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

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	17-10

Document Change History

Date	Release	Changed by	Description
2017-10-27	17-10	AUTOSAR Release Management	• Design of service oriented communication between CP and AP • Design of signal oriented communication between CP and AP • Deployment by means of `SoftwareCluster` • Removed concept of TransportLayerIndependentInstanceId
2017-03-31	17-03	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

Bibliography

[1] Methodology
AUTOSAR_TR_Methodology

[2] Requirements on Methodology
AUTOSAR_RS_Methodology

[3] Standardization Template
AUTOSAR_TPS_StandardizationTemplate

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

[5] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate

[6] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[7] Specification of ECU Resource Template
AUTOSAR_TPS_ECUResourceTemplate

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 added to or removed from the list of executables and may be started, executed or stopped, if available, 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 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 `task`s and `work product`s, 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. `ECU`s.

This document contains specification items in textual form that are distinguished from the rest of the text by a unique numerical ID, a headline, and the actual text starting after the ⌈ character and terminated by the ⌋ character. The conventions for requirements traceability follow [TPS_STDT_00080], see Standardization Template ([3]).

1.4 Methodology Concepts

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

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

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

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.

1.5 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_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]
[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_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_00084]	The AUTOSAR methodology shall relate templates to a distributed development process	[TR_AMETH_00027] [TR_AMETH_00028]
[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]
[RS_METH_00202]	The methodology shall explain how to develop an Adaptive Application	[TR_AMETH_00002] [TR_AMETH_00010] [TR_AMETH_00011] [TR_AMETH_00012] [TR_AMETH_00013] [TR_AMETH_00014] [TR_AMETH_00015] [TR_AMETH_00018] [TR_AMETH_00205] [TR_AMETH_00207] [TR_AMETH_00208] [TR_AMETH_00209] [TR_AMETH_00210]
[RS_METH_00203]	The methodology shall explain the high-level usage of the Manifest Specification	[TR_AMETH_00003] [TR_AMETH_00004] [TR_AMETH_00005] [TR_AMETH_00021] [TR_AMETH_00022] [TR_AMETH_00023] [TR_AMETH_00024] [TR_AMETH_00025] [TR_AMETH_00026] [TR_AMETH_00027] [TR_AMETH_00028] [TR_AMETH_00029] [TR_AMETH_00033]
[RS_METH_00204]	The methodology shall describe how to configure a machine for the Adaptive Platform	[TR_AMETH_00003] [TR_AMETH_00021] [TR_AMETH_00022] [TR_AMETH_00023] [TR_AMETH_00031]
[RS_METH_00205]	The methodology shall describe how to deploy software on the Adaptive Platform	[TR_AMETH_00006] [TR_AMETH_00031] [TR_AMETH_00032] [TR_AMETH_00206]
[RS_METH_00206]	The methodology shall explain how to configure the instances of services of a system	[TR_AMETH_00005] [TR_AMETH_00027] [TR_AMETH_00028] [TR_AMETH_00029] [TR_AMETH_00033]

[RS_METH_00207]	The methodology shall explain how to develop Platform Software for the Adaptive Platform	[TR_AMETH_00017] [TR_AMETH_00019] [TR_AMETH_00020] [TR_AMETH_00034] [TR_AMETH_00035]

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].

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 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 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

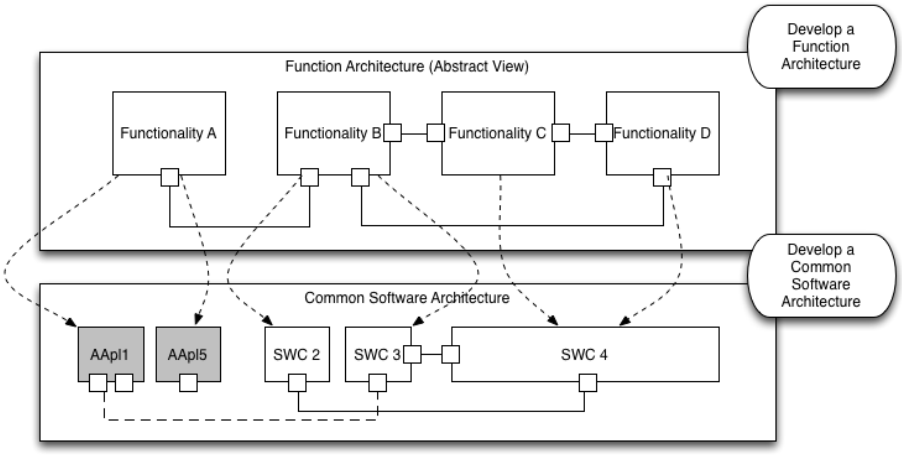

Figure 2.1: From the `Function Architecture` to a `Common Software Architecture`

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

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

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

This activity is optional. ⌋(*RS_METH_00032*)

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

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

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

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

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

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

This activity is optional. ⌋(*RS_METH_00032*)

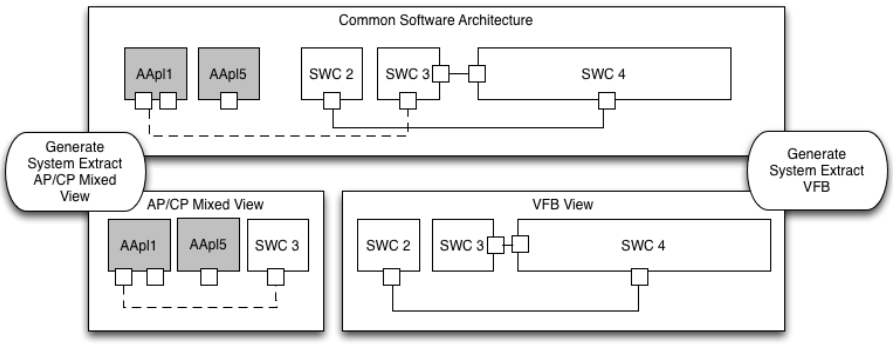

Figure 2.2: Views of subsystems enable to emphasize on relevant aspects

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

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

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

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

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

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

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

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

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

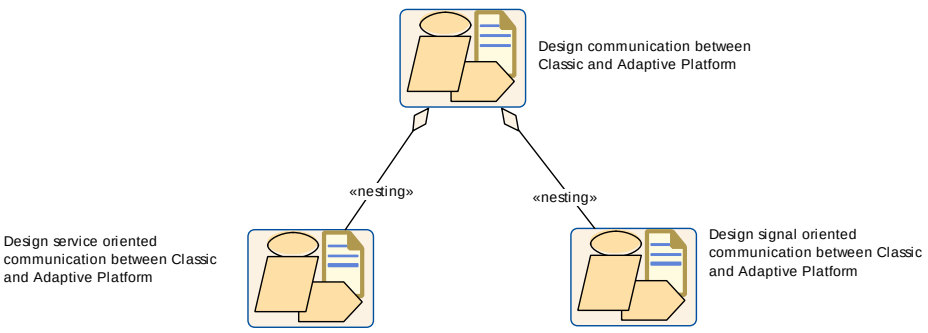

Figure 2.3: Design Communication between Classic Platform and Adaptive Platform

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

Table 2.1: Design communication between Classic and Adaptive Platform

2.1.2.2.3 System

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

[TR_AMETH_00204] Develop the System ⌈

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

- *Development of the System (TR_METH_01046)* and *(Develop) the overall system (TR_METH_01048)*, which talk about the refinement of the `VFB` by the definition of a topology of `ECU`s and networks and the deployment of software components onto `ECU`s, with the extensions necessary for the `Common Software Architecture` and the additions to specify `machine`s and the corresponding mapping of `machine`s to `ECU`s.
- *Two phase development approach (TR_METH_01047)* and *Interaction between organizations (TR_METH_01049)*, which structures the collaboration between different parties, like between OEMs and their suppliers.

⌋*(RS_METH_00041)*

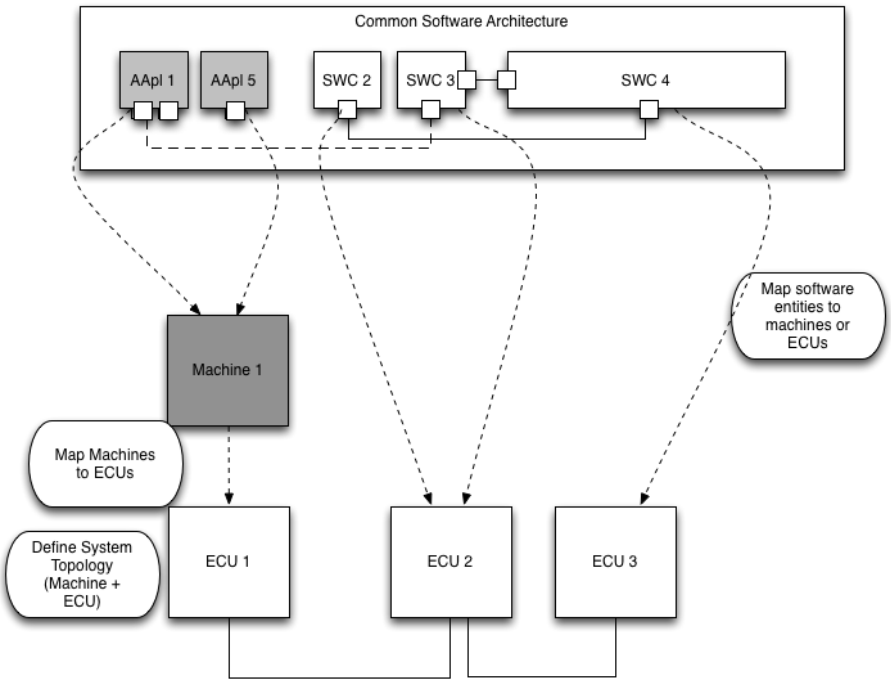

Figure 2.4: System development: `ECUs`, `machines`, communication networks, mapping of software entities to `ECUs` or `machines`

2.1.2.2.4 Software Development

[TR_AMETH_00002] Develop the software for `AdaptiveAutosarApplications`
⌈ Once the service interfaces have been defined, software for `AdaptiveAutosarApplications` of category application-level and platform-level can be developed. The development may include several sub-activities like analysis, design, implementation or test.

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

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

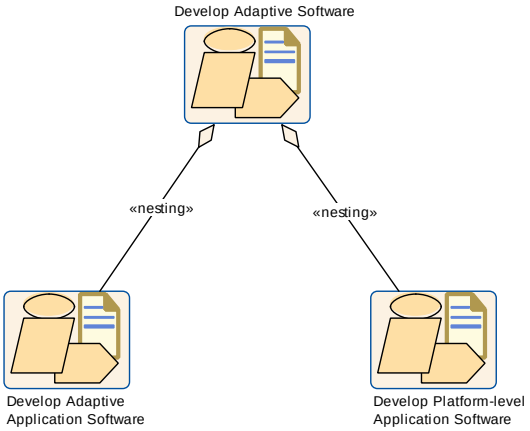

Figure 2.5: Develop Adaptive Software

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

Table 2.2: Develop Adaptive Software

2.1.2.2.5 Integration and Deployment

The term *Integration 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 Cluster`s , in order to satisfy all requirements. To start the integration, at least the operating system (kernel and some basic operating system functionality) needs to be ported for the dedicated machine.

[TR_AMETH_00205] Integrate Software to form `AdaptiveAutosarApplications` ⌈ 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 the actual executables of an `AdaptiveAutosarApplication` for a specific machine and notably its application binary interface (ABI).

Note, that an `AdaptiveAutosarApplication` may consist of several executables.

This activity does not include instantiation, i.e., the binding of actual executables of an AdaptiveAutosarApplication to the context of processes (at least one per executable).

Thus, the output of this activity are one or more executables of an AdaptiveAutosarApplication, which may be input for the instantiation.

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

[TR_AMETH_00003] Configuration of the machine ⌈ Independent of the definition of the service interfaces and the development of the software, the machine can be defined and configured. The machine's network connections will be configured and a specific designated IP multicast address and port number is given for service discovery message exchange. The available hardware resources for the machine will be described. In addition, the OS will be configured. All these configuration aspects are contained in the Machine Manifest. For details see Section 2.4.2. ⌋(*RS_METH_00204, RS_METH_00203*)

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

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

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

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

[TR_AMETH_00006] Deployment of the application software ⌈ Application software is deployed by means of SoftwareClusters.The deployed application software obviously need to be aligned, i.e., integrated, for the particular machines in the field.

For details see Section 2.4.6 and 2.4.7. ⌋(*RS_METH_00205*)

2.1.2.3 Workflow

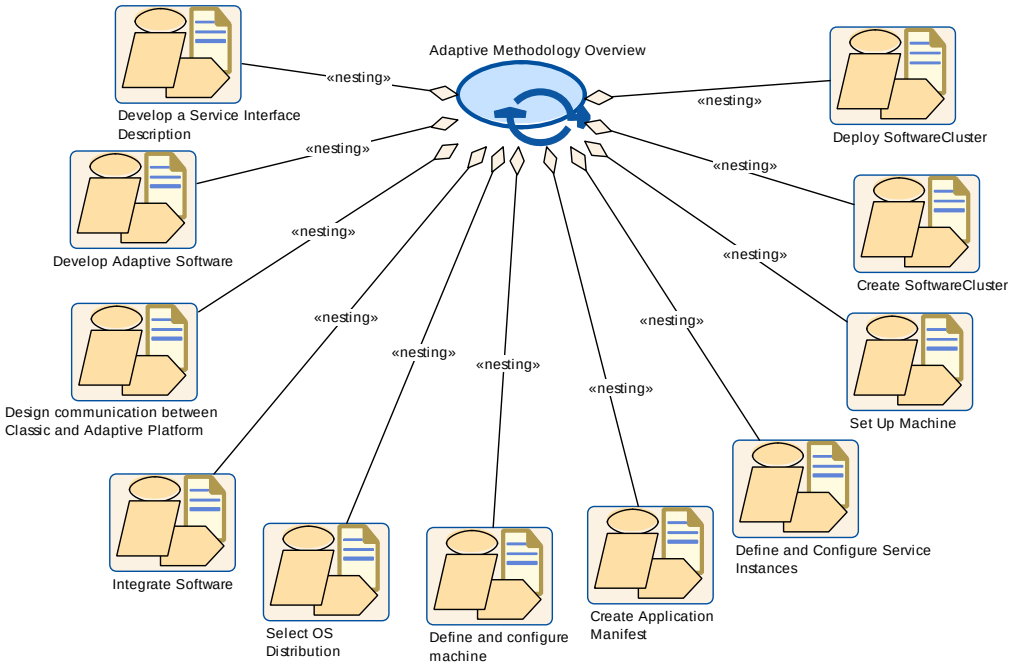

Figure 2.6: Adaptive Methodology Overview: Overall Structure

Process Pattern	**Adaptive Methodology Overview**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Adaptive Methodology Overview		
Brief Description	High-level view of the adaptive AUTOSAR methodology		
Description	This Process Pattern contains the typical activities to develop an Adaptive AUTOSAR system.		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Create Application Manifest	1	
Aggregates	Create Software Cluster	1	
Aggregates	Define and Configure Service Instances	1	
Aggregates	Define and configure machine	1	
Aggregates	Deploy Software Cluster	1	
Aggregates	Design communication between Classic and Adaptive Platform	1	
Aggregates	Develop Adaptive Software	1	
Aggregates	Develop Platform-level Application Software	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Develop a Service Interface Description	1	
Aggregates	Integrate Software	1	
Aggregates	Select OS Distribution	1	
Aggregates	Set Up Machine	1	

Table 2.3: Adaptive Methodology Overview

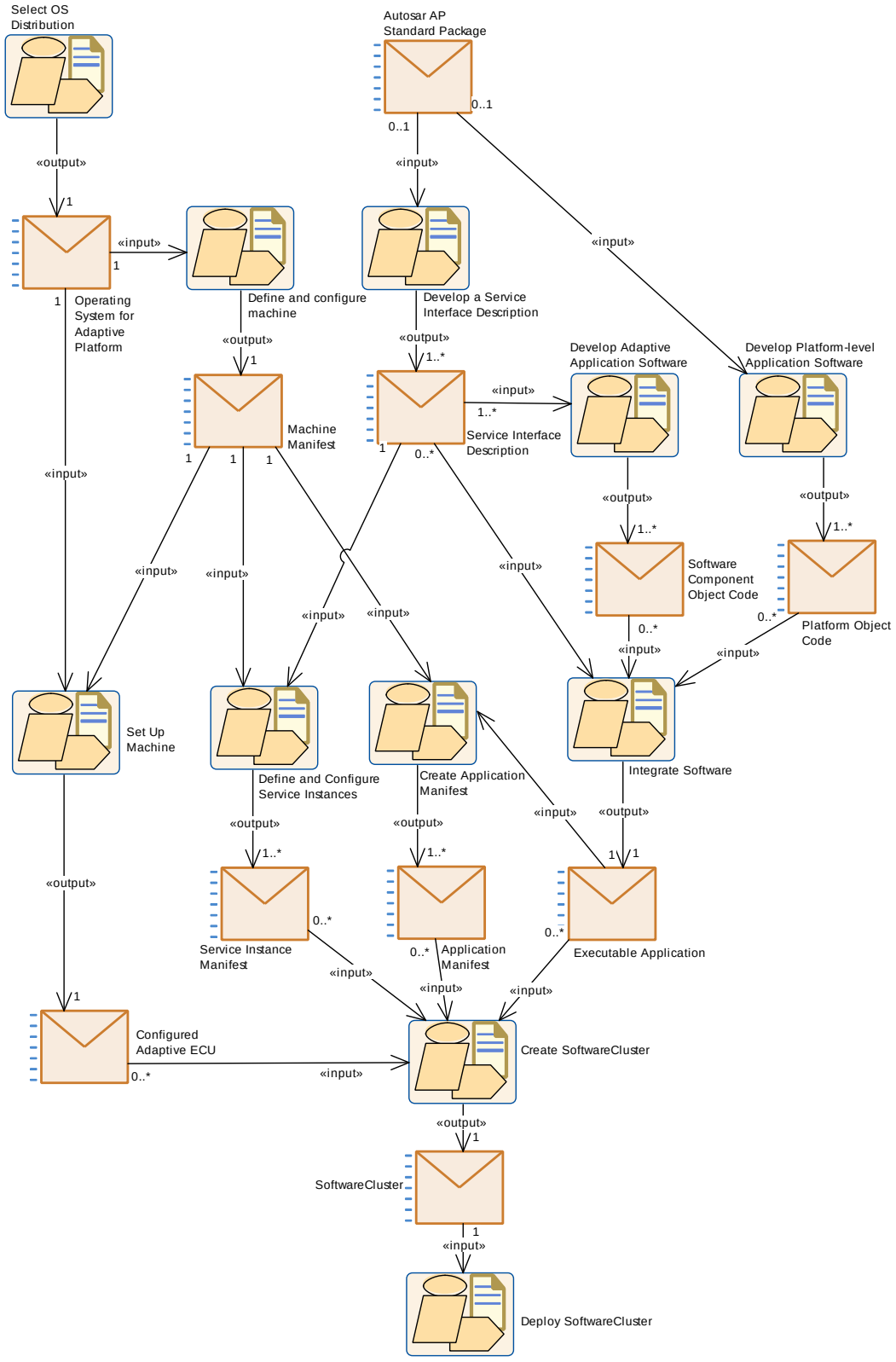

Figure 2.7: Adaptive Methodology Overview: Workflow

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

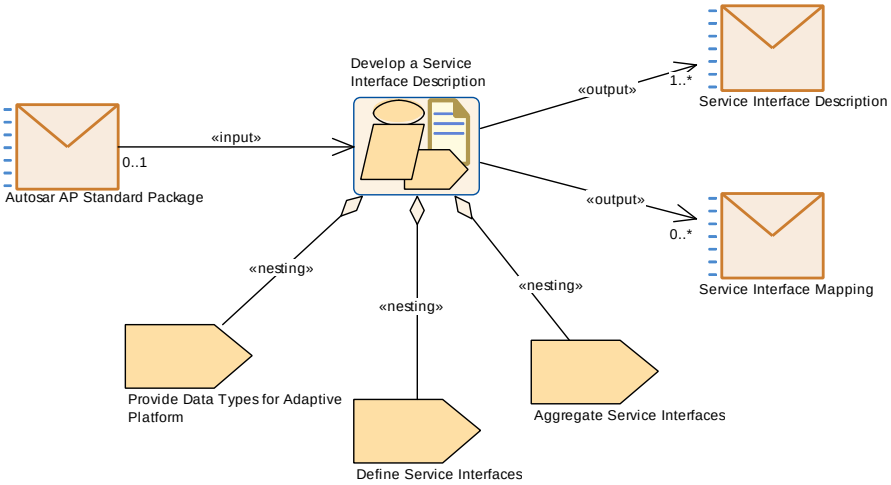

Figure 2.8: Develop a Service Interface Description

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

Table 2.4: Develop a Service Interface Description

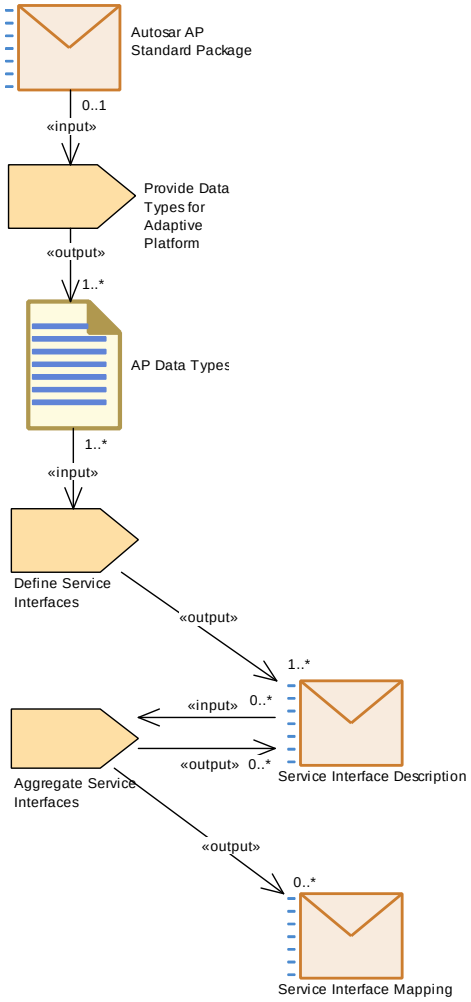

Figure 2.9: Workflow for defining Service Interfaces

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] Map a single ServiceInterface to PortInterface elements ⌈
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 ClientServer-Interface to a method located in a ServiceInterface.
- to map event(s), i.e., to map a VariableDataPrototype located in a Sender-ReceiverInterface 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 “Fireand Forget” method located in a ServiceInterface to a VariableDataPrototype in a SenderReceiverInterface or to a trigger of a TrigerInterface.

⌋(*RS_METH_00202*)

2.2.2.1.3 Workflow

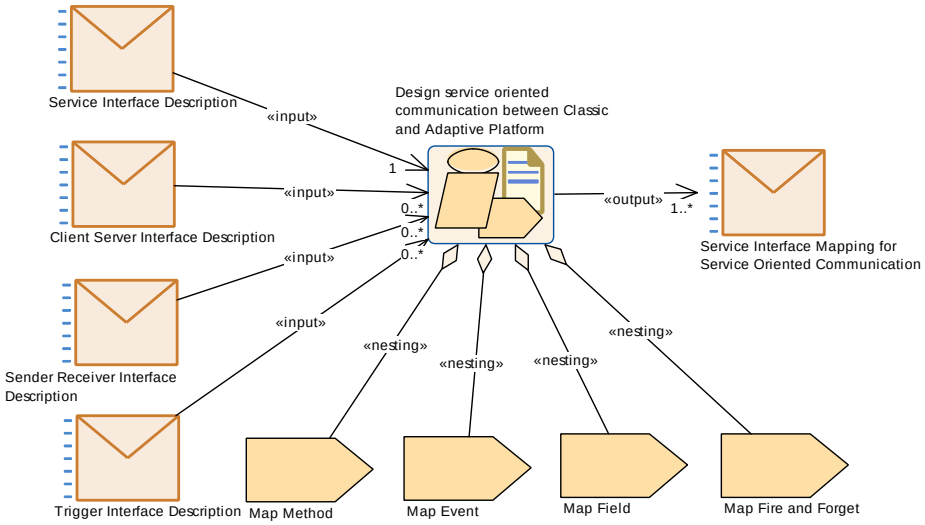

Figure 2.10: Design service oriented communication

Activity	**Design service oriented communication between Classic and Adaptive Platform**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Communication		
Brief Description	Design service oriented communication between CPand AP		
Description	The background of this activity is the request to enable service oriented communication between applications of a Classic Platform (CP) ECU and those of an Adaptive Platform (AP) machine via SOME/IP. Unfortunately, the AUTOSAR Classic Platform does not support ServiceInterfaces. Thus, a SOME/IP service may be composed of different types of Classic Platform PortInterfaces like SenderReceiverInterfaces, ClientServiceInterfaces or TriggerInterfaces. In order to describe the communication over SOME/IP between the CP ECU and a AP machine, this activity describes the mapping of the elements of the PortInterfaces of the Classical Platform to the elements of a single ServiceInterface of the Apdaptive Platform. The mapping description serves currently only for documentation.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Client Server Interface Description	1	The descriptions of Client Server Interfaces of CP are used to map a ClientServerOperation to a method in a ServiceInterface or to map a ClientServerOperation (representing getter or setter methods) to a field in a ServiceInterface
Consumes	Sender Receiver Interface Description	1	The descriptions of Sender Receiver Interfaces of CP are used to map a VariableDataPrototype to an Event in a ServiceInterface or to map a VariableDataPrototype to the notifier of a Field of a ServiceInterface or to map a Fire&Forget Method that is located in a ServiceInterface to a VariableDataPrototype in a SenderReceiverInterface
Consumes	Service Interface Description	1	Description of the Service Interface which communicates to CP in a service-oriented manner
Consumes	Trigger Interface Description	1	The descriptions of Trigger Interfaces are used to map a Fire&Forget Method that is located in ServiceInterface to a Trigger in a TriggerInterface
Produces	Service Interface Mapping for Service Oriented Communication	1..*	An InterfaceMapping results from the design of service-oriented communication between CP and AP
Aggregates	Map Event	1	
Aggregates	Map Field	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Map Fire and Forget	1	
Aggregates	Map Method	1	

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 SignalBasedServiceInterface 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 ManifestSpecification [6]. ⌋ *(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 SignalBasedMethod to ISignalTriggerings, according to TPS_MANI_03125 of the ManifestSpecification [6]
- to map SignalBasedEvent to ISignalTriggerings, according to TPS_MANI_03124 of the ManifestSpecification [6]
- to map SignalBasedField to ISignalTriggerings, according to TPS_MANI_03126 of the ManifestSpecification [6]
- to map a ServiceInstance to a PortPrototype, according to TPS_MANI_03000 of the ManifestSpecification [6]

⌋(*RS_METH_00202*)

2.2.2.2.3 Workflow

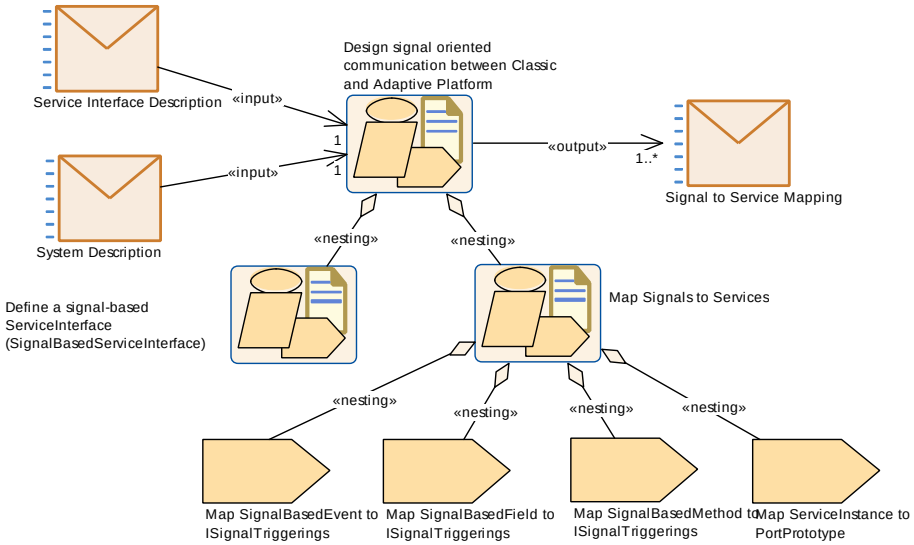

Figure 2.11: Design signal oriented communication

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

Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Define a signal-based Service Interface (Signal BasedService Interface)	1	
Aggregates	Map Signals to Services	1	

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

Activity	**Map Signals to Services**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Communication		
Brief Description	Map Signals to Services		
Description	Describe the mapping of ServiceInstance elements of a specific AdaptivePlatformServiceInstance defined in the context of a process to ISignalTriggerings. The prerequisite is the definition of the SignalBasedServiceInterface.		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Map ServiceInstance to Port Prototype	1	
Aggregates	Map SignalBased Event to ISignal Triggerings	1	
Aggregates	Map SignalBased Field to ISignal Triggerings	1	
Aggregates	Map SignalBased Method to ISignal Triggerings	1	

Table 2.7: Map Signals to Services

Activity	**Define a signal-based ServiceInterface (SignalBasedServiceInterface)**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Architecture and Design::Communication		
Brief Description	Define SignalBasedServiceInterface		
Description	Express that a ServiceInterface will be transmitted via a signal-based communication protocol like CAN or FlexRay.		
Relation Type	*Related Element*	*Mul.*	*Note*

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

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 2.12. The artifact-based workflow is depicted in Figure 2.13.

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

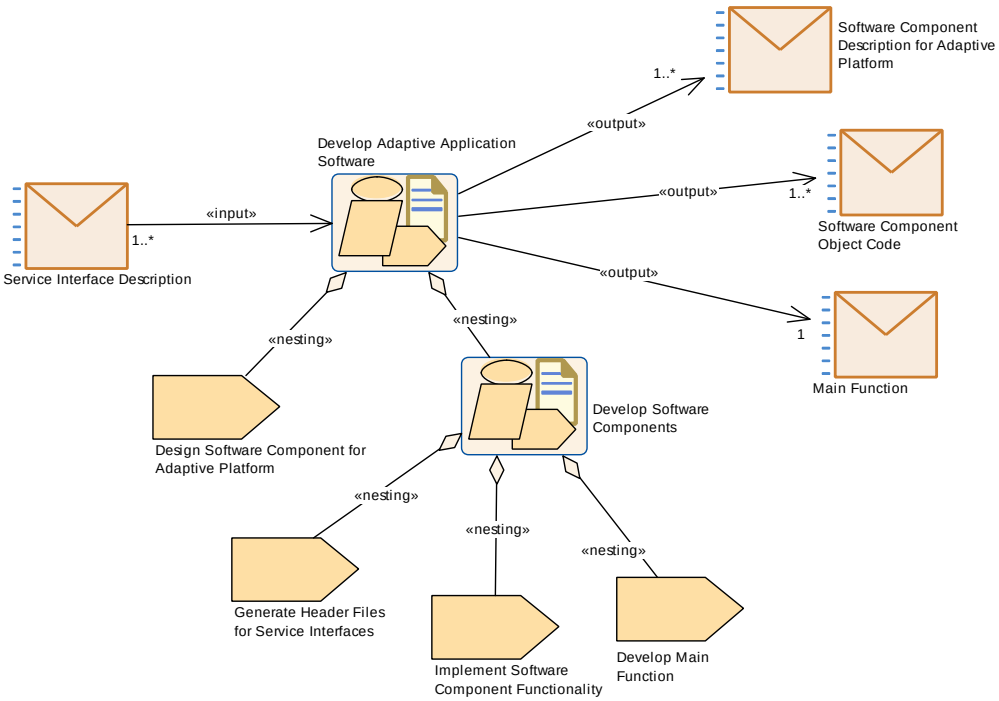

Figure 2.12: Develop Adaptive Application Software

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

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

Table 2.9: Develop Adaptive Application Software

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

Table 2.10: Develop Software Components

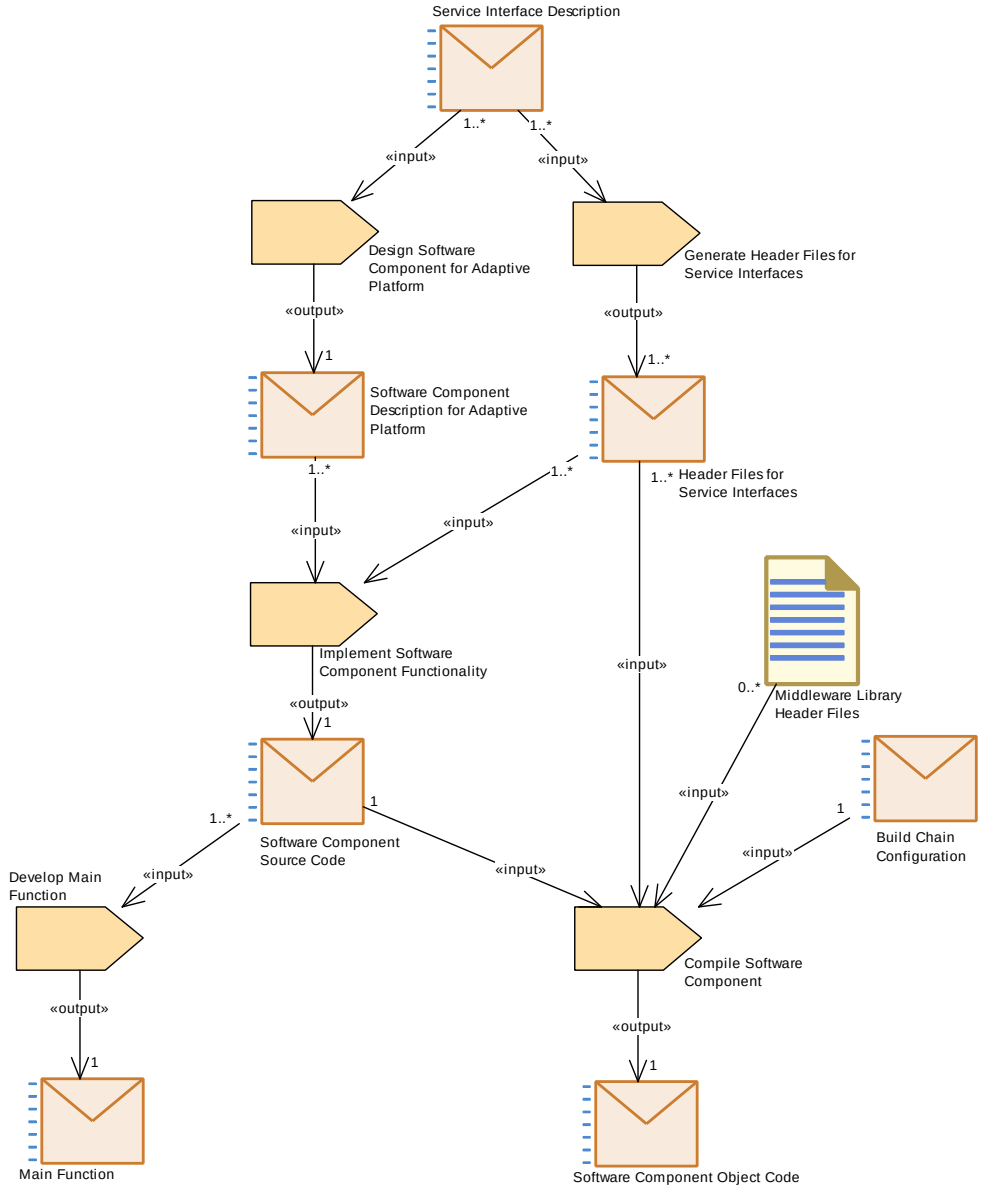

Figure 2.13: Workflow for developing application-level software for the Adaptive Platform

2.3.2 Develop Platform-level Application Software

2.3.2.1 Purpose

This section explains how to develop platform-level software for the Adaptive Platform. The artifact workflow is depicted in Figure 2.14.

2.3.2.2 Description

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

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

2.3.2.3 Workflow

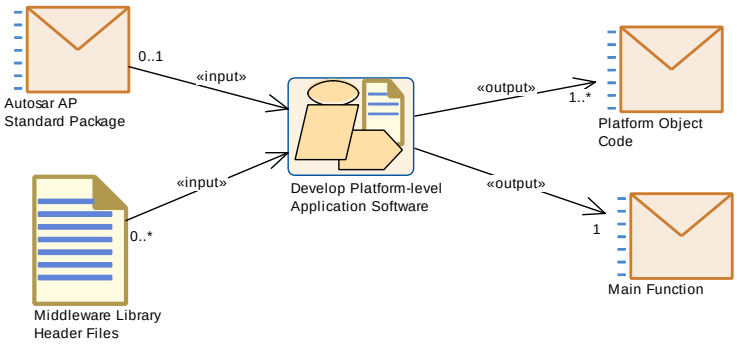

Figure 2.14: Develop Platform-level Application Software

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

Table 2.11: Develop Platform-level Application Software

2.4 Integration and Deployment

2.4.1 Integrate Software

2.4.1.1 Purpose

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

2.4.1.2 Description

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

[TR_AMETH_00017] Implementation of service proxies and skeletons ⌈ The service proxies and skeletons, which are contained in the `Header Files for Service Interfaces` and used within the software components, need to be implemented. For this implementation, the serialization of data needs to be known. ⌋*(RS_METH_00207)*

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

2.4.1.3 Workflow

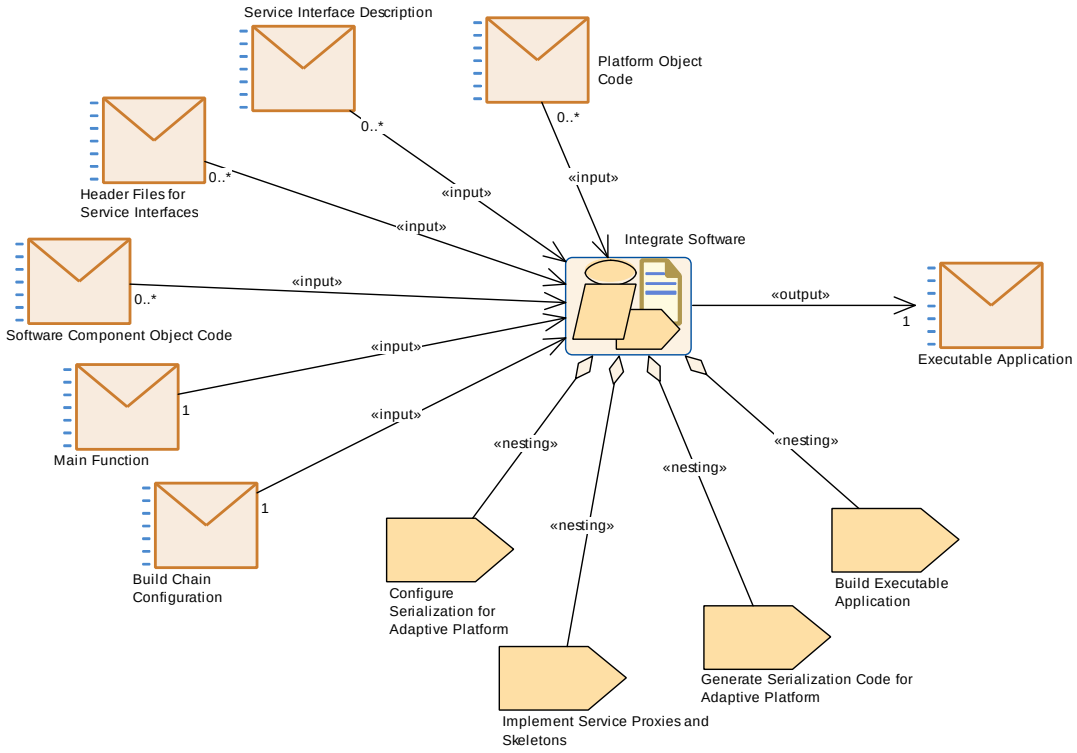

Figure 2.15: Integrate the software components

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

Relation Type	*Related Element*	*Mul.*	*Note*
Produces	Executable Application	1	Software is integrated into one executable application
Aggregates	Build Executable Application	1	
Aggregates	Configure Serialization for Adaptive Platform	1	
Aggregates	Generate Serialization Code for Adaptive Platform	1	
Aggregates	Implement Service Proxies and Skeletons	1	

Table 2.12: Integrate Software

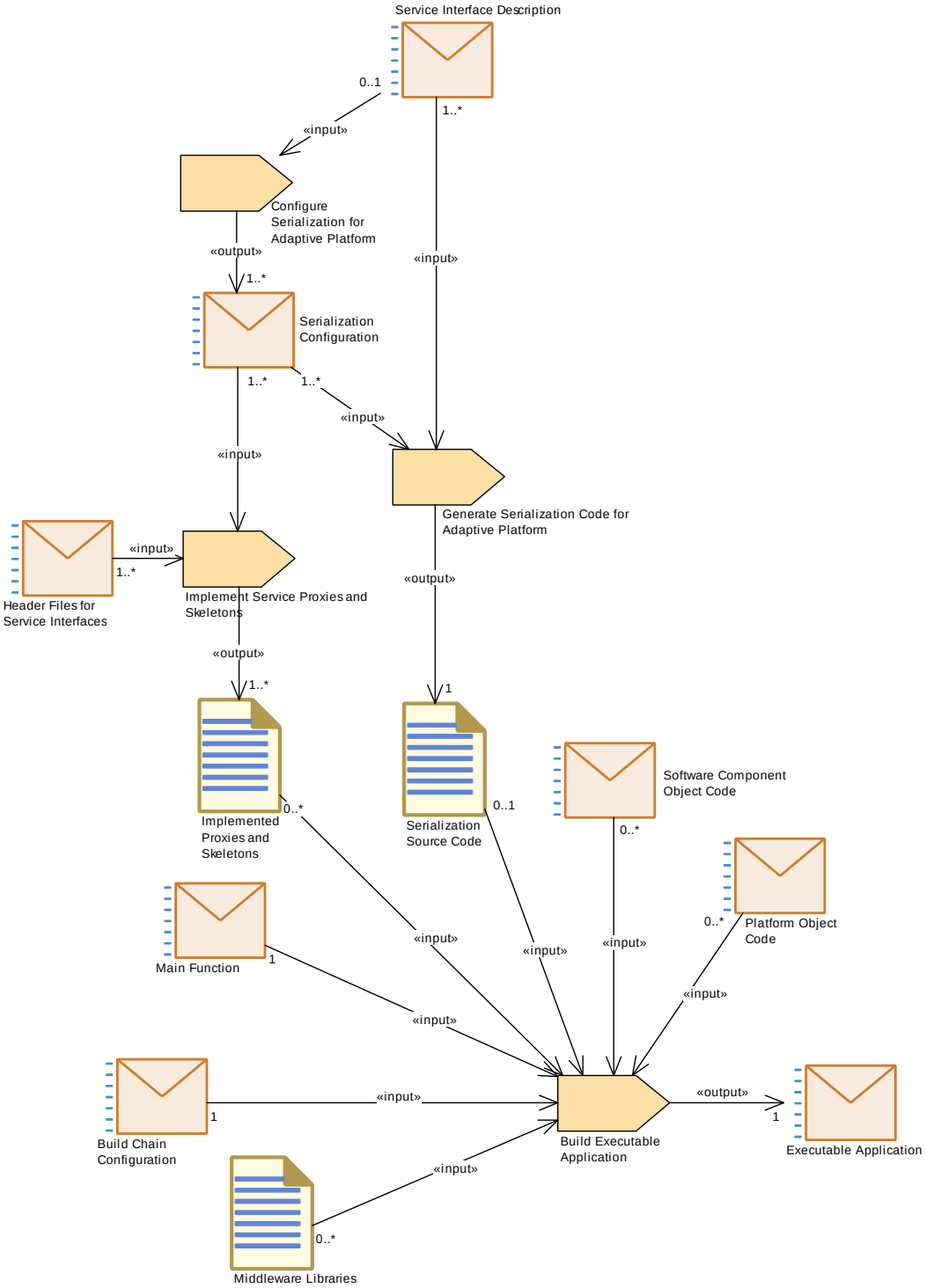

Figure 2.16: Workflow for integrating the software

2.4.2 Define and Configure Machine

The machine is an instance of the AUTOSAR Adaptive Platform. Before all necessary activities for configuring the machine can be described, the basis for this configuration, i.e. the Adaptive Platform itself needs to be set up.

2.4.2.1 Describe Platform

2.4.2.1.1 Purpose

This first step covers the tasks for describing the platform, independent of the instantiation in terms of a machine yet.

2.4.2.1.2 Description

[TR_AMETH_00019] Description of the Adaptive Platform ⌈ As a first step, the underlying hardware for the Adaptive Platform can be described. The description of all hardware elements like processing units, memory, sensors and actuators, pins is given in the `ECU Resources Description`. ⌋*(RS_METH_00207, RS_METH_00041)*

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

[TR_AMETH_00034] Selecting the `Operating System for Adaptive Platform` ⌈ For the platform, the operating system needs to be selected and assembled. The workflow for the platform modules as the OS is different to the workflow of platform-level applications (see section 2.3.2), which will be instantiated with an Application Manifest. ⌋*(RS_METH_00207, RS_METH_00041)*

2.4.2.1.3 Workflow

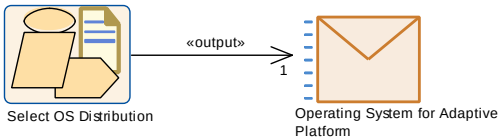

Figure 2.17: Select the OS Distribution

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

Table 2.13: Select OS Distribution

2.4.2.2 Configure Machine

2.4.2.2.1 Purpose

The machine describes the computing resource on which the Adaptive AUTOSAR Software Stack is executed. 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 2.18. The workflow is described in Figure 2.19.

2.4.2.2.2 Description

[TR_AMETH_00021] Configuration of network communication for machine ⌈ For the communication on the network, the machine's network connections need to be configured. In more detail, IPv4 or IPv6 addresses are defined. Additionally, in order to exchange service discovery messages with SOME/IP, a specifically designated IP multicast address and a UDP Port is specified. ⌋*(RS_METH_00204, RS_METH_00203)*

[TR_AMETH_00022] Definition of machine states and resources ⌈ 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 startup configuration of a process, which might depend on the machine states.
Optionally, based on the `ECU Resources Description` the available hardware resources for the machine can be described. ⌋*(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)*

2.4.2.2.3 Workflow

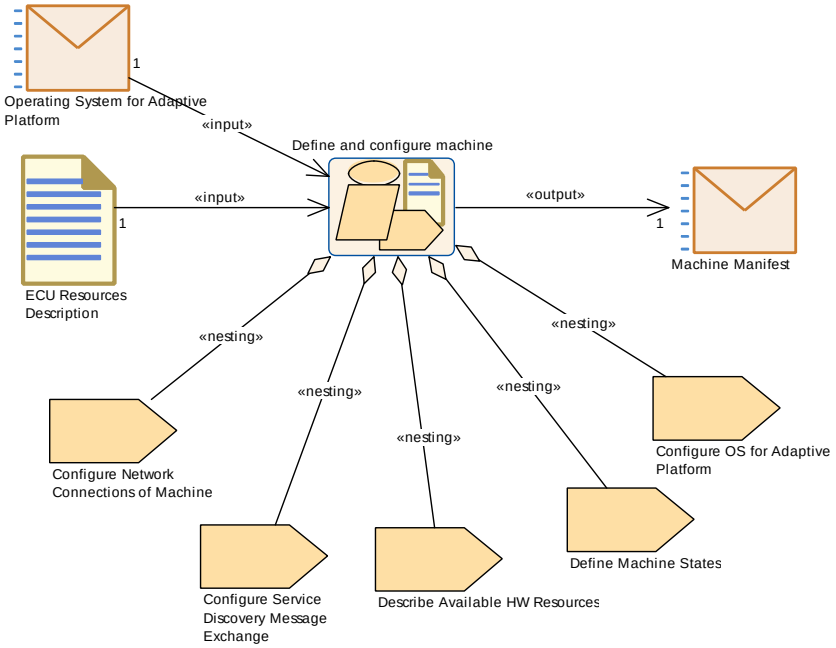

Figure 2.18: Define and Configure Machine

Activity	**Define and configure machine**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Define and Configure Machine::Machine Configuration		
Brief Description	Configuration of the machine independent of deployment information of applications or service instances		
Description	The activity describes tasks for the configuration of the machine, which do not depend on deployment information of applications or service instances. This includes the configuration for the communication on the network based on service discovery, the description of all machine states and the available resources as well as dedicated configuration of the OS.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	ECU Resources Description	1	All resources which are available for the ECU
Consumes	Operating System for Adaptive Platform	1	OS to be configured
Produces	Machine Manifest	1	The machine manifest describes all the configuration settings for one machine
Aggregates	Configure Network Connections of Machine	1	
Aggregates	Configure OS for Adaptive Platform	1	
Aggregates	Configure Service Discovery Message Exchange	1	

Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Define Machine States	1	
Aggregates	Describe Available HW Resources	1	

Table 2.14: Define and configure machine

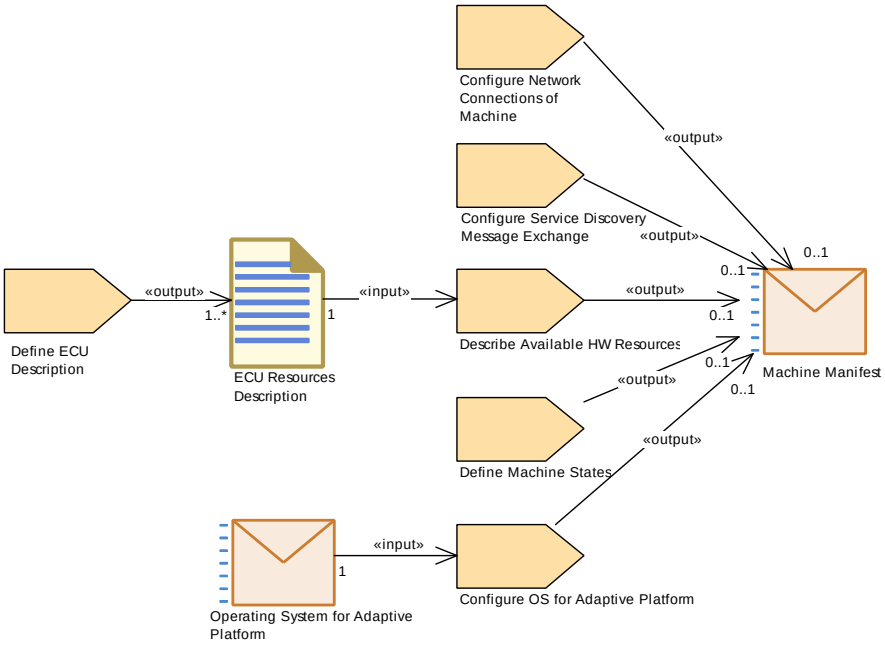

Figure 2.19: Workflow for defining and configuring an machine

2.4.3 Create Application Manifest

2.4.3.1 Purpose

This use case defines all tasks, which are necessary in order to instantiate the `Executable Application`. For on overview see Figure 2.20. The workflow is given in Figure 2.21.

2.4.3.2 Description

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

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

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

2.4.3.3 Workflow

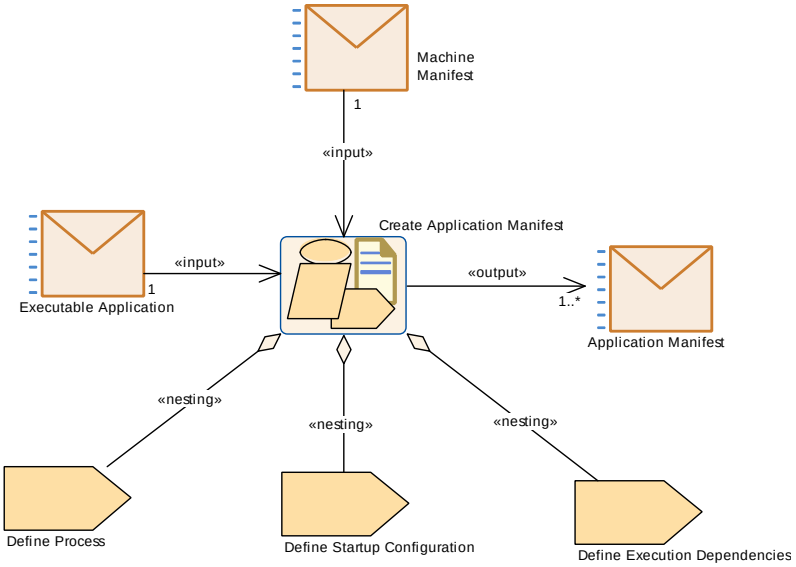

Figure 2.20: Create an Application Manifest

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

Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Define Process	1	
Aggregates	Define Startup Configuration	1	

Table 2.15: Create Application Manifest

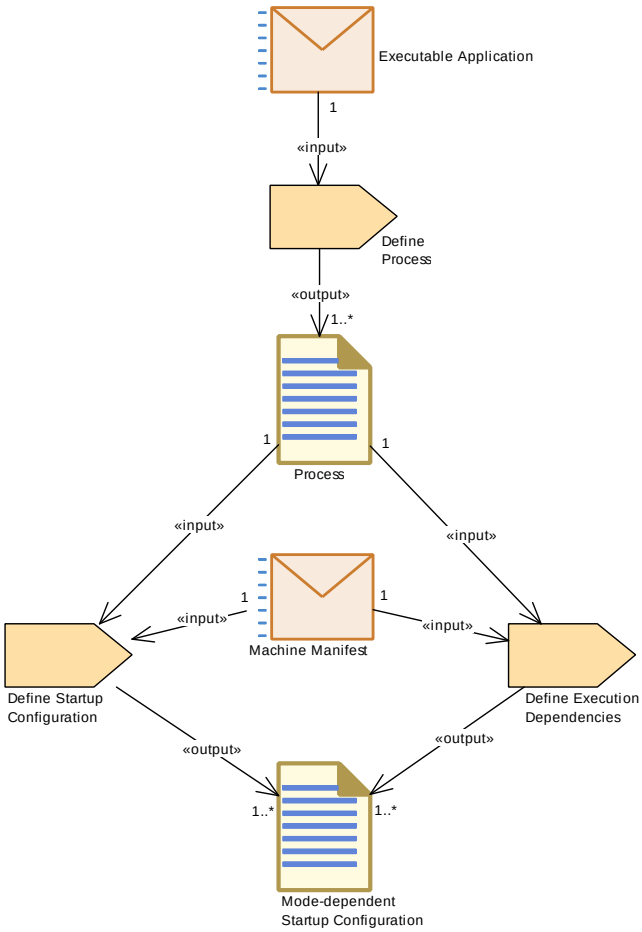

Figure 2.21: Workflow for defining a Process

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 2.22. For the workflow see Figure 2.23. The outcome of this activity is the `Service Instance Manifest`.

2.4.4.2 Description

[TR_AMETH_00027] Configuration of Service Interface Deployment ⌈ The system responsible needs to define how the service interfaces shall be deployed. In particular, for each used transport layer, the binding of the service interface to this transport layer needs to be given.
For SOME/IP deployment, an ID for each service interface is defined. This ID needs to be unique in the system. Additionally, methodId, eventId as well as event groups are defined. ⌋*(RS_METH_00206, RS_METH_00203, RS_METH_00084)*

[TR_AMETH_00028] Configuration of Service Instances ⌈ Afterwards, the system responsible defines instances of the deployed service interfaces and decides whether the service instance is provided or consumed. In order to set up the service-oriented communication, the search or offer criteria for all service instances are described. ⌋ *(RS_METH_00206, RS_METH_00203, RS_METH_00084)*

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

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

2.4.4.3 Workflow

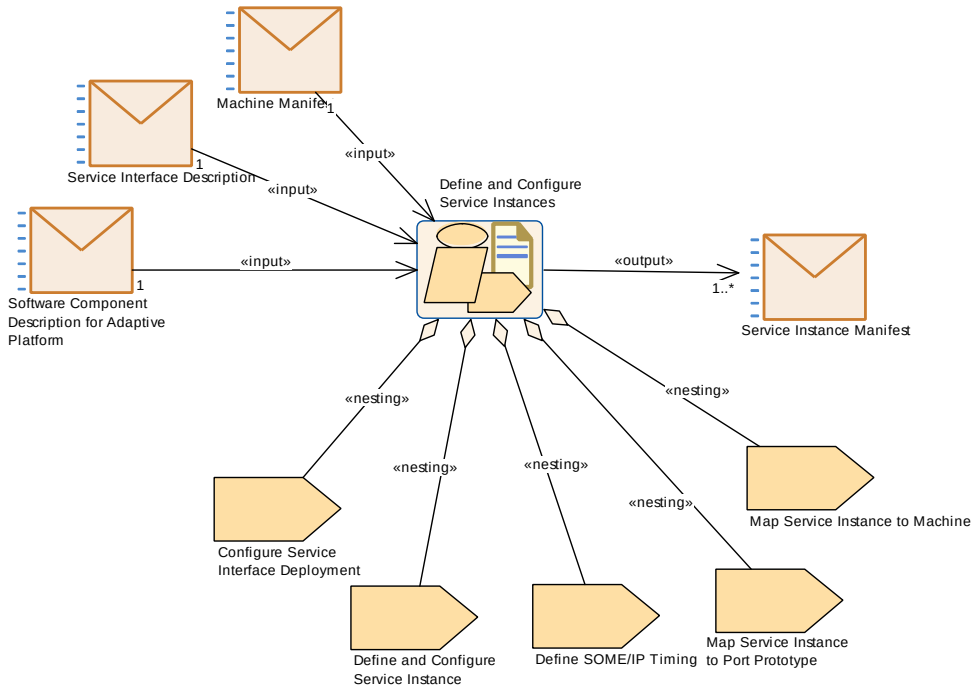

Figure 2.22: Define and Configure Service Instances

Activity	**Define and Configure Service Instances**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Service Instance Definition		
Brief Description	Configuration of service interface deployment and service instances		
Description	This activity describes two main steps. The first step describes the configuration of the service interfaces for the used network layer, independent of any instantiation. In the second step, the service instances are defined and configured.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Machine Manifest	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	Configure Service Interface Deployment	1	
Aggregates	Define SOME/IP Timing	1	

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

Table 2.16: Define and Configure Service Instances

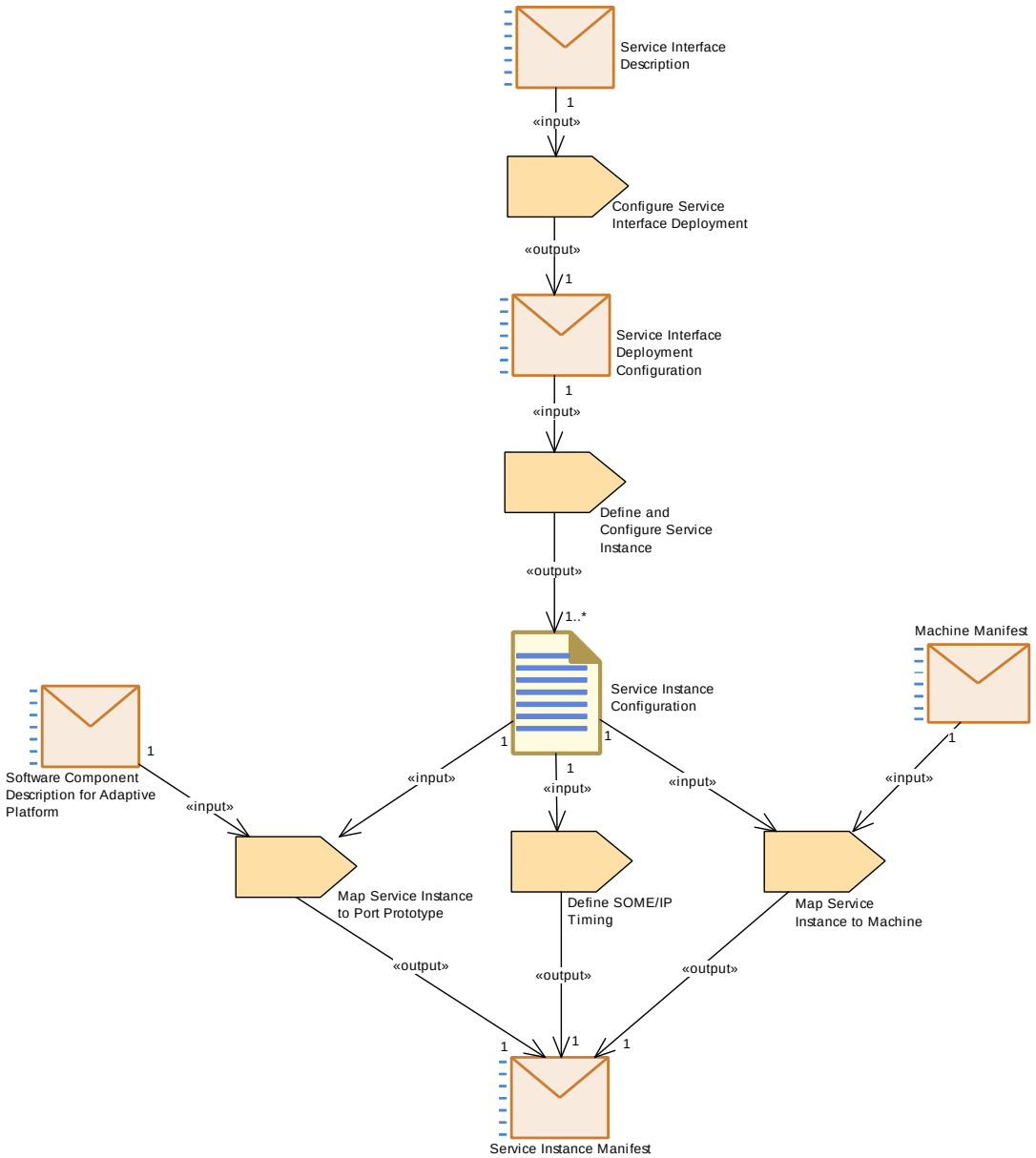

Figure 2.23: Workflow for defining and configuring service instances

2.4.5 Set up the Machine

2.4.5.1 Purpose

This activity describes how a machine is set up so that software can be deployed on it. The overview and workflow is depicted in Figure 2.24.

2.4.5.2 Description

[TR_AMETH_00031] Setting up the machine ⌈ The `Operating System for Adaptive Platform` has been selected for a specific Adaptive Platform type. The instantiation of an Adaptive Platform results in a machine. The necessary configuration settings for this instantiation is given in the `Machine Manifest`. In this step, the machine will be set up based on the configuration settings and the OS will be installed on the machine. This step is still independent of any application-level or platform-level software. These applications can be uploaded at a later point in time. ⌋*(RS_METH_00205, RS_METH_00204)*

2.4.5.3 Workflow

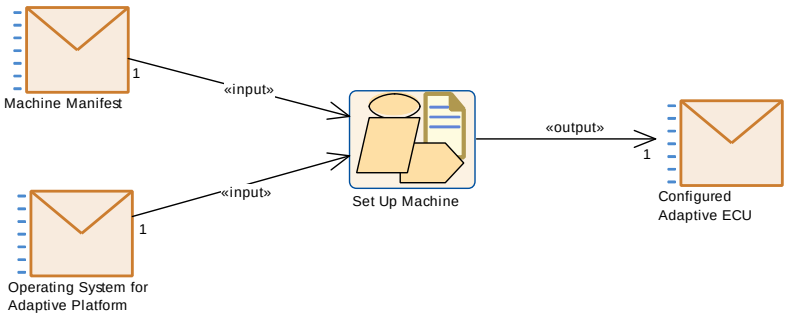

Figure 2.24: Set up machine

Activity	**Set Up Machine**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Define and Configure Machine::Setup Machine		
Brief Description	Set up the machine based on the machine manifest		
Description	Configure and install the OS on the machine. The configuration settings are given by the Machine Manifest. In addition, the network connections as well as machine states are set up.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Machine Manifest	1	Containing all configuration settings for the machine
Consumes	Operating System for Adaptive Platform	1	OS to be installed on machine

Relation Type	*Related Element*	*Mul.*	*Note*
Produces	Configured Adaptive ECU	1	Machine is configured and software can now be deployed

Table 2.17: Set Up Machine

2.4.6 Create `SoftwareCluster`

2.4.6.1 Purpose

This use case comprises activities to specify a `SoftwareCluster`.

The respective inputs and the output deliverable are depicted in Figure 2.25.

Note, that the definition of `SoftwareCluster` is work-in-progress and therefore subject to change.

2.4.6.2 Description

[TR_AMETH_00206] Create `SoftwareCluster` ⌈ A `SoftwareCluster` is the "atomic" entity (container) for deployment as well as for automotive diagnosis.

In this respect, `SoftwareCluster`s need to be identifiable by a unique number.

The `SoftwareCluster` may be empty in extreme cases. For its main use-cases, however, a `SoftwareCluster` may contain `AdaptiveAutosarApplication`(s), `Application Manifest`(s), `Service Instance Manifest`(s), the `Machine Manifest` and other artifacts. ⌋*(RS_METH_00205)*

2.4.6.3 Workflow

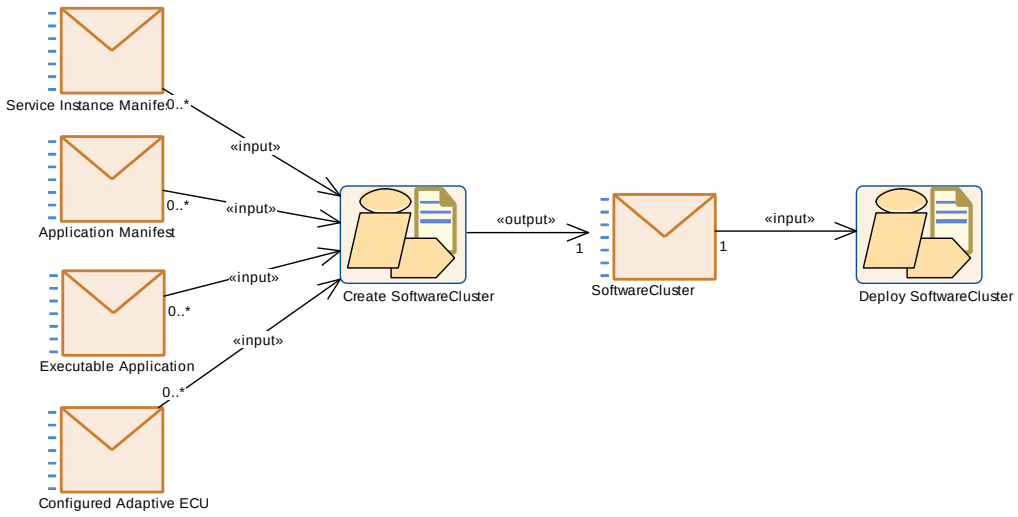

Figure 2.25: Create SoftwareCluster

Activity	**Create SoftwareCluster**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment		
Brief Description	Create a SoftwareCluster		
Description	This activity describes the creation of a SoftwareCluster, i.e., to bind artifacts that belong together in an unique and identifiable container. SoftwareCluster are used to deploy software to machines as well as "atomic" entities in case of automotive diagnosis.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	Application Manifest	0..*	Several processes can be deployed
Consumes	Configured Adaptive ECU	0..*	SW package will be deployed on one configured adaptive ECU
Consumes	Executable Application	0..*	Executables of deployed processes
Consumes	Service Instance Manifest	0..*	Several service instances can be deployed
Produces	SoftwareCluster	1	SoftwareCluster as atomic entity for deployment defined

Table 2.18: Create SoftwareCluster

2.4.7 Deploy Software

2.4.7.1 Purpose

Once the a particular `SoftwareCluster` has been created, it can be deployed to a specific machine.

This is shown in Figure 2.26.

Note, that this use case is work-in-progress and therefore subject to change.

2.4.7.2 Description

[TR_AMETH_00032] Deploying the SoftwareCluster ⌈ Dedicated `SoftwareClusters` are deployed to dedicated machines in the field (vehicle). ⌋*(RS_METH_00205)*

2.4.7.3 Workflow

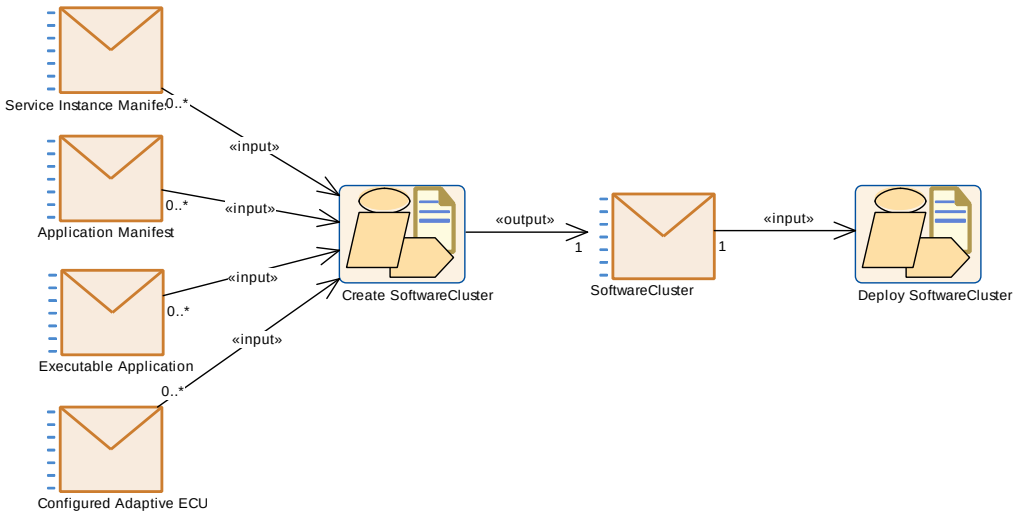

Figure 2.26: Deployment of a SoftwareCluster

Activity	**Deploy SoftwareCluster**		
Package	AUTOSAR Root::M2::Methodology::Methodology Use Cases::Adaptive Platform::Deployment		
Brief Description	Deployment of software to a machine		
Description	In this activity, application software by means of SoftwareClusters are deployed.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	SoftwareCluster	1	Deploy software on machine by means of SoftwareCluster

Table 2.19: Deploy SoftwareCluster

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 Service Interface

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

3.1.1 Tasks

3.1.1.1 Provide Data Types for Adaptive Platform

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

Table 3.1: Provide Data Types for Adaptive Platform

3.1.1.2 Define Service Interfaces

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

Table 3.2: Define Service Interfaces

3.1.1.3 Aggregate Service Interfaces

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

Table 3.3: Aggregate Service Interfaces

3.1.2 Work Products

3.1.2.1 AUTOSAR AP Standard Package

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

Table 3.4: Autosar AP Standard Package

3.1.2.2 AP Data Types

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

Table 3.5: AP Data Types

3.1.2.3 Service Interface Description

Deliverable	**Service Interface Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Interface Definition::Work Products		
Brief Description	Collection of service interfaces with events, methods and fields.		
Description	Collection of service interfaces. Service interfaces can consist of events, methods and fields and are the basis for the generation of header files for a software component. In addition, the namespace used for the header file generation can be defined.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define Service Interfaces	1..*	Collection of all service interfaces
Produced by	Develop a Service Interface Description	1..*	All service interfaces, which are used for communication
Produced by	Aggregate Service Interfaces	0..*	Coarse-grained service interfaces
Consumed by	Configure Service Interface Deployment	1	Deployment is configured for each service interface
Consumed by	Define and Configure Service Instances	1	Deployment of service interfaces needs to be configured
Consumed by	Design service oriented communication between Classic and Adaptive Platform	1	Description of the Service Interface which communicates to CP in a service-oriented manner
Consumed by	Design signal oriented communication between Classic and Adaptive Platform	1	Description of the Service Interface which communicates to CP in a signal-oriented manner
Consumed by	Design Software Component for Adaptive Platform	1..*	All service interfaces that shall be implemented by the software component
Consumed by	Develop Adaptive Application Software	1..*	Service Interfaces are the basis for the development of adaptive application software
Consumed by	Generate Header Files for Service Interfaces	1..*	For all service interfaces header files are generated.
Consumed by	Generate Serialization Code for Adaptive Platform	1..*	Service interfaces that are implemented by the software components are needed for generating the serialization code
Consumed by	Configure Serialization for Adaptive Platform	0..1	Optional if you only configure default values for the serialization
Consumed by	Aggregate Service Interfaces	0..*	Fine-grained service interfaces
Consumed by	Integrate Software	0..*	Needed for defining the serialization

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

Table 3.6: Service Interface Description

3.1.2.4 Service Interface Mapping

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

Table 3.7: Service Interface Mapping

3.2 Communication Mapping

3.2.1 Tasks

3.2.1.1 Map Method

Task Definition	**Map Method**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map Method		
Description	Map a ClientServerOperation located in a ClientServerInterface to a method located in a ServiceInterface. see TPS_MANI_03111 of AUTOSAR_TPS_ManifestSpecification		
Relation Type	*Related Element*	*Mul.*	*Note*

Table 3.8: Map Method

3.2.1.2 Map Event

Task Definition	**Map Event**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map Event		
Description	Map a VariableDataPrototype located in a SenderReceiverInterface to an event located in a ServiceInterface. see TPS_MANI_03112 of of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***

Table 3.9: Map Event

3.2.1.3 Map Field

Task Definition	**Map Field**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map Field		
Description	Map operations located in ClientServerOperations to getter and setter methods of a ServiceInterface. Map a VariableDataPrototype of a SenderReceiverInterface to the field notifier of the ServiceInterface. see TPS_MANI_03113 of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***

Table 3.10: Map Field

3.2.1.4 Map Fire and Forget

Task Definition	**Map Fire and Forget**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map Fire and Forget		
Description	Map a Fire&Forget method located in a ServiceInterface to a VariableDataPrototype in a SenderReceiverInterface or to a trigger of a TrigerInterface. see TPS_MANI_03115 of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***

Table 3.11: Map Fire and Forget

3.2.1.5 Map SignalBasedMethod to ISignalTriggerings

Task Definition	**Map SignalBasedMethod to ISignalTriggerings**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map SignalBasedMethod to ISignalTriggerings		
Description	see TPS_MANI_03125 of of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***

Table 3.12: Map SignalBasedMethod to ISignalTriggerings

3.2.1.6 Map SignalBasedEvent to ISignalTriggerings

Task Definition	**Map SignalBasedEvent to ISignalTriggerings**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map SignalBasedEvent to ISignalTriggerings		
Description	see TPS_MANI_03124 of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***

Table 3.13: Map SignalBasedEvent to ISignalTriggerings

3.2.1.7 Map SignalBasedField to ISignalTriggerings

Task Definition	**Map SignalBasedField to ISignalTriggerings**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map SignalBasedField to ISignalTriggerings		
Description	see TPS_MANI_03126 of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***

Table 3.14: Map SignalBasedField to ISignalTriggerings

3.2.1.8 Map ServiceInstance to PortPrototype

Task Definition	**Map ServiceInstance to PortPrototype**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Tasks		
Brief Description	Map ServiceInstance to PortPrototype		
Description	see TPS_MANI_03000 of AUTOSAR_TPS_ManifestSpecification		
Relation Type	***Related Element***	***Mul.***	***Note***

Table 3.15: Map ServiceInstance to PortPrototype

3.2.2 Work Products

3.2.2.1 Client Server Interface Description

Deliverable	**Client Server Interface Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Work Products		
Brief Description	Client Server Interface Description		
Description	This represents the particular description of a ClientServerInterface of the Classic Platform.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Design service oriented communication between Classic and Adaptive Platform	1	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 3.16: Client Server Interface Description

3.2.2.2 Sender Receiver Interface Description

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

Table 3.17: Sender Receiver Interface Description

3.2.2.3 Trigger Interface Description

Deliverable	**Trigger Interface Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Work Products		
Brief Description	Trigger Interface Description		
Description	This represents the particular description of the Trigger Interface of the Classic Platform.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumed by	Design service oriented communication between Classic and Adaptive Platform	1	The descriptions of Trigger Interfaces are used to map a Fire&Forget Method that is located in ServiceInterface to a Trigger in a TriggerInterface

Table 3.18: Trigger Interface Description

3.2.2.4 Service Interface Mapping for Service Oriented Communication

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

Table 3.19: Service Interface Mapping for Service Oriented Communication

3.2.2.5 System Description

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

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

Table 3.20: System Description

3.2.2.6 Signal to Service Mapping

Deliverable	**Signal to Service Mapping**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Architecture and Design::Communication Mapping::Work Products		
Brief Description	Collection of mappings for signal oriented communication		
Description	Collection of mappings of ServiceInstances to ISignalTriggerings.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Design signal oriented communication between Classic and Adaptive Platform	1..*	A signal-to-service mapping results from the design of signal-oriented communication between CP and AP

Table 3.21: Signal to Service Mapping

3.3 Adaptive Application

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

3.3.1 Tasks

3.3.1.1 Generate Header Files for Service Interfaces

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

Table 3.22: Generate Header Files for Service Interfaces

3.3.1.2 Design Software Component for Adaptive Platform

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

Table 3.23: Design Software Component for Adaptive Platform

3.3.1.3 Implement Software Component Functionality

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

Table 3.24: Implement Software Component Functionality

3.3.1.4 Compile Software Component

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

Table 3.25: Compile Software Component

3.3.1.5 Develop Main Function

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

Table 3.26: Develop Main Function

3.3.1.6 Configure Serialization for Adaptive Platform

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

Table 3.27: Configure Serialization for Adaptive Platform

3.3.1.7 Generate Serialization Code for Adaptive Platform

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

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

Table 3.28: Generate Serialization Code for Adaptive Platform

3.3.1.8 Implement Service Proxies and Skeletons

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

Table 3.29: Implement Service Proxies and Skeletons

3.3.1.9 Build Executable Application

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

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

Table 3.30: Build Executable Application

3.3.2 Work Products

3.3.2.1 Header Files for Service Interfaces

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

Table 3.31: Header Files for Service Interfaces

3.3.2.2 Software Component Description for Adaptive Platform

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

Table 3.32: Software Component Description for Adaptive Platform

3.3.2.3 Build Chain Configuration

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

Table 3.33: Build Chain Configuration

3.3.2.4 Software Component Source Code

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

Table 3.34: Software Component Source Code

3.3.2.5 Software Component Object Code

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

Table 3.35: Software Component Object Code

3.3.2.6 Serialization Configuration for Adaptive Platform

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

Table 3.36: Serialization Configuration

3.3.2.7 Serialization Source Code

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

Table 3.37: Serialization Source Code

3.3.2.8 Implemented Service Proxies and Skeletons

Artifact	**Implemented Proxies and Skeletons**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
Brief Description	Implemented service proxies and skeletons		
Description	Implemented source code for the service proxies and skeletons.		
Kind	Source Code		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Implement Service Proxies and Skeletons	1..*	Implementation of service proxies and skeletons given as source code
Consumed by	Build Executable Application	0..*	Source code of service proxies and skeletons

Table 3.38: Implemented Proxies and Skeletons

3.3.2.9 Main Function

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

Table 3.39: Main Function

3.3.2.10 Executable Application

Deliverable	**Executable Application**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Adaptive Application::Work Products		
Brief Description	Executable application containing several software components		
Description	The executable application, or just executable, can contain an arbitrary hierarchy of software components. The software components contain the functionality of the executable. Several executables can be packaged into an Adaptive AUTOSAR Application. They can be of category application-level or platform-level.		
Kind	Executable		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Build Executable Application	1	One executable is built
Produced by	Integrate Software	1	Software is integrated into one executable application
Consumed by	Create Application Manifest	1	One executable can be instantiated several times
Consumed by	Define Process	1	Executable to be instantiated
Consumed by	Create Software Cluster	0..*	Executables of deployed processes

Table 3.40: Executable Application

3.4 Platform and Machine

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

3.4.1 Tasks

3.4.1.1 Configure Network Connections of Machine

Task Definition	**Configure Network Connections of Machine**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
Brief Description	Definition of all network endpoints with corresponding IP address.		
Description	Define all network connections of a machine and their configuration out of contracting. All network endpoints with corresponding IP address are specified.		
Relation Type	***Related Element***	***Mul.***	***Note***
Produces	Machine Manifest	0..1	Configuration settings of network connections of machine

Table 3.41: Configure Network Connections of Machine

3.4.1.2 Configure Service Discovery Message Exchange

Task Definition	**Configure Service Discovery Message Exchange**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
Brief Description	Definition of ports and multicast IP addresses for service discovery message exchange		
Description	Define ports and multicast IP address over which the service discovery messages are exchanged.		
Relation Type	***Related Element***	***Mul.***	***Note***
Produces	Machine Manifest	0..1	Configuration settings of machine for service discovery message exchange

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

Table 3.42: Configure Service Discovery Message Exchange

3.4.1.3 Define ECU Description

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

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

Table 3.43: Define ECU Description

3.4.1.4 Describe Available HW Resources

Task Definition	**Describe Available HW Resources**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Tasks		
Brief Description	Description of available hardware resources for the machine		
Description	Optional step for describing available hardware resources for the machine.		
Relation Type	***Related Element***	***Mul.***	***Note***
Consumes	ECU Resources Description	1	Definition of available HW resources for the machine based on the description of the ECU
Produces	Machine Manifest	0..1	Available hardware resources of machine

Table 3.44: Describe Available HW Resources

3.4.1.5 Define Machine States

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

Table 3.45: Define Machine States

3.4.1.6 Configure OS for Adaptive Platform

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

Table 3.46: Configure OS for Adaptive Platform

3.4.2 Work Products

3.4.2.1 Middleware Library Header Files

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

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

Table 3.47: Middleware Library Header Files

3.4.2.2 Middleware Libraries

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

Table 3.48: Middleware Libraries

3.4.2.3 ECU Resources Description

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

Artifact	**ECU Resources Description**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::System:: Work products		
Brief Description	Definition of the resources available on an ECU.		
Description	Definition of the resources available on an ECU. It mainly contains a description of hardware elements (like physical memory sections or peripherals, pins, hardware connections) which need to be referred by a software component or a basic software description. The focus is to describe an already engineered piece of hardware, its content and structure. It is not in the focus of the ECU Resource Description to support the design of electronics hardware itself. In the XML it is represented as a set of HwDescriptionEntity -s		
Kind	AUTOSAR XML		
Relation Type	*Related Element*	*Mul.*	*Note*
Aggregated by	Complete ECU Description	1	
Produced by	Define ECU Description	1..*	
Consumed by	Define and configure machine	1	All resources which are available for the ECU
Consumed by	Describe Available HW Resources	1	Definition of available HW resources for the machine based on the description of the ECU
Consumed by	Define System Topology	1..*	
Consumed by	Define BSW Interfaces	0..1	

Relation Type	*Related Element*	*Mul.*	*Note*
Consumed by	Define ECU Abstraction Component	0..1	
Consumed by	Extend Topology	0..1	
Consumed by	Generate ECU Executable	0..1	may be used to set up build environment Meth.bindingTime = CompileTime
Consumed by	Implement a BSW Module	0..1	Meth.bindingTime = SystemDesignTime
Consumed by	Measure Component Resources	0..1	
Consumed by	Measure Resources	0..1	
Consumed by	Define Complex Driver Component	0..*	
Consumed by	Define VFB Sensor or Actuator Component	0..*	
Use meta model element	HwElement	1	

Table 3.49: ECU Resources Description

3.4.2.4 Configured Adaptive ECU

Deliverable	**Configured Adaptive ECU**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Work Products		
Brief Description	Configured Adaptive Platform instance		
Description	This work product is a configured Adaptive Platform instance, i.e. a configured machine, where software can be deployed on. The configuration settings are based on the Machine Manifest.		
Kind	Custom		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Set Up Machine	1	Machine is configured and software can now be deployed
Consumed by	Create Software Cluster	0..*	SW package will be deployed on one configured adaptive ECU

Table 3.50: Configured Adaptive ECU

3.4.2.5 Machine Manifest

Deliverable	**Machine Manifest**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Machine Configuration::Work Products		
Brief Description	Configuration of the machine		
Description	Description of deployment content for the configuration of the machine, independent of any service instances or applications.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define and configure machine	1	The machine manifest describes all the configuration settings for one machine
Produced by	Configure Network Connections of Machine	0..1	Configuration settings of network connections of machine
Produced by	Configure OS for Adaptive Platform	0..1	Configuration settings of OS
Produced by	Configure Service Discovery Message Exchange	0..1	Configuration settings of machine for service discovery message exchange
Produced by	Define Machine States	0..1	States defined for the machine
Produced by	Describe Available HW Resources	0..1	Available hardware resources of machine
Consumed by	Create Application Manifest	1	Instantiation is defined on one specific machine
Consumed by	Define Execution Dependencies	1	Execution dependencies are defined per machine mode.
Consumed by	Define Startup Configuration	1	Startup configuration is defined per machine mode given in the Machine Manifest
Consumed by	Define and Configure Service Instances	1	Service instances will be mapped to machine
Consumed by	Map Service Instance to Machine	1	Description of machine that the service instances shall be mapped to
Consumed by	Set Up Machine	1	Containing all configuration settings for the machine

Table 3.51: Machine Manifest

3.4.2.6 Platform Object Code

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

Table 3.52: Platform Object Code

3.4.2.7 Operating System for Adaptive Platform

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

Table 3.53: Operating System for Adaptive Platform

3.5 Application Manifest

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

3.5.1 Tasks

3.5.1.1 Define Process

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

Table 3.54: Define Process

3.5.1.2 Define Startup Configuration

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

Table 3.55: Define Startup Configuration

3.5.1.3 Define Execution Dependencies

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

Table 3.56: Define Execution Dependencies

3.5.2 Work Products

3.5.2.1 Application Manifest

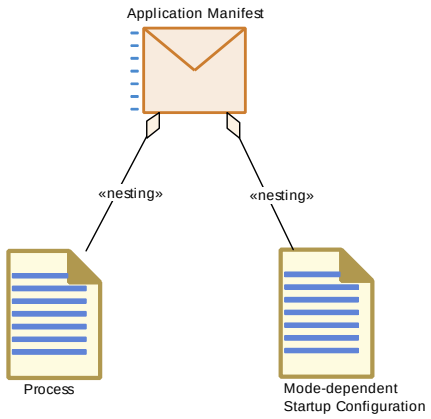

Figure 3.1: Structure of Deliverable `Application Manifest`

Deliverable	**Application Manifest**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Application Manifest::Work Products		
Brief Description	Definition of a process and all its properties		
Description	The application manifest defines the process with all its properties. It is defined for a specific machine by referencing its modes in the startup configuration. One application manifest is defined per process.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Aggregates	Mode-dependent Startup Configuration	1	For each process the startup configuration can be defined in the Application Manifest

Relation Type	*Related Element*	*Mul.*	*Note*
Aggregates	Process	1	The process is defined via the Application Manifest
Produced by	Create Application Manifest	1..*	One application manifest per instantiated executable
Consumed by	Create Software Cluster	0..*	Several processes can be deployed

Table 3.57: Application Manifest

3.5.2.2 Process

Artifact	**Process**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Application Manifest::Work Products		
Brief Description	Instantiation of an executable		
Description	The process is the top-level element of the Application Manifest and references an executable. It is the unit of deployment on the AUTOSAR adaptive platform and refers to a POSIX process.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define Process	1..*	Different instantiation of executables can result in different processes.
Consumed by	Define Execution Dependencies	1	Execution dependencies defined for one process
Consumed by	Define Startup Configuration	1	Startup configuration to be defined for process

Table 3.58: Process

3.5.2.3 Mode-dependent Startup Configuration

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

Table 3.59: Mode-dependent Startup Configuration

3.6 Service Instance

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

3.6.1 Tasks

3.6.1.1 Configure Service Interface Deployment

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

Table 3.60: Configure Service Interface Deployment

3.6.1.2 Define and Configure Service Instance

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

Table 3.61: Define and Configure Service Instance

3.6.1.3 Define SOME/IP timing

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

Table 3.62: Define SOME/IP Timing

3.6.1.4 Map Service Instance to Port Prototype

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

Table 3.63: Map Service Instance to Port Prototype

3.6.1.5 Map Service Instance to Machine

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

Table 3.64: Map Service Instance to Machine

3.6.2 Work Products

3.6.2.1 Service Interface Deployment Configuration

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

Table 3.65: Service Interface Deployment Configuration

3.6.2.2 Service Instance Configuration

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

Table 3.66: Service Instance Configuration

3.6.2.3 Service Instance Manifest

Deliverable	**Service Instance Manifest**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Service Instance Manifest::Work Products		
Brief Description	Definition and configuration of a service instance		
Description	Definition of a service instance with its configuration for the service discovery. The mapping of the service instances to the machine is defined. Optionally, the mapping of service instances to the software component ports is specified.		
Kind	AUTOSAR XML		
Relation Type	***Related Element***	***Mul.***	***Note***
Produced by	Define SOME/IP Timing	1	Timing for service instances contributes to Service Instance Manifest
Produced by	Map Service Instance to Machine	1	Mapping contributes to Service Instance Manifest
Produced by	Map Service Instance to Port Prototype	1	Mapping contributes to Service Instance Manifest
Produced by	Define and Configure Service Instances	1..*	Contains all configuration settings for the service instance on a specific machine
Consumed by	Create Software Cluster	0..*	Several service instances can be deployed

Table 3.67: Service Instance Manifest

3.7 Deployment

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

3.7.1 Work Products

3.7.1.1 SoftwareCluster

Deliverable	**SoftwareCluster**		
Package	AUTOSAR Root::M2::Methodology::Methodology Library::Adaptive Platform::Deployment::Work Products		
Brief Description	Atomic container to deploy software artifacts to a machine		
Description	A SoftwareCluster is a container and the "atomic" entity for deployment as well as for automotive diagnosis. In this respect, SoftwareClusters are identifiable by a unique number. The SoftwareCluster may be empty in extreme cases. For its main use-cases, however, a SoftwareCluster may contain AdaptiveAutosarApplication(s), Application Manifest(s), Service Instance Manifest(s), theMachine Manifest and other artifacts.		
Kind	Custom		
Relation Type	***Related Element***	***Mul.***	***Note***

Relation Type	*Related Element*	*Mul.*	*Note*
Produced by	Create Software Cluster	1	SoftwareCluster as atomic entity for deployment defined
Consumed by	Deploy Software Cluster	1	Deploy software on machine by means of SoftwareCluster

Table 3.68: SoftwareCluster

A Change History

A.1 Change History for AP 17-10

A.1.1 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.1: Added specification items in AP 17-10

A.1.2 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.2: Changed specification items in AP 17-10

A.1.3 Deleted Specification Items in AP 17-10

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

Table A.3: Deleted specification items in AP 17-10

A.2 Change History for AP 17-03

A.2.1 Added Specification Items in AP 17-03

Number	Heading
[TR_AMETH_00100]	Scope of the Methodology for the Adaptive Platform
[TR_AMETH_00101]	Definition of tasks, work products and use cases
[TR_AMETH_00102]	Types of work products
[TR_AMETH_00001]	Description of the services in a system
[TR_AMETH_00002]	Development of the software
[TR_AMETH_00003]	Configuration of the machine
[TR_AMETH_00004]	Creation of the `Application Manifest`
[TR_AMETH_00005]	Configuration of the service instances
[TR_AMETH_00006]	Deployment of the application software
[TR_AMETH_00007]	Definition of data types for the Adaptive Platform
[TR_AMETH_00008]	Definition of service interfaces for the Adaptive Platform
[TR_AMETH_00009]	Aggregating service interfaces for reducing the bus load
[TR_AMETH_00010]	Application-level Software
[TR_AMETH_00011]	Design of the software components
[TR_AMETH_00012]	Generation of the header files for service interface
[TR_AMETH_00013]	Implementation and compilation of software components
[TR_AMETH_00014]	Development with knowledge of the `Build Chain Configuration`
[TR_AMETH_00015]	Development without knowledge of the `Build Chain Configuration`
[TR_AMETH_00016]	Development of serialization properties
[TR_AMETH_00017]	Implementation of service proxies and skeletons
[TR_AMETH_00018]	Building the `Executable Application`
[TR_AMETH_00019]	Description of the Adaptive Platform
[TR_AMETH_00020]	Development of `Platform Software`
[TR_AMETH_00021]	Configuration of network communication for machine
[TR_AMETH_00022]	Definition of machine states and resources
[TR_AMETH_00023]	Configuration of the operating system
[TR_AMETH_00024]	Instantiation of `Executable Application`
[TR_AMETH_00025]	Defintion of startup behavior of a process
[TR_AMETH_00026]	Defintion of `Application Manifest`
[TR_AMETH_00027]	Configuration of Service Interface Deployment
[TR_AMETH_00028]	Configuration of Service Instances
[TR_AMETH_00029]	Deployment of Service Instances
[TR_AMETH_00030]	Machine-driven and model-driven approach
[TR_AMETH_00031]	Setting up the machine
[TR_AMETH_00032]	Deploying the Software Package
[TR_AMETH_00033]	Mapping of Service Instances to Application Endpoints
[TR_AMETH_00034]	Selecting the `Operating System for Adaptive Platform`
[TR_AMETH_00035]	Platform-level Software

Table A.4: Added specification items in AP 17-03

A.2.2 Changed Specification Items in AP 17-03

N/A

A.2.3 Deleted Specification Items in AP 17-03

N/A