

<b>Document Title</b>	Methodology for Adaptive Platform
<b>Document Owner</b>	AUTOSAR
<b>Document Responsibility</b>	AUTOSAR
<b>Document Identification No</b>	709

<b>Document Status</b>	Final
<b>Part of AUTOSAR Standard</b>	Adaptive Platform
<b>Part of Standard Release</b>	19-03

<b>Document Change History</b>			
<b>Date</b>	<b>Release</b>	<b>Changed by</b>	<b>Description</b>
2019-03-29	19-03	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>No content changes</li> </ul>
2018-10-31	18-10	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>renamed <code>Application Manifest</code> to <code>Execution Manifest</code></li> <li>moved references from <code>spec.item</code> body to foot notes</li> <li>editorial changes</li> </ul>
2018-03-29	18-03	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>Split of machine design and machine configuration</li> <li>Added diagnostic mapping</li> <li>Added roles</li> <li>Reviewed sections about deployment of Software Packages</li> </ul>
2017-10-27	17-10	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>Design of service oriented communication between CP and AP</li> <li>Design of signal oriented communication between CP and AP</li> <li>Deployment by means of <code>SoftwareCluster</code></li> <li>Removed concept of <code>TransportLayerIndependentInstanceId</code></li> </ul>
2017-03-31	17-03	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>Initial release</li> </ul>

## 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

1	Introduction	10
1.1	Objective and Scope	10
1.2	Document Outline	11
1.3	Document Conventions	11
1.4	Abbreviations	11
1.5	Methodology Concepts	13
1.6	Requirements Traceability	13
1.7	Known Limitations	16
2	Use Cases for the Adaptive Platform	17
2.1	Overall View	17
2.1.1	Purpose	17
2.1.2	Description	17
2.1.2.1	Domains of Development	17
2.1.2.2	Fundamental Activities	18
2.1.2.3	Workflow	25
2.2	Architecture and Design	28
2.2.1	Develop a Service Interface Description	28
2.2.1.1	Purpose	28
2.2.1.2	Description	28
2.2.1.3	Workflow	29
2.2.2	Design communication between Classic Platform and Adaptive Platform	30
2.2.2.1	Design service oriented communication between Classic Platform and Adaptive Platform	30
2.2.2.2	Design signal oriented communication between Classic Platform and Adaptive Platform	33
2.2.3	Develop the communication structure by means of Machine Design	37
2.2.3.1	Purpose	37
2.2.3.2	Description	37
2.2.3.3	Workflow	38
2.2.4	Create a Diagnostic Mapping	39
2.2.4.1	Purpose	39
2.2.4.2	Description	40
2.2.4.3	Workflow	41
2.3	Software Development	43
2.3.1	Develop Adaptive Application Software	43
2.3.1.1	Purpose	43
2.3.1.2	Description	43
2.3.1.3	Workflow	44
2.3.2	Develop Adaptive Platform-level Software	47
2.3.2.1	Purpose	47
2.3.2.2	Description	47

2.3.2.3	Workflow	47
2.4	Integration and Deployment	48
2.4.1	Integrate Software	48
2.4.1.1	Purpose	48
2.4.1.2	Description	49
2.4.1.3	Workflow	50
2.4.2	Define and configure a Machine	52
2.4.2.1	Preparatory steps	53
2.4.2.2	Configure the Machine	54
2.4.3	Create Execution Manifest	58
2.4.3.1	Purpose	58
2.4.3.2	Description	59
2.4.3.3	Workflow	59
2.4.4	Define and Configure Service Instances	61
2.4.4.1	Purpose	61
2.4.4.2	Description	61
2.4.4.3	Workflow	63
2.4.5	Set up an initial Machine	65
2.4.5.1	Purpose	65
2.4.5.2	Description	66
2.4.5.3	Workflow	67
2.4.6	Create Software Packages	68
2.4.6.1	Purpose	68
2.4.6.2	Description	68
2.4.6.3	Workflow	73
2.4.7	Management and provision of Software Packages	76
2.4.7.1	Purpose	76
2.4.7.2	Description	76
2.4.7.3	Workflow	77
3	Adaptive Methodology Library	80
3.1	Roles	80
3.1.1	OEM	80
3.1.2	Tier 1	81
3.1.3	Tier 2	81
3.2	Service Interface	82
3.2.1	Tasks	82
3.2.1.1	Provide Data Types for Adaptive Platform	82
3.2.1.2	Define Service Interfaces	83
3.2.1.3	Aggregate Service Interfaces	83
3.2.2	Work Products	83
3.2.2.1	AUTOSAR AP Standard Package	83
3.2.2.2	AP Data Types	84
3.2.2.3	Service Interface Description	84
3.2.2.4	Service Interface Mapping	86
3.3	Communication Mapping	87

3.3.1	Tasks	87
3.3.1.1	Map Method	87
3.3.1.2	Map Event	87
3.3.1.3	Map Field	88
3.3.1.4	Map Fire and Forget	88
3.3.1.5	Map SignalBasedMethod to ISignalTriggerings	89
3.3.1.6	Map SignalBasedEvent to ISignalTriggerings	89
3.3.1.7	Map SignalBasedField to ISignalTriggerings	90
3.3.1.8	Map ServiceInstance to PortPrototype	90
3.3.2	Work Products	91
3.3.2.1	Client Server Interface Description	91
3.3.2.2	Sender Receiver Interface Description	91
3.3.2.3	Trigger Interface Description	92
3.3.2.4	Service Interface Mapping Set	92
3.3.2.5	Service Interface Mapping for Service Oriented Communication	93
3.3.2.6	System Description	93
3.3.2.7	Service Instance To Signal Mapping Set	95
3.3.2.8	Service Instance To Signal Mapping	96
3.4	Machine Design	97
3.4.1	Tasks	97
3.4.1.1	Define and configure the network connections of a Machine	97
3.4.1.2	Configure the Service Discovery Message Exchange	97
3.4.2	Work Products	97
3.4.2.1	Machine Design	97
3.5	Diagnostic Mapping	98
3.5.1	Tasks	98
3.5.1.1	Map Diagnostic Data	98
3.5.1.2	Map Diagnostic Enable Condition to Ports	99
3.5.1.3	Map Diagnostic Event to Ports	99
3.5.1.4	Map Diagnostic Storage Condition to Ports	100
3.5.1.5	Map Diagnostic Software Mapping	100
3.5.1.6	Map Diagnostic Operation Cycle to Ports	101
3.5.1.7	Associate a DiagnosticMapping with a ProcessDesign	101
3.5.2	Work Products	102
3.5.2.1	Diagnostic Machine Extract	103
3.5.2.2	DID	104
3.5.2.3	Diagnostic Enable Condition	104
3.5.2.4	Diagnostic Event	104
3.5.2.5	Diagnostic Mapping	105
3.5.2.6	Diagnostic Operation Cycle	106
3.5.2.7	Diagnostic Storage Condition	106
3.6	Adaptive Application	106
3.6.1	Tasks	106
3.6.1.1	Generate Header Files for Service Interfaces	106

3.6.1.2	Design Software Component for Adaptive Platform . . . . .	107
3.6.1.3	Implement Software Component Functionality . . . . .	107
3.6.1.4	Compile Software Component . . . . .	108
3.6.1.5	Develop Main Function . . . . .	109
3.6.1.6	Configure Serialization for Adaptive Platform . . . . .	109
3.6.1.7	Generate Serialization Code for Adaptive Platform . . . . .	109
3.6.1.8	Implement Service Proxies and Skeletons . . . . .	110
3.6.1.9	Build Executable Application . . . . .	110
3.6.2	Work Products . . . . .	111
3.6.2.1	Header Files for Service Interfaces . . . . .	111
3.6.2.2	Software Component Description for Adaptive Platform . . . . .	111
3.6.2.3	Build Chain Configuration . . . . .	112
3.6.2.4	Software Component Source Code . . . . .	113
3.6.2.5	Software Component Object Code . . . . .	113
3.6.2.6	Serialization Configuration for Adaptive Platform . . . . .	114
3.6.2.7	Serialization Source Code . . . . .	114
3.6.2.8	Implemented Service Proxies and Skeletons . . . . .	115
3.6.2.9	Main Function . . . . .	115
3.6.2.10	Executable Application . . . . .	116
3.7	Platform and Machine . . . . .	117
3.7.1	Tasks . . . . .	117
3.7.1.1	Define ECU Description . . . . .	117
3.7.1.2	Describe Available HW Resources . . . . .	117
3.7.1.3	Define Machine States . . . . .	118
3.7.1.4	Define Function Groups . . . . .	118
3.7.1.5	Define State Timeouts . . . . .	118
3.7.1.6	Map Process To Machine . . . . .	119
3.7.1.7	Configure OS for Adaptive Platform . . . . .	119
3.7.1.8	Configure Log and Trace module . . . . .	119
3.7.1.9	Configure DoLP . . . . .	120
3.7.1.10	Configure NM module . . . . .	120
3.7.2	Work Products . . . . .	120
3.7.2.1	Middleware Library Header Files . . . . .	120
3.7.2.2	Middleware Libraries . . . . .	121
3.7.2.3	ECU Resources Description . . . . .	121
3.7.2.4	Configured Machine on Adaptive ECU . . . . .	122
3.7.2.5	Machine Manifest . . . . .	123
3.7.2.6	Platform Object Code . . . . .	123
3.7.2.7	Operating System for Adaptive Platform . . . . .	124
3.7.2.8	Process to Machine Mapping . . . . .	124
3.7.2.9	Function Groups . . . . .	125
3.7.2.10	Machine States . . . . .	125
3.7.2.11	PerState Timeouts . . . . .	126
3.8	Execution Manifest . . . . .	126
3.8.1	Tasks . . . . .	126
3.8.1.1	Define Process . . . . .	126

3.8.1.2	Define Startup Configuration . . . . .	126
3.8.1.3	Define Execution Dependencies . . . . .	127
3.8.1.4	Associate Process with Process Design . . . . .	127
3.8.2	Work Products . . . . .	128
3.8.2.1	Execution Manifest . . . . .	128
3.8.2.2	Process . . . . .	129
3.8.2.3	Mode-dependent Startup Configuration . . . . .	129
3.8.2.4	Process Design . . . . .	130
3.9	Service Instance . . . . .	130
3.9.1	Tasks . . . . .	130
3.9.1.1	Configure Service Interface Deployment . . . . .	130
3.9.1.2	Define and Configure Service Instance . . . . .	131
3.9.1.3	Define SOME/IP timing . . . . .	132
3.9.1.4	Map Service Instance to Port Prototype . . . . .	132
3.9.1.5	Map Service Instance to Machine . . . . .	133
3.9.2	Work Products . . . . .	133
3.9.2.1	Service Interface Deployment Configuration . . . . .	133
3.9.2.2	Service Instance Configuration . . . . .	134
3.9.2.3	Service Instance To Machine Mapping . . . . .	134
3.9.2.4	Service Instance To Port Prototype Mapping . . . . .	134
3.9.2.5	Service Instance Manifest . . . . .	135
3.10	Deployment . . . . .	136
3.10.1	Tasks . . . . .	136
3.10.1.1	Create an initial Software Package Manifest . . . . .	136
3.10.1.2	Identify necessary (software) artifacts . . . . .	137
3.10.1.3	Collect belonging (software) artifacts of Sub Software Clusters . . . . .	138
3.10.1.4	Model dependencies between Software Clusters . . . . .	138
3.10.1.5	Create installation instructions . . . . .	139
3.10.2	Work Products . . . . .	140
3.10.2.1	Software Cluster Design . . . . .	140
3.10.2.2	Software Package . . . . .	141
3.10.2.3	Software Cluster . . . . .	142
3.10.2.4	Software Package Manifest . . . . .	142
3.10.2.5	(Sub) Software Cluster Group . . . . .	143
3.10.2.6	Uploadable Design Artifacts . . . . .	144
3.10.2.7	Back-end server . . . . .	145
A	Change History . . . . .	146
A.1	Change History for AP 19-03 . . . . .	146
A.1.1	Added Constraints in 19-03 . . . . .	146
A.1.2	Changed Constraints in 19-03 . . . . .	146
A.1.3	Deleted Constraints in 19-03 . . . . .	146
A.1.4	Added Traceables in 19-03 . . . . .	146
A.1.5	Changed Traceables in 19-03 . . . . .	146
A.1.6	Deleted Traceables in 19-03 . . . . .	146

A.2	Change History for AP 18-10	146
A.2.1	Added Constraints in 18-10	146
A.2.2	Changed Constraints in 18-10	146
A.2.3	Deleted Constraints in 18-10	147
A.2.4	Added Traceables in 18-10	147
A.2.5	Changed Traceables in 18-10	147
A.2.6	Deleted Traceables in 18-10	147
A.3	Change History for AP 18-03	147
A.3.1	Added Specification Items in AP 18-03	147
A.3.2	Changed Specification Items in AP 18-03	148
A.3.3	Deleted Specification Items in AP 18-03	148
A.4	Change History for AP 17-10	148
A.4.1	Added Specification Items in AP 17-10	148
A.4.2	Changed Specification Items in AP 17-10	148
A.4.3	Deleted Specification Items in AP 17-10	149
A.5	Change History for AP 17-03	149
A.5.1	Added Specification Items in AP 17-03	149
A.5.2	Changed Specification Items in AP 17-03	150
A.5.3	Deleted Specification Items in AP 17-03	150
B	Used classes in Manifest files	151
B.1	Used classes in Machine Manifest	151
B.2	Used classes in Execution Manifest	152
B.3	Used classes in Service Instance Manifest	153



## References

- [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.

<i>Abbreviation</i>	<i>Meaning</i>
ABI	Application Binary Interface
AP	AUTOSAR Adaptive Platform
API	Application Programming Interface



△

<i>Abbreviation</i>	<i>Meaning</i>
ARXML	AUTOSAR XML
CP	AUTOSAR Classic Platform
DoIP	Diagnostics over IP
DM	Diagnostic Manager
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 `activities`, entities to aggregate `tasks` and their corresponding `work products`. Tasks are defined as reusable elements: input information (e.g., stored within particular `work products`) is processed in order to generate new `work products`. ]([RS\\_METH\\_00018](#))<sup>1</sup>

**[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]
[RS_METH_00020]	The methodology shall support round-trip engineering	[TR_AMETH_00100]

<sup>1</sup>This document describes use cases in Section 2, `tasks` and `work products` in Section 3.

[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	<a href="#">[TR_AMETH_00002]</a> <a href="#">[TR_AMETH_00010]</a> <a href="#">[TR_AMETH_00011]</a> <a href="#">[TR_AMETH_00012]</a> <a href="#">[TR_AMETH_00013]</a> <a href="#">[TR_AMETH_00014]</a> <a href="#">[TR_AMETH_00015]</a> <a href="#">[TR_AMETH_00018]</a> <a href="#">[TR_AMETH_00205]</a> <a href="#">[TR_AMETH_00207]</a> <a href="#">[TR_AMETH_00208]</a> <a href="#">[TR_AMETH_00209]</a> <a href="#">[TR_AMETH_00210]</a>
[RS_METH_00203]	The methodology shall explain the high-level usage of the Manifest Specification	<a href="#">[TR_AMETH_00003]</a> <a href="#">[TR_AMETH_00004]</a> <a href="#">[TR_AMETH_00005]</a> <a href="#">[TR_AMETH_00021]</a> <a href="#">[TR_AMETH_00022]</a> <a href="#">[TR_AMETH_00023]</a> <a href="#">[TR_AMETH_00024]</a> <a href="#">[TR_AMETH_00025]</a> <a href="#">[TR_AMETH_00026]</a> <a href="#">[TR_AMETH_00027]</a> <a href="#">[TR_AMETH_00028]</a> <a href="#">[TR_AMETH_00029]</a> <a href="#">[TR_AMETH_00033]</a> <a href="#">[TR_AMETH_00214]</a> <a href="#">[TR_AMETH_00215]</a> <a href="#">[TR_AMETH_00216]</a> <a href="#">[TR_AMETH_00217]</a>
[RS_METH_00204]	The methodology shall describe how to configure a machine for the Adaptive Platform	<a href="#">[TR_AMETH_00003]</a> <a href="#">[TR_AMETH_00021]</a> <a href="#">[TR_AMETH_00022]</a> <a href="#">[TR_AMETH_00023]</a> <a href="#">[TR_AMETH_00031]</a> <a href="#">[TR_AMETH_00214]</a> <a href="#">[TR_AMETH_00215]</a> <a href="#">[TR_AMETH_00216]</a> <a href="#">[TR_AMETH_00217]</a>
[RS_METH_00205]	The methodology shall describe how to deploy software on the Adaptive Platform	<a href="#">[TR_AMETH_00006]</a> <a href="#">[TR_AMETH_00031]</a> <a href="#">[TR_AMETH_00206]</a>
[RS_METH_00206]	The methodology shall explain how to configure the instances of services of a system	<a href="#">[TR_AMETH_00005]</a> <a href="#">[TR_AMETH_00027]</a> <a href="#">[TR_AMETH_00028]</a> <a href="#">[TR_AMETH_00029]</a> <a href="#">[TR_AMETH_00033]</a>
[RS_METH_00207]	The methodology shall explain how to develop Platform Software for the Adaptive Platform	<a href="#">[TR_AMETH_00017]</a> <a href="#">[TR_AMETH_00019]</a> <a href="#">[TR_AMETH_00020]</a> <a href="#">[TR_AMETH_00034]</a> <a href="#">[TR_AMETH_00035]</a> <a href="#">[TR_AMETH_00212]</a> <a href="#">[TR_AMETH_00213]</a>

## 1.7 Known Limitations

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](#))

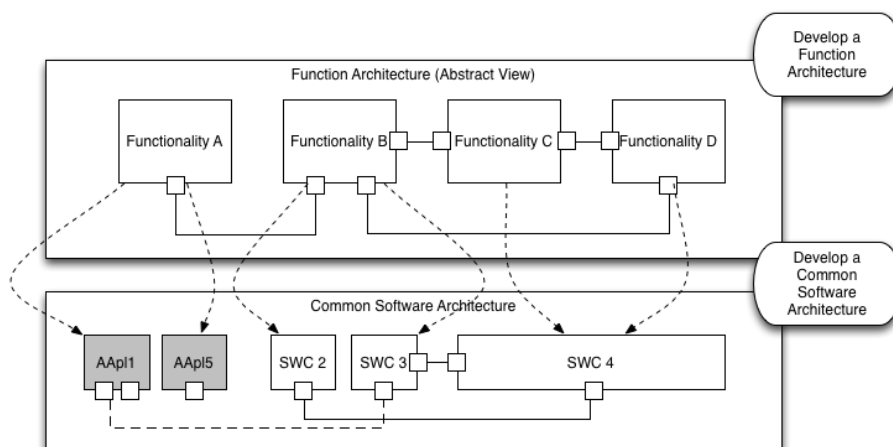
## 2.1.2.2 Fundamental Activities

### 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



**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.

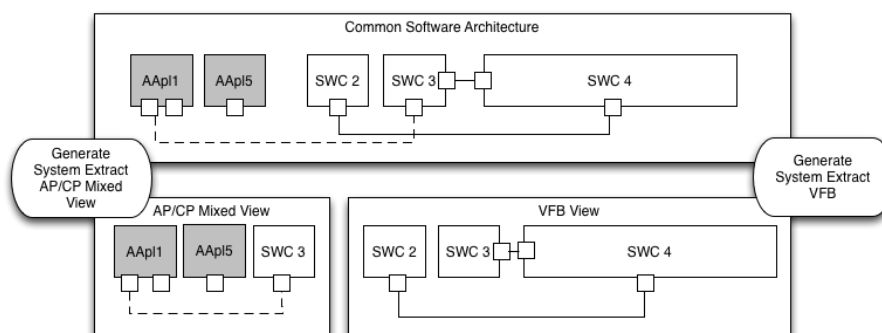
The Common Software Architecture provides a dedicated view of all software entities and their communication relation within the E/E vehicle system. In this light, the Common Software Architecture comprises both types, AUTOSAR software components of the Classic Platform as well as those entities that 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.

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](#))<sup>1</sup>



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

<sup>1</sup> 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.

**[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.

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 (namely AUTOSAR Adaptive Platform or AUTOSAR Classic Platform).

This activity is optional. ]([RS\\_METH\\_00078](#), [RS\\_METH\\_00079](#))<sup>2</sup>

**[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. ]([RS\\_METH\\_00201](#), [RS\\_METH\\_00032](#))<sup>3</sup>

**[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.

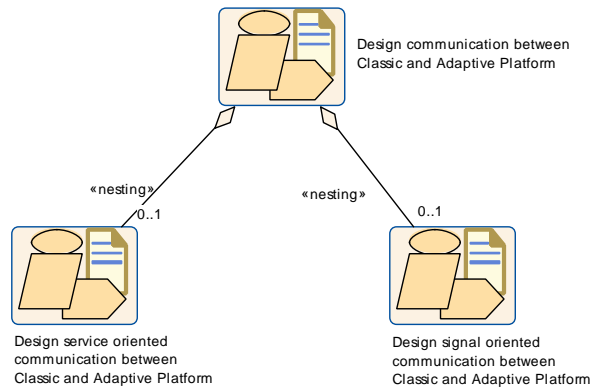
If the counterpart on a Classic Platform ECU, however, communicates only in a signal-based way, a `Signal-to-Service` translation is needed. ]([RS\\_METH\\_00202](#))<sup>4</sup>

---

<sup>2</sup>Figure 2.2 shows two possible views on subsystems deduced from the `Common Software Architecture`.

<sup>3</sup>This use case is elaborated in section 2.2.1.

<sup>4</sup>The use case of `SOME/IP` communication is elaborated in section 2.2.2.1. The use case of `Signal-to-Service` translation is elaborated in section 2.2.2.2.



**Figure 2.3: Design Communication between Classic Platform and Adaptive Platform**

<b>Activity</b>	<b>Design communication between Classic and Adaptive Platform</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Communication		
<b>Brief Description</b>	Design communication between CP and AP		
<b>Description</b>	Higher level activity that encloses all activities which are necessary to design communication between a Classic Platform (ECU) and a Adaptive Platform.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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 `ECUs` or `machines`. 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 [

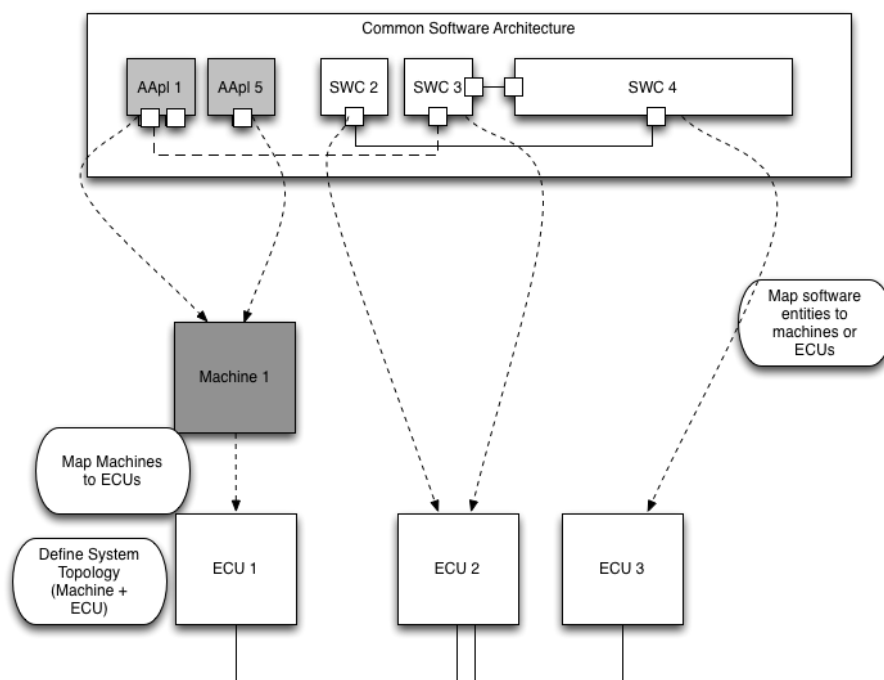
The subsequent specifications of the Classic Platform methodology shall also be applicable for the Adaptive Platform (by following their general meanings):

- *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 `ECUs` and networks and the deployment of software components

onto ECUs, with the extensions necessary for the Common Software Architecture and the additions to specify machines and the corresponding mapping of machines to ECUs.

- *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)



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

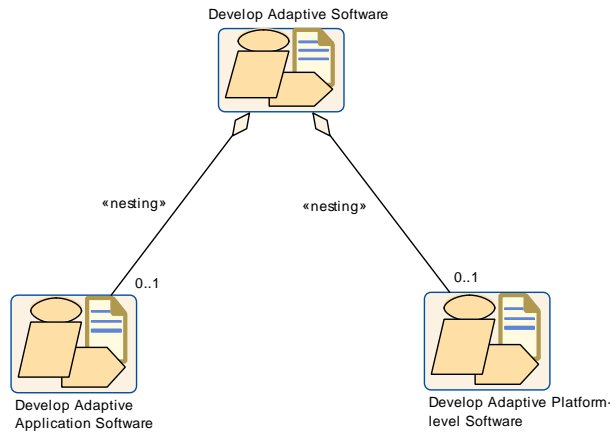
#### 2.1.2.2.4 Software Development

##### [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. ](RS\_METH\_00202, RS\_METH\_00032)<sup>5</sup>

<sup>5</sup>Sections 2.3.1 and 2.3.2 will refine the necessary activities associated with the development of application-level and platform-level software.



**Figure 2.5: Develop Adaptive Software**

<b>Activity</b>	<b>Develop Adaptive Software</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Develop Adaptive Application		
<b>Brief Description</b>	Develop Adaptive Software		
<b>Description</b>	This higher level activity encloses the development of Adaptive Applications with category application-level as well as platform-level.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Aggregates	Develop Adaptive Application Software	0..1	
Aggregates	Develop Adaptive Platform-level 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). ](RS\_METH\_00202, RS\_METH\_00032)<sup>6</sup>

<sup>6</sup>Section 2.4.1 will refine the necessary activities associated with the integration of software.



**[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`.
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`.

]([RS\\_METH\\_00204](#), [RS\\_METH\\_00203](#))<sup>7</sup>

**[TR\_AMETH\_00004] Creation of the `Execution Manifest`** [ Executables of an `AdaptiveAutosarApplication` are instantiated by means of the `Execution 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 `Execution Manifest` may also define dependencies of processes. ]([RS\\_METH\\_00203](#))<sup>8</sup>

**[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`. ]([RS\\_METH\\_00206](#), [RS\\_METH\\_00203](#))<sup>9</sup>

**[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 Packages`. This means that:

---

<sup>7</sup>See Section 2.2.3 for details regarding `Machine Design`. See Section 2.4.2 for details regarding `Machine Manifest`.

<sup>8</sup>The creation of the `Execution Manifest` is detailed in Section 2.4.3.

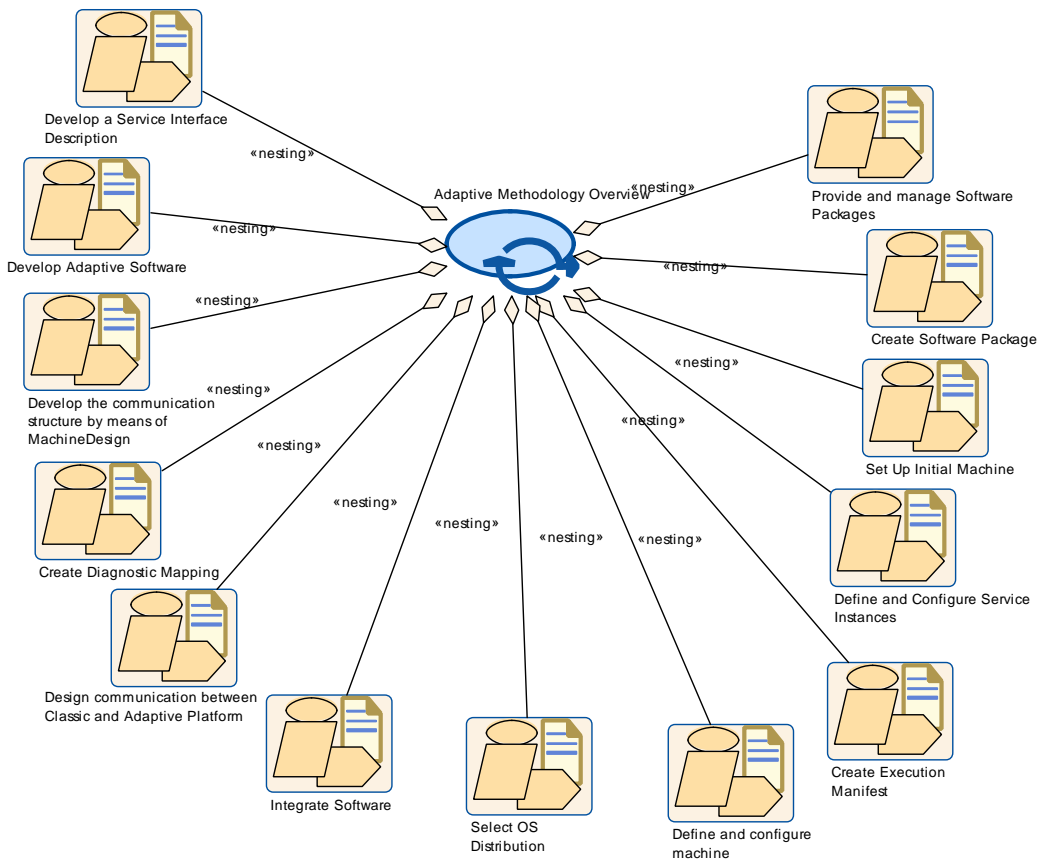
<sup>9</sup>See Section 2.4.4 for details.



1. associated software artifacts need to be compiled into a dedicated Software Package.
2. Software Packages are provided by an OEM-specific Back-end server in order to be accessible by the machines in the field.

|(RS\_METH\_00205)<sup>10</sup>

### 2.1.2.3 Workflow



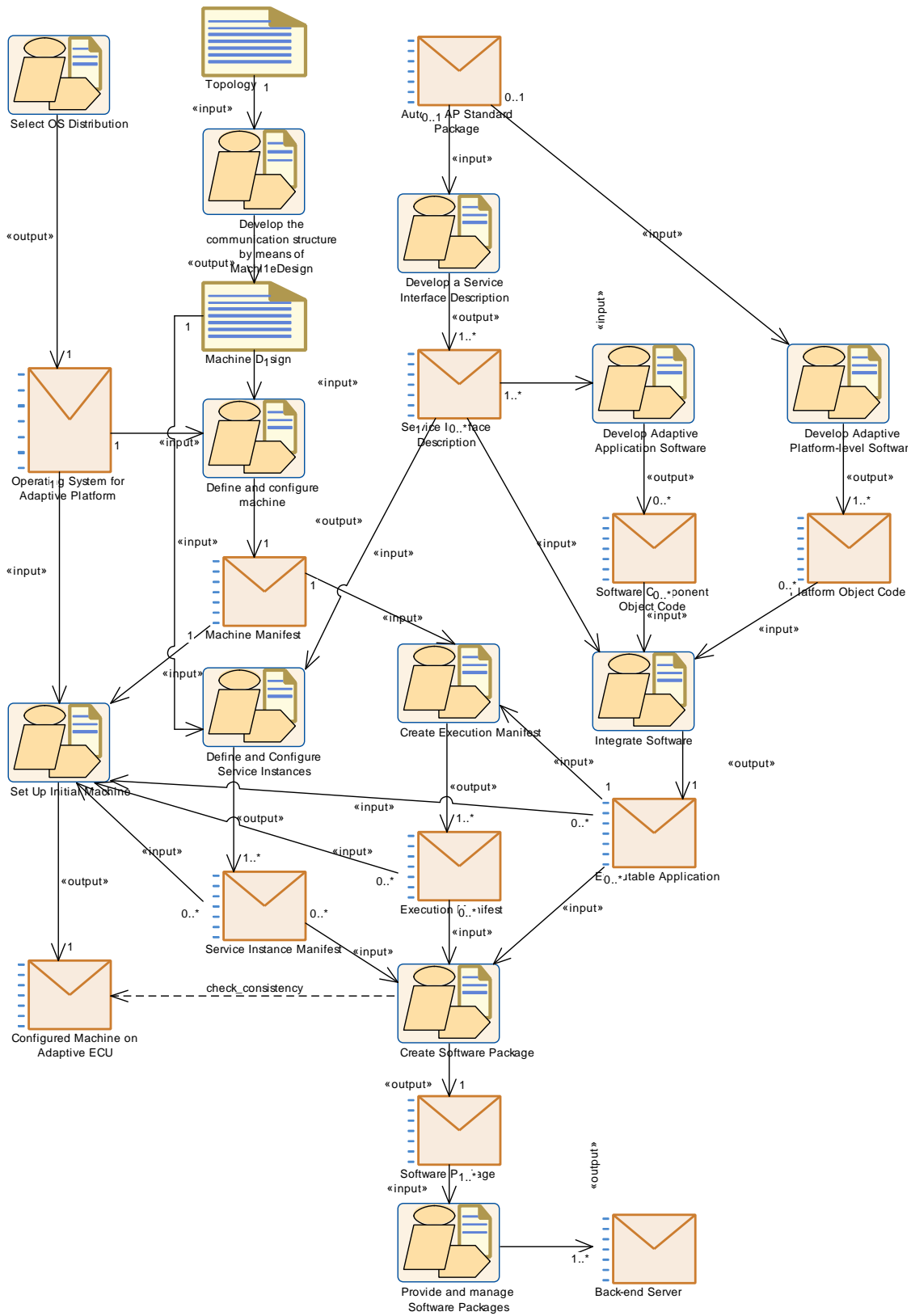
**Figure 2.6: Adaptive Methodology Overview: Overall Structure**

<b>Process Pattern</b>	<b>Adaptive Methodology Overview</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Adaptive Methodology Overview		
<b>Brief Description</b>	High-level view of the adaptive AUTOSAR methodology		
<b>Description</b>	This process pattern covers the typical activities to develop an Adaptive AUTOSAR system.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Aggregates	Create Diagnostic Mapping	1	

<sup>10</sup>See section 2.4.6 regarding create Software Packages. See section) 2.4.7 regarding deploy Software Packages.

<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Aggregates	Create Execution Manifest	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 Platform-level Software	1	
Aggregates	Develop Adaptive 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**



**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

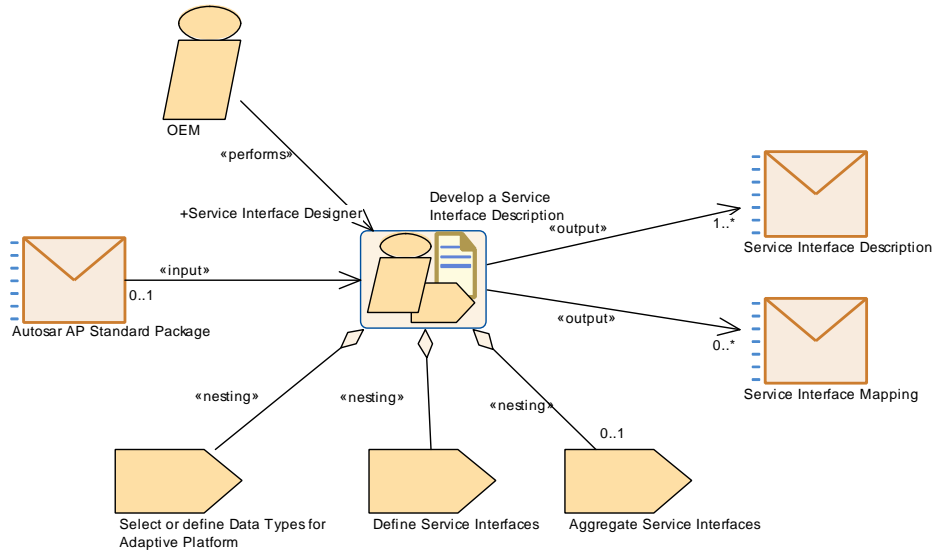
**[TR\_AMETH\_00007] Definition of data types for the Adaptive Platform** [ Data types for the Adaptive Platform can be defined based on standardized data types from AUTOSAR. As on the Classic Platform, data types are defined on different levels of abstractions: application data types, implementation 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](#))

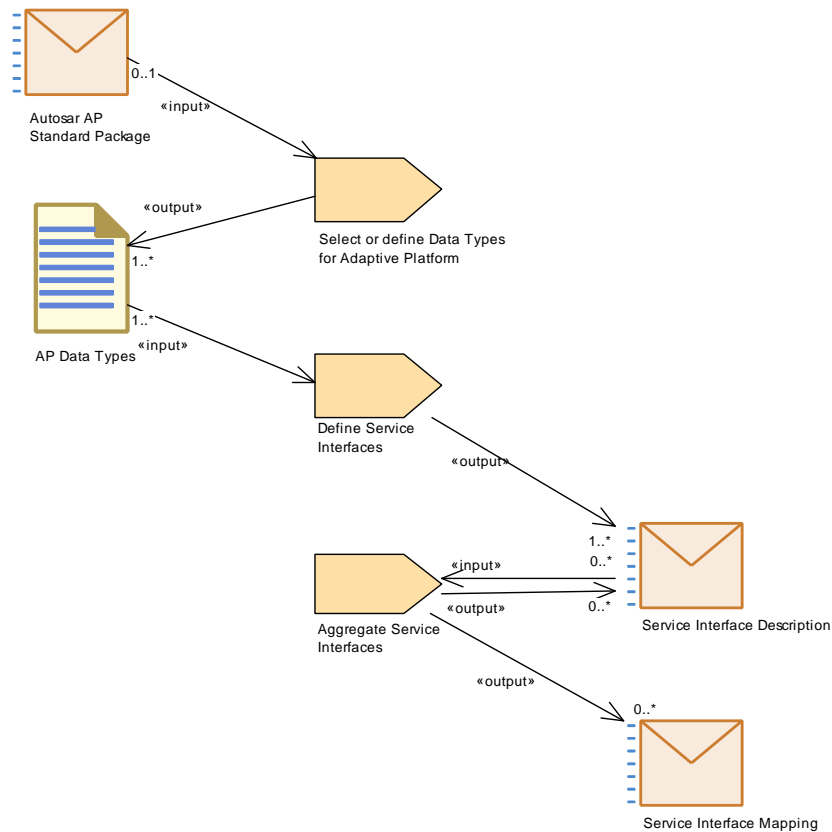
**2.2.1.3 Workflow**



**Figure 2.8: Develop a Service Interface Description**

<b>Activity</b>	<b>Develop a Service Interface Description</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Service Interface Definition		
<b>Brief Description</b>	Define all service interfaces used in the system		
<b>Description</b>	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.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Autosar AP Standard Package</a>	0..1	Optional input for defining data types and service interfaces for the adaptive platform
Produces	<a href="#">Service Interface Description</a>	1..*	All service interfaces, which are used for communication
Produces	<a href="#">Service Interface Mapping</a>	0..*	Optionally, coarse-grained service interfaces are defined by a service interface mapping
Aggregates	<a href="#">Aggregate Service Interfaces</a>	0..1	
Aggregates	<a href="#">Define Service Interfaces</a>	1	
Aggregates	<a href="#">Select or define Data Types for Adaptive Platform</a>	1	
Performed by	<a href="#">OEM</a>	1	Service Interface Designer: This activity will probably be performed by a Service Interface Designer

**Table 2.4: Develop a Service Interface Description**



**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 `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 an 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 Adaptive Platform.

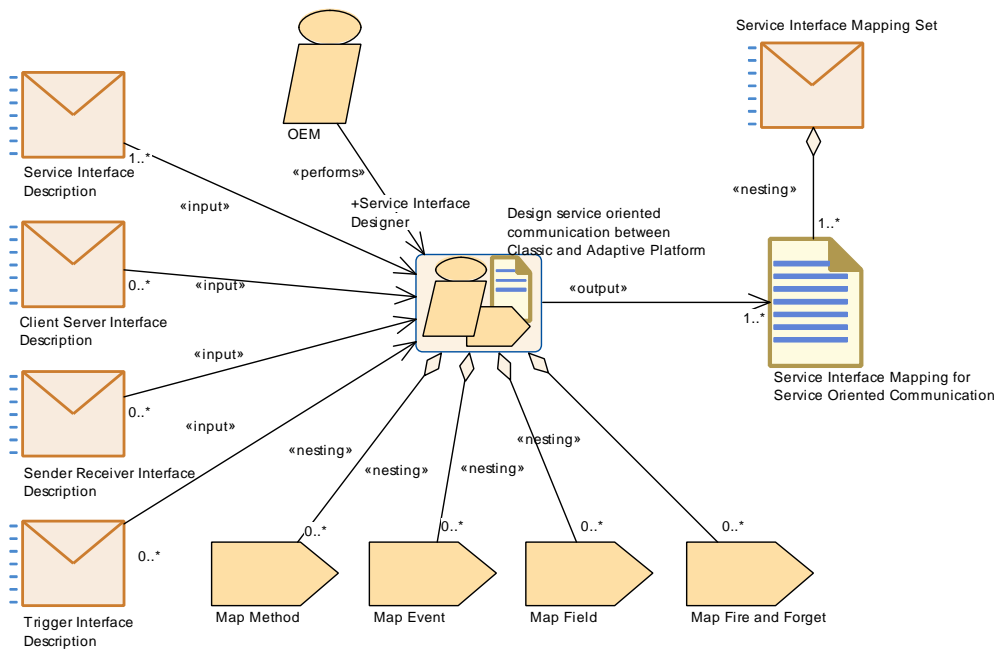
Thus, the main objective of this activity is to map a single `ServiceInterface` to `PortInterface` elements, in detail:

- 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 `ClientServerOperations` 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



**Figure 2.10: Design service oriented communication**

<b>Activity</b>	<b>Design service oriented communication between Classic and Adaptive Platform</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Communication		
<b>Brief Description</b>	Design service oriented communication between CP and AP		
<b>Description</b>	<p>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.</p> <p>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.</p> <p>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 Adaptive Platform.</p> <p>The mapping description serves currently only for documentation.</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Client Server Interface Description</a>	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



<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Sender Receiver Interface Description</a>	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	<a href="#">Service Interface Description</a>	1..*	Description of the Service Interfaces which communicate to CP in a service-oriented manner
Consumes	<a href="#">Trigger Interface Description</a>	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	<a href="#">Service Interface Mapping for Service Oriented Communication</a>	1..*	An InterfaceMapping results from the design of service-oriented communication between CP and AP
Aggregates	<a href="#">Map Event</a>	0..*	
Aggregates	<a href="#">Map Field</a>	0..*	
Aggregates	<a href="#">Map Fire and Forget</a>	0..*	
Aggregates	<a href="#">Map Method</a>	0..*	
Performed by	<a href="#">OEM</a>	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`.

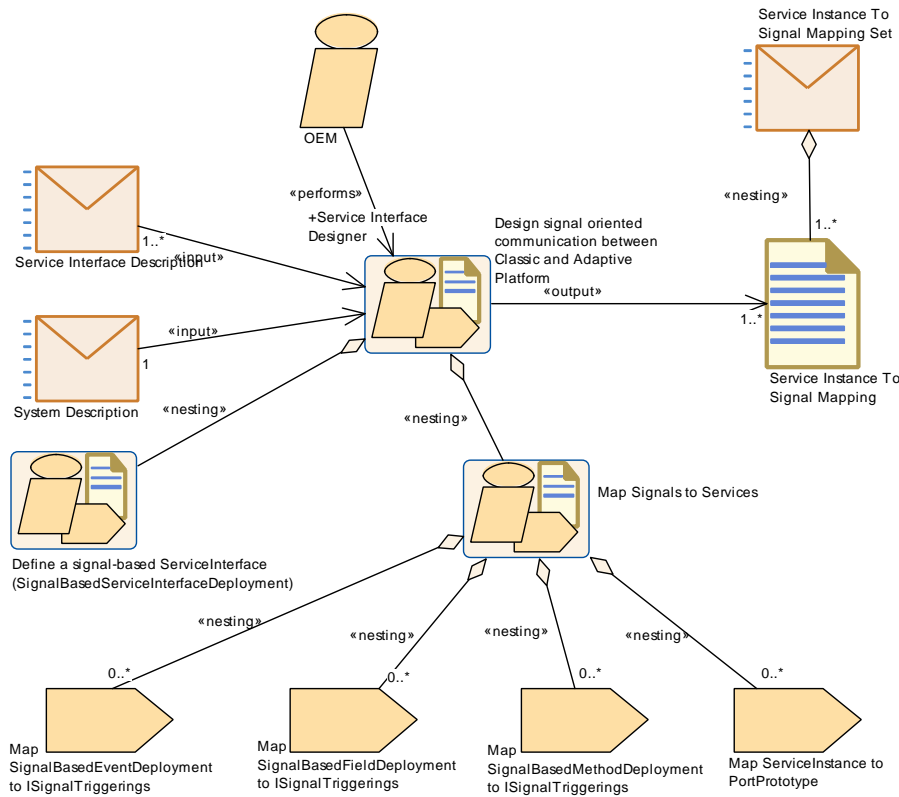
Details are provided by the specifications `TPS_MANI_03120`, `TPS_MANI_03121`, `TPS_MANI_03122` and `TPS_MANI_03123` of the Manifest specification [6]. ]  
([RS\\_METH\\_00200](#), [RS\\_METH\\_00202](#))

**[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**



**Figure 2.11: Design signal oriented communication**

<b>Activity</b>	<b>Design signal oriented communication between Classic and Adaptive Platform</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Communication		
<b>Brief Description</b>	Design signal oriented communication between CP and AP		
<b>Description</b>	<p>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.</p> <p>If the counterpart on a Classic Platform ECU, however, communicates only in a signal-based way, a Signal-to-Service translation is needed.</p> <p>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.</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Service Interface Description</a>	1..*	Description of the Service Interfaces which communicate to CP in a signal-oriented manner

<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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**

<b>Activity</b>	<b>Map Signals to Services</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Communication		
<b>Brief Description</b>	Map Signals to Services		
<b>Description</b>	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.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Aggregates	Map ServiceInstance to Port Prototype	0..*	
Aggregates	Map SignalBased EventDeployment to ISignal Triggerings	0..*	
Aggregates	Map SignalBased FieldDeployment to ISignal Triggerings	0..*	
Aggregates	Map SignalBased MethodDeployment to ISignal Triggerings	0..*	

**Table 2.7: Map Signals to Services**

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

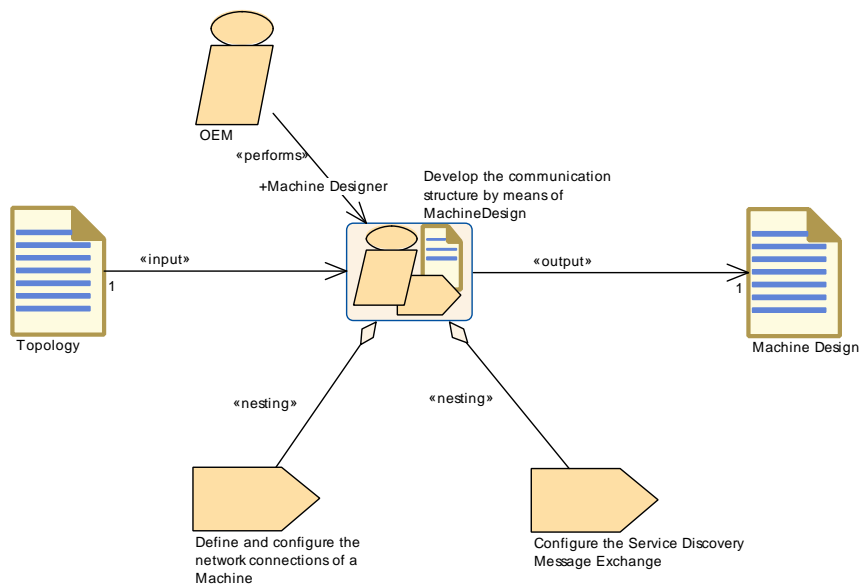


Figure 2.12: Develop the communication structure by means of `Machine Design`

<b>Activity</b>	<b>Develop the communication structure by means of MachineDesign</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Develop Machine Design		
<b>Brief Description</b>	placeholder during the design phase for an adaptive ECU(Machine) in the Scope of an System.		
<b>Description</b>	<p>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.</p> <p>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.</p> <p>This activity will aggregate the following tasks:</p> <ul style="list-style-type: none"> <li>• Define and configure the network connection of a prospective machine</li> <li>• Configure the service discovery message exchange of a prospective machine</li> </ul>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	Topology	1	Description of (inter)connections between Machines.
Produces	<a href="#">Machine Design</a>	1	Configuration settings of the network connections and service discovery network exchange of a Machine
Aggregates	<a href="#">Configure the Service Discovery Message Exchange</a>	1	
Aggregates	<a href="#">Define and configure the network connections of a Machine</a>	1	
Performed by	<a href="#">OEM</a>	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](#).

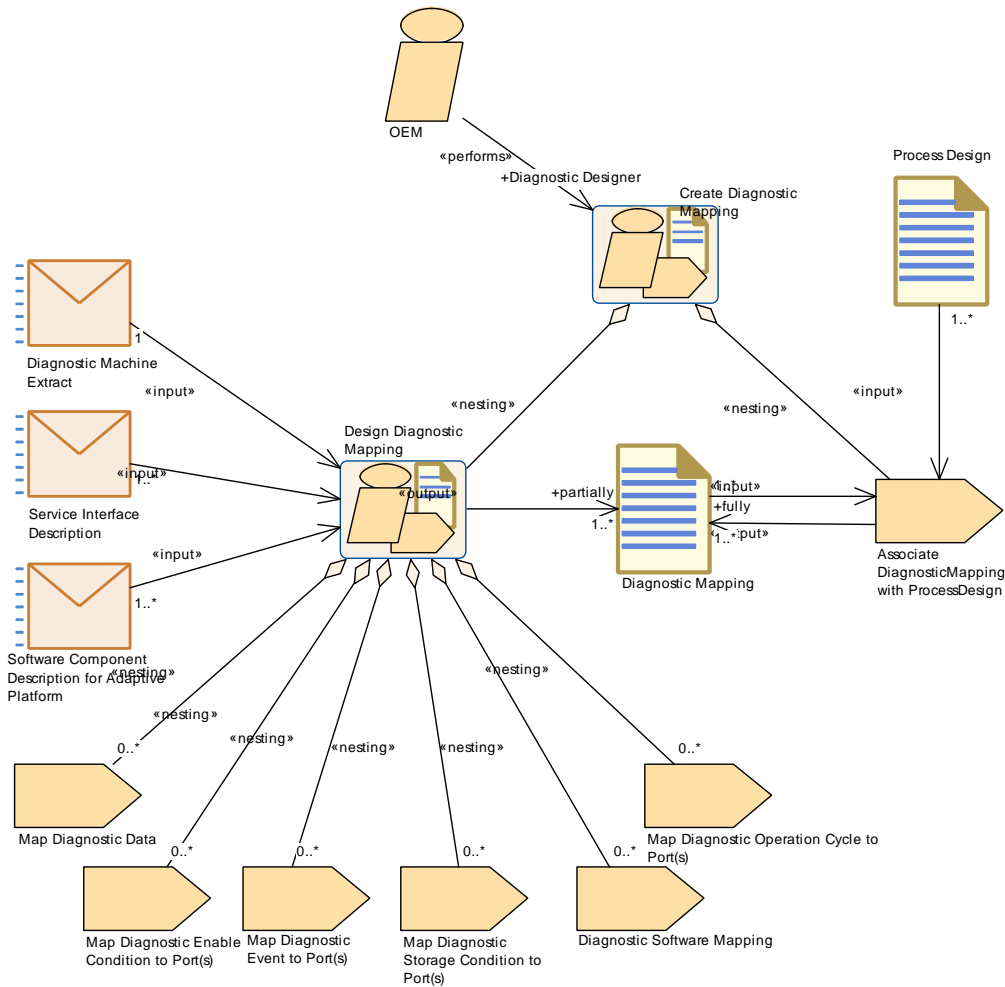
To accommodate for this potential modeling, the reference from a diagnostic mapping to ProcessDesign has been decorated by stereotype «atpSplitable».

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**



**Figure 2.13: Create a Diagnostic Mapping**

<b>Activity</b>	<b>Create Diagnostic Mapping</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Diagnostic Mapping		
<b>Brief Description</b>	Create diagnostic mappings		
<b>Description</b>	<p>This activity comprises all necessary tasks to create complete diagnostic mappings.</p> <p>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.</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Aggregates	<a href="#">Associate DiagnosticMapping with Process Design</a>	1	

<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Aggregates	<a href="#">Design Diagnostic Mapping</a>	1	
Performed by	<a href="#">OEM</a>	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**

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

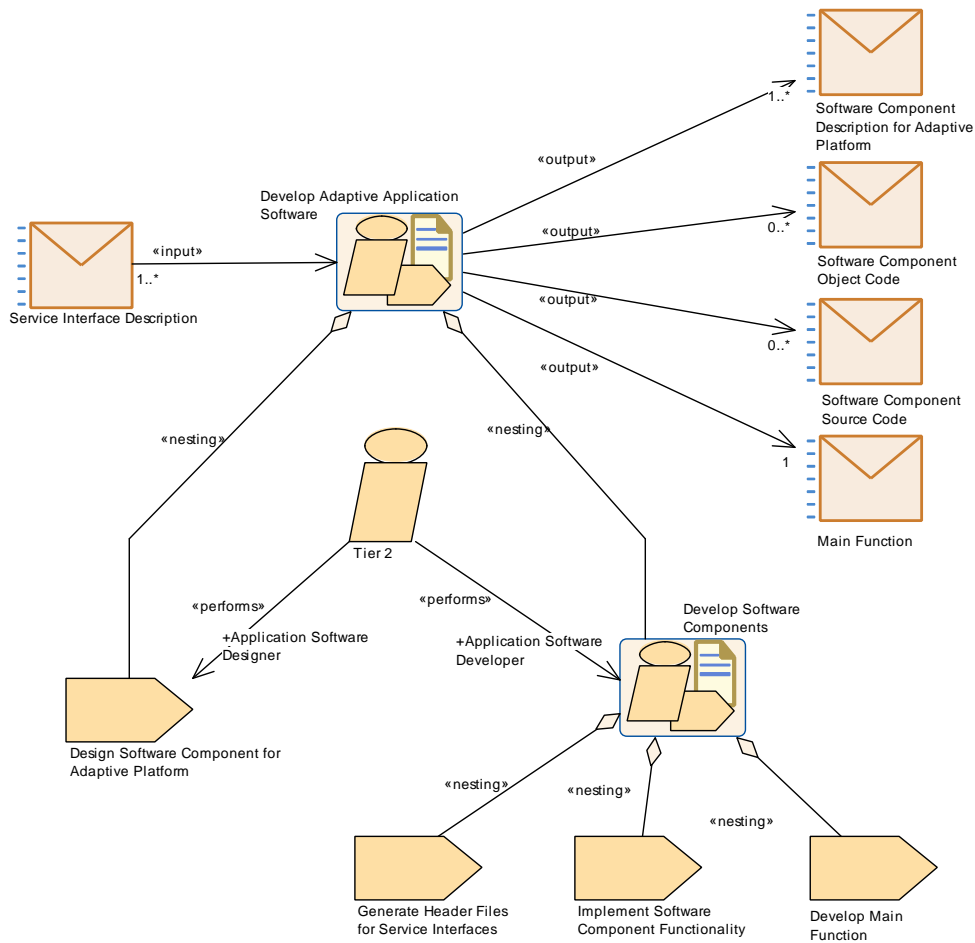
The generation includes the generation of service proxies and skeletons, which need to be implemented for a specific platform. ]([RS\\_METH\\_00202](#), [RS\\_METH\\_00066](#))

**[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**



**Figure 2.14: Develop Adaptive Application Software**

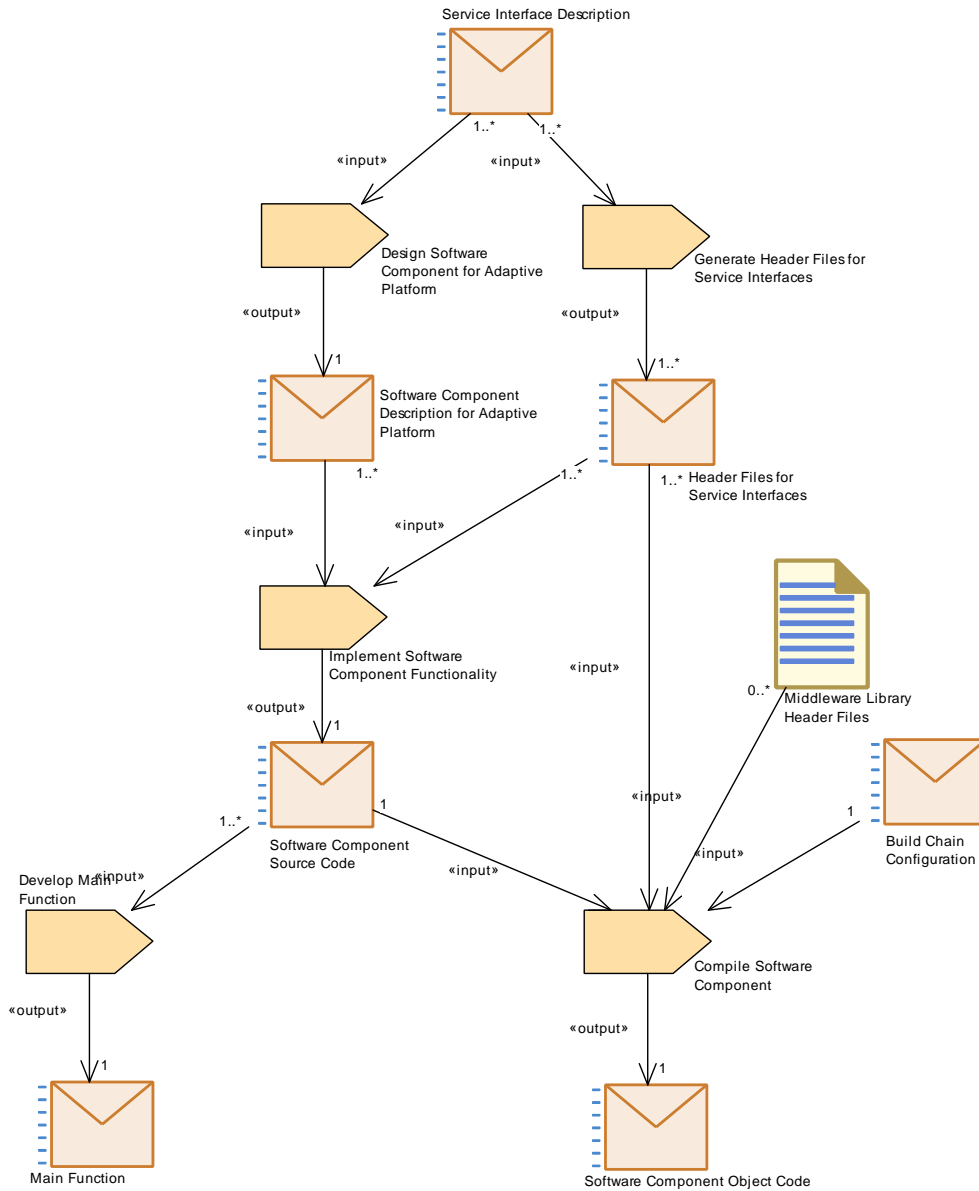
<b>Activity</b>	<b>Develop Adaptive Application Software</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Develop Adaptive Application		
<b>Brief Description</b>	Design and development of software components for Adaptive Platform		
<b>Description</b>	Develop 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.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Service Interface Description</a>	1..*	Service Interfaces are the basis for the development of adaptive application software
Produces	<a href="#">Main Function</a>	1	One main function per executable is produced
Produces	<a href="#">Software Component Description for Adaptive Platform</a>	1..*	Output of component model for the software components
Produces	<a href="#">Software Component Object Code</a>	0..*	Compiled software components
Produces	<a href="#">Software Component Source Code</a>	0..*	Software components as source code
Aggregates	<a href="#">Design Software Component for Adaptive Platform</a>	1	
Aggregates	<a href="#">Develop Software Components</a>	1	

**Table 2.12: Develop Adaptive Application Software**

<b>Activity</b>	<b>Develop Software Components</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Develop Adaptive Application		
<b>Brief Description</b>	Implement the core functionality of one executable application		
<b>Description</b>	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.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Aggregates	<a href="#">Develop Main Function</a>	1	
Aggregates	<a href="#">Generate Header Files for Service Interfaces</a>	1	
Aggregates	<a href="#">Implement Software Component Functionality</a>	1	

Relation Type	Related Element	Mul.	Note
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**



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

## 2.3.2 Develop Adaptive Platform-level 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 Execution 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

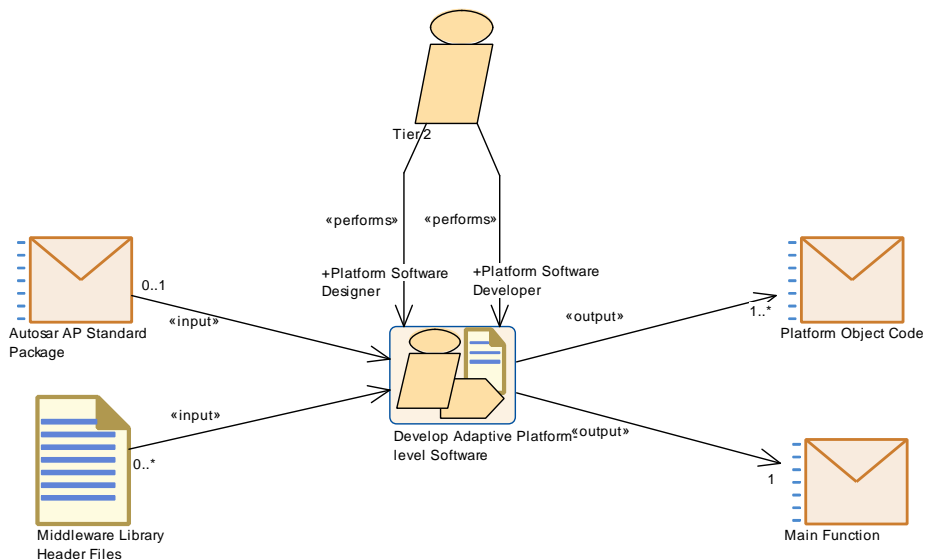


Figure 2.16: Develop Adaptive Platform-level Software

<b>Activity</b>	<b>Develop Adaptive Platform-level Software</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Develop Adaptive Application		
<b>Brief Description</b>	Develop an Adaptive Software of category platform-level.		
<b>Description</b>	Develop an Adaptive Software of category platform-level. These platform software modules consist of an executable and are deployed together with an Execution Manifest onto the machine (in contrast to e.g. the OS). This activity also includes the implementation of the corresponding main function.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Autosar AP Standard Package</a>	0..1	In case standardized service interfaces are used for adaptive platform-level software
Consumes	<a href="#">Middleware Library Header Files</a>	0..*	Library header files needed for compiling the adaptive platform-level software
Produces	<a href="#">Main Function</a>	1	Main function for platform-level executable
Produces	<a href="#">Platform Object Code</a>	1..*	Object code of platform module
Performed by	<a href="#">Tier 2</a>	1	Platform Software Designer: The design tasks within the development of Platform-level Software will probably be performed by a Platform Software Designer of a Tier 2 company
Performed by	<a href="#">Tier 2</a>	1	Platform Software Developer: The real development tasks (i.e., to write source code and the like) within the development of Platform-level Software will probably be performed by a Platform Software Developer of a Tier 2 company

**Table 2.14: Develop Adaptive Platform-level 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

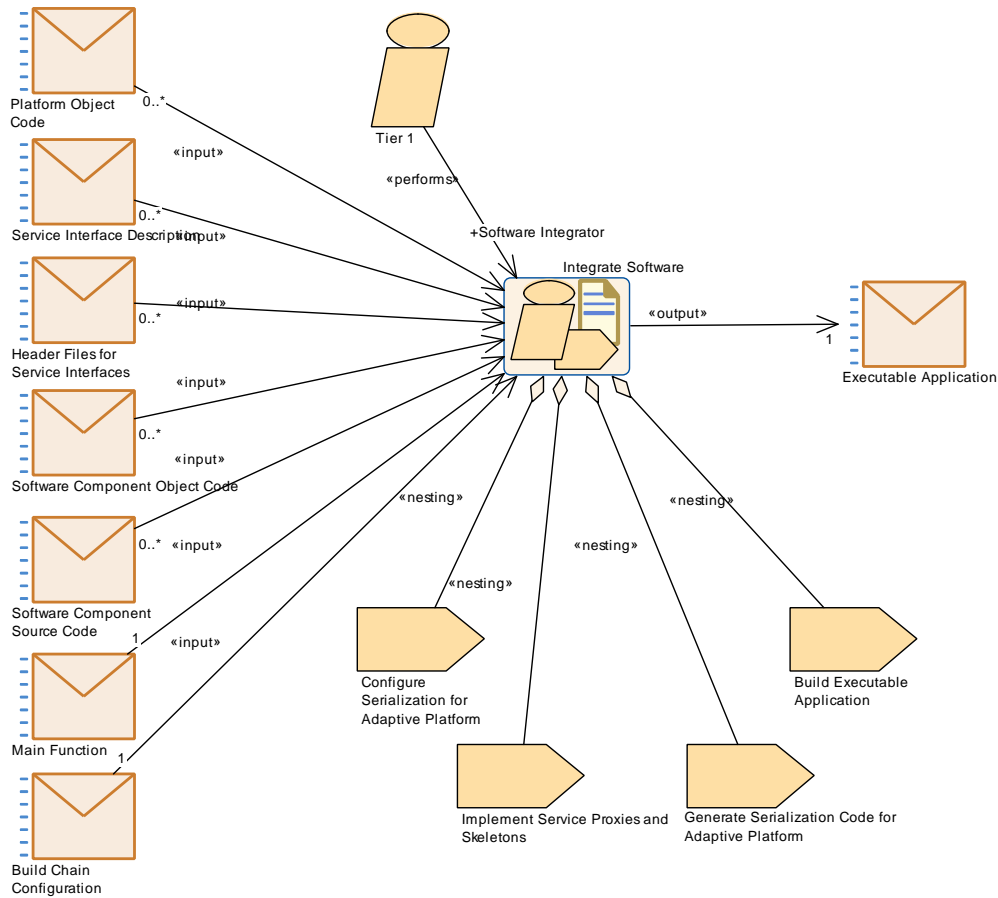
**[TR\_AMETH\_00016] Development of serialization properties** [ It needs to be described how the data in the service interfaces shall be serialized for the transport on the network. In particular, this is important for the communication over SOME/IP between Classic and Adaptive Platform.

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

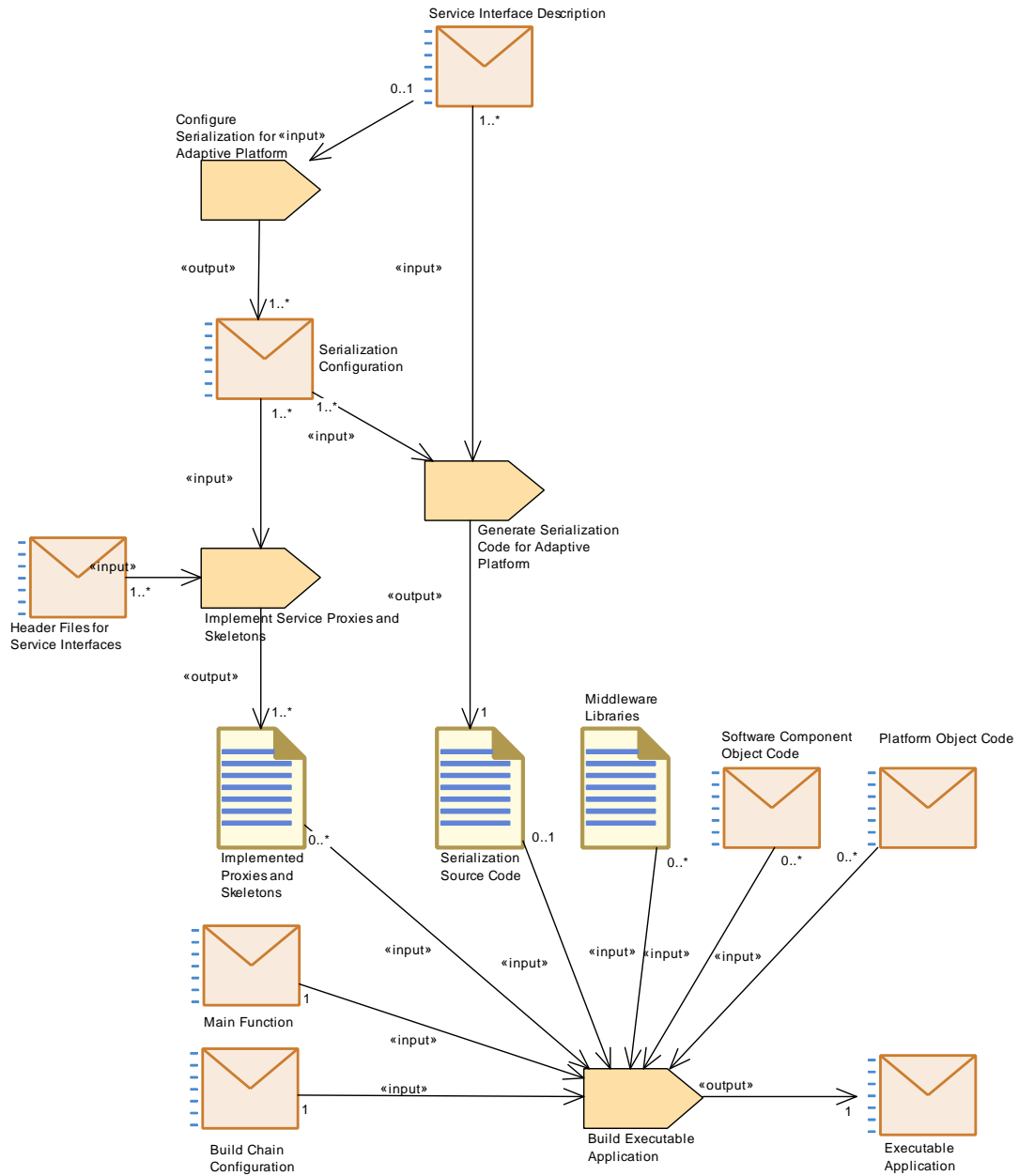


**Figure 2.17: Integrate the software components**

<b>Activity</b>	<b>Integrate Software</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Integration::Integrate Software		
<b>Brief Description</b>	Integrate software to one executable		
<b>Description</b>	<p>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.</p> <p>Several executables can later be packaged into an Adaptive AUTOSAR Application.</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Build Chain Configuration</a>	1	Needed for linking all artifacts
Consumes	<a href="#">Header Files for Service Interfaces</a>	0..*	Proxies and skeletons to be implemented
Consumes	<a href="#">Main Function</a>	1	One main function per executable
Consumes	<a href="#">Platform Object Code</a>	0..*	Object code for platform-level executable

<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Service Interface Description</a>	0..*	Needed for defining the serialization
Consumes	<a href="#">Software Component Object Code</a>	0..*	Object code for application-level executable
Consumes	<a href="#">Software Component Source Code</a>	0..*	Source code for application-level executable
Produces	<a href="#">Executable Application</a>	1	Software is integrated into one executable application
Aggregates	<a href="#">Build Executable Application</a>	1	
Aggregates	<a href="#">Configure Serialization for Adaptive Platform</a>	1	
Aggregates	<a href="#">Generate Serialization Code for Adaptive Platform</a>	1	
Aggregates	<a href="#">Implement Service Proxies and Skeletons</a>	1	
Performed by	<a href="#">Tier 1</a>	1	Software Integrator: This activity will probably be performed by a Software Integrator of a Tier 1 company

**Table 2.15: Integrate Software**



**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

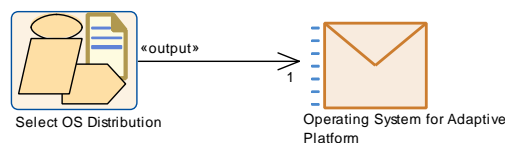
**[TR\_AMETH\_00019] Description of the Adaptive Platform** [ As a first preparatory step, the available hardware elements of the particular Adaptive Platform need to be specified. This can be done by means of the [ECU Resources Description](#) which enables to describe all hardware elements, like processing 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 not having an [Execution Manifest](#). Note, that its development workflow will differ from the workflow of platform-level software. ]([RS\\_METH\\_00207](#), [RS\\_METH\\_00041](#)) <sup>11</sup>

### 2.4.2.1.3 Workflow



**Figure 2.19: Select the OS Distribution**

<b>Activity</b>	<b>Select OS Distribution</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Define and Configure Machine::Develop Platform Software		
<b>Brief Description</b>	Select and assemble an operating system		
<b>Description</b>	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 Execution Manifest.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>

<sup>11</sup>see section [2.3.2](#) for platform-level software development workflow details.

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produces	<a href="#">Operating System for Adaptive Platform</a>	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

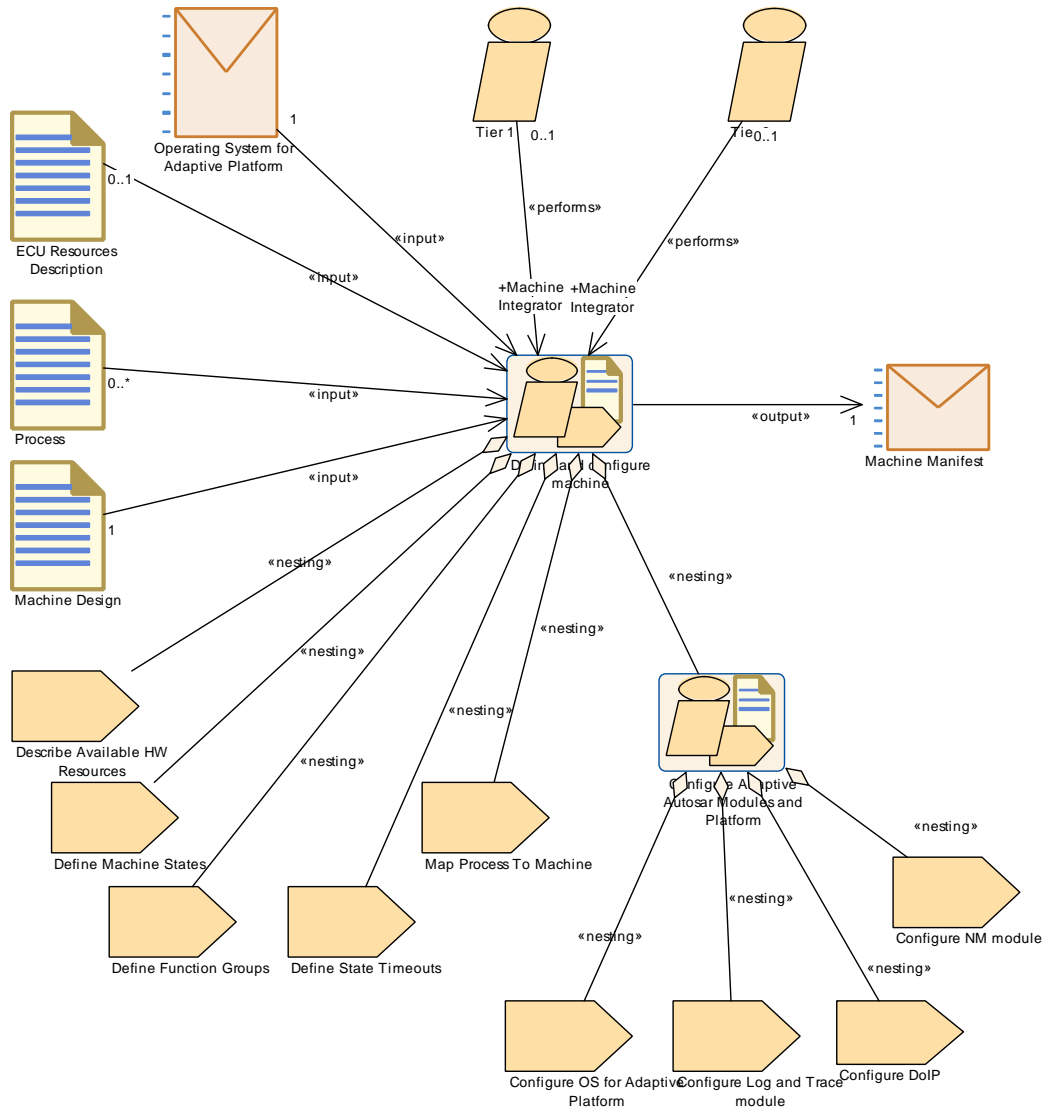
]([RS\\_METH\\_00204](#), [RS\\_METH\\_00203](#))

**[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**

<b>Activity</b>	<b>Define and configure machine</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Define and Configure Machine::Machine Configuration		
<b>Brief Description</b>	Configuration of the machine independent of deployment information of applications or service instances		
<b>Description</b>	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.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>



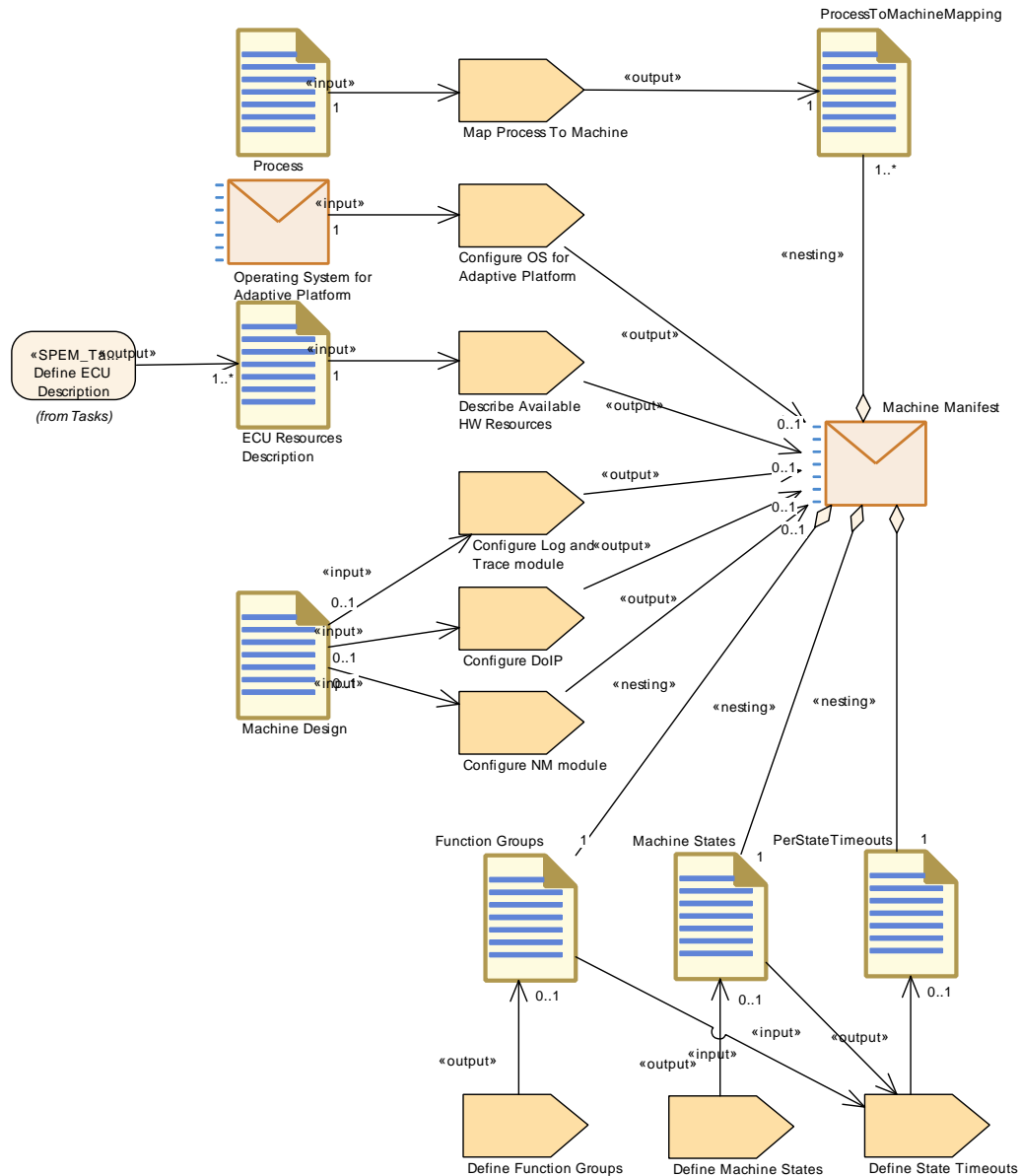
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumes	ECU Resources Description	0..1	All resources which are available for the ECU
Consumes	Machine Design	1	Configuration settings of the network connections and service discovery network exchange of a Machine
Consumes	Operating 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 Time-outs	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**

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

Relation Type	Related Element	Mul.	Note
---------------	-----------------	------	------

**Table 2.18: Configure Adaptive Autosar Modules and Platform**



**Figure 2.21: Workflow for defining and configuring an machine**

## 2.4.3 Create Execution Manifest

### 2.4.3.1 Purpose

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

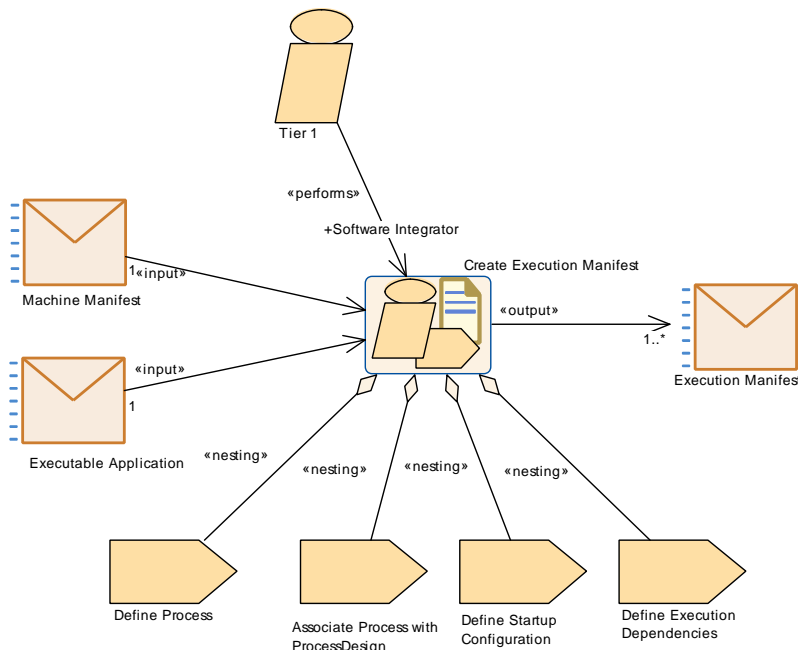
### 2.4.3.2 Description

**[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 Execution Manifest** [ The **Execution Manifest** aggregates the process and its startup configuration. Therefore, one **Execution Manifest** is defined per process. ] ([RS\\_METH\\_00203](#))

### 2.4.3.3 Workflow



**Figure 2.22: Create an Execution Manifest**

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

**Table 2.19: Create Execution Manifest**

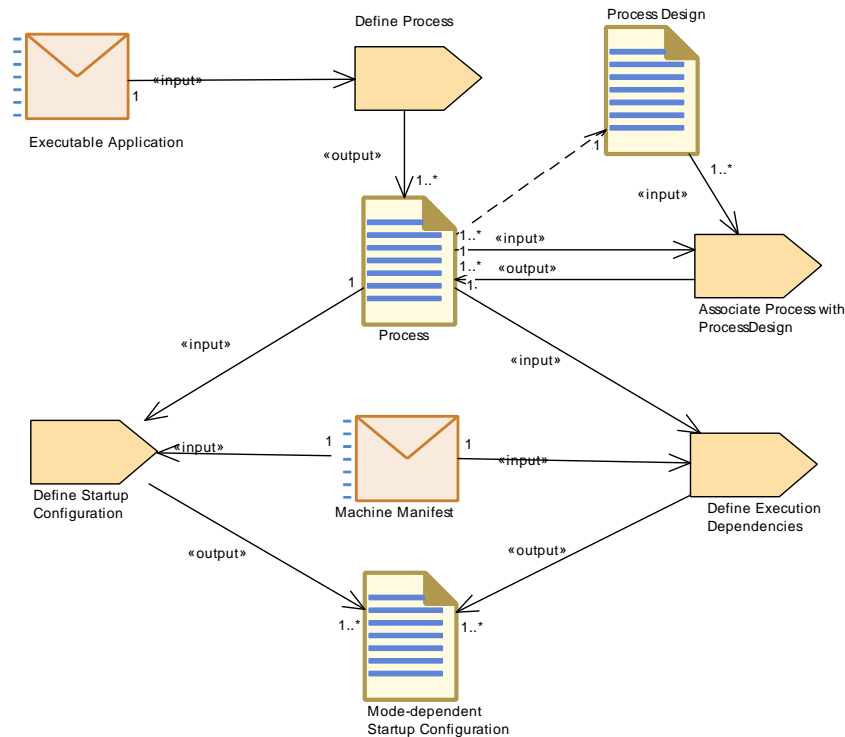


Figure 2.23: Workflow for defining a Process

## 2.4.4 Define and Configure Service Instances

### 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 specifies in [Service Interface Deployment Configuration](#) how the service interfaces shall be deployed. This includes the properties describing the individual transport layer binding of the service interface.

E.g. 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 unambiguously in the scope of the SOME/IP service interface deployment. ] ([RS METH\\_00206](#), [RS METH\\_00203](#))<sup>12</sup>

**[TR\_AMETH\_00028] Configuration of Service Instances** [ Afterwards, the system responsible defines instances of the deployed service interfaces and decides

<sup>12</sup>see 3.9.2.1

whether the service instance is provided or consumed. In order to set up the service-oriented communication [Service Instance Configuration](#) includes properties for search or offer criteria.

E.g. for SOME/IP, an ID for each provided service instance is defined. This ID needs to be unique in the system (and should be globally unambiguous). For required service instances SOME/IP allows to specify optionally a required service instance ID (which ofcourse should be provided somewhere). ]([RS\\_METH\\_00206](#), [RS\\_METH\\_00203](#))<sup>13</sup>

**[TR\_AMETH\_00029] Mapping of Service Instances to Machine** [ The service instances will be deployed to a Machine (i.e. a Adaptive Platform instance) that will execute the service instance. This [Service Instance To Machine Mapping](#) includes technology specific properties.

E.g. 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](#))<sup>14</sup>

**[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. to instances of port prototypes) via the [Service Instance To Port Prototype Mapping](#). This mapping is necessary in order to ensure a unique relationship between locally implemented service instances within the application and global service instances available on the network. The [Service Instance To Port Prototype Mapping](#) includes technology specific properties.

E.g. for SOME/IP the provided (and optionally also required) service instance IDs are specified. ]([RS\\_METH\\_00206](#), [RS\\_METH\\_00203](#), [RS\\_METH\\_00078](#))<sup>15</sup>

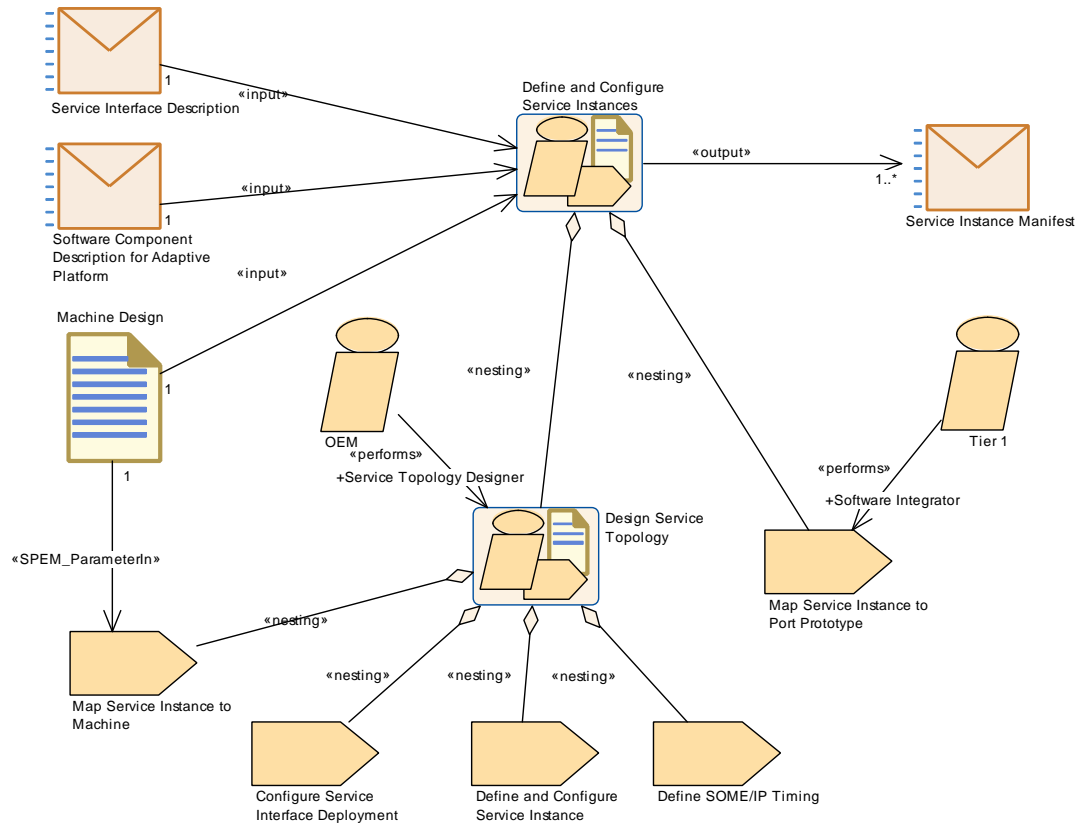
---

<sup>13</sup>see [3.9.2.2](#)

<sup>14</sup>see [3.9.1.5](#)

<sup>15</sup>see [3.9.1.4](#)

### 2.4.4.3 Workflow



**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	

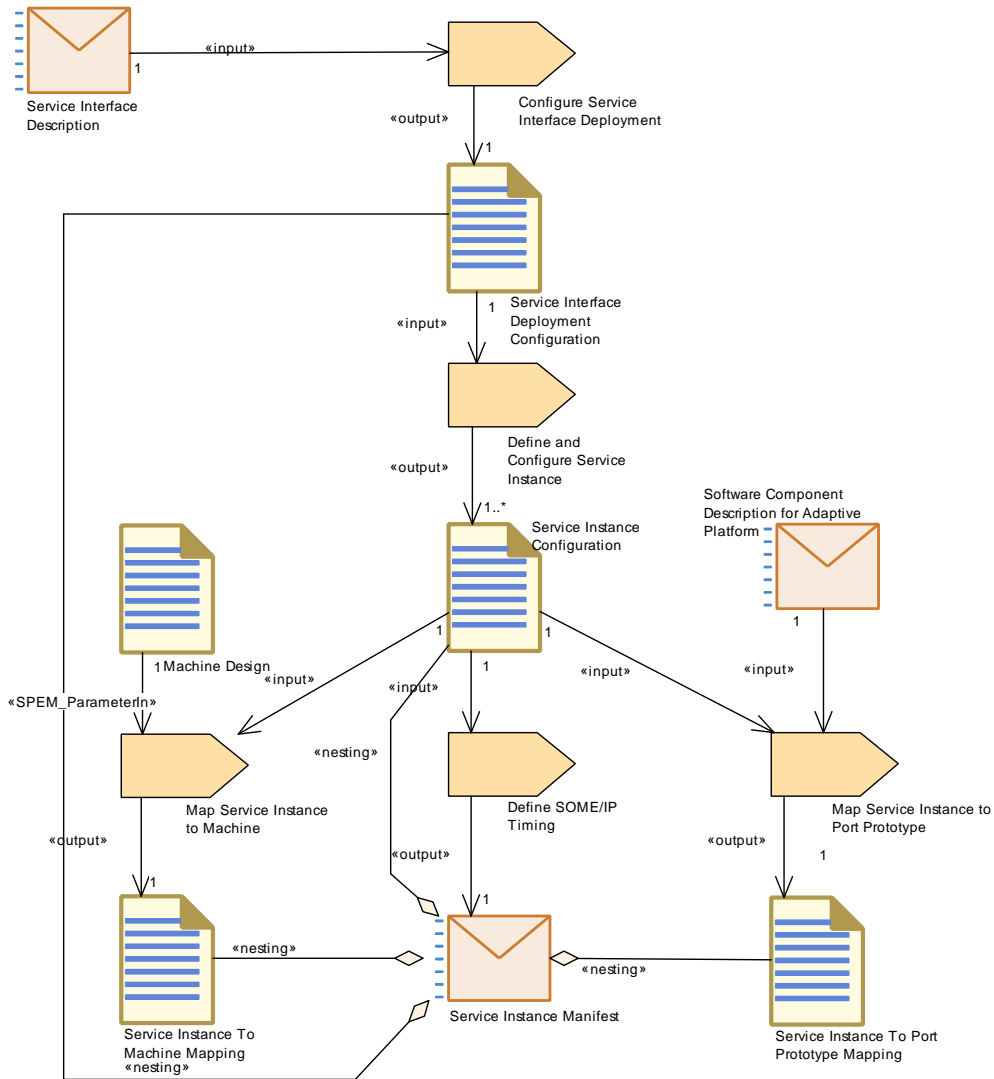
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Aggregates	Map Service Instance to Port Prototype	1	

**Table 2.20: Define and Configure Service Instances**

<i>Activity</i>	<b>Design Service Topology</b>		
<i>Package</i>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Service Instance Definition		
<i>Brief Description</i>	Design Service Topology		
<i>Description</i>	This activity subsumed all design tasks which are related to the design of a network topology		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
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**





**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](#).

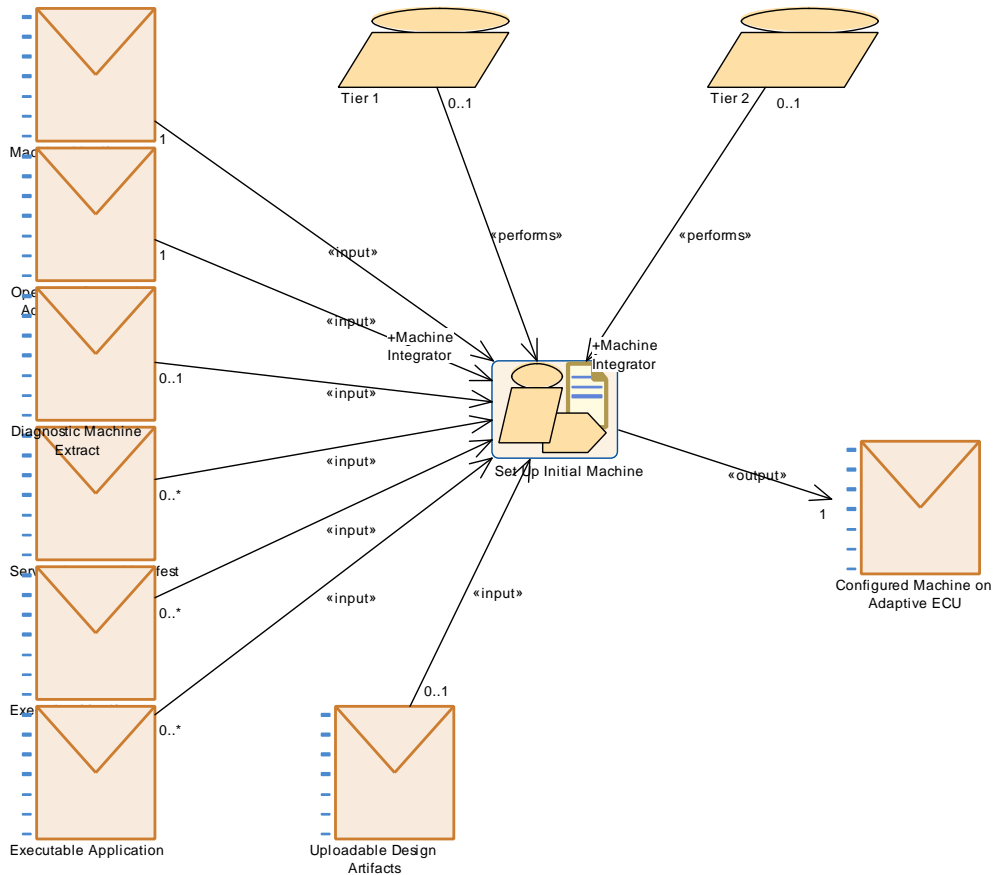
In order to be able to execute this activity, the following inputs are necessary:

- 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
- Execution 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**

<b>Activity</b>	<b>Set Up Initial Machine</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Define and Configure Machine::Setup Machine		
<b>Brief Description</b>	Set up the machine based on the machine manifest		
<b>Description</b>	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.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	Diagnostic Machine Extract	0..1	Diagnostic extract for a Machine
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.

<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Execution Manifest</a>	0..*	All Execution Manifests needed to run the desired adaptive application (instances or Processes) on a Machine
Consumes	<a href="#">Machine Manifest</a>	1	Containing all configuration settings for the Machine
Consumes	<a href="#">Operating System for Adaptive Platform</a>	1	OS to be installed on machine
Consumes	<a href="#">Service Instance Manifest</a>	0..*	All Service Instance Manifests needed to run the desired adaptive application (instances or Processes) on a Machine
Consumes	<a href="#">Uploadable Design Artifacts</a>	0..1	Optional input: Additional design data which are not part of an Application or Machine Manifest
Produces	<a href="#">Configured Machine on Adaptive ECU</a>	1	Machine is configured and software can now be deployed
Performed by	<a href="#">Tier 1</a>	0..1	Machine Integrator: This activity will probably be performed by a Machine Integrator of a Tier 1 company
Performed by	<a href="#">Tier 2</a>	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](#), [Execution 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 Clusters](#) (of any category)

|(RS\_METH\_00205)<sup>16</sup>

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 Clusters](#), dedicated diagnostic addresses as well as internal and external dependencies of [Software Clusters](#).

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 a 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](#) or 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 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)
- Collect belonging (software) artifacts of *Sub Software Clusters*

---

<sup>16</sup>Figure 2.27 shows the corresponding input and output deliverables.

- Collect belonging (software) artifacts of *Sub Software Clusters* 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 sub-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

]()

**[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.

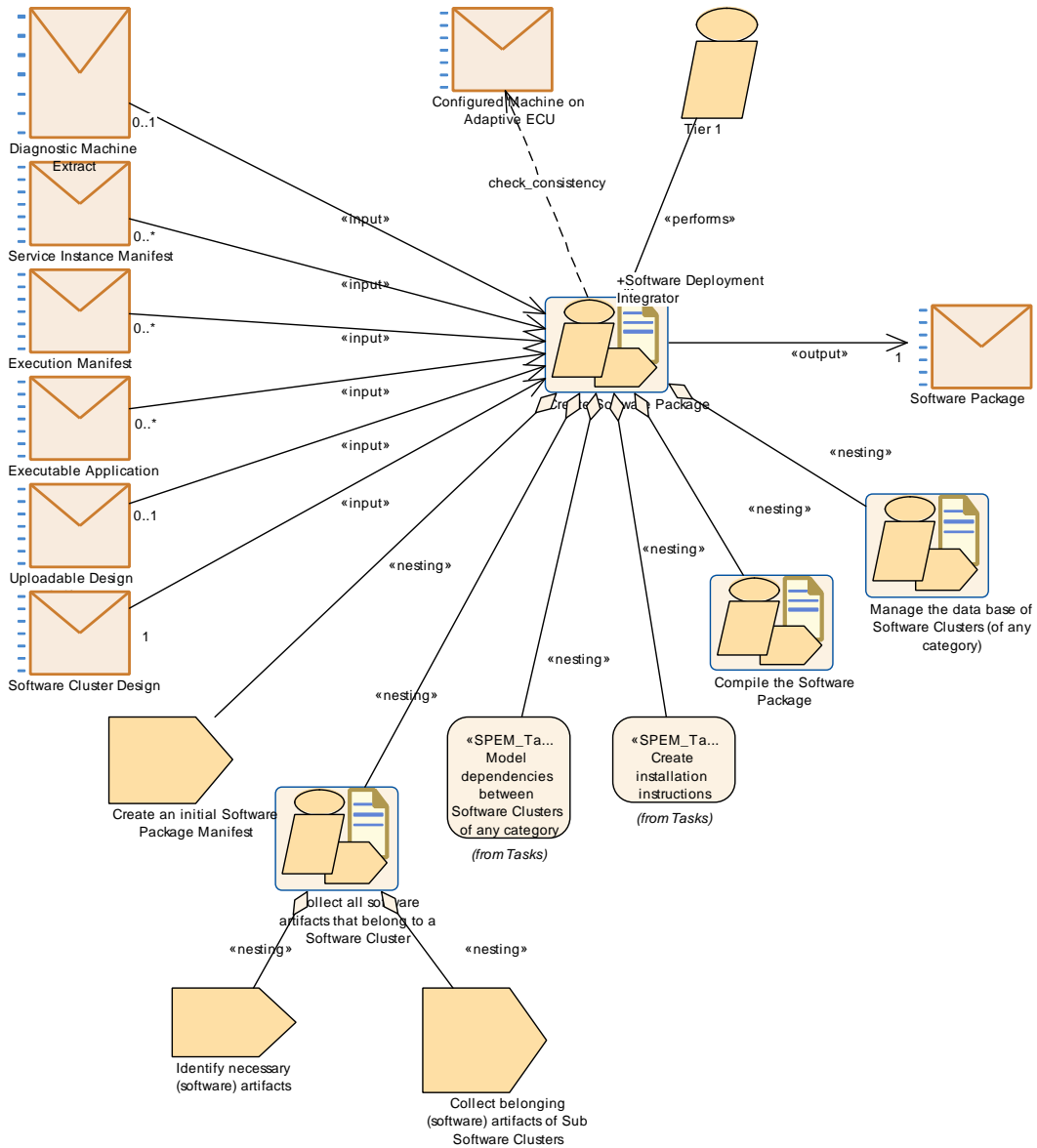
]()

**[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**



**Figure 2.27: Create a Software Package**

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

<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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**

<b>Activity</b>	<b>Collect all software artifacts that belong to a Software Cluster</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Packaging and Provision		
<b>Brief Description</b>	Collect all software artifacts		
<b>Description</b>	<p>On base of the Software Cluster Design o the newly created Software Package Manifest, this step includes the following tasks:</p> <ul style="list-style-type: none"> <li>• Identify and gather all needed (software) artifacts in order to build the Software Package, also with respect to their versions</li> <li>• Execute a receiving inspection (optional)</li> <li>• Store incoming artifacts into a repository</li> <li>• 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</li> <li>• 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</li> <li>• 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)</li> </ul>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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**

<b>Activity</b>	<b>Compile the Software Package</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Packaging and Provision		
<b>Brief Description</b>	Compile the Software Package		
<b>Description</b>	<p>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.</p> <p>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.</p> <p>Therefore, the structure of this activity by tasks is also specific to particular OEMs and their suppliers.</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produces	<a href="#">Software Package</a>	1	Compiled Software Package

**Table 2.25: Compile the Software Package**

<b>Activity</b>	<b>Manage the data base of Software Clusters (of any category)</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Packaging and Provision		
<b>Brief Description</b>	Manage the data base of Software Clusters		
<b>Description</b>	<p>A general activity may be the management of the data base of Software Clusters with respect to all their versions, dependencies and further aspects.</p> <p>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.</p>		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumes	<a href="#">Software Cluster</a>	1..*	Store and manage software cluster within a repository
Consumes	<a href="#">Software Package Manifest</a>	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.

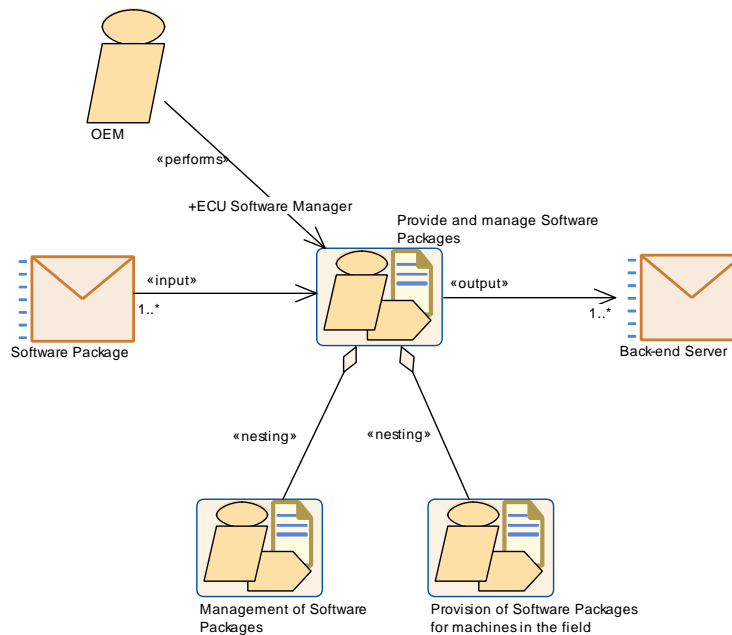
Since the management of *Software Packages* is an immanent task of an *OEM* and will differ between the companies, this activity will not be detailed further. ]()

**[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**



**Figure 2.28: Provision of Software Packages**

<b>Activity</b>	<b>Provide and manage Software Packages</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Packaging and Provision		
<b>Brief Description</b>	Provide and manage Software Packages		
<b>Description</b>	<p>This activity may comprise two aspects:</p> <ul style="list-style-type: none"> <li>• The management of Software Packages ready to upload onto the machines</li> <li>• The provision of Software Packages for the upload</li> </ul>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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**

<b>Activity</b>	<b>Management of Software Packages</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Packaging and Provision		
<b>Brief Description</b>	Management of Software Packages		
<b>Description</b>	<p>Once Software Packages have been created, they are generally ready to be deployed to dedicated machines (Adaptive ECUs) in the field.</p> <p>In order to do so, the Software Package may be stored, e.g., into a repository of packages located on a Back-end server.</p> <p>The management of this repository of the Software Packages may be supported by means of data bases.</p> <p>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.</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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)

<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Produces	<a href="#">Back-end Server</a>	1..*	<p>Software Packages are stored into a repository of Software Packages.</p> <p>In addition, update of a common data base of available Software Packages including their history.</p>

**Table 2.28: Management of Software Packages**

<b>Activity</b>	<b>Provision of Software Packages for machines in the field</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Packaging and Provision		
<b>Brief Description</b>	Provision of Software Packages		
<b>Description</b>	<p>Present the Software Packages in a way, that the UCM of machines are able to access the respective Software Packages.</p> <p>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.</p> <p>The handling of a concrete upload procedure is specified by diagnostic standards to some extent. 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.</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Back-end Server</a>	1	Status quo of the presentation layer of the Back-end Server
Produces	<a href="#">Back-end Server</a>	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

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

<b>Role</b>	<b>Tier 1</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Common Elements::Roles		
<b>Brief Description</b>	Direct (major) suppliers of parts to OEMs		
<b>Description</b>	Tier 1 companies are direct (major) suppliers of parts to OEMs.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Performs	Create Execution 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

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

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

<b>Task Definition</b>	<b>Select or define Data Types for Adaptive Platform</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Interface Definition::Tasks		
<b>Brief Description</b>	Define a set of AP data types for a specific project, which are not already defined by Autosar.		
<b>Description</b>	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.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Autosar AP Standard Package</a>	0..1	Use standardized elements (e.g. data types, compu methods) to create the corresponding elements of the specific project.
Produces	<a href="#">AP Data Types</a>	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

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

**Table 3.5: Define Service Interfaces**

### 3.2.1.3 Aggregate Service Interfaces

<i>Task Definition</i>	<b>Aggregate Service Interfaces</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Interface Definition::Tasks		
<b>Brief Description</b>	Aggregate service interfaces to a coarse-grained service interface.		
<b>Description</b>	<p>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.</p> <p>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.</p>		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumes	<a href="#">Service Interface Description</a>	0..*	Fine-grained service interfaces
Produces	<a href="#">Service Interface Description</a>	0..*	Coarse-grained service interfaces
Produces	<a href="#">Service Interface Mapping</a>	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

<b>Deliverable</b>	<b>Autosar AP Standard Package</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Interface Definition::Work Products		
<b>Brief Description</b>	Package with standardized AUTOSAR elements for the Adaptive Platform.		
<b>Description</b>	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.		
<b>Kind</b>	AUTOSAR XML		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumed by	<a href="#">Develop Adaptive Platform-level Software</a>	0..1	In case standardized service interfaces are used for adaptive platform-level software
Consumed by	<a href="#">Develop a Service Interface Description</a>	0..1	Optional input for defining data types and service interfaces for the adaptive platform
Consumed by	<a href="#">Select or define Data Types for Adaptive Platform</a>	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

<b>Artifact</b>	<b>AP Data Types</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Interface Definition::Work Products		
<b>Brief Description</b>	Definition of data types for the Adaptive Platform		
<b>Description</b>	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.		
<b>Kind</b>	AUTOSAR XML		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Produced by	<a href="#">Select or define Data Types for Adaptive Platform</a>	1..*	Defined AP Data Types for a specific project
Consumed by	<a href="#">Define Service Interfaces</a>	1..*	Used for specifying DataElements in service interfaces

**Table 3.8: AP Data Types**

### 3.2.2.3 Service Interface Description

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

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

**Table 3.9: Service Interface Description**

### 3.2.2.4 Service Interface Mapping

<i>Deliverable</i>	<b>Service Interface Mapping</b>		
<i>Package</i>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Interface Definition::Work Products		
<i>Brief Description</i>	Mapping from fine-grained service interfaces to coarse-grained service interface.		
<i>Description</i>	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.		
<i>Kind</i>	AUTOSAR XML		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produced by	<a href="#">Aggregate Service Interfaces</a>	0..*	Mapping between fine-grained service and coarse-grained service interfaces
Produced by	<a href="#">Develop a Service Interface Description</a>	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

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

**Table 3.11: Map Method**

#### 3.3.1.2 Map Event

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

**Table 3.12: Map Event**

### 3.3.1.3 Map Field

<b>Task Definition</b>	<b>Map Field</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
<b>Brief Description</b>	Map Field		
<b>Description</b>	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		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Client Server Interface Description</a>	1..*	The descriptions of Client Server Interfaces of CP
Consumes	<a href="#">Sender Receiver Interface Description</a>	1..*	The descriptions of Sender Receiver Interfaces of CP
Consumes	<a href="#">Service Interface Description</a>	1..*	Description of the Service Interfaces which communicate to CP in a service-oriented manner
Produces	<a href="#">Service Interface Mapping for Service Oriented Communication</a>	1..*	Service Interface Mappings

**Table 3.13: Map Field**

### 3.3.1.4 Map Fire and Forget



<b>Task Definition</b>	<b>Map Fire and Forget</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
<b>Brief Description</b>	Map Fire and Forget		
<b>Description</b>	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		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Service Interface Description</a>	1..*	Description of the Service Interface which communicates to CP in a service-oriented manner
Consumes	<a href="#">Sender Receiver Interface Description</a>	0..*	The descriptions of Sender Receiver Interfaces of CP
Consumes	<a href="#">Trigger Interface Description</a>	0..*	The descriptions of Trigger Interfaces
Produces	<a href="#">Service Interface Mapping for Service Oriented Communication</a>	1..*	Service Interface Mappings

**Table 3.14: Map Fire and Forget**

### 3.3.1.5 Map SignalBasedMethod to ISignalTriggerings

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

**Table 3.15: Map SignalBasedMethodDeployment to ISignalTriggerings**

### 3.3.1.6 Map SignalBasedEvent to ISignalTriggerings

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

**Table 3.16: Map SignalBasedEventDeployment to ISignalTriggerings**

### 3.3.1.7 Map SignalBasedField to ISignalTriggerings

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

**Table 3.17: Map SignalBasedFieldDeployment to ISignalTriggerings**

### 3.3.1.8 Map ServiceInstance to PortPrototype

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

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produces	<a href="#">Service Instance To Signal Mapping</a>	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

<i>Deliverable</i>	<b>Client Server Interface Description</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Work Products		
<b>Brief Description</b>	Client Server Interface Description		
<b>Description</b>	This represents the particular description of a ClientServerInterface of the Classic Platform.		
<b>Kind</b>	AUTOSAR XML		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumed by	<a href="#">Map Field</a>	1..*	The descriptions of Client Server Interfaces of CP
Consumed by	<a href="#">Map Method</a>	1..*	The descriptions of Client Server Interfaces of CP
Consumed by	<a href="#">Design service oriented communication between Classic and Adaptive Platform</a>	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

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

<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumed by	<a href="#">Design service oriented communication between Classic and Adaptive Platform</a>	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	<a href="#">Map Fire and Forget</a>	0..*	The descriptions of Sender Receiver Interfaces of CP

**Table 3.20: Sender Receiver Interface Description**

### 3.3.2.3 Trigger Interface Description

<b>Deliverable</b>	<b>Trigger Interface Description</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Work Products		
<b>Brief Description</b>	Trigger Interface Description		
<b>Description</b>	This represents the particular description of the Trigger Interface of the Classic Platform.		
<b>Kind</b>	AUTOSAR XML		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumed by	<a href="#">Design service oriented communication between Classic and Adaptive Platform</a>	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	<a href="#">Map Fire and Forget</a>	0..*	The descriptions of Trigger Interfaces

**Table 3.21: Trigger Interface Description**

### 3.3.2.4 Service Interface Mapping Set

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

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Aggregates	<a href="#">Service Interface Mapping for Service Oriented Communication</a>	1..*	

**Table 3.22: Service Interface Mapping Set**

### 3.3.2.5 Service Interface Mapping for Service Oriented Communication

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

**Table 3.23: Service Interface Mapping for Service Oriented Communication**

### 3.3.2.6 System Description

<b>Deliverable</b>	<b>System Description</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
<b>Brief Description</b>	Partial Extract of a System		
<b>Description</b>	<p>Generic deliverable for defining a System. It is used in different roles within the methodology.</p> <p>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.</p> <p>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".</p> <p>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.</p>		
<b>Kind</b>	Delivered		
<b>Extended by</b>	Abstract System Description, System Configuration Description, System Constraint Description, System Extract		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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..*	

<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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 (Signal BasedService InterfaceDeployment)	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 SignalBased EventDeployment to ISignal Triggerings	1	The System Description based on the System Template on the AUTOSAR classic platform
Consumed by	Map SignalBased FieldDeployment to ISignal Triggerings	1	The System Description based on the System Template on the AUTOSAR classic platform
Consumed by	Map SignalBased MethodDeployment 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

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

**Table 3.25: Service Instance To Signal Mapping Set**

### 3.3.2.8 Service Instance To Signal Mapping

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

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

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

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

**Table 3.29: Machine Design**

## 3.5 Diagnostic Mapping

### 3.5.1 Tasks

#### 3.5.1.1 Map Diagnostic Data

<b>Task Definition</b>	<b>Map Diagnostic Data</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Tasks		
<b>Brief Description</b>	Mapping between a diagnostic data element and an event or field		
<b>Description</b>	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].		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Diagnostic Machine Extract</a>	1	All available diagnostic information at the design time
Consumes	<a href="#">Service Interface Description</a>	1..*	Collection of service interfaces. Service interfaces can consist of events, methods and fields.
Consumes	<a href="#">Software Component Description for Adaptive Platform</a>	1..*	Description of a software component for the Adaptive Platform with all its ports, available at design time.
Produces	<a href="#">Diagnostic Mapping</a>	1	One diagnostic data mapping

**Table 3.30: Map Diagnostic Data**

### 3.5.1.2 Map Diagnostic Enable Condition to Ports

<b>Task Definition</b>	<b>Map Diagnostic Enable Condition to Port(s)</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Tasks		
<b>Brief Description</b>	Mapping of a diagnostic enable condition to one or many service ports		
<b>Description</b>	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]		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Diagnostic Machine Extract</a>	1	All available diagnostic information at the design time
Consumes	<a href="#">Software Component Description for Adaptive Platform</a>	1..*	Description of software component for the Adaptive Platform with all their (service) ports, known at design time.
Produces	<a href="#">Diagnostic Mapping</a>	1	One diagnostic EnableConditionToPorts mapping

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

### 3.5.1.3 Map Diagnostic Event to Ports

<b>Task Definition</b>	<b>Map Diagnostic Event to Port(s)</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Tasks		
<b>Brief Description</b>	Mapping of a diagnostic event to one or many service ports		
<b>Description</b>	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].		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Diagnostic Machine Extract</a>	1	All available diagnostic information at the design time
Consumes	<a href="#">Software Component Description for Adaptive Platform</a>	1..*	Description of software component for the Adaptive Platform with all their (service) ports, known at design time.
Produces	<a href="#">Diagnostic Mapping</a>	1	One diagnostic EventToPort mapping

**Table 3.32: Map Diagnostic Event to Port(s)**

### 3.5.1.4 Map Diagnostic Storage Condition to Ports

<b>Task Definition</b>	<b>Map Diagnostic Storage Condition to Port(s)</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Tasks		
<b>Brief Description</b>	Mapping of a diagnostic storage condition to one or many service ports		
<b>Description</b>	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]		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Diagnostic Machine Extract</a>	1	All available diagnostic information at the design time
Consumes	<a href="#">Software Component Description for Adaptive Platform</a>	1..*	Description of software component for the Adaptive Platform with all their (service) ports, known at design time.
Produces	<a href="#">Diagnostic Mapping</a>	1	One diagnostic StorageConditionToPorts mapping

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

### 3.5.1.5 Map Diagnostic Software Mapping

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

**Table 3.34: Diagnostic Software Mapping**

### 3.5.1.6 Map Diagnostic Operation Cycle to Ports

<b>Task Definition</b>	<b>Map Diagnostic Operation Cycle to Port(s)</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Tasks		
<b>Brief Description</b>	Mapping of a diagnostic operation cycle to one or many service ports		
<b>Description</b>	<p>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].</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Diagnostic Machine Extract</a>	1	All available diagnostic information at the design time
Consumes	<a href="#">Software Component Description for Adaptive Platform</a>	1..*	Description of software component for the Adaptive Platform with all their (service) ports, known at design time.
Produces	<a href="#">Diagnostic Mapping</a>	1	One diagnostic OperationCycleToPorts mapping

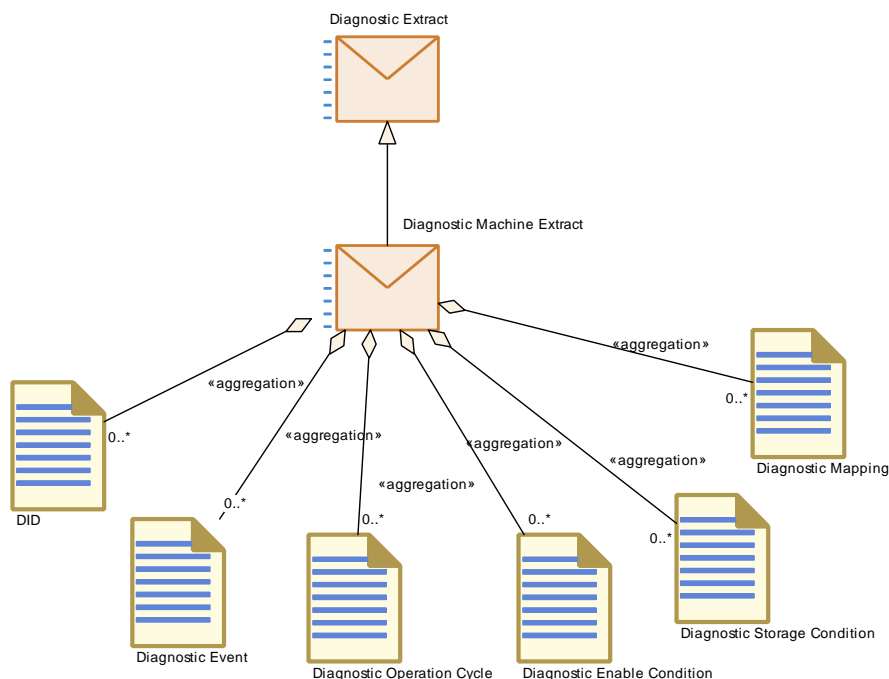
**Table 3.35: Map Diagnostic Operation Cycle to Port(s)**

### 3.5.1.7 Associate a DiagnosticMapping with a ProcessDesign

<b>Task Definition</b>	<b>Associate DiagnosticMapping with ProcessDesign</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Tasks		
<b>Brief Description</b>	Associate one DiagnosticMapping with one ProcessDesign		
<b>Description</b>	<p>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.</p> <p>This assignment may be done in a final extra step, represented by this task.</p> <p>To accommodate for this potential modeling, the reference from a diagnostic mapping to ProcessDesign has been decorated by stereotype "atpSplittable".</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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

<b>Deliverable</b>	<b>Diagnostic Machine Extract</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Work Products		
<b>Brief Description</b>	Diagnostic information of a Machine		
<b>Description</b>	This deliverable contains diagnostic information of a Machine.		
<b>Kind</b>	AUTOSAR XML		
<b>Extends</b>	Diagnostic Extract		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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

<b>Artifact</b>	<b>DID</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Work Products		
<b>Brief Description</b>			
<b>Description</b>	<p>This represents the definition of a diagnostic data identifier.</p> <p>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.</p>		
<b>Kind</b>			
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Aggregated by	Diagnostic Machine Extract	0..*	

**Table 3.38: DID**

### 3.5.2.3 Diagnostic Enable Condition

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

**Table 3.39: Diagnostic Enable Condition**

### 3.5.2.4 Diagnostic Event

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

**Table 3.40: Diagnostic Event**



### 3.5.2.5 Diagnostic Mapping

<b>Artifact</b>	<b>Diagnostic Mapping</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Diagnostic Mapping::Work Products		
<b>Brief Description</b>	Diagnostic Mappings		
<b>Description</b>	<p>This represents the mapping of information related to the diagnostic protocol content and the application software.</p> <p>In detail, it contains the results of the following tasks:</p> <ul style="list-style-type: none"> <li>• DiagnosticServiceDataMapping</li> <li>• DiagnosticServiceSwMapping</li> <li>• DiagnosticEventPortMapping</li> <li>• DiagnosticOperationCyclePortMapping</li> <li>• DiagnosticEnableConditionPortMapping</li> <li>• DiagnosticStorageConditionPortMapping</li> </ul>		
<b>Kind</b>	AUTOSAR XML		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Aggregated by	<a href="#">Diagnostic Machine Extract</a>	0..*	
Produced by	<a href="#">Diagnostic Software Mapping</a>	1	One diagnostic software mapping
Produced by	<a href="#">Map Diagnostic Data</a>	1	One diagnostic data mapping
Produced by	<a href="#">Map Diagnostic Enable Condition to Port(s)</a>	1	One diagnostic EnableConditionToPorts mapping
Produced by	<a href="#">Map Diagnostic Event to Port(s)</a>	1	One diagnostic EventToPort mapping
Produced by	<a href="#">Map Diagnostic Operation Cycle to Port(s)</a>	1	One diagnostic OperationCycleToPorts mapping
Produced by	<a href="#">Map Diagnostic Storage Condition to Port(s)</a>	1	One diagnostic StorageConditionToPorts mapping
Produced by	<a href="#">Associate DiagnosticMapping with Process Design</a>	1..*	fully: The linkage between the diagnostic mappings and the corresponding ProcessDesigns
Produced by	<a href="#">Design Diagnostic Mapping</a>	1..*	partially: The diagnostic mapping for a Machine, except the linkage between the mappings and the corresponding ProcessDesigns
Consumed by	<a href="#">Associate DiagnosticMapping with Process Design</a>	1..*	The diagnostic mapping for a Machine, except the linkage between the mappings and the corresponding ProcessDesigns

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
----------------------	------------------------	-------------	-------------

**Table 3.41: Diagnostic Mapping**

### 3.5.2.6 Diagnostic Operation Cycle

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

**Table 3.42: Diagnostic Operation Cycle**

### 3.5.2.7 Diagnostic Storage Condition

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

**Table 3.43: Diagnostic Storage Condition**

## 3.6 Adaptive Application

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

### 3.6.1 Tasks

#### 3.6.1.1 Generate Header Files for Service Interfaces

<b>Task Definition</b>	<b>Generate Header Files for Service Interfaces</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
<b>Brief Description</b>	Generate header files for service interfaces with proxies and skeletons		
<b>Description</b>	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.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Service Interface Description</a>	1..*	For all service interfaces header files are generated.
Produces	<a href="#">Header Files for Service Interfaces</a>	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

<b>Task Definition</b>	<b>Design Software Component for Adaptive Platform</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
<b>Brief Description</b>	Design a software component with ports that implement service interfaces.		
<b>Description</b>	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.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Performed by	<a href="#">Tier 2</a>	1	Application Software Designer: The design of software components will probably be performed by an Application Software Designer of a Tier 2 company
Consumes	<a href="#">Service Interface Description</a>	1..*	All service interfaces that shall be implemented by the software component
Produces	<a href="#">Software Component Description for Adaptive Platform</a>	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

<b>Task Definition</b>	<b>Implement Software Component Functionality</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
<b>Brief Description</b>	Implement the core functionality of the software component.		
<b>Description</b>	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.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Header Files for Service Interfaces</a>	1..*	Proxy and skeleton header files are the basis for implementing the software component
Consumes	<a href="#">Software Component Description for Adaptive Platform</a>	1..*	The software component model as input for the implementation of the software component.
Produces	<a href="#">Software Component Source Code</a>	1	The source code of the software component

**Table 3.46: Implement Software Component Functionality**

### 3.6.1.4 Compile Software Component

<b>Task Definition</b>	<b>Compile Software Component</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
<b>Brief Description</b>	Compile the software component in order to produce object code.		
<b>Description</b>	<p>Compile the software component together with the header files for service interfaces.</p> <p>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.</p> <p>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.</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Build Chain Configuration</a>	1	Settings used for compiling the software component
Consumes	<a href="#">Software Component Source Code</a>	1	Source code of the software component for compilation
Consumes	<a href="#">Header Files for Service Interfaces</a>	1..*	Used header files of the software component for compilation
Consumes	<a href="#">Middleware Library Header Files</a>	0..*	Library header files needed for compiling the software components
Produces	<a href="#">Software Component Object Code</a>	1	Object code of the software component after compilation

**Table 3.47: Compile Software Component**

### 3.6.1.5 Develop Main Function

<i>Task Definition</i>	<b>Develop Main Function</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
<b>Brief Description</b>	Develop the main function for one executable.		
<b>Description</b>	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.		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumes	<a href="#">Software Component Source Code</a>	1..*	Scheduling and communication of several software components within one executable is defined
Produces	<a href="#">Main Function</a>	1	One main function per executable

**Table 3.48: Develop Main Function**

### 3.6.1.6 Configure Serialization for Adaptive Platform

<i>Task Definition</i>	<b>Configure Serialization for Adaptive Platform</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
<b>Brief Description</b>	Define serialization properties for the Adaptive Platform		
<b>Description</b>	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.		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumes	<a href="#">Service Interface Description</a>	0..1	Optional if you only configure default values for the serialization
Produces	<a href="#">Serialization Configuration</a>	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

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

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumes	<a href="#">Service Interface Description</a>	1..*	Service interfaces that are implemented by the software components are needed for generating the serialization code
Produces	<a href="#">Serialization Source Code</a>	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

<i>Task Definition</i>	<b>Implement Service Proxies and Skeletons</b>		
<i>Package</i>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Tasks		
<i>Brief Description</i>	Implement service proxies and skeletons for an Adaptive Platform		
<i>Description</i>	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.		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumes	<a href="#">Header Files for Service Interfaces</a>	1..*	Header files contain proxies and skeletons to be implemented
Consumes	<a href="#">Serialization Configuration</a>	1..*	Serialization of data is needed for implementing service proxies and skeletons
Produces	<a href="#">Implemented Proxies and Skeletons</a>	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

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

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumes	<a href="#">Platform Object Code</a>	0..*	Platform modules to be linked together to one executable
Consumes	<a href="#">Software Component Object Code</a>	0..*	Software component to be linked together to one executable
Produces	<a href="#">Executable Application</a>	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

<i>Deliverable</i>	<b>Header Files for Service Interfaces</b>		
<i>Package</i>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
<i>Brief Description</i>	Header files generated for service interfaces		
<i>Description</i>	<p>The generated header files of service interfaces consist of</p> <ul style="list-style-type: none"> <li>• proxy header files for service discovery and method invocation as well as event subscription and reception</li> <li>• skeleton header files for method calls and event publishing</li> </ul> <p>The header files are the basis for implementing the functionality of a software component.</p>		
<i>Kind</i>	Source Code		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produced by	<a href="#">Generate Header Files for Service Interfaces</a>	1..*	One proxy header file and one skeleton header file per service interface are generated.
Consumed by	<a href="#">Compile Software Component</a>	1..*	Used header files of the software component for compilation
Consumed by	<a href="#">Implement Service Proxies and Skeletons</a>	1..*	Header files contain proxies and skeletons to be implemented
Consumed by	<a href="#">Implement Software Component Functionality</a>	1..*	Proxy and skeleton header files are the basis for implementing the software component
Consumed by	<a href="#">Integrate Software</a>	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



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



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

**Table 3.55: Build Chain Configuration**

### 3.6.2.4 Software Component Source Code

<b>Deliverable</b>	<b>Software Component Source Code</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
<b>Brief Description</b>	Source code of the core functionality of a software component		
<b>Description</b>	<p>This deliverable contains the source code of the core functionality of a software component. The deliverable includes documentation of the software component.</p> <p>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.</p>		
<b>Kind</b>	Source Code		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Produced by	<a href="#">Implement Software Component Functionality</a>	1	The source code of the software component
Produced by	<a href="#">Develop Adaptive Application Software</a>	0..*	Software components as source code
Consumed by	<a href="#">Compile Software Component</a>	1	Source code of the software component for compilation
Consumed by	<a href="#">Develop Main Function</a>	1..*	Scheduling and communication of several software components within one executable is defined
Consumed by	<a href="#">Integrate Software</a>	0..*	Source code for application-level executable

**Table 3.56: Software Component Source Code**

### 3.6.2.5 Software Component Object Code

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

**Table 3.57: Software Component Object Code**

### 3.6.2.6 Serialization Configuration for Adaptive Platform

<b>Deliverable</b>	<b>Serialization Configuration</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
<b>Brief Description</b>	Configuration of serialization of the data in the service interface		
<b>Description</b>	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.		
<b>Kind</b>	AUTOSAR XML		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Produced by	<a href="#">Configure Serialization for Adaptive Platform</a>	1..*	Serialization properties for the service interfaces
Consumed by	<a href="#">Generate Serialization Code for Adaptive Platform</a>	1..*	Configuration settings are the basis for generating the serialization code.
Consumed by	<a href="#">Implement Service Proxies and Skeletons</a>	1..*	Serialization of data is needed for implementing service proxies and skeletons

**Table 3.58: Serialization Configuration**

### 3.6.2.7 Serialization Source Code

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

**Table 3.59: Serialization Source Code**

### 3.6.2.8 Implemented Service Proxies and Skeletons

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

**Table 3.60: Implemented Proxies and Skeletons**

### 3.6.2.9 Main Function

<b>Deliverable</b>	<b>Main Function</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
<b>Brief Description</b>	Main function of executable application		
<b>Description</b>	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.		
<b>Kind</b>	Source Code		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Produced by	<a href="#">Develop Adaptive Application Software</a>	1	One main function per executable is produced
Produced by	<a href="#">Develop Adaptive Platform-level Software</a>	1	Main function for platform-level executable

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produced by	Develop Main Function	1	One main function per 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

<i>Deliverable</i>	<b>Executable Application</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
<b>Brief Description</b>	Executable application containing several software components		
<b>Description</b>	<p>The executable application, or just executable, can contain an arbitrary hierarchy of software components. The software components contain the functionality of the executable.</p> <p>Executables can be of category application-level or platform-level.</p>		
<b>Kind</b>	Executable		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
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 Execution 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].

<i>Task Definition</i>	<b>Define ECU Description</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::System::Tasks		
<b>Brief Description</b>	Define a particular ECU's resources.		
<b>Description</b>	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 HW Pins 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.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Performed by	System Engineer	1	
Produces	<a href="#">ECU Resources Description</a>	1..*	Description of the ECU

**Table 3.63: Define ECU Description**

##### 3.7.1.2 Describe Available HW Resources

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

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumes	<a href="#">ECU Resources Description</a>	1	Definition of available HW resources for the Machine based on the description of the ECU
Produces	<a href="#">Machine Manifest</a>	0..1	Available hardware resources of machine

**Table 3.64: Describe Available HW Resources**

### 3.7.1.3 Define Machine States

<i>Task Definition</i>	<b>Define Machine States</b>		
<i>Package</i>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
<i>Brief Description</i>	Define additional states of the machine		
<i>Description</i>	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.		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produces	<a href="#">Machine States</a>	0..1	States defined for the Machine

**Table 3.65: Define Machine States**

### 3.7.1.4 Define Function Groups

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

**Table 3.66: Define Function Groups**

### 3.7.1.5 Define State Timeouts

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

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumes	<a href="#">Machine States</a>	1	Machine States of a Machine
Produces	<a href="#">PerStateTimeouts</a>	0..1	PerState Timeouts defined for a Machine

**Table 3.67: Define State Timeouts**

### 3.7.1.6 Map Process To Machine

<i>Task Definition</i>	<b>Map Process To Machine</b>		
<i>Package</i>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
<i>Brief Description</i>	Map processes to a particular Machine		
<i>Description</i>	Map processes to a particular Machine.		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumes	<a href="#">Process</a>	1	Description of a dedicated Process
Produces	<a href="#">ProcessToMachineMapping</a>	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

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

**Table 3.69: Configure OS for Adaptive Platform**

### 3.7.1.8 Configure Log and Trace module

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

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumes	<a href="#">Machine Design</a>	0..1	Configuration settings of the network connections and service discovery network exchange of a Machine
Produces	<a href="#">Machine Manifest</a>	1	Configuration of the Log and Trace module

**Table 3.70: Configure Log and Trace module**

### 3.7.1.9 Configure DoIP

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

**Table 3.71: Configure DoIP**

### 3.7.1.10 Configure NM module

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



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

**Table 3.73: Middleware Library Header Files**

### 3.7.2.2 Middleware Libraries

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

<b>Artifact</b>	<b>ECU Resources Description</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::System::Work products		
<b>Brief Description</b>	Definition of the resources available on an ECU.		
<b>Description</b>	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		
<b>Kind</b>	AUTOSAR XML		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>

<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Aggregated by	Complete ECU Description	1	
Produced by	<a href="#">Define ECU Description</a>	1..*	Decription of the ECU
Consumed by	<a href="#">Describe Available HW Resources</a>	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	<a href="#">Define and configure machine</a>	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

<b>Deliverable</b>	<b>Configured Machine on Adaptive ECU</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Work Products		
<b>Brief Description</b>	Configured Adaptive Platform instance		
<b>Description</b>	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.		
<b>Kind</b>	Custom		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Produced by	<a href="#">Set Up Initial Machine</a>	1	Machine is configured and software can now be deployed

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
----------------------	------------------------	-------------	-------------

**Table 3.76: Configured Machine on Adaptive ECU**

### 3.7.2.5 Machine Manifest

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

**Table 3.77: Machine Manifest**

### 3.7.2.6 Platform Object Code

<b>Deliverable</b>	<b>Platform Object Code</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Platform::Work Products		
<b>Brief Description</b>	Object code of platform-level software		
<b>Description</b>	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.		
<b>Kind</b>	Object Code		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Produced by	<a href="#">Develop Adaptive Platform-level Software</a>	1..*	Object code of platform module
Consumed by	<a href="#">Build Executable Application</a>	0..*	Platform modules to be linked together to one executable
Consumed by	<a href="#">Integrate Software</a>	0..*	Object code for platform-level executable

**Table 3.78: Platform Object Code**

### 3.7.2.7 Operating System for Adaptive Platform

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

**Table 3.79: Operating System for Adaptive Platform**

### 3.7.2.8 Process to Machine Mapping

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

**Table 3.80: ProcessToMachineMapping**

### 3.7.2.9 Function Groups

<b>Artifact</b>	<b>Function Groups</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Define and Configure Machine::Machine Configuration		
<b>Brief Description</b>			
<b>Description</b>	This artifact contains the configuration of function groups of a machine.		
<b>Kind</b>			
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Produced by	<a href="#">Define Function Groups</a>	0..1	Function groups defined for the Machine
Consumed by	<a href="#">Define State Time-outs</a>	1	Function Groups of a Machine

**Table 3.81: Function Groups**

### 3.7.2.10 Machine States

<b>Artifact</b>	<b>Machine States</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment::Define and Configure Machine::Machine Configuration		
<b>Brief Description</b>			
<b>Description</b>	This artifact contains the configuration of machine states of a machine.		
<b>Kind</b>			
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Produced by	<a href="#">Define Machine States</a>	0..1	States defined for the Machine
Consumed by	<a href="#">Define State Time-outs</a>	1	Machine States of a Machine

**Table 3.82: Machine States**

### 3.7.2.11 PerState Timeouts

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

**Table 3.83: PerStateTimeouts**

## 3.8 Execution Manifest

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

### 3.8.1 Tasks

#### 3.8.1.1 Define Process

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

**Table 3.84: Define Process**

#### 3.8.1.2 Define Startup Configuration

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

**Table 3.85: Define Startup Configuration**

### 3.8.1.3 Define Execution Dependencies

<b>Task Definition</b>	<b>Define Execution Dependencies</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Execution Manifest::Tasks		
<b>Brief Description</b>	Define execution dependencies to other processes		
<b>Description</b>	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.		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Machine Manifest</a>	1	Execution dependencies are defined per machine mode.
Consumes	<a href="#">Process</a>	1	Execution dependencies defined for one process
Produces	<a href="#">Mode-dependent Startup Configuration</a>	1..*	Execution dependencies of a process for each mode

**Table 3.86: Define Execution Dependencies**

### 3.8.1.4 Associate Process with Process Design

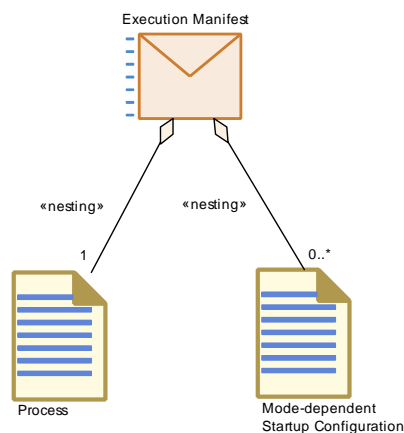
<b>Task Definition</b>	<b>Associate Process with ProcessDesign</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Execution Manifest::Tasks		
<b>Brief Description</b>			
<b>Description</b>	Establish a 1:1 relation between a actual process and its placeholder during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

Relation Type	Related Element	Mul.	Note
Consumes	<a href="#">Process Design</a>	1..*	ProcessDesign as placeholder during design time for the real Process
Produces	<a href="#">Process</a>	1..*	A Process references a respective ProcessDesign

**Table 3.87: Associate Process with ProcessDesign**

### 3.8.2 Work Products

#### 3.8.2.1 Execution Manifest



**Figure 3.2: Structure of Deliverable [Execution Manifest](#)**

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



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

**Table 3.88: Execution Manifest**

### 3.8.2.2 Process

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

**Table 3.89: Process**

### 3.8.2.3 Mode-dependent Startup Configuration

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

**Table 3.90: Mode-dependent Startup Configuration**

### 3.8.2.4 Process Design

<b>Artifact</b>	<b>Process Design</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Common Design Artifacts::Work Products		
<b>Brief Description</b>	Proxy for a Process at design time		
<b>Description</b>	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.		
<b>Kind</b>	AUTOSAR XML		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumed by	<a href="#">Associate DiagnosticMapping with Process Design</a>	1..*	All dedicated ProssesDesigns for a Machine
Consumed by	<a href="#">Associate Process with Process Design</a>	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

<b>Task Definition</b>	<b>Configure Service Interface Deployment</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Tasks		
<b>Brief Description</b>	Configure the binding of a Service Interface to a transport layer		
<b>Description</b>	<p>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.</p> <p>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.</p> <p>The User Defined service interface deployment can e.g. be used machine local IPC communication.</p> <p>The responsibility of the configuration of service interface deployment lies with the system responsible.</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Service Interface Description</a>	1	Deployment is configured for each service interface
Produces	<a href="#">Service Interface Deployment Configuration</a>	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

<b>Task Definition</b>	<b>Define and Configure Service Instance</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Tasks		
<b>Brief Description</b>	Define the service instances and configure their search or offer criteria		
<b>Description</b>	<p>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.</p> <p>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 be defined. The instance IDs need to be defined system-wide.</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	<a href="#">Service Interface Deployment Configuration</a>	1	Instances of service interfaces to be defined

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produces	<a href="#">Service Instance Configuration</a>	1..*	Service instances and their configuration defined

**Table 3.93: Define and Configure Service Instance**

### 3.9.1.3 Define SOME/IP timing

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

<i>Task Definition</i>	<b>Map Service Instance to Port Prototype</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Tasks		
<b>Brief Description</b>	Define mapping of service instance to a port prototype		
<b>Description</b>	<p>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).</p>		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Performed by	<a href="#">Tier 1</a>	1	Software Integrator: This activity will probably be performed by a Software Integrator of a Tier 1 company
Consumes	<a href="#">Service Instance Configuration</a>	1	Service instances to be mapped to port prototypes
Consumes	<a href="#">Software Component Description for Adaptive Platform</a>	1	In case the service instances are mapped to ports of a software component

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produces	<a href="#">Service Instance To Port Prototype Mapping</a>	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

<i>Task Definition</i>	<b>Map Service Instance to Machine</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Tasks		
<b>Brief Description</b>	Define mapping of service instance to machine		
<b>Description</b>	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.		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumes	<a href="#">Machine Design</a>	1	Description of machine that the service instances shall be mapped to
Consumes	<a href="#">Service Instance Configuration</a>	1	Service instances to be mapped to machine
Produces	<a href="#">Service Instance To Machine Mapping</a>	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

<i>Artifact</i>	<b>Service Interface Deployment Configuration</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Work Products		
<b>Brief Description</b>	Deployment configuration for a service interface		
<b>Description</b>	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.		
<b>Kind</b>	AUTOSAR XML		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produced by	<a href="#">Configure Service Interface Deployment</a>	1	Configuration of binding of a service interface to a transport layer
Consumed by	<a href="#">Define and Configure Service Instance</a>	1	Instances of service interfaces to be defined

**Table 3.97: Service Interface Deployment Configuration**

### 3.9.2.2 Service Instance Configuration

<i>Artifact</i>	<b>Service Instance Configuration</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Work Products		
<b>Brief Description</b>	Definition and configuration of the service instances		
<b>Description</b>	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.		
<b>Kind</b>	AUTOSAR XML		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produced by	<a href="#">Define and Configure Service Instance</a>	1..*	Service instances and their configuration defined
Consumed by	<a href="#">Define SOME/IP Timing</a>	1	Timing for service instances to be defined
Consumed by	<a href="#">Map Service Instance to Machine</a>	1	Service instances to be mapped to machine
Consumed by	<a href="#">Map Service Instance to Port Prototype</a>	1	Service instances to be mapped to port prototypes

**Table 3.98: Service Instance Configuration**

### 3.9.2.3 Service Instance To Machine Mapping

<i>Artifact</i>	<b>Service Instance To Machine Mapping</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Work Products		
<b>Brief Description</b>			
<b>Description</b>	Service Instances shall be mapped to a Machine (to be more precise: to a communication connector of a Machine)		
<b>Kind</b>			
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produced by	<a href="#">Map Service Instance to Machine</a>	1	Mapping contributes to Service Instance Manifest

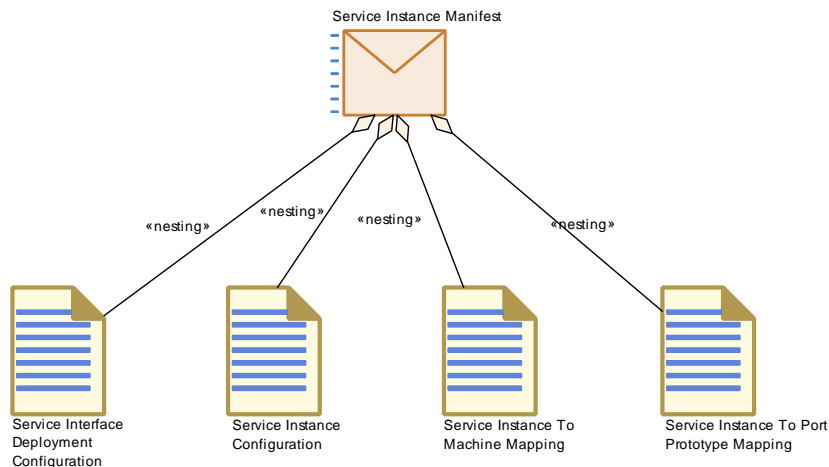
**Table 3.99: Service Instance To Machine Mapping**

### 3.9.2.4 Service Instance To Port Prototype Mapping

<b>Artifact</b>	<b>Service Instance To Port Prototype Mapping</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Work Products		
<b>Brief Description</b>			
<b>Description</b>	<p>Service Instances need to be mapped to Port Prototypes in instance context (the instance context includes process, executable, all nesting levels of the software composition and the port prototype).</p> <p>With this mapping it is possible to define how specific Port Prototypes are represented in the middleware in terms of service configuration.</p>		
<b>Kind</b>			
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Produced by	<a href="#">Map Service Instance to Port Prototype</a>	1	Mapping contributes to Service Instance Manifest

**Table 3.100: Service Instance To Port Prototype Mapping**

### 3.9.2.5 Service Instance Manifest



**Figure 3.3: Parts of the Service Instance Manifest**

<b>Deliverable</b>	<b>Service Instance Manifest</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Work Products		
<b>Brief Description</b>	Definition and configuration of a service instance		
<b>Description</b>	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.		
<b>Kind</b>	AUTOSAR XML		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Aggregates	<a href="#">Service Instance Configuration</a>	1	

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Aggregates	<a href="#">Service Instance To Machine Mapping</a>	1	
Aggregates	<a href="#">Service Instance To Port Prototype Mapping</a>	1	
Aggregates	<a href="#">Service Interface Deployment Configuration</a>	1	
Produced by	<a href="#">Define SOME/IP Timing</a>	1	Timing for service instances contributes to Service Instance Manifest
Produced by	<a href="#">Define and Configure Service Instances</a>	1..*	Contains all configuration settings for the service instance on a specific machine
Consumed by	<a href="#">Collect belonging (software) artifacts of Sub Software Clusters</a>	0..*	Several service instance manifests can be deployed
Consumed by	<a href="#">Create Software Package</a>	0..*	Several service instance manifests can be deployed
Consumed by	<a href="#">Identify necessary (software) artifacts</a>	0..*	Several service instance manifests can be deployed
Consumed by	<a href="#">Set Up Initial Machine</a>	0..*	All Service Instance Manifests needed to run the desired adaptive application (instances or Processes) on a Machine

**Table 3.101: 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](#)



<b>Task Definition</b>	<b>Create an initial Software Package Manifest</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Tasks		
<b>Brief Description</b>	Create an initial Software Package Manifest		
<b>Description</b>	<p>The main input for this step are the requirements of the OEM given by means of the Software Cluster Design.</p> <p>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.</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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 3.102: Create an initial Software Package Manifest**

### 3.10.1.2 Identify necessary (software) artifacts

<b>Task Definition</b>	<b>Identify necessary (software) artifacts</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Tasks		
<b>Brief Description</b>	Identify necessary artifacts		
<b>Description</b>	<p>Identify necessary (software) artifacts in order to build the Software Package, also with respect to their versions.</p> <p>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.</p> <p>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)</p>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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	Executable Application	0..*	Executables of deployed processes

<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	Execution Manifest	0..*	Several processes can be deployed
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 3.103: Identify necessary (software) artifacts**

### 3.10.1.3 Collect belonging (software) artifacts of Sub Software Clusters

<b>Task Definition</b>	<b>Collect belonging (software) artifacts of Sub Software Clusters</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Tasks		
<b>Brief Description</b>	Collect belonging artifacts		
<b>Description</b>	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		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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	Executable Application	0..*	Executables of deployed processes
Consumes	Execution Manifest	0..*	Several processes can be deployed
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 3.104: Collect belonging (software) artifacts of Sub Software Clusters**

### 3.10.1.4 Model dependencies between Software Clusters

<b>Task Definition</b>	<b>Model dependencies between Software Clusters of any category</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Tasks		
<b>Brief Description</b>	Model dependencies		
<b>Description</b>	<p>Thus, this activity describes the handling of dependencies by at least the following tasks:</p> <ul style="list-style-type: none"> <li>• Check, whether the dependencies between Software Clusters of the same or different categories, given by the respective SoftwareClusterDesign are still valid</li> <li>• Determine changes between the actual and required dependencies between Software Clusters of any category</li> <li>• If necessary, re-model the Software Package Manifest in accordance with the outcomes of the both tasks above</li> </ul>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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 3.105: Model dependencies between Software Clusters of any category**

### 3.10.1.5 Create installation instructions

<b>Task Definition</b>	<b>Create installation instructions</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Tasks		
<b>Brief Description</b>	Create installation instructions		
<b>Description</b>	<p>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.</p> <p>Thus, this activity may includes the tasks:</p> <ul style="list-style-type: none"> <li>• Specify installation instructions per Software Cluster (of any category)</li> <li>• Develop update campaigns (optional)</li> </ul>		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Consumes	Software Package Manifest	1	Software Package Manifest without or incomplete installation instructions

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produces	<a href="#">Software Package Manifest</a>	1	Software Package Manifest, enhanced by installation instruction

**Table 3.106: Create installation instructions**

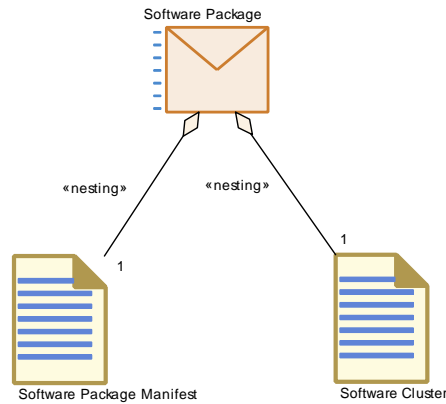
## 3.10.2 Work Products

### 3.10.2.1 Software Cluster Design

<b><i>Deliverable</i></b>	<b>Software Cluster Design</b>		
<b><i>Package</i></b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Work Products		
<b><i>Brief Description</i></b>	Software Cluster Design		
<b><i>Description</i></b>	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.		
<b><i>Kind</i></b>	AUTOSAR XML		
<b><i>Relation Type</i></b>	<b><i>Related Element</i></b>	<b><i>Mul.</i></b>	<b><i>Note</i></b>
Consumed by	<a href="#">Create Software Package</a>	1	Requirements of the OEM wrt. package structure and parameters given by means of the meta model element SoftwareClusterDesign.
Consumed by	<a href="#">Create an initial Software Package Manifest</a>	1	Requirements regarding Software Clusters by the OEM
Consumed by	<a href="#">Identify necessary (software) artifacts</a>	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 3.107: Software Cluster Design**

### 3.10.2.2 Software Package



**Figure 3.4: Parts of a Software Package**

<b>Deliverable</b>	<b>Software Package</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Work Products		
<b>Brief Description</b>	Container to deploy software artifacts to a machine		
<b>Description</b>	<p>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).</p> <p>In fact, a Software Package consists of two main parts:</p> <ul style="list-style-type: none"> <li>• a bundle of the actual software artifacts, referred to as Software Cluster</li> <li>• corresponding model data needed to control the upload and installation process of this Software Cluster executed by the UCM</li> </ul>		
<b>Kind</b>	Custom		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
Aggregates	<a href="#">Software Cluster</a>	1	
Aggregates	<a href="#">Software Package Manifest</a>	1	
Produced by	<a href="#">Compile the Software Package</a>	1	Compiled Software Package
Produced by	<a href="#">Create Software Package</a>	1	Software Package for deployment defined
Consumed by	<a href="#">Management of Software Packages</a>	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	<a href="#">Provide and manage Software Packages</a>	1..*	Deploy software on a Back-end server by means of Software Package

<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
----------------------	------------------------	-------------	-------------

**Table 3.108: Software Package**

### 3.10.2.3 Software Cluster

<i>Artifact</i>	<b>Software Cluster</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Work Products		
<b>Brief Description</b>	Software Cluster		
<b>Description</b>	<p>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), Execution 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.</p> <p>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.</p> <p>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.</p>		
<b>Kind</b>	Custom		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Consumed by	<a href="#">Manage the data base of Software Clusters (of any category)</a>	1..*	Store and manage software cluster within a repository

**Table 3.109: Software Cluster**

### 3.10.2.4 Software Package Manifest

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

<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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 3.110: Software Package Manifest**

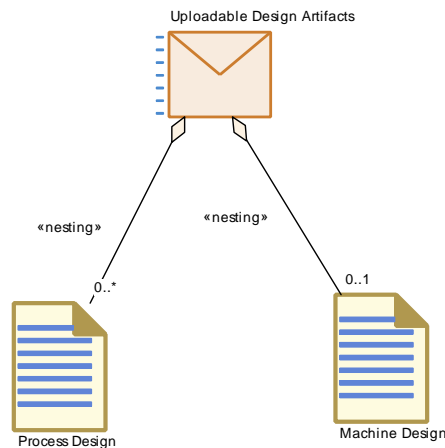
### 3.10.2.5 (Sub) Software Cluster Group

<b>Deliverable</b>	<b>(Sub) Software Cluster Group</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Work Products		
<b>Brief Description</b>	(Sub) Software Cluster Group		
<b>Description</b>	Basket to collect the (software) artifacts of a Sub Software Cluster		
<b>Kind</b>	Custom		
<b>Relation Type</b>	<b>Related Element</b>	<b>Mul.</b>	<b>Note</b>
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 3.111: (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		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Common Design Artifacts::Work Products		
<b>Brief Description</b>	Design artifacts needed needed to be uploaded to the Machine		
<b>Description</b>	Covers design artifacts, i.e., 'Machine Design' and 'Process Design', that are needed to be uploaded to the Machine in addition to the Manifests.		
<b>Kind</b>	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



<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
----------------------	------------------------	-------------	-------------

**Table 3.112: Uploadable Design Artifacts**

### 3.10.2.7 Back-end server

<i>Deliverable</i>	<b>Back-end Server</b>		
<b>Package</b>	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Work Products		
<b>Brief Description</b>	Repository of uploadable packages on a Back-end server		
<b>Description</b>	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.		
<b>Kind</b>	Custom		
<i>Relation Type</i>	<i>Related Element</i>	<i>Mul.</i>	<i>Note</i>
Produced by	<a href="#">Provision of Software Packages for machines in the field</a>	1	Organize the Back-end Server in accordance with the requirements of an OEM
Produced by	<a href="#">Management of Software Packages</a>	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	<a href="#">Provide and manage Software Packages</a>	1..*	Store uploadable packages (Software Packages) into a repository of a Back-end server
Consumed by	<a href="#">Provision of Software Packages for machines in the field</a>	1	Status quo of the presentation layer of the Back-end Server

**Table 3.113: Back-end Server**

## **A Change History**

### **A.1 Change History for AP 19-03**

#### **A.1.1 Added Constraints in 19-03**

none

#### **A.1.2 Changed Constraints in 19-03**

none

#### **A.1.3 Deleted Constraints in 19-03**

none

#### **A.1.4 Added Traceables in 19-03**

none

#### **A.1.5 Changed Traceables in 19-03**

none

#### **A.1.6 Deleted Traceables in 19-03**

none

### **A.2 Change History for AP 18-10**

#### **A.2.1 Added Constraints in 18-10**

none

#### **A.2.2 Changed Constraints in 18-10**

none

### A.2.3 Deleted Constraints in 18-10

none

### A.2.4 Added Traceables in 18-10

none

### A.2.5 Changed Traceables in 18-10

Number	Heading
[TR_AMETH_00004]	Creation of the <a href="#">Execution Manifest</a>
[TR_AMETH_00020]	Development of <a href="#">Platform Object Code</a>
[TR_AMETH_00026]	Definition of <a href="#">Execution Manifest</a>
[TR_AMETH_00031]	Setting up an initial machine
[TR_AMETH_00034]	Select the <a href="#">Operating System</a> for the Adaptive Platform

**Table A.1: Changed Traceables in 18-10**

### A.2.6 Deleted Traceables in 18-10

Number	Heading
[TR_AMETH_00211]	Pool <a href="#">Executables</a> together to form <a href="#">ExecutableGroups</a>

**Table A.2: Deleted Traceables in 18-10**

## A.3 Change History for AP 18-03

### A.3.1 Added Specification Items in AP 18-03

Number	Heading
[TR_AMETH_00211]	Pool <a href="#">Executables</a> together to form <a href="#">ExecutableGroups</a>
[TR_AMETH_00212]	Design a diagnostic mapping
[TR_AMETH_00213]	Relate diagnostic mappings to instances of <a href="#">Executables</a>
[TR_AMETH_00214]	Configuration of <a href="#">Platform Services</a>
[TR_AMETH_00215]	Configuration of <a href="#">Platform Foundation Modules</a>
[TR_AMETH_00216]	Map <a href="#">Processes</a> to a particular machine
[TR_AMETH_00217]	Definition of resources
[TR_AMETH_00218]	Create an initial <a href="#">Software Package Manifest</a>
[TR_AMETH_00219]	Collect all software artifacts that belong to a <a href="#">Software Cluster</a> , structure and model them
[TR_AMETH_00220]	Model dependencies between <a href="#">Software Clusters</a> of any category
[TR_AMETH_00221]	Develop installation instructions
[TR_AMETH_00222]	Create the <a href="#">Software Package</a>
[TR_AMETH_00223]	Manage the data base of <a href="#">Software Clusters</a> (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.3: Added specification items in AP 18-03**

### A.3.2 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.4: Changed specification items in AP 18-03**

### A.3.3 Deleted Specification Items in AP 18-03

Number	Heading
TR_AMETH_00032	Deploying the Software Package

**Table A.5: Deleted specification items in AP 18-03**

## A.4 Change History for AP 17-10

### A.4.1 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.6: Added specification items in AP 17-10**

### A.4.2 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.7: Changed specification items in AP 17-10**

### A.4.3 Deleted Specification Items in AP 17-10

Number	Heading
[TR_AMETH_00030]	Machine-driven and model-driven approach

**Table A.8: Deleted specification items in AP 17-10**

## A.5 Change History for AP 17-03

### A.5.1 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 <a href="#">Build Chain Configuration</a>
[TR_AMETH_00015]	Development without knowledge of the <a href="#">Build Chain Configuration</a>
[TR_AMETH_00016]	Development of serialization properties
[TR_AMETH_00017]	Implementation of service proxies and skeletons
[TR_AMETH_00018]	Building the <a href="#">Executable Application</a>
[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 <a href="#">Executable Application</a>
[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 <a href="#">Operating System for Adaptive Platform</a>
[TR_AMETH_00035]	Platform-level Software

**Table A.9: Added specification items in AP 17-03**

### A.5.2 Changed Specification Items in AP 17-03

N/A

### A.5.3 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 Execution 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 ExecutionManifest**



### 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**