Document Title	Methodology for Adaptive Platform
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	709

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	18-03

Document Change History

Date	Release	Changed by	Description
2018-03-29	18-03	AUTOSAR Release Management	• Split of machine design and machine configuration • Added diagnostic mapping • Added roles • Reviewed sections about deployment of Software Packages
2017-10-27	17-10	AUTOSAR Release Management	• Design of service oriented communication between CP and AP • Design of signal oriented communication between CP and AP • Deployment by means of SoftwareCluster • Removed concept of TransportLayerIndependentInstanceId
2017-03-31	17-03	AUTOSAR Release Management	• Initial release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

Bibliography

[1] Methodology
AUTOSAR_TR_Methodology

[2] Requirements on Methodology
AUTOSAR_RS_Methodology

[3] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

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

[5] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[6] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[7] Specification of ECU Resource Template
AUTOSAR_TPS_ECUResourceTemplate

[8] Glossary
AUTOSAR_TR_Glossary

[9] Specification of Update and Configuration Management
AUTOSAR_SWS_UpdateAndConfigManagement

1 Introduction

1.1 Objective and Scope

AUTOSAR requires a common technical approach for at least the major development steps, called the AUTOSAR methodology.

The methodology for the AUTOSAR Classic Platform is given by [1], whereas this document defines the methodology for the AUTOSAR Adaptive Platform.

The corresponding requirements are defined in [2].

The present expansion was necessary, because the AUTOSAR Adaptive Platform has introduced new concepts.

In contrast to the AUTOSAR Classic Platform, instances of Adaptive Applications, for example, are executed within the context of processes, entities managed by the operating system. If permitted by the configuration of the operating system, processes may be started, executed or stopped, at any time during the life cycle of a machine. As a consequence, the way of configuration (by the means of Manifests) or when and how software packages are deployed (e.g., by software updates over-the-air) clearly differ from the concepts of the AUTOSAR Classic Platform.

Moreover, the term machine has been newly introduced with the AUTOSAR Adaptive Platform. A machine is quasi a virtualized ECU, an entity where software can be deployed to. In this spirit, one real ECU could run several machines, even though the methodology will not detail this. In the simplest case the term machine may only be a synonym for ECU.

Although the list is not complete, aforementioned aspects may serve as sufficient motivation to provide a separate methodology for the AUTOSAR Adaptive Platform.

Despite all the differences, there are also many commonalities, such as the description of the system features, like topologies or hardware capabilities. This document, however, will rather focus on the specifics of the AUTOSAR Adaptive platform, in order to avoid duplications. The specification of the common aspects of both platforms may be the subject of a separate document (foundation document) later.

[TR_AMETH_00100] Scope of the Methodology for the AUTOSAR Adaptive Platform ⌈The methodology for the AUTOSAR Adaptive Platform describes main aspects (use-cases, tasks, work products, ...) necessary to build an Adaptive AUTOSAR system and how they relate to each other. However, the methodology does neither provide a complete process description, nor does it stipulate a precise order of activities. Iterations of activities are possible, but it is not described how and when iterations shall be carried out. ⌋*(RS_METH_00006, RS_METH_00020, RS_METH_00056)*

1.2 Document Outline

This document will follow the policies of the AUTOSAR Classic Platform, i.e., the way how to model use-cases, how to structure the document and the way to specify.

Thus, the outline of this document follows roughly its counterpart of the AUTOSAR Classic Platform:

The rest of this section documents the policies utilized and the requirements traceability map.

Section 2 describes the major use cases for the development of a system implementing an AUTOSAR Adaptive Platform. Note that the description of the life cycle of a Software Package is not included in the AUTOSAR methodology.

Section 3 lists and describes all tasks and work products, which are used in the descriptions of the use cases in section 2.

1.3 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. ECUs.

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 ([3]).

1.4 Abbreviations

The following table contains a list of abbreviations used in the scope of this document along with the spelled-out meaning of each of the abbreviations.

Abbreviation	*Meaning*
ABI	Application Binary Interface
AP	AUTOSAR Adaptive Platform
API	Application Programming Interface
ARXML	AUTOSAR XML
CP	AUTOSAR Classic Platform
DoIP	Diagnostics over IP
DM	Diagnostic Manager

Abbreviation	*Meaning*
DTC	Diagnostic Trouble Code
ECU	Electrical Control Unit
E/E system	Electric and Electronic system
HW	Hardware
ID	Identifier
IP	Internet Protocol
JSON	JavaScript Object Notation
NM	Network Management
NV	Non-Volatile
OEM	Original Equipment Manufacturer
OS	Operating System
PHM	Platform Health Management
POSIX	Portable Operating System Interface
SD	Service Discovery
SOME/IP	Scalable service-Oriented MiddlewarE over IP
SWC	Software Component
TCP	Transport Control Protocol
TLV	Tag Length Value
UCM	Update and Configuration Management
UDS	Unified Diagnostic Services
UDP	User datagram Protocol
UML	Unified Modeling Language
UUID	Universally Unique Identifier
VFB	Virtual Functional Bus
XML	Extensible Markup Language
XSD	XML Schema Definition

Table 1.1: Abbreviations used in the scope of this Document

1.5 Methodology Concepts

The concepts of the methodology for the Adaptive Platform are identical with the concepts of the methodology for the Classic Platform. Hence, we will only mention the main principles here. Please refer to section 1.5 in [1] for further details.

[TR_AMETH_00101] Definition of tasks, work products and use cases ⌈ The methodology describes typical use cases by means of `activity`s, entities to aggregate `task`s and their corresponding `work product`s. `Task`s are defined as reusable elements: input information (e.g., stored within particular `work product`s) is processed in order to generate new work products. This document describes use cases in Section 2, tasks and work products in Section 3. ⌋*(RS_METH_00018)*

[TR_AMETH_00102] Types and kinds of work products ⌈ Work products are either artifacts or deliverables and can be of the kind AUTOSAR XML, source code, object code, executable, text or custom. ⌋*(RS_METH_00018)*

[TR_AMETH_00226] Documentation of work products ⌈ In order to document design decisions or restrictions during the development process, each work product may aggregate a corresponding documentation. ⌋*(RS_METH_00069)*

The definitions and the figures are made according to the Software Process Engineering Meta-Model Specification (SPEM) [4]. The symbols are those used by the Enterprise Architect modeling tool.

1.6 Requirements Traceability

The following table references the requirements specified in the corresponding requirements document [2].

Requirement	Description	Satisfied by
[RS_METH_00006]	The methodology shall explain how to build an AUTOSAR system	[TR_AMETH_00016] [TR_AMETH_00100]
[RS_METH_00015]	The methodology shall be independent of programming languages	[TR_AMETH_00013]
[RS_METH_00016]	The methodology shall support building a system of both AUTOSAR and Non-AUTOSAR ECUs	[TR_AMETH_00212] [TR_AMETH_00213]
[RS_METH_00018]	The methodology shall be modular	[TR_AMETH_00101] [TR_AMETH_00102] [TR_AMETH_00200] [TR_AMETH_00211]
[RS_METH_00020]	The methodology shall support round-trip engineering	[TR_AMETH_00100] [TR_AMETH_00211]
[RS_METH_00032]	The methodology shall support different levels of abstractions	[TR_AMETH_00001] [TR_AMETH_00002] [TR_AMETH_00200] [TR_AMETH_00201] [TR_AMETH_00202] [TR_AMETH_00205]
[RS_METH_00041]	The methodology shall support top-down and bottom-up approaches	[TR_AMETH_00019] [TR_AMETH_00020] [TR_AMETH_00034] [TR_AMETH_00035] [TR_AMETH_00204]
[RS_METH_00042]	The methodology shall incorporate the usage of industry standard tools	[TR_AMETH_00013] [TR_AMETH_00018]

[RS_METH_00056]	The AUTOSAR methodology shall not be bound to a particular life-cycle model	[TR_AMETH_00100]
[RS_METH_00066]	The methodology shall allow activities that reference tools	[TR_AMETH_00012] [TR_AMETH_00013] [TR_AMETH_00016] [TR_AMETH_00018]
[RS_METH_00069]	It shall be possible to add precise and human readable documentation to each work product	[TR_AMETH_00226]
[RS_METH_00077]	The methodology shall support different views on the SW-C structure by OEMs and suppliers	[TR_AMETH_00014] [TR_AMETH_00015] [TR_AMETH_00016] [TR_AMETH_00024]
[RS_METH_00078]	The methodology shall explain the typical usage of different views on the system of the OEM	[TR_AMETH_00029] [TR_AMETH_00033] [TR_AMETH_00203]
[RS_METH_00079]	The methodology shall explain the typical usage of different views on the system of the supplier	[TR_AMETH_00203]
[RS_METH_00200]	The methodology shall support building a system consisting of several AUTOSAR platforms	[TR_AMETH_00208] [TR_AMETH_00209] [TR_AMETH_00210]
[RS_METH_00201]	The methodology shall explain how to design the services of a system	[TR_AMETH_00001] [TR_AMETH_00007] [TR_AMETH_00008] [TR_AMETH_00009] [TR_AMETH_00212] [TR_AMETH_00213]
[RS_METH_00202]	The methodology shall explain how to develop an Adaptive Application	[TR_AMETH_00002] [TR_AMETH_00010] [TR_AMETH_00011] [TR_AMETH_00012] [TR_AMETH_00013] [TR_AMETH_00014] [TR_AMETH_00015] [TR_AMETH_00018] [TR_AMETH_00205] [TR_AMETH_00207] [TR_AMETH_00208] [TR_AMETH_00209] [TR_AMETH_00210]

[RS_METH_00203]	The methodology shall explain the high-level usage of the Manifest Specification	[TR_AMETH_00003] [TR_AMETH_00004] [TR_AMETH_00005] [TR_AMETH_00021] [TR_AMETH_00022] [TR_AMETH_00023] [TR_AMETH_00024] [TR_AMETH_00025] [TR_AMETH_00026] [TR_AMETH_00027] [TR_AMETH_00028] [TR_AMETH_00029] [TR_AMETH_00033] [TR_AMETH_00214] [TR_AMETH_00215] [TR_AMETH_00216] [TR_AMETH_00217]
[RS_METH_00204]	The methodology shall describe how to configure a machine for the Adaptive Platform	[TR_AMETH_00003] [TR_AMETH_00021] [TR_AMETH_00022] [TR_AMETH_00023] [TR_AMETH_00031] [TR_AMETH_00214] [TR_AMETH_00215] [TR_AMETH_00216] [TR_AMETH_00217]
[RS_METH_00205]	The methodology shall describe how to deploy software on the Adaptive Platform	[TR_AMETH_00006] [TR_AMETH_00031] [TR_AMETH_00206]
[RS_METH_00206]	The methodology shall explain how to configure the instances of services of a system	[TR_AMETH_00005] [TR_AMETH_00027] [TR_AMETH_00028] [TR_AMETH_00029] [TR_AMETH_00033]
[RS_METH_00207]	The methodology shall explain how to develop Platform Software for the Adaptive Platform	[TR_AMETH_00017] [TR_AMETH_00019] [TR_AMETH_00020] [TR_AMETH_00034] [TR_AMETH_00035] [TR_AMETH_00212] [TR_AMETH_00213]

1.7 Known Limitations

Changes can be expected for Section 2.4.4 (Define and Configure Service Instances) for the next release.

The sections related to the deployment of Software Packages, i.e., Section 2.4.5 (Set up an initial Machine), Section 2.4.6 (Create Software Packages) and Section 2.4.7 (Management and provision of Software Packages), are still under discussion.

2 Use Cases for the Adaptive Platform

This section describes the main use cases for building a system based on the AUTOSAR Adaptive Platform.

Each section consists of subsections for the overall purpose of the use case, the description in terms of specifications, and the modeled workflow according to [4].

Please be aware that the roles shown in the diagrams may only be regarded as a good approximation.

2.1 Overall View

2.1.1 Purpose

This section provides an overview of the design and development steps to build a system based on the AUTOSAR Adaptive Platform. The main activities of the overall development are depicted in Figure 2.6. An overview of the workflow including relevant work products is given in Figure 2.7. A brief description of these main steps is given below in Section 2.1.2. For a detailed description please refer to the relevant sections.

2.1.2 Description

2.1.2.1 Domains of Development

It is good practice to decompose the development of complex systems into different work phases, for example analysis, design, implementation and the like. Each work phase will thereby be linked to a different level of abstraction. Moreover, each stakeholder of this development will need a distinct view on the system in order to emphasize on its particular aspects.

Thus, all this needs to somehow be represented by the methodology, too. In this respect, the methodology of the AUTOSAR Classic Platform is structured into so-called domains of development [1], which is in some way a mix of the concepts *separation of concerns* and *abstraction*.

The methodology of the AUTOSAR Adaptive Platform will follow this approach.

[TR_AMETH_00200] Domains of development utilized for the methodology of the AUTOSAR Adaptive Platform ⌈ The methodology of the Adaptive Platform shall be structured by the following domains of development:

- `Analysis`
- `Architecture and Design`
- `System`

- Software Development
- Integration and Deployment

⌋(*RS_METH_00018*, *RS_METH_00032*)

2.1.2.2 Fundamental Activities

2.1.2.2.1 Analysis

Analysis tasks are often necessary for the purpose of preparing later decisions. One line of inquiry may be to identify and investigate timing critical event chains between sensors and actuators of a vehicle function in order to comply with the required timing behavior.

Although the present version does not, later versions of this document will specify corresponding use-cases/activities.

2.1.2.2.2 Architecture and Design

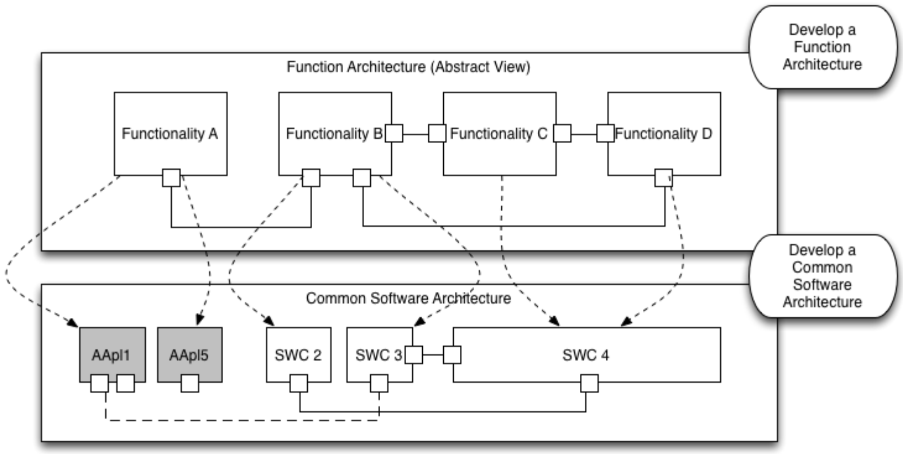
Develop a Function Architecture
Function Architecture (Abstract View)
Functionality A
Functionality B
Functionality C
Functionality D
Develop a Common Software Architecture
Common Software Architecture
AApl1
AApl5
SWC 2
SWC 3
SWC 4

Figure 2.1: From the `Function Architecture` to a `Common Software Architecture`

[TR_AMETH_00201] Develop a Function Architecture ⌈ An engineer, e.g., an E/E system architect, may evaluate features and requirements necessary for a specific E/E vehicle project in order to form an appropriate `Function Architecture` during the activity `Develop a Function Architecture`.

The `Function Architecture` is composed of a number of function networks. A function network consists of a set of function blocks with their interfaces and corresponding interconnections. One functionality is encapsulated within one function block. Therefore, a particular function network represents all functionality that is needed to execute a particular feature (vehicle function). Note, that function blocks may be realized in software or hardware or as a mix of both.

The result of this activity, i.e., the Function Architecture can be specified by means of the Abstract System Description.

This activity is optional. ⌋*(RS_METH_00032)*

[TR_AMETH_00202] Develop a Common Software Architecture ⌈ Another engineer, e.g., a software architect, could take the Function Architecture as one input to deduce a corresponding Common Software Architecture while executing an activity Develop a Common Software Architecture.

The Common Software Architecture provides a dedicated view of all software entities and their communication relation within the E/E vehicle system. In this light, the Common Software Architecture comprises both types, AUTOSAR software components of the Classic Platform as well as those entities that form later an Adaptive Application Software deployed to an Adaptive Platform-based machine. It is important to stress this, because not only software components of the same platform type communicate among each other. There is also a service oriented communication possible between software components or entities that belong to different platform types.

The communication entry and exit points of components are ports typed by a particular interface definition. In case of the Adaptive Platform, interfaces are expressed as Service Interfaces. In this respect, typed ports are means to instantiate specific interface definitions.

Figure 2.1 shows that a functionality may be implemented by one or more software components, by software components which are finally be mapped either to a machine running an AUTOSAR Adaptive Platform (gray boxes, named AApl for Adaptive Application) or to a Classic Platform ECU.

The term *component* may also include the term *compositions of components*. An Adaptive Application Software may also be subdivided into more fine-granular components.

The result of this activity, i.e., the Common Software Architecture can be specified by means of the System Description.

This activity is optional. ⌋*(RS_METH_00032)*

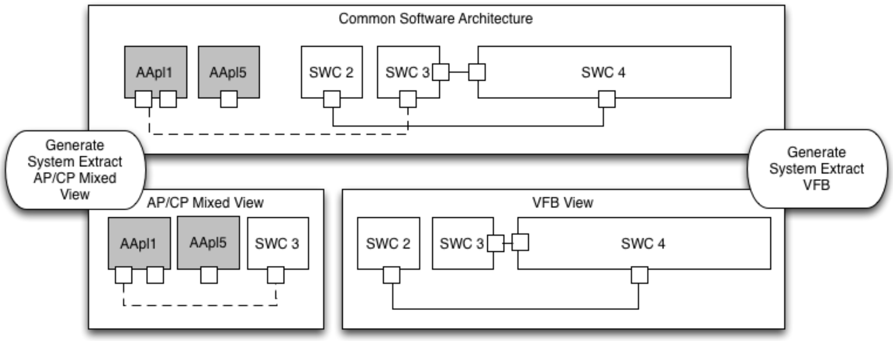
Common Software Architecture
AApl1
AApl5
SWC 2
SWC 3
SWC 4
Generate System Extract AP/CP Mixed View
AP/CP Mixed View
AApl1
AApl5
SWC 3
VFB View
SWC 2
SWC 3
SWC 4
Generate System Extract VFB

Figure 2.2: Views of subsystems enable to emphasize on relevant aspects

[TR_AMETH_00203] Provide views of subsystems ⌈ A subsystem is a reduced part of the overall technical system and emphasizes on relevant aspects of it.

Figure 2.2 shows two possible views on subsystems deduced from the Common Software Architecture. It is absolutely feasible, for example, to generate a pure VFB view or a view on a mixed Adaptive/Classic Platform subsystem.

Latter could contain all those software entities which communicate at least to one other Adaptive Application Software. It may be usable to develop the interfaces for communication between software components/entities which belong to different platforms, AUTOSAR Adaptive Platform or AUTOSAR Classic Platform.

This activity is optional. ⌋*(RS_METH_00078, RS_METH_00079)*

[TR_AMETH_00001] Develop Service Interfaces ⌈ During this activity, services for service-oriented communication are specified, i.e., particular events, methods and fields per interface. It may be done independently of any assignation to specific software components or any instantiation. In this respect it may be seen as a preparation step towards the development of Adaptive Application Software entities.

This use case is elaborated in section 2.2.1. ⌋*(RS_METH_00201, RS_METH_00032)*

[TR_AMETH_00207] Design communication between Classic Platform ECUs and Adaptive Platform machines ⌈ Adaptive Applications communicate in a service oriented manner. However, a typical vehicle will also be equipped with ECUs developed for the Classical Platform. Thus, it is very likely that ECUs of different types need to communicate.

In case that the Classic Platform ECU implements SOME/IP they can communicate in service oriented way. However, in order to describe this kind of communication a mapping between the elements of the ServiceInterface and the corresponding elements of the respective PortInterface of the Classic Platform needs to be specified. This use case is elaborated in section 2.2.2.1.

If the counterpart on a Classic Platform ECU, however, communicates only in a signal-based way, a Signal-to-Servicetranslation is needed. This use case is elaborated in section 2.2.2.2. ⌋*(RS_METH_00202)*

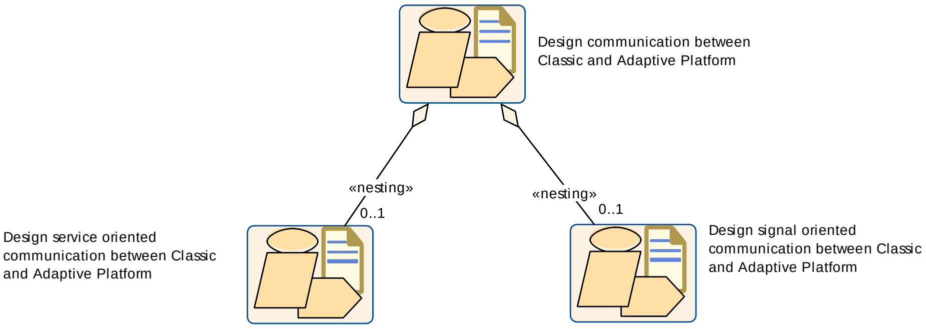
Design communication between Classic and Adaptive Platform
«nesting»
0..1
«nesting»
0..1
Design service oriented communication between Classic and Adaptive Platform
Design signal oriented communication between Classic and Adaptive Platform

Figure 2.3: Design Communication between Classic Platform and Adaptive Platform

Activity	**Design communication between Classic and Adaptive Platform**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Communication		
Brief Description	Design communication between CP and AP		
Description	Higher level activity that encloses all activities which are necessary to design communication between a Classic Platform (ECU) and a Adaptive Platform.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Design service oriented communication between Classic and Adaptive Platform	0..1	
Aggregates	Design signal oriented communication between Classic and Adaptive Platform	0..1	

Table 2.1: Design communication between Classic and Adaptive Platform

2.1.2.2.3 System

Like for the CP methodology [1], this development domain will cover activities which refine the `Common Software Architecture` into a system defined by specific `ECU`s or `machine`s. In this respect, the main activities/issues specified there will be in principle also valid here (see Figure 2.4).

[TR_AMETH_00204] Develop the System ⌈

The subsequent specifications of the Classic Platform methodology shall also be applicable for the Adaptive Platform (by following their general meanings):

- *Development of the System (TR_METH_01046)* and *(Develop) the overall system (TR_METH_01048)*, which talk about the refinement of the `VFB` by the definition of a topology of `ECU`s and networks and the deployment of software components onto `ECU`s, with the extensions necessary for the `Common Software Architecture` and the additions to specify `machine`s and the corresponding mapping of `machine`s to `ECU`s.

- *Two phase development approach (TR_METH_01047)* and *Interaction between organizations (TR_METH_01049)*, which structures the collaboration between different parties, like between OEMs and their suppliers.

⌋*(RS_METH_00041)*

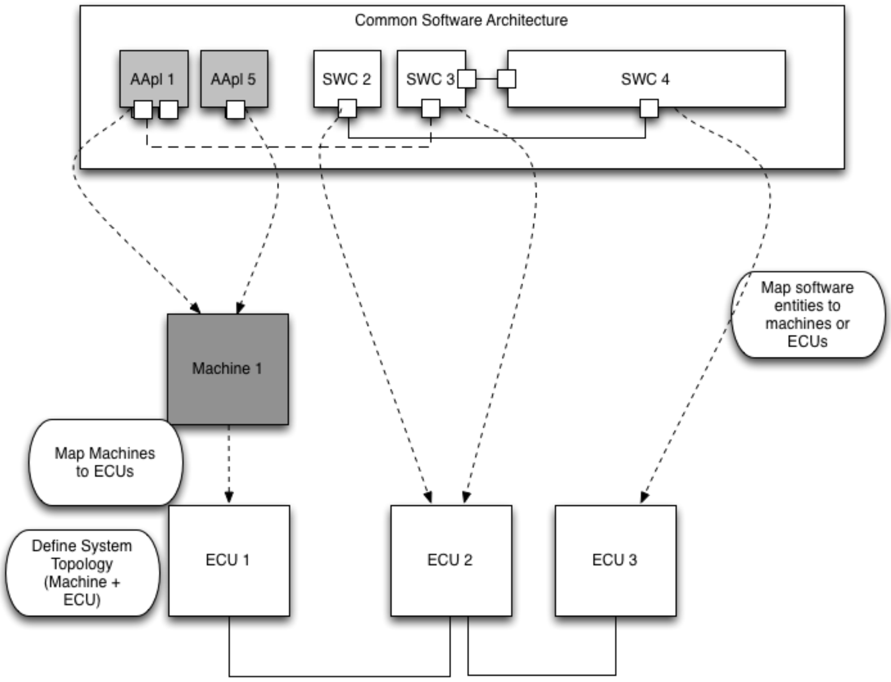
Common Software Architecture
AApl 1
AApl 5
SWC 2
SWC 3
SWC 4
Map software entities to machines or ECUs
Machine 1
Map Machines to ECUs
Define System Topology (Machine + ECU)
ECU 1
ECU 2
ECU 3

Figure 2.4: System development: `ECUs`, `machines`, communication networks, mapping of software entities to `ECUs` or `machines`

2.1.2.2.4 Software Development

[TR_AMETH_00002] Develop the software for `AdaptiveAutosarApplications`

⌈Once the service interfaces have been defined, software for `AdaptiveAutosarApplications` of category application-level and platform-level can be developed. The development may include several sub-activities like analysis, design, implementation or test.

The most important artifacts of this activity are either source-code or object-code files, depending on whether or not the developer knows the `Build Chain Configuration` beforehand. The artifacts are handed over to an integrator.

Sections 2.3.1 and 2.3.2 will refine the necessary activities associated with the development of application-level and platform-level software. ⌋*(RS_METH_00202, RS_METH_00032)*

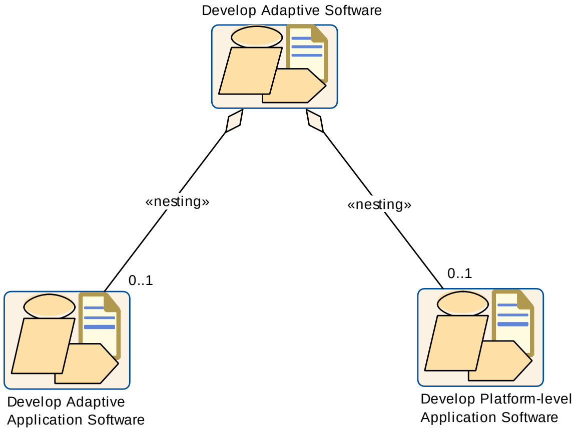
Develop Adaptive Software
«nesting»
«nesting»
0..1
0..1
Develop Adaptive Application Software
Develop Platform-level Application Software

Figure 2.5: Develop Adaptive Software

Activity	**Develop Adaptive Software**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Develop Adaptive Application		
Brief Description	Develop Adaptive Software		
Description	This higher level activity encloses the development of Adaptive Applications with category application-level as well as platform-level.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Develop Adaptive Application Software	0..1	
Aggregates	Develop Platform-level Application Software	0..1	

Table 2.2: Develop Adaptive Software

2.1.2.2.5 Integration and Deployment

The term *Integration and deployment of software* (on the Adaptive Platform) refers to all activities that are necessary to make designated software run on a specific machine, determined by its hardware, connected networks, its operating system and (some) Functional Clusters , in order to satisfy all requirements.

[TR_AMETH_00205] Integrate Software ⌈ An integrator will either take source-code or object-code files delivered by the software development and will bind them together in order to form an Executable for a specific machine and notably its application binary interface (ABI).

This activity does not include instantiation, i.e., the binding of an actual Executable to the context of an Process (exactly one Executable per Process).

Section 2.4.1 will refine the necessary activities associated with the integration of software. ⌋*(RS_METH_00202, RS_METH_00032)*

[TR_AMETH_00211] Pool Executables together to form ExecutableGroups ⌈ Executables may be pooled together to form ExecutableGroups in order to serve as deliverables (inputs) for deployment. ⌋*(RS_METH_00020, RS_METH_00018)*

[TR_AMETH_00003] Configuration of the machine ⌈ In AUTOSAR adaptive the meta model element Machine already represents a specific ECU implementation with dedicated configurations. In this respect, the Machine is more a model entity in the scope of an integrator of a Tier 1 company, than in the scope of on an communication designer of an OEM.

Therefore, the meta model element MachineDesign has been introduced. It allows a communication designer of an OEM to define requirements on a machine in the context of a System during the system design stage. In this sense, MachineDesign acts as a placeholder for a real adaptive ECU instance in early development phases.

In addition, the respective Machine Design will be uploaded onto the machines as part of Uploadable Design Artifacts. Since a particular Machine model will reference a particular MachineDesign model, the configurations of Machine Design will also contribute to the Machine Manifest.

Thus, the configuration of the machine is subdivided into two process steps:

1. The first step is the configuration of the communication structure of a prospective machine and will be performed by a communication designer of an OEM as part of the (system) design phase. It will result in an Machine Design. This step results in a Machine Design. See Section 2.2.3 for details.

2. The second step covers activities and tasks for the configuration of a real adaptive ECU. It will be executed by an integrator of a Tier 1 company. The resulting configuration is then part of the actual result Machine Manifest. Section 2.4.2 is dedicated to this step and gives details.

⌋*(RS_METH_00204, RS_METH_00203)*

[TR_AMETH_00004] Creation of the Application Manifest ⌈ Executables of an AdaptiveAutosarApplication are instantiated by means of the Application Manifest. Instantiation here means to bind the executables to the context of specific processes of the operating system. Each process may start with a different start-up configuration depending on a machine mode. Further on, the Application Manifest may also define dependencies of processes.

The creation of the Application Manifest is detailed in Section 2.4.3. ⌋*(RS_METH_00203)*

[TR_AMETH_00005] Configuration of the service instances ⌈ During this activity, the service instances are configured, notably the binding of the service interfaces to a chosen transport layer, whether a specific service instance is either provided or required and the mapping to a dedicated machine. The configurations of the service instance are manifested in the Service Instance Manifest.

The details are given in Section 2.4.4 ⌋*(RS_METH_00206, RS_METH_00203)*

[TR_AMETH_00006] Deployment of the application software ⌈ Software is deployed to a machine, i.e., a particular Adaptive AUTOSAR Platform instance, by means of `Software Package`s. This means that:

1. associated software artifacts need to be compiled into a dedicated `Software Package` (see Sections 2.4.6)
2. `Software Package`s are provided by an OEM-specific Back-end server in order to be accessible by the machines in the field (see Section) 2.4.7

⌋*(RS_METH_00205)*

2.1.2.3 Workflow

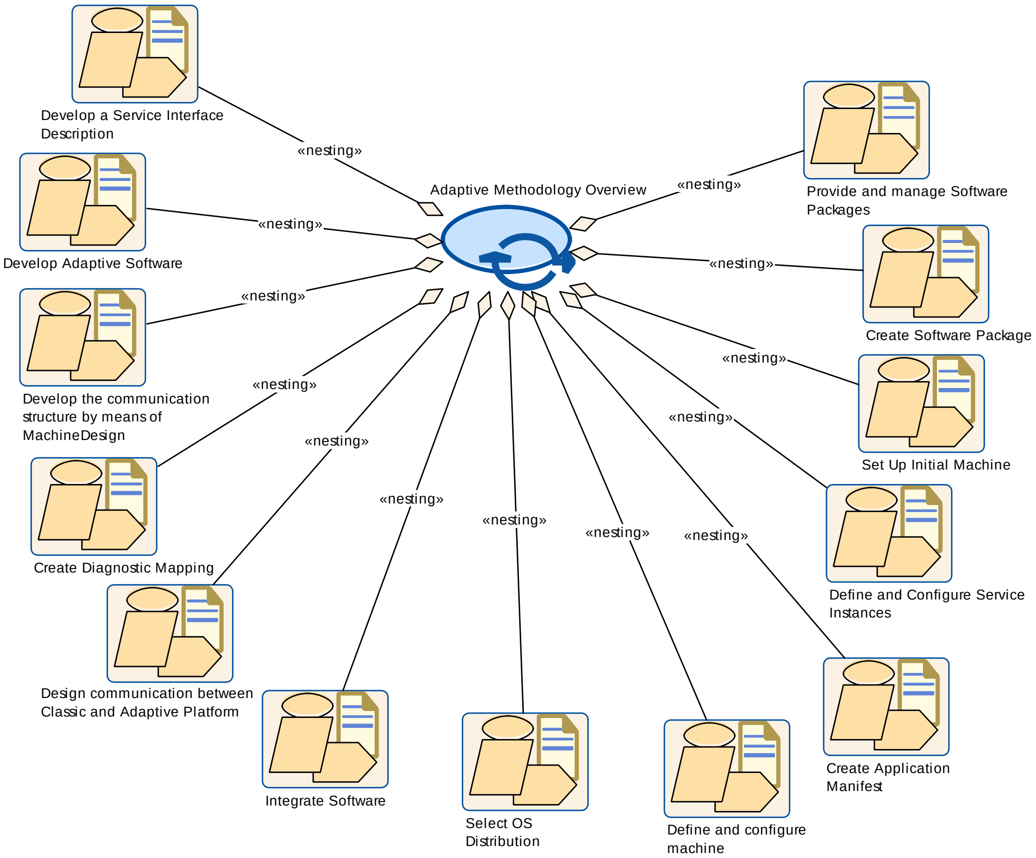
Adaptive Methodology Overview
«nesting»
Develop a Service Interface Description
Develop Adaptive Software
Develop the communication structure by means of MachineDesign
Create Diagnostic Mapping
Design communication between Classic and Adaptive Platform
Integrate Software
Select OS Distribution
Define and configure machine
Create Application Manifest
Define and Configure Service Instances
Set Up Initial Machine
Create Software Package
Provide and manage Software Packages

Figure 2.6: Adaptive Methodology Overview: Overall Structure

Process Pattern	**Adaptive Methodology Overview**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Adaptive Methodology Overview		
Brief Description	High-level view of the adaptive AUTOSAR methodology		
Description	This process pattern covers the typical activities to develop an Adaptive AUTOSAR system.		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Create Application Manifest	1	
Aggregates	Create Diagnostic Mapping	1	
Aggregates	Create Software Package	1	
Aggregates	Define and Configure Service Instances	1	
Aggregates	Define and configure machine	1	
Aggregates	Design communication between Classic and Adaptive Platform	1	
Aggregates	Develop Adaptive Software	1	
Aggregates	Develop Platform-level Application Software	1	
Aggregates	Develop a Service Interface Description	1	
Aggregates	Develop the communication structure by means of MachineDesign	1	
Aggregates	Integrate Software	1	
Aggregates	Provide and manage Software Packages	1	
Aggregates	Select OS Distribution	1	
Aggregates	Set Up Initial Machine	1	

Table 2.3: Adaptive Methodology Overview

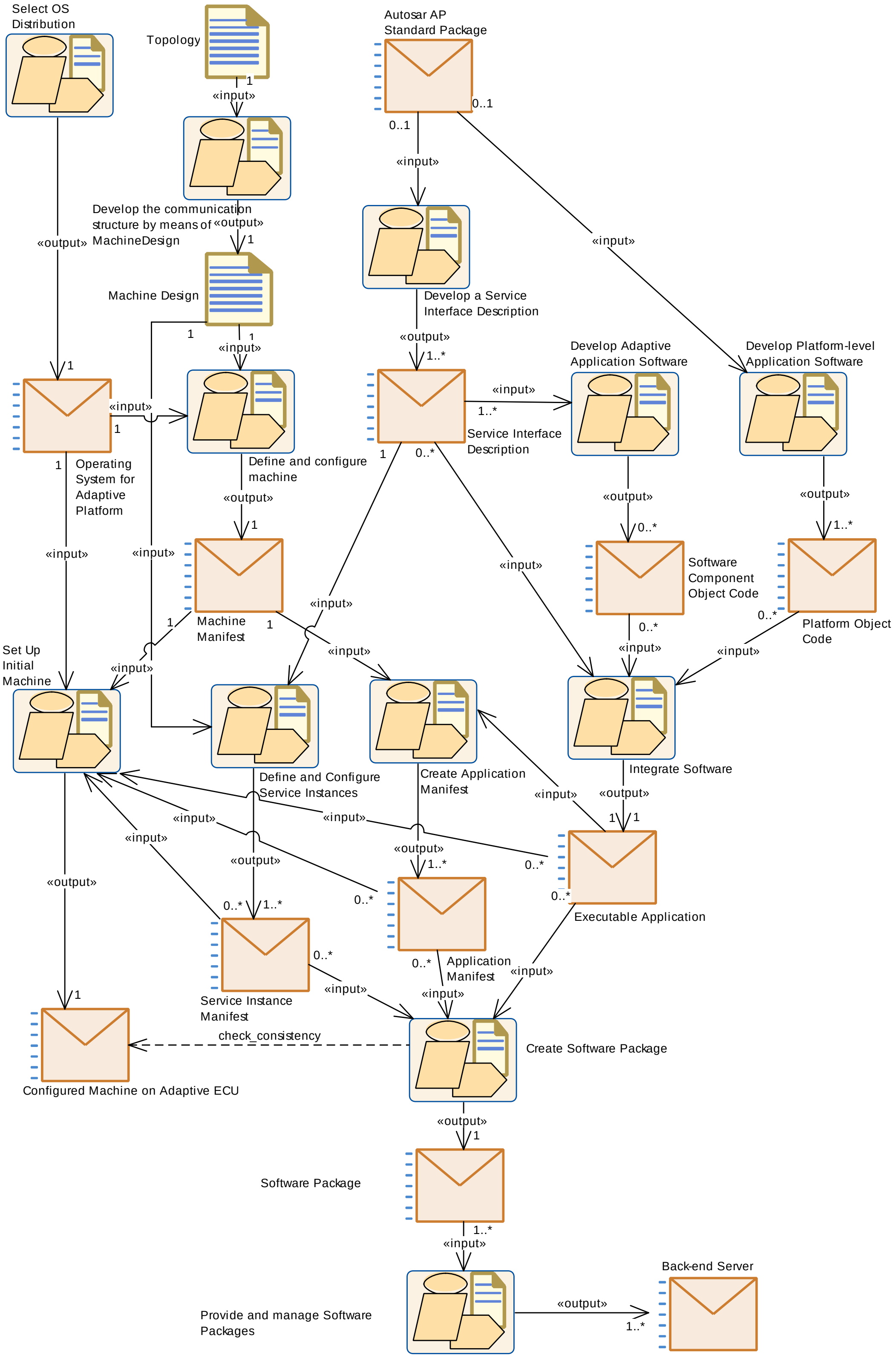
Select OS Distribution
Topology
Autosar AP Standard Package
Develop the communication structure by means of MachineDesign
Machine Design
Develop a Service Interface Description
Develop Adaptive Application Software
Develop Platform-level Application Software
Operating System for Adaptive Platform
Define and configure machine
Service Interface Description
Software Component Object Code
Platform Object Code
Machine Manifest
Set Up Initial Machine
Define and Configure Service Instances
Create Application Manifest
Integrate Software
Executable Application
Application Manifest
Service Instance Manifest
Configured Machine on Adaptive ECU
check_consistency
Create Software Package
Software Package
Provide and manage Software Packages
Back-end Server
«input»
«output»
0..1
1
1..*
0..*

Figure 2.7: Adaptive Methodology Overview: Workflow

2.2 Architecture and Design

2.2.1 Develop a Service Interface Description

2.2.1.1 Purpose

This use case gives an outline of the definition of the services in a system, independent of any instantiation. All relevant tasks and deliverables for this use case are given in Figure 2.8. The workflow is depicted in Figure 2.9.

2.2.1.2 Description

[TR_AMETH_00007] Definition of data types for the Adaptive Platform ⌈ Data types for the Adaptive Platform can be defined based on standardized data types from AUTOSAR. As on the Classic Platform, data types are defined on different levels of abstractions: application data types, implementation data types and base types. Most concepts and data types can be taken over from the Classic Platform. However, in order to cope with the C++ programming language, for the Adaptive Platform also vectors, strings and maps can be defined. ⌋*(RS_METH_00201)*

For more information on data types as specified for the Classic Platform and the extensions for the Adaptive Platform, see [5] and [6].

[TR_AMETH_00008] Definition of service interfaces for the Adaptive Platform ⌈ All service interfaces, which are used in a system, need to be defined. Service interfaces aggregate elements as events, methods and fields. They are the basis for the header file generation. Therefore, it is also possible to define namespaces within a service interface, which has a direct influence on the generated code. ⌋ *(RS_METH_00201)*

[TR_AMETH_00009] Aggregating service interfaces for reducing the bus load ⌈ Optionally, service interfaces can be aggregated to more coarse-grained service interfaces by defining a service interface mapping or a service interface element mapping respectively. This results in an update of the `Service Interface Description`. The newly defined coarse-grained service interfaces are then used for the network-based communication. ⌋*(RS_METH_00201)*

2.2.1.3 Workflow

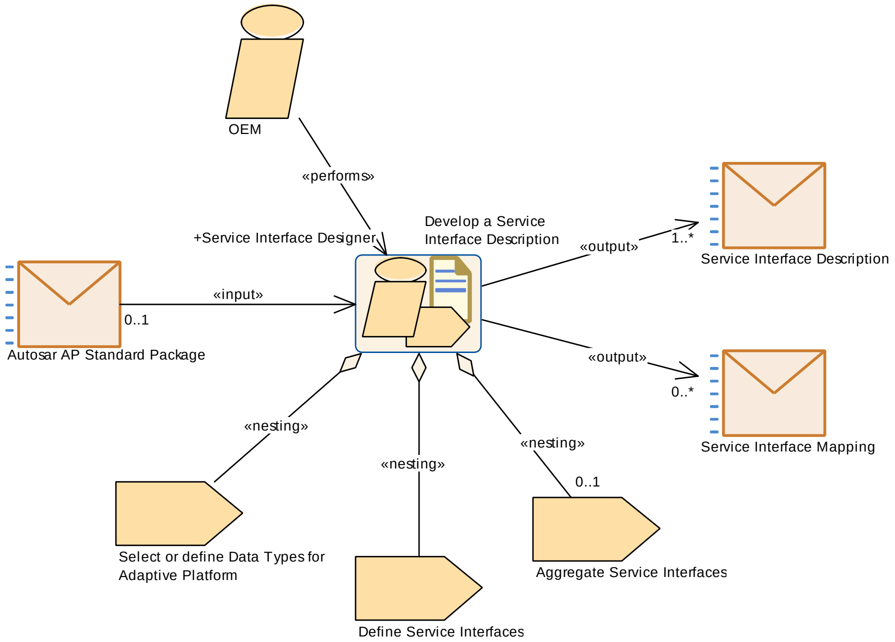
OEM
«performs»
+Service Interface Designer
Develop a Service Interface Description
«output»
1..*
Service Interface Description
«input»
0..1
Autosar AP Standard Package
«output»
0..*
Service Interface Mapping
«nesting»
«nesting»
«nesting»
0..1
Select or define Data Types for Adaptive Platform
Define Service Interfaces
Aggregate Service Interfaces

Figure 2.8: Develop a Service Interface Description

Activity	**Develop a Service Interface Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Service Interface Definition		
Brief Description	Define all service interfaces used in the system		
Description	This activity describes the definition of the service interfaces, aggregating events, methods and fields, including the definition of data types. In addition, coarse-grained service interfaces can be defined for the network-based communication.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Autosar AP Standard Package	0..1	Optional input for defining data types and service interfaces for the adaptive platform
Produces	Service Interface Description	1..*	All service interfaces, which are used for communication
Produces	Service Interface Mapping	0..*	Optionally, coarse-grained service interfaces are defined by a service interface mapping
Aggregates	Aggregate Service Interfaces	0..1	
Aggregates	Define Service Interfaces	1	
Aggregates	Select or define Data Types for Adaptive Platform	1	
Performed by	OEM	1	Service Interface Designer: This activity will probably be performed by a Service Interface Designer

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

Table 2.4: Develop a Service Interface Description

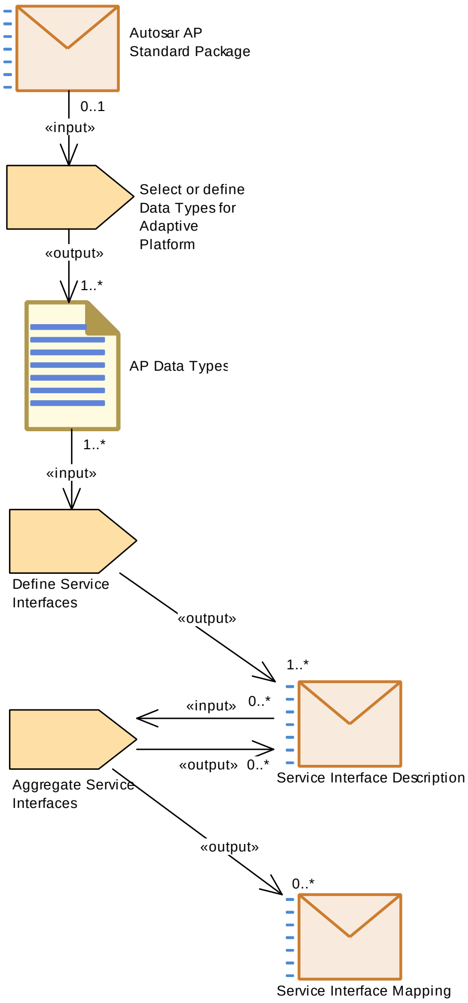
Autosar AP Standard Package
0..1
«input»
Select or define Data Types for Adaptive Platform
«output»
1..*
AP Data Types
1..*
«input»
Define Service Interfaces
«output»
1..*
«input»
0..*
«output»
0..*
Aggregate Service Interfaces
Service Interface Description
«output»
0..*
Service Interface Mapping

Figure 2.9: Workflow for defining Service Interfaces

2.2.2 Design communication between Classic Platform and Adaptive Platform

2.2.2.1 Design service oriented communication between Classic Platform and Adaptive Platform

2.2.2.1.1 Purpose

This use case covers the activities necessary to design service oriented communication between applications of a Classic Platform ECU and those of an Adaptive Platform machine via SOME/IP.

The respective deliverables, activities and tasks are depicted in Figure 2.10.

2.2.2.1.2 Description

[TR_AMETH_00208] Design service oriented communication between `Classic Platform` and `Adaptive Platform` ⌈ The background of this activity is the request to enable service oriented communication between applications of a Classic Platform (CP) ECU and those of an Adaptive Platform (AP) machine via SOME/IP.

Unfortunately, the AUTOSAR Classic Platform does not support `ServiceInterface`s. Thus, a `SOME/IP` service may be composed of different types of Classic Platform `PortInterface`s like `SenderReceiverInterface`s, `ClientServiceInterface`s or `TriggerInterface`s.

In order to describe the communication over `SOME/IP` between the `CP ECU` and an `AP machine`, this activity describes the mapping of the elements of the `PortInterface`s of the Classical Platform to the elements of a single `ServiceInterface` of the Apdaptive Platform.

Thus, the main objective of this activity is to map a single `ServiceInterface` to `PortInterface` elements, in detail:

- to map method(s), i.e., to map a `ClientServerOperation` located in a `ClientServerInterface` to a method located in a `ServiceInterface`.
- to map event(s), i.e., to map a `VariableDataPrototype` located in a `SenderReceiverInterface` to an event located in a `ServiceInterface`.
- to map field(s), i.e., to map operations located in `ClientServerOperation`s to getter and setter methods of a `ServiceInterface` and to map a `VariableDataPrototype` of a `SenderReceiverInterface` to the field notifier of the `ServiceInterface`.
- to map “Fire and Forget”, i.e., to map a “Fire and Forget” method located in a `ServiceInterface` to a `VariableDataPrototype` in a `SenderReceiverInterface` or to a trigger of a `TrigerInterface`.

The mapping description serves currently only for documentation.

⌋*(RS_METH_00200, RS_METH_00202)*

2.2.2.1.3 Workflow

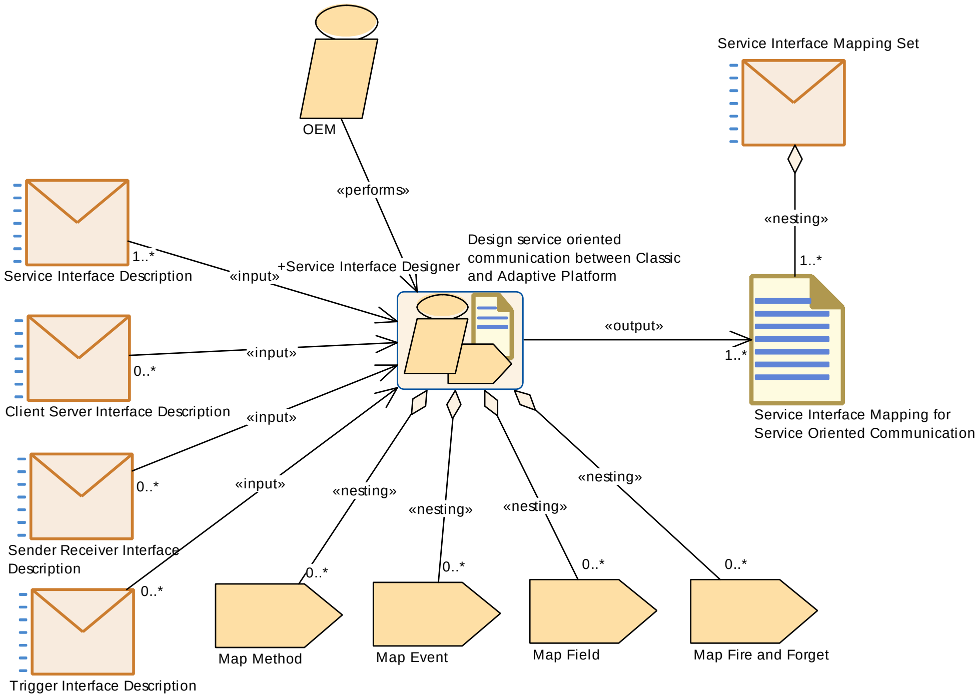
OEM
«performs»
+Service Interface Designer
Service Interface Mapping Set
«nesting»
1..*
Design service oriented communication between Classic and Adaptive Platform
«output»
1..*
Service Interface Mapping for Service Oriented Communication
1..*
Service Interface Description
«input»
0..*
Client Server Interface Description
«input»
«input»
0..*
Sender Receiver Interface Description
«input»
0..*
Trigger Interface Description
«nesting»
«nesting»
«nesting»
«nesting»
0..*
0..*
0..*
0..*
Map Method
Map Event
Map Field
Map Fire and Forget

Figure 2.10: Design service oriented communication

Activity	**Design service oriented communication between Classic and Adaptive Platform**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Communication		
Brief Description	Design service oriented communication between CPand AP		
Description	The background of this activity is the request to enable service oriented communication between applications of a Classic Platform (CP) ECU and those of an Adaptive Platform (AP) machine via SOME/IP. Unfortunately, the AUTOSAR Classic Platform does not support ServiceInterfaces. Thus, a SOME/IP service may be composed of different types of Classic Platform PortInterfaces like SenderReceiverInterfaces, ClientServiceInterfaces or TriggerInterfaces. In order to describe the communication over SOME/IP between the CP ECU and a AP machine, this activity describes the mapping of the elements of the PortInterfaces of the Classical Platform to the elements of a single ServiceInterface of the Apdaptive Platform. The mapping description serves currently only for documentation.		
Relation Type	**Related Element**	**Mul.**	**Note**

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Client Server Interface Description	0..*	The descriptions of Client Server Interfaces of CP are used to map a ClientServerOperation to a method in a ServiceInterface or to map a ClientServerOperation (representing getter or setter methods) to a field in a ServiceInterface
Consumes	Sender Receiver Interface Description	0..*	The descriptions of Sender Receiver Interfaces of CP are used to map a VariableDataPrototype to an Event in a ServiceInterface or to map a VariableDataPrototype to the notifier of a Field of a ServiceInterface or to map a Fire&Forget Method that is located in a ServiceInterface to a VariableDataPrototype in a SenderReceiverInterface
Consumes	Service Interface Description	1..*	Description of the Service Interfaces which communicate to CP in a service-oriented manner
Consumes	Trigger Interface Description	0..*	The descriptions of Trigger Interfaces are used to map a Fire&Forget Method that is located in ServiceInterface to a Trigger in a TriggerInterface
Produces	Service Interface Mapping for Service Oriented Communication	1..*	An InterfaceMapping results from the design of service-oriented communication between CP and AP
Aggregates	Map Event	0..*	
Aggregates	Map Field	0..*	
Aggregates	Map Fire and Forget	0..*	
Aggregates	Map Method	0..*	
Performed by	OEM	1	Service Interface Designer: This activity will probably be performed by a Service Interface Designer of an OEM

Table 2.5: Design service oriented communication between Classic and Adaptive Platform

2.2.2.2 Design signal oriented communication between `Classic Platform` and `Adaptive Platform`

2.2.2.2.1 Purpose

This use case comprises activities to specify a signal oriented communication between `Classic Platform` and `Adaptive Platform` applications, if there is no service oriented communication possible.

The associated elements, i.e, deliverables, activities and tasks and their relations are depicted in Figure 2.11.

2.2.2.2.2 Description

[TR_AMETH_00209] Define a signal-based ServiceInterface ⌈ As a prerequisite for the mapping of ServiceInterface elements to ISignalTriggerings, the definition of a SignalBasedServiceInterfaceDeployment is needed. It specifies the configuration settings for a ServiceInterface from which the content will be transmitted in the signal-based way over a communication medium and therefore provides the ability to bind a ServiceInterface to a signal-based communication protocol like CAN or FlexRay.

Details are provided by the specifications TPS_MANI_03120, TPS_MANI_03121, TPS_MANI_03122 and TPS_MANI_03123 of the Manifest specification [6]. ⌋ *(RS_METH_00200, RS_METH_00202)*

[TR_AMETH_00210] Map signals to services ⌈ In a second step, the mapping of ServiceInstance elements of a specific AdaptivePlatformServiceInstance defined in the context of a process to ISignalTriggerings is described, in detail:

- to map SignalBasedMethodDeployment to ISignalTriggerings, according to TPS_MANI_03125 of the Manifest specification [6]
- to map SignalBasedEventDeployment to ISignalTriggerings, according to TPS_MANI_03124 of the Manifest specification [6]
- to map SignalBasedFieldDeployment to ISignalTriggerings, according to TPS_MANI_03126 of the Manifest specification [6]
- to map a ServiceInstance to a PortPrototype, according to TPS_MANI_03000 of the Manifest specification [6]

⌋ *(RS_METH_00200, RS_METH_00202)*

2.2.2.2.3 Workflow

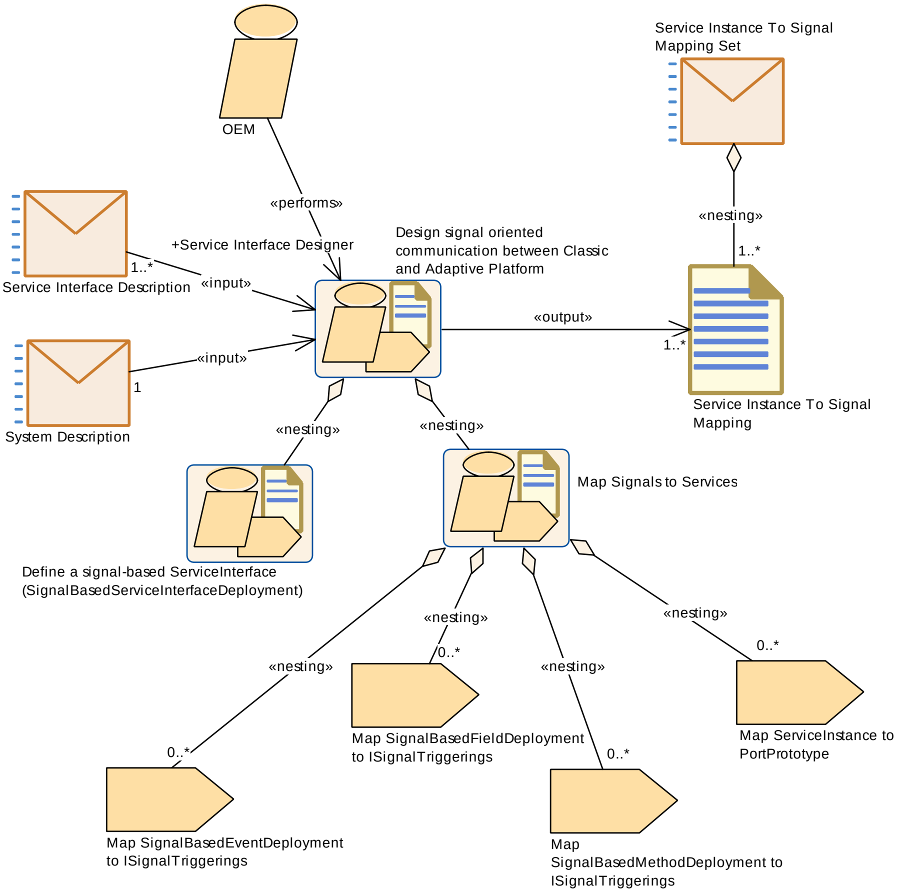
OEM
«performs»
+Service Interface Designer
Service Instance To Signal Mapping Set
«nesting»
1..*
Service Interface Description
1..*
«input»
Design signal oriented communication between Classic and Adaptive Platform
«output»
1..*
«input»
1
System Description
Service Instance To Signal Mapping
«nesting»
«nesting»
Map Signals to Services
Define a signal-based ServiceInterface (SignalBasedServiceInterfaceDeployment)
«nesting»
«nesting»
0..*
«nesting»
«nesting»
0..*
Map SignalBasedFieldDeployment to ISignalTriggerings
0..*
0..*
Map ServiceInstance to PortPrototype
Map SignalBasedEventDeployment to ISignalTriggerings
Map SignalBasedMethodDeployment to ISignalTriggerings

Figure 2.11: Design signal oriented communication

Activity	**Design signal oriented communication between Classic and Adaptive Platform**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Communication		
Brief Description	Design signal oriented communication between CP and AP		
Description	Usually, Adaptive Applications communicate between each other in a service oriented manner. There is even an option that applications deployed to an Adaptive Platform and Classic Platform communicate in a service oriented way via SOME/IP. If the counterpart on a Classic Platform ECU, however, communicates only in a signal-based way, a Signal-to-Service translation is needed. This activity encompasses the description of the mapping of signals to elements of a particular ServiceInterface. It will be the base for the configuration of the translation application.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Service Interface Description	1..*	Description of the Service Interfaces which communicate to CP in a signal-oriented manner

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	System Description	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
Produces	Service Instance To Signal Mapping	1..*	A signal-to-service mapping results from the design of signal-oriented communication between CP and AP
Aggregates	Define a signal-based Service Interface (Signal BasedService InterfaceDeployment)	1	
Aggregates	Map Signals to Services	1	
Performed by	OEM	1	Service Interface Designer: This activity will probably be performed by a Service Interface Designer of an OEM

Table 2.6: Design signal oriented communication between Classic and Adaptive Platform

Activity	**Map Signals to Services**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Communication		
Brief Description	Map Signals to Services		
Description	Describe the mapping of ServiceInstance elements of a specific AdaptivePlatformServiceInstance defined in the context of a process to ISignalTriggerings. The prerequisite is the definition of the SignalBasedServiceInterface.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Map ServiceInstance to Port Prototype	0..*	
Aggregates	Map SignalBased EventDeployment to ISignal Triggerings	0..*	
Aggregates	Map SignalBased FieldDeployment to ISignalTriggerings	0..*	
Aggregates	Map SignalBased MethodDeployment to ISignal Triggerings	0..*	

Table 2.7: Map Signals to Services

Activity	**Define a signal-based ServiceInterface (SignalBasedServiceInterfaceDeployment)**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Communication		
Brief Description	Define SignalBasedServiceInterface		
Description	Express that a ServiceInterface will be transmitted via a signal-based communication protocol like CAN or FlexRay.		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Service Interface Description	1..*	Description of the Service Interfaces
Consumes	System Description	1	The System Description based on the System Template on the AUTOSAR classic platform

Table 2.8: Define a signal-based ServiceInterface (SignalBasedServiceInterfaceDeployment)

2.2.3 Develop the communication structure by means of Machine Design

2.2.3.1 Purpose

By means of this activity, an OEM specifies the communication structure as well as corresponding configuration parameters of prospective machines, already during the (system) design phase.

2.2.3.2 Description

A primary task of an OEM is to specify entities which are associated with the topology, network and the system design, already in early development phases.

[TR_AMETH_00021] Define and configure the network communication for machine ⌈ This activity will cover the definition and configuration of the network communication for a prospective machine and consists of the following tasks:

- Define and configure the network connection of a prospective machine, i.e., define all network endpoint with corresponding IP address (IPv4 or IPv6)
- Configure the service discovery message exchange of a prospective machine, i.e., specify all designated multicast IP addresses and a UDP port

⌋*(RS_METH_00204, RS_METH_00203)*

The Machine is a model entity which already represents a specific ECU implementation with dedicated configurations. Therefore, it should not be used during system design.

The meta model element MachineDesign has been introduced in order to allow the communication designer to define a placeholder for an adaptive ECU (Machine) in the scope of the System. In this respect, the element MachineDesign corresponds to the EcuInstance of AUTOSAR classic.

Hence, the design activities of this step will result in a deliverable Machine Design, which will contribute to the Machine Manifest, since a particular Machine model will reference a particular MachineDesign model.

Since the configuration elements of Machine Design are needed during run-time, Machine Design needs to be uploaded to the target machine. Thus, Machine Design needs to be part of Uploadable Design Artifacts.

Figure 2.12 shows the involved entities – inputs, outputs, tasks – necessary to perform this activity.

2.2.3.3 Workflow

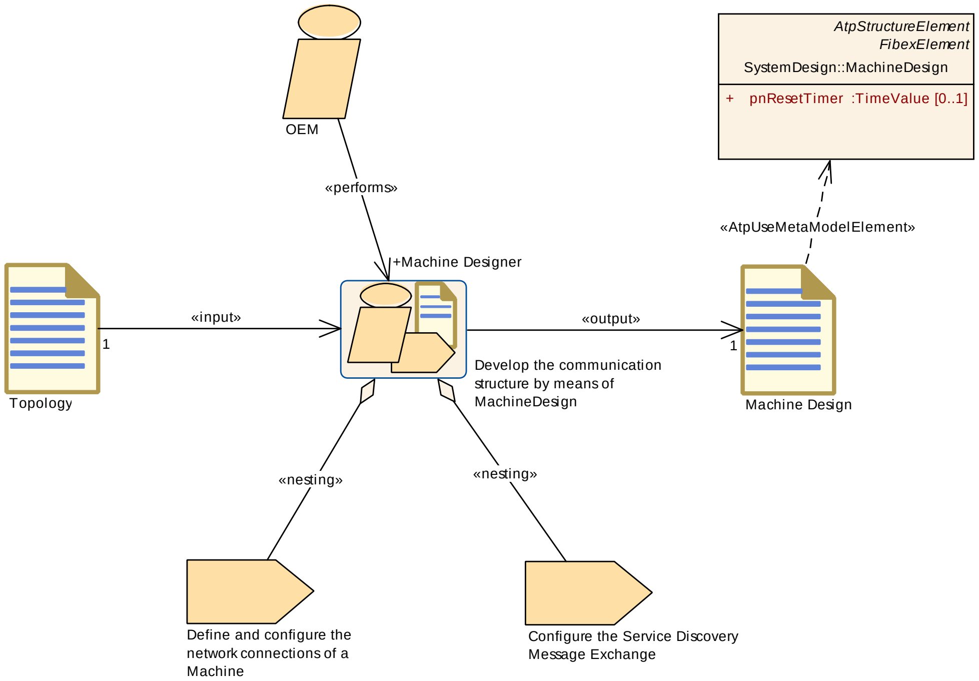
OEM
«performs»
+Machine Designer
AtpStructureElement
FibexElement
SystemDesign::MachineDesign
+ pnResetTimer :TimeValue [0..1]
«AtpUseMetaModelElement»
«input»
1
Topology
«output»
1
Machine Design
Develop the communication structure by means of MachineDesign
«nesting»
«nesting»
Define and configure the network connections of a Machine
Configure the Service Discovery Message Exchange

Figure 2.12: Develop the communication structure by means of Machine Design

Activity	**Develop the communication structure by means of MachineDesign**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Develop Machine Design		
Brief Description	placeholder during the design phase for an adaptive ECU(Machine) in the Scope of an System.		
Description	The Machine is a model entity which already represents a specific ECU implementation with dedicated configurations. Therefore, it should not be used during system design. The element MachineDesign has been introduced in order to allow the communication designer to define a placeholder for an adaptive ECU (Machine) in the scope of the System. The element MachineDesign corresponds to the EcuInstance of AUTOSAR classic, in this respect. This activity will aggregate the following tasks: • Define and configure the network connection of a prospective machine • Configure the service discovery message exchange of a prospective machine		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Topology	1	Description of (inter)connections between Machines.
Produces	Machine Design	1	Configuration settings of the network connections and service discovery network exchange of a Machine
Aggregates	Configure the Service Discovery Message Exchange	1	
Aggregates	Define and configure the network connections of a Machine	1	
Performed by	OEM	1	Machine Designer: This activity will probably be performed by a dedicated designer of an OEM.

Table 2.9: Develop the communication structure by means of MachineDesign

2.2.4 Create a Diagnostic Mapping

2.2.4.1 Purpose

This activity associates given diagnostic information (diagnostic data, diagnostic enable conditions, diagnostic events, diagnostic operation cycles) with the software structure (applications, instances, components, ports, events, data) of a particular machine.

2.2.4.2 Description

[TR_AMETH_00212] Design a diagnostic mapping ⌈ This activity covers all necessary tasks to perform the diagnostic mapping, except the task which associates corresponding ProcessDesign(s) and DiagnosticMapping(s).

These tasks are in detail:

- Map Diagnostic Data
- Map Diagnostic Enable Condition to Port(s)
- Map Diagnostic Event to Port(s)
- Map Diagnostic Storage Condition to Port(s)
- Diagnostic Software Mapping
- Map Diagnostic Operation Cycle to Port(s)

In order to perform the particular tasks, the following inputs are necessary:

- The `Diagnostic Machine Extract` that contains the diagnostic information
- `Service Interface Description` which collects the descriptions of the service interfaces with their events, methods and fields
- `Software Component Description for Adaptive Platform` which collects the description of software components and their ports

This step results in partly filled in artifact `Diagnostic Mapping`.

⌋*(RS_METH_00207, RS_METH_00201, RS_METH_00016)*

[TR_AMETH_00213] Relate diagnostic mappings to instances of Executables ⌈ It may be necessary that different instances of a particular application software (i.e., different Processes based on the very same Executable) require different diagnostic mappings. Therefore, a relation between a particular diagnostic mapping and a particular Process needs to be established. Since Processes at design do not exist, yet, the (meta) model element ProcessDesign may stand in as a proxy.

This assignment may be independent of the step of designing diagnostic mappings and may be done in a final extra step, separately; the corresponding task: Associate DiagnosticMapping with ProcessDesign.

To accommodate for this potential modeling, the reference from a diagnostic mapping to ProcessDesign has been decorated by stereotype «atpSplitable».

This step takes the partly filled in artifact `Diagnostic Mapping` and the artifact `ProcessDesign` as inputs and results in a completely filled in `Diagnostic Mapping`. ⌋*(RS_METH_00207, RS_METH_00201, RS_METH_00016)*

Figure 2.13 depicts an overview of diagnostic mapping; how the involved deliverables, activities and tasks are related to each other.

2.2.4.3 Workflow

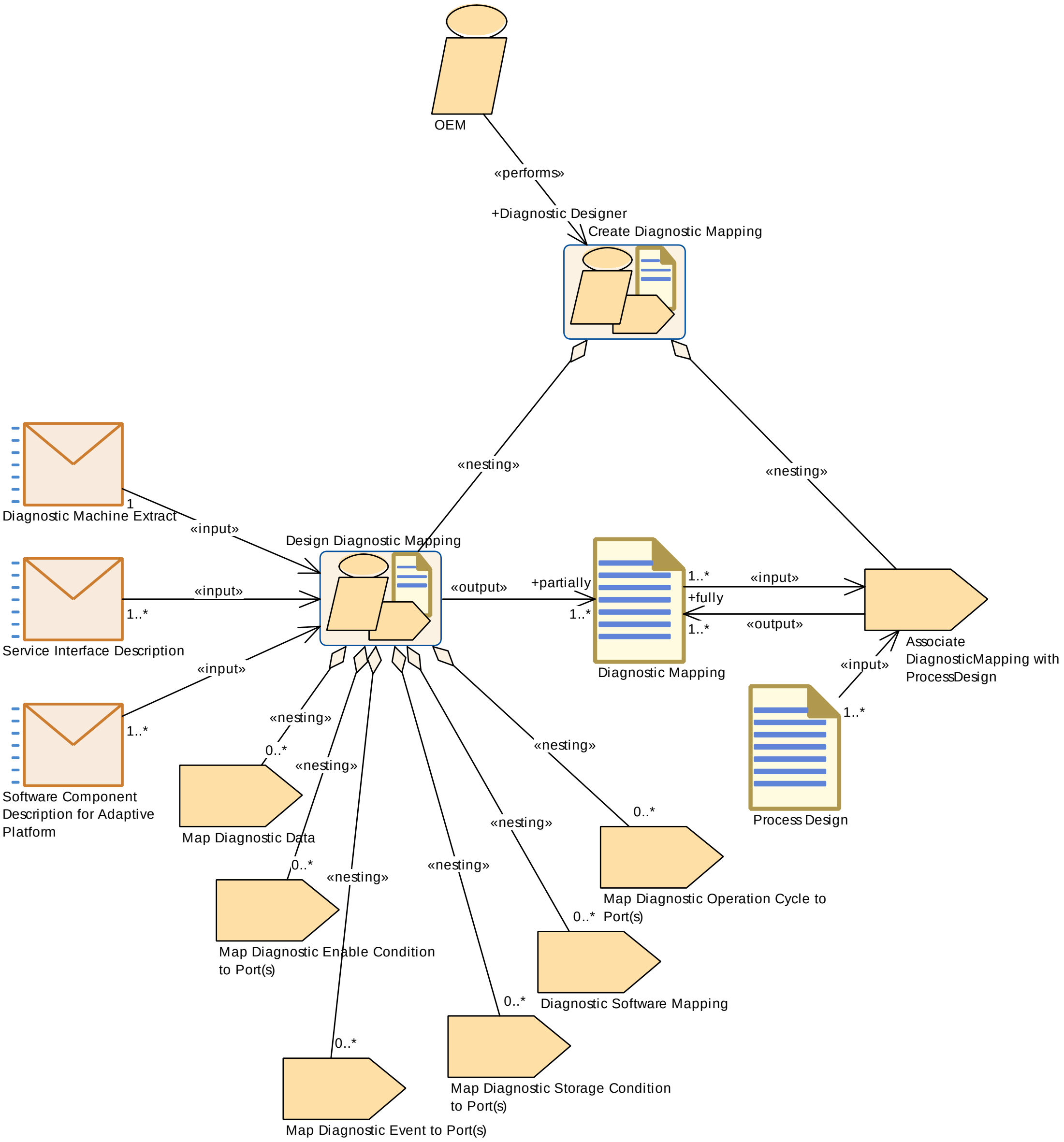
OEM
«performs»
+Diagnostic Designer
Create Diagnostic Mapping
«nesting»
«nesting»
Diagnostic Machine Extract
1
«input»
Design Diagnostic Mapping
Service Interface Description
«input»
1..*
«input»
Software Component Description for Adaptive Platform
1..*
«output»
+partially
1..*
Diagnostic Mapping
1..*
+fully
1..*
«input»
«output»
Associate DiagnosticMapping with ProcessDesign
«input»
1..*
Process Design
«nesting»
0..*
Map Diagnostic Data
«nesting»
0..*
Map Diagnostic Enable Condition to Port(s)
«nesting»
0..*
Map Diagnostic Event to Port(s)
«nesting»
0..*
Map Diagnostic Storage Condition to Port(s)
«nesting»
0..*
Diagnostic Software Mapping
«nesting»
0..*
Map Diagnostic Operation Cycle to Port(s)

Figure 2.13: Create a Diagnostic Mapping

Activity	**Create Diagnostic Mapping**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Diagnostic Mapping		
Brief Description	Create diagnsotic mappings		
Description	This activity comprises all necessary tasks to create complete diagnostic mappings. A diagnostic mapping is a formal model for the relation between the adaptive diagnostic management (module) and specific endpoints in the application software. This mapping enables the configuration of the service-oriented communication middleware, so that the service discovery can connect the corresponding endpoints correctly.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Associate DiagnosticMapping with Process Design	1	
Aggregates	Design Diagnostic Mapping	1	
Performed by	OEM	1	Diagnostic Designer: The activity of designing the diagnostic mapping will probably be performed by a Diagnostic Designer of an OEM

Table 2.10: Create Diagnostic Mapping

Activity	**Design Diagnostic Mapping**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Diagnostic Mapping		
Brief Description	Perform diagnostic mappings		
Description	This activity covers all necessary tasks to perform the diagnostic mapping, except the task which associates corresponding ProcessDesign(s) and DiagnosticMapping(s).		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Diagnostic Machine Extract	1	All available diagnostic information at the design time
Consumes	Service Interface Description	1..*	Collection of service interfaces. Service interfaces can consist of events, methods and fields.
Consumes	Software Component Description for Adaptive Platform	1..*	Description of a software component for the Adaptive Platform with all its ports, available at design time.
Produces	Diagnostic Mapping	1..*	partially: The diagnostic mapping for a Machine, except the linkage between the mappings and the corresponding ProcessDesigns
Aggregates	Diagnostic Software Mapping	0..*	
Aggregates	Map Diagnostic Data	0..*	

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Map Diagnostic Enable Condition to Port(s)	0..*	
Aggregates	Map Diagnostic Event to Port(s)	0..*	
Aggregates	Map Diagnostic Operation Cycle to Port(s)	0..*	
Aggregates	Map Diagnostic Storage Condition to Port(s)	0..*	

Table 2.11: Design Diagnostic Mapping

2.3 Software Development

2.3.1 Develop Adaptive Application Software

2.3.1.1 Purpose

This section explains how to develop application-level software for the Adaptive Platform. First, the design of the software components is described. Based on this description, the functionality can be implemented. An overview of all relevant tasks for this use case is given in Figure 2.14. The artifact-based workflow is depicted in Figure 2.15.

2.3.1.2 Description

[TR_AMETH_00010] Application-level Software ⌈ An Adaptive Application of category application-level is a collection of executables. The executables themselves can be derived from several software components. ⌋*(RS_METH_00202)*

[TR_AMETH_00011] Design of the software components ⌈ Based on the service interfaces, the development of adaptive application software starts with the design of the software components. The software components can have an hierarchical structure. For all software components it is defined if service interfaces are required or provided. This behavior is designed by using the corresponding ports for the software components.
⌋*(RS_METH_00202)*

[TR_AMETH_00012] Generation of the header files for service interface ⌈ In parallel, the header files for the service interfaces are generated. This step is independent of the design of the software component and therefore its ports. Instead, the header files are generated for all service interfaces and afterwards, the relevant ones are used for the development of the software component.

The generation includes the generation of service proxies and skeletons, which need to be implemented for a specific platform. ⌋*(RS_METH_00202, RS_METH_00066)*

[TR_AMETH_00013] Implementation and compilation of software components ⌈ The generated header files are the basis for the implementation of the core functionality of a software component. Two typical use cases for the development exist that depend on the fact if the Build Chain Configuration is known or not known and therefore if source code or object code is delivered by the application developer. ⌋*(RS_METH_00202, RS_METH_00015, RS_METH_00066, RS_METH_00042)*

[TR_AMETH_00014] Development with knowledge of the Build Chain Configuration ⌈ In this approach, the integrator hands over the Build Chain Configuration to the software developer beforehand. The software developer can build his software component against this build chain and can deliver object code back to the integrator. ⌋*(RS_METH_00202, RS_METH_00077)*

[TR_AMETH_00015] Development without knowledge of the Build Chain Configuration ⌈ For this use case, the application developer is not aware of the Build Chain Configuration and needs to deliver source code to the integrator. The integrator then takes over the compilation of the the software component. ⌋*(RS_METH_00202, RS_METH_00077)*

2.3.1.3 Workflow

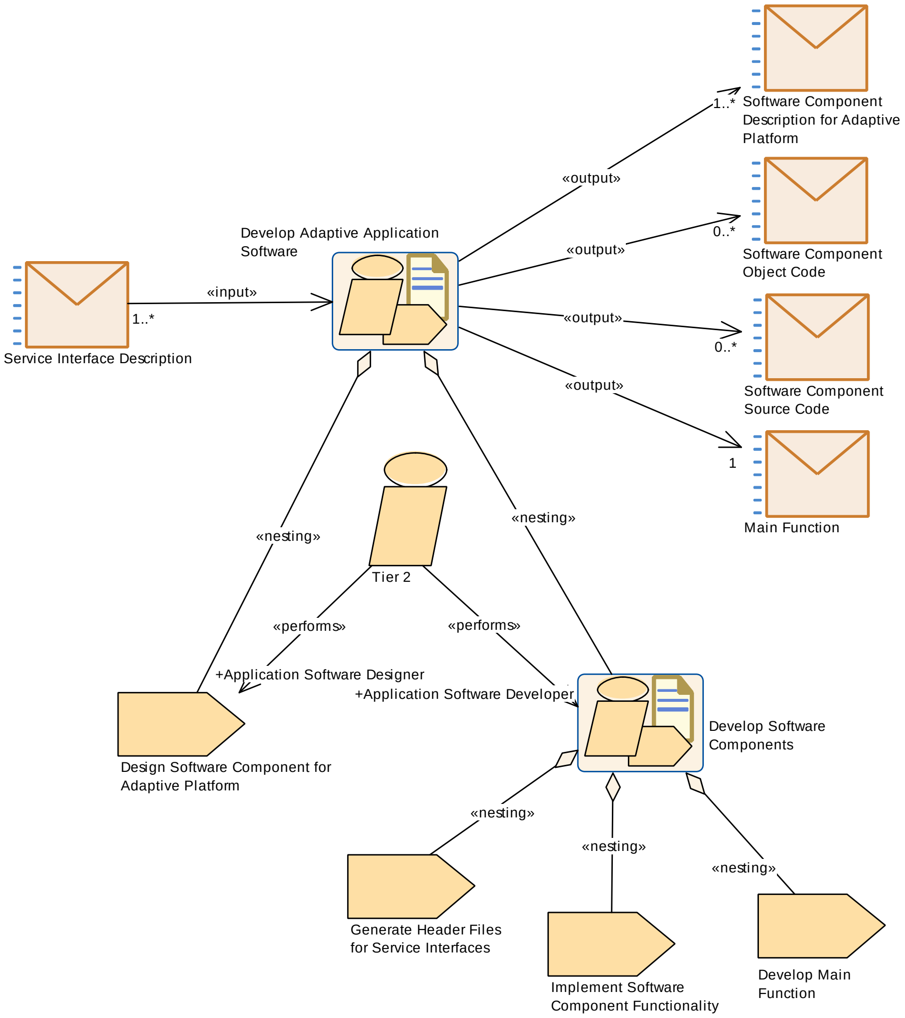
Software Component Description for Adaptive Platform
1..*
«output»
Software Component Object Code
0..*
«output»
Develop Adaptive Application Software
«input»
1..*
Service Interface Description
«output»
0..*
Software Component Source Code
«output»
1
Main Function
«nesting»
«nesting»
Tier 2
«performs»
«performs»
+Application Software Designer
+Application Software Developer
Develop Software Components
Design Software Component for Adaptive Platform
«nesting»
«nesting»
«nesting»
Generate Header Files for Service Interfaces
Implement Software Component Functionality
Develop Main Function

Figure 2.14: Develop Adaptive Application Software

Activity	**Develop Adaptive Application Software**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Develop Adaptive Application		
Brief Description	Design and development of software components for Adaptive Platform		
Description	Develop an Adaptive Application with category application-level. In this activity, Adaptive Application Software in terms of Software Component Object Code for the Adaptive Platform is developed. In addition, the main function for the executable is developed. The integration of these is done in the proceeding step. The software component description is needed as deliverable for a later mapping of service instances to port prototypes.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Service Interface Description	1..*	Service Interfaces are the basis for the development of adaptive application software
Produces	Main Function	1	One main function per executable is produced

Relation Type	*Related Element*	*Mul.*	*Note*
Produces	Software Component Description for Adaptive Platform	1..*	Output of component model for the software components
Produces	Software Component Object Code	0..*	Compiled software components
Produces	Software Component Source Code	0..*	Software components as source code
Aggregates	Design Software Component for Adaptive Platform	1	
Aggregates	Develop Software Components	1	

Table 2.12: Develop Adaptive Application Software

Activity	**Develop Software Components**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Develop Adaptive Application		
Brief Description	Implement the core functionality of one executable application		
Description	In this activity, the software components for one executable are implemented and compiled. After the header files for the service interfaces are generated, the functionality can be implemented. For each executable, a main function needs to be implemented, which defines the internal communication and scheduling.		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Develop Main Function	1	
Aggregates	Generate Header Files for Service Interfaces	1	
Aggregates	Implement Software Component Functionality	1	
Performed by	Tier 2	1	Application Software Developer: This activity will probably be performed by an Application Software Developer of a Tier 2 company

Table 2.13: Develop Software Components

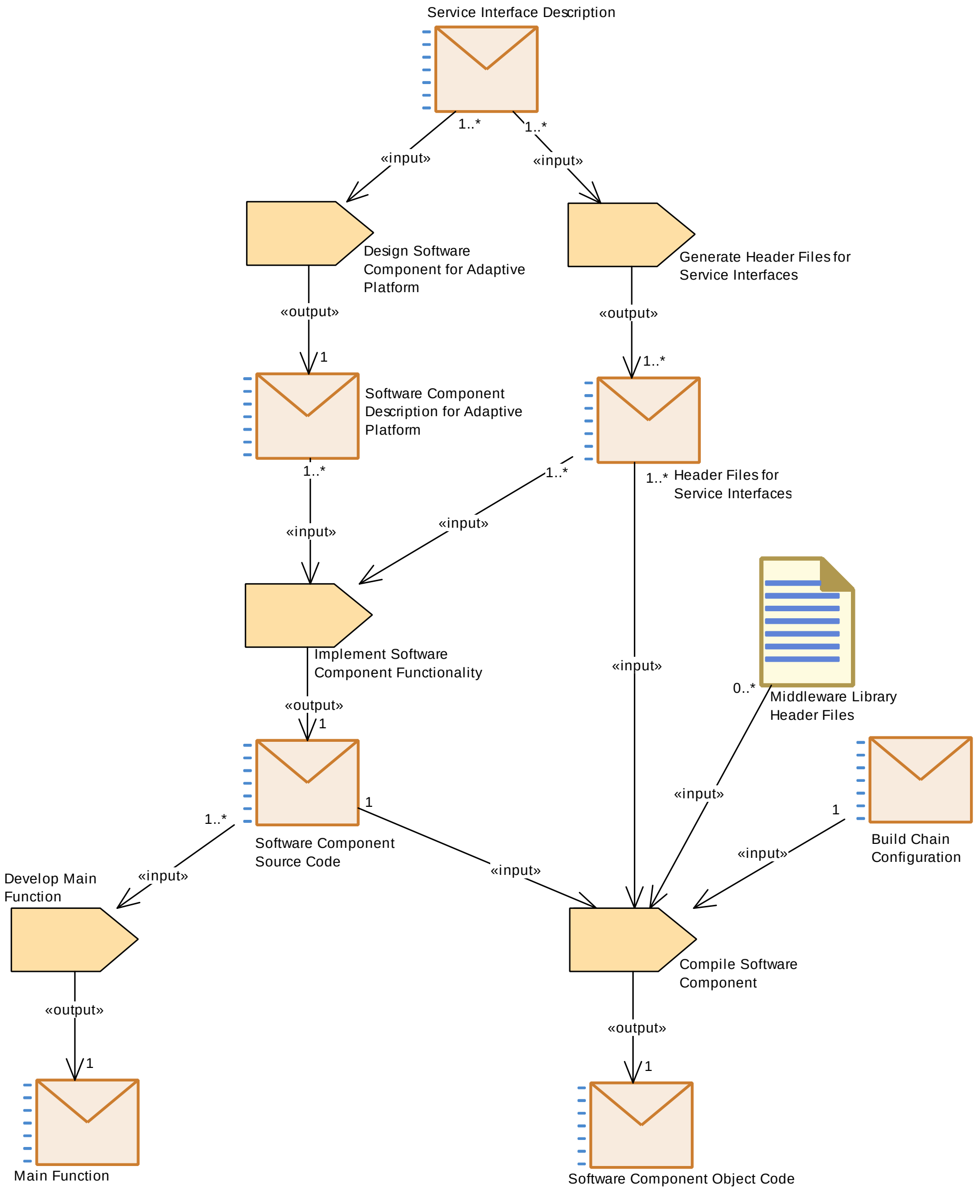
Service Interface Description
1..*
1..*
«input»
«input»
Design Software Component for Adaptive Platform
Generate Header Files for Service Interfaces
«output»
«output»
1
1..*
Software Component Description for Adaptive Platform
1..*
1..*
1..* Header Files for Service Interfaces
«input»
«input»
Implement Software Component Functionality
«input»
0..*
Middleware Library Header Files
«output»
1
1
Build Chain Configuration
1..*
1
«input»
«input»
Software Component Source Code
Develop Main Function
«input»
«input»
Compile Software Component
«output»
«output»
1
1
Main Function
Software Component Object Code

Figure 2.15: Workflow for developing application-level software for the Adaptive Platform

2.3.2 Develop Platform-level Application Software

2.3.2.1 Purpose

This section explains how to develop platform-level software for the Adaptive Platform. The artifact workflow is depicted in Figure 2.16.

2.3.2.2 Description

[TR_AMETH_00035] Platform-level Software ⌈ An Adaptive Application of category platform-level is a collection of executables. The executable may consist of software components if these are based on standardized service interfaces, but may also be directly implemented without a software component model. ⌋*(RS_METH_00207, RS_METH_00041)*

[TR_AMETH_00020] Development of `Platform Object Code` ⌈ The platform modules, which consist of an executable, need to be developed. Similar as application-level software, they are later instantiated in terms of an Application Manifest and then deployed on the machine. For each executable the corresponding main function needs to be developed as well. ⌋*(RS_METH_00207, RS_METH_00041)*

2.3.2.3 Workflow

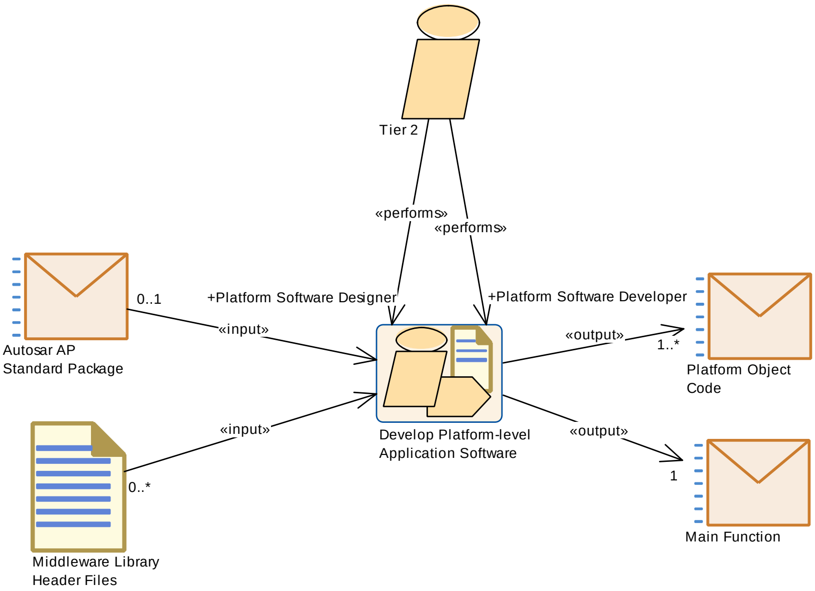
Tier 2
«performs»
«performs»
+Platform Software Designer
+Platform Software Developer
0..1
«input»
Autosar AP Standard Package
«input»
0..*
Middleware Library Header Files
Develop Platform-level Application Software
«output»
1..*
Platform Object Code
«output»
1
Main Function

Figure 2.16: Develop Platform-level Application Software

Activity	**Develop Platform-level Application Software**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Develop Adaptive Application		
Brief Description	Develop an Adaptive Application with category platform-level		
Description	Develop an Adaptive Application with category platform-level. These applications are platform modules, which consist of an executable and are deployed together with an Application Manifest onto the machine (in contrast to e.g. the OS). This activity also includes the implementation of the corresponding main function.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Autosar AP Standard Package	0..1	In case standardized service interfaces are used for platform-level applications

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Middleware Library Header Files	0..*	Library header files needed for compiling the platform-level applications
Produces	Main Function	1	Main function for platform-level executable
Produces	Platform Object Code	1..*	Object code of platform module
Performed by	Tier 2	1	Platform Software Designer: The design tasks within the development of Platform-level Application Software will probably be performed by a Platform Software Designer of a Tier 2 company
Performed by	Tier 2	1	Platform Software Developer: The real development tasks (i.e., to write source code and the like) within the development of Platform-level Application Software will probably be performed by a Platform Software Developer of a Tier 2 company

Table 2.14: Develop Platform-level Application Software

2.4 Integration and Deployment

2.4.1 Integrate Software

2.4.1.1 Purpose

After the implementation and compilation of the software, it needs to be integrated into one executable. Since the executable also contains platform-specific aspects, this process step also describes other activities as e.g. the development of the serialization for a specific platform and the implementation of the proxies and skeletons.

2.4.1.2 Description

[TR_AMETH_00016] Development of serialization properties ⌈ It needs to be described how the data in the service interfaces shall be serialized for the transport on the network. In particular, this is important for the communication over SOME/IP between Classic and Adaptive Platform.
For the service interfaces, the properties of the serialization will be defined. For SOME/IP, this includes the alignment, the configuration of length fields that are added in front of arrays or structures, etc. Based on this Serialization Configuration, the serialization code can be generated. The serialization is developed for a dedicated Adaptive Platform. ⌋*(RS_METH_00006, RS_METH_00077, RS_METH_00066)*

[TR_AMETH_00017] Implementation of service proxies and skeletons ⌈ The service proxies and skeletons, which are contained in the Header Files for Service

Interfaces and used within the software components, need to be implemented. For this implementation, the serialization of data needs to be known. ⌋*(RS_METH_00207)*

[TR_AMETH_00018] Building the Executable Application ⌈The Executable Application can be built based on application-level Software Component Object Code or platform-level Platform Object Code together with the respective Main Function. Additionally, the Serialization Source Code and all necessary libraries and implementations are linked to one Executable Application. ⌋*(RS_METH_00202, RS_METH_00066, RS_METH_00042)*

2.4.1.3 Workflow

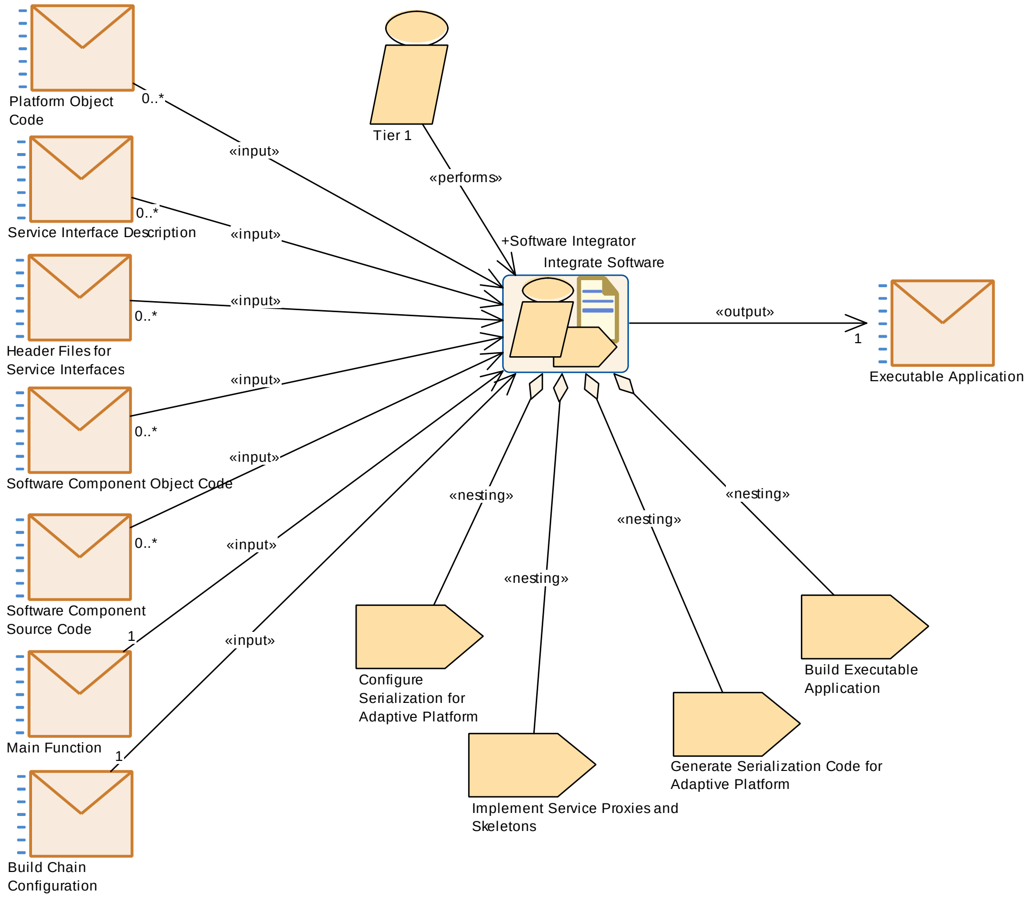
Platform Object Code
0..*
«input»
Service Interface Description
0..*
«input»
Header Files for Service Interfaces
0..*
«input»
Software Component Object Code
0..*
«input»
Software Component Source Code
0..*
«input»
Main Function
1
«input»
Build Chain Configuration
1
«input»
Tier 1
«performs»
+Software Integrator
Integrate Software
«output»
1
Executable Application
«nesting»
«nesting»
«nesting»
«nesting»
«nesting»
Configure Serialization for Adaptive Platform
Implement Service Proxies and Skeletons
Generate Serialization Code for Adaptive Platform
Build Executable Application

Figure 2.17: Integrate the software components

Activity	**Integrate Software**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Integration::Integrate Software		
Brief Description	Integrate software to one executable		
Description	In this activity, the compiled software and one main function are integrated into one executable. For this step, several other artifacts may be necessary, as the serialization code, the implemented proxies and skeletons and necessary middleware libraries. Several executables can later be packaged into an Adaptive AUTOSAR Application.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Build Chain Configuration	1	Needed for linking all artifacts
Consumes	Header Files for Service Interfaces	0..*	Proxies and skeletons to be implemented
Consumes	Main Function	1	One main function per executable
Consumes	Platform Object Code	0..*	Object code for platform-level executable
Consumes	Service Interface Description	0..*	Needed for defining the serialization
Consumes	Software Component Object Code	0..*	Object code for application-level executable
Consumes	Software Component Source Code	0..*	Source code for application-level executable
Produces	Executable Application	1	Software is integrated into one executable application
Aggregates	Build Executable Application	1	
Aggregates	Configure Serialization for Adaptive Platform	1	
Aggregates	Generate Serialization Code for Adaptive Platform	1	
Aggregates	Implement Service Proxies and Skeletons	1	
Performed by	Tier 1	1	Software Integrator: This activity will probably be performed by a Software Integrator of a Tier 1 company

Table 2.15: Integrate Software

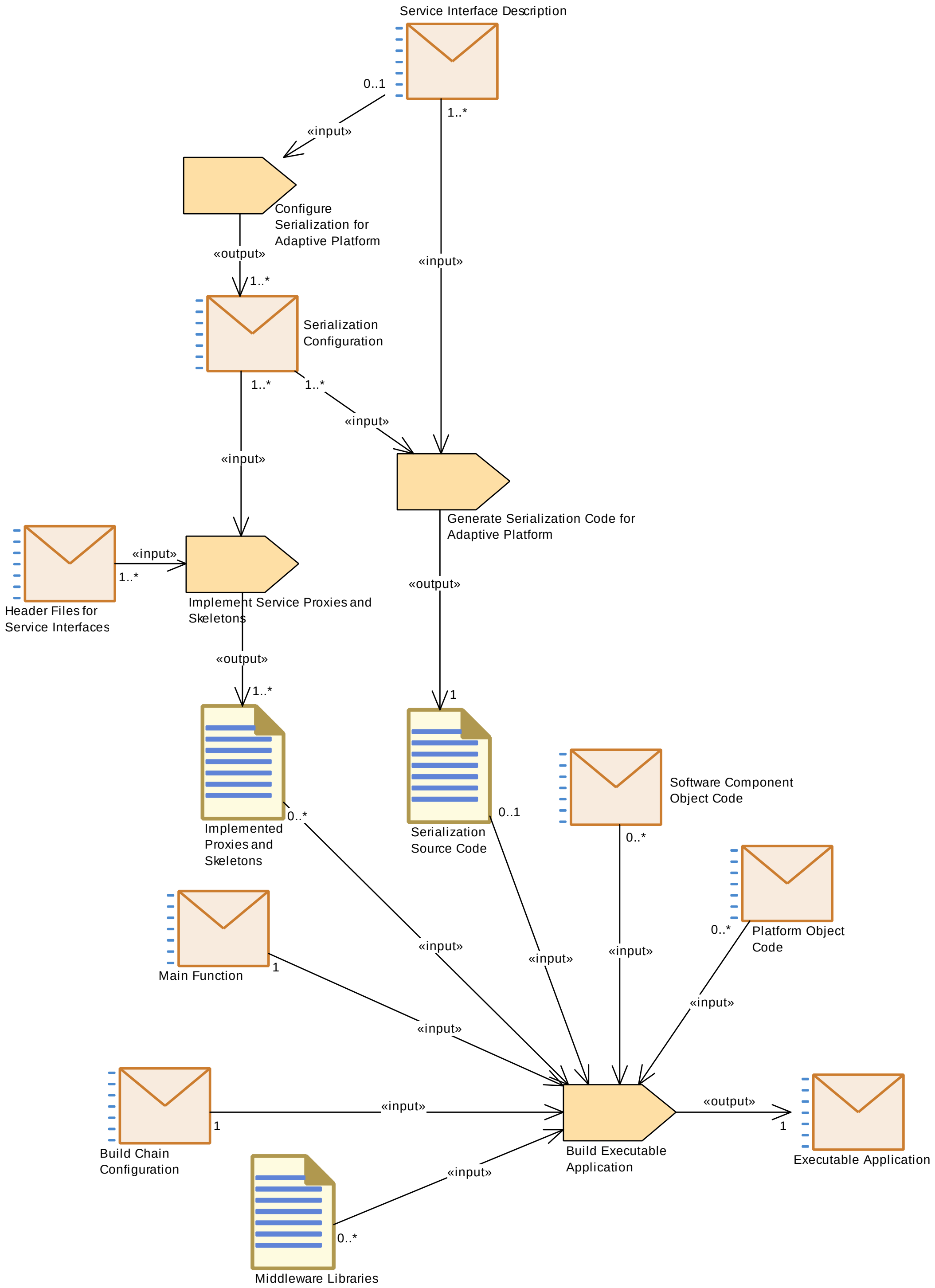
Service Interface Description
0..1
1..*
«input»
Configure Serialization for Adaptive Platform
«output»
1..*
«input»
Serialization Configuration
1..*
1..*
«input»
«input»
Generate Serialization Code for Adaptive Platform
Header Files for Service Interfaces
«input»
1..*
Implement Service Proxies and Skeletons
«output»
«output»
1..*
1
Implemented Proxies and Skeletons
0..*
Serialization Source Code
0..1
Software Component Object Code
0..*
Platform Object Code
0..*
Main Function
1
«input»
«input»
«input»
«input»
«input»
Build Chain Configuration
1
«input»
Build Executable Application
«output»
1
Executable Application
Middleware Libraries
0..*
«input»

Figure 2.18: Workflow for integrating the software

2.4.2 Define and configure a Machine

As outlined in [TR_AMETH_00003], the definition and configuration is subdivided into two process steps. This section here will deal with the second one, the activities and tasks necessary for the configuration of a real adaptive ECU in order to obtain a complete Machine Manifest.

2.4.2.1 Preparatory steps

2.4.2.1.1 Purpose

This subsection describes some preparatory activities towards the real configuration step of the machine.

2.4.2.1.2 Description

[TR_AMETH_00019] Description of the Adaptive Platform ⌈ As a first preparatory step, the available hardware elements of the particular Adaptive Platform need to be specified. This can be done by means of the ECU Resources Description which enables to describe all hardware elements, like processing units, memories, sensors, actuators or pins. ⌋*(RS_METH_00207, RS_METH_00041)*

ECU resources can be specified based on the ECU Resource Template [7].

[TR_AMETH_00034] Select the Operating System for the Adaptive Platform ⌈ Furthermore, an operating system (OS) needs to be selected for a particular Adaptive Platform and assembled. To that, it might be necessary to port or at least to adjust the OS for the specific hardware.

The OS for the Adaptive Platform is a platform module and will therefore not have an Application Manifest. Note, that its development workflow will differ from the workflow of platform-level applications (see Section 2.3.2). ⌋*(RS_METH_00207, RS_METH_00041)*

2.4.2.1.3 Workflow

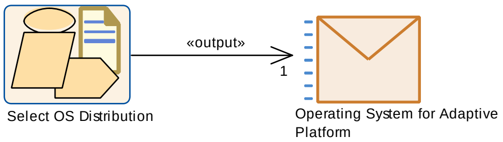

Figure 2.19: Select the OS Distribution

Activity	**Select OS Distribution**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Define and Configure Machine::Develop Platform Software		
Brief Description	Select and assemble an operating system		
Description	Select an operating system and assemble it. The workflow for the platform modules as the OS is different to the workflow of platform-level applications, which will be instantiated with an Application Manifest.		
Relation Type	***Related Element***	***Mul.***	***Note***
Produces	Operating System for Adaptive Platform	1	Selected OS distribution

Table 2.16: Select OS Distribution

2.4.2.2 Configure the Machine

2.4.2.2.1 Purpose

The machine describes the computing resource on which the Adaptive AUTOSAR Software Stack is executed.

Based on the assumptions of [TR_AMETH_00003], this use case describes all definition and configuration activities for the machine, independent of the deployment information of applications or service instances. All produced content will be part of the `Machine Manifest`.

The overview of inputs, outputs and all tasks is given in Figure 2.20. The workflow is described in Figure 2.21.

2.4.2.2.2 Description

[TR_AMETH_00022] Definition of machine states, function group states and per-state timeouts ⌈ The configuration of a machine includes the definition of machine states, function group states and per-state timeouts.

A machine can have several machine states, in which certain processes will be activated or deactivated. These states need to be defined and can then be used for the start-up configuration of a process, which might depend on the machine states.

Function groups with function group states individually control groups of functionally coherent application processes.

It is possible to define timeouts by means of `EnterExitTimeouts` for selected machine states (modes) or function group states. ⌋*(RS_METH_00204, RS_METH_00203)*

[TR_AMETH_00217] Definition of resources ⌈The configuration of a machine may include the definition of resources. Based on the ECU Resources Description (as an input), available hardware resources for a machine can be described . ⌋ *(RS_METH_00204, RS_METH_00203)*

[TR_AMETH_00216] Map Processes to a particular machine ⌈The configuration of the machine includes the mapping of Processes to a particular machine by means of the meta model element ProcessToMachineMapping, assuming that one Process shall only be mapped once, to exactly one machine.

To perform this, a list of Processes supposed to run on the machine is required as input. ⌋*(RS_METH_00204, RS_METH_00203)*

[TR_AMETH_00023] Configuration of the operating system ⌈The configuration of the operating system is defined via the AdaptiveModuleInstantiation meta class. For a specific instantiation of the operating system, resource groups as well as the supported timer granularity can be defined. ⌋*(RS_METH_00204, RS_METH_00203)*

[TR_AMETH_00214] Configuration of Platform Services ⌈The configuration of a machine includes the machine-specific configuration of Adaptive Platform Services, like the machine-specific configuration of

- the NM module
- DoIP

⌋*(RS_METH_00204, RS_METH_00203)*

[TR_AMETH_00215] Configuration of Platform Foundation Modules ⌈Beside the configuration of the Operating System, the configuration of a machine also includes the machine-specific configuration of the Adaptive Platform Foundation Modules, like the machine-specific configuration of

- the Log & Trace module

⌋*(RS_METH_00204, RS_METH_00203)*

2.4.2.2.3 Workflow

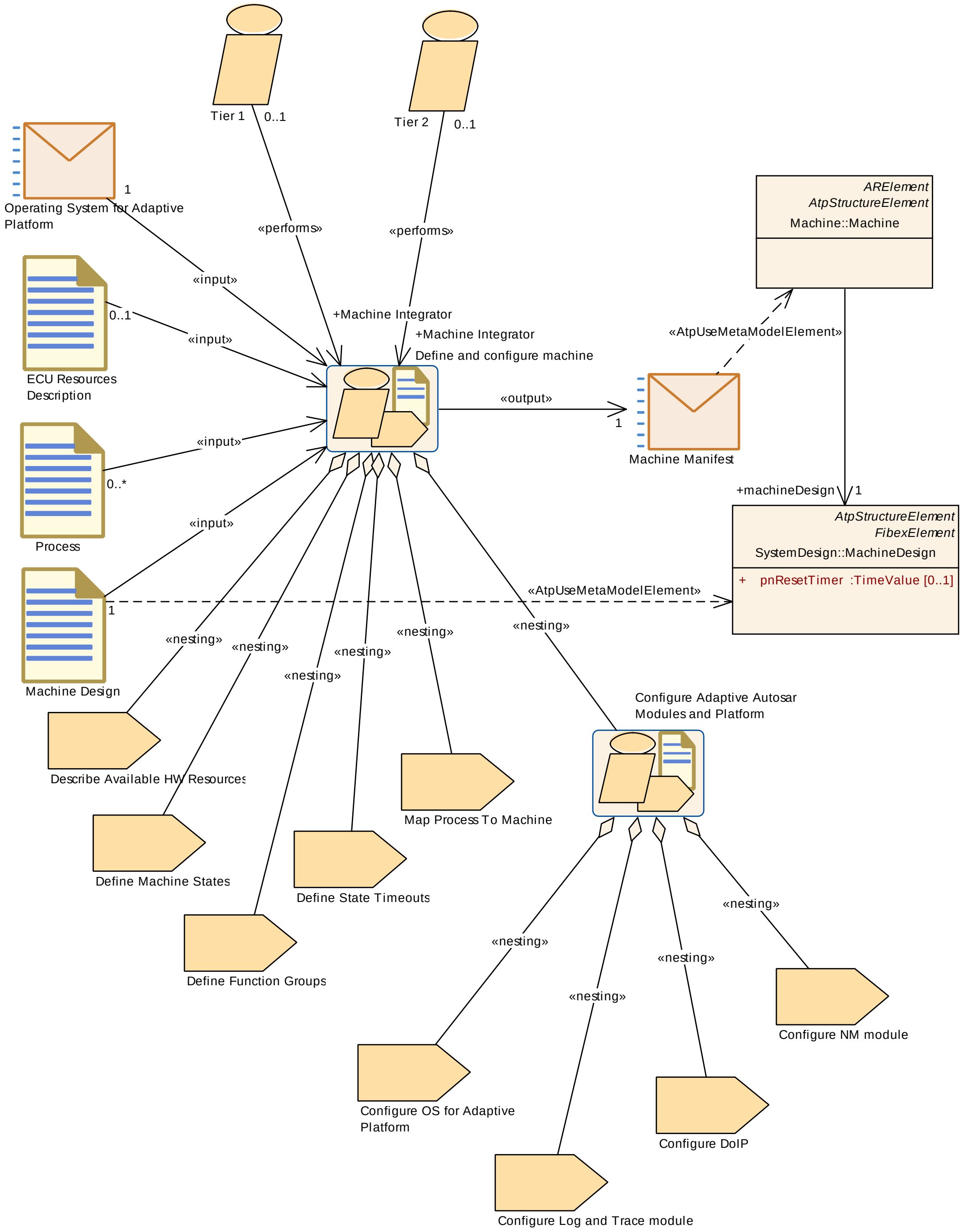

Figure 2.20: Define and Configure Machine

Activity	**Define and configure machine**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Define and Configure Machine::Machine Configuration		
Brief Description	Configuration of the machine independent of deployment information of applications or service instances		
Description	The activity describes tasks for the configuration of the machine, which do not depend on deployment information of applications or service instances. This includes the configuration for the communication on the network based on service discovery, the description of all machine states and the available resources as well as dedicated configuration of the OS.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	ECU Resources Description	0..1	All resources which are available for the ECU
Consumes	Machine Design	1	Configuration settings of the network connections and service discovery network exchange of a Machine
Consumes	Operating System for Adaptive Platform	1	OS to be configured
Consumes	Process	0..*	Processes dedicated to run Executables on a Machine
Produces	Machine Manifest	1	The machine manifest describes all the configuration settings for one Machine
Aggregates	Configure Adaptive Autosar Modules and Platform	1	
Aggregates	Define Function Groups	1	
Aggregates	Define Machine States	1	
Aggregates	Define State Timeouts	1	
Aggregates	Describe Available HW Resources	1	
Aggregates	Map Process To Machine	1	
Performed by	Tier 1	0..1	Machine Integrator: This activity will probably be performed by a Machine Integrator of a Tier 1 company
Performed by	Tier 2	0..1	Machine Integrator: Alternatively, this activity could also be performed by a Machine Integrator of a Tier 2 company

Table 2.17: Define and configure machine

Activity	**Configure Adaptive Autosar Modules and Platform**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Define and Configure Machine::Machine Configuration		
Brief Description			
Description	Configure individual Adaptive Autosar modules, i.e., the OS as well as non-OS modules.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Configure DoIP	1	
Aggregates	Configure Log and Trace module	1	
Aggregates	Configure NM module	1	
Aggregates	Configure OS for Adaptive Platform	1	

Table 2.18: Configure Adaptive Autosar Modules and Platform

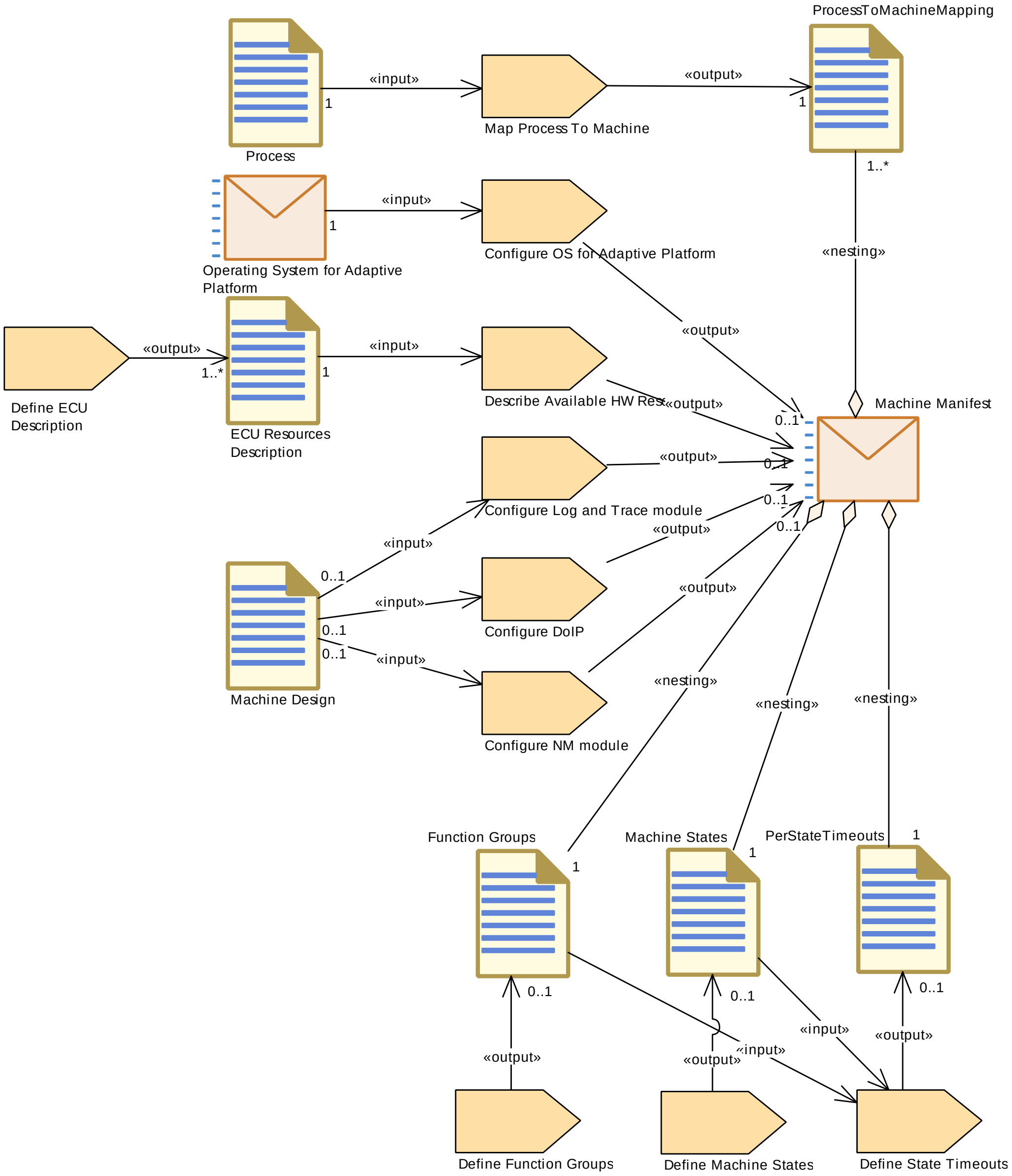

Figure 2.21: Workflow for defining and configuring an machine

2.4.3 Create Application Manifest

2.4.3.1 Purpose

This use case defines all tasks, which are necessary in order to instantiate the Executable Application. For on overview see Figure 2.22. The workflow is given in Figure 2.23.

2.4.3.2 Description

[TR_AMETH_00024] Instantiation of Executable Application ⌈ Define the instantiation of an Executable Application on a specific machine in terms of a process. One executable can be instantiated several times and in different ways, e.g. varying in the definition of the startup behavior. This results in several processes. ⌋ *(RS_METH_00203, RS_METH_00077)*

[TR_AMETH_00025] Definition of startup behavior of a process ⌈ For each process the startup behavior can be defined depending on a machine state. Therefore, the process might have a different startup behavior in one machine state compared to a second machine state. This behavior can e.g. vary in terms of the scheduling priority or the execution dependencies to other processes. ⌋*(RS_METH_00203)*

[TR_AMETH_00026] Definition of Application Manifest ⌈ The Application Manifest aggregates the process and its startup configuration. Therefore, one Application Manifest is defined per process. ⌋*(RS_METH_00203)*

2.4.3.3 Workflow

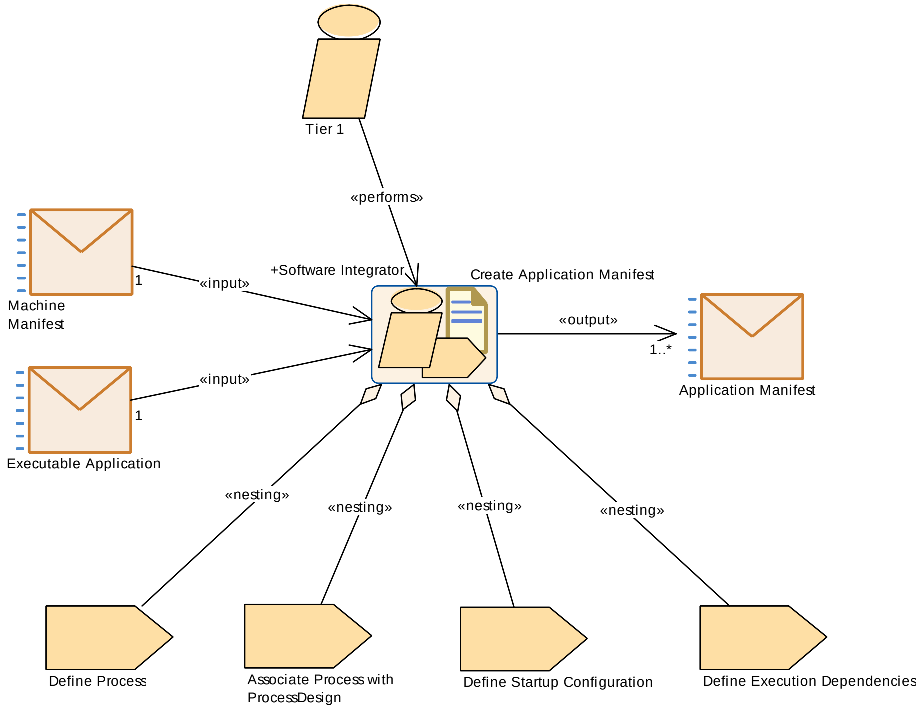

Figure 2.22: Create an Application Manifest

Activity	**Create Application Manifest**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Application Manifest		
Brief Description	Instantiation-specific configuration of executable		
Description	In this activity, the processes are defined. One executable can be instantiated several times, which results in multiple processes for one executable. One Application Manifest is defined per process and contains all its attributes including startup configuration and execution dependencies.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Executable Application	1	One executable can be instantiated several times
Consumes	Machine Manifest	1	Instantiation is defined on one specific machine
Produces	Application Manifest	1..*	One application manifest per instantiated executable
Aggregates	Associate Process with Process Design	1	
Aggregates	Define Execution Dependencies	1	
Aggregates	Define Process	1	
Aggregates	Define Startup Configuration	1	
Performed by	Tier 1	1	Software Integrator: This activity will probably be performed by a Software Integrator of a Tier 1 company

Table 2.19: Create Application Manifest

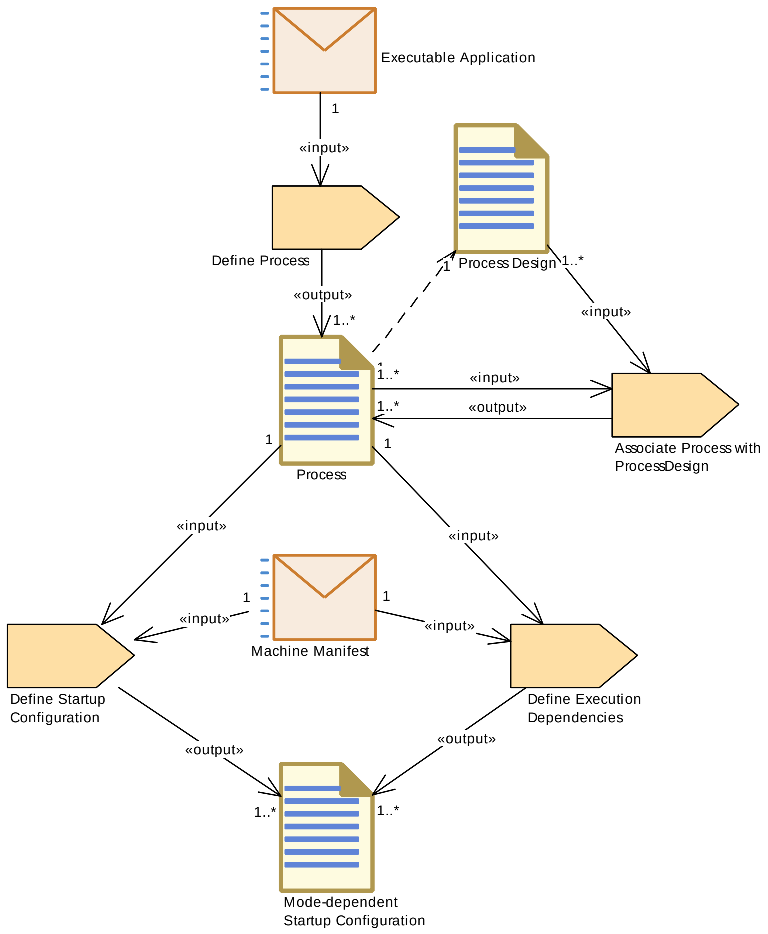

Figure 2.23: Workflow for defining a Process

2.4.4 Define and Configure Service Instances

Disclaimer: the content of this section is under discussion. Changes can be expected for the next release.

2.4.4.1 Purpose

This use case describes the definition and configuration of service instances in the system. For an overview of all tasks see Figure 2.24. For the workflow see Figure 2.25. The outcome of this activity is the Service Instance Manifest.

2.4.4.2 Description

[TR_AMETH_00027] Configuration of Service Interface Deployment ⌈ The system responsible needs to define how the service interfaces shall be deployed. In particular,

for each used transport layer, the binding of the service interface to this transport layer needs to be given.
For SOME/IP deployment, an ID for each service interface is defined. This ID needs to be unique in the system. Additionally, methodId, eventId as well as event groups are defined. ⌋*(RS_METH_00206, RS_METH_00203)*

[TR_AMETH_00028] Configuration of Service Instances ⌈ Afterwards, the system responsible defines instances of the deployed service interfaces and decides whether the service instance is provided or consumed. In order to set up the service-oriented communication, the search or offer criteria for all service instances are described. ⌋*(RS_METH_00206, RS_METH_00203)*

[TR_AMETH_00029] Mapping of Service Instances to Machine ⌈ The service instances will be deployed to the Adaptive Platform instance that will execute the service instance via the ServiceInstanceToMachineMapping. For SOME/IP, the TP and IP configuration for the client and the server are described. ⌋*(RS_METH_00206, RS_METH_00203, RS_METH_00078)*

[TR_AMETH_00033] Mapping of Service Instances to Port Prototypes ⌈ In addition, the service instances need to be mapped to their representation in the application (i.e., port prototypes) via the ServiceInstanceToPortPrototypeMapping. This mapping is necessary in order to ensure a unique relationship between locally used service instances within the application and global service instances on the network (e.g. SOME/IP service instances). ⌋*(RS_METH_00206, RS_METH_00203, RS_METH_00078)*

2.4.4.3 Workflow

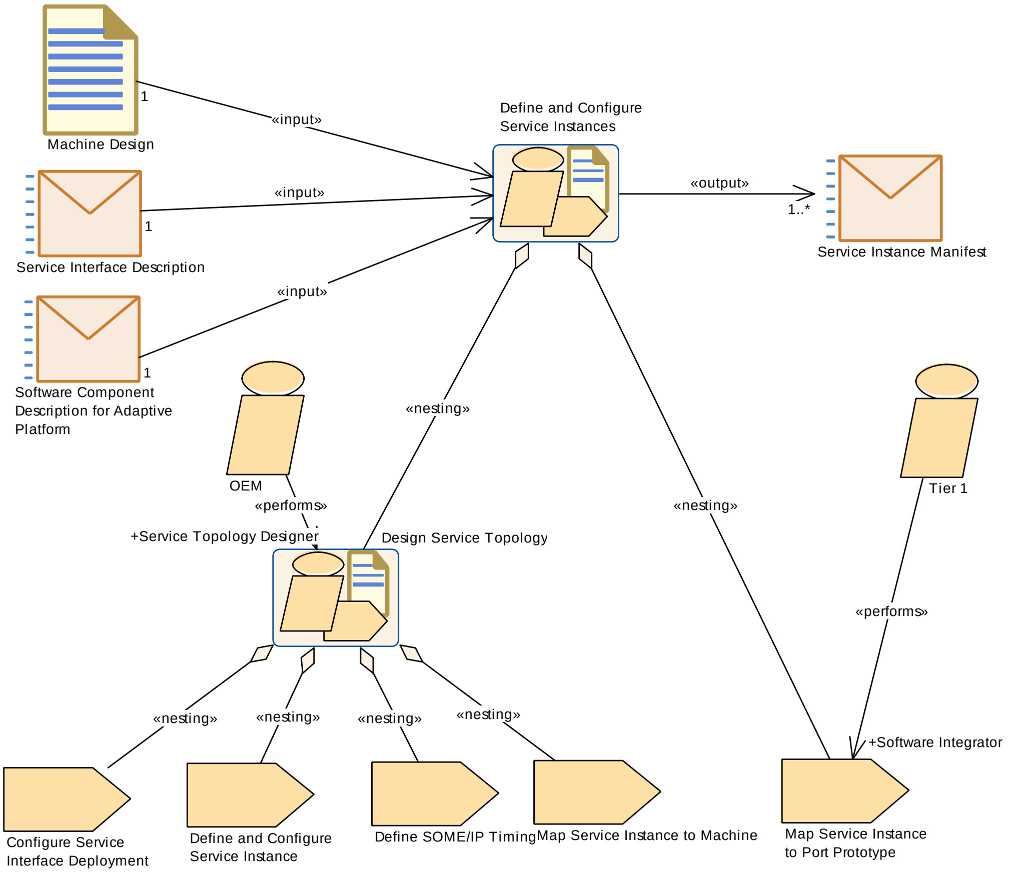
Machine Design
1
«input»
Define and Configure Service Instances
«input»
Service Interface Description
1
«input»
Software Component Description for Adaptive Platform
1
«output»
1..*
Service Instance Manifest
OEM
«performs»
+Service Topology Designer
«nesting»
Design Service Topology
«nesting»
Tier 1
«performs»
+Software Integrator
«nesting»
«nesting»
«nesting»
«nesting»
Configure Service Interface Deployment
Define and Configure Service Instance
Define SOME/IP Timing
Map Service Instance to Machine
Map Service Instance to Port Prototype

Figure 2.24: Define and Configure Service Instances

Activity	**Define and Configure Service Instances**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Service Instance Definition		
Brief Description	Configuration of service interface deployment and service instances		
Description	This activity covers the configuration of the service interfaces for the used network layer, independent of any instantiation on the one hand as well as the definition and configuration of service instances on the other.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Machine Design	1	Service instances will be mapped to machine
Consumes	Service Interface Description	1	Deployment of service interfaces needs to be configured
Consumes	Software Component Description for Adaptive Platform	1	Used to map the service instances to ports of a software component
Produces	Service Instance Manifest	1..*	Contains all configuration settings for the service instance on a specific machine
Aggregates	Design Service Topology	1	

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Map Service Instance to Port Prototype	1	

Table 2.20: Define and Configure Service Instances

Activity	**Design Service Topology**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Service Instance Definition		
Brief Description	Design Service Topology		
Description	This activity subsumed all design tasks which are related to the design of a network topology		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Configure Service Interface Deployment	1	
Aggregates	Define SOME/IP Timing	1	
Aggregates	Define and Configure Service Instance	1	
Aggregates	Map Service Instance to Machine	1	
Performed by	OEM	1	Service Topology Designer: This activity will probably be performed by a Service Topology Designer of an OEM

Table 2.21: Design Service Topology

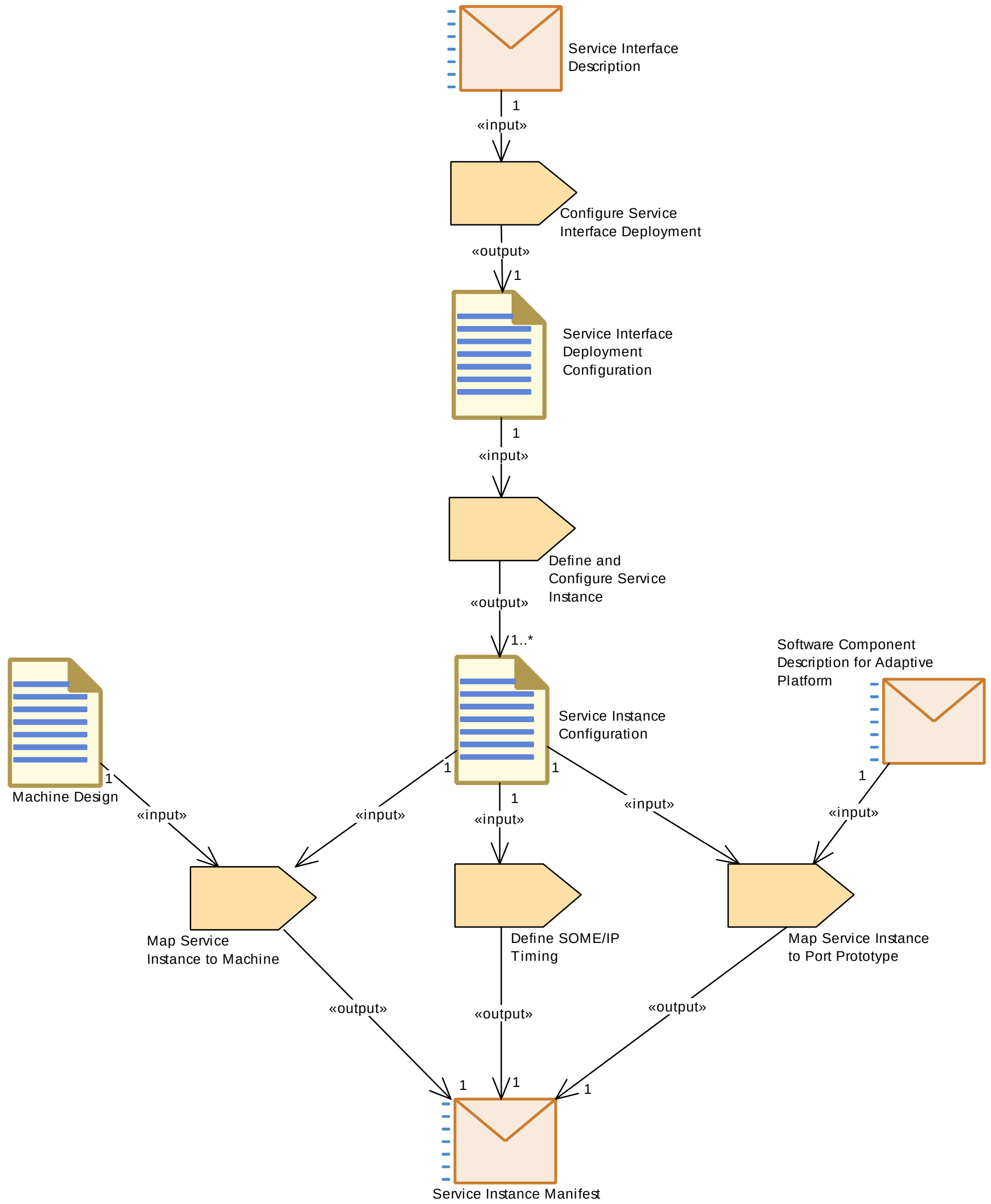
Service Interface Description
1
«input»
Configure Service Interface Deployment
«output»
1
Service Interface Deployment Configuration
1
«input»
Define and Configure Service Instance
«output»
1..*
Service Instance Configuration
Machine Design
Software Component Description for Adaptive Platform
1
«input»
1
«input»
1
1
«input»
1
«input»
1
«input»
Map Service Instance to Machine
Define SOME/IP Timing
Map Service Instance to Port Prototype
«output»
«output»
«output»
1
1
1
Service Instance Manifest

Figure 2.25: Workflow for defining and configuring service instances

2.4.5 Set up an initial Machine

Disclaimer: the content of this section is under discussion.

2.4.5.1 Purpose

This activity describes how a machine is set up so that software can be deployed onto it.

2.4.5.2 Description

[TR_AMETH_00031] Setting up an initial machine ⌈ The aim of this activity is to obtain a machine that is initially set up. 'Initially set up' means here, that the machine is able to upload and install additional software by means of Software Packages. For this purpose at least the Platform module UCM and dependent modules (like the diagnostic communication module) need to run on the initially set up machine. Thus, this activity will (at least) include the following tasks:

1. Install the selected Operating System on the selected target (machine).
2. Install all necessary Platform modules on top of the installed OS in order to be able to perform the upload and the installation of additional application software by means of Software Packages.

In order to be able to execute this activity, the following inputs are necessary:

- A selected Operating System for Adaptive Platform
- The configuration settings by means of the Machine Manifest
- Possibly, design artifacts like the Machine Design
- The Executables of the Platform and Application modules which shall be installed
- Application Manifests and Service Instance Manifests of the Platform and Application modules which shall be installed
- Possibly, diagnostic information by means of the Diagnostic Machine Extract since the upload and installation process may use the diagnostic environment

⌋*(RS_METH_00205, RS_METH_00204)*

Figure 2.26 shows the aforementioned; illustrating the relations of the involved entities.

2.4.5.3 Workflow

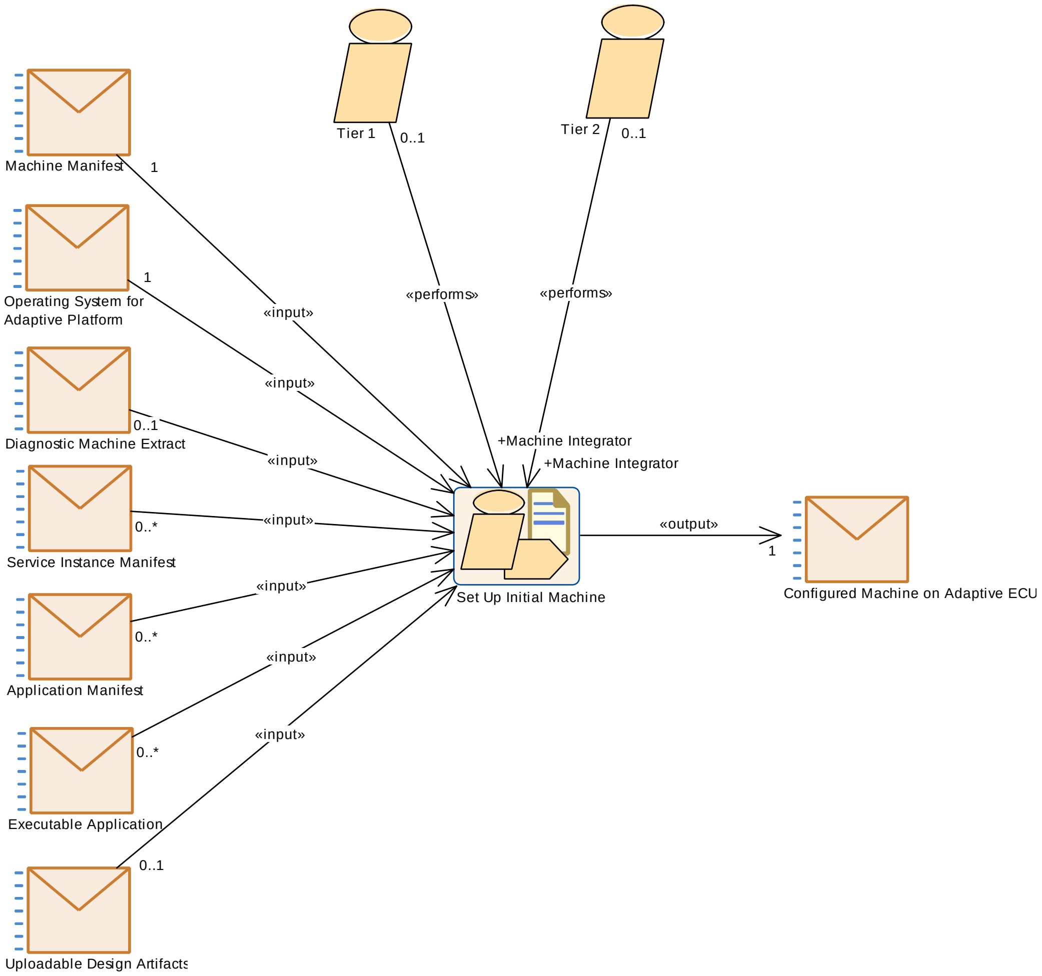

Figure 2.26: Set up initial machine

Activity	**Set Up Initial Machine**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Define and Configure Machine::Setup Machine		
Brief Description	Set up the machine based on the machine manifest		
Description	Configure and install the OS and other necessary platform modules (e.g., UCM) on the machine. The configuration settings are given by the Machine Manifest. In addition, the network connections as well as machine states are set up.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Application Manifest	0..*	All Application Manifests needed to run the desired adaptive application (instances or Processes) on a Machine
Consumes	Diagnostic Machine Extract	0..1	Diagnostic extract for a Machine

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Executable Application	0..*	Executables of those Platform modules and Adaptive Applications that should run on a initially configured machine. Beside the OS, at least the UCM and connected Platform modules (e.g., a diagnostic communication manager) need to be installed in order to be able to upload other software.
Consumes	Machine Manifest	1	Containing all configuration settings for the Machine
Consumes	Operating System for Adaptive Platform	1	OS to be installed on machine
Consumes	Service Instance Manifest	0..*	All Service Instance Manifests needed to run the desired adaptive application (instances or Processes) on a Machine
Consumes	Uploadable Design Artifacts	0..1	Optional input: Additional design data which are not part of an Application or Machine Manifest
Produces	Configured Machine on Adaptive ECU	1	Machine is configured and software can now be deployed
Performed by	Tier 1	0..1	Machine Integrator: This activity will probably be performed by a Machine Integrator of a Tier 1 company
Performed by	Tier 2	0..1	Machine Integrator: Alternatively, this activity could also be performed by a Machine Integrator of a Tier 2 company

Table 2.22: Set Up Initial Machine

2.4.6 Create Software Packages

Disclaimer: the content of this section is under discussion.

2.4.6.1 Purpose

This use case comprises all activities and tasks to specify Software Packages.

2.4.6.2 Description

The AUTOSAR Adaptive Platform offers the ability to upload software onto machines (AUTOSAR Adaptive Platform instances) without to reflash everything.

According to the glossary [8], Software Packages are the units for deployment onto machines (AUTOSAR Adaptive Platform instances). In this respect, they are inputs for and processed by the Adaptive Platform Service UCM.

In fact, a Software Package consists of two main parts:

- a bundle of the actual software artifacts, referred to as Software Cluster here
- corresponding model data needed to control the upload and installation process of this Software Cluster executed by the UCM [9], referred to as Software Package Manifest here

Thus, from an UCM point of view, the term Software Cluster identifies a bundle of software artifacts that are uploaded together in order to be installed by the UCM. In general, a Software Cluster may contain Executables, Application Manifests, Service Instance Manifests, Machine Manifests and other development artifacts. It should be mentioned, that a Software Cluster may be structured into sub-blocks in order to mimic the CP diagnostic workflow, where blocks are the smallest parts of update and to enable the execution of update campaigns (see details in [9]).

Otherwise, the term Software Cluster may also refer to a set of installed software entities (processes that run Executables, data or manifests) which form a logical group and which are addressable by the diagnostic management by a shared diagnostic address.

Not surprisingly, both definitions match in the sense that the bundle of software uploaded are needed to form the set of installed software entities addressed by the same diagnostic address.

A Software Cluster (in the UCM sense) is described by its model, collected in the Software Package Manifest. The root-element of this description is called SoftwareCluster (category ROOT_SOFTWARE_CLUSTER) [6]. From a model point of view, the sub-blocks, mentioned above, can be expressed likewise by the same meta model element SoftwareCluster, but in the role subSoftwareCluster (or category SUB_SOFTWARE_CLUSTER) [6].

The meta model supports also the expression of dependencies between SoftwareClusters or subSoftwareClusters [6], the assignment of a diagnostic address for SoftwareCluster of category ROOT_SOFTWARE_CLUSTER and, of course, information about which artifact belongs to which SoftwareCluster. See [6] for a deeper insight into the respective modeling.

In general, it might be useful for integrator to store incoming artifacts as well as assembled Software Clusters into repository and manage them by some sort of data base.

Note, that the real format of the Software Package is implementation specific and not covered by any specification [9].

[TR_AMETH_00206] Create a `Software Package` ⌈ The following activities/tasks are needed in order to obtain a `Software Package`:

- Create an initial `Software Package Manifest`
- Collect all software artifacts that belong to a `Software Cluster`, structure and model them
- Model dependencies between `Software Cluster` of any category
- Develop installation instructions
- Create the `Software Package`
- Manage the data base of `Software Cluster`s (of any category)

Figure 2.27 shows the corresponding input and output deliverables. ⌋ *(RS_METH_00205)*

One input of this activity is the deliverable `Software Cluster Design` based on the meta model element `SoftwareClusterDesign` [6]. The deliverable `Software Cluster Design` contains the requirements that have initially been formulated by an OEM. The formal structure of the `SoftwareClusterDesign` is similar to `SoftwareCluster` [6]. Thus, by means of this, the OEM is able to define the composition and structure of `Software Cluster`s, dedicated diagnostic addresses as well as internal and external dependencies of `Software Cluster`s.

The clear separation of the meta model elements `SoftwareCluster` and `SoftwareClusterDesign` is motivated from a methodology point of view, because different parties are involved at different design stages. To specify requirements for the structure of `Software Packages` is the genuine interest of an OEM, because he knows best about its IT- and vehicle infrastructure, whereas (most probably) a `Tier 1` company is responsible for the integration and deployment processes.

[TR_AMETH_00218] Create an initial `Software Package Manifest` ⌈ The main input for this step are the requirements of the OEM given by means of `Software Cluster Design`. Thus, this task is about to create an new `Software Package Manifest` and to transfer the structure and the entries of the given `Software Cluster Design` into the newly created `Software Package Manifest`. ⌋*()*

[TR_AMETH_00219] Collect all software artifacts that belong to a `Software Cluster`, structure and model them ⌈ On base of the `Software Cluster Design` o the newly created `Software Package Manifest`, this step includes the following sub-tasks:

- Identify necessary (software) artifacts
  - Identify necessary (software) artifacts in order to build the `Software Package`, also with respect to their versions
  - Check, whether there are deviations between the required and actual sets of *Sub Software Cluster*s (by means of the aggregated artifacts and versions) ,

if necessary solve them and re-model the Software Package Manifest accordingly

  - Check, whether there are discrepancies between the required and actual set of the (root) Software Cluster (by means of its aggregated *Sub Software Clusters* and versions)

- Collect belonging (software) artifacts of *Sub Software Cluster*s
  - Collect belonging (software) artifacts of *Sub Software Cluster*s into separate baskets ((Sub) Software Cluster Groups) in order to prepare the final step of creating the Software Package
  - Execute a receiving inspection (optional)
  - Store incoming artifacts into a repository

⌋*()*

[TR_AMETH_00220] Model dependencies between Software Clusters of any category ⌈ Dependencies between Software Clusters of the same or different categories may already be given by the requirements of an OEM by means of a SoftwareClusterDesign. Dependencies to Software Clusters are specified by means of their identification (name) and version.

Therefore, the respective SoftwareClusterDesign is will be one input for this activity.

However, dependencies may change during the development process and the activity needs to consider it.

Thus, this task describes the handling of dependencies by at least the following subtasks:

- Check, whether the dependencies between Software Clusters of the same or different categories, given by the respective SoftwareClusterDesign are still valid
- Determine changes between the actual and required dependencies between Software Clusters of any category
- If necessary, re-model the Software Package Manifest in accordance with the outcomes of the both tasks above

⌋*()*

[TR_AMETH_00221] Develop installation instructions ⌈ Installation instruction control the behavior of the UCM during the update of Software Packages. Installation instructions can either be 'add/update' meaning to install a package or 'remove' to express that a package shall be uninstalled and deleted from the machine. Installation instructions are defined per Software Cluster, independent of its category. For details, see [9].

Thus, this task may includes the sub-tasks:

- Specify installation instructions per Software Cluster (of any category)
- Develop update campaigns (optional)

The particular installation instructions are part of the Software Package Manifest.

⌋()

[TR_AMETH_00222] Create the Software Package ⌈ The format of the Software Package as well as the update strategy, i.e., whether you go for a complete or a delta update are implementation specific. Both issues will not be specified by AUTOSAR.

Thus, this activity handles the compilation of Software Cluster and Software Package Manifest into a Software Package.

Since AUTOSAR does not specify how the Software Package looks like, the breakdown of this activity into tasks is also specific to particular OEMs and their suppliers.

⌋()

[TR_AMETH_00223] Manage the data base of Software Clusters (of any category) ⌈ A general activity may be the management of the data base of Software Clusters with respect to all their versions, dependencies and further aspects.

It is assumed that this activity is also specific to particular OEMs/suppliers. Therefore a more fine-granular task structure will not be specified here. ⌋()

2.4.6.3 Workflow

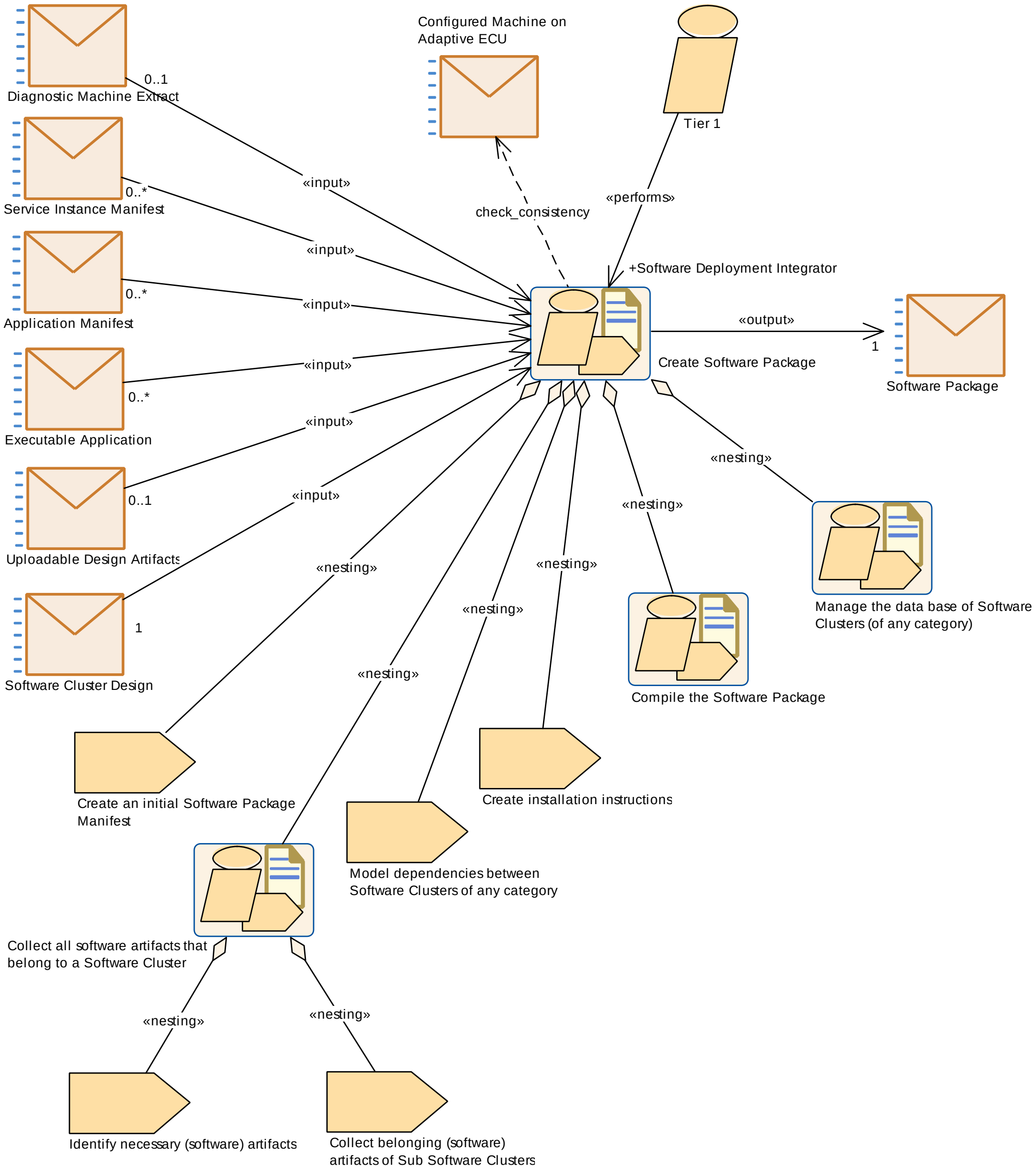
Diagnostic Machine Extract
0..1
Service Instance Manifest
0..*
Application Manifest
0..*
Executable Application
0..*
Uploadable Design Artifacts
0..1
Software Cluster Design
1
«input»
«input»
«input»
«input»
«input»
«input»
Configured Machine on Adaptive ECU
check_consistency
Tier 1
«performs»
+Software Deployment Integrator
«output»
1
Create Software Package
Software Package
«nesting»
«nesting»
«nesting»
«nesting»
«nesting»
«nesting»
Manage the data base of Software Clusters (of any category)
Compile the Software Package
Create installation instructions
Create an initial Software Package Manifest
Model dependencies between Software Clusters of any category
Collect all software artifacts that belong to a Software Cluster
«nesting»
«nesting»
Identify necessary (software) artifacts
Collect belonging (software) artifacts of Sub Software Clusters

Figure 2.27: Create a Software Package

Activity	**Create Software Package**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Packaging and Provision		
Brief Description	Create a Software Package		
Description	This activity describes the creation of a Software Package.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Application Manifest	0..*	Several processes can be deployed
Consumes	Diagnostic Machine Extract	0..1	Diagnostic extract for a Machine

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Executable Application	0..*	Executables of deployed processes
Consumes	Service Instance Manifest	0..*	Several service instance manifests can be deployed
Consumes	Software Cluster Design	1	Requirements of the OEM wrt. package structure and parameters given by means of the meta model element SoftwareClusterDesign.
Consumes	Uploadable Design Artifacts	0..1	Optional input: Additional design data which are not part of an Application or Machine Manifest
Produces	Software Package	1	Software Package for deployment defined
Aggregates	Collect all software artifacts that belong to a Software Cluster	1	
Aggregates	Compile the Software Package	1	
Aggregates	Create an initial Software Package Manifest	1	
Aggregates	Create installation instructions	1	
Aggregates	Manage the data base of Software Clusters (of any category)	1	
Aggregates	Model dependencies between Software Clusters of any category	1	
Performed by	Tier 1	1	Software Deployment Integrator: This activity will probably be performed by a Software Deployment Integrator of a Tier 1 company

Table 2.23: Create Software Package

Activity	**Collect all software artifacts that belong to a Software Cluster**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Packaging and Provision		
Brief Description	Collect all software artifacts		
Description	On base of the Software Cluster Design o the newly created Software Package Manifest, this step includes the following tasks: • Identify and gather all needed (software) artifacts in order to build the Software Package, also with respect to their versions • Execute a receiving inspection (optional) • Store incoming artifacts into a repository • Assemble belonging (software) artifacts for Sub Software Clusters into separate 'baskets' (Software Cluster Groups) in order to prepare the final step of creating the Software Package • Check, whether there are divergences within the required and actual sets of Sub Software Clusters (by means of the aggregated artifacts and versions) . If necessary solve them and re-model the Software Package Manifest, accordingly • Check, whether there are discrepancies between the required and actual set of the Root Software Cluster (by means of its aggregated Sub Software Clusters and versions)		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Collect belonging (software) artifacts of Sub Software Clusters	1	
Aggregates	Identify necessary (software) artifacts	1	

Table 2.24: Collect all software artifacts that belong to a Software Cluster

Activity	**Compile the Software Package**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Packaging and Provision		
Brief Description	Compile the Software Package		
Description	The format of the Software Package as well as the update strategy, i.e., whether you go for a complete or a delta update are implementation specific. Both issues will not be specified by AUTOSAR. Thus, this activity copes with compilation of the belonging parts into a Software Package, without being able to specify how the Software Package looks like. Therefore, the structure of this activity by tasks is also specific to particular OEMs and their suppliers.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	(Sub) Software Cluster Group	0..*	Compile all Sub Software Clusters into the Software Package
Consumes	Software Package Manifest	1	Integrate the Software Package Manifest into the Software Package

Relation Type	*Related Element*	*Mul.*	*Note*
Produces	Software Package	1	Compiled Software Package

Table 2.25: Compile the Software Package

Activity	**Manage the data base of Software Clusters (of any category)**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Packaging and Provision		
Brief Description	Manage the data base of Software Clusters		
Description	A general activity may be the management of the data base of Software Clusters with respect to all their versions, dependencies and further aspects. It is assumed that this activity is also specific to particular OEMs/suppliers. Therefore a more fine-granular task structure will not be specified here.		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Software Cluster	1..*	Store and manage software cluster within a repository
Consumes	Software Package Manifest	1..*	Manage meta data of corresponding Software Cluster

Table 2.26: Manage the data base of Software Clusters (of any category)

2.4.7 Management and provision of Software Packages

Disclaimer: the content of this section is under discussion.

2.4.7.1 Purpose

This activity may comprise two aspects:

- The management of Software Packages ready to upload onto the machines
- The provision of Software Packages for the upload

2.4.7.2 Description

[TR_AMETH_00224] Management of Software Packages ⌈ Once Software Packages have been created, they are generally ready to be deployed to dedicated machines (Adaptive ECUs) in the field.

In order to do so, the Software Package may be stored, e.g., into a repository of packages located on a Back-end server.

The management of this repository of the Software Packages may be supported by means of data bases.

Since the management of Software Packages is an immanent task of an OEM and will differ between the companies, this activity will not be detailed further. ⌋()

[TR_AMETH_00225] Provision of Software Packages for machines in the field
⌈ A Back-end server may also provide some sort of (sophisticated) business logic. It may enable, e.g., a tester not only to access particular versions of particular Software Packages for upload, but also to provide change sets of different versions of Software Packages.

The handling of a concrete upload procedure is specified by diagnostic standards to some extend. However, as mentioned before, the format of the Software Package as well as the update strategy are not specified. There will be differences in handling and procedures among OEMs and therefore, this activity will not be further subdivided.
⌋()

2.4.7.3 Workflow

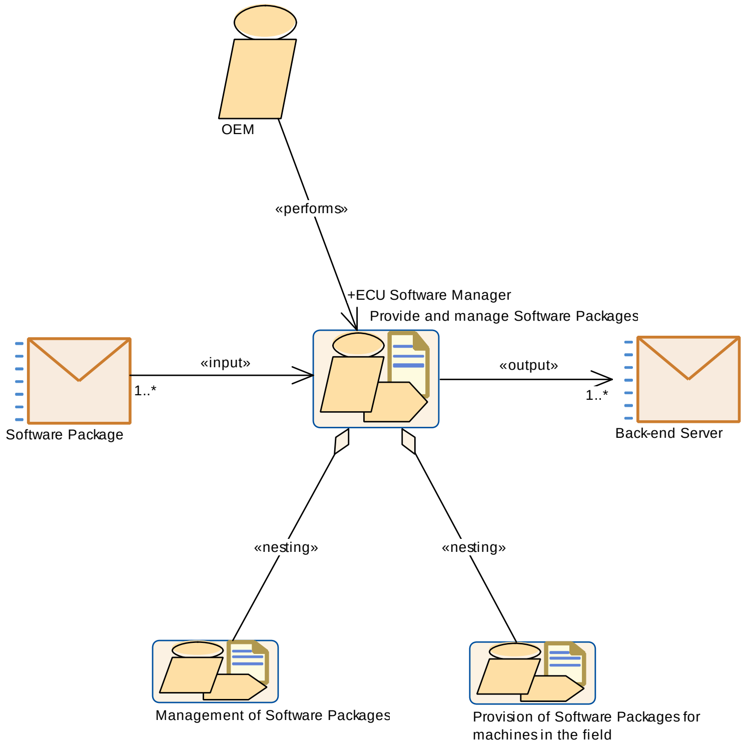
OEM
«performs»
+ECU Software Manager
Provide and manage Software Packages
«input»
1..*
Software Package
«output»
1..*
Back-end Server
«nesting»
«nesting»
Management of Software Packages
Provision of Software Packages for machines in the field

Figure 2.28: Provision of Software Packages

Activity	**Provide and manage Software Packages**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Packaging and Provision		
Brief Description	Provide and manage Software Packages		
Description	This activity may comprise two aspects: • The management of Software Packages ready to upload onto the machines • The provision of Software Packages for the upload		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Software Package	1..*	Deploy software on a Back-end server by means of Software Package
Produces	Back-end Server	1..*	Store uploadable packages (Software Packages) into a repository of a Back-end server
Aggregates	Management of Software Packages	1	
Aggregates	Provision of Software Packages for machines in the field	1	
Performed by	OEM	1	ECU Software Manager: This activity will be probably performed by an ECU Software Manager of an OEM

Table 2.27: Provide and manage Software Packages

Activity	**Management of Software Packages**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Packaging and Provision		
Brief Description	Management of Software Packages		
Description	Once Software Packages have been created, they are generally ready to be deployed to dedicated machines (Adaptive ECUs) in the field. In order to do so, the Software Package may be stored, e.g., into a repository of packages located on a Back-end server. The management of this repository of the Software Packages may be supported by means of data bases. Since the management of Software Packages is an immanent task of an OEM and will differ between the companies, this activity will not be detailed further.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Software Package	1..*	Newly created or updated Software Packages are stored into a repository and subject of the management of all available Software Packages (including their history)

Relation Type	***Related Element***	***Mul.***	***Note***
Produces	Back-end Server	1..*	Software Packages are stored into a repository of Software Packages. In addition, update of a common data base of available Software Packages including their history.

Table 2.28: Management of Software Packages

Activity	**Provision of Software Packages for machines in the field**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Packaging and Provision		
Brief Description	Provision of Software Packages		
Description	Present the Software Packages in a way, that the UCM of machines are able to access the respective Software Packages. A Back-end server may also provide some sort of (sophisticated) business logic. It may enable, e.g., a tester not only to access particular versions of particular Software Packages for upload, but also to provide change sets of different versions of Software Packages. The handling of a concrete upload procedure is specified by diagnostic standards to some extend. However, as mentioned before, the format of the Software Package as well as the update strategy are not specified. There will be differences in handling and procedures among OEMs and therefore, this activity will not be further subdivided.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Back-end Server	1	Status quo of the presentation layer of the Back-end Server
Produces	Back-end Server	1	Organize the Back-end Server in accordance with the requirements of an OEM

Table 2.29: Provision of Software Packages for machines in the field

3 Adaptive Methodology Library

The Adaptive Methodology Library lists all work products and tasks that are used for modeling the use cases in section 2.

3.1 Roles

3.1.1 OEM

Role	**OEM**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Common Elements::Roles		
Brief Description	OEM - Original Equipment Manufacturer		
Description	OEM - Original Equipment Manufacturer An OEM refers to a company that makes a final product for the consumer marketplace.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performs	Create Diagnostic Mapping	1	Diagnostic Designer: The activity of designing the diagnostic mapping will probably be performed by a Diagnostic Designer of an OEM
Performs	Design Service Topology	1	Service Topology Designer: This activity will probably be performed by a Service Topology Designer of an OEM
Performs	Design service oriented communication between Classic and Adaptive Platform	1	Service Interface Designer: This activity will probably be performed by a Service Interface Designer of an OEM
Performs	Design signal oriented communication between Classic and Adaptive Platform	1	Service Interface Designer: This activity will probably be performed by a Service Interface Designer of an OEM
Performs	Develop a Service Interface Description	1	Service Interface Designer: This activity will probably be performed by a Service Interface Designer
Performs	Develop the communication structure by means of MachineDesign	1	Machine Designer: This activity will probably be performed by a dedicated designer of an OEM.
Performs	Provide and manage Software Packages	1	ECU Software Manager: This activity will be probably performed by an ECU Software Manager of an OEM

Table 3.1: OEM

3.1.2 Tier 1

Role	**Tier 1**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Common Elements::Roles		
Brief Description	Direct (major) suppliers of parts to OEMs		
Description	Tier 1 companies are direct (major) suppliers of parts to OEMs.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performs	Create Application Manifest	1	Software Integrator: This activity will probably be performed by a Software Integrator of a Tier 1 company
Performs	Create Software Package	1	Software Deployment Integrator: This activity will probably be performed by a Software Deployment Integrator of a Tier 1 company
Performs	Integrate Software	1	Software Integrator: This activity will probably be performed by a Software Integrator of a Tier 1 company
Performs	Map Service Instance to Port Prototype	1	Software Integrator: This activity will probably be performed by a Software Integrator of a Tier 1 company
Performs	Define and configure machine	0..1	Machine Integrator: This activity will probably be performed by a Machine Integrator of a Tier 1 company
Performs	Set Up Initial Machine	0..1	Machine Integrator: This activity will probably be performed by a Machine Integrator of a Tier 1 company

Table 3.2: Tier 1

3.1.3 Tier 2

Role	**Tier 2**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Common Elements::Roles		
Brief Description	Key suppliers to tier 1 suppliers,		
Description	Tier 2 companies are key suppliers to tier 1 suppliers, without supplying a product directly to OEM companies.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performs	Design Software Component for Adaptive Platform	1	Application Software Designer: The design of software components will probably be performed by an Application Software Designer of a Tier 2 company
Performs	Develop Platform-level Application Software	1	Platform Software Designer: The design tasks within the development of Platform-level Application Software will probably be performed by a Platform Software Designer of a Tier 2 company

Relation Type	***Related Element***	***Mul.***	***Note***
Performs	Develop Platform-level Application Software	1	Platform Software Developer: The real development tasks (i.e., to write source code and the like) within the development of Platform-level Application Software will probably be performed by a Platform Software Developer of a Tier 2 company
Performs	Develop Software Components	1	Application Software Developer: This activity will probably be performed by an Application Software Developer of a Tier 2 company
Performs	Define and configure machine	0..1	Machine Integrator: Alternatively, this activity could also be performed by a Machine Integrator of a Tier 2 company
Performs	Set Up Initial Machine	0..1	Machine Integrator: Alternatively, this activity could also be performed by a Machine Integrator of a Tier 2 company

Table 3.3: Tier 2

3.2 Service Interface

This chapter contains the definition of work products and tasks used for the definition of service interfaces for the Adaptive Platform.

3.2.1 Tasks

3.2.1.1 Provide Data Types for Adaptive Platform

Task Definition	**Select or define Data Types for Adaptive Platform**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Interface Definition::Tasks		
Brief Description	Define a set of AP data types for a specific project, which are not already defined by Autosar.		
Description	Select or define a set of data types, which are required for the Adaptive Platform Instance, but which are not already defined by AUTOSAR. Standardized data types can be used as input in order to copy and refine them. Already existing data types can be reused. The AP Data Types are used for specifying DataElements in service interfaces. The focus is on the definition application data types and implementation data types and the necessary data type mapping sets.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Autosar AP Standard Package	0..1	Use standardized elements (e.g. data types, compu methods) to create the corresponding elements of the specific project.
Produces	AP Data Types	1..*	Defined AP Data Types for a specific project

Table 3.4: Select or define Data Types for Adaptive Platform

3.2.1.2 Define Service Interfaces

Task Definition	**Define Service Interfaces**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Interface Definition::Tasks		
Brief Description	Define the service interfaces that are used for the header file generation.		
Description	Define service interfaces by defining events, methods and fields. Additionally, a namespace for the header file generation can be defined.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	AP Data Types	1..*	Used for specifying DataElements in service interfaces
Produces	Service Interface Description	1..*	Collection of all service interfaces

Table 3.5: Define Service Interfaces

3.2.1.3 Aggregate Service Interfaces

Task Definition	**Aggregate Service Interfaces**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Interface Definition::Tasks		
Brief Description	Aggregate service interfaces to a coarse-grained service interface.		
Description	In this optional task, it is possible to define coarse-grained service interfaces, which are used for network communication with the help of a service interface mapping. The service interface mapping maps the fine-grained service interfaces to the coarse-grained service interfaces. Alternatively, if the service interface mapping would result in a name clash due to equal names of some elements of the service interfaces, then the elements can be mapped by using the service interface element mapping.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Service Interface Description	0..*	Fine-grained service interfaces
Produces	Service Interface Description	0..*	Coarse-grained service interfaces
Produces	Service Interface Mapping	0..*	Mapping between fine-grained service and coarse-grained service interfaces

Table 3.6: Aggregate Service Interfaces

3.2.2 Work Products

3.2.2.1 AUTOSAR AP Standard Package

Deliverable	**Autosar AP Standard Package**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Interface Definition::Work Products		
Brief Description	Package with standardized AUTOSAR elements for the Adaptive Platform.		
Description	Package with standardized AUTOSAR elements (e.g. data types, service interfaces) for the Adaptive Platform. This deliverable is released by AUTOSAR and is read only within the methodology.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Develop Platform-level Application Software	0..1	In case standardized service interfaces are used for platform-level applications
Consumed by	Develop a Service Interface Description	0..1	Optional input for defining data types and service interfaces for the adaptive platform
Consumed by	Select or define Data Types for Adaptive Platform	0..1	Use standardized elements (e.g. data types, compu methods) to create the corresponding elements of the specific project.

Table 3.7: Autosar AP Standard Package

3.2.2.2 AP Data Types

Artifact	**AP Data Types**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Interface Definition::Work Products		
Brief Description	Definition of data types for the Adaptive Platform		
Description	Data types, which are required for the Adaptive Platform Instance and not already defined by AUTOSAR. The AP Data Types are used for specifying DataElements in service interfaces.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Select or define Data Types for Adaptive Platform	1..*	Defined AP Data Types for a specific project
Consumed by	Define Service Interfaces	1..*	Used for specifying DataElements in service interfaces

Table 3.8: AP Data Types

3.2.2.3 Service Interface Description

Deliverable	**Service Interface Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Interface Definition::Work Products		
Brief Description	Collection of service interfaces with events, methods and fields.		
Description	Collection of service interfaces. Service interfaces can consist of events, methods and fields and are the basis for the generation of header files for a software component. In addition, the namespace used for the header file generation can be defined.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define Service Interfaces	1..*	Collection of all service interfaces
Produced by	Develop a Service Interface Description	1..*	All service interfaces, which are used for communication
Produced by	Aggregate Service Interfaces	0..*	Coarse-grained service interfaces
Consumed by	Configure Service Interface Deployment	1	Deployment is configured for each service interface
Consumed by	Define and Configure Service Instances	1	Deployment of service interfaces needs to be configured
Consumed by	Define a signal-based Service Interface (SignalBasedServiceInterfaceDeployment)	1..*	Description of the Service Interfaces
Consumed by	Design Diagnostic Mapping	1..*	Collection of service interfaces. Service interfaces can consist of events, methods and fields.
Consumed by	Design Software Component for Adaptive Platform	1..*	All service interfaces that shall be implemented by the software component
Consumed by	Design service oriented communication between Classic and Adaptive Platform	1..*	Description of the Service Interfaces which communicate to CP in a service-oriented manner
Consumed by	Design signal oriented communication between Classic and Adaptive Platform	1..*	Description of the Service Interfaces which communicate to CP in a signal-oriented manner
Consumed by	Develop Adaptive Application Software	1..*	Service Interfaces are the basis for the development of adaptive application software
Consumed by	Generate Header Files for Service Interfaces	1..*	For all service interfaces header files are generated.

Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Generate Serialization Code for Adaptive Platform	1..*	Service interfaces that are implemented by the software components are needed for generating the serialization code
Consumed by	Map Diagnostic Data	1..*	Collection of service interfaces. Service interfaces can consist of events, methods and fields.
Consumed by	Map Event	1..*	Description of the Service Interfaces which communicate to CP in a service-oriented manner
Consumed by	Map Field	1..*	Description of the Service Interfaces which communicate to CP in a service-oriented manner
Consumed by	Map Fire and Forget	1..*	Description of the Service Interface which communicates to CP in a service-oriented manner
Consumed by	Map Method	1..*	Description of the Service Interfaces which communicate to CP in a service-oriented manner
Consumed by	Map ServiceInstance to Port Prototype	1..*	Description of the Service Interfaces
Consumed by	Map SignalBased EventDeployment to ISignal Triggerings	1..*	Description of the Service Interfaces
Consumed by	Map SignalBased FieldDeployment to ISignalTriggerings	1..*	Description of the Service Interfaces
Consumed by	Map SignalBased MethodDeployment to ISignal Triggerings	1..*	Description of the Service Interfaces
Consumed by	Configure Serialization for Adaptive Platform	0..1	Optional if you only configure default values for the serialization
Consumed by	Aggregate Service Interfaces	0..*	Fine-grained service interfaces
Consumed by	Integrate Software	0..*	Needed for defining the serialization

Table 3.9: Service Interface Description

3.2.2.4 Service Interface Mapping

Deliverable	**Service Interface Mapping**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Interface Definition::Work Products		
Brief Description	Mapping from fine-grained service interfaces to coarse-grained service interface.		
Description	The service interface mapping maps the fine-grained service interfaces to the coarse-grained service interfaces. In case of an element mapping, this work product contains the mapping of the elements of interfaces.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Aggregate Service Interfaces	0..*	Mapping between fine-grained service and coarse-grained service interfaces
Produced by	Develop a Service Interface Description	0..*	Optionally, coarse-grained service interfaces are defined by a service interface mapping

Table 3.10: Service Interface Mapping

3.3 Communication Mapping

3.3.1 Tasks

3.3.1.1 Map Method

Task Definition	**Map Method**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map Method		
Description	Map a ClientServerOperation located in a ClientServerInterface to a method located in a ServiceInterface. see TPS_MANI_03111 of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Client Server Interface Description	1..*	The descriptions of Client Server Interfaces of CP
Consumes	Service Interface Description	1..*	Description of the Service Interfaces which communicate to CP in a service-oriented manner
Produces	Service Interface Mapping for Service Oriented Communication	1..*	Service Interface Mappings

Table 3.11: Map Method

3.3.1.2 Map Event

Task Definition	**Map Event**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map Event		
Description	Map a VariableDataPrototype located in a SenderReceiverInterface to an event located in a ServiceInterface. see TPS_MANI_03112 of of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Sender Receiver Interface Description	1..*	The descriptions of Sender Receiver Interfaces of CP
Consumes	Service Interface Description	1..*	Description of the Service Interfaces which communicate to CP in a service-oriented manner
Produces	Service Interface Mapping for Service Oriented Communication	1..*	Service Interface Mappings

Table 3.12: Map Event

3.3.1.3 Map Field

Task Definition	**Map Field**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map Field		
Description	Map operations located in ClientServerOperations to getter and setter methods of a ServiceInterface. Map a VariableDataPrototype of a SenderReceiverInterface to the field notifier of the ServiceInterface. see TPS_MANI_03113 of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Client Server Interface Description	1..*	The descriptions of Client Server Interfaces of CP
Consumes	Sender Receiver Interface Description	1..*	The descriptions of Sender Receiver Interfaces of CP
Consumes	Service Interface Description	1..*	Description of the Service Interfaces which communicate to CP in a service-oriented manner
Produces	Service Interface Mapping for Service Oriented Communication	1..*	Service Interface Mappings

Table 3.13: Map Field

3.3.1.4 Map Fire and Forget

Task Definition	**Map Fire and Forget**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map Fire and Forget		
Description	Map a Fire&Forget method located in a ServiceInterface to a VariableDataPrototype in a SenderReceiverInterface or to a trigger of a TrigerInterface. see TPS_MANI_03115 of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Service Interface Description	1..*	Description of the Service Interface which communicates to CP in a service-oriented manner
Consumes	Sender Receiver Interface Description	0..*	The descriptions of Sender Receiver Interfaces of CP
Consumes	Trigger Interface Description	0..*	The descriptions of Trigger Interfaces
Produces	Service Interface Mapping for Service Oriented Communication	1..*	Service Interface Mappings

Table 3.14: Map Fire and Forget

3.3.1.5 Map SignalBasedMethod to ISignalTriggerings

Task Definition	**Map SignalBasedMethodDeployment to ISignalTriggerings**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map SignalBasedMethod to ISignalTriggerings		
Description	see TPS_MANI_03125 of of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	System Description	1	The System Description based on the System Template on the AUTOSAR classic platform
Consumes	Service Interface Description	1..*	Description of the Service Interfaces
Produces	Service Instance To Signal Mapping	1..*	Mapping of SignalBasedMethodDeployment to ISignalTriggerings

Table 3.15: Map SignalBasedMethodDeployment to ISignalTriggerings

3.3.1.6 Map SignalBasedEvent to ISignalTriggerings

Task Definition	**Map SignalBasedEventDeployment to ISignalTriggerings**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map SignalBasedEvent to ISignalTriggerings		
Description	see TPS_MANI_03124 of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	System Description	1	The System Description based on the System Template on the AUTOSAR classic platform
Consumes	Service Interface Description	1..*	Description of the Service Interfaces
Produces	Service Instance To Signal Mapping	1..*	Mapping of SignalBasedEventDeployment to ISignalTriggerings

Table 3.16: Map SignalBasedEventDeployment to ISignalTriggerings

3.3.1.7 Map SignalBasedField to ISignalTriggerings

Task Definition	**Map SignalBasedFieldDeployment to ISignalTriggerings**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map SignalBasedField to ISignalTriggerings		
Description	see TPS_MANI_03126 of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	System Description	1	The System Description based on the System Template on the AUTOSAR classic platform
Consumes	Service Interface Description	1..*	Description of the Service Interfaces
Produces	Service Instance To Signal Mapping	1..*	Mapping of SignalBasedFieldDeployment to ISignalTriggerings

Table 3.17: Map SignalBasedFieldDeployment to ISignalTriggerings

3.3.1.8 Map ServiceInstance to PortPrototype

Task Definition	**Map ServiceInstance to PortPrototype**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map ServiceInstance to PortPrototype		
Description	see TPS_MANI_03000 of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	System Description	1	The System Description based on the System Template on the AUTOSAR classic platform
Consumes	Service Interface Description	1..*	Description of the Service Interfaces

Relation Type	*Related Element*	*Mul.*	*Note*
Produces	Service Instance To Signal Mapping	1..*	Mapping of ServiceInstance to PortPrototype

Table 3.18: Map ServiceInstance to PortPrototype

3.3.2 Work Products

3.3.2.1 Client Server Interface Description

Deliverable	**Client Server Interface Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Work Products		
Brief Description	Client Server Interface Description		
Description	This represents the particular description of a ClientServerInterface of the Classic Platform.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumed by	Map Field	1..*	The descriptions of Client Server Interfaces of CP
Consumed by	Map Method	1..*	The descriptions of Client Server Interfaces of CP
Consumed by	Design service oriented communication between Classic and Adaptive Platform	0..*	The descriptions of Client Server Interfaces of CP are used to map a ClientServerOperation to a method in a ServiceInterface or to map a ClientServerOperation (representing getter or setter methods) to a field in a ServiceInterface

Table 3.19: Client Server Interface Description

3.3.2.2 Sender Receiver Interface Description

Deliverable	**Sender Receiver Interface Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Work Products		
Brief Description	Sender Receiver Interface Description		
Description	This represents a particular description of a SenderReceiverInterface of the Classic Platform.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumed by	Map Event	1..*	The descriptions of Sender Receiver Interfaces of CP
Consumed by	Map Field	1..*	The descriptions of Sender Receiver Interfaces of CP

Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Design service oriented communication between Classic and Adaptive Platform	0..*	The descriptions of Sender Receiver Interfaces of CP are used to map a VariableDataPrototype to an Event in a ServiceInterface or to map a VariableDataPrototype to the notifier of a Field of a ServiceInterface or to map a Fire&Forget Method that is located in a ServiceInterface to a VariableDataPrototype in a SenderReceiverInterface
Consumed by	Map Fire and Forget	0..*	The descriptions of Sender Receiver Interfaces of CP

Table 3.20: Sender Receiver Interface Description

3.3.2.3 Trigger Interface Description

Deliverable	**Trigger Interface Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Work Products		
Brief Description	Trigger Interface Description		
Description	This represents the particular description of the Trigger Interface of the Classic Platform.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Design service oriented communication between Classic and Adaptive Platform	0..*	The descriptions of Trigger Interfaces are used to map a Fire&Forget Method that is located in ServiceInterface to a Trigger in a TriggerInterface
Consumed by	Map Fire and Forget	0..*	The descriptions of Trigger Interfaces

Table 3.21: Trigger Interface Description

3.3.2.4 Service Interface Mapping Set

Deliverable	**Service Interface Mapping Set**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Work Products		
Brief Description	Service Interface Mapping Set		
Description	Collection of Service Interface mappings		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Service Interface Mapping for Service Oriented Communication	1..*	

Table 3.22: Service Interface Mapping Set

3.3.2.5 Service Interface Mapping for Service Oriented Communication

Artifact	**Service Interface Mapping for Service Oriented Communication**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Work Products		
Brief Description	Mappings for service oriented communication		
Description	Mappings of elements of AP-based ServiceInterfaces to elements of corresponding elements of CP-based SenderReceiverInterfaces, ClientServerInterfaces and TriggerInterfaces.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Design service oriented communication between Classic and Adaptive Platform	1..*	An InterfaceMapping results from the design of service-oriented communication between CP and AP
Produced by	Map Event	1..*	Service Interface Mappings
Produced by	Map Field	1..*	Service Interface Mappings
Produced by	Map Fire and Forget	1..*	Service Interface Mappings
Produced by	Map Method	1..*	Service Interface Mappings

Table 3.23: Service Interface Mapping for Service Oriented Communication

3.3.2.6 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	Define a signal-based Service Interface (SignalBasedServiceInterfaceDeployment)	1	The System Description based on the System Template on the AUTOSAR classic platform
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	Map ServiceInstance to Port Prototype	1	The System Description based on the System Template on the AUTOSAR classic platform
Consumed by	Map SignalBasedEventDeployment to ISignal Triggerings	1	The System Description based on the System Template on the AUTOSAR classic platform
Consumed by	Map SignalBasedFieldDeployment to ISignalTriggerings	1	The System Description based on the System Template on the AUTOSAR classic platform
Consumed by	Map SignalBasedMethodDeployment to ISignal Triggerings	1	The System Description based on the System Template on the AUTOSAR classic platform
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 3.24: System Description

3.3.2.7 Service Instance To Signal Mapping Set

Deliverable	**Service Instance To Signal Mapping Set**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Work Products		
Brief Description	Service Instance To Signal Mapping Set		
Description	Collection of Service Instance to Signal mappings		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Service Instance To Signal Mapping	1..*	

Table 3.25: Service Instance To Signal Mapping Set

3.3.2.8 Service Instance To Signal Mapping

Artifact	**Service Instance To Signal Mapping**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Work Products		
Brief Description	Mappings for signal oriented communication		
Description	Mappings of ServiceInstances to ISignalTriggerings.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Design signal oriented communication between Classic and Adaptive Platform	1..*	A signal-to-service mapping results from the design of signal-oriented communication between CP and AP
Produced by	Map ServiceInstance to Port Prototype	1..*	Mapping of ServiceInstance to PortPrototype
Produced by	Map SignalBased EventDeployment to ISignal Triggerings	1..*	Mapping of SignalBasedEventDeployment to ISignalTriggerings
Produced by	Map SignalBased FieldDeployment to ISignalTriggerings	1..*	Mapping of SignalBasedFieldDeployment to ISignalTriggerings
Produced by	Map SignalBased MethodDeployment to ISignal Triggerings	1..*	Mapping of SignalBasedMethodDeployment to ISignalTriggerings

Table 3.26: Service Instance To Signal Mapping

3.4 Machine Design

3.4.1 Tasks

3.4.1.1 Define and configure the network connections of a Machine

Task Definition	**Define and configure the network connections of a Machine**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Machine Design::Tasks		
Brief Description	Definition of all network endpoints with corresponding IP address.		
Description	Define all network connections of a Machine and their configuration out of contracting. All network endpoints with corresponding IP address are specified.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Topology	1	Description of (inter)connections between machines.
Produces	Machine Design	0..1	Definition of all network connections of a Machine and their configuration

Table 3.27: Define and configure the network connections of a Machine

3.4.1.2 Configure the Service Discovery Message Exchange

Task Definition	**Configure the Service Discovery Message Exchange**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Machine Design::Tasks		
Brief Description	Definition of ports and multicast IP addresses for service discovery message exchange		
Description	Define ports and multicast IP address over which the service discovery messages are exchanged.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Topology	1	Description of (inter)connections between machines.
Produces	Machine Design	0..1	Definition of ports and multicast IP address over which the service discovery messages are exchanged.

Table 3.28: Configure the Service Discovery Message Exchange

3.4.2 Work Products

3.4.2.1 Machine Design

Artifact	**Machine Design**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Machine Design::Work Products		
Brief Description	Proxy for a Machine at design time		
Description	This element stands in as a proxy for a Machine at the time when it does not exist, yet, i.e., at design time.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Develop the communication structure by means of MachineDesign	1	Configuration settings of the network connections and service discovery network exchange of a Machine
Produced by	Configure the Service Discovery Message Exchange	0..1	Definition of ports and multicast IP address over which the service discovery messages are exchanged.
Produced by	Define and configure the network connections of a Machine	0..1	Definition of all network connections of a Machine and their configuration
Consumed by	Define and Configure Service Instances	1	Service instances will be mapped to machine
Consumed by	Define and configure machine	1	Configuration settings of the network connections and service discovery network exchange of a Machine
Consumed by	Map Service Instance to Machine	1	Description of machine that the service instances shall be mapped to
Consumed by	Configure DoIP	0..1	Configuration settings of the network connections and service discovery network exchange of a Machine
Consumed by	Configure Log and Trace module	0..1	Configuration settings of the network connections and service discovery network exchange of a Machine
Consumed by	Configure NM module	0..1	Configuration settings of the network connections and service discovery network exchange of a Machine
Use meta model element	MachineDesign	1	

Table 3.29: Machine Design

3.5 Diagnostic Mapping

3.5.1 Tasks

3.5.1.1 Map Diagnostic Data

Task Definition	**Map Diagnostic Data**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Tasks		
Brief Description	Mapping between a diagnostic data element and an event or field		
Description	This task covers the mapping between a diagnostic data element (as part of the diagnostic protocol) and an event or field or even an element of an event or field of a DataPrototype aggregated by a ServiceInterface in the context of a PortPrototype. See [TPS_MANI_1037], [TPS_MANI_01060] and [constr_MANI_1496].		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Diagnostic Machine Extract	1	All available diagnostic information at the design time
Consumes	Service Interface Description	1..*	Collection of service interfaces. Service interfaces can consist of events, methods and fields.
Consumes	Software Component Description for Adaptive Platform	1..*	Description of a software component for the Adaptive Platform with all its ports, available at design time.
Produces	Diagnostic Mapping	1	One diagnostic data mapping

Table 3.30: Map Diagnostic Data

3.5.1.2 Map Diagnostic Enable Condition to Ports

Task Definition	**Map Diagnostic Enable Condition to Port(s)**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Tasks		
Brief Description	Mapping of a diagnostic enable condition to one or many service ports		
Description	This task covers the mapping of a diagnostic enable condition (as part of the diagnostic protocol) to one or many service ports of a particular application (instance) by means of SwcServiceDependency. See [TPS_MANI_01050] and [constr_1502]		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Diagnostic Machine Extract	1	All available diagnostic information at the design time
Consumes	Software Component Description for Adaptive Platform	1..*	Description of software component for the Adaptive Platform with all their (service) ports, known at design time.
Produces	Diagnostic Mapping	1	One diagnostic EnableConditionToPorts mapping

Table 3.31: Map Diagnostic Enable Condition to Port(s)

3.5.1.3 Map Diagnostic Event to Ports

Task Definition	**Map Diagnostic Event to Port(s)**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Tasks		
Brief Description	Mapping of a diagnostic event to one or many service ports		
Description	This task covers the mapping of a diagnostic event (as part of the diagnostic protocol) to one or many service ports of a particular application (instance) by means of SwcServiceDependency. See [TPS_MANI_01048] and [constr_1500].		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Diagnostic Machine Extract	1	All available diagnostic information at the design time
Consumes	Software Component Description for Adaptive Platform	1..*	Description of software component for the Adaptive Platform with all their (service) ports, known at design time.
Produces	Diagnostic Mapping	1	One diagnostic EventToPort mapping

Table 3.32: Map Diagnostic Event to Port(s)

3.5.1.4 Map Diagnostic Storage Condition to Ports

Task Definition	**Map Diagnostic Storage Condition to Port(s)**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Tasks		
Brief Description	Mapping of a diagnostic storage condition to one or many service ports		
Description	This task covers the mapping of a diagnostic storage condition (as part of the diagnostic protocol) to one or many service ports of a particular application (instance) by means of SwcServiceDependency. See [TPS_MANI_01051] and [constr_1503]		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Diagnostic Machine Extract	1	All available diagnostic information at the design time
Consumes	Software Component Description for Adaptive Platform	1..*	Description of software component for the Adaptive Platform with all their (service) ports, known at design time.
Produces	Diagnostic Mapping	1	One diagnostic StorageConditionToPorts mapping

Table 3.33: Map Diagnostic Storage Condition to Port(s)

3.5.1.5 Map Diagnostic Software Mapping

Task Definition	**Diagnostic Software Mapping**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Tasks		
Brief Description	Mapping between a DiagnosticServiceInstance and a SwcServiceDependency		
Description	This task covers the mapping between a DiagnosticServiceInstance and a SwcServiceDependency, defined in the context of an AdaptiveApplicationSwComponent Type. See [TPS_MANI_01038] and [constr_1499].		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Diagnostic Machine Extract	1	All available diagnostic information at the design time
Produces	Diagnostic Mapping	1	One diagnostic software mapping

Table 3.34: Diagnostic Software Mapping

3.5.1.6 Map Diagnostic Operation Cycle to Ports

Task Definition	**Map Diagnostic Operation Cycle to Port(s)**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Tasks		
Brief Description	Mapping of a diagnostic operation cycle to one or many service ports		
Description	This task covers the mapping of a diagnostic operation cycle (as part of the diagnostic protocol) to one or many service ports of a particular application (instance) by means of SwcServiceDependency. See [TPS_MANI_01049] and [constr_1501].		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Diagnostic Machine Extract	1	All available diagnostic information at the design time
Consumes	Software Component Description for Adaptive Platform	1..*	Description of software component for the Adaptive Platform with all their (service) ports, known at design time.
Produces	Diagnostic Mapping	1	One diagnostic OperationCycleToPorts mapping

Table 3.35: Map Diagnostic Operation Cycle to Port(s)

3.5.1.7 Associate a DiagnosticMapping with a ProcessDesign

Task Definition	**Associate DiagnosticMapping with ProcessDesign**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Tasks		
Brief Description	Associate one DiagnosticMapping with one ProcessDesign		
Description	It may be necessary that different instances of a particular application software require different diagnostic mappings. Therefore, a relation between a particular diagnostic mapping and a particular Process needs to be established. This assignment may be done in a final extra step, represented by this task. To accommodate for this potential modeling, the reference from a diagnostic mapping to ProcessDesign has been decorated by stereotype "atpSplitable".		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Diagnostic Mapping	1..*	The diagnostic mapping for a Machine, except the linkage between the mappings and the corresponding ProcessDesigns
Consumes	Process Design	1..*	All dedicated ProssesDesigns for a Machine
Produces	Diagnostic Mapping	1..*	fully: The linkage between the diagnostic mappings and the corresponding ProcessDesigns

Table 3.36: Associate DiagnosticMapping with ProcessDesign

3.5.2 Work Products

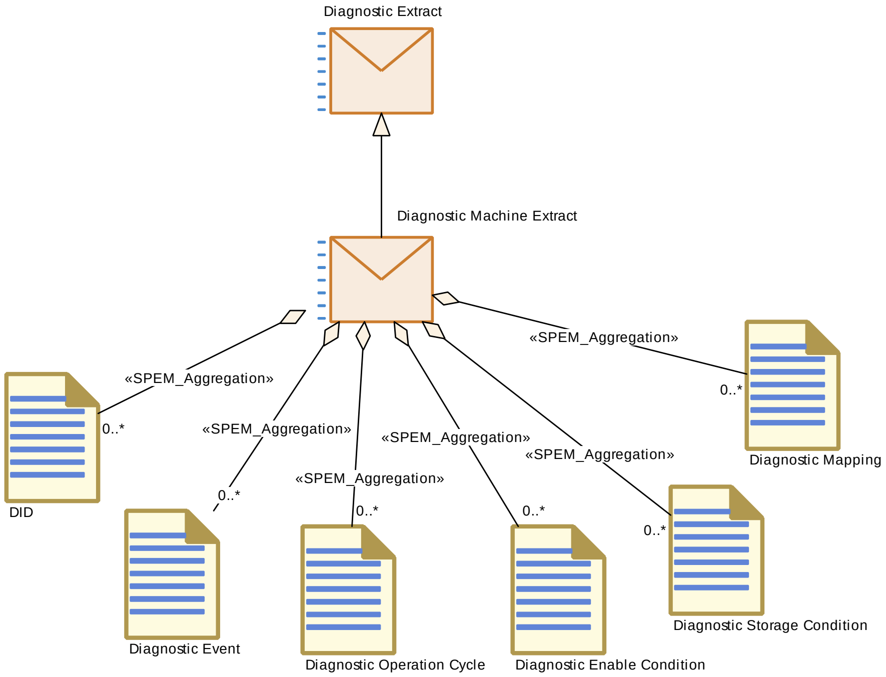

Figure 3.1: Structure of the Diagnostic Machine Extract

3.5.2.1 Diagnostic Machine Extract

Deliverable	**Diagnostic Machine Extract**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Work Products		
Brief Description	Diagnostic information of a Machine		
Description	This deliverable contains diagnostic information of a Machine.		
Kind	AUTOSAR XML		
Extends	Diagnostic Extract		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	DID	0..*	
Aggregates	Diagnostic Enable Condition	0..*	
Aggregates	Diagnostic Event	0..*	
Aggregates	Diagnostic Mapping	0..*	
Aggregates	Diagnostic Operation Cycle	0..*	
Aggregates	Diagnostic Storage Condition	0..*	
Consumed by	Design Diagnostic Mapping	1	All available diagnostic information at the design time
Consumed by	Diagnostic Software Mapping	1	All available diagnostic information at the design time
Consumed by	Map Diagnostic Data	1	All available diagnostic information at the design time
Consumed by	Map Diagnostic Enable Condition to Port(s)	1	All available diagnostic information at the design time
Consumed by	Map Diagnostic Event to Port(s)	1	All available diagnostic information at the design time
Consumed by	Map Diagnostic Operation Cycle to Port(s)	1	All available diagnostic information at the design time
Consumed by	Map Diagnostic Storage Condition to Port(s)	1	All available diagnostic information at the design time
Consumed by	Collect belonging (software) artifacts of Sub Software Clusters	0..1	Diagnostic extract for a Machine
Consumed by	Create Software Package	0..1	Diagnostic extract for a Machine
Consumed by	Identify necessary (software) artifacts	0..1	Diagnostic extract for a Machine
Consumed by	Set Up Initial Machine	0..1	Diagnostic extract for a Machine

Table 3.37: Diagnostic Machine Extract

3.5.2.2 DID

Artifact	**DID**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Work Products		
Brief Description			
Description	This represents the definition of a diagnostic data identifier. Data Identified according to ISO 14229-1[1]. This 16 bit value uniquely defines one ore more data elements (parameters) that can are used in diagnostics to read, write or control data.		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Diagnostic Machine Extract	0..*	

Table 3.38: DID

3.5.2.3 Diagnostic Enable Condition

Artifact	**Diagnostic Enable Condition**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Work Products		
Brief Description			
Description	Represents the definition of a diagnostic enable condition.		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Diagnostic Machine Extract	0..*	

Table 3.39: Diagnostic Enable Condition

3.5.2.4 Diagnostic Event

Artifact	**Diagnostic Event**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Work Products		
Brief Description			
Description	Represents the definition of a diagnostic event. A diagnostic event uniquely identifies a fault path of the system. An application monitors the system and reports events to the DM.		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Diagnostic Machine Extract	0..*	

Table 3.40: Diagnostic Event

3.5.2.5 Diagnostic Mapping

Artifact	**Diagnostic Mapping**
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Work Products
Brief Description	Diagnostic Mappings
Description	This represents the mapping of information related to the diagnostic protocol content and the application software. In detail, it contains the results of the following tasks: • DiagnosticServiceDataMapping • DiagnosticServiceSwMapping • DiagnosticEventPortMapping • DiagnosticOperationCyclePortMapping • DiagnosticEnableConditionPortMapping • DiagnosticStorageConditionPortMapping
Kind	AUTOSAR XML

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Diagnostic Machine Extract	0..*	
Produced by	Diagnostic Software Mapping	1	One diagnostic software mapping
Produced by	Map Diagnostic Data	1	One diagnostic data mapping
Produced by	Map Diagnostic Enable Condition to Port(s)	1	One diagnostic EnableConditionToPorts mapping
Produced by	Map Diagnostic Event to Port(s)	1	One diagnostic EventToPort mapping
Produced by	Map Diagnostic Operation Cycle to Port(s)	1	One diagnostic OperationCycleToPorts mapping
Produced by	Map Diagnostic Storage Condition to Port(s)	1	One diagnostic StorageConditionToPorts mapping
Produced by	Associate DiagnosticMapping with Process Design	1..*	fully: The linkage between the diagnostic mappings and the corresponding ProcessDesigns
Produced by	Design Diagnostic Mapping	1..*	partially: The diagnostic mapping for a Machine, except the linkage between the mappings and the corresponding ProcessDesigns
Consumed by	Associate DiagnosticMapping with Process Design	1..*	The diagnostic mapping for a Machine, except the linkage between the mappings and the corresponding ProcessDesigns

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

Table 3.41: Diagnostic Mapping

3.5.2.6 Diagnostic Operation Cycle

Artifact	**Diagnostic Operation Cycle**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Work Products		
Brief Description			
Description	Represents a definition of an operation cycle that is base of the event qualifying and for DEM scheduling. An operation cycle is the execution of monitor within an application, from a start point to a defined end point inside the application run.		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Diagnostic Machine Extract	0..*	

Table 3.42: Diagnostic Operation Cycle

3.5.2.7 Diagnostic Storage Condition

Artifact	**Diagnostic Storage Condition**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Work Products		
Brief Description			
Description	Represents the definition of a diagnostic storage condition.		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Diagnostic Machine Extract	0..*	

Table 3.43: Diagnostic Storage Condition

3.6 Adaptive Application

This chapter contains the definition of work products and tasks used for the defintion of service interfaces for the Adaptive Platform.

3.6.1 Tasks

3.6.1.1 Generate Header Files for Service Interfaces

Task Definition	**Generate Header Files for Service Interfaces**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
Brief Description	Generate header files for service interfaces with proxies and skeletons		
Description	Header files are generated based on service interfaces. Therefore, the header files are generated regardless of the usage of services by a specific software component. For each service interface one proxy header file and one skeleton header file is generated. The generation contains the header files for the implementation of the software component as well as the service proxies and skeletons, which need to be implemented.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Service Interface Description	1..*	For all service interfaces header files are generated.
Produces	Header Files for Service Interfaces	1..*	One proxy header file and one skeleton header file per service interface are generated.

Table 3.44: Generate Header Files for Service Interfaces

3.6.1.2 Design Software Component for Adaptive Platform

Task Definition	**Design Software Component for Adaptive Platform**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
Brief Description	Design a software component with ports that implement service interfaces.		
Description	A software component is defined with its ports. Each port implements a service interface. If a software component requires a service interface, an RPort is used. If it provides a service interface, an PPort is used. A hierarchy of software components is described by a composition.		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Tier 2	1	Application Software Designer: The design of software components will probably be performed by an Application Software Designer of a Tier 2 company
Consumes	Service Interface Description	1..*	All service interfaces that shall be implemented by the software component
Produces	Software Component Description for Adaptive Platform	1	Software component model with the ports that implement service interfaces

Table 3.45: Design Software Component for Adaptive Platform

3.6.1.3 Implement Software Component Functionality

Task Definition	**Implement Software Component Functionality**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
Brief Description	Implement the core functionality of the software component.		
Description	In this task, the core functionality of the software component is implemented. This can be done independently of the main function of the executable, where the scheduling local to the executable is described.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Header Files for Service Interfaces	1..*	Proxy and skeleton header files are the basis for implementing the software component
Consumes	Software Component Description for Adaptive Platform	1..*	The software component model as input for the implementation of the software component.
Produces	Software Component Source Code	1	The source code of the software component

Table 3.46: Implement Software Component Functionality

3.6.1.4 Compile Software Component

Task Definition	**Compile Software Component**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
Brief Description	Compile the software component in order to produce object code.		
Description	Compile the software component together with the header files for service interfaces. This task can be performed by the application developer in case software component object code shall be delivered. In this case, the used compiler and compiler settings need to be agreed on between application developer and integrator. This Build Chain Configuration is given beforehand to the application developer. On the other hand, this task can be performed by the integrator. In this case, the application developer has delivered the source code directly to the integrator.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Build Chain Configuration	1	Settings used for compiling the software component
Consumes	Software Component Source Code	1	Source code of the software component for compilation
Consumes	Header Files for Service Interfaces	1..*	Used header files of the software component for compilation
Consumes	Middleware Library Header Files	0..*	Library header files needed for compiling the software components
Produces	Software Component Object Code	1	Object code of the software component after compilation

Table 3.47: Compile Software Component

3.6.1.5 Develop Main Function

Task Definition	**Develop Main Function**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
Brief Description	Develop the main function for one executable.		
Description	For one executable, which can contain several software components, one main function is developed. The main function defines the control flow of the executable including the scheduling of the software components inside the executable.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Software Component Source Code	1..*	Scheduling and communication of several software components within one executable is defined
Produces	Main Function	1	One main function per executable

Table 3.48: Develop Main Function

3.6.1.6 Configure Serialization for Adaptive Platform

Task Definition	**Configure Serialization for Adaptive Platform**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
Brief Description	Define serialization properties for the Adaptive Platform		
Description	Define the properties of the serialization, i.e. how the data in the service interfaces shall be serialized for the transport on SOME/IP. The alignment, session handling, size of length indicator and endianness needs to be defined.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Service Interface Description	0..1	Optional if you only configure default values for the serialization
Produces	Serialization Configuration	1..*	Serialization properties for the service interfaces

Table 3.49: Configure Serialization for Adaptive Platform

3.6.1.7 Generate Serialization Code for Adaptive Platform

Task Definition	**Generate Serialization Code for Adaptive Platform**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
Brief Description	Generate serialization code for service interfaces.		
Description	Generate the serialization code based on the configuration settings.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Serialization Configuration	1..*	Configuration settings are the basis for generating the serialization code.

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Service Interface Description	1..*	Service interfaces that are implemented by the software components are needed for generating the serialization code
Produces	Serialization Source Code	1	Source code for the serialization can be generated

Table 3.50: Generate Serialization Code for Adaptive Platform

3.6.1.8 Implement Service Proxies and Skeletons

Task Definition	**Implement Service Proxies and Skeletons**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
Brief Description	Implement service proxies and skeletons for an Adaptive Platform		
Description	Service proxies and skeletons for an Adaptive Platform, i.e. the method calls that are used for service-oriented communication, are implemented. The implementation is based on the serialization settings for the platform.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Header Files for Service Interfaces	1..*	Header files contain proxies and skeletons to be implemented
Consumes	Serialization Configuration	1..*	Serialization of data is needed for implementing service proxies and skeletons
Produces	Implemented Proxies and Skeletons	1..*	Implementation of service proxies and skeletons given as source code

Table 3.51: Implement Service Proxies and Skeletons

3.6.1.9 Build Executable Application

Task Definition	**Build Executable Application**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
Brief Description	Build executable application based on several software components.		
Description	The software components are linked together with the serialization code and necessary middleware libraries. Together with the main function, the executable application is build.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Build Chain Configuration	1	Settings for the compiler and linker
Consumes	Main Function	1	One main function per executable
Consumes	Serialization Source Code	0..1	Serialization for the executable
Consumes	Implemented Proxies and Skeletons	0..*	Source code of service proxies and skeletons
Consumes	Middleware Libraries	0..*	Libraries needed to build the executable

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Platform Object Code	0..*	Platform modules to be linked together to one executable
Consumes	Software Component Object Code	0..*	Software component to be linked together to one executable
Produces	Executable Application	1	One executable is built

Table 3.52: Build Executable Application

3.6.2 Work Products

3.6.2.1 Header Files for Service Interfaces

Deliverable	**Header Files for Service Interfaces**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
Brief Description	Header files generated for service interfaces		
Description	The generated header files of service interfaces consist of • proxy header files for service discovery and method invocation as well as event subscription and reception • skeleton header files for method calls and event publishing The header files are the basis for implementing the functionality of a software component.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Generate Header Files for Service Interfaces	1..*	One proxy header file and one skeleton header file per service interface are generated.
Consumed by	Compile Software Component	1..*	Used header files of the software component for compilation
Consumed by	Implement Service Proxies and Skeletons	1..*	Header files contain proxies and skeletons to be implemented
Consumed by	Implement Software Component Functionality	1..*	Proxy and skeleton header files are the basis for implementing the software component
Consumed by	Integrate Software	0..*	Proxies and skeletons to be implemented

Table 3.53: Header Files for Service Interfaces

3.6.2.2 Software Component Description for Adaptive Platform

Deliverable	**Software Component Description for Adaptive Platform**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
Brief Description	Description of a software component for the Adaptive Platform		
Description	Description of a software component for the Adaptive Platform with all its ports. A RPort is used, if the software component requires a service interface. A PPort is used, if the software component provides a service interface. A software component can also be of type composition.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Design Software Component for Adaptive Platform	1	Software component model with the ports that implement service interfaces
Produced by	Develop Adaptive Application Software	1..*	Output of component model for the software components
Consumed by	Define and Configure Service Instances	1	Used to map the service instances to ports of a software component
Consumed by	Map Service Instance to Port Prototype	1	In case the service instances are mapped to ports of a software component
Consumed by	Design Diagnostic Mapping	1..*	Description of a software component for the Adaptive Platform with all its ports, available at design time.
Consumed by	Implement Software Component Functionality	1..*	The software component model as input for the implementation of the software component.
Consumed by	Map Diagnostic Data	1..*	Description of a software component for the Adaptive Platform with all its ports, available at design time.
Consumed by	Map Diagnostic Enable Condition to Port(s)	1..*	Description of software component for the Adaptive Platform with all their (service) ports, known at design time.
Consumed by	Map Diagnostic Event to Port(s)	1..*	Description of software component for the Adaptive Platform with all their (service) ports, known at design time.
Consumed by	Map Diagnostic Operation Cycle to Port(s)	1..*	Description of software component for the Adaptive Platform with all their (service) ports, known at design time.
Consumed by	Map Diagnostic Storage Condition to Port(s)	1..*	Description of software component for the Adaptive Platform with all their (service) ports, known at design time.

Table 3.54: Software Component Description for Adaptive Platform

3.6.2.3 Build Chain Configuration

Deliverable	**Build Chain Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
Brief Description	Used compiler and compiler settings for building the executable		
Description	The Build Chain Configuration contains the used compiler and compiler settings. These settings are platform implementation specific.		
Kind	Text		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Build Executable Application	1	Settings for the compiler and linker
Consumed by	Compile Software Component	1	Settings used for compiling the software component
Consumed by	Integrate Software	1	Needed for linking all artifacts

Table 3.55: Build Chain Configuration

3.6.2.4 Software Component Source Code

Deliverable	**Software Component Source Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
Brief Description	Source code of the core functionality of a software component		
Description	This deliverable contains the source code of the core functionality of a software component. The deliverable includes documentation of the software component. In case the integrator is completely responsible for the compilation of the software components and the build of the executable, the source code will be delivered directly.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Implement Software Component Functionality	1	The source code of the software component
Produced by	Develop Adaptive Application Software	0..*	Software components as source code
Consumed by	Compile Software Component	1	Source code of the software component for compilation
Consumed by	Develop Main Function	1..*	Scheduling and communication of several software components within one executable is defined
Consumed by	Integrate Software	0..*	Source code for application-level executable

Table 3.56: Software Component Source Code

3.6.2.5 Software Component Object Code

Deliverable	**Software Component Object Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
Brief Description	Object code of one software component		
Description	Compiled software component source code. Since these software components belong to application-level executables, their implementation is restricted to use the standardized ara API.		
Kind	Object Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Compile Software Component	1	Object code of the software component after compilation
Produced by	Develop Adaptive Application Software	0..*	Compiled software components
Consumed by	Build Executable Application	0..*	Software component to be linked together to one executable
Consumed by	Integrate Software	0..*	Object code for application-level executable

Table 3.57: Software Component Object Code

3.6.2.6 Serialization Configuration for Adaptive Platform

Deliverable	**Serialization Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
Brief Description	Configuration of serialization of the data in the service interface		
Description	Settings necessary for the serialization of the data in the service interfaces. For SOME/IP, this is e.g. the length of length fields that is put in front of an array.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Configure Serialization for Adaptive Platform	1..*	Serialization properties for the service interfaces
Consumed by	Generate Serialization Code for Adaptive Platform	1..*	Configuration settings are the basis for generating the serialization code.
Consumed by	Implement Service Proxies and Skeletons	1..*	Serialization of data is needed for implementing service proxies and skeletons

Table 3.58: Serialization Configuration

3.6.2.7 Serialization Source Code

Artifact	**Serialization Source Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
Brief Description	Serialization of data		
Description	Source code for serializing data with SOME/IP.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Generate Serialization Code for Adaptive Platform	1	Source code for the serialization can be generated
Consumed by	Build Executable Application	0..1	Serialization for the executable

Table 3.59: Serialization Source Code

3.6.2.8 Implemented Service Proxies and Skeletons

Artifact	**Implemented Proxies and Skeletons**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
Brief Description	Implemented service proxies and skeletons		
Description	Implemented source code for the service proxies and skeletons.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Implement Service Proxies and Skeletons	1..*	Implementation of service proxies and skeletons given as source code
Consumed by	Build Executable Application	0..*	Source code of service proxies and skeletons

Table 3.60: Implemented Proxies and Skeletons

3.6.2.9 Main Function

Deliverable	**Main Function**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
Brief Description	Main function of executable application		
Description	This artifact is the main function for one executable. It contains the control flow of the executable including the scheduling of the software components inside the executable.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Develop Adaptive Application Software	1	One main function per executable is produced
Produced by	Develop Main Function	1	One main function per executable

Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Develop Platform-level Application Software	1	Main function for platform-level executable
Consumed by	Build Executable Application	1	One main function per executable
Consumed by	Integrate Software	1	One main function per executable

Table 3.61: Main Function

3.6.2.10 Executable Application

Deliverable	**Executable Application**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
Brief Description	Executable application containing several software components		
Description	The executable application, or just executable, can contain an arbitrary hierarchy of software components. The software components contain the functionality of the executable. Several executables can be packaged into an ExecutableGroup. They can be of category application-level or platform-level.		
Kind	Executable		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Build Executable Application	1	One executable is built
Produced by	Integrate Software	1	Software is integrated into one executable application
Consumed by	Create Application Manifest	1	One executable can be instantiated several times
Consumed by	Define Process	1	Executable to be instantiated
Consumed by	Collect belonging (software) artifacts of Sub Software Clusters	0..*	Executables of deployed processes
Consumed by	Create Software Package	0..*	Executables of deployed processes
Consumed by	Identify necessary (software) artifacts	0..*	Executables of deployed processes
Consumed by	Set Up Initial Machine	0..*	Executables of those Platform modules and Adaptive Applications that should run on a initially configured machine. Beside the OS, at least the UCM and connected Platform modules (e.g., a diagnostic communication manager) need to be installed in order to be able to upload other software.

Table 3.62: Executable Application

3.7 Platform and Machine

This chapter contains the definition of work products and tasks, which are used for the definition and configuration of a machine.

3.7.1 Tasks

3.7.1.1 Define ECU Description

The reference to the performing role is given in [1].

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..*	Decription of the ECU

Table 3.63: Define ECU Description

3.7.1.2 Describe Available HW Resources

Task Definition	**Describe Available HW Resources**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
Brief Description	Description of available hardware resources for the machine		
Description	Optional step for describing available hardware resources for the Machine.		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	ECU Resources Description	1	Definition of available HW resources for the Machine based on the description of the ECU
Produces	Machine Manifest	0..1	Available hardware resources of machine

Table 3.64: Describe Available HW Resources

3.7.1.3 Define Machine States

Task Definition	**Define Machine States**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
Brief Description	Define additional states of the machine		
Description	Define states (modes) of the Machine. These states can later be used for defining a startup configuration and execution dependencies for a process per machine state.		
Relation Type	***Related Element***	***Mul.***	***Note***
Produces	Machine States	0..1	States defined for the Machine

Table 3.65: Define Machine States

3.7.1.4 Define Function Groups

Task Definition	**Define Function Groups**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
Brief Description	Define Function groups of the Machine		
Description	Define function group states of the Machine. Function groups with function group states individually control groups of functionally coherent Application processes.		
Relation Type	***Related Element***	***Mul.***	***Note***
Produces	Function Groups	0..1	Function groups defined for the Machine

Table 3.66: Define Function Groups

3.7.1.5 Define State Timeouts

Task Definition	**Define State Timeouts**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
Brief Description	Define timeouts for machine states (modes) or function group states		
Description	Define timeouts for machine states (modes) or function group states. It is possible to define EnterExitTimeouts for selected machine states or function group states.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Function Groups	1	Function Groups of a Machine

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Machine States	1	Machine States of a Machine
Produces	PerStateTimeouts	0..1	PerState Timeouts defined for a Machine

Table 3.67: Define State Timeouts

3.7.1.6 Map Process To Machine

Task Definition	**Map Process To Machine**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
Brief Description	Map processes to a particular Machine		
Description	Map processes to a particular Machine.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Process	1	Description of a dedicated Process
Produces	ProcessToMachineMapping	1	Mapping of exactly one Process to exactly one Machine

Table 3.68: Map Process To Machine

3.7.1.7 Configure OS for Adaptive Platform

Task Definition	**Configure OS for Adaptive Platform**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
Brief Description	Configuration of the platform and the platform modules		
Description	Configure the operating system, e.g. the resource groups and the timer granularity can be defined.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Operating System for Adaptive Platform	1	OS to be configured
Produces	Machine Manifest	0..1	Configuration settings of OS

Table 3.69: Configure OS for Adaptive Platform

3.7.1.8 Configure Log and Trace module

Task Definition	**Configure Log and Trace module**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
Brief Description	Configure the Log and Trace module		
Description	Define the Machine-specific configuration settings for the Log and Trace functional cluster.		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Machine Design	0..1	Configuration settings of the network connections and service discovery network exchange of a Machine
Produces	Machine Manifest	1	Configuration of the Log and Trace module

Table 3.70: Configure Log and Trace module

3.7.1.9 Configure DoIP

Task Definition	**Configure DoIP**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
Brief Description	Configure DoIP		
Description	Define the Machine-specific configuration settings for DoIP.		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Machine Design	0..1	Configuration settings of the network connections and service discovery network exchange of a Machine
Produces	Machine Manifest	0..1	Configuration of DoIP

Table 3.71: Configure DoIP

3.7.1.10 Configure NM module

Task Definition	**Configure NM module**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
Brief Description	Configure the NM module		
Description	Define the Machine-specific configuration settings for the NM module.		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Machine Design	0..1	Configuration settings of the network connections and service discovery network exchange of a Machine
Produces	Machine Manifest	0..1	Configuration of the NM module

Table 3.72: Configure NM module

3.7.2 Work Products

3.7.2.1 Middleware Library Header Files

Artifact	**Middleware Library Header Files**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Platform::Work Products		
Brief Description	Header files of middleware libraries		
Description	Header files of middleware libraries, which are needed for application development.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Compile Software Component	0..*	Library header files needed for compiling the software components
Consumed by	Develop Platform-level Application Software	0..*	Library header files needed for compiling the platform-level applications

Table 3.73: Middleware Library Header Files

3.7.2.2 Middleware Libraries

Artifact	**Middleware Libraries**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Platform::Work Products		
Brief Description	Middleware libraries that are needed in order to build the executable		
Description	Object code of middleware libraries. These are linked together with other object code in order to build an Executable Application.		
Kind	Object Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Build Executable Application	0..*	Libraries needed to build the executable

Table 3.74: Middleware Libraries

3.7.2.3 ECU Resources Description

The references to other tasks and work products are given in [1].

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***

Relation Type	***Related Element***	***Mul.***	***Note***
Aggregated by	Complete ECU Description	1	
Produced by	Define ECU Description	1..*	Decription of 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	Define and configure machine	0..1	All resources which are available for the ECU
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 3.75: ECU Resources Description

3.7.2.4 Configured Machine on Adaptive ECU

Deliverable	**Configured Machine on Adaptive ECU**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Work Products		
Brief Description	Configured Adaptive Platform instance		
Description	This work product is a configured Adaptive Platform instance, i.e. a configured machine, where software can be deployed on. The configuration settings are based on the Machine Manifest.		
Kind	Custom		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Set Up Initial Machine	1	Machine is configured and software can now be deployed

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

Table 3.76: Configured Machine on Adaptive ECU

3.7.2.5 Machine Manifest

Deliverable	**Machine Manifest**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Work Products		
Brief Description	Configuration of the machine		
Description	Description of deployment content for the configuration of the machine, independent of any service instances or applications.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Function Groups	1	Function Groups configuration of a machine
Aggregates	Machine States	1	Machine Groups configuration of a Machine
Aggregates	PerStateTimeouts	1	PerState Timeouts configuration of a Machine
Aggregates	ProcessToMachineMapping	1..*	All ProcessToMachineMappings of a Machine
Produced by	Configure Log and Trace module	1	Configuration of the Log and Trace module
Produced by	Define and configure machine	1	The machine manifest describes all the configuration settings for one Machine
Produced by	Configure DoIP	0..1	Configuration of DoIP
Produced by	Configure NM module	0..1	Configuration of the NM module
Produced by	Configure OS for Adaptive Platform	0..1	Configuration settings of OS
Produced by	Describe Available HW Resources	0..1	Available hardware resources of machine
Consumed by	Create Application Manifest	1	Instantiation is defined on one specific machine
Consumed by	Define Execution Dependencies	1	Execution dependencies are defined per machine mode.
Consumed by	Define Startup Configuration	1	Startup configuration is defined per machine mode given in the Machine Manifest
Consumed by	Set Up Initial Machine	1	Containing all configuration settings for the Machine
Use meta model element	Machine	1	

Table 3.77: Machine Manifest

3.7.2.6 Platform Object Code

Deliverable	**Platform Object Code**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Platform::Work Products		
Brief Description	Object code of platform-level software		
Description	This is the object code of platform modules. It might be based on standardized service interfaces, as e.g. for the Adaptive Diagnostic Manager, where part of the platform module has been implemented in terms of a software component. Alternatively, the implementation is not based on software components and hence pure platform object code (as e.g. Execution Management). A main function is needed in order to build the executable application.		
Kind	Object Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Develop Platform-level Application Software	1..*	Object code of platform module
Consumed by	Build Executable Application	0..*	Platform modules to be linked together to one executable
Consumed by	Integrate Software	0..*	Object code for platform-level executable

Table 3.78: Platform Object Code

3.7.2.7 Operating System for Adaptive Platform

Deliverable	**Operating System for Adaptive Platform**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Platform::Work Products		
Brief Description	Operating System for the Adaptive Platform		
Description	The operating system for the Adaptive Platform is a platform module, which does not have an Application Manifest and therefore does not follow the workflow of platform-level applications. The OS is the basis for configuring and setting up the machine.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Select OS Distribution	1	Selected OS distribution
Consumed by	Configure OS for Adaptive Platform	1	OS to be configured
Consumed by	Define and configure machine	1	OS to be configured
Consumed by	Set Up Initial Machine	1	OS to be installed on machine

Table 3.79: Operating System for Adaptive Platform

3.7.2.8 Process to Machine Mapping

Artifact	**ProcessToMachineMapping**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Define and Configure Machine::Machine Configuration		
Brief Description			
Description	An ProcessToMachineMapping links exactly one Process to one machine.		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Map Process To Machine	1	Mapping of exactly one Process to exactly one Machine

Table 3.80: ProcessToMachineMapping

3.7.2.9 Function Groups

Artifact	**Function Groups**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Define and Configure Machine::Machine Configuration		
Brief Description			
Description	This artifact contains the configuration of function groups of a machine.		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define Function Groups	0..1	Function groups defined for the Machine
Consumed by	Define State Timeouts	1	Function Groups of a Machine

Table 3.81: Function Groups

3.7.2.10 Machine States

Artifact	**Machine States**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Define and Configure Machine::Machine Configuration		
Brief Description			
Description	This artifact contains the configuration of machine states of a machine.		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define Machine States	0..1	States defined for the Machine
Consumed by	Define State Timeouts	1	Machine States of a Machine

Table 3.82: Machine States

3.7.2.11 PerState Timeouts

Artifact	**PerStateTimeouts**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Define and Configure Machine::Machine Configuration		
Brief Description			
Description	This artifact contains the configuration of timeouts for selected machine states and function group states.		
Kind			
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define State Timeouts	0..1	PerState Timeouts defined for a Machine

Table 3.83: PerStateTimeouts

3.8 Application Manifest

This chapter contains the definition of work products and tasks, which are used for creating the application manifest.

3.8.1 Tasks

3.8.1.1 Define Process

Task Definition	**Define Process**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Application Manifest::Tasks		
Brief Description	Define a process as an instantiation of an executable		
Description	Define the instantiation of executables. An executable can be instantiated several times (e.g. with different startup parameters) resulting in different processes.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Executable Application	1	Executable to be instantiated
Produces	Process	1..*	Different instantiation of executables can result in different processes.

Table 3.84: Define Process

3.8.1.2 Define Startup Configuration

Task Definition	**Define Startup Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Application Manifest::Tasks		
Brief Description	Define the startup configuration for one process		
Description	Define the startup configuration for one process per machine mode.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Machine Manifest	1	Startup configuration is defined per machine mode given in the Machine Manifest
Consumes	Process	1	Startup configuration to be defined for process
Produces	Mode-dependent Startup Configuration	1..*	Startup configuration of a process for each mode

Table 3.85: Define Startup Configuration

3.8.1.3 Define Execution Dependencies

Task Definition	**Define Execution Dependencies**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Application Manifest::Tasks		
Brief Description	Define execution dependencies to other processes		
Description	Define the execution dependencies for one process to other processes per machine mode. Referencing other processes means that they shall be launched before this process is started.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Machine Manifest	1	Execution dependencies are defined per machine mode.
Consumes	Process	1	Execution dependencies defined for one process
Produces	Mode-dependent Startup Configuration	1..*	Execution dependencies of a process for each mode

Table 3.86: Define Execution Dependencies

3.8.1.4 Associate Process with Process Design

Task Definition	**Associate Process with ProcessDesign**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Application Manifest::Tasks		
Brief Description			
Description	Establish a 1:1 relation between a actual process and its placeholder during the design phase ProcessDesign.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Process	1..*	Process as input in order to link it to the respective ProcessDesign

Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Process Design	1..*	ProcessDesign as placeholder during design time for the real Process
Produces	Process	1..*	A Process references a respective ProcessDesign

Table 3.87: Associate Process with ProcessDesign

3.8.2 Work Products

3.8.2.1 Application Manifest

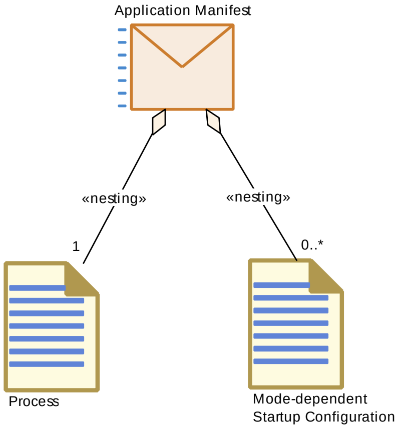

Figure 3.2: Structure of Deliverable Application Manifest

Deliverable	**Application Manifest**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Application Manifest::Work Products		
Brief Description	Definition of a process and all its properties		
Description	The application manifest defines the process with all its properties. It is defined for a specific machine by referencing its modes in the startup configuration. One application manifest is defined per process.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Process	1	The process is defined via the Application Manifest
Aggregates	Mode-dependent Startup Configuration	0..*	For each process the startup configuration can be defined in the Application Manifest
Produced by	Create Application Manifest	1..*	One application manifest per instantiated executable
Consumed by	Collect belonging (software) artifacts of Sub Software Clusters	0..*	Several processes can be deployed
Consumed by	Create Software Package	0..*	Several processes can be deployed

Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Identify necessary (software) artifacts	0..*	Several processes can be deployed
Consumed by	Set Up Initial Machine	0..*	All Application Manifests needed to run the desired adaptive application (instances or Processes) on a Machine

Table 3.88: Application Manifest

3.8.2.2 Process

Artifact	**Process**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Application Manifest::Work Products		
Brief Description	Instantiation of an executable		
Description	The process is the top-level element of the Application Manifest and references an executable. It is the unit of deployment on the AUTOSAR adaptive platform and refers to a POSIX process.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Associate Process with Process Design	1..*	A Process references a respective ProcessDesign
Produced by	Define Process	1..*	Different instantiation of executables can result in different processes.
Consumed by	Define Execution Dependencies	1	Execution dependencies defined for one process
Consumed by	Define Startup Configuration	1	Startup configuration to be defined for process
Consumed by	Map Process To Machine	1	Description of a dedicated Process
Consumed by	Associate Process with Process Design	1..*	Process as input in order to link it to the respective ProcessDesign
Consumed by	Define and configure machine	0..*	Processes dedicated to run Executables on a Machine

Table 3.89: Process

3.8.2.3 Mode-dependent Startup Configuration

Artifact	**Mode-dependent Startup Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Application Manifest::Work Products		
Brief Description	Startup configuration of a process		
Description	Startup configuration for one process and depending on the machine mode.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define Execution Dependencies	1..*	Execution dependencies of a process for each mode
Produced by	Define Startup Configuration	1..*	Startup configuration of a process for each mode

Table 3.90: Mode-dependent Startup Configuration

3.8.2.4 Process Design

Artifact	**Process Design**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Common Design Artifacts::Work Products		
Brief Description	Proxy for a Process at design time		
Description	This element stands in as a proxy for a Process at the time when it does not exist, yet, i.e., at design time, although the element Process is needed during runtime in order to distinguish different instances of Executables.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Associate DiagnosticMapping with Process Design	1..*	All dedicated ProssesDesigns for a Machine
Consumed by	Associate Process with Process Design	1..*	ProcessDesign as placeholder during design time for the real Process

Table 3.91: Process Design

3.9 Service Instance

This chapter contains the definition of work products and tasks necessary for instantiating the services.

3.9.1 Tasks

3.9.1.1 Configure Service Interface Deployment

Task Definition	**Configure Service Interface Deployment**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Tasks		
Brief Description	Configure the binding of a Service Interface to a transport layer		
Description	Define the transport layer (e.g. SOME/IP or User Defined) and configure the binding of a service interface to this transport layer. For all elements of the service interface, i.e., events, methods and fields, the deployment is configured. For SOME/IP, an identifier for the service interface is defined. This ID needs to be uniquely defined system-wide and is send as service ID in SOME/IP service discovery messages. In addition, message IDs and SOME/IP event groups for a logical grouping of events are defined. The IDs for messages and event groups need to be uniquely defined in the context of the enclosing SomeipServiceInterface. The User Defined service interface deployment can e.g. be used machine local IPC communication. The responsibility of the configuration of service interface deployment lies with the system responsible.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Service Interface Description	1	Deployment is configured for each service interface
Produces	Service Interface Deployment Configuration	1	Configuration of binding of a service interface to a transport layer

Table 3.92: Configure Service Interface Deployment

3.9.1.2 Define and Configure Service Instance

Task Definition	**Define and Configure Service Instance**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Tasks		
Brief Description	Define the service instances and configure their search or offer criteria		
Description	Define service instances. A service interface can be instantiated several times for different purposes resulting in several service instances. There can be provided service instances (server) if the functionality of a service interface is provided, and there can be required service instances (client) in case a service is required. Configure search criteria for required service instances and offer criteria for provided service instances. For search criteria in SOME/IP, the required service instance IDs and required service interface version needs to be defined. Also, required event groups can be specified. For offer criteria in SOME/IP, the provided service instance IDs need to defined. The instance IDs need to be defined system-wide.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Service Interface Deployment Configuration	1	Instances of service interfaces to be defined

Relation Type	***Related Element***	***Mul.***	***Note***
Produces	Service Instance Configuration	1..*	Service instances and their configuration defined

Table 3.93: Define and Configure Service Instance

3.9.1.3 Define SOME/IP timing

Task Definition	**Define SOME/IP Timing**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Tasks		
Brief Description	Define the timing for SOME/IP for the server and the client		
Description	Define SOME/IP timing for the server (SomeipSdServerServiceInstanceConfig, SomeipSdServerEventTimingConfig) and the client (SomeipSdClientServiceInstanceConfig, SomeipSdClientEventGroupTimingConfig). This task is optional and only necessary if communication via SOME/IP is used.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Service Instance Configuration	1	Timing for service instances to be defined
Produces	Service Instance Manifest	1	Timing for service instances contributes to Service Instance Manifest

Table 3.94: Define SOME/IP Timing

3.9.1.4 Map Service Instance to Port Prototype

Task Definition	**Map Service Instance to Port Prototype**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Tasks		
Brief Description	Define mapping of service instance to a port prototype		
Description	Map service instance to a software component port using the ServiceInstanceToPortPrototypeMapping. This mapping is needed in order to ensure a unique relationship between all local service instances within the application (represented by software component ports) and the service instances on the network (e.g. SOME/IP service instances).		
Relation Type	***Related Element***	***Mul.***	***Note***
Performed by	Tier 1	1	Software Integrator: This activity will probably be performed by a Software Integrator of a Tier 1 company
Consumes	Service Instance Configuration	1	Service instances to be mapped to port prototypes
Consumes	Software Component Description for Adaptive Platform	1	In case the service instances are mapped to ports of a software component

Relation Type	*Related Element*	*Mul.*	*Note*
Produces	Service Instance Manifest	1	Mapping contributes to Service Instance Manifest

Table 3.95: Map Service Instance to Port Prototype

3.9.1.5 Map Service Instance to Machine

Task Definition	**Map Service Instance to Machine**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Tasks		
Brief Description	Define mapping of service instance to machine		
Description	Map service instance to a machine via a communication connector using the ServiceInstanceToMachineMapping. This allows to configure the communication without any assumptions on the applications. For SOME/IP, IP and TP configuration for the client and the server are defined.		
Relation Type	*Related Element*	*Mul.*	*Note*
Consumes	Machine Design	1	Description of machine that the service instances shall be mapped to
Consumes	Service Instance Configuration	1	Service instances to be mapped to machine
Produces	Service Instance Manifest	1	Mapping contributes to Service Instance Manifest

Table 3.96: Map Service Instance to Machine

3.9.2 Work Products

3.9.2.1 Service Interface Deployment Configuration

Artifact	**Service Interface Deployment Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Work Products		
Brief Description	Deployment configuration for a service interface		
Description	Description of deployment configuration with respect to a transport layer for a service interface. For SOME/IP, service interface ID, message IDs and event groups are defined.		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Configure Service Interface Deployment	1	Configuration of binding of a service interface to a transport layer
Consumed by	Define and Configure Service Instance	1	Instances of service interfaces to be defined

Table 3.97: Service Interface Deployment Configuration

3.9.2.2 Service Instance Configuration

Artifact	**Service Instance Configuration**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Work Products		
Brief Description	Definition and configuration of the service instances		
Description	Required as well as provided service instances are defined and configured. For the configuration, the search criteria for required service instances and offer criteria for provided service instances are specified.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define and Configure Service Instance	1..*	Service instances and their configuration defined
Consumed by	Define SOME/IP Timing	1	Timing for service instances to be defined
Consumed by	Map Service Instance to Machine	1	Service instances to be mapped to machine
Consumed by	Map Service Instance to Port Prototype	1	Service instances to be mapped to port prototypes

Table 3.98: Service Instance Configuration

3.9.2.3 Service Instance Manifest

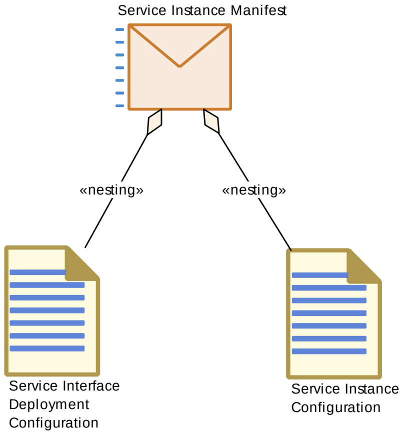
Service Instance Manifest
«nesting»
«nesting»
Service Interface Deployment Configuration
Service Instance Configuration

Figure 3.3: Parts of the Service Instance Manifest

Deliverable	**Service Instance Manifest**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Work Products		
Brief Description	Definition and configuration of a service instance		
Description	Definition of a service instance with its configuration for the service discovery. The mapping of the service instances to the machine is defined. Optionally, the mapping of service instances to the software component ports is specified.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Service Instance Configuration	1	
Aggregates	Service Interface Deployment Configuration	1	
Produced by	Define SOME/IP Timing	1	Timing for service instances contributes to Service Instance Manifest
Produced by	Map Service Instance to Machine	1	Mapping contributes to Service Instance Manifest
Produced by	Map Service Instance to Port Prototype	1	Mapping contributes to Service Instance Manifest
Produced by	Define and Configure Service Instances	1..*	Contains all configuration settings for the service instance on a specific machine
Consumed by	Collect belonging (software) artifacts of Sub Software Clusters	0..*	Several service instance manifests can be deployed
Consumed by	Create Software Package	0..*	Several service instance manifests can be deployed
Consumed by	Identify necessary (software) artifacts	0..*	Several service instance manifests can be deployed
Consumed by	Set Up Initial Machine	0..*	All Service Instance Manifests needed to run the desired adaptive application (instances or Processes) on a Machine

Table 3.99: Service Instance Manifest

3.10 Deployment

This chapter contains the definition of work products and tasks necessary for deploying Software Packages.

3.10.1 Tasks

3.10.1.1 Create an initial `Software Package Manifest`

Task Definition	**Create an initial Software Package Manifest**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Tasks		
Brief Description	Create an initial Software Package Manifest		
Description	The main input for this step are the requirements of the OEM given by means of the Software Cluster Design. This task is about to create an new Software Package Manifest and to transfer the structure and the entries of the given Software Cluster Design into the newly created Software Package Manifest.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Software Cluster Design	1	Requirements regarding Software Clusters by the OEM
Produces	Software Package Manifest	1	partially: Initial meta data of a respective Software Package

Table 3.100: Create an initial Software Package Manifest

3.10.1.2 Identify necessary (software) artifacts

Task Definition	**Identify necessary (software) artifacts**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Tasks		
Brief Description	Identify necessary artifacts		
Description	Identify necessary (software) artifacts in order to build the Software Package, also with respect to their versions. Check, whether there are divergences within the required and actual sets of Sub Software Clusters (by means of the aggregated artifacts and versions) , if necessary solve them and re-model the Software Package Manifest accordingly. Check, whether there are discrepancies between the required and actual set of the Root Software Cluster (by means of its aggregated Sub Software Clusters and versions)		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Diagnostic Machine Extract	0..1	Diagnostic extract for a Machine
Consumes	Software Cluster Design	0..1	Requirements that have initially been formulated by an OEM Here, not necessarily needed since the data is already available in Software Package Manifest
Consumes	Software Package Manifest	0..1	Meta data which are already transferred from Software Cluster Design
Consumes	Uploadable Design Artifacts	0..1	Optional input: Additional design data which are not part of an Application or Machine Manifest
Consumes	Application Manifest	0..*	Several processes can be deployed

Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Executable Application	0..*	Executables of deployed processes
Consumes	Service Instance Manifest	0..*	Several service instance manifests can be deployed
Produces	Software Package Manifest	1	Updates of the meta data after checks

Table 3.101: Identify necessary (software) artifacts

3.10.1.3 Collect belonging (software) artifacts of Sub Software Clusters

Task Definition	**Collect belonging (software) artifacts of Sub Software Clusters**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Tasks		
Brief Description	Collect belonging artifacts		
Description	Collect belonging (software) artifacts of Sub Software Clusters into separate baskets (Sub Software Cluster Group) in order to prepare the final step of creating the Software Package (Optional) Execute a receiving inspection of the software artifacts		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Software Package Manifest	1	Already consolidated meta data (after checks and re-modeling)
Consumes	Diagnostic Machine Extract	0..1	Diagnostic extract for a Machine
Consumes	Uploadable Design Artifacts	0..1	Optional input: Additional design data which are not part of an Application or Machine Manifest
Consumes	Application Manifest	0..*	Several processes can be deployed
Consumes	Executable Application	0..*	Executables of deployed processes
Consumes	Service Instance Manifest	0..*	Several service instance manifests can be deployed
Produces	(Sub) Software Cluster Group	0..*	Collection of corresponding artifacts (per Sub Software Cluster)

Table 3.102: Collect belonging (software) artifacts of Sub Software Clusters

3.10.1.4 Model dependencies between Software Clusters

Task Definition	**Model dependencies between Software Clusters of any category**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Tasks		
Brief Description	Model dependencies		
Description	Thus, this activity describes the handling of dependencies by at least the following tasks: • Check, whether the dependencies between Software Clusters of the same or different categories, given by the respective SoftwareClusterDesign are still valid • Determine changes between the actual and required dependencies between Software Clusters of any category • If necessary, re-model the Software Package Manifest in accordance with the outcomes of the both tasks above		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Software Package Manifest	1	Dependencies of the Software Package Manifest were transferred from the Software Cluster Design
Consumes	(Sub) Software Cluster Group	0..*	Optional source in order to check dependencies between Software Clusters (of any category)
Produces	Software Package Manifest	1	Re-modeled (consolidated) dependencies between Software Clusters of any category

Table 3.103: Model dependencies between Software Clusters of any category

3.10.1.5 Create installation instructions

Task Definition	**Create installation instructions**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Tasks		
Brief Description	Create installation instructions		
Description	Installation instruction control the behavior of the UCM during the update of Software Packages. Installation instructions can either be ’add/update’ meaning to install a package or ’remove’ to express that a package shall be uninstalled and deleted from the machine. Installation instructions are defined per Software Cluster, independent of its category. Thus, this activity may includes the tasks: • Specify installation instructions per Software Cluster (of any category) • Develop update campaigns (optional)		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Software Package Manifest	1	Software Package Manifest without or incomplete installation instructions

Relation Type	***Related Element***	***Mul.***	***Note***
Produces	Software Package Manifest	1	Software Package Manifest, enhanced by installation instruction

Table 3.104: Create installation instructions

3.10.2 Work Products

3.10.2.1 Software Cluster Design

Deliverable	**Software Cluster Design**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Work Products		
Brief Description	Software Cluster Design		
Description	The deliverable Software Cluster Design contains the requirements that have initially been formulated by an OEM. The formal structure of the corresponding meta model element SoftwareClusterDesign is similar to its counterpart SoftwareCluster. Thus, by means of this, the OEM is able to define the composition and structure of Software Clusters, dedicated diagnostic addresses as well as internal and external dependencies of Software Cluster.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Create Software Package	1	Requirements of the OEM wrt. package structure and parameters given by means of the meta model element SoftwareClusterDesign.
Consumed by	Create an initial Software Package Manifest	1	Requirements regarding Software Clusters by the OEM
Consumed by	Identify necessary (software) artifacts	0..1	Requirements that have initially been formulated by an OEM Here, not necessarily needed since the data is already available in Software Package Manifest
Use meta model element	SoftwareCluster Design	1	

Table 3.105: Software Cluster Design

3.10.2.2 Software Package

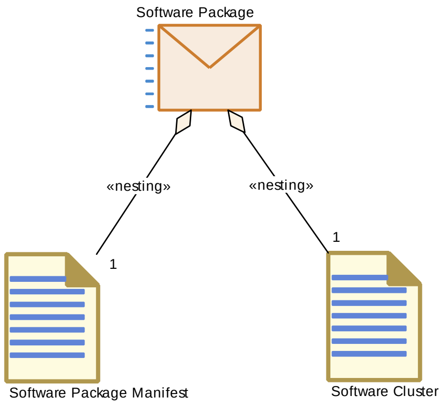

Figure 3.4: Parts of a `Software Package`

Deliverable	**Software Package**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Work Products		
Brief Description	Container to deploy software artifacts to a machine		
Description	According to the AUTOSAR glossary, Software Packages are the units for deployment onto machines (AUTOSAR Adaptive Platform instances). In this respect, they are inputs for and processed by the Adaptive Platform Service UCM} (Update and Configuration Management). In fact, a Software Package consists of two main parts: • a bundle of the actual software artifacts, referred to as Software Cluster • corresponding model data needed to control the upload and installation process of this Software Cluster executed by the UCM		
Kind	Custom		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Software Cluster	1	
Aggregates	Software Package Manifest	1	
Produced by	Compile the Software Package	1	Compiled Software Package
Produced by	Create Software Package	1	Software Package for deployment defined
Consumed by	Management of Software Packages	1..*	Newly created or updated Software Packages are stored into a repository and subject of the management of all available Software Packages (including their history)
Consumed by	Provide and manage Software Packages	1..*	Deploy software on a Back-end server by means of Software Package

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

Table 3.106: Software Package

3.10.2.3 Software Cluster

Artifact	**Software Cluster**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Work Products		
Brief Description	Software Cluster		
Description	Thus, from an UCM point of view, the term Software Cluster identifies a bundle of software artifacts that are uploaded together in order to be installed by the UCM. In general, a Software Cluster may contain Executable(s), Application Manifest(s), Service Instance Manifest(s), Machine Manifest(s) and other development artifacts. It should be mentioned, that a Software Cluster may be structured into sub-blocks in order to mimic the CP diagnostic workflow, where blocks are the smallest parts of update and to enable the execution of update campaigns. Otherwise, the term Software Cluster may also refer to a set of installed software entities (processes that run executables, data or manifests) which form a logical group and which are addressable by the diagnostic management by a shared diagnostic address. Not surprisingly, both definitions match in the sense that the bundle of software uploaded are needed to form the set of installed software entities addressed by the same diagnostic address.		
Kind	Custom		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Manage the data base of Software Clusters (of any category)	1..*	Store and manage software cluster within a repository

Table 3.107: Software Cluster

3.10.2.4 Software Package Manifest

Artifact	**Software Package Manifest**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Work Products		
Brief Description	Software Package Manifest		
Description	Model, based on meta model element SoftwareCluster, needed to control the upload and installation process of a Software Cluster executed by the UCM.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Create an initial Software Package Manifest	1	partially: Initial meta data of a respective Software Package
Produced by	Create installation instructions	1	Software Package Manifest, enhanced by installation instruction
Produced by	Identify necessary (software) artifacts	1	Updates of the meta data after checks
Produced by	Model dependencies between Software Clusters of any category	1	Re-modeled (consolidated) dependencies between Software Clusters of any category
Consumed by	Collect belonging (software) artifacts of Sub Software Clusters	1	Already consolidated meta data (after checks and re-modeling)
Consumed by	Compile the Software Package	1	Integrate the Software Package Manifest into the Software Package
Consumed by	Create installation instructions	1	Software Package Manifest without or incomplete installation instructions
Consumed by	Model dependencies between Software Clusters of any category	1	Dependencies of the Software Package Manifest were transferred from the Software Cluster Design
Consumed by	Manage the data base of Software Clusters (of any category)	1..*	Manage meta data of corresponding Software Cluster
Consumed by	Identify necessary (software) artifacts	0..1	Meta data which are already transferred from Software Cluster Design
Use meta model element	SoftwareCluster	1	

Table 3.108: Software Package Manifest

3.10.2.5 (Sub) Software Cluster Group

Deliverable	**(Sub) Software Cluster Group**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Work Products		
Brief Description	(Sub) Software Cluster Group		
Description	Basket to collect the (software) artifacts of a Sub Software Cluster		
Kind	Custom		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Collect belonging (software) artifacts of Sub Software Clusters	0..*	Collection of corresponding artifacts (per Sub Software Cluster)
Consumed by	Compile the Software Package	0..*	Compile all Sub Software Clusters into the Software Package

Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Model dependencies between Software Clusters of any category	0..*	Optional source in order to check dependencies between Software Clusters (of any category)

Table 3.109: (Sub) Software Cluster Group

3.10.2.6 Uploadable Design Artifacts

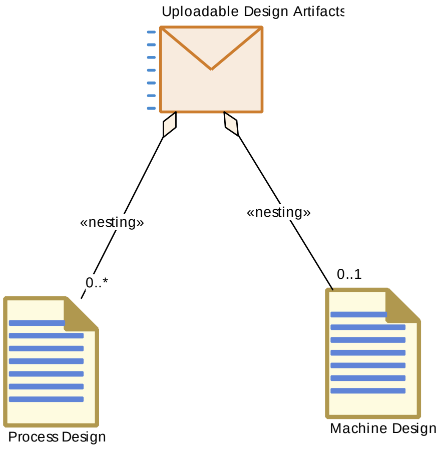

Figure 3.5: Design artifacts needed to be uploaded to the Machine

Deliverable	**Uploadable Design Artifacts**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Common Design Artifacts::Work Products		
Brief Description	Design artifacts needed needed to be uploaded to the Machine		
Description	Covers design artifacts, i.e., ’Machine Design’ and ’Process Design’, that are needed to be uploaded to the Machine in addition to the Manifests.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Machine Design	0..1	
Aggregates	Process Design	0..*	
Consumed by	Collect belonging (software) artifacts of Sub Software Clusters	0..1	Optional input: Additional design data which are not part of an Application or Machine Manifest
Consumed by	Create Software Package	0..1	Optional input: Additional design data which are not part of an Application or Machine Manifest
Consumed by	Identify necessary (software) artifacts	0..1	Optional input: Additional design data which are not part of an Application or Machine Manifest
Consumed by	Set Up Initial Machine	0..1	Optional input: Additional design data which are not part of an Application or Machine Manifest

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

Table 3.110: Uploadable Design Artifacts

3.10.2.7 Back-end server

Deliverable	**Back-end Server**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Work Products		
Brief Description	Repository of uploadable packages on a Back-end server		
Description	Repository of uploadable packages (Software Packages) including corresponding data bases and server programs in order to provide dedicated versions, change sets and the like to the Machines (Adaptive ECUs) in the field.		
Kind	Custom		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Provision of Software Packages for machines in the field	1	Organize the Back-end Server in accordance with the requirements of an OEM
Produced by	Management of Software Packages	1..*	Software Packages are stored into a repository of Software Packages. In addition, update of a common data base of available Software Packages including their history.
Produced by	Provide and manage Software Packages	1..*	Store uploadable packages (Software Packages) into a repository of a Back-end server
Consumed by	Provision of Software Packages for machines in the field	1	Status quo of the presentation layer of the Back-end Server

Table 3.111: Back-end Server

A Change History

A.1 Change History for AP 18-03

A.1.1 Added Specification Items in AP 18-03

Number	Heading
[TR_AMETH_00211]	Pool Executables together to form ExecutableGroups
[TR_AMETH_00212]	Design a diagnostic mapping
[TR_AMETH_00213]	Relate diagnostic mappings to instances of Executables
[TR_AMETH_00214]	Configuration of Platform Services
[TR_AMETH_00215]	Configuration of Platform Foundation Modules
[TR_AMETH_00216]	Map Processes to a particular machine
[TR_AMETH_00217]	Definition of resources
[TR_AMETH_00218]	Create an initial Software Package Manifest
[TR_AMETH_00219]	Collect all software artifacts that belong to a Software Cluster, structure and model them
[TR_AMETH_00220]	Model dependencies between Software Clusters of any category
[TR_AMETH_00221]	Develop installation instructions
[TR_AMETH_00222]	Create the Software Package
[TR_AMETH_00223]	Manage the data base of Software Clusters (of any category)
[TR_AMETH_00224]	Management of Software Packages
[TR_AMETH_00225]	Provision of Software Packages for machines in the field
[TR_AMETH_00226]	Documentation of work products

Table A.1: Added specification items in AP 18-03

A.1.2 Changed Specification Items in AP 18-03

Number	Heading
[TR_AMETH_00205]	Integrate Software
[TR_AMETH_00206]	Create a Software Package
[TR_AMETH_00021]	Configuration of network communication for machine
[TR_AMETH_00208]	Map a single ServiceInterface to PortInterface elements
[TR_AMETH_00031]	Setting up an initial machine
[TR_AMETH_00022]	Definition of machine states, function group states and per-state timeouts

Table A.2: Changed specification items in AP 18-03

A.1.3 Deleted Specification Items in AP 18-03

Number	Heading
TR_AMETH_00032	Deploying the Software Package

Table A.3: Deleted specification items in AP 18-03

A.2 Change History for AP 17-10

A.2.1 Added Specification Items in AP 17-10

Number	Heading
[TR_AMETH_00200]	Domains of development utilized for the methodology of the AUTOSAR Adaptive Platform
[TR_AMETH_00201]	Develop a Function Architecture
[TR_AMETH_00202]	Develop a Common Software Architecture
[TR_AMETH_00203]	Provide views of subsystems
[TR_AMETH_00204]	Develop the System
[TR_AMETH_00205]	Integrate Software to form AdaptiveAutosarApplications
[TR_AMETH_00206]	Create SoftwareCluster
[TR_AMETH_00207]	Design communication between Classic Platform ECUs and Adaptive Platform machines
[TR_AMETH_00208]	Map a single ServiceInterface to PortInterface elements
[TR_AMETH_00209]	Define a signal-based ServiceInterface
[TR_AMETH_00210]	Map signals to services

Table A.4: Added specification items in AP 17-10

A.2.2 Changed Specification Items in AP 17-10

Number	Heading
[TR_AMETH_00100]	Scope of the Methodology for the Adaptive Platform
[TR_AMETH_00101]	Definition of tasks, work products and use cases
[TR_AMETH_00102]	Types of work products
[TR_AMETH_00001]	Description of the services in a system
[TR_AMETH_00002]	Development of the software
[TR_AMETH_00006]	Deployment of the application software
TR_AMETH_00032	Deploying the Software Package
[TR_AMETH_00033]	Mapping of Service Instances to Port Prototypes

Table A.5: Changed specification items in AP 17-10

A.2.3 Deleted Specification Items in AP 17-10

Number	Heading
TR_AMETH_00030	Machine-driven and model-driven approach

Table A.6: Deleted specification items in AP 17-10

A.3 Change History for AP 17-03

A.3.1 Added Specification Items in AP 17-03

Number	Heading
[TR_AMETH_00100]	Scope of the Methodology for the Adaptive Platform
[TR_AMETH_00101]	Definition of tasks, work products and use cases

[TR_AMETH_00102]	Types of work products
[TR_AMETH_00001]	Description of the services in a system
[TR_AMETH_00002]	Development of the software
[TR_AMETH_00003]	Configuration of the machine
[TR_AMETH_00004]	Creation of the Application Manifest
[TR_AMETH_00005]	Configuration of the service instances
[TR_AMETH_00006]	Deployment of the application software
[TR_AMETH_00007]	Definition of data types for the Adaptive Platform
[TR_AMETH_00008]	Definition of service interfaces for the Adaptive Platform
[TR_AMETH_00009]	Aggregating service interfaces for reducing the bus load
[TR_AMETH_00010]	Application-level Software
[TR_AMETH_00011]	Design of the software components
[TR_AMETH_00012]	Generation of the header files for service interface
[TR_AMETH_00013]	Implementation and compilation of software components
[TR_AMETH_00014]	Development with knowledge of the Build Chain Configuration
[TR_AMETH_00015]	Development without knowledge of the Build Chain Configuration
[TR_AMETH_00016]	Development of serialization properties
[TR_AMETH_00017]	Implementation of service proxies and skeletons
[TR_AMETH_00018]	Building the Executable Application
[TR_AMETH_00019]	Description of the Adaptive Platform
[TR_AMETH_00020]	Development of Platform Software
[TR_AMETH_00021]	Configuration of network communication for machine
[TR_AMETH_00022]	Definition of machine states and resources
[TR_AMETH_00023]	Configuration of the operating system
[TR_AMETH_00024]	Instantiation of Executable Application
[TR_AMETH_00025]	Defintion of startup behavior of a process
[TR_AMETH_00026]	Defintion of Application Manifest
[TR_AMETH_00027]	Configuration of Service Interface Deployment
[TR_AMETH_00028]	Configuration of Service Instances
[TR_AMETH_00029]	Deployment of Service Instances
TR_AMETH_00030	Machine-driven and model-driven approach
[TR_AMETH_00031]	Setting up the machine
TR_AMETH_00032	Deploying the Software Package
[TR_AMETH_00033]	Mapping of Service Instances to Application Endpoints
[TR_AMETH_00034]	Selecting the Operating System for Adaptive Platform
[TR_AMETH_00035]	Platform-level Software

Table A.7: Added specification items in AP 17-03

A.3.2 Changed Specification Items in AP 17-03

N/A

A.3.3 Deleted Specification Items in AP 17-03

N/A

B Used classes in Manifest files

B.1 Used classes in Machine Manifest

Used classes	Base
AdaptiveModuleInstantiation	other
CommunicationConnector	other
CryptoDriver	PackageableElement
CryptoDriverToCryptoJobMapping	other
CryptoJob	other
CryptoKeySlot	other
CryptoModuleInstantiation	other
CryptoNeedToCryptoJobMapping	other
CryptoPrimitive	other
DoIpInstantiation	other
EnterExitTimeout	other
EthernetCluster	PackageableElement
EthernetCommunicationConnector	other
EthernetNetworkConfiguration	other
EthernetPhysicalChannel	other
GenericModuleInstantiation	other
LogAndTraceInstantiation	other
MacMulticastGroup	other
Machine	PackageableElement
MachineDesign	PackageableElement
ModeDeclaration	other
ModeDeclarationGroup	PackageableElement
ModeDeclarationGroupPrototype	other
NetworkConfiguration	other
NetworkEndpoint	other
NetworkEndpointAddress	other
NmCluster	other
NmConfig	PackageableElement
NmInstantiation	other
NmNode	other
NonOsModuleInstantiation	other
OsModuleInstantiation	other
PerStateTimeout	other
Processor	other
ProcessorCore	other
PskIdentityToKeySlotMapping	other
PureLocalTimeBase	other
ResourceGroup	other
SecOcDeployment	other
SecOcJobMapping	other
SecureCommunicationDeployment	other
ServiceDiscoveryConfiguration	other
SomeipServiceDiscovery	other
SynchronizedMasterTimeBase	other
SynchronizedSlaveTimeBase	other
TimeBaseResource	other
TimeSyncModuleInstantiation	other

TlsDeployment	other
TlsJobMapping	other
UdpNmCluster	other
UdpNmNode	other

Table B.1: Used classes in MachineManifest

B.2 Used classes in Application Manifest

Used classes	Base
Action	other
ActionItem	other
ActionList	other
AliveSupervision	other
ApplicationActionItem	other
Arbitration	other
CheckpointTransition	other
DeadlineSupervision	other
ExecutionDependency	other
GlobalSupervision	other
HealthChannel	other
HealthChannelExternalStatus	other
HealthChannelSupervision	other
HttpAcceptEncoding	other
LocalSupervision	other
LogicalExpression	other
LogicalSupervision	other
ModeDeclaration	other
ModeDeclarationGroup	PackageableElement
ModeDeclarationGroupPrototype	other
ModeDependentStartupConfig	other
PersistencyFile	PackageableElement
PersistencyFileArray	PackageableElement
PersistencyKeyValueDatabase	PackageableElement
PersistencyKeyValuePair	other
PersistencyPortPrototypeToFileArrayMapping	PackageableElement
PersistencyPortPrototypeToKeyValueDatabaseMapping	PackageableElement
PhmContributionToMachineMapping	PackageableElement
PlatformActionItem	other
PlatformHealthManagementContribution	PackageableElement
Process	PackageableElement
ProcessToMachineMapping	other
ProcessToMachineMappingSet	PackageableElement
RestHttpPortPrototypeMapping	PackageableElement
Rule	other
ServiceInstanceToPortPrototypeMapping	PackageableElement
StartupConfig	other
StartupConfigSet	PackageableElement
StartupOption	other
SupervisionCheckpoint	other
WatchdogActionItem	other

Table B.2: Used classes in ApplicationManifest

B.3 Used classes in Service Instance Manifest

Used classes	Base
AdaptivePlatformServiceInstance	PackageableElement
DdsEventDeployment	other
DdsServiceInstanceToMachineMapping	PackageableElement
DdsServiceInterfaceDeployment	PackageableElement
E2EProfileConfiguration	other
E2EProfileConfigurationSet	PackageableElement
End2EndEventProtectionProps	other
InitialSdDelayConfig	other
PresharedKeyIdentity	other
ProvidedApServiceInstance	PackageableElement
ProvidedDdsEventQosProps	other
ProvidedDdsServiceInstance	PackageableElement
ProvidedSomeipServiceInstance	PackageableElement
ProvidedUserDefinedServiceInstance	PackageableElement
RequestResponseDelay	other
RequiredApServiceInstance	PackageableElement
RequiredDdsEventQosProps	other
RequiredDdsServiceInstance	PackageableElement
RequiredSomeipServiceInstance	PackageableElement
RequiredUserDefinedServiceInstance	PackageableElement
SecOcJobRequirement	other
SecOcSecureComProps	other
SecureComProps	other
SecureComPropsSet	PackageableElement
ServiceEventDeployment	other
ServiceFieldDeployment	other
ServiceInstanceToMachineMapping	PackageableElement
ServiceInterfaceDeployment	PackageableElement
ServiceInterfaceElementSecureComConfig	other
ServiceMethodDeployment	other
SomeipEventDeployment	other
SomeipEventGroup	other
SomeipEventProps	other
SomeipFieldDeployment	other
SomeipMethodDeployment	other
SomeipMethodProps	other
SomeipProvidedEventGroup	other
SomeipRequiredEventGroup	other
SomeipSdClientEventGroupTimingConfig	other
SomeipSdClientServiceInstanceConfig	other
SomeipSdServerEventTimingConfig	other
SomeipSdServerServiceInstanceConfig	other
SomeipServiceInstanceToMachineMapping	PackageableElement
SomeipServiceInterfaceDeployment	PackageableElement
SomeipServiceInterfaceVersion	other
SomeipTimingProps	other
TagWithOptionalValue	other
TlsCipherSuite	other
TlsJobRequirement	other
TlsSecureComProps	other

UserDefinedEventDeployment	other
UserDefinedFieldDeployment	other
UserDefinedMethodDeployment	other
UserDefinedServiceInstanceToMachineMapping	PackageableElement
UserDefinedServiceInterfaceDeployment	PackageableElement

Table B.3: Used classes in ServiceInstanceManifest