTime = SystemDesignTime

Table 3.264: Define ECU Timing

3.6.1.6 Configure EcuC

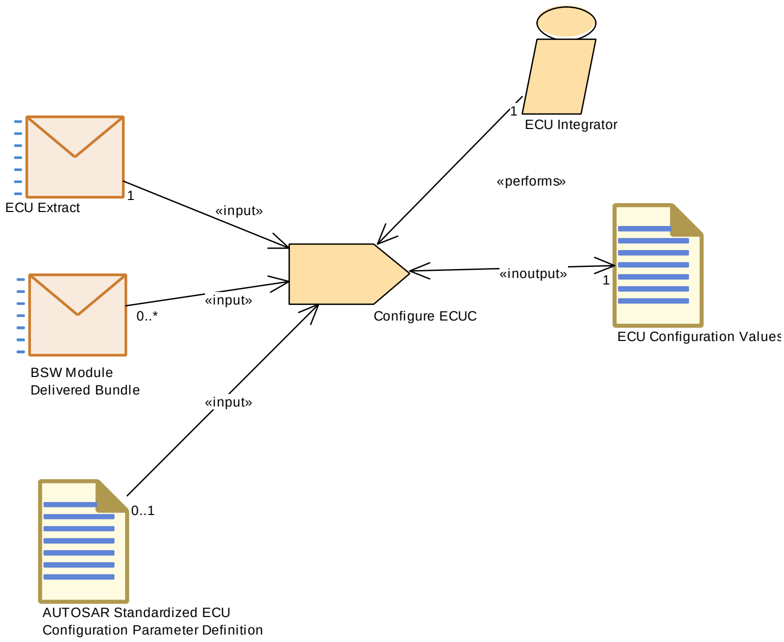

Figure 3.127: Configure EcuC

Task Definition	**Configure ECUC**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Set the general ECU configuration values.		
Description	Set the general ECU configuration values, the so-called EcuC parameters. These are the configuration parameters which are not related to a particular module, but are relevant for the ECU in general. The EcuC parameters consist of the following parts: • Collection of all Pdu objects flowing through the Com-Stack. • Definition of partitions for the ECU (One partition will be implemented using one OS application). The memory partitions have to be known before doing the OS configuration. • Collection of PredefinedVariant elements which shall be applied when resolving the variability during ECU Configuration. • Collection of mappings between ECU hardware memory segments (defined in ECU Resources Description) and SwAddrMethod elements (defined in VFB Types). The name of each such EcucMemoryMappingElement could be used as to predefine the logical memory segment for the linker configuration. Note: The usage of EcucMemoryMappingElement is deprecated in R4.0 rev.2, because the configuration of the "MemMap" module has been added which allows a more fined grained memory mapping than SwAddrmethod. A relatonship to hardware elements from this fine grained mapping is currently not provided. See task definition Configure Memmap Allocation. Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Extract	1	
Consumes	AUTOSAR Standardized ECU Configuration Parameter Definition	0..1	
Consumes	BSW Module Delivered Bundle	0..*	
In/out	ECU Configuration Values	1	

Table 3.265: Configure ECUC

3.6.1.7 Configure OS

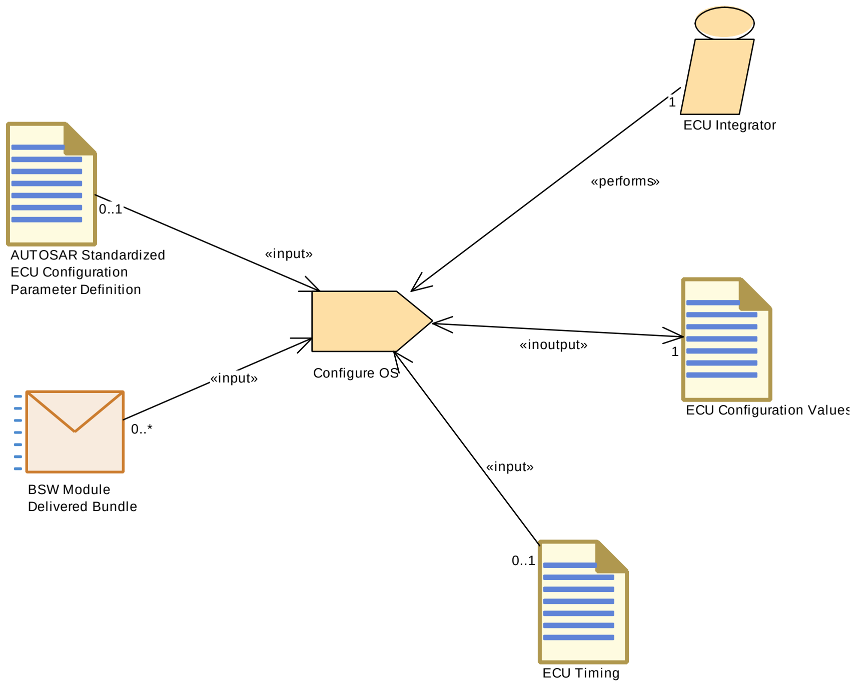

Figure 3.128: Configure OS

Task Definition	**Configure OS**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Configure the OS by creating the Tasks, events, alarms, etc.		
Description	The OS configuration process may be highly iterative between RTE and OS, e.g. RTE needs some OsTasks or OsScheduleTables to map Runnables into them. To finalize a ECU Configuration the OS is the last BSW module to configure (with the exception of the debugger). To use multi-core ECUs the EcuC Configuration needs to be provided beforehand to the OS Configuration to map the cores. There cannot be specified a precedence which configuration parameter values should be set first for OsAlarm, OsApplication, OsCounter, OsIsr, OsOs, OsResource, OsScheduleTable, OsSpinlock, OsTask. This is dependent on the development and configuration process. Application + Basic Software requirements and fulfill those with OS artifacts. Mandatory Inputs: • RTE part of the ECU Configuration • EcuC part of the ECU Configuration Outputs: • OS part of the ECU Configuration • RTE part of the ECU Configuration The following steps are needed to perform the task : • Map OS Configuration to Cores only in the case of multiple core ECU. • Define the OSTasks and OSSchedule : Tables based on the events/runnables of the application & bsw components, create the OSTasks that will invoke them. • Map Runnables into OSTasks and OSSchedule Tables : Assign all the runnables to the OSTasks • Steps for "OsAlarm, OsApplication, OsCounter, OsIsr, OsOs, OsResource, OsScheduleTable, OsSpinlock, OsTask." Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	AUTOSAR Standardized ECU Configuration Parameter Definition	0..1	
Consumes	ECU Timing	0..1	

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	BSW Module Delivered Bundle	0..*	OS Resources required by Basic Software. Optional Input: Basic Software Module Timing, e.g. execution order constraints.
In/out	ECU Configuration Values	1	

Table 3.266: Configure OS

3.6.1.8 Configure RTE

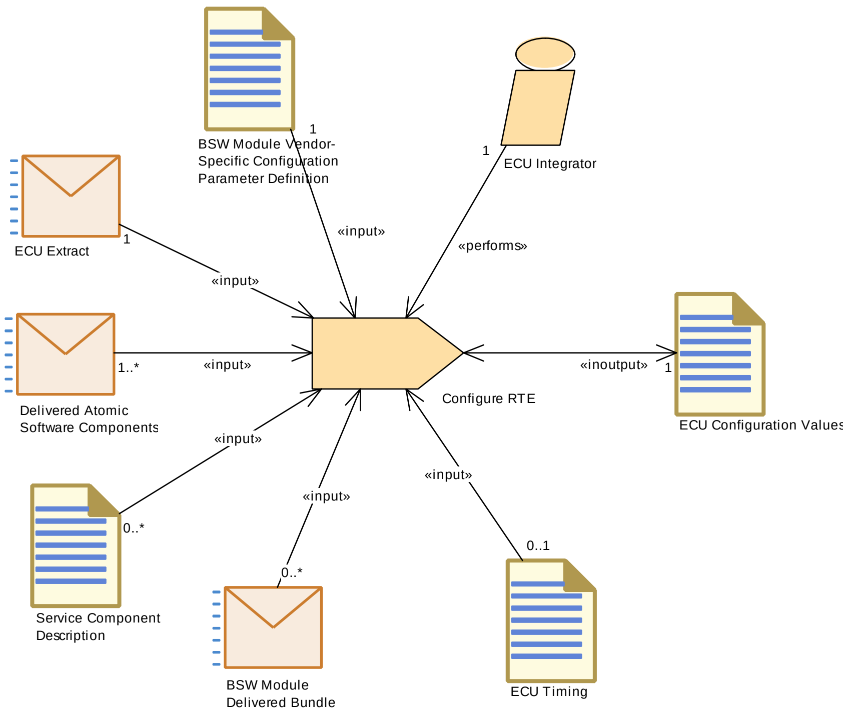

Figure 3.129: Configure RTE

Task Definition	**Configure RTE**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Describes the steps required to successfully configure the AUTOSAR RTE.		
Description	Configure the RTE to correctly interact with AUTOSAR COM and the OS. The specification of the OS objects used by the generated RTE are configured in this task. In addition, configuration includes setting RTE specific options and the handling of measurement and calibration data. Post-build variants which shall be supported by the RTE code must be referenced by the configuration. The following steps are usualy done to configure the RTE : 1.Setup RTE General Configuration 2.Select Software Component Implementations 3.Select BSW Module Implementations 4.Each Runnable needs to be assigned to an Operating System Task in order to be invoked. 5.Map BSW Executables to tasks 6.Resolve Exclusive Areas 7.Select Implicit Communication behavior 8.Select Calibration Support 9.Configure Non Volatile Memory Block Component (only needed if decisions on the configuration have to be taken during ECU Configuration) 10.Select the supported post-build variants Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	BSW Module Vendor-Specific Configuration Parameter Definition	1	The definitions for the module RTE
Consumes	ECU Extract	1	Elements of the System Description and VFB Description are referred by the RTE configuration. Optional Input: ECU Extract of System Timing, e.g. execution order constraints.
Consumes	Delivered Atomic Software Components	1..*	Required input: • References to all component implementation descriptions on this ECU • SwcInternalBehavior (for example to map the runnables to tasks) which was used in the contract phase of the software components on this ECU
Consumes	ECU Timing	0..1	
Consumes	BSW Module Delivered Bundle	0..*	Input from the BSW Module Description is needed related to Scheduling, Exclusive Areas, Triggers and Modes. Optional Input: Basic Software Module Timing, e.g. execution order constraints.

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Service Component Description	0..*	The Internal Behavior of Service Components contributes to the RTE configuration.
In/out	ECU Configuration Values	1	

Table 3.267: Configure RTE

3.6.1.9 Configure Watchdog Manager

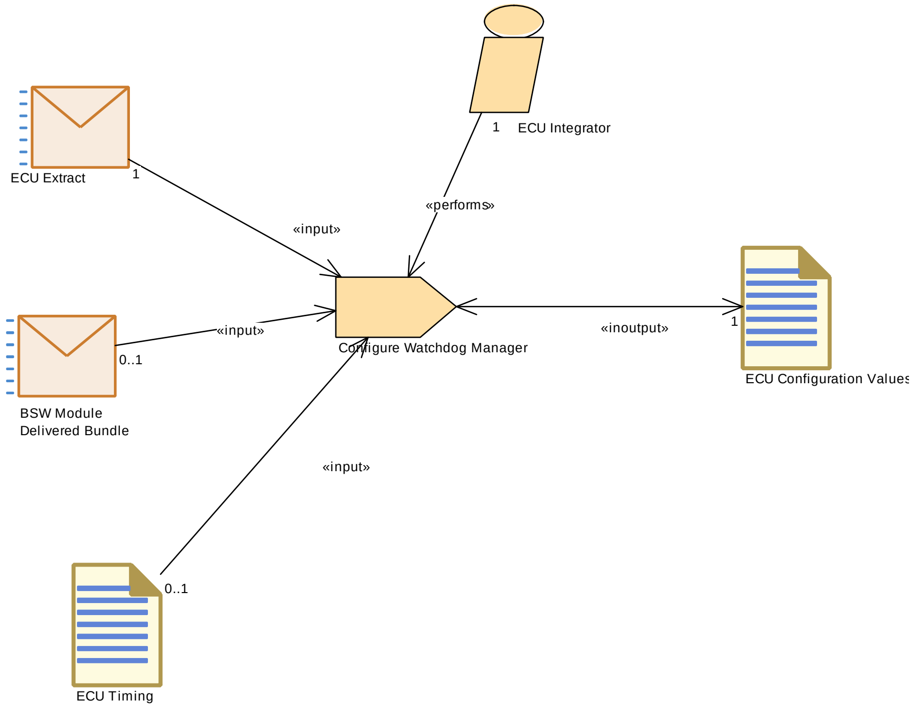

Figure 3.130: Configure Watchdog Manager

Task Definition	**Configure Watchdog Manager**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Describes the steps required to succesfully configure the Watchdog Manager		
Description	Configured Top-Down. Service needs determine what kind of watchdog manager you need. For each service need there is one interface. You can connect several of these interfaces to one watchdog manager Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Extract	1	Application software requirements for WdgM, especially SwcServiceDependency and ServiceNeeds.

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	BSW Module Delivered Bundle	0..1	Predefined or recommended configuration values, vendor specific parameters, ServiceNeeds defined by BSW.
Consumes	ECU Timing	0..1	
In/out	ECU Configuration Values	1	

Table 3.268: Configure Watchdog Manager

3.6.1.10 Configure Mode Management

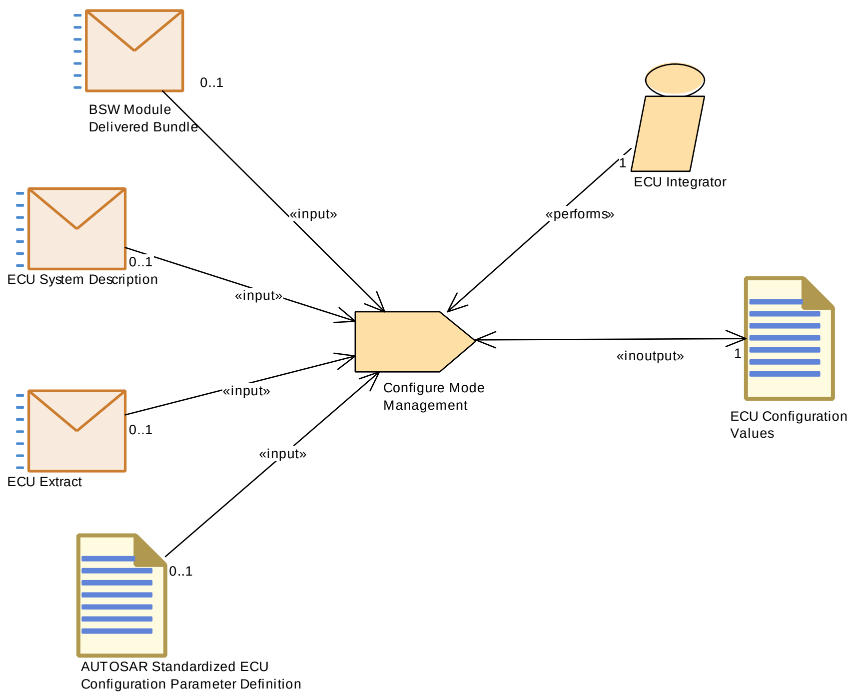

Figure 3.131: Configure Mode Management

Task Definition	Configure Mode Management		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Configure the Mode Managers in the Basic Software for this ECU.		
Description	Configure the Mode Managers in the Basic Software for this ECU. In the methodology library this is modeled as a single task (for simplicity) though in practice it may consist of several single tasks. In general, there are two approaches that are supported by AUTOSAR: • Top-down approach: the software components are available and the mode management can be configured using the data elements, i.e. mode requests, inside a port of a software component. • Bottom-up approach: the software components are not available and the mode management can be configured using a reference to a data element (stating the mode requests) in an interface, that is not yet used by a port of a software component. Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	AUTOSAR Standardized ECU Configuration Parameter Definition	0..1	
Consumes	BSW Module Delivered Bundle	0..1	Predefined or recommended configuration values, vendor specific parameters, ServiceNeeds defined by BSW.
Consumes	ECU Extract	0..1	Application software requirements for NvM, especially SwcServiceDependency and ServiceNeeds. Input in case atomic software components are available.
Consumes	ECU System Description	0..1	Input in case ECU Extract is not available (atomic software components not available)
In/out	ECU Configuration Values	1	

Table 3.269: Configure Mode Management

3.6.1.11 Configure NvM

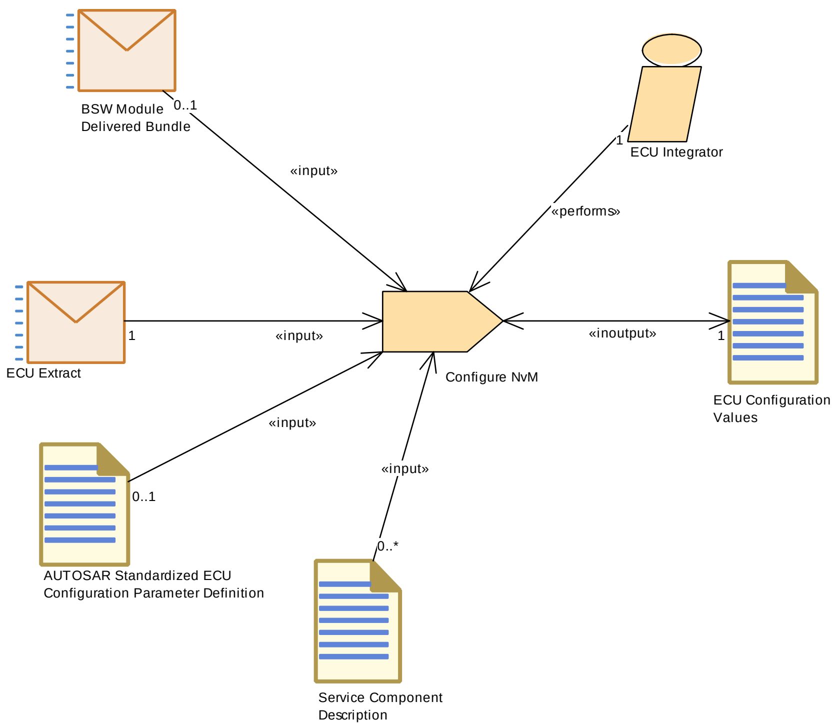

Figure 3.132: Configure NvM

Task Definition	**Configure NvM**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Configure the NvM stack for this ECU.		
Description	Configure the NvM stack for this ECU. In the methodology library this is modeled as a single task (for simplicity) though in practice it may consist of several single tasks. Requirements for the configuration of NvM can be collected • from the upstream information about ServiceDependencies and ServiceNeeds in the ECU Extract and BSW Modules • from existing ECU configuration values • from Service Component Descriptions created for other Services (e.g. DEM) Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Extract	1	Application software requirements for NvM, especially SwcServiceDependency and ServiceNeeds.

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	AUTOSAR Standardized ECU Configuration Parameter Definition	0..1	
Consumes	BSW Module Delivered Bundle	0..1	Predefined or recommended configuration values, vendor specific parameters, ServiceNeeds defined by BSW.
Consumes	Service Component Description	0..*	The configuration of diagnostics, especially of the DEM, typically leads to the definition of additional data to be stored in NvM. One possibility to handle this is to create ServiceNeeds on the level ServiceComponentType which is then taken into account for the configuration of the NvM.
In/out	ECU Configuration Values	1	

Table 3.270: Configure NvM

3.6.1.12 Configure Diagnostics

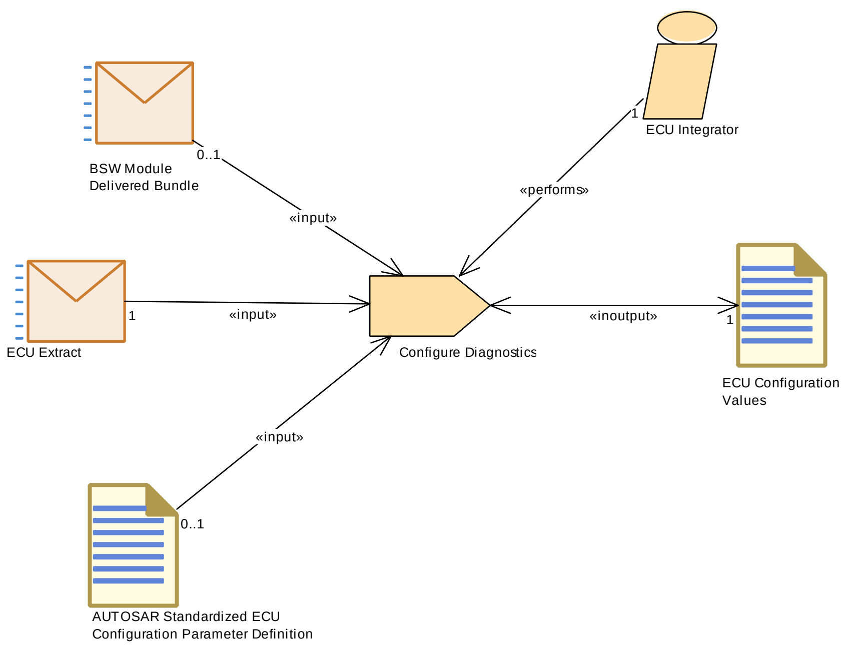

Figure 3.133: Configure Diagnostics

Task Definition	**Configure Diagnostics**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Configure the diagnostic modules for this ECU		
Description	Configure the diagnostic modules for this ECU. In the methodology library this is modeled as a single task (for simplicity) though in practice it may consist of several single tasks. Meth.bindingTime = SystemDesignTime		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	ECU Integrator	1	
Consumes	ECU Extract	1	Application software requirements for diagnostics, especially SwcServiceDependency and ServiceNeeds.
Consumes	AUTOSAR Standardized ECU Configuration Parameter Definition	0..1	
Consumes	BSW Module Delivered Bundle	0..1	Predefined or recommended configuration values, vendor specific parameters, ServiceNeeds defined by BSW.
In/out	ECU Configuration Values	1	Configuration Values for DEM, DCM, DLT, FIM.

Table 3.271: Configure Diagnostics

3.6.1.13 Create Service Component

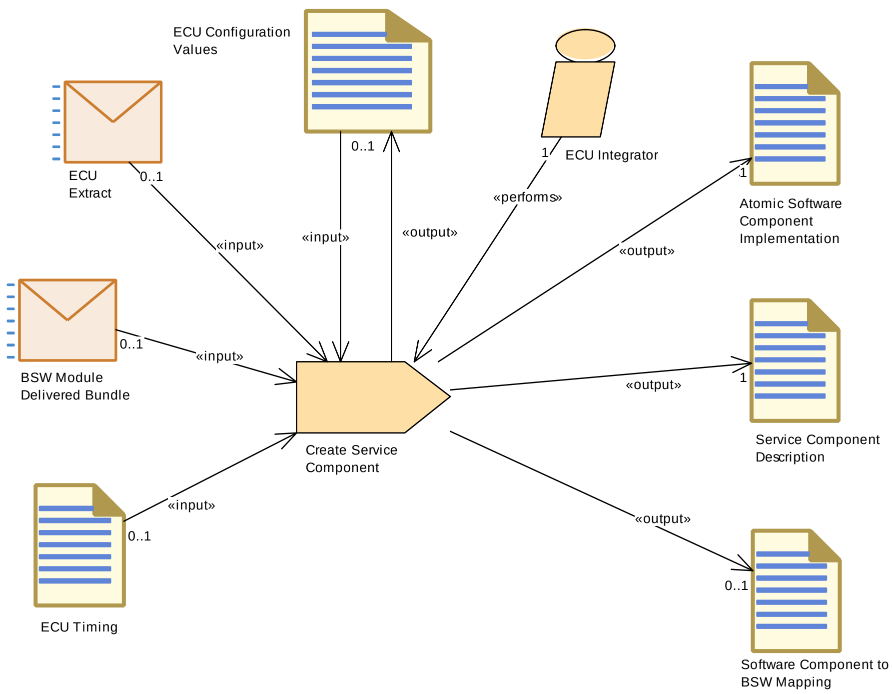

Figure 3.134: Create Service Component

Relation Type	***Related Element***	***Mul.***	***Note***
Task Definition	**Create Service Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Create an instances for all required Service Components, configure them, create necessary ports and connectors to the respective application software components. This completes the ECU Software Composition.		
Description	The ECU Extract contains all information about which components are mapped to a specific ECU. In a new "flat" Software Composition (meta-class RootSwCompositionPrototype) all other compositions have been removed. This has to be extended by an aggregation of the SwComponentPrototypes which describe the Services required by all application components on the ECU: • For each mapped SwComponentPrototype of type AtomicSwComponentType, the PortPrototypes requiring a particular Service and the asscociated SwcServiceDependency-s and ServiceNeeds are collected. Based on this information, a ServiceSwComponentType and its prototype is created exactly once per service with the corresponding number of PortPrototypes, thus that all service-type PortPrototypes of the Application Components have their PortPrototype counterpart on the ServiceSwComponentType. • RTE generation requires that an InternalBehavior and Implementation is created for each ServiceSwComponentType. In particular, the port defined argument values required for the usage of some service interfaces are configured, and the required RunnableEntities and RTEEvents are set up. It is also required to define a mapping between elements of the generated SWC and existing or generated elements of the BSW module description. • The evaluation of the input might result in further ServiceNeeds to be added to the generated InternalBehavior - for example a ServiceSwComponentType created for the DEM might include ServiceNeeds for NVRAM blocks. It is assumed, that such interdependencies are incrementally resolved within this task for all involved Service Components such that the outputs are consistent. Note that this is just one possibility to handle the situation - another option is to resolve the interdependencies only within the ECU configuration tasks (Configure Diagnostics, Configure NvM) without creating additional ServiceNeeds. Depending on the details of the configuration process for the particular module (namely which parts are generated or manually created), the steps described above can be done before, in parallel or after setting up the ECU configuration of the involved BSW modules. Likewise, the information used to create the ServiceSwComponentType(s) can come directly as input from the ECU Extract, or via the ECU Configuration. Therefore both artifacts are shown as optional input. The ECU Configuration is also an output, because a reference to the created SwComponentPrototype(s) must be entered here. The creation of connectors between the service and application components is a separate task.. Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	ECU Integrator	1	
Consumes	BSW Module Delivered Bundle	0..1	Required in order to define a mapping between SWC and BSW. In addition, the Build Action Manifest may be used.
Consumes	ECU Configuration Values	0..1	The creation of Service Component details may depend on ECU configuration values, especially for the DCM.
Consumes	ECU Extract	0..1	Input information about the Service Ports and Service Dependencies of the software components.
Consumes	ECU Timing	0..1	Additional information for fine tuning configuration decisions.
Produces	Atomic Software Component Implementation	1	In order to generate the RTE, one needs to create a kind of dummy Implementation element for the Service Component, however this should not be filled with descriptive elements, e.g. resource consumption, as these are already defined by the Basic Software Module Implementation Description. Meth.bindingTime = SystemDesignTime
Produces	ECU Configuration Values	1	Enter links to the created SwComponentPrototypes. Meth.bindingTime = SystemDesignTime
Produces	Service Component Description	1	Meth.bindingTime = SystemDesignTime
Produces	Software Component to BSW Mapping	0..1	Meth.bindingTime = SystemDesignTime

Table 3.272: Create Service Component

3.6.1.14 Connect Service Component

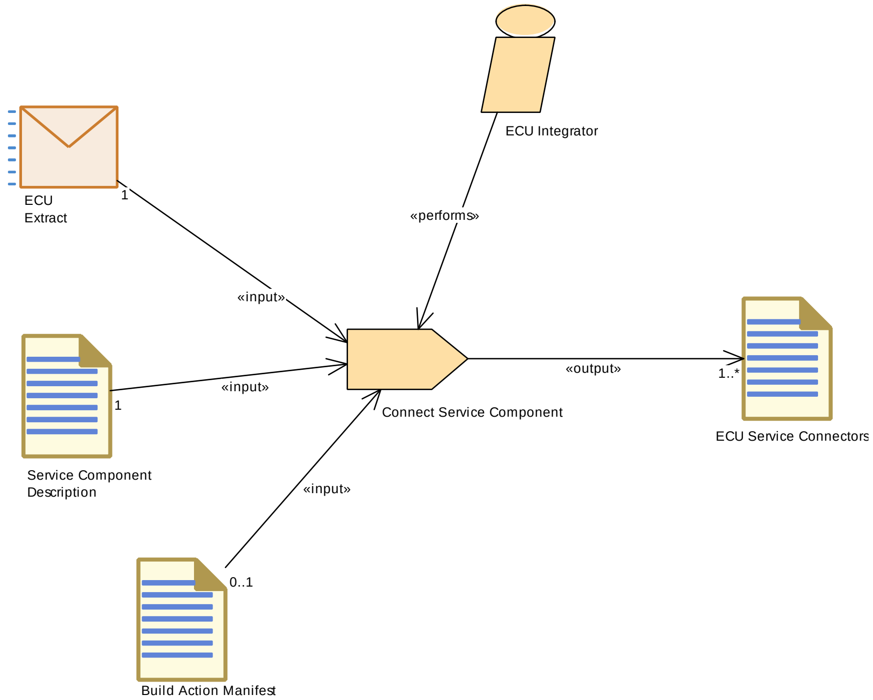

Figure 3.135: Connect Service Component

Task Definition	**Connect Service Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description			
Description	In order to connect the "isService"-ports of the application components to a particular ServiceSwComponentType, AssemblyConnectorPrototypes are generated. The ECU Extract with its RootSwCompositionPrototype, extended by the Service Components and their connectors, finally serves as input for generating the RTE. Meth.bindingTime = SystemDesignTime		
Relation Type	**Related Element**	**Mul.**	**Note**
Performed by	ECU Integrator	1	
Consumes	ECU Extract	1	Find the ports on the application side to be connected to the Service Component.
Consumes	Service Component Description	1	Required in order to define the connector links to the ports on the BSW side.
Consumes	Build Action Manifest	0..1	The task may be controlled by a Build Action Manifest.
Produces	ECU Service Connectors	1..*	Meth.bindingTime = SystemDesignTime

Table 3.273: Connect Service Component

3.6.1.15 Configure COM

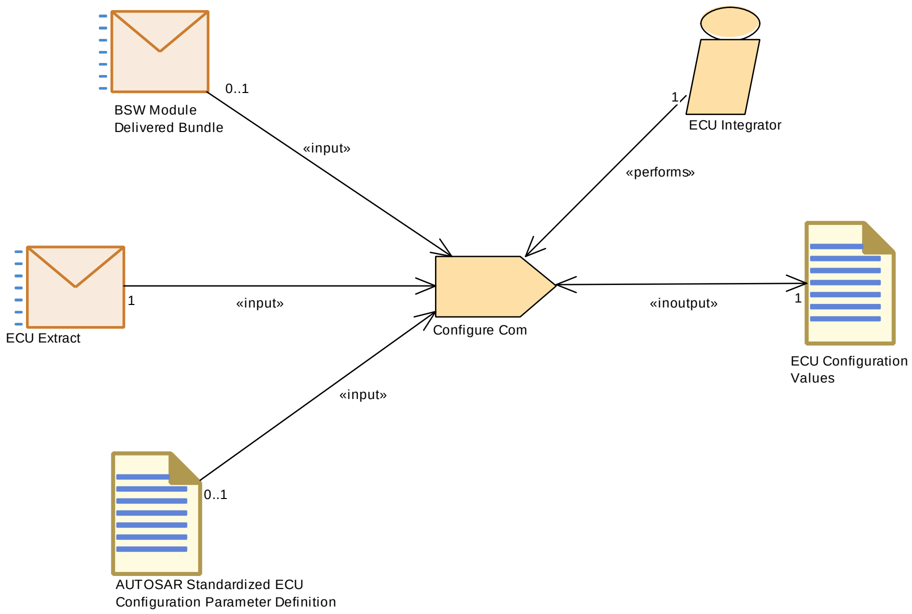

Figure 3.136: Configure COM

Task Definition	**Configure Com**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Configure the COM stack modules within an ECU		
Description	The ECU Extract of the System Configuration contains the major part of information that is needed to configure the COM Stack modules. Many parameter values of the ECU configuration can be derived from the ECU extract.The missing ECU specific configuration parameters that can not be derived from the System Description need to be set in this phase, e.g. Vendor-Specific Configuration Parameters. The following steps will be needed to perform the task : 1- Derive configuration parameter values from ECU extract : The System Template Specification describes rules on how the individual ECU configuration parameters shall be derived from the Upstream Templates (SWC Template, System Template, ECU Resource Template). This rules shall be followed. 2- Derive global PDUs from ECU extract : A global PDU has to be configured for each I-PDU flow and is added to the PDU collection of the module EcuC. Derived from the ECU Extract all PDUs that traverse through the COM Stack have to be created. 3- Create PDU References from the BSW Module PDUs to the global PDUs in the module EcuC:As soon as these global PDUs are created the references from the local module PDUs to the appropriate global PDUs need to be configured. 4-Set Missing and Vendor-Specific Parameter Values:Missing and Vendor-Specific Parameter Values need to be set 5-Set BSW Module specific PDU handle IDs:The last step is the assignment of the actual values for the Handle IDs. This can be achieved by an automatic tool which might be run directly before the generation of the module. Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Extract	1	
Consumes	AUTOSAR Standardized ECU Configuration Parameter Definition	0..1	
Consumes	BSW Module Delivered Bundle	0..1	
In/out	ECU Configuration Values	1	

Table 3.274: Configure Com

3.6.1.16 Configure IO Hardware Abstraction

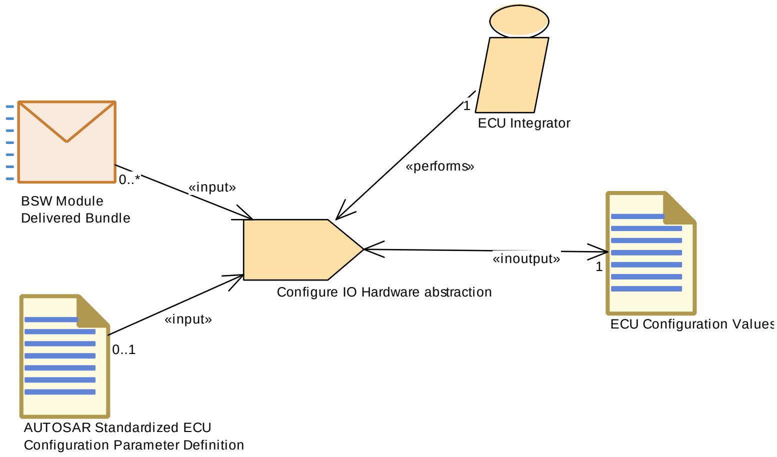

Figure 3.137: Configure IO Hardware Abstraction

Task Definition	**Configure IO Hardware abstraction**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Configure I/O Hardware Abstraction		
Description	Configure the I/O Hardware Abstraction modules. Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	AUTOSAR Standardized ECU Configuration Parameter Definition	0..1	
Consumes	BSW Module Delivered Bundle	0..*	
In/out	ECU Configuration Values	1	

Table 3.275: Configure IO Hardware abstraction

3.6.1.17 Configure MCAL

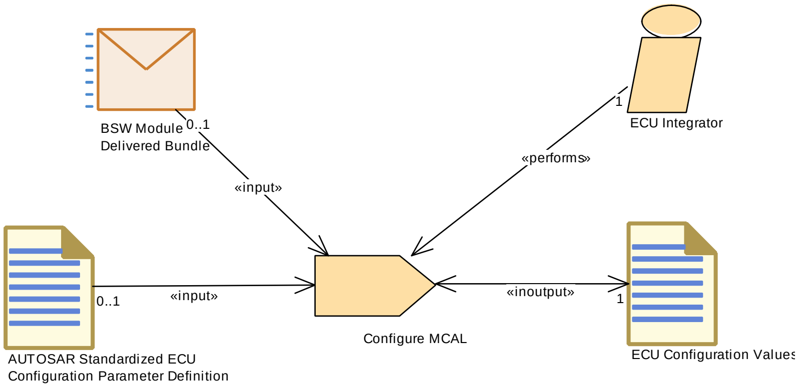

Figure 3.138: Configure MCAL

Task Definition	**Configure MCAL**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Configure the Microcontroller Abstraction Layer for this ECU.		
Description	Configure the Microcontroller Abstraction Layer for this ECU. Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	AUTOSAR Standardized ECU Configuration Parameter Definition	0..1	
Consumes	BSW Module Delivered Bundle	0..1	
In/out	ECU Configuration Values	1	

Table 3.276: Configure MCAL

3.6.1.18 Configure Debug

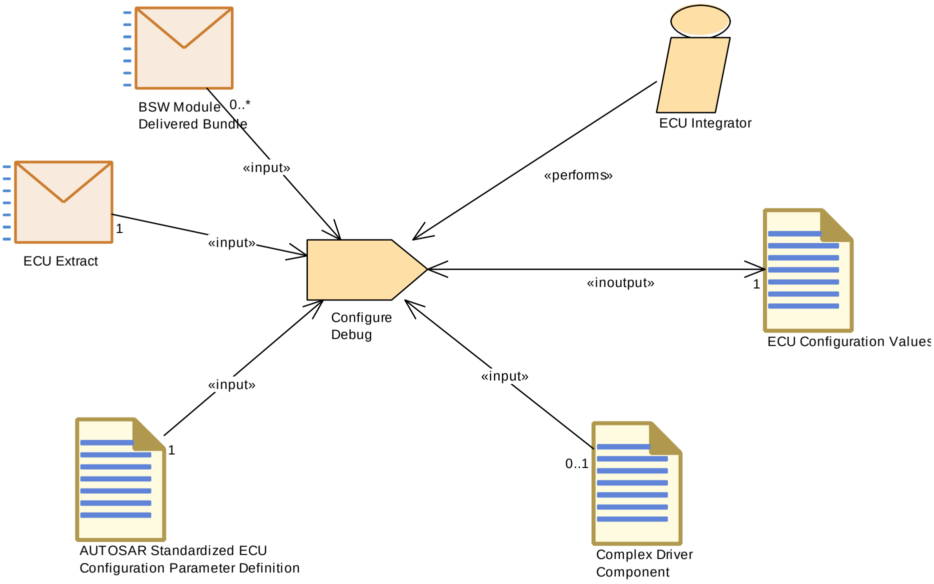

Figure 3.139: Configure Debug

Task Definition	**Configure Debug**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Configure the AUTOSARdebugger Module		
Description	The AUTOSAR Debugger Module (Dbg) handles the interaction between the Debugger Host and the AUTOSAR ECU. It is split into the "core" and the "communication" part. Each BSW has an ID & Each API has an ID. (e.g. module 84, api 5). The Debugger Host (shortly called Host) may be connected via 1. Existing communication buses which are also used for the functional behavior of the ECU. 2. A dedicated debugging line which is not used for functional behavior of the ECU. (e.g. via Complex Driver) Since Dbg needs information on the debugged software, it is configured quite late in the ECU Configuration steps. Other modules must be configured before the debug. Even after changes of the OS configuration, Dbg needs to be updated as well. The input to the Dbg ECU Configuration are: 1. ECU Configuration Values description • If existing communication buses are used, Dbg needs to transmit and receive I-Pdus which then are handled in the COM-Stack. Those I-Pdus need to be created / referenced. • Usage of OsAlarm • Usage of GptChannel (optional, for time stamping) 2. BSW Module Descriptions of the debugged modules in order to identify which variables / functions can be debugged. Prerequisites are: The variables need to be placed in global accessible memory; the data types of these variables need to be defined in the header files. Meth.bindingTime = SystemDesignTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	AUTOSAR Standardized ECU Configuration Parameter Definition	1	
Consumes	ECU Extract	1	
Consumes	Complex Driver Component	0..1	
Consumes	BSW Module Delivered Bundle	0..*	
In/out	ECU Configuration Values	1	

Table 3.277: Configure Debug

The task to configure the debug module consists of the following detailed steps (not shown in the table above due to formating reasons):

1. RTE VFB-Tracing if needed : The `RTE ECU Configuration` shall contain a "`RteVfbTraceClientPrefix = Dbg`".

2. Periodic Data Collection if needed : Configure the reference to the `OsAlarm` which will invoke the periodic data collection. Note that the `OsAlarm` needs to be configured in the `Os ECU Configuration` (before or after).

3. Timestamp Measurement if needed : Configure the size of the timestamp (16 or 32 bit) then configure the reference to the GptChannel which will provide the timestamp information. Note that the GptChannel needs to be configured in the Gpt ECU Configuration (before or after).

4. Configure the Buffering of the Debug : Size, Strategy (last-is-best/queued) and behavior.

5. AUTOSAR Communication stack : Configure the used Tx and Rx I-Pdus, the corresponding I-Pdus need to be configured in the EcuC Module and the rest of the COM-Stack. If Complex Driver is used for communication, configure Complex Driver.

6. Configure the to be debugged elements - BSW only - Prerequisite: The BSW Module shall be already configured and generated therefore there is an updated BSW-Module Description available of the actually generated BSW Module.The first work will be to get the list of traceable API calls out of the BSWMD of the BSW Module. Then select which API calls shall be traced (e.g. call "CanIf_Transmit" from the "PduR" to the "CanIf") and configure each trace function: buffering, timestamp.

7. Configure the to be debugged elements - RTE only - Prerequisite: The RTE has been generated, therefore there is an updated BSW-Module Description available of the actually generated RTE. Attention: The RTE shall not be re-configured after the Dbg has been configured, otherwise the Dbg needs to be re-configured as well. The first work will be get the list of available VFB-Trace functions out of the BSWMD of the RTE. Then, Select which VFB-Trace functions shall be traced (e.g. Rte_Dbg_Runnable_component_re_Start()), configure each VFB-Trace function: Buffering,Timestamp, in case of Rte-Com tracing: which Com-Signal is traced, in case of VFB-Signal tracing: which VariablePrototype is traced, in case of Client-Server tracing: which OperationPrototype is traced, in case of RunnableEntity tracing: which RunnableEntity is traced.

8. Configure the to be debugged elements - BSW and RTE - Prerequisite: The RTE has been generated, therefore there is an updated BSW-Module Description available . Attention: The RTE shall not be re-configured after the Dbg has been configured.The first step will be out of the BSWMD of the BSW and the RTE to extract the list of available debuggable variables and provide it to the Dbg configuration. Then, select which variables shall be debugged (e.g. internal states of the module), configure each individual DID with symbol name, optional size, optional absolute address, buffering, timestamp, collection frequency Note: Size and address (e.g. for an ECU register) could be resolved by the linker, hence optional here.

9. Generate the Dbg Module: Generate the c and header files of the Dbg, use the additional header files of the to be debugged modules in order to perform a "sizeof()" operation in the compiler, compile Dbg Module (and other to-be-debugged modules), analyze the object file in order to update the ECU Configuration Values description which additional information the length information for each DID (out of the sizeof() operation). Host application uses this information (ECU configuration of debug module, BSW module description of the debug module and the to-be-debugged modules) in order to send the correct DIDs.

3.6.1.19 Configure Transformer

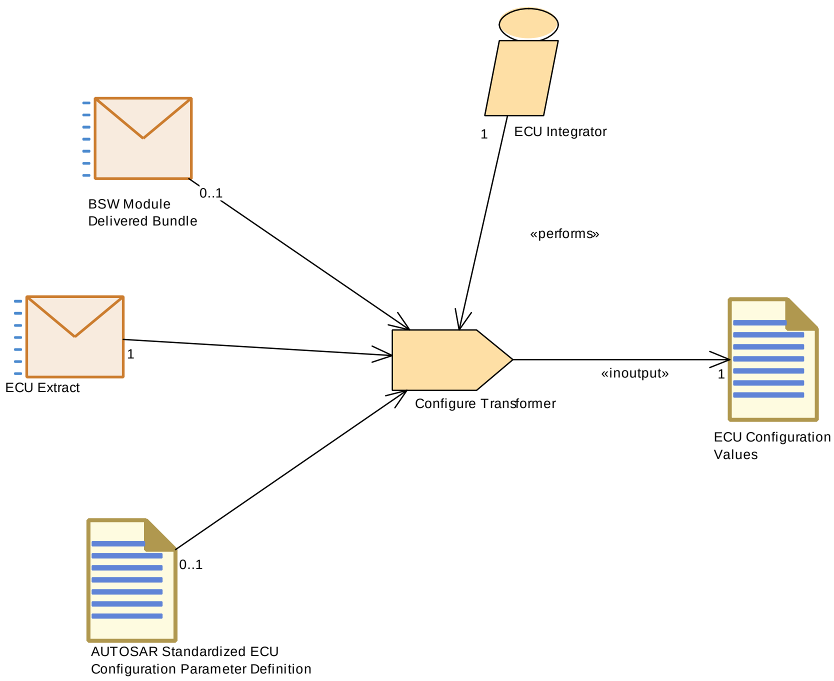

Figure 3.140: Configure Transformer

Task Definition	**Configure Transformer**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description			
Description	Configure the Transformer modules for this ECU.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
In/out	ECU Configuration Values	1	
	ECU Extract	1	
	AUTOSAR Standardized ECU Configuration Parameter Definition	0..1	
	BSW Module Delivered Bundle	0..1	

Relation Type	Related Element	Mul.	Note

Table 3.278: Configure Transformer

3.6.1.20 Generate BSW Configuration Code and Model Extensions

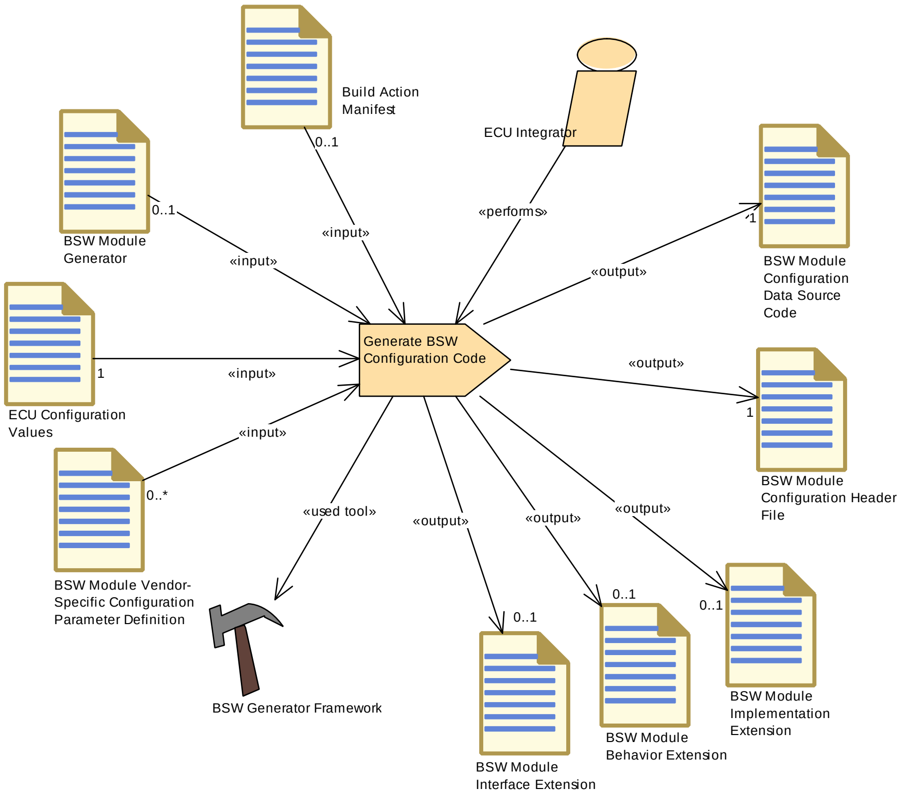

Figure 3.141: Generate BSW Code and model extensions

Task Definition	**Generate BSW Configuration Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Tasks		
Brief Description	Generate source code which implements configuration data for link- or compile-time configuration.		
Description	A generator reads the relevant parameters from the ECU Configuration Description and creates a separate code file that implements the specified configuration. This task is used for link-time configuration, i.e. the configuration code can be produced at link-time of the core code or for compile-time configuration, if the configuration code cannot be put into a header file (e.g. for tables), even if the core code and the configuration code shall be compiled at the same time. A header file may be produced in addition, to declare the data. Furthermore the generator may produce extensions of the BSW module description artifacts as a result of configuration parameter values which are set at integration time.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Configuration Values	1	
Consumes	BSW Module Generator	0..1	This is an input in case a generator framework is used which has to run some module specific generator code.
Consumes	Build Action Manifest	0..1	The task may be controlled by a Build Action Manifest.
Consumes	BSW Module Vendor-Specific Configuration Parameter Definition	0..*	
Produces	BSW Module Configuration Data Source Code	1	
Produces	BSW Module Configuration Header File	1	
Produces	BSW Module Behavior Extension	0..1	
Produces	BSW Module Implementation Extension	0..1	
Produces	BSW Module Interface Extension	0..1	
Used tool	BSW Generator Framework	1	

Table 3.279: Generate BSW Configuration Code

3.6.1.21 Generate Local MC Data Support

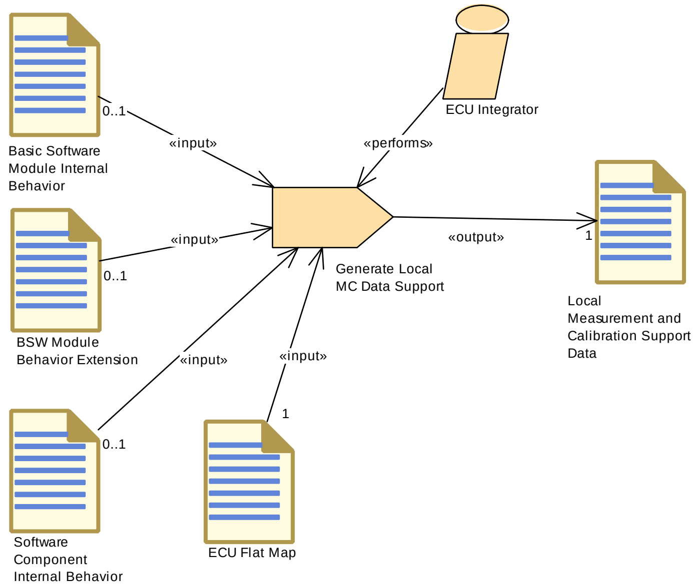

Figure 3.142: Generate Local MC Data Support

Task Definition	**Generate Local MC Data Support**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Generate Local MC Support Data		
Description	Generate the support data needed for measurement and calibration of those parameters and variables (roles constantMemory and staticMemory), which are owned locally by the code of a module or component (in contrast to those, which are owned by the RTE). The declaration of local variables/parameters is read from the Internal Behavior of either a BSW module or an Atomic Software Component, therefore these can be considered as alternative inputs.The ECU Flat Map is needed as input in order to resolve possible name conflicts. This task can be combined with RTE generation for practical reasons, but it is considered as an independent task. Note that calibration data that need software emulation support by the RTE cannot be handled by this task; they need to be processed by the task Generate RTE. Meth.bindingTime = CodeGenerationTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Flat Map	1	Meth.bindingTime = SystemDesignTime
Consumes	BSW Module Behavior Extension	0..1	Meth.bindingTime = SystemDesignTime
Consumes	Basic Software Module Internal Behavior	0..1	Meth.bindingTime = SystemDesignTime

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Software Component Internal Behavior	0..1	Meth.bindingTime = SystemDesignTime
Produces	Local Measurement and Calibration Support Data	1	Meth.bindingTime = CodeGenerationTime

Table 3.280: Generate Local MC Data Support

3.6.1.22 Create MC Function Model

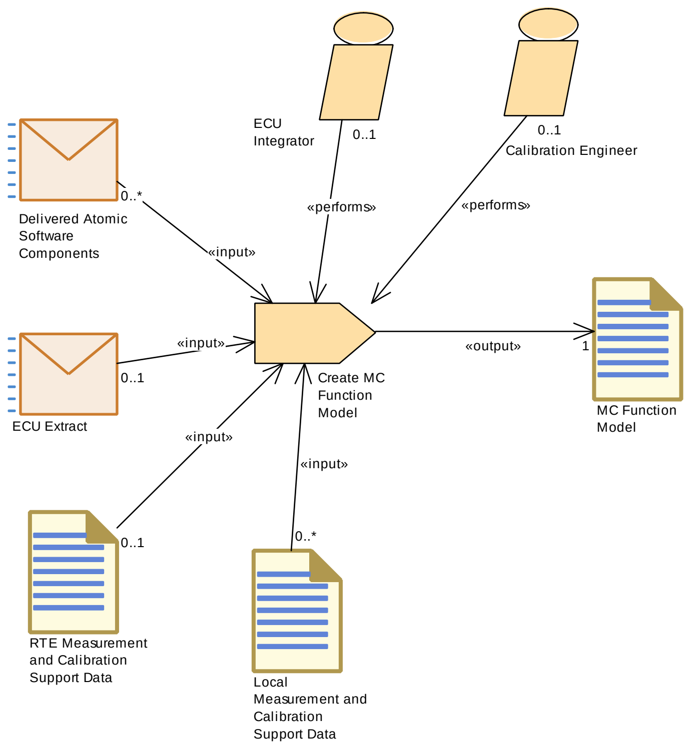

Figure 3.143: Create MC Function Model

Task Definition	**Create MC Function Model**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Define a model of McFunctions.		
Description	Create (manually or by generator) a functional model of measurement and calibration data on an ECU. Such a model may be derived from the logical structure of software components, ports etc. but the rules for this transformation are not standardized. This task may be performed before the RTE code is generated. Then the model will be based on the data defined in the ECU Flat Map. The task may also be performed at the same time as or after the generation of Measurement and Calibration Support Data. In this case it is possible (but not mandatory) to base the model on these support data only. The task may be supported by the RTE generator (not a standardized feature) or another tool.		
Relation Type	*Related Element*	*Mul.*	*Note*
Performed by	Calibration Engineer	0..1	
Performed by	ECU Integrator	0..1	
Consumes	ECU Extract	0..1	The ECU Flat Map can be used to define references to variables and parameters which are later visible in A2L. Furthermore, the ECU Extract can be used to find the relevant software components.
Consumes	RTE Measurement and Calibration Support Data	0..1	Used if the MC Function Model shall refer to McDataInstances allocated by the RTE.
Consumes	Delivered Atomic Software Components	0..*	The component model may be used to derive an MC Function Model.
Consumes	Local Measurement and Calibration Support Data	0..*	Used if the MC Function Model shall refer to McDataInstances allocated by BSW modules without RTE support.
Produces	MC Function Model	1	

Table 3.281: Create MC Function Model

3.6.1.23 Generate RTE

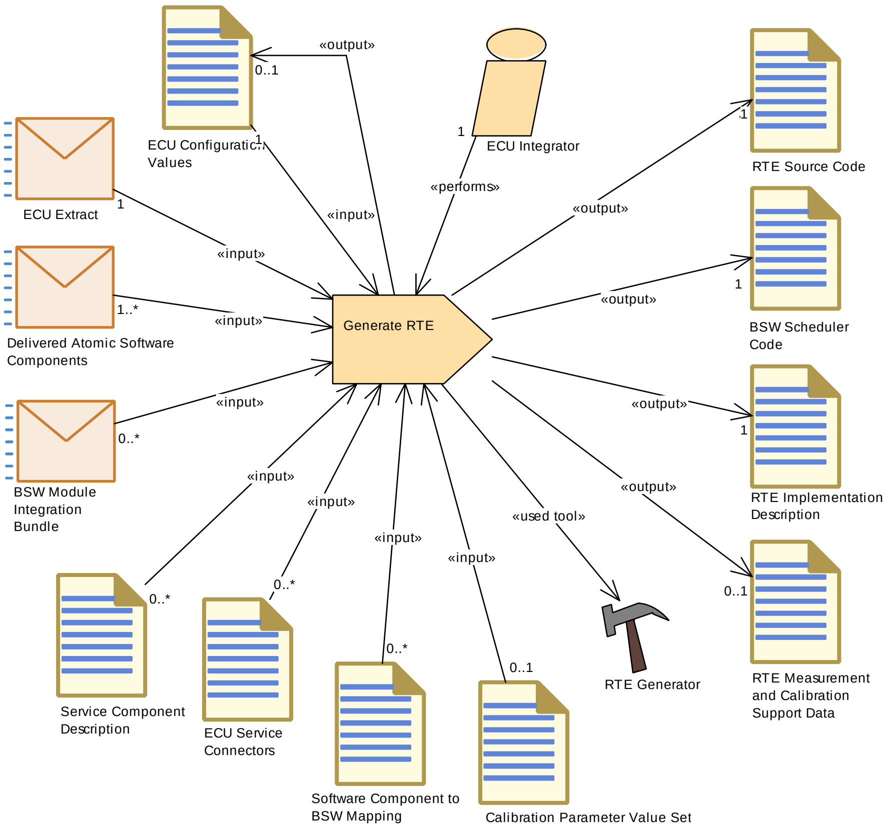

Figure 3.144: Generate RTE

Task Definition	**Generate RTE**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Generate the RTE and several further artifacts.		
Description	Generate the RTE and several further artifacts from the input XML descriptions in the scope of a given ECU: • RTE Core Source Code • BSW Scheduler Code • RTE Implementation Description • RTE Measurement and Calibration Support Data In an optional mode, this task can also write into the ECU configuration, especially for the configuration of the OS. This mode is used to pre-configure parts of the ECU configuration. It shall support the integrator in setting up the configuration in an iterative way. In the so-called strict mode, the ECU configuration is not changed but assumed to be complete. This mode shall be used before the final build. A PredefinedVariant in the input data (referred in the EcuC configuration, see task Configure EcuC) can be used to bind variation points at code generation time. For variation points with latest binding time "code generation time" this is mandatory. Unbound variation points can still be present in the generated code. Meth.bindingTime = CodeGenerationTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Configuration Values	1	Meth.bindingTime = SystemDesignTime
Consumes	ECU Extract	1	Find the VFB description of all Atomic Software Components on this ECU and the relevant parts of the system description. The ECU Flat Map is also an input. Meth.bindingTime = SystemDesignTime
Consumes	Delivered Atomic Software Components	1..*	Required input: • References to all component implementation descriptions on this ECU • SwcInternalBehavior which was used in the contract phase of the software components on this ECU • (optional) Software Component to BSW Mapping Meth.bindingTime = SystemDesignTime
Consumes	Calibration Parameter Value Set	0..1	Meth.bindingTime = SystemDesignTime

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	BSW Module Integration Bundle	0..*	Input for BSW scheduling, BSW mode and trigger declaration, BSW exclusive areas, BSW calibration parameters that need RTE support (for software emulation). Optionally, a Build Action Manifest maybe be used to control the generator steps. Meth.bindingTime = SystemDesignTime
Consumes	ECU Service Connectors	0..*	Meth.bindingTime = SystemDesignTime
Consumes	Service Component Description	0..*	Meth.bindingTime = SystemDesignTime
Consumes	Software Component to BSW Mapping	0..*	This input is explicitly stated because the mapping may be created during ECU integration and thus is not necessarily part of the Delivered Atomic Software Components. Meth.bindingTime = SystemDesignTime
Produces	BSW Scheduler Code	1	Meth.bindingTime = CodeGenerationTime
Produces	RTE Implementation Description	1	Meth.bindingTime = CodeGenerationTime
Produces	RTE Source Code	1	Meth.bindingTime = CodeGenerationTime
Produces	ECU Configuration Values	0..1	Optional output for the configuration of the OS. Meth.bindingTime = CodeGenerationTime
Produces	RTE Measurement and Calibration Support Data	0..1	Meth.bindingTime = CodeGenerationTime
Used tool	RTE Generator	1	

Table 3.282: Generate RTE

3.6.1.24 Generate Scheduler

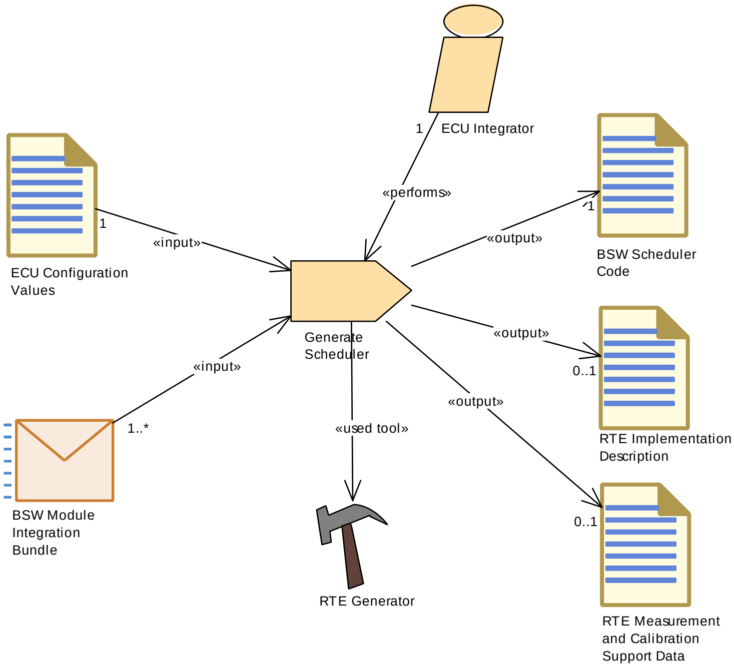

Figure 3.145: Generate Scheduler

Task Definition	**Generate Scheduler**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Generate the BSW Scheduler		
Description	Optional task of the RTE generator which only produces the code of the BSW Scheduler and related artifacts. Meth.bindingTime = CodeGenerationTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Configuration Values	1	Configuration values for the BSW Scheduler (subset of RTE configuration). Meth.bindingTime = SystemDesignTime
Consumes	BSW Module Integration Bundle	1..*	Input for BSW scheduling, BSW mode and trigger declaration, BSW exclusive areas, BSW calibration parameters that need support for software emulation. Optionally, a Build Action Manifest maybe be used to control the generator steps. Meth.bindingTime = SystemDesignTime
Produces	BSW Scheduler Code	1	Meth.bindingTime = CodeGenerationTime

Relation Type	***Related Element***	***Mul.***	***Note***
Produces	RTE Implementation Description	0..1	Creates a subset of the RTE implementation description that contains only the description of data owned by the BSW Scheduler. Meth.bindingTime = CodeGenerationTime
Produces	RTE Measurement and Calibration Support Data	0..1	Creates a subset of the measurement & calibration support data related only to the data owned by the BSW Scheduler. Meth.bindingTime = CodeGenerationTime
Used tool	RTE Generator	1	

Table 3.283: Generate Scheduler

3.6.1.25 Generate OS

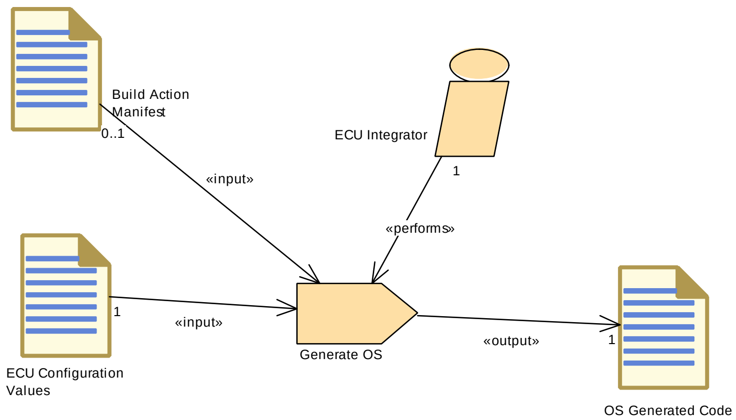

Figure 3.146: Generate OS

Task Definition	**Generate OS**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Generate the OS Generated Code files		
Description	Generate the OS Generated Code files using the OS configuration values from the ECU Configuration . Meth.bindingTime = CodeGenerationTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Configuration Values	1	Meth.bindingTime = SystemDesignTime
Consumes	Build Action Manifest	0..1	The task may be controlled by a Build Action Manifest.
Produces	OS Generated Code	1	Meth.bindingTime = CodeGenerationTime

Table 3.284: Generate OS

3.6.1.26 Generate RTE Prebuild Dataset

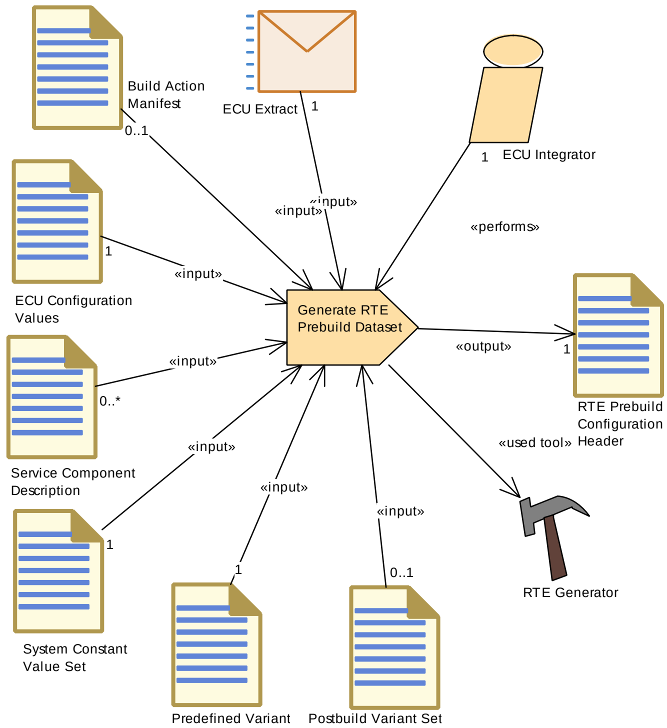

Figure 3.147: Generate RTE Prebuild Dataset

Task Definition	**Generate RTE Prebuild Dataset**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Prebuild Data Set Generation Phase for the RTE: It binds all variations which are later than code generation time		
Description	Prebuild Data Set Generation Phase for the RTE: It binds all variations which are later than code generation time but before build time. The output is a configuration header which is used for the build. The actually supported variant are defined by the PredefinedVariant referred in the EcuC configuration (see task Configure EcuC). Meth.bindingTime = PreCompileTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Configuration Values	1	find the Predefiined Variant to be used Meth.bindingTime = CodeGenerationTime
Consumes	ECU Extract	1	Meth.bindingTime = CodeGenerationTime
Consumes	Predefined Variant	1	
Consumes	System Constant Value Set	1	
Consumes	Build Action Manifest	0..1	The task may be controlled by a Build Action Manifest.

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Postbuild Variant Set	0..1	
Consumes	Service Component Description	0..*	Meth.bindingTime = CodeGenerationTime
Produces	RTE Prebuild Configuration Header	1	Meth.bindingTime = PreCompileTime
Used tool	RTE Generator	1	

Table 3.285: Generate RTE Prebuild Dataset

3.6.1.27 Compile ECU Source Code

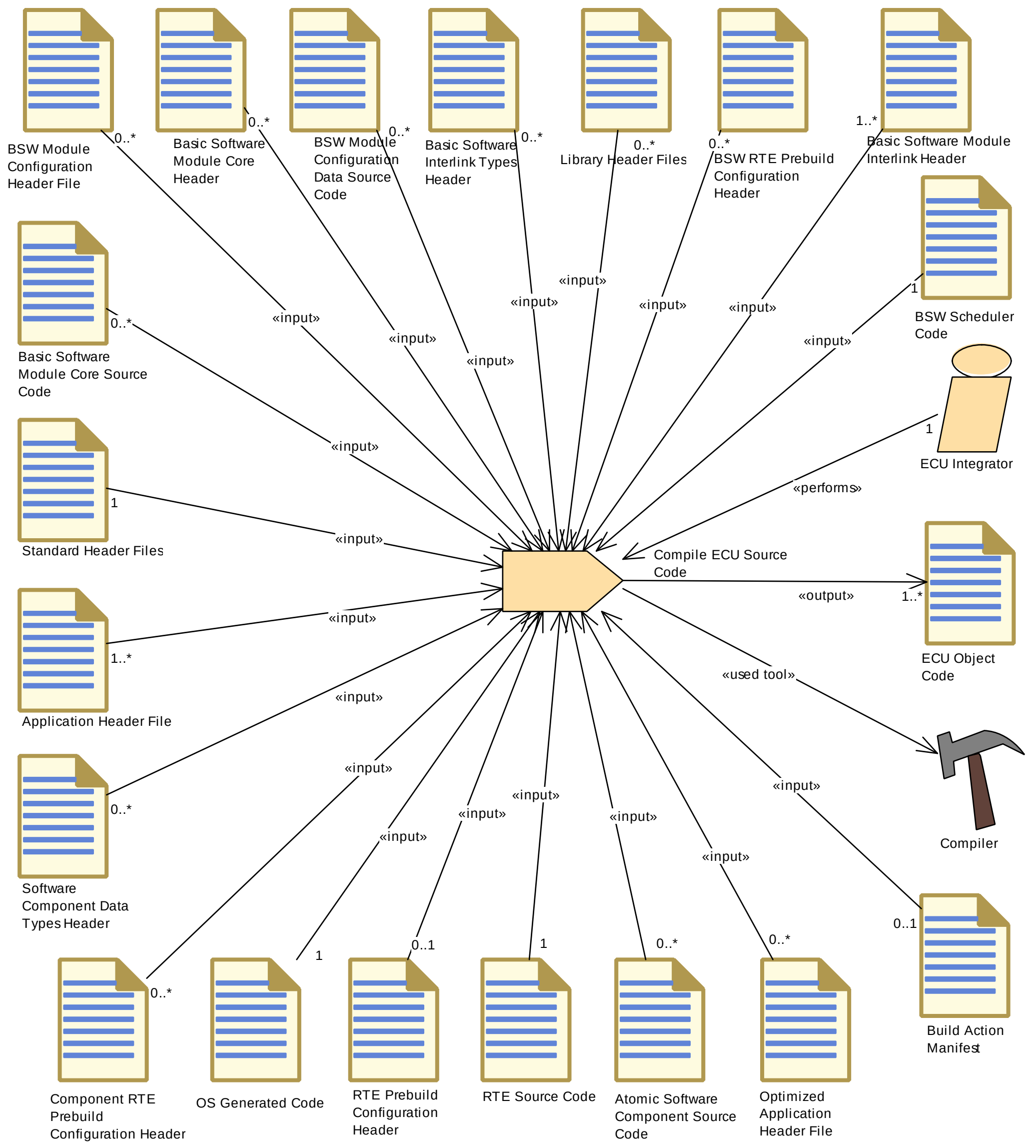

Figure 3.148: Compile ECU Source Code

Task Definition	**Compile ECU Source Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Compile Source Code for an ECU		
Description	Compile all the source code required for ECU integration, i.e. all source code except the code which is delivered as object code. Meth.bindingTime = CompileTime		
Relation Type	**Related Element**	**Mul.**	**Note**
Performed by	ECU Integrator	1	
Consumes	BSW Scheduler Code	1	Meth.bindingTime = CodeGenerationTime
Consumes	OS Generated Code	1	Meth.bindingTime = CodeGenerationTime
Consumes	RTE Source Code	1	Meth.bindingTime = CodeGenerationTime
Consumes	Standard Header Files	1	Meth.bindingTime = CodeGenerationTime
Consumes	Application Header File	1..*	Meth.bindingTime = CodeGenerationTime
Consumes	Basic Software Module Interlink Header	1..*	Meth.bindingTime = CodeGenerationTime
Consumes	Build Action Manifest	0..1	The task may be controlled by a Build Action Manifest.
Consumes	RTE Prebuild Configuration Header	0..1	Meth.bindingTime = PreCompileTime
Consumes	Atomic Software Component Source Code	0..*	Meth.bindingTime = CodeGenerationTime
Consumes	BSW Module Configuration Data Source Code	0..*	Meth.bindingTime = CodeGenerationTime
Consumes	BSW Module Configuration Header File	0..*	Meth.bindingTime = CodeGenerationTime
Consumes	BSW RTE Prebuild Configuration Header	0..*	Meth.bindingTime = PreCompileTime
Consumes	Basic Software Interlink Types Header	0..*	Meth.bindingTime = CodeGenerationTime
Consumes	Basic Software Module Core Header	0..*	Meth.bindingTime = CodeGenerationTime
Consumes	Basic Software Module Core Source Code	0..*	Meth.bindingTime = CodeGenerationTime
Consumes	Component RTE Prebuild Configuration Header	0..*	Meth.bindingTime = CodeGenerationTime

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Library Header Files	0..*	Meth.bindingTime = CodeGenerationTime
Consumes	Optimized Application Header File	0..*	Meth.bindingTime = CodeGenerationTime
Consumes	Software Component Data Types Header	0..*	Meth.bindingTime = CodeGenerationTime
Produces	ECU Object Code	1..*	Meth.bindingTime = CompileTime
Used tool	Compiler	1	

Table 3.286: Compile ECU Source Code

3.6.1.28 Generate ECU Executable

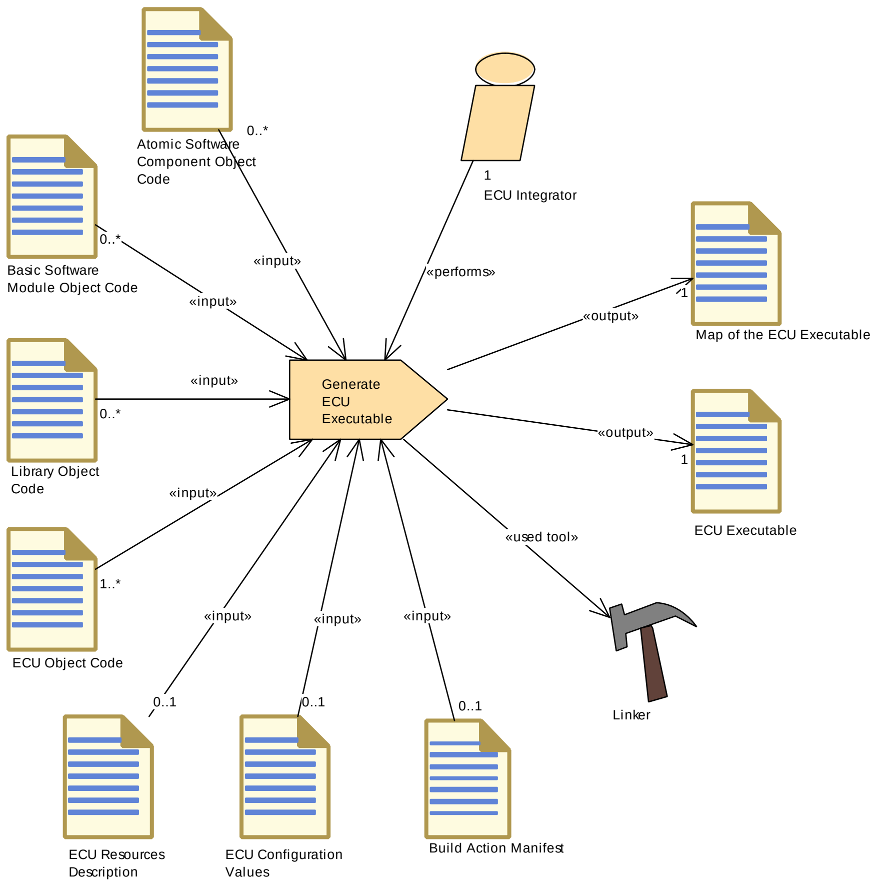

Figure 3.149: Generate ECU Executable

Task Definition	**Generate ECU Executable**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Generate the executable code of the ECU out of the object files and linker configuration.		
Description	The steps to generate the code for an ECU resemble today's development practice. However, it is important to note that this activity is more than a simple linker step. Information from the ECU Configuration Description might be used to generate specially configured executable software. The ECU Configuration Description is needed as input to the Generate Executable activity, because it contains the information which BSW modules and SWC implementations are used to create the executable and further information about the memory mapping. The output of this activity is the ECU Executable and the Map of Executable (which is typically the log file from linking the ECU Executable). The detailed input and output formats of this task are not standardized by AUTOSAR, therefore this task is only included for informative purposes. Note that ECU Configuration is shown as an input to get the overall picture, however in practice more specific artifacts (e.g. linker settings, make file etc.) will have to be generated out of the ECU configuration before the actual software build can be started. Especially, the information about the mapping of the physical memory sections to the memory section used in the software, which is described in the so-called EcuC parameter values, is needed in order to generate the linker settings. Meth.bindingTime = LinkTime		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Object Code	1..*	from generated or delivered source code Meth.bindingTime = CompileTime
Consumes	Build Action Manifest	0..1	The task may be controlled by a Build Action Manifest.
Consumes	ECU Configuration Values	0..1	may be used to set up build environment Meth.bindingTime = CompileTime
Consumes	ECU Resources Description	0..1	may be used to set up build environment Meth.bindingTime = CompileTime
Consumes	Atomic Software Component Object Code	0..*	Meth.bindingTime = CompileTime
Consumes	Basic Software Module Object Code	0..*	for object code delivery Meth.bindingTime = CompileTime
Consumes	Library Object Code	0..*	for object code delivery Meth.bindingTime = CompileTime
Produces	ECU Executable	1	Meth.bindingTime = LinkTime
Produces	Map of the ECU Executable	1	Meth.bindingTime = LinkTime
Used tool	Linker	1	

Relation Type	***Related Element***	***Mul.***	***Note***
Predecessor	Encapsulate SW-C	1	
Predecessor	Generate BSW and RTE	1	

Table 3.287: Generate ECU Executable

3.6.1.29 Generate RTE Postbuild Dataset

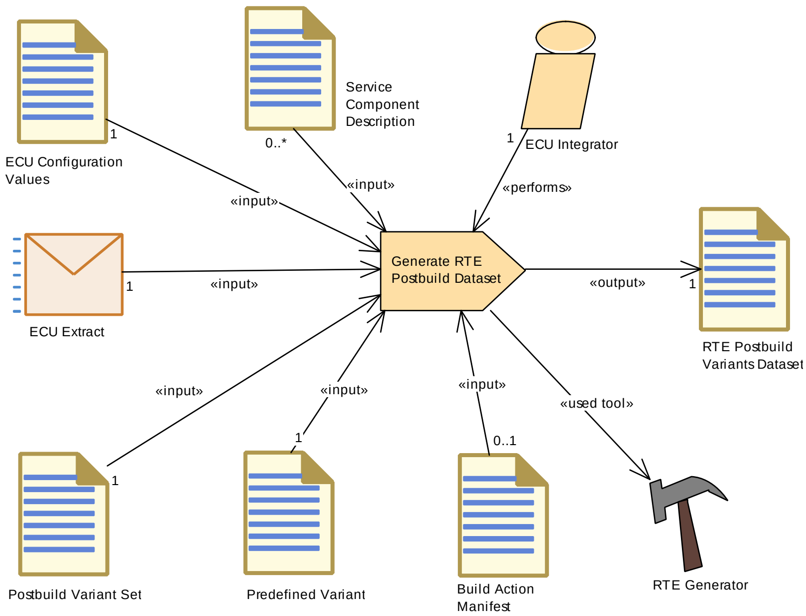

Figure 3.150: Generate RTE Postbuild Dataset

Task Definition	**Generate RTE Postbuild Dataset**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Postbuild Data Set Generation Phase for the RTE: It binds all variations which are for postbuild time.		
Description	Data Set Generation Phase for the RTE: It binds all variations which are for postbuild time. The output is a data set which can be used to build an image separately from the main code. The supported post-build variants are defined by the PredefinedVariants referred in the post-build section of the RTE configuration. At runtime, only one of those variants can be active. This selection is done via the initialization structure for the BSW Scheduler. The actual value for this iniialization structure used for runtime initialization is defined by the configuration of the ECU State Manager. Meth.bindingTime = PostBuild		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Configuration Values	1	Meth.bindingTime = LinkTime

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	ECU Extract	1	Meth.bindingTime = LinkTime
Consumes	Postbuild Variant Set	1	
Consumes	Predefined Variant	1	
Consumes	Build Action Manifest	0..1	The task may be controlled by a Build Action Manifest.
Consumes	Service Component Description	0..*	Meth.bindingTime = LinkTime
Produces	RTE Postbuild Variants Dataset	1	Meth.bindingTime = PostBuild
Used tool	RTE Generator	1	

Table 3.288: Generate RTE Postbuild Dataset

3.6.1.30 Generate A2L

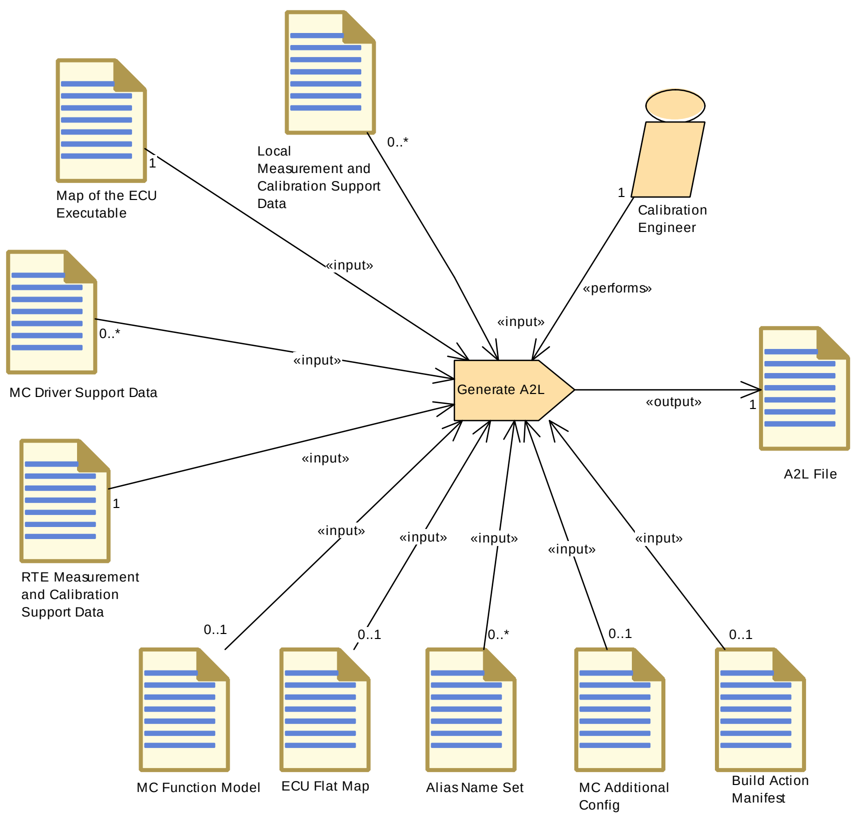

Figure 3.151: Generate A2L

Task Definition	**Generate A2L**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Generate the A2L File for an ECU.		
Description	The A2L File created by this task is the final representation of the data given by RTE Measurement and Calibration Support Data and Local Measurement and Calibration Support Data. The main purpose of this task is to replace all symbolic information on data location found in these input data by actual addresses. Optionally, it replaces identifiers by alias names given in Alias Name Set(s). Finally is completes the A2L file with configuration from ECU driver software (MC Driver Support Data) and configuration not determined by AUTOSAR artifacts (MC Additional Configuration). This task is not part of AUTOSAR, it is only included for completeness of the use cases. The Map of the ECU Executable (linker map file) is shown as input in order to illustrate the principle use case only. Note that one needs additional information, like the .ELF or .COFF file, to resolve addresses of elements of composite C-variables.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Calibration Engineer	1	
Consumes	Map of the ECU Executable	1	
Consumes	RTE Measurement and Calibration Support Data	1	
Consumes	Build Action Manifest	0..1	The task may be controlled by a Build Action Manifest.
Consumes	ECU Flat Map	0..1	The ECU Flat Map is needed in case the A2L generator has to process an MC Function Model that relates to data in the ECU Flat Map.
Consumes	MC Additional Config	0..1	
Consumes	MC Function Model	0..1	This input is needed if the keyword FUNCTION shall be supported in the generated A2L.
Consumes	Alias Name Set	0..*	
Consumes	Local Measurement and Calibration Support Data	0..*	
Consumes	MC Driver Support Data	0..*	
Produces	A2L File	1	Meth.bindingTime = CodeGenerationTime

Table 3.289: Generate A2L

3.6.1.31 Measure Resources

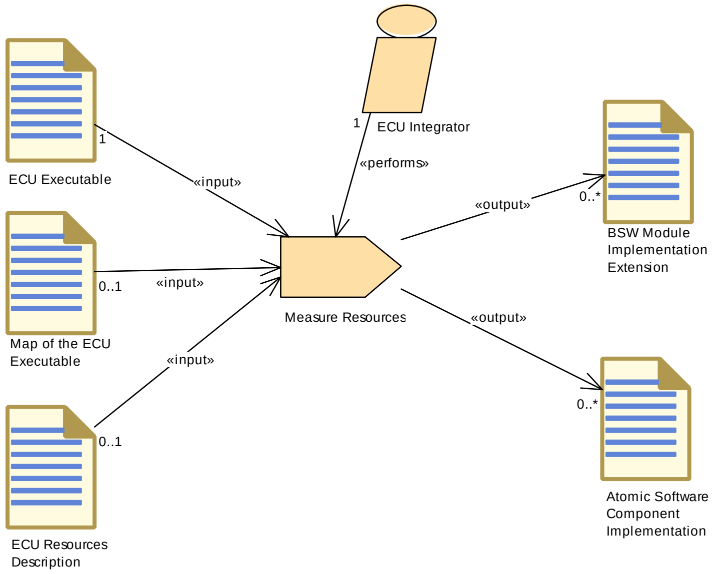

Figure 3.152: Measure Resources

Task Definition	**Measure Resources**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description	Measure the resource consumption and update the implementation section of the Application SWC and BSW Module Descriptions.		
Description	Measure the resource consumption and update the implementation section of the Application SWC and BSW Module Descriptions.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Executable	1	
Consumes	ECU Resources Description	0..1	
Consumes	Map of the ECU Executable	0..1	
Produces	Atomic Software Component Implementation	0..*	Add extensions to the Implementation Description. Meth.bindingTime = PostBuild
Produces	BSW Module Implementation Extension	0..*	Meth.bindingTime = PostBuild

Table 3.290: Measure Resources

3.6.1.32 Refine Rapid Prototyping Scenario

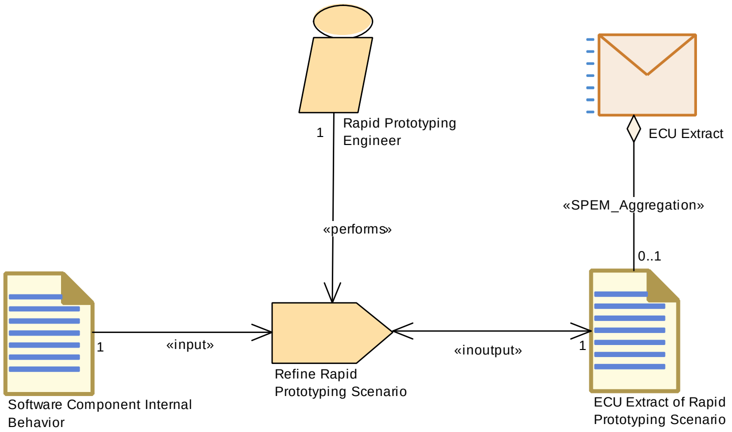

Figure 3.153: Refine Rapid Prototyping Scenario

Task Definition	**Refine Rapid Prototyping Scenario**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Tasks		
Brief Description			
Description	Add missing ECU specific information in the Rapid Prototyping Scenario, e.g. missing RptHooks or hook implementation decisions.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Rapid Prototyping Engineer	1	
Consumes	Software Component Internal Behavior	1	
In/out	ECU Extract of Rapid Prototyping Scenario	1	
Predecessor	Generate ECU Extract	1	

Table 3.291: Refine Rapid Prototyping Scenario

3.6.2 Work Products

3.6.2.1 BSW Module Integration Bundle

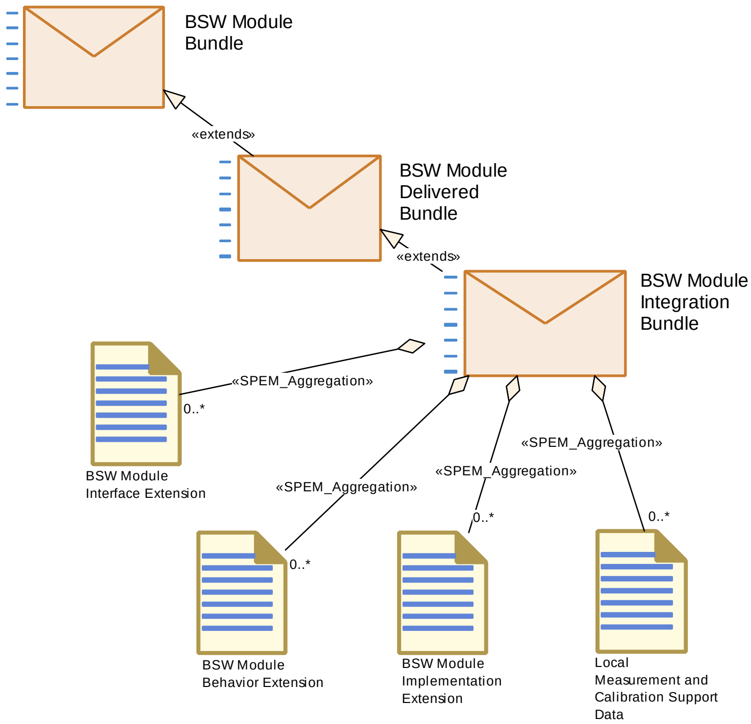

Figure 3.154: BSW Module Integration Bundle

Deliverable	**BSW Module Integration Bundle**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description			
Description	Contains the BSW artifacts for one or more BSW modules completed during integration.		
Kind	Delivered		
Extends	BSW Module Delivered Bundle		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregates	BSW Module Behavior Extension	0..*	
Aggregates	BSW Module Implementation Extension	0..*	
Aggregates	BSW Module Interface Extension	0..*	
Aggregates	Local Measurement and Calibration Support Data	0..*	

Relation Type	*Related Element*	*Mul.*	*Note*
Consumed by	Generate Scheduler	1..*	Input for BSW scheduling, BSW mode and trigger declaration, BSW exclusive areas, BSW calibration parameters that need support for software emulation. Optionally, a Build Action Manifest maybe be used to control the generator steps. Meth.bindingTime = SystemDesignTime
Consumed by	Generate RTE	0..*	Input for BSW scheduling, BSW mode and trigger declaration, BSW exclusive areas, BSW calibration parameters that need RTE support (for software emulation). Optionally, a Build Action Manifest maybe be used to control the generator steps. Meth.bindingTime = SystemDesignTime

Table 3.292: BSW Module Integration Bundle

3.6.2.2 ECU Software Delivered

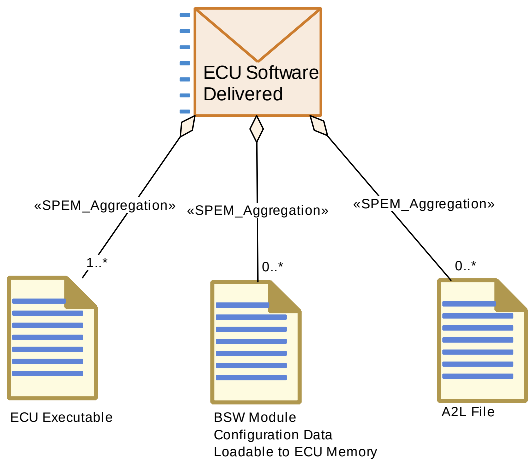

Figure 3.155: ECU Software Delivered

Deliverable	**ECU Software Delivered**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	All the work products that form the deliverable of an AUTOSAR ECU.		
Description	All the work products that form the deliverable of an AUTOSAR ECU software build. ECU in this context means processor, so if an electronic control unit consists of several processors, one "ECU Software Delivered" will be needed for each processor. Note that the detailed format for all parts of this deliverable is not defined by AUTOSAR.		
Kind	Delivered		
Relation Type	**Related Element**	**Mul.**	**Note**
Aggregates	ECU Executable	1..*	
Aggregates	A2L File	0..*	
Aggregates	BSW Module Configuration Data Loadable to ECU Memory	0..*	
Produced by	Integrate Software for ECU	1	

Table 3.293: ECU Software Delivered

3.6.2.3 Service Component Description

Artifact	**Service Component Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	Describes the RTE relevant part of an AUTOSAR Service on a given ECU in form of a ServcieComponentType with all its ports and an internal behavior.		
Description	Describes the RTE relevant part of an AUTOSAR Service on a given ECU in form of a ServiceComponentType with all its ports and an internal behavior. This artifact must be generated during the ECU configuration process, latest before the RTE is generated. It depends on the needs of the software components for this AUTOSAR Service.		
Kind	AUTOSAR XML		
Relation Type	**Related Element**	**Mul.**	**Note**
Produced by	Create Service Component	1	Meth.bindingTime = SystemDesignTime
Consumed by	Connect Service Component	1	Required in order to define the connector links to the ports on the BSW side.

Relation Type	Related Element	Mul.	Note
Consumed by	Configure NvM	0..*	The configuration of diagnostics, especially of the DEM, typically leads to the definition of additional data to be stored in NvM. One possibility to handle this is to create ServiceNeeds on the level ServiceComponentType which is then taken into account for the configuration of the NvM.
Consumed by	Configure RTE	0..*	The Internal Behavior of Service Components contributes to the RTE configuration.
Consumed by	Generate RTE	0..*	Meth.bindingTime = SystemDesignTime
Consumed by	Generate RTE Postbuild Dataset	0..*	Meth.bindingTime = LinkTime
Consumed by	Generate RTE Prebuild Dataset	0..*	Meth.bindingTime = CodeGenerationTime
Use meta model element	ServiceSwComponentType	1	
Use meta model element	SwcInternalBehavior	1	

Table 3.294: Service Component Description

3.6.2.4 ECU Service Connectors

Artifact	**ECU Service Connectors**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	The conectors to the Service Components which complete the complete Software Composition predefined in the ECU extract.		
Description	The assembly connectors to the Service Components which complete the Software Composition predefined in the ECU extract. These connectores are added during ECU integration as a separate artifact to the already defined composition of Atomic Software Components.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Connect Service Component	1..*	Meth.bindingTime = SystemDesignTime
Consumed by	Define ECU Timing	1..*	
Consumed by	Generate RTE	0..*	Meth.bindingTime = SystemDesignTime
Use meta model element	AssemblySw Connector	1	

Table 3.295: ECU Service Connectors

3.6.2.5 ECU Timing

Artifact	**ECU Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	TimingDescription and TimingConstraints for a concrete ECU		
Description	TimingDescription and TimingConstraints defined for a concrete ECU taking the ECU configuration and the ECU Software Composition (including their implementation) into account.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define ECU Timing	1	Meth.bindingTime = SystemDesignTime
Consumed by	Configure OS	0..1	
Consumed by	Configure RTE	0..1	
Consumed by	Configure Watchdog Manager	0..1	
Consumed by	Create Service Component	0..1	Additional information for fine tuning configuration decisions.
Use meta model element	EcuTiming	1	

Table 3.296: ECU Timing

3.6.2.6 BSW Module Interface Extension

Artifact	**BSW Module Interface Extension**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description			
Description	Additions to the BSW Module on the interface level during integration. It is used for example to add Basic Software Module Entries in response to the ECU configuration, for example callback declarations.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	BSW Module Integration Bundle	0..*	
Produced by	Generate BSW Configuration Code	0..1	
Use meta model element	BswModuleDescription	1	
Use meta model element	BswModuleEntry	1	

Table 3.297: BSW Module Interface Extension

3.6.2.7 BSW Module Behavior Extension

Artifact	**BSW Module Behavior Extension**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description			
Description	Additions to the BSW Module on the behavior level during integration. It can for example be used to add local data declaration (constantMemory, staticMemory, perInstanceMemory) for debug or calibration purposes in response to configuration parameters.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	BSW Module Integration Bundle	0..*	
Produced by	Generate BSW Configuration Code	0..1	
Consumed by	Generate Local MC Data Support	0..1	Meth.bindingTime = SystemDesignTime
Use meta model element	BswInternalBehavior	1	

Table 3.298: BSW Module Behavior Extension

3.6.2.8 BSW Module Implementation Extension

Artifact	**BSW Module Implementation Extension**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description			
Description	Additions to the BSW Module on the implementation level during integration. It is used for example to add information on resource consumption.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	BSW Module Integration Bundle	0..*	
Produced by	Generate BSW Configuration Code	0..1	
Produced by	Measure Resources	0..*	Meth.bindingTime = PostBuild
Use meta model element	BswImplementation	1	

Table 3.299: BSW Module Implementation Extension

3.6.2.9 ECU Configuration Values

Artifact	**ECU Configuration Values**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	The collection of all configuration values for an ECU.		
Description	First of all, the ECU Configuration Values contain a link to the System element which comes with the ECU Extract thus it can be used as a root element for integration on this ECU. Furtheron, it contains a collection of all configuration values for an ECU, which is gradually filled. Starting with the root element EcucValueCollection it contains the actual configuration settings EcucModuleConfigurationValues for each module including the RTE. Note that due to their strong interrelation, these parts are not considered as separate artifacts in the use cases for ECU integration. A special set of configuration values is the so-called EcuC-configuration: It contains the configuration values which are relevant for the whole ECU. Tools that interpret the configuration values need to know the underlying parameter definition. Therefore, in addition to the configuration values, each EcucValueCollection contains a link and the version of the parameter definition to which it adheres. This parameter definition is either part of the AUTOSAR Standardized ECU Configuration Parameter Definition or, in case of vendor specific extensions, is given by the artifact Basic Software Module Vendor-Specific Configuration Parameter Definition.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Configure Memmap Allocation	1	MemMapAllocation: Meth.bindingTime = SystemDesignTime
Produced by	Create Service Component	1	Enter links to the created SwComponentPrototypes. Meth.bindingTime = SystemDesignTime
Produced by	Generate Base Ecu Configuration	1	Meth.bindingTime = SystemDesignTime
Produced by	Prepare ECU Configuration	1	
Produced by	Generate RTE	0..1	Optional output for the configuration of the OS. Meth.bindingTime = CodeGenerationTime
In/out	Configure BSW and RTE	1	
In/out	Configure Com	1	
In/out	Configure Debug	1	
In/out	Configure Diagnostics	1	Configuration Values for DEM, DCM, DLT, FIM.
In/out	Configure ECUC	1	
In/out	Configure IO Hardware abstraction	1	
In/out	Configure MCAL	1	

Relation Type	*Related Element*	*Mul.*	*Note*
In/out	Configure Mode Management	1	
In/out	Configure NvM	1	
In/out	Configure OS	1	
In/out	Configure RTE	1	
In/out	Configure Transformer	1	
In/out	Configure Watchdog Manager	1	
In/out	Generate Updated ECU Configuration	1	The task "Generate Updated ECU Configuration" consumes the initial ECU configuration values and produces the updated ECU configuration values.
Consumed by	Define ECU Timing	1	
Consumed by	Generate BS W Configuration Code	1	
Consumed by	Generate BSW Memory Mapping Header	1	MemMapAllocation: Mapping of the abstract sections (SwAddressMethods for generic mapping resp. MemorySection Elements for specific mapping) to the compiler specific MemMapAddressingModes. Meth.bindingTime = SystemDesignTime
Consumed by	Generate BSW Postbuild Configuration Code	1	
Consumed by	Generate BSW Precompile Configuration Header	1	
Consumed by	Generate BSW Source Code	1	
Consumed by	Generate BSW and RTE	1	
Consumed by	Generate OS	1	Meth.bindingTime = SystemDesignTime
Consumed by	Generate RTE	1	Meth.bindingTime = SystemDesignTime
Consumed by	Generate RTE Postbuild Dataset	1	Meth.bindingTime = LinkTime
Consumed by	Generate RTE Prebuild Dataset	1	find the Predefiined Variant to be used Meth.bindingTime = CodeGenerationTime
Consumed by	Generate SWC Memory Mapping Header	1	MemMapAllocation: Mapipng of the abstract sections (SwAddressMethods for generic mapping resp. MemorySection Elements for specific mapping) to the compiler specific MemMapAddressingModes. Meth.bindingTime = SystemDesignTime

Relation Type	Related Element	Mul.	Note
Consumed by	Generate Scheduler	1	Configuration values for the BSW Scheduler (subset of RTE configuration). Meth.bindingTime = SystemDesignTime
Consumed by	Create Service Component	0..1	The creation of Service Component details may depend on ECU configuration values, especially for the DCM.
Consumed by	Generate BSW Memory Mapping Header	0..1	moduleDescription: List of used BSW modules (EcucValueCollection.ecucValue.moduleDescription) Meth.bindingTime = SystemDesignTime
Consumed by	Generate Compiler Configuration	0..1	ModuleDescription: List of used BSW modules (EcucValueCollection.ecucValue.moduleDescription) Meth.bindingTime = SystemDesignTime
Consumed by	Generate Compiler Configuration	0..1	RteImplementationRef: Existence of SWCs could be identified by usage of the RTE ECU Configuration "RteSwComponentType.RteImplementationRef" Meth.bindingTime = SystemDesignTime
Consumed by	Generate ECU Executable	0..1	may be used to set up build environment Meth.bindingTime = CompileTime
Consumed by	Generate SWC Memory Mapping Header	0..1	RteImplementationRef: Existence of SWCs could be identified by usage of the RTE ECU Configuration "RteSwComponentType.RteImplementationRef" Meth.bindingTime = SystemDesignTime
Use meta model element	EcucModuleConfigurationValues	1	
Use meta model element	EcucValueCollection	1	

Table 3.300: ECU Configuration Values

3.6.2.10 RTE Implementation Description

Artifact	**RTE Implementation Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	Implementation Description for the RTE, generated by the RTE generator.		
Description	Implementation Description for the RTE, generated by the RTE generator. Uses the format of BswImplementation. This artifact is required to provide information for other generators and the build process, namely debugging information, memory section. It aggregates also the support data for measurement and calibration, which is considered as a separate artifact.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*

Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Generate RTE	1	Meth.bindingTime = CodeGenerationTime
Produced by	Generate Scheduler	0..1	Creates a subset of the RTE implementation description that contains only the description of data owned by the BSW Scheduler. Meth.bindingTime = CodeGenerationTime
Use meta model element	BswImplementation	1	

Table 3.301: RTE Implementation Description

3.6.2.11 RTE Prebuild Configuration Header

Artifact	**RTE Prebuild Configuration Header**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	RTE Prebuild Configuration Header File. It defines all variants for the RTE code which have to be bound later than code generation time but before build time.		
Description	RTE Prebuild Configuration Header File. It defines the setting of all variants for the RTE code (via macro code) which have to be bound later than code generation time but before build time.		
Kind	Bound Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Generate RTE Prebuild Dataset	1	Meth.bindingTime = PreCompileTime
Consumed by	Compile ECU Source Code	0..1	Meth.bindingTime = PreCompileTime

Table 3.302: RTE Prebuild Configuration Header

3.6.2.12 Calibration Parameter Value Set

Artifact	**Calibration Parameter Value Set**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	Calibration Parameter Value Setting		
Description	A set of calibration parameter values used to initialize the memory objects which implement calibration parameters. The values are specific for the software component instances in ECU scope. They will override any initial values defined for those parameters within the ECU Extract. The parameter values can be defined as ApplicationDataTypes or as ImplementationDataTypes which has several use cases. These two use cases are supported by the RTE generation phase: • Parameter values defined as ImplementationDataTypes can be used as instance specific initialization for calibration parameters within components as soon as the respective ImplementationDataTypes are available (which must be the case for RTE generation anyhow). • Parameter values defined as ApplicationDataTypes can be used as instance specific initialization for calibration parameters which are only defined with ApplicationDataTypes. The next case is not modelled within AUTOSAR in detail: • Parameter values defined as ApplicationDataTypes can be used to exchange initial values with the component vendor not publishing the transformation algorithm between ApplicationDataTypes and ImplementationDataTypes		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Provide RTE Calibration Dataset	1	
Consumed by	Generate RTE	0..1	Meth.bindingTime = SystemDesignTime
Use meta model element	CalibrationParameterValueSet	1	

Table 3.303: Calibration Parameter Value Set

3.6.2.13 MC Function Model

Artifact	**MC Function Model**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	A functional model to be used for A2L generation.		
Description	As set of nested McFunction elements to be used as input to generate A2L. Its purpose is to • assign calibration parameters to a logical function • assign measurement variables to a logical function • structure functions hierarchically It shall support the generation of the FUNCTION keyword and related elements defined in ASAM MCD-2 MC. An MC Function Model refers to the data descriptions in other AUTOSAR XML artifacts either via entries in the ECU Flat Map or via McDataInstances being part of Measurement and Calibration Support Data.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Create MC Function Model	1	
Consumed by	Generate A2L	0..1	This input is needed if the keyword FUNCTION shall be supported in the generated A2L.
Use meta model element	McFunction	1	

Table 3.304: MC Function Model

3.6.2.14 Local Measurement and Calibration Support Data

Artifact	**Local Measurement and Calibration Support Data**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	Generated artifact, which supports the later generation of "A2L"-files for measurement and calibration data which are owned locally by a component or module.		
Description	Generated artifact which is used as an input for the later generation of "A2L"-files for measurement and calibration. It relates the measurment and calibration data listed in the ECU FlatMap to the C-variables used locally within a component or module (this is relevant only valid for those parameters and variables, which are not implemented by the RTE) . In addition, it contains all configuration data which are relevant for the A2L generator (e.g. the access method to calibration data whithin a Complex Driver). This XML-artifact is linked via a (splitable) aggregation to the Implementation Description of the component or module, but it is considered as a separate artifact.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	BSW Module Integration Bundle	0..*	
Produced by	Generate Local MC Data Support	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Create MC Function Model	0..*	Used if the MC Function Model shall refer to McDataInstances allocated by BSW modules without RTE support.
Consumed by	Generate A2L	0..*	
Use meta model element	McSupportData	1	

Table 3.305: Local Measurement and Calibration Support Data

3.6.2.15 RTE Measurement and Calibration Support Data

Artifact	**RTE Measurement and Calibration Support Data**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	RTE generator output, which supports the later generation of "A2L"-files for the measurement and calibration data which are owned by the RTE.		
Description	RTE generator output, which is used as an input for the later generation of "A2L"-files for measurement and calibration. It relates the measurement and calibration data listed in the ECU FlatMap to the C-variables of the generated RTE code. For all these data it contains copies of the attributes which are relevant for A2L generation. In additions it contains all configuration data which are relevant for the A2L generator (namely the access method to calibration data which is supported by the RTE). This XML-artifact is linked via a (splitable) aggregation to the RTE Implementation Description, but is considered as a separate artifact. The most important attributes for each data instance are: • Its shortName copied from the ECU Flat Map to be used as identifier and for display by the MC system. • The category copied from the corresponding data type (ApplicationDataType if defined, otherwise ImplementationDataType) as far as applicable. • The symbol used in the programing language. It will be used to find out the actual memory address by the final generation tool with the help of linker generated information. • All aggregated and referred elements like CompuMethod or BaseType describing the data (with the exception of the Flat Map) are completely copied from "upstream" information. Therefore this artifact is a self-contained description which can be forwarded to the A2L generator without needing related descriptions.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Generate RTE	0..1	Meth.bindingTime = CodeGenerationTime
Produced by	Generate Scheduler	0..1	Creates a subset of the measurement & calibration support data related only to the data owned by the BSW Scheduler. Meth.bindingTime = CodeGenerationTime
Consumed by	Generate A2L	1	
Consumed by	Create MC Function Model	0..1	Used if the MC Function Model shall refer to McDataInstances allocated by the RTE.
Use meta model element	McSupportData	1	

Table 3.306: RTE Measurement and Calibration Support Data

3.6.2.16 RTE Source Code

Artifact	**RTE Source Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	Source code implementiing the RTE on a CPU.		
Description	Source code implementing the RTE on a CPU. The output of an RTE generator can consist of both generated code and configuration for "library" code that may be supplied as either object code or source code. Both configured and generated code reference standard definitions that are defined in one of two standardized header files: The RTE Header File and the Lifecycle Header File. These header files are not explicitly shown in the methodology, as in all tasks they appear with the RTE source code. For details refer to AUTOSAR_SWS_RTE.pdf. Apart from this, the file structure is not standardized, and therefore represented as one single artifact in the methodology. In general, the RTE code can be partitioned in several files. The partitioning depends on the RTE vendor's software design and generation strategy. Nevertheless it shall be possible to clearly identify code and header files which are part of the RTE module.		
Kind	Source Code		
Relation Type	**Related Element**	**Mul.**	**Note**
Produced by	Generate BSW and RTE	1	
Produced by	Generate RTE	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile ECU Source Code	1	Meth.bindingTime = CodeGenerationTime

Table 3.307: RTE Source Code

3.6.2.17 BSW Scheduler Code

Artifact	**BSW Scheduler Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	Generated Code implementing the BSW Scheduler.		
Description	Generated Code implementing the BSW Scheduler. It can be source or macro code.		
Kind	Source Code		
Relation Type	**Related Element**	**Mul.**	**Note**
Produced by	Generate RTE	1	Meth.bindingTime = CodeGenerationTime
Produced by	Generate Scheduler	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile ECU Source Code	1	Meth.bindingTime = CodeGenerationTime

Relation Type	*Related Element*	*Mul.*	*Note*

Table 3.308: BSW Scheduler Code

3.6.2.18 OS Generated Code

Artifact	**OS Generated Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	OS configuration generated code		
Description	OS configuration generated code. OS configuration code are composed of header and C files. These will be compiled with the source code in the build process (see Compile Source Code).		
Kind	Source Code		
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Generate OS	1	Meth.bindingTime = CodeGenerationTime
Consumed by	Compile ECU Source Code	1	Meth.bindingTime = CodeGenerationTime

Table 3.309: OS Generated Code

3.6.2.19 RTE Postbuild Variants Dataset

Artifact	**RTE Postbuild Variants Dataset**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	Generated code used to resolve postbuild variants in the RTE.		
Description	Generated code used to resolve postbuild variants in the RTE. It consists of a c-file and a header file: • The RTE generator must generate a Rte_PBCfg.c file containing the declarations and initializations of one or more RTE post build variants. Only one of these variants can be active at runtime. • The RTE generator shall generate in the Rte_PBCfg.h file the SchM_ConfigType type declaration of the predefined post build variants data structure. This header file must be used by other RTE modules to resolve their runtime variabilities.		
Kind	Bound Source Code		
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Generate RTE Postbuild Dataset	1	Meth.bindingTime = PostBuild

Table 3.310: RTE Postbuild Variants Dataset

3.6.2.20 ECU Object Code

Artifact	**ECU Object Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description			
Description	Object code file produced by compilation during ECU integration. To be distinguished from code files which are already delivered as object code for integration (see Basic Software Module Object Code or Atomic Software Component Object Code).		
Kind	Object Code		
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Compile ECU Source Code	1..*	Meth.bindingTime = CompileTime
Consumed by	Generate ECU Executable	1..*	from generated or delivered source code Meth.bindingTime = CompileTime
Consumed by	Link ECU Code during Link Time Configuration	1..*	

Table 3.311: ECU Object Code

3.6.2.21 ECU Executable

Artifact	**ECU Executable**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	The executable image containing all the fully integrated software ready to download to an ECU.		
Description	The executable image containing all the fully integrated software ready to download to an ECU. This work product and its format is not defined by AUTOSAR, it is only included for completeness of the use cases.		
Kind	Executable		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	ECU Software Delivered	1..*	
Produced by	Generate ECU Executable	1	Meth.bindingTime = LinkTime
Produced by	Link ECU Code after Precompile Configuration	1	
Produced by	Link ECU Code during Link Time Configuration	1	
Consumed by	Measure Resources	1	

Table 3.312: ECU Executable

3.6.2.22 Map of the ECU Executable

Artifact	**Map of the ECU Executable**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	Linker map file of the excecutable.		
Description	Linker map file of the excecutable. This work product and its format is not defined by AUTOSAR, it is only included for completeness of the use cases.		
Kind	Text		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Generate ECU Executable	1	Meth.bindingTime = LinkTime
Consumed by	Generate A2L	1	
Consumed by	Measure Resources	0..1	

Table 3.313: Map of the ECU Executable

3.6.2.23 A2L File

Artifact	**A2L File**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	Input file for measurment and calibration tools.		
Description	Input file for measurement and calibration tools related to one ECU. This format is not in the scope of AUTOSAR, it is defined by the ASAM organization. The work product is only included for completeness of the use cases.		
Kind	Text		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	ECU Software Delivered	0..*	
Produced by	Generate A2L	1	Meth.bindingTime = CodeGenerationTime

Table 3.314: A2L File

3.6.2.24 MC Driver Support Data

Artifact	**MC Driver Support Data**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	Support data describing the specific access of a driver (e.g. XCP) for exchange of data for measurement and calibration.		
Description	Support data describing the specific access method of a driver (e.g. XCP) in order to exchange data for measurement and calibration. These are the so-called IF-DATA needed in the A2L files. This artifact shall be generated by a driver(e.g. XCP) specific generator out of its ECU configuration. This format is not defined by AUTOSAR. The work product is only included for completeness of the use cases.		
Kind	Custom		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumed by	Generate A2L	0..*	

Table 3.315: MC Driver Support Data

3.6.2.25 MC Additional Config

Artifact	**MC Additional Config**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Work Products		
Brief Description	External configuration data nedded to generate the A2L file.		
Description	Additional configuration data needed to generate the A2L file. This format is not defined by AUTOSAR. The work product is only included for completeness of the use cases.		
Kind	Custom		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumed by	Generate A2L	0..1	

Table 3.316: MC Additional Config

3.6.3 Tools

3.6.3.1 RTE Generator

Tool	**RTE Generator**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Guidance		
Brief Description			
Description	RTE Generator used for several tasks during ECU integration.		
Kind			
Relation Type	*Related Element*	*Mul.*	*Note*
Used	Generate RTE	1	
Used	Generate RTE Postbuild Dataset	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Used	Generate RTE Prebuild Dataset	1	
Used	Generate Scheduler	1	

Table 3.317: RTE Generator

3.6.3.2 BSW Generator Framework

Tool	**BSW Generator Framework**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::Guidance		
Brief Description			
Description	Framework that uses BSW generators that are being delivered as part of individual modules.		
Kind			
Relation Type	*Related Element*	*Mul.*	*Note*
Used	Generate BSW Configuration Code	1	

Table 3.318: BSW Generator Framework

3.6.4 ECU Config Classes

3.6.4.1 Tasks

3.6.4.1.1 Compile Unconfigured Bsw

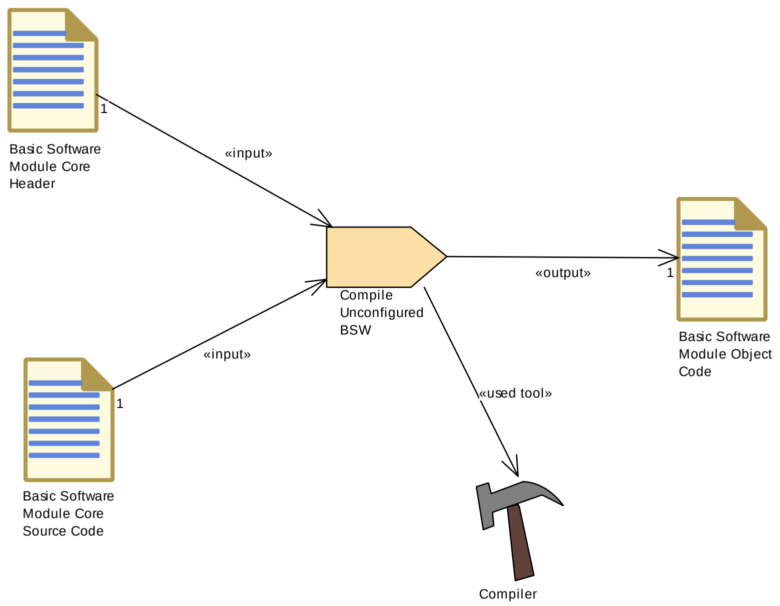

Figure 3.156: Compile Unconfigured Bsw

Task Definition	**Compile Unconfigured BSW**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Tasks		
Brief Description	Compile unconfigured BSW to get a BSW Module Object Code.		
Description	Compile Unconfigured BSW is the usual step to compile files without any configuration data when no configuration is needed. This can be use either in the pre-compile, link or post-build time.		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Basic Software Module Core Header	1	
Consumes	Basic Software Module Core Source Code	1	
Produces	Basic Software Module Object Code	1	
Used tool	Compiler	1	

Table 3.319: Compile Unconfigured BSW

3.6.4.1.2 Compile Configured Bsw

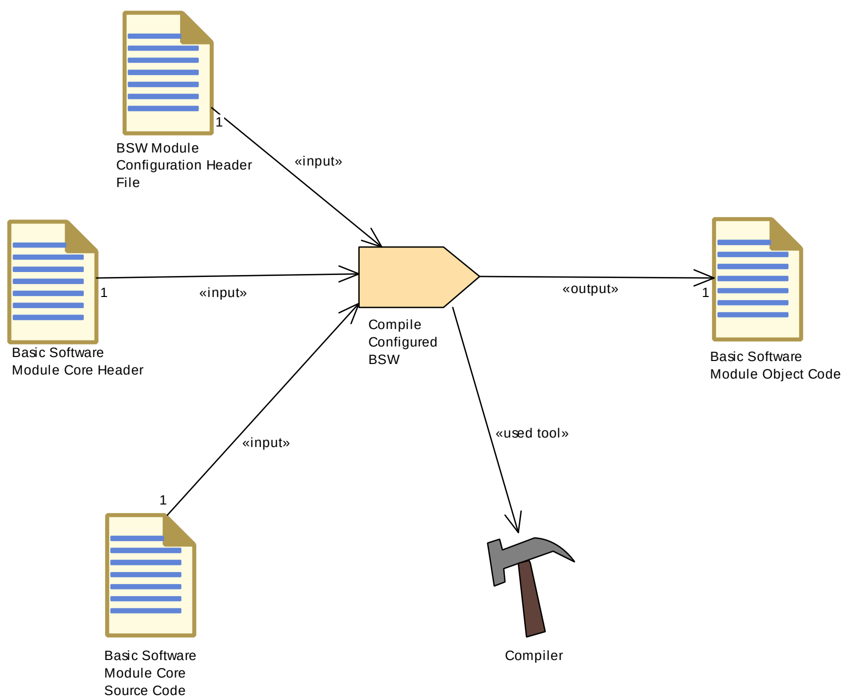

Figure 3.157: Compile Configured Bsw

Task Definition	**Compile Configured BSW**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Tasks		
Brief Description	Compile Configured BSW to get a BSW Module Object Code		
Description	Compile Configured BSW to get a Basic Software Module Object Code used in the link steps. This Configured BSW is representing C files that have already included all needed configured data. This is done in the pre-compile time.		
Relation Type	**Related Element**	**Mul.**	**Note**
Consumes	BSW Module Configuration Header File	1	
Consumes	Basic Software Module Core Header	1	
Consumes	Basic Software Module Core Source Code	1	
Produces	Basic Software Module Object Code	1	
Used tool	Compiler	1	

Table 3.320: Compile Configured BSW

3.6.4.1.3 Compile BSW Configuration Data

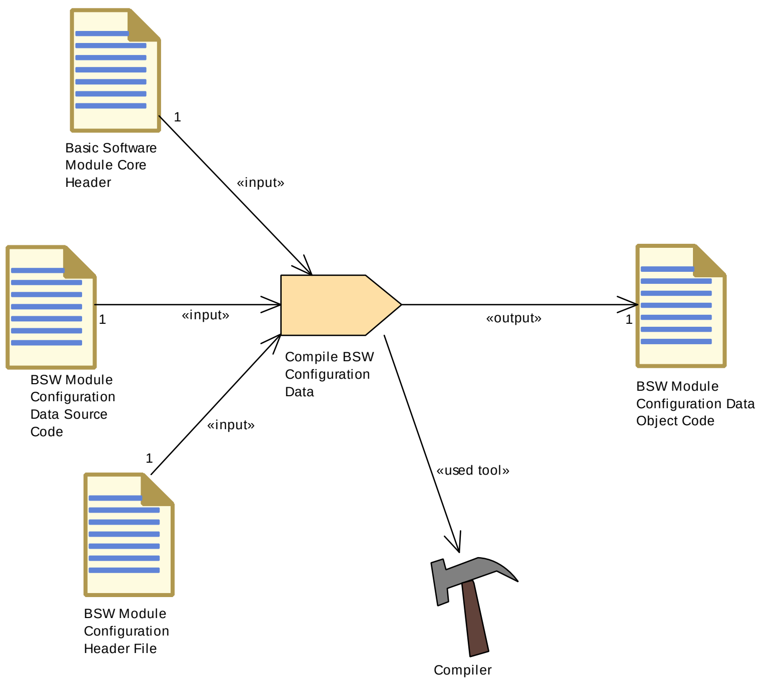

Figure 3.158: Compile BSW Configuration Data

Task Definition	**Compile BSW Configuration Data**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Tasks		
Brief Description	Compile BSW Configuration Data during link time		
Description	Compile BSW Configuration Data during link-time- or post-build configuration to get the Basic Software Module Configuration Data Object Code used in the link steps.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	BSW Module Configuration Data Source Code	1	
Consumes	BSW Module Configuration Header File	1	
Consumes	Basic Software Module Core Header	1	
Produces	BSW Module Configuration Data Object Code	1	
Used tool	Compiler	1	

Table 3.321: Compile BSW Configuration Data

3.6.4.1.4 Compile Generated BSW

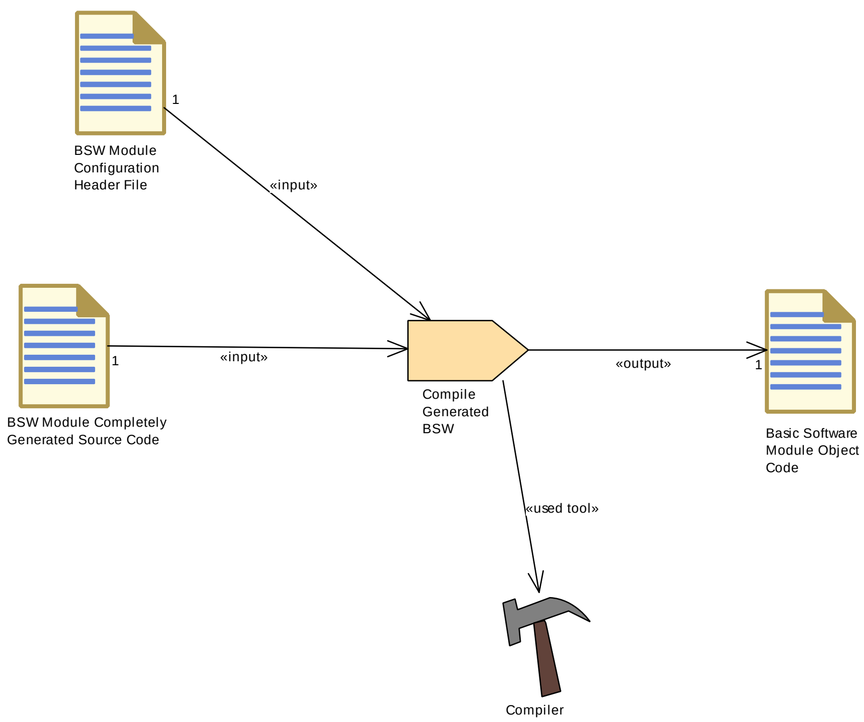

Figure 3.159: Compile Generated BSW

Task Definition	**Compile Generated BSW**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Tasks		
Brief Description	Compile generated BSW in the pre-compile time:		
Description	Compile generated BSW in the pre-compile time: this generated BSW has been generated with a BSW Configuration generator which generates the complete configuration-specific code.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	BSW Module Completely Generated Source Code	1	
Consumes	BSW Module Configuration Header File	1	
Produces	Basic Software Module Object Code	1	
Used tool	Compiler	1	

Table 3.322: Compile Generated BSW

3.6.4.1.5 Generate BSW Precompile Configuration Header

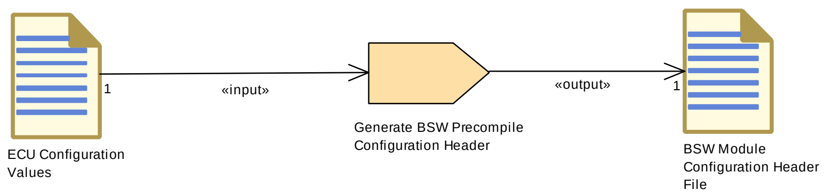

Figure 3.160: Generate BSW Precompile Configuration Header

Task Definition	**Generate BSW Precompile Configuration Header**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Tasks		
Brief Description	Generate BSW Precompile Configuration Header		
Description	Generate BSW Pre-compile Configuration Header. The header is used for definition or declaration (in case source code is needed) of the pre-compile configuration data code.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	ECU Configuration Values	1	
Produces	BSW Module Configuration Header File	1	

Table 3.323: Generate BSW Precompile Configuration Header

3.6.4.1.6 Generate BSW Source Code

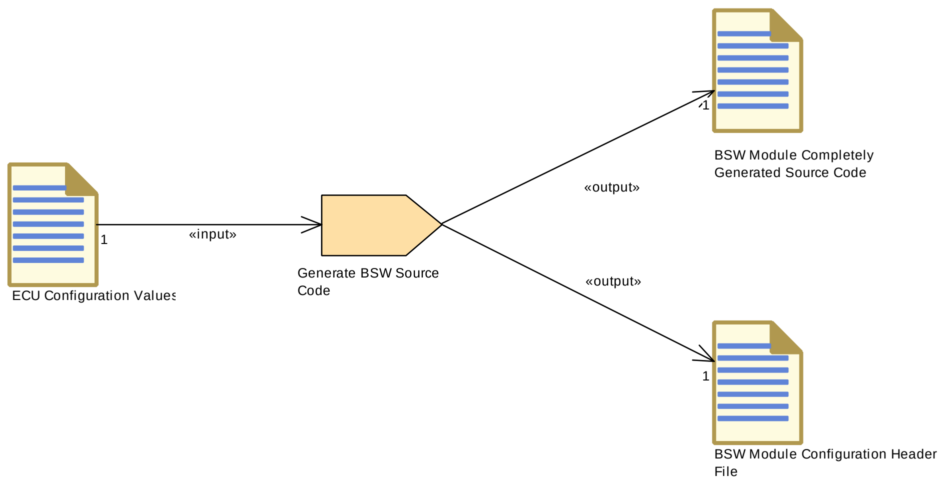

Figure 3.161: Generate BSW Source Code

Task Definition	**Generate BSW Source Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Tasks		
Brief Description	Generate the source code of a module completely from its precompile configuration.		
Description	Generate the source code of a BSW module completely from its pre-compile configuration. A header file may be produced in addition, if required.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	ECU Configuration Values	1	
Produces	BSW Module Completely Generated Source Code	1	
Produces	BSW Module Configuration Header File	1	

Table 3.324: Generate BSW Source Code

3.6.4.1.7 Generate BSW Configuration Code

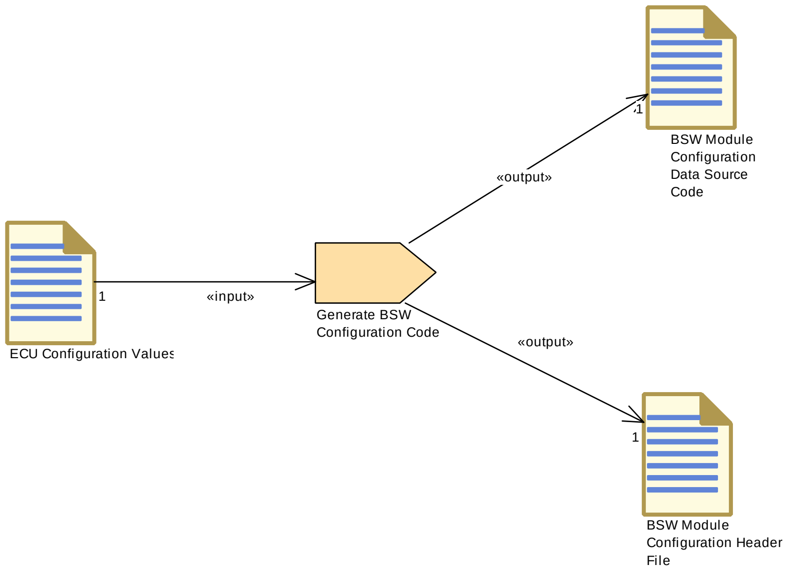

Figure 3.162: Generate BSW Configuration Code

Task Definition	**Generate BSW Configuration Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Tasks		
Brief Description	Generate source code which implements configuration data for link- or compile-time configuration.		
Description	A generator reads the relevant parameters from the ECU Configuration Description and creates a separate code file that implements the specified configuration. This task is used for link-time configuration, i.e. the configuration code can be produced at link-time of the core code or for compile-time configuration, if the configuration code cannot be put into a header file (e.g. for tables), even if the core code and the configuration code shall be compiled at the same time. A header file may be produced in addition, to declare the data. Furthermore the generator may produce extensions of the BSW module description artifacts as a result of configuration parameter values which are set at integration time.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	ECU Integrator	1	
Consumes	ECU Configuration Values	1	
Consumes	BSW Module Generator	0..1	This is an input in case a generator framework is used which has to run some module specific generator code.
Consumes	Build Action Manifest	0..1	The task may be controlled by a Build Action Manifest.

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	BSW Module Vendor-Specific Configuration Parameter Definition	0..*	
Produces	BSW Module Configuration Data Source Code	1	
Produces	BSW Module Configuration Header File	1	
Produces	BSW Module Behavior Extension	0..1	
Produces	BSW Module Implementation Extension	0..1	
Produces	BSW Module Interface Extension	0..1	
Used tool	BSW Generator Framework	1	

Table 3.325: Generate BSW Configuration Code

3.6.4.1.8 Generate BSW Postbuild Configuration Code

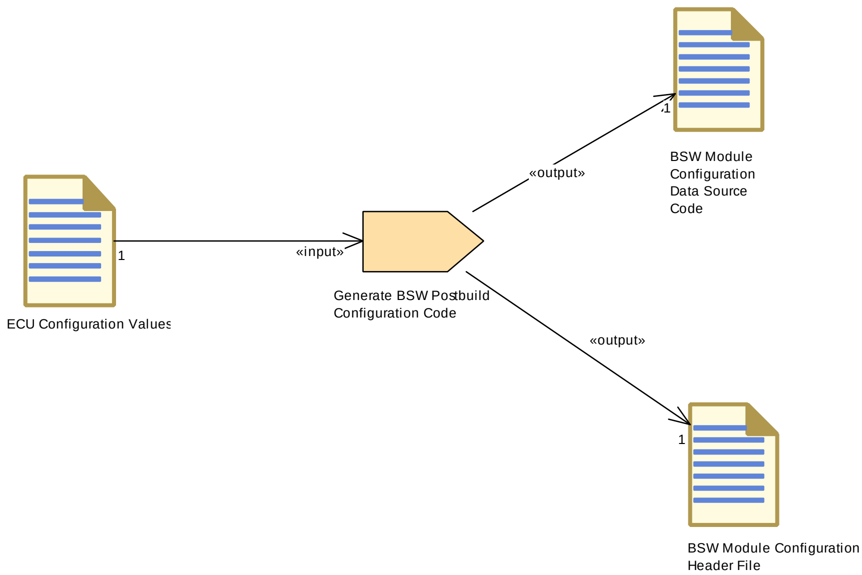

Figure 3.163: Generate BSW Postbuild Configuration Code

Task Definition	**Generate BSW Postbuild Configuration Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Tasks		
Brief Description	Generate the code for data structures that can be used for postbuild configuration.		
Description	Generate the source code and associated header for data structures that can be used for postbuild configuration.		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	ECU Configuration Values	1	
Produces	BSW Module Configuration Data Source Code	1	
Produces	BSW Module Configuration Header File	1	

Table 3.326: Generate BSW Postbuild Configuration Code

3.6.4.1.9 Link ECU after Precompile Configuration

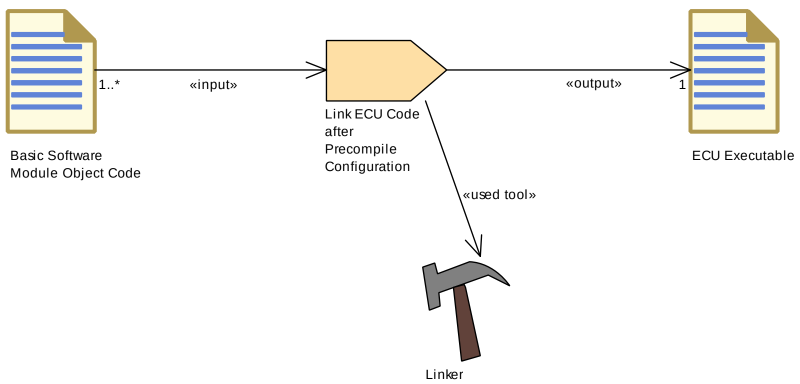

Figure 3.164: Link ECU after Precompile Configuration

Task Definition	**Link ECU Code after Precompile Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Tasks		
Brief Description	Link the ECU code in the pre-compile time Configuration Class		
Description	Link the different BSW modules object code in the pre-compile Configuration Class. All parameters values for configurable elements have been already fixed and are effective after compilation time.		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Basic Software Module Object Code	1..*	
Produces	ECU Executable	1	
Used tool	Linker	1	

Table 3.327: Link ECU Code after Precompile Configuration

3.6.4.1.10 Link ECU Code During Link Time Configuration

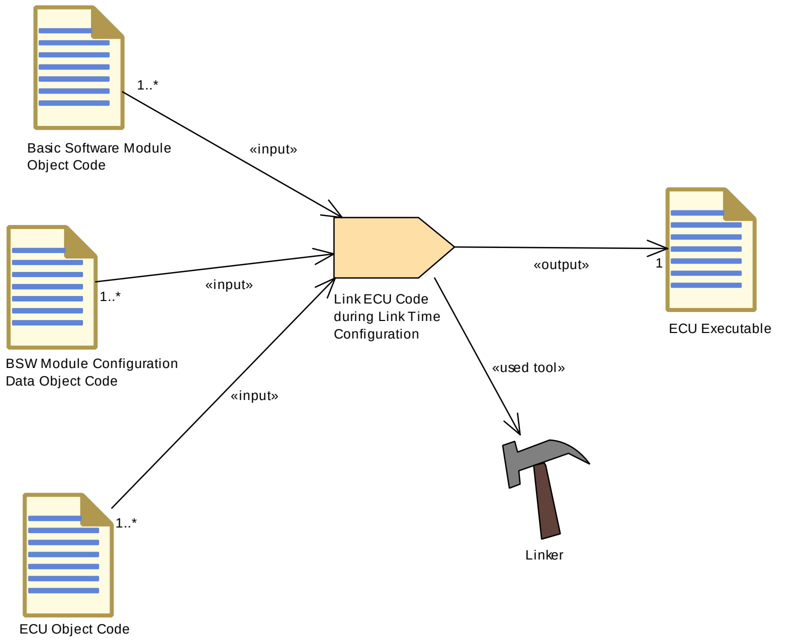

Figure 3.165: Link ECU Code During Link Time Configuration

Task Definition	**Link ECU Code during Link Time Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Tasks		
Brief Description	Link ECU Code during Link Time		
Description	Link ECU Code during Link Time		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	BSW Module Configuration Data Object Code	1..*	
Consumes	Basic Software Module Object Code	1..*	
Consumes	ECU Object Code	1..*	
Produces	ECU Executable	1	
Used tool	Linker	1	

Table 3.328: Link ECU Code during Link Time Configuration

3.6.4.1.11 Link ECU Code During Post-build Time

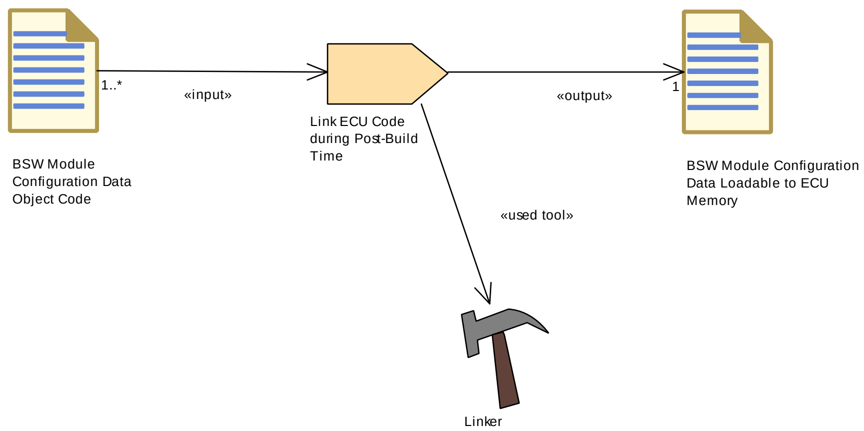

Figure 3.166: Link ECU Code During Post-build Time

Task Definition	**Link ECU Code during Post-Build Time**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Tasks		
Brief Description	Link ECU Code during post-build time loadable .		
Description	Link ECU Code during post-build time. The objects used for this link are coming from configuration data file that contain all configured parameters. The result of the link is a hex file that will be loadable in the ECU memory.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	BSW Module Configuration Data Object Code	1..*	
Produces	BSW Module Configuration Data Loadable to ECU Memory	1	
Used tool	Linker	1	

Table 3.329: Link ECU Code during Post-Build Time

3.6.4.2 Work Products

3.6.4.2.1 BSW Module Configuration Header File

Artifact	**BSW Module Configuration Header File**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Work Products		
Brief Description	C-header file generated from the configuration data of a BSW module.		
Description	C-header file generated from the configuration data of a BSW module, defining the data (only possible for pre-compile configuration) or containing additional declarations (needed by generated configuration code only).		
Kind	Bound Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Generate BSW Configuration Code	1	
Produced by	Generate BSW Postbuild Configuration Code	1	
Produced by	Generate BSW Precompile Configuration Header	1	
Produced by	Generate BSW Source Code	1	
Produced by	Generate BSW and RTE	1	
Consumed by	Compile BSW Configuration Data	1	
Consumed by	Compile Configured BSW	1	
Consumed by	Compile Generated BSW	1	
Consumed by	Compile ECU Source Code	0..*	Meth.bindingTime = CodeGenerationTime

Table 3.330: BSW Module Configuration Header File

3.6.4.2.2 BSW Module Completely Generated Source Code

Artifact	**BSW Module Completely Generated Source Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Work Products		
Brief Description	Generated BSW source code implementing the complete module after inclusion of pre-compilation configuration data.		
Description	Generated BSW source code implementing the complete module after inclusion of pre-compilation configuration data. In this case, no core code is delivered by the module vendor.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Generate BSW Source Code	1	
Consumed by	Compile Generated BSW	1	

Relation Type	**Related Element**	**Mul.**	**Note**

Table 3.331: BSW Module Completely Generated Source Code

3.6.4.2.3 BSW Module Configuration Data Source Code

Artifact	**BSW Module Configuration Data Source Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Work Products		
Brief Description	BSW source code generated from configuration data, implementing only the data.		
Description	BSW source code generated from configuration data, implementing only the data.		
Kind	Bound Source Code		
Relation Type	**Related Element**	**Mul.**	**Note**
Produced by	Generate BSW Configuration Code	1	
Produced by	Generate BSW Postbuild Configuration Code	1	
Produced by	Generate BSW and RTE	1	
Consumed by	Compile BSW Configuration Data	1	
Consumed by	Compile ECU Source Code	0..*	Meth.bindingTime = CodeGenerationTime

Table 3.332: BSW Module Configuration Data Source Code

3.6.4.2.4 BSW Module Configuration Data Object Code

Artifact	**BSW Module Configuration Data Object Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Work Products		
Brief Description	Generated data for link-time or postbuild configuration of a BSW module.		
Description	Generated & compiled configuration data for link-time or postbuild configuration of a BSW module.		
Kind	Object Code		
Relation Type	**Related Element**	**Mul.**	**Note**
Produced by	Compile BSW Configuration Data	1	
Consumed by	Link ECU Code during Link Time Configuration	1..*	
Consumed by	Link ECU Code during Post-Build Time	1..*	

Relation Type	***Related Element***	***Mul.***	***Note***

Table 3.333: BSW Module Configuration Data Object Code

3.6.4.2.5 BSW Module Configuration Data Loadable to ECU Memory

Artifact	**BSW Module Configuration Data Loadable to ECU Memory**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Ecu::ECU Config Classes::Work Products		
Brief Description	Generated loadable configuration data for post-build configuration of a BSW module.		
Description	Generated loadable configuration data for post-build configuration of a BSW module.		
Kind	Configuration Data Set		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	ECU Software Delivered	0..*	
Produced by	Link ECU Code during Post-Build Time	1	

Table 3.334: BSW Module Configuration Data Loadable to ECU Memory

A History of Constraints and Specification Items

A.1 Constraint History of this Document according to AUTOSAR R4.1.1

A.1.1 Added Specification Items in R4.1.1

Number	Heading
[TR_METH_00001]	Definition of Binding Time for Tasks
[TR_METH_00002]	Definition of Binding Time for Artifacts
[TR_METH_00003]	Definition of Binding Time for Artifacts in the context of particular tasks
[TR_METH_01000]	Domains of the AUTOSAR methodology
[TR_METH_01001]	AUTOSAR methodology assets
[TR_METH_01002]	AUTOSAR methodology use cases
[TR_METH_01003]	Scope of the AUTOSAR methodology
[TR_METH_01004]	Support for various stakeholders by the AUTOSAR methodology
[TR_METH_01005]	Restrictions of AUTOSAR methodology
[TR_METH_01006]	General AUTOSAR methodology concepts
[TR_METH_01007]	Method Library
[TR_METH_01008]	Method Library Element
[TR_METH_01009]	Relation of Method Library and Method Library Element to the SPEM meta model
[TR_METH_01010]	Overview of Method Library Elements
[TR_METH_01011]	Task Definition
[TR_METH_01012]	Task semantics
[TR_METH_01013]	Task usage
[TR_METH_01014]	Work Product Definition
[TR_METH_01015]	Relationship between Roles and Work Products
[TR_METH_01017]	Artifact Definition
[TR_METH_01018]	Kinds of Artifacts
[TR_METH_01019]	Properties of Artifacts
[TR_METH_01020]	Relationship between Artifacts and meta model elements
[TR_METH_01021]	Deliverable Definition
[TR_METH_01022]	Aggregation of Work Products
[TR_METH_01023]	Role Definition
[TR_METH_01024]	Role assignment
[TR_METH_01025]	Tool Definition
[TR_METH_01026]	Guidance definition
[TR_METH_01027]	Guidance kinds
[TR_METH_01028]	Usage of tables
[TR_METH_01029]	Capability Patterns definition
[TR_METH_01030]	Composition of Capability Patterns
[TR_METH_01031]	Adaptability of the AUTOSAR methodology
[TR_METH_01032]	Use case elements
[TR_METH_01033]	Definition of Activities
[TR_METH_01034]	Composition of Activities
[TR_METH_01035]	Definition of Processes
[TR_METH_01036]	Description of overall Use Cases
[TR_METH_01037]	Precise description of Use Cases
[TR_METH_01038]	Detailed description of the work flow
[TR_METH_01039]	AUTOSAR System development overview

[TR_METH_01040]	Support of different system views
[TR_METH_01041]	Abstract system
[TR_METH_01042]	Overall technical system
[TR_METH_01043]	Sub-System
[TR_METH_01044]	Development of a functional view on the system
[TR_METH_01045]	Development of the Overall VFB System
[TR_METH_01046]	Development of the system
[TR_METH_01047]	Two phase development approach
[TR_METH_01048]	The overall system
[TR_METH_01049]	Interaction between organizations
[TR_METH_01050]	Abstract System Description activity
[TR_METH_01051]	Creation of an overall abstract system
[TR_METH_01052]	Definition of a constraints in the context of an abstract system
[TR_METH_01053]	Definition of a System Description in the context of an abstract system
[TR_METH_01054]	Virtual Functional Bus
[TR_METH_01055]	Data Model Development activity
[TR_METH_01056]	Definition of the VFB
[TR_METH_01057]	Top-Down approach
[TR_METH_01058]	Bottom-Up approach
[TR_METH_01059]	Kinds of VFB Atomic Software Components
[TR_METH_01060]	Develop an Atomic Software Component activity
[TR_METH_01061]	Develop Application Software activity
[TR_METH_01065]	Develop System and Develop Sub-System activities
[TR_METH_01066]	Creation of a System Extract and a ECU Extract
[TR_METH_01067]	Abstract System Description deliverable
[TR_METH_01068]	Inputs and Output of the Design System activity
[TR_METH_01069]	Deployment of AUTOSAR Software Components
[TR_METH_01070]	Description of network signals
[TR_METH_01071]	Description of design constraints
[TR_METH_01075]	Design Sub-System activity
[TR_METH_01076]	Collaboration between different organizations
[TR_METH_01077]	Transformation changes during the Design Sub-System activity
[TR_METH_01078]	Mapping of different views
[TR_METH_01079]	Use Case: Substitution of existing components
[TR_METH_01080]	Use Case: Mapping of requirements to the solution
[TR_METH_01081]	Use Case: Reorganization of the software structure
[TR_METH_01082]	Use Case: Description of changes between different versions of System Descriptions
[TR_METH_01083]	Design Basic Software activity
[TR_METH_01084]	Separation of design and development of basic software
[TR_METH_01085]	Develop BSW Module activity
[TR_METH_01086]	Integrate Software for ECU activity
[TR_METH_01087]	Scope of Integrate Software for ECU activity
[TR_METH_01088]	Prepare ECU Configuration activity
[TR_METH_01089]	Configure BSW and RTE activity
[TR_METH_01090]	Configure RTE task
[TR_METH_01091]	Configure Debug task
[TR_METH_01092]	Generating BSW modules, RTE, and OS source files
[TR_METH_01093]	Building ECU Executable
[TR_METH_01095]	Configuration Class: Pre-compile Time
[TR_METH_01096]	Generating header files only
[TR_METH_01097]	Generating header and source files
[TR_METH_01098]	Configuration Class: Link Time

[TR_METH_01099]	Generation and compilation of BSW Configuration Code
[TR_METH_01100]	Definition of configuration data
[TR_METH_01101]	Separate compilation of module source and configuration file
[TR_METH_01102]	Linking process
[TR_METH_01103]	Re-generation in case of configuration value changes
[TR_METH_01104]	Configuration Class: Post-build Time
[TR_METH_01105]	Generate BSW Postbuild Configuration Code
[TR_METH_01106]	Generate BSW Configuration Data Loadable
[TR_METH_01107]	Configuration Class: Post-build Time Selectable
[TR_METH_01108]	Generating multiple post-build configuration variants
[TR_METH_01109]	Producing ECU-specific deliverables
[TR_METH_01110]	Development of Software Components
[TR_METH_01111]	Development of Basic Software modules
[TR_METH_01112]	Integration of AUTOSAR ECUs
[TR_METH_01113]	Usage of hyperlinks
[TR_METH_01120]	Definition of Consistency Needs
[TR_METH_01121]	Building the AUTOSAR methodology document
[TR_METH_01122]	Relations between AUTOSAR Work Products
[TR_METH_01123]	Traceability to external artifacts
[TR_METH_01124]	Documentation of Work Products
[TR_METH_02000]	Use of AUTOSAR Services
[TR_METH_02001]	Define Cross-component Calibration Parameters activity
[TR_METH_02002]	Define Local Calibration Parameters activity
[TR_METH_02003]	Provide Unique Parameter Names activity
[TR_METH_02004]	Re-generate RTE and Calibration Support activity
[TR_METH_02005]	Memory sections for data and code
[TR_METH_02006]	E2E Protection
[TR_METH_02007]	Define E2E Protection Set activity
[TR_METH_02008]	Regenerate E2E Protection Wrapper activity
[TR_METH_02009]	Variation points in Variant Handling
[TR_METH_02010]	Predefined Variants in Variant Handling
[TR_METH_02011]	Types of binding times
[TR_METH_02012]	Definition of a binding time
[TR_METH_02013]	Latest Binding Time
[TR_METH_02014]	Actual Binding Time
[TR_METH_02015]	Definition of variants
[TR_METH_02016]	Evaluated Variant Set
[TR_METH_02017]	Use of Predefined Variant
[TR_METH_02018]	Choosing variants
[TR_METH_02020]	Definition of latest Binding Time for a variation point in the meta-model
[TR_METH_03000]	Name spaces via ARPackages
[TR_METH_03001]	Reasons for name conflicts in “downstream” artifacts
[TR_METH_03002]	Conflict solution at system design time
[TR_METH_03003]	Conflict solution at coding time
[TR_METH_03004]	Conflict solution at ECU integration time
[TR_METH_03005]	Conflict solution via SymbolProps
[TR_METH_03006]	Conflict solution via literal prefixes
[TR_METH_03007]	Conflict solution in names of runnable entities
[TR_METH_03008]	Conflict solution via FlatMap
[TR_METH_03009]	Conflict solution via AliasNameSet
[TR_METH_03010]	Conflict solution via API Infixes

Table A.1: Added Specification Items in 4.1.1

A.2 Constraint History of this Document according to AUTOSAR R4.1.2

A.2.1 Added Specification Items in R4.1.2

Number	Heading
[TR_METH_01114]	Input sources for ECU Configuration
[TR_METH_01115]	A mix of parameters with different configuration classes within a BSW module is allowed
[TR_METH_01116]	ECU Configuration Value description contains the configuration of all BSW modules in a single ECU
[TR_METH_01117]	BSW implementation shall be chosen for each BSW module that is present in the ECU

Table A.2: Added Specification Items in 4.1.2

A.3 Constraint History of this Document according to AUTOSAR R4.1.3

A.3.1 Added Specification Items in R4.1.3

Number	Heading
[TR_METH_01125]	`Create ECU System Description` activity
[TR_METH_01126]	Using the `System Extract` as the structural basis for the ECU development
[TR_METH_01127]	Creating a new structure for the ECU development

Table A.3: Added Specification Items in 4.1.3

A.3.2 Changed Specification Items in R4.1.3

Number	Heading
[TR_METH_01049]	Interaction between organizations
[TR_METH_01066]	Creation of a `System Extract` and an `ECU Extract`
[TR_METH_01075]	`Design Sub-System` activity
[TR_METH_01076]	Collaboration between different organizations

Table A.4: Changed Specification Items in 4.1.3

A.4 Constraint History of this Document according to AUTOSAR R4.2.1

A.4.1 Added Specification Items in R4.2.1

Number	Heading
[TR_METH_01128]	Integration of Non AUTOSAR Systems in the context of an abstract system
[TR_METH_01129]	`Integrate Non AUTOSAR System at VFB level` activity
[TR_METH_01130]	`Design Transformer` activity

[TR_METH_01131]	Output of Design Transformer activity
[TR_METH_01132]	Definition of a Rapid Prototyping Scenario
[TR_METH_01133]	Content of Rapid Prototyping Scenario artifact
[TR_METH_01134]	Component wrapper method
[TR_METH_01135]	Direct buffer access method
[TR_METH_01136]	Content of Diagnostic Extract
[TR_METH_01137]	Diagnostic Extract category
[TR_METH_01138]	Decentralized configuration
[TR_METH_01139]	Roles
[TR_METH_01140]	Develop Diagnostic Abstract System Description activity
[TR_METH_01141]	Development of diagnostic requirements
[TR_METH_01142]	Diagnostic information in the context of SW-C development
[TR_METH_01143]	Integration of diagnostic information
[TR_METH_01144]	Activity Define Safety Information
[TR_METH_01145]	Creation of Safety Requirements
[TR_METH_01146]	Allocation of Safety Requirements
[TR_METH_01147]	Decomposition of Safety Requirements
[TR_METH_01148]	Definition of Safety Measures
[TR_METH_01149]	Definition of VFB relevant safety information
[TR_METH_01150]	Including different post-build variants
[TR_METH_01151]	Update ECU Configuration activity
[TR_METH_01153]	Configuration and Generation of the E2E Transformer
[TR_METH_01154]	Define E2E Transformer Technology Task

Table A.5: Added Specification Items in 4.2.1

A.4.2 Changed Specification Items in R4.2.1

Number	Heading
[TR_METH_01059]	Kinds of VFB Atomic Software Components
[TR_METH_01046]	Development of the system
[TR_METH_01065]	Develop System and Develop Sub-System activities
[TR_METH_01060]	Develop an Atomic Software Component activity
[TR_METH_01065]	Develop System and Develop Sub-System activities
[TR_METH_01104]	Configuration Class: Post-build Time
[TR_METH_01105]	Generate BSW Postbuild Configuration Code
[TR_METH_01108]	Generating multiple post-build configuration variants
[TR_METH_02006]	E2E Protection

Table A.6: Changed Specification Items in 4.2.1

A.4.3 Deleted Specification Items in R4.2.1

Number	Heading
[TR_METH_01106]	Generate BSW Configuration Data Loadable
[TR_METH_01107]	Configuration Class: Post-build Time Selectable
[TR_METH_02007]	Define E2E Protection Set activity
[TR_METH_02008]	Regenerate E2E Protection Wrapper activity

Table A.7: Deleted Specification Items in 4.2.1

A.5 Constraint History of this Document according to AUTOSAR R4.2.2

No changes.

A.6 Constraint History of this Document according to AUTOSAR R4.3.0

A.6.1 Added Specification Items in R4.3.0

Number	Heading
[TR_METH_01155]	Definition of serialization
[TR_METH_01156]	Use case: Serialization based on network representation
[TR_METH_01157]	Use case: Serialization based on implementation data types
[TR_METH_01202]	Create a Profile of Data Exchange Point
[TR_METH_01204]	Agreement on a profile for data exchange points
[TR_METH_01205]	Validation based on an Agreed Profile of Data Exchange Point

Table A.8: Added Specification Items in 4.3.0

A.6.2 Changed Specification Items in R4.3.0

Number	Heading
[TR_METH_01006]	General AUTOSAR methodology concepts
[TR_METH_01013]	Task usage
[TR_METH_01032]	Use case elements
[TR_METH_01036]	Description of overall Use Cases
[TR_METH_01037]	Precise description of Use Cases
[TR_METH_01000]	Domains of the AUTOSAR methodology
[TR_METH_01039]	Virtual Functional Bus View
[TR_METH_01040]	Support of different system views
[TR_METH_01044]	Development of a functional view on the system
[TR_METH_01045]	Development of the Overall VFB System
[TR_METH_01046]	Development of the system
[TR_METH_01047]	Two phase development approach
[TR_METH_01049]	Interaction between organizations
[TR_METH_01109]	Producing ECU-specific deliverables
[TR_METH_01110]	Development of Software Components
[TR_METH_01112]	Integration of AUTOSAR ECUs
[TR_METH_01093]	Building ECU Executable
[TR_METH_01071]	Description of design constraints
[TR_METH_01130]	Design Custom Transformer activity
[TR_METH_01131]	Output of Design Transformer activity

Table A.9: Changed Specification Items in 4.3.0

A.7 Constraint History of this Document according to AUTOSAR R4.3.1

A.7.1 Added Specification Items in R4.3.1

N/A

A.7.2 Changed Specification Items in R4.3.1

Number	Heading
[TR_METH_01014]	Work Product Definition

Table A.10: Changed Specification Items in 4.3.1

A.7.3 Deleted Specification Items in R4.3.1

N/A