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

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R24-11

Document Change History

Date	Release	Changed by	Description
2024-11-27	R24-11	AUTOSAR Release Management	• Editorial changes
2023-11-23	R23-11	AUTOSAR Release Management	• Editorial changes - clean up in chapters 3.9, 3.10, 3.11
2022-11-24	R22-11	AUTOSAR Release Management	• Consolidate the usage of Glossary
2021-11-25	R21-11	AUTOSAR Release Management	• New description of the Use Cases for the Adaptive Platform, starting with the big picture of the work flow to give an overall view of the methodology flow. • Introduced top-down and bottom-up usage scenarios • spec.items kept their ID and semantics but have been heavily reworked as per updated methodology description, task and artifact names etc.
2020-11-30	R20-11	AUTOSAR Release Management	• Add signal-based Service Interface

△

2019-11-28	R19-11	AUTOSAR Release Management	• disentangle service interface handling • remove machine state • Changed Document Status from Final to published • editorial changes
2019-03-29	R19-03	AUTOSAR Release Management	• No content changes
2018-10-31	R18-10	AUTOSAR Release Management	• renamed `Application Manifest` to `Execution Manifest` • moved references from spec.item body to foot notes • editorial changes
2018-03-29	R18-03	AUTOSAR Release Management	• Split of machine design and machine configuration • Added diagnostic mapping • Added roles • Reviewed sections about deployment of Software Packages
2017-10-27	R17-10	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	R17-03	AUTOSAR Release Management	• Initial release

Disclaimer

Contents

References

[1] Methodology for Classic Platform
AUTOSAR_CP_TR_Methodology

[2] Requirements on Methodology for Classic and Adaptive Platform
AUTOSAR_FO_RS_Methodology

[3] Standardization Template
AUTOSAR_FO_TPS_StandardizationTemplate

[4] Glossary
AUTOSAR_FO_TR_Glossary

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

[6] Specification of Manifest
AUTOSAR_AP_TPS_ManifestSpecification

[7] Explanation of Adaptive Platform Software Architecture
AUTOSAR_AP_EXP_SWArchitecture

[8] Layered Software Architecture
AUTOSAR_CP_EXP_LayeredSoftwareArchitecture

[9] Specification of Abstract Platform
AUTOSAR_FO_TPS_AbstractPlatformSpecification

[10] System Template
AUTOSAR_CP_TPS_SystemTemplate

[11] Software Component Template
AUTOSAR_CP_TPS_SoftwareComponentTemplate

[12] Diagnostic Extract Template
AUTOSAR_CP_TPS_DiagnosticExtractTemplate

[13] Specification of Execution Management
AUTOSAR_AP_SWS_ExecutionManagement

[14] Specification of Update and Configuration Management
AUTOSAR_AP_SWS_UpdateAndConfigurationManagement

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

Moreover, the term Machine has been newly introduced with the AUTOSAR Adaptive Platform. A Machine is quasi a virtualized ECU-HW, an entity where software can be deployed to. In this spirit, one real ECU-HW could run several Machines, even though the methodology will not detail this. In the simplest case there is a 1:1 mapping between a Machine and a ECU-HW.

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

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

[TR_AMETH_00100] Scope of the Methodology for the AUTOSAR Adaptive Platform

Upstream requirements: RS_METH_00006, RS_METH_00020, RS_METH_00056

⌈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.⌋

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:

- Section 2 (Use Cases for the Adaptive Platform) 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 (Adaptive Methodology Library) lists and describes all `task`s and `work product`s, which are used in the descriptions of the use cases in section 2.
- The rest of this Introduction section documents the policies utilized and the requirements traceability map.

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. `Machine`. As a general rule, plural forms of technical terms are created by adding "s" to the singular form, e.g. `Machine`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 Abbreviations and Technical Terms

The main list of terms and abbreviations are defined in [4]. The following table contains the list of terms and abbreviations used in the scope of this document which are not already defined in [4] along with the spelled-out meaning of each of the abbreviations.

Abbreviation	*Meaning*
-	-

Table 1.1: Abbreviations used in the scope of this Document

Technical Term	*Meaning*
Adaptive Software	An application-level or platform-level software entity tailored to run on the AUTOSAR Adaptive Platform. These software entities may consist of multiple `Executable`s running in multiple OS `Process`es.
Bottom-up approach	A methodology pattern in which the work/design flows from a concrete/fine grained methodology to an abstract/coarse grained methodology, e.g. SWC → binary → software cluster → System
Diagnostic Manager	The term Diagnostic Management is a placeholder for the complete functionality of diagnostic communication and event handling.
Function Group	A Function Group defines a state-machine for a set of cohesive instances (i.e. `Process`es) likely having a run-time interdependency. The `Process` specifies it's participation in a Function Group.
Instance Specifier	An Instance Specifier is a shortened, "stringified" representation of an instance reference. In terms of `ARXML` an instance reference is stereotyped with ≪`InstanceRef`≫ and if it is not necessary on target also with ≪`atpUriDef`≫: in this scenario only the Instance Specifier is visible in the configuration data on target.
Top-down approach	A methodology pattern in which the work/design flows from a abstract/coarse grained methodology to a concrete/fine grained methodology, e.g. System → software cluster → binary → SWC

Table 1.2: Technical terms used in the scope of this Document

1.5 Methodology Concepts

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

[TR_AMETH_00101] Definition of tasks, work products and use cases

Upstream requirements: RS_METH_00018

⌈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.⌋

[1]

[TR_AMETH_00102] Types and kinds of work products

Upstream requirements: RS_METH_00018

⌈`Work product`s are either `artifact`s or `deliverable`s and can be of the kind `AUTOSAR XML`, `source code`, `object code`, `executable`, `text` or `custom`.⌋

[1] This document describes use cases in Section 2, `task`s and `work product`s in Section 3.

[TR_AMETH_00226] Documentation of work products

Upstream requirements: RS_METH_00069

⌈In order to document design decisions or restrictions during the development process, each `work product` may aggregate a corresponding documentation.⌋

The definitions and figures follow mostly the Software Process Engineering Meta-Model Specification [5, SPEM] using symbols as per SPEM support of Enterprise Architect modeling tool.

1.6 Requirements Traceability

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

Requirement	Description	Satisfied by
[RS_METH_00006]	The methodology shall explain how to build an AUTOSAR system	[TR_AMETH_00016] [TR_AMETH_00100]
[RS_METH_00015]	The methodology shall be independent of programming languages	[TR_AMETH_00013]
[RS_METH_00016]	The methodology shall support building a system of both AUTOSAR and Non-AUTOSAR ECUs	[TR_AMETH_00212] [TR_AMETH_00213]
[RS_METH_00018]	The methodology shall be modular	[TR_AMETH_00101] [TR_AMETH_00102] [TR_AMETH_00200]
[RS_METH_00020]	The methodology shall support round-trip engineering	[TR_AMETH_00100]
[RS_METH_00032]	The methodology shall support different abstraction levels	[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_00062]	The methodology shall support configuration of parameters with different binding time.	[TR_AMETH_00251]
[RS_METH_00066]	The methodology shall allow activities that reference tools	[TR_AMETH_00012] [TR_AMETH_00013] [TR_AMETH_00016] [TR_AMETH_00018]
[RS_METH_00069]	It shall be possible to add precise and human readable documentation to each work product	[TR_AMETH_00226]
[RS_METH_00074]	The methodology shall specify binding times	[TR_AMETH_00251]
[RS_METH_00075]	The methodology shall specify the tasks of resolving variant	[TR_AMETH_00251]

▽

△

Requirement	Description	Satisfied by
[RS_METH_00076]	The methodology shall specify a work product for values of variant selectors	[TR_AMETH_00251]
[RS_METH_00077]	The methodology shall support different views on the SW-C structure by OEMs and suppliers	[TR_AMETH_00014] [TR_AMETH_00015] [TR_AMETH_00016] [TR_AMETH_00024]
[RS_METH_00078]	The methodology shall explain the typical usage of different views on the system of the OEM	[TR_AMETH_00029] [TR_AMETH_00033] [TR_AMETH_00203]
[RS_METH_00079]	The methodology shall explain the typical usage of different views on the system of the supplier	[TR_AMETH_00203]
[RS_METH_00200]	The methodology shall support building a system consisting of several AUTOSAR platforms	[TR_AMETH_00208] [TR_AMETH_00210]
[RS_METH_00201]	The methodology shall explain how to design the services of a system	[TR_AMETH_00001] [TR_AMETH_00007] [TR_AMETH_00008] [TR_AMETH_00009] [TR_AMETH_00212] [TR_AMETH_00213]
[RS_METH_00202]	The methodology shall explain how to develop an Adaptive Application	[TR_AMETH_00002] [TR_AMETH_00010] [TR_AMETH_00011] [TR_AMETH_00012] [TR_AMETH_00013] [TR_AMETH_00014] [TR_AMETH_00015] [TR_AMETH_00018] [TR_AMETH_00205] [TR_AMETH_00207] [TR_AMETH_00208] [TR_AMETH_00210]
[RS_METH_00203]	The methodology shall explain the high-level usage of the Manifest Specification	[TR_AMETH_00003] [TR_AMETH_00004] [TR_AMETH_00005] [TR_AMETH_00021] [TR_AMETH_00022] [TR_AMETH_00023] [TR_AMETH_00024] [TR_AMETH_00025] [TR_AMETH_00026] [TR_AMETH_00027] [TR_AMETH_00028] [TR_AMETH_00029] [TR_AMETH_00033] [TR_AMETH_00214] [TR_AMETH_00215] [TR_AMETH_00216] [TR_AMETH_00217]
[RS_METH_00204]	The methodology shall describe how to configure a machine for the Adaptive Platform	[TR_AMETH_00003] [TR_AMETH_00021] [TR_AMETH_00022] [TR_AMETH_00023] [TR_AMETH_00031] [TR_AMETH_00214] [TR_AMETH_00215] [TR_AMETH_00216] [TR_AMETH_00217]
[RS_METH_00205]	The methodology shall describe how to deploy software on the Adaptive Platform	[TR_AMETH_00006] [TR_AMETH_00031] [TR_AMETH_00206] [TR_AMETH_00218] [TR_AMETH_00219] [TR_AMETH_00220] [TR_AMETH_00221] [TR_AMETH_00222] [TR_AMETH_00223] [TR_AMETH_00224] [TR_AMETH_00225]
[RS_METH_00206]	The methodology shall explain how to configure the instances of services of a system	[TR_AMETH_00005] [TR_AMETH_00027] [TR_AMETH_00028] [TR_AMETH_00029] [TR_AMETH_00033]
[RS_METH_00207]	The methodology shall explain how to develop Platform Software for the Adaptive Platform	[TR_AMETH_00017] [TR_AMETH_00019] [TR_AMETH_00020] [TR_AMETH_00034] [TR_AMETH_00035] [TR_AMETH_00212] [TR_AMETH_00213]
[RS_METH_00208]	The methodology shall support the data exchange between different stakeholders	[TR_AMETH_00204]

Table 1.3: Requirements Tracing

1.7 Known Limitations

Elaboration of chapter 2.10 / 3.10 Software Cluster Integration is pending

Elaboration of chapter 2.11 / 3.11 Software Packaging is pending

1.8 Design vs. Deployment

Please note that the workflow of an individual artifact (or deliverable)

1. may be associated exclusively to design steps
2. may be associated to both design and deployment steps [2]
3. may be associated exclusively to deployment steps
4. may be associated to deployment steps depending on information from design level [3]
   (a) this may include design data items replicated into deployment artifacts [4]
   (b) this may need only structural information from design level [5]
5. may be associated to deployment steps having direct counterparts on design level [6]

which indicates that the border between design and deployment on the AUTOSAR adaptive platform is not as easily defined as on the AUTOSAR classic platform.

[2] e.g. for service interfaces
[3] e.g. `COM` configuration may depend on com-specs of software component ports
[4] e.g. for `PHM` configuration and checkpoint ID
[5] e.g. an instance specifier reflects the content of a reference to a port of a software component instance. See also [6], the content of references marked with ≪`atpUriDef`≫ is in scope of design only.
[6] e.g. process vs. process design

2 Use Cases for the Adaptive Platform

This chapter describes the main use cases for building a system based on the AUTOSAR Adaptive Platform in terms of tasks and work products according to [5, Software Process Engineering Meta-Model Specification].

[TR_AMETH_00200] Domains of development utilized for the methodology of the AUTOSAR Adaptive Platform

Upstream requirements: RS_METH_00018, RS_METH_00032

⌈The methodology of the Adaptive Platform shall be structured by the following domains of development:

- Analysis
- Architecture and Design
- `System` Development
- Software Development
- Integration and Deployment

⌋

[TR_AMETH_00204] Develop the System

Status: DRAFT
Upstream requirements: RS_METH_00041, RS_METH_00208

⌈The subsequent specifications of the Classic Platform methodology [1] 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-Instance`s and networks and the deployment of software components onto `ECU-Instance`s, with the extensions necessary for the `Vehicle Software Architecture` and the additions to specify `Machine`s and the corresponding mapping of `Machine`s to `ECU-HW`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.

⌋

[TR_AMETH_00251] Variant handling

Status: DRAFT
Upstream requirements: RS_METH_00062, RS_METH_00074, RS_METH_00075, RS_METH_-00076

⌈The variant mechanisms as known from CP apply to AP only for configuration items shared with CP.

Any design time related AP configuration item has implicitly a latest binding time of code-generation or pre-compile time.

Any AP configuration item defined as manifest content is subject to post-build and run-time variance: the behavior of an Adaptive Software can be changed by replacing manifest files only.⌋

Figure 2.1 shows the main areas of the Adaptive Platform methodology reaching from High level architecture via several design and integration steps to application implementation and SW packaging.

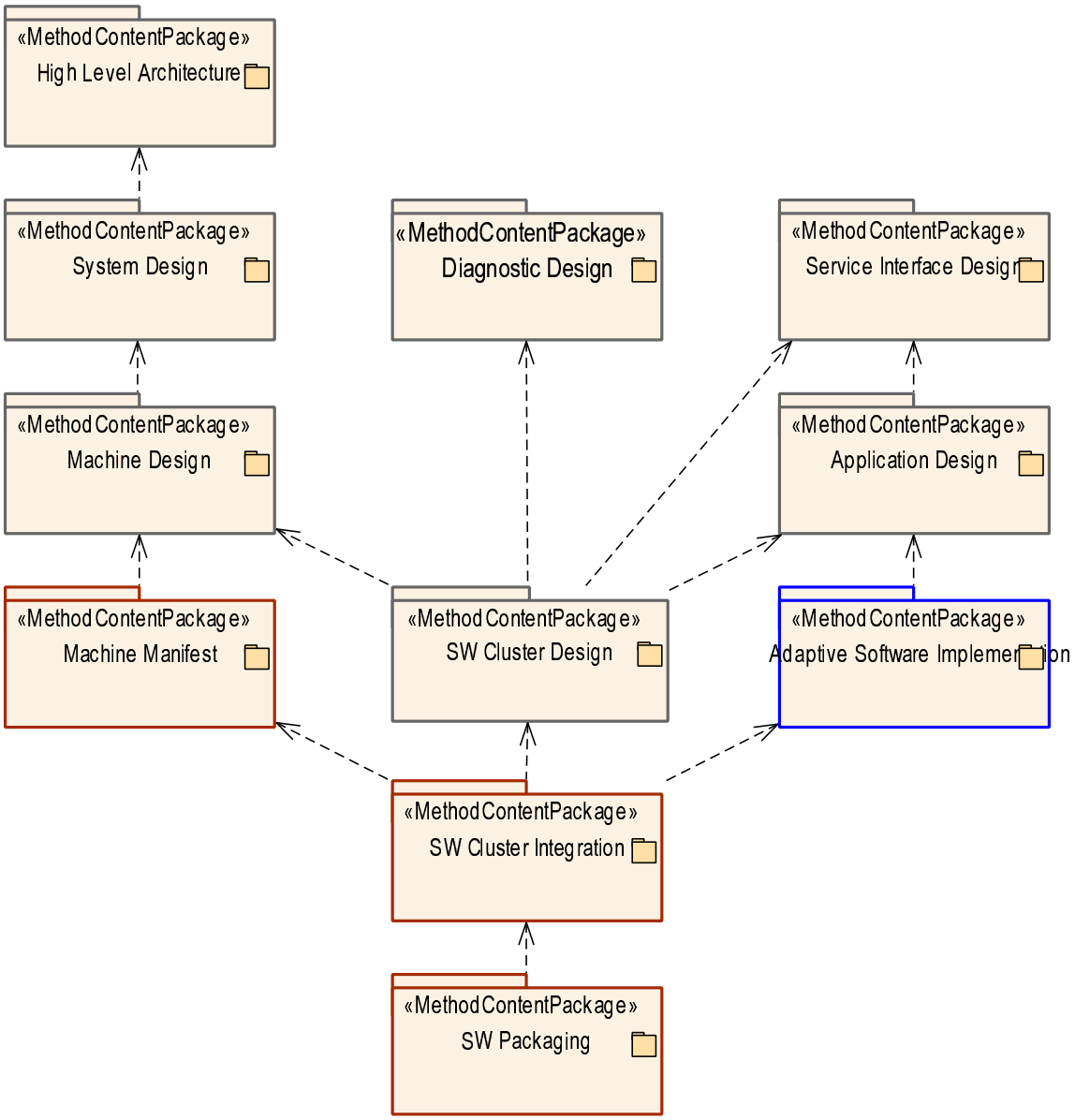

Figure 2.1: Adaptive Methodology

2.1 System - High Level Architecture

A typical vehicle will be most likely equipped with ECUs hosting
– ECU-Instances with software developed for the AUTOSAR classic platform (CP)
– and Machines with software developed for the AUTOSAR adaptive platform (AP)
– and eventually also HW entities with non-AUTOSAR software.

The software architecture and System design for the entire vehicle has therefore to cover all these ECUs as well as the communication between these ECUs, considering
– communication between AUTOSAR AP and CP hardware and software entities
– communication between AUTOSAR and non-AUTOSAR hardware and software entities).

Figure 2.2 gives an overview of the tasks and work products in scope of High Level Architecture. Please find the detailed definitions of these tasks and work products in Adaptive Methodology Library (see chapter 3.1).

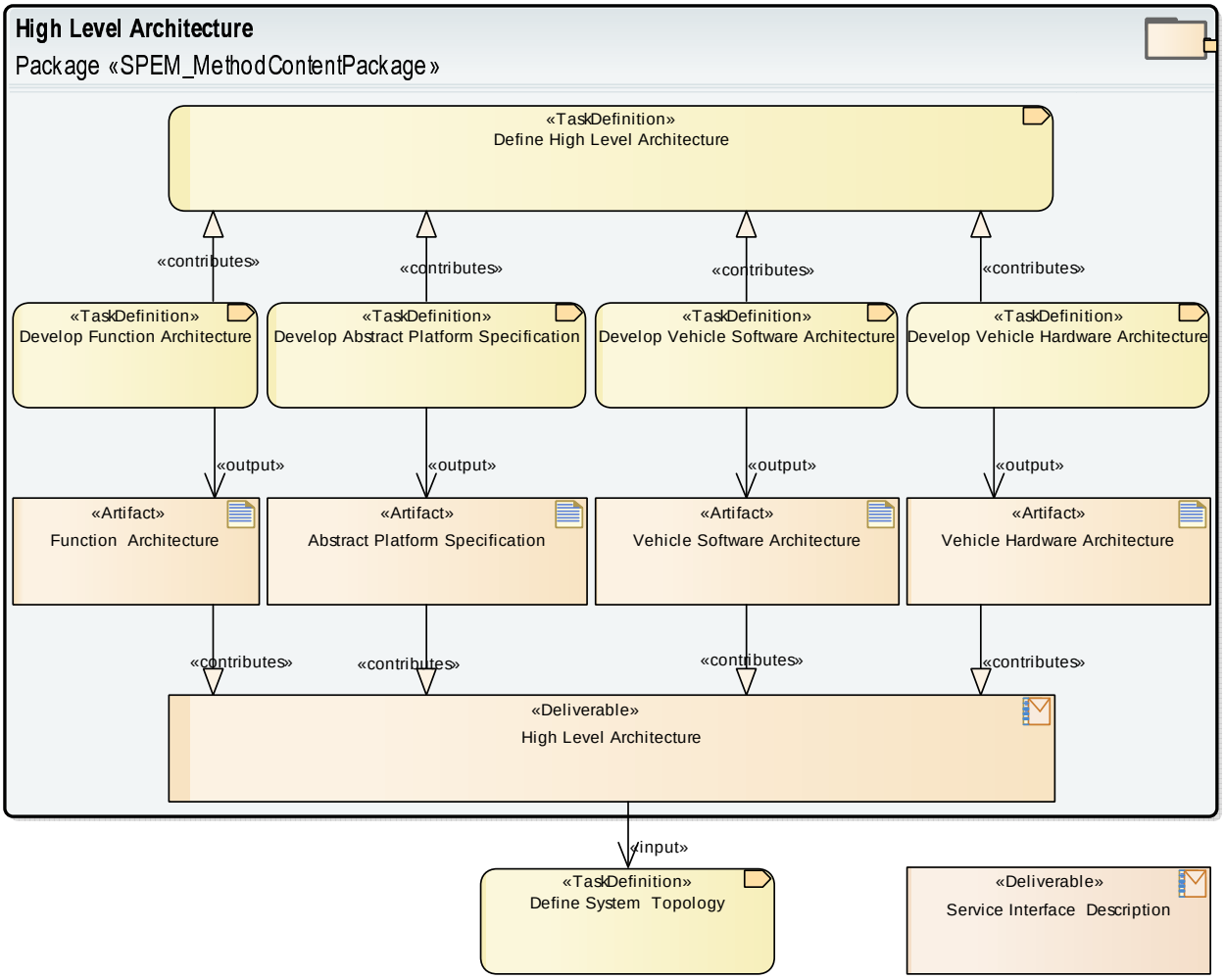

Figure 2.2: High Level Architecture

In a top-down approach the activities related to Define High Level Architecture start with Develop Function Architecture and/or Develop Abstract Platform Specification resulting in the Function Architecture and Abstract Platform Specification contributions to High Level Architecture:

1. Develop Function Architecture puts the focus on vehicle functions abstracting from their future implementation: An E/E system architect evaluates and specifies vehicle functions, features and requirements necessary for a specific E/E vehicle project or project family. The resulting Function Architecture (see [TR_AMETH_00201]) is likely a non-AUTOSAR document (or model) describing function networks representing functionalities that are needed to execute particular vehicle functions.

2. Develop Abstract Platform Specification puts the focus on a platform independent component model for vehicle functions and their interaction. The resulting Abstract Platform Specification identifies

   - abstract components disregarding their future implementation as AUTOSAR AP / CP / non-AUTOSAR software entities or as hardware entities.
   - communication endpoints based on abstract PortInterface descriptions (see [TR_AMETH_00001]).
   - connections between communication endpoints disregarding their future implementation via AUTOSAR AP service discovery or CP connectors or cross platform communication.

3. Develop Vehicle Software Architecture puts the focus on software components targeting at AUTOSAR AP or CP in any combination. [1] The resulting Vehicle Software Architecture (see [TR_AMETH_00202])

   - allocates functionalities to AP and CP platforms
   - and defines the required AP and CP intra and cross platform communication as well as the demand for "external" communication with non-AUTOSAR entities.

   Figure 2.3 outlines the derivation of Vehicle Software Architecture from Function Architecture.

   Please see also explanatory documents for AP and CP software architecture [7, Explanation of Adaptive Platform Software Architecture] and [8, Layered Software Architecture].

4. Develop Vehicle Hardware Architecture puts the focus on ECUs hosting the software entities from Vehicle Software Architecture. The resulting Vehicle Hardware Architecture specifies the required ECUs with

[1] Non-AUTOSAR software is out of scope here. In case of demand for "external" communication with non-AUTOSAR entities both sides need to exchange messages compatible on bus-level.

their HW resources for AP (Machines), CP (ECU-Instances) and communication infrastructure.
Figure 2.6 outlines the association of software entities from Vehicle Software Architecture to hardware entities from Vehicle Hardware Architecture.

[TR_AMETH_00001] Identify Abstract Port Interfaces

Status: DRAFT
Upstream requirements: RS_METH_00201, RS_METH_00032

⌈This activity specifies capabilities of PortInterfaces required for communication endpoints in Abstract Platform Specification.

The abstract PortInterfaces as defined in [9, Specification of Abstract Platform] specify

- *commands* as an abstracted form of AP methods and CP client/server operations
- *indications* as an abstracted form of AP events or fields and CP sender/receiver data prototypes
- *attributes* as an abstracted form of AP fields and collected CP client/server operations, sender/receiver data prototypes
- all using abstracted data types outlining the demand for future concrete AP and CP data types.

Please be aware that this is independent of
– any assignment to a specific network binding,
– any assignment to platform specific software components types
and may be seen as a preparation step towards the development of AP or CP Software entities.

This activity is optional and will not show up in a bottom-up approach.⌋

2.1.1 Overall System View

[TR_AMETH_00201] Develop a Function Architecture

Status: DRAFT
Upstream requirements: RS_METH_00032

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

The Function Architecture consists of a number of vehicle functions with their interfaces and the corresponding connections.

In an analysis step, a vehicle function may be detailed into a number of function blocks which later lead to the abstract SWCs in the derived Abstract Platform Specification:

- A function block encapsulates a specific functionality.
  Note, that function blocks may be realized in software or hardware or as a mix of both.
- A composition of function blocks represents the functionality that is needed to execute a particular vehicle function.

This activity is optional and will not show up in a bottom-up approach.⌋

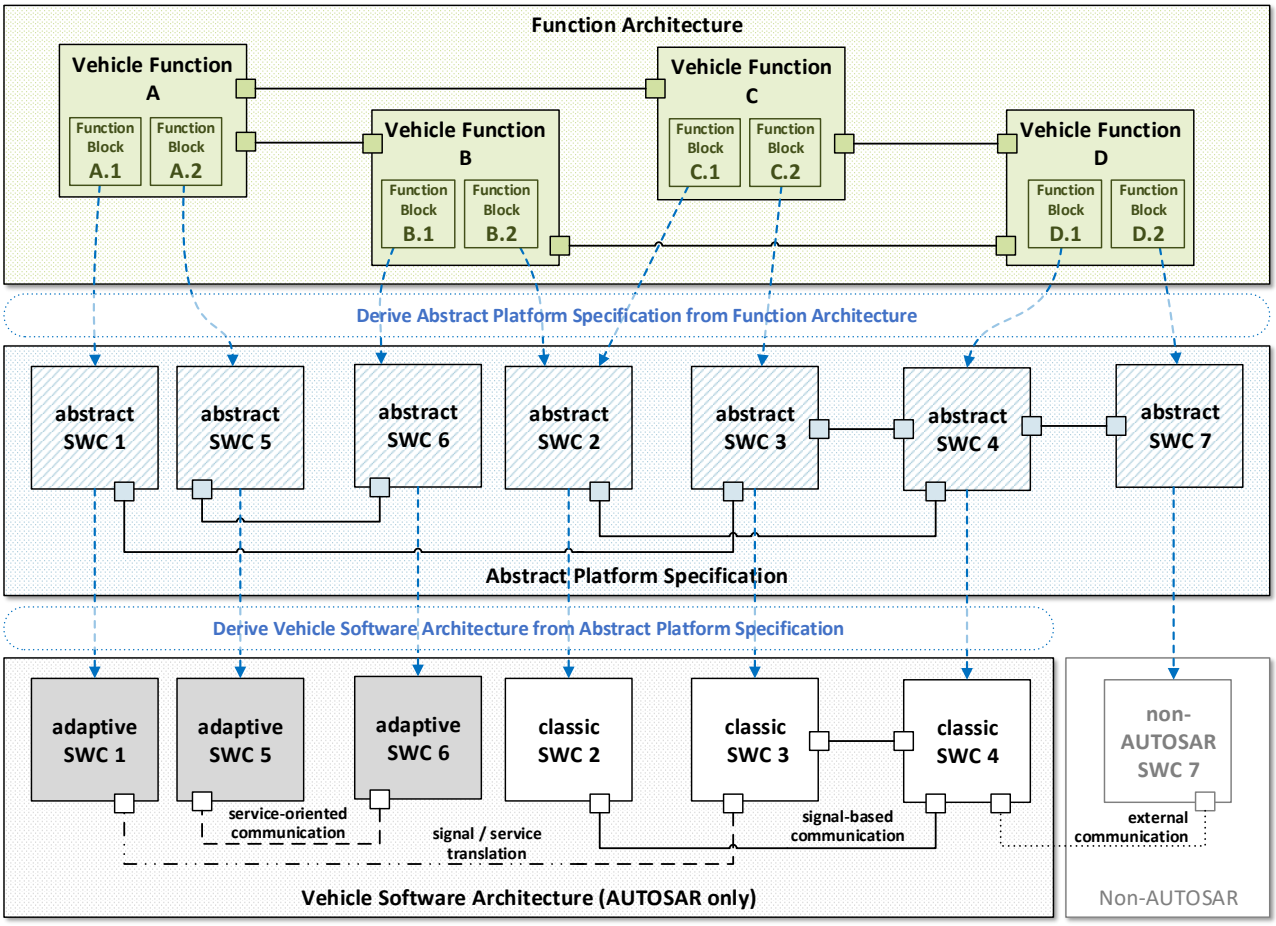

Figure 2.3: From the Function Architecture to a Vehicle Software Architecture

[TR_AMETH_00202] Develop a Vehicle Software Architecture

Status: DRAFT
Upstream requirements: RS_METH_00032

⌈An engineer, e.g. a software architect, could take the Function Architecture and/or a Abstract Platform Specification as input to derive a corresponding

Vehicle Software Architecture while executing the activity Develop Vehicle Software Architecture.

The Vehicle Software Architecture provides a dedicated view of all AUTOSAR software entities and their communication relations within the E/E vehicle system, with

- AUTOSAR software components of the Adaptive Platform (AP SWCs),
- AUTOSAR software components of the Classic Platform (CP SWCs),
- AUTOSAR software compositions with arbitrary combinations of AP and/or CP SWCs,
- the communication between software components
  - local to Vehicle Software Architecture with
    - adaptive (i.e. service oriented) communication between AP SWCs
    - classic (i.e. signal based) communication between CP SWCs
    - service oriented communication between AP and CP SWCs via SOME/IP
    - communication between AP and CP SWCs via signal/service translation
  - with external software entities (e.g. non-AUTOSAR SWCs not covered by Vehicle Software Architecture).

This activity is optional and will not show up in a bottom-up approach.⌋

2

Please be aware that SWCs allocated to different ECUs may exchange data through the communication infrastructure.
The communication end points of SWCs are Ports typed by a particular PortInterface definition. In case of the Adaptive Platform, interfaces are expressed as Service Interfaces tailored for the individual service oriented communication use cases.

[2]Figure 2.3 From the Function Architecture to a Vehicle Software Architecture shows that a vehicle function may be implemented by one or more software components targeting either at an AUTOSAR Adaptive Platform stack either at an AUTOSAR Classic Platform stack.

2.1.2 Derive Sub-Systems

[TR_AMETH_00203] Derive Sub-Systems

Status: DRAFT
Upstream requirements: RS_METH_00078, RS_METH_00079

⌈A sub-system is a reduced part of the overall technical system and emphasizes on relevant aspects of it. Feasible use cases are

- derive a pure AUTOSAR Classic Platform sub-system as `VFB` view
- derive a pure AUTOSAR Adaptive Platform sub-system
- derive a view on a mixed Adaptive/Classic Platform sub-system.

Motivations for the decomposition into sub-systems are

- functional aspects, e.g. scoping of sub-system by vehicle functions
- technology aspects, e.g. scoping of sub-system by AUTOSAR platform
- logistics, e.g. scoping of sub-system by intended sub-contracting

Any sub-system may interact with other sub-systems: in such cases software components allocated to different sub-systems communicate. To support this the sub-system shall include `PortInterface` definitions at least for the AUTOSAR Adaptive and Classic Platform software component ports interfacing to other sub-systems.

This activity is optional.⌋

[3]

[3] Figure 2.4 Extract sub-systems from `Vehicle Software Architecture` shows three possible views on sub-systems derived from the `Vehicle Software Architecture`.

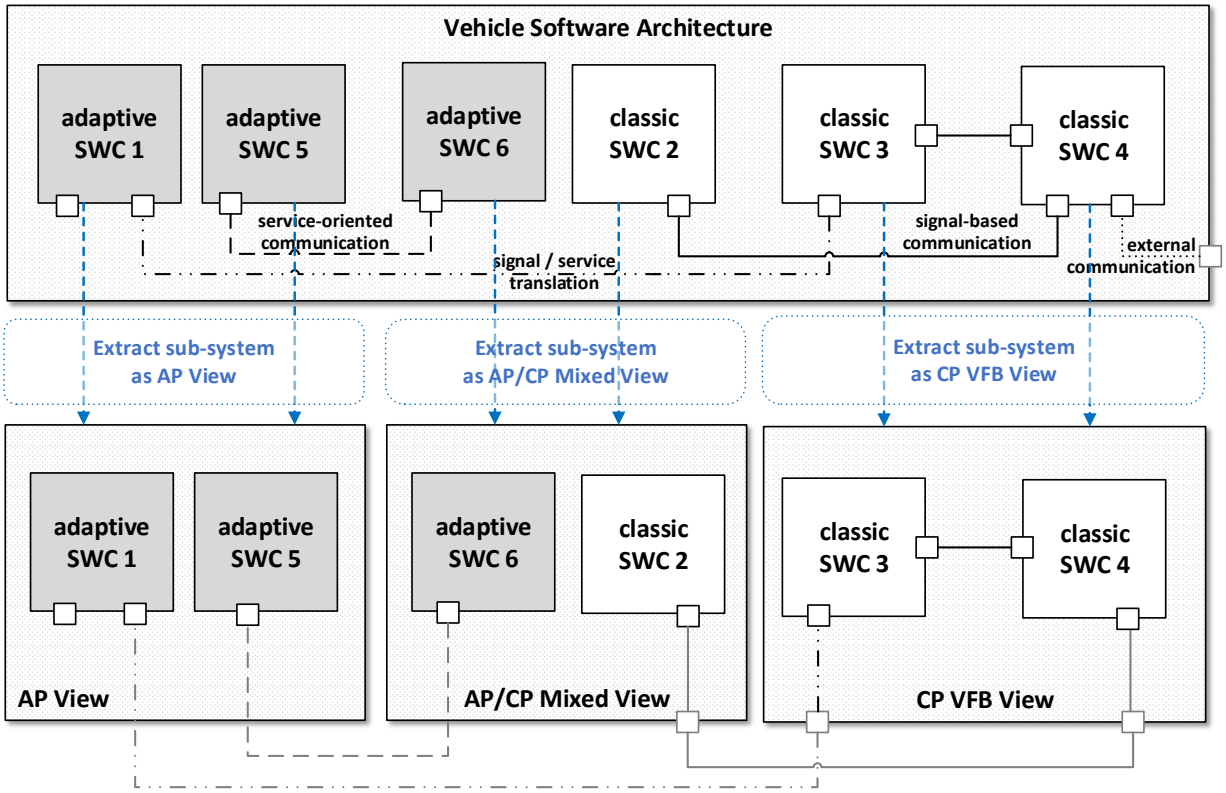

Figure 2.4: Extract sub-systems from `Vehicle Software Architecture`

The overall `Vehicle Software Architecture` covers a whole `System` considering all AUTOSAR software entities of a vehicle. The next steps in a `top-down approach` are related to the decomposition into sub-systems - eventually with arbitrary intermediate levels - until platform specific sub-systems (see [TR_AMETH_00203]) are reached.

- A mixed `AP` and `CP` sub-system will be handled like `Vehicle Software Architecture` with limited scope (covering only parts of the AUTOSAR software in a vehicle).
- A `CP` sub-system extracted from `Vehicle Software Architecture` covers `CP` software of arbitrary `CP ECU-Instance`s in a vehicle. This `CP` system will be handled as per `CP` Methodology with the step by step decomposition into System Extract for multiple ECUs, System Extract for a single ECU and ECU Extract.
- A `AP` sub-system extracted from `Vehicle Software Architecture` covers `AP` software of arbitrary `AP Machine`s in a vehicle. This `AP` System will be handled as per `AP` Methodology with the decomposition into Machine (Design) Extract down to Software Cluster Extract.
- At least the communication endpoints of extracted sub-systems need (outlined but technology specific) `PortInterface` descriptions (as concretion of the abstract `PortInterface`s from [TR_AMETH_00001]).

Figure 2.4 outlines the derivation of sub-systems from Vehicle Software Architecture.

Please be aware that Vehicle Software Architecture is required only in a top-down approach at Define High Level Architecture.
In a bottom-up approach the Vehicle Software Architecture is clearly optional but may be aggregated from the individually created CP and/or AP sub-systems.

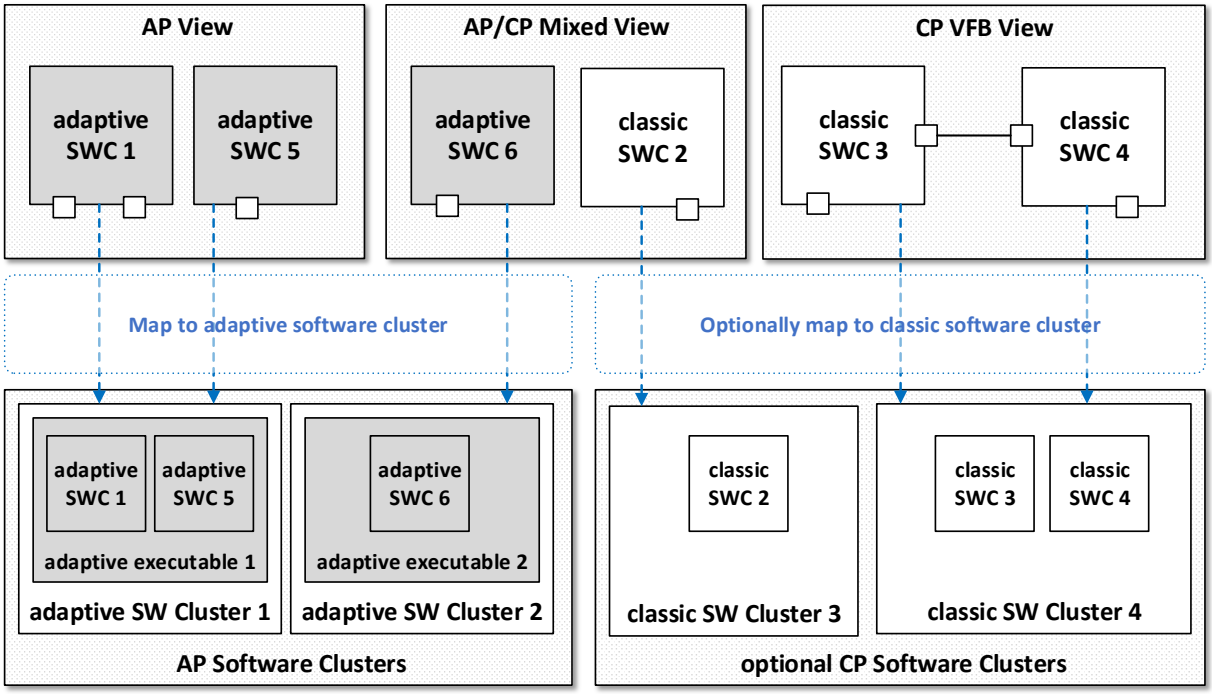

Figure 2.5: Specify software clusters for sub-systems

The adaptive and classic SWCs identified in Vehicle Software Architecture need to be deployed on ECUs identified in Vehicle Hardware Architecture:

- Figure 2.5 outlines the grouping of SWCs into software clusters.
  - This is mandatory in AP only: here SWCs are integrated to an executable that can be deployed to a specific Machine.
  - For CP software clusters allow a grouping of SWCs to be deployed together to a specific ECU-Instance.
- Finally we can map the software clusters to HW entities.
  Figure 2.6 outlines the allocation of software clusters from Vehicle Software Architecture to HW entities for AP (Machines) and CP (ECU-Instances) from Vehicle Hardware Architecture.

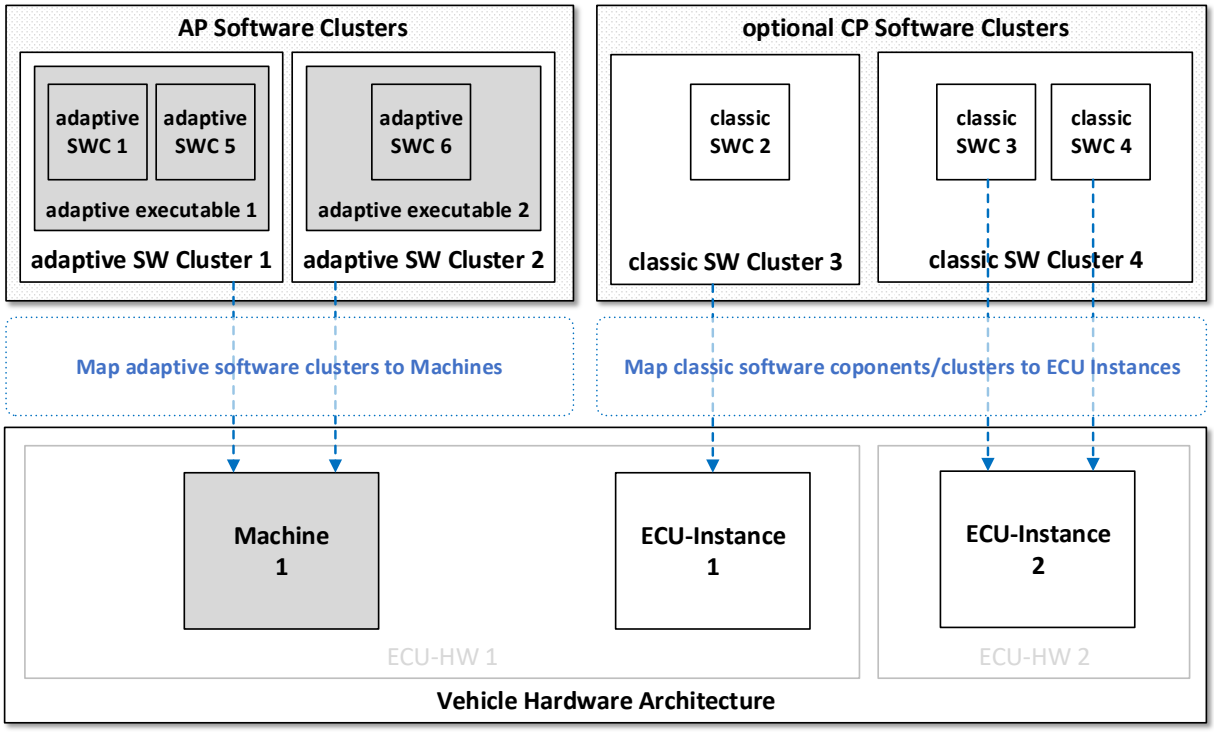

Figure 2.6: Map software entities from `Vehicle Software Architecture` to `ECU-Instance`s or `Machine`s

The overall `Vehicle Hardware Architecture` (as outlined in figure 2.7) covers the topology of all `ECU`s along with the interconnecting communication infrastructure in a vehicle. This hardware topology consists of

- Nodes representing physical or virtual `ECU-HW` items that host
  either a specific `CP ECU-Instance`
  or a specific `AP Machine`
  or a specific combination of these.
- Connectors between nodes representing the communication possibilities.
  This allows to take part in the communication clusters of the required communication technologies like `CAN` or `Ethernet` for `CP` and `SOME/IP` for `AP`.
- Connectors for over-the-air communication (e.g. for vehicle-to-vehicle communication use cases or software update over-the-air use cases).

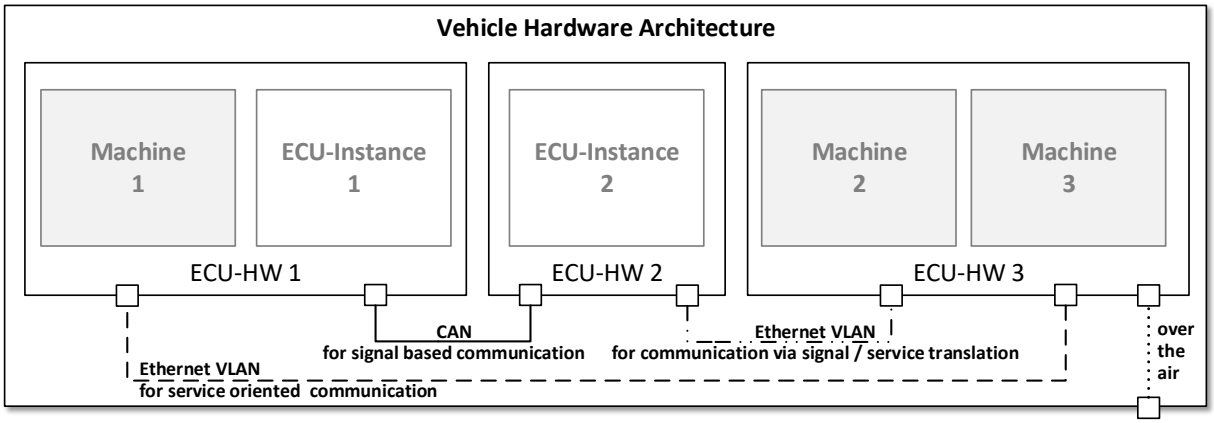

Figure 2.7: Specify the overall `Vehicle Hardware Architecture`

2.2 System Design

Figure 2.8 gives an overview of the tasks and work products in scope of `System Design`. Please find the detailed definitions of these tasks and work products in Adaptive Methodology Library (see chapter 3.2).

Main inputs are `Vehicle Software Architecture` and `Vehicle Hardware Architecture` from `High Level Architecture`, see also chapter 2.1.2. The `System Design` represents a – preferably platform specific – sub-system derived from `High Level Architecture`.

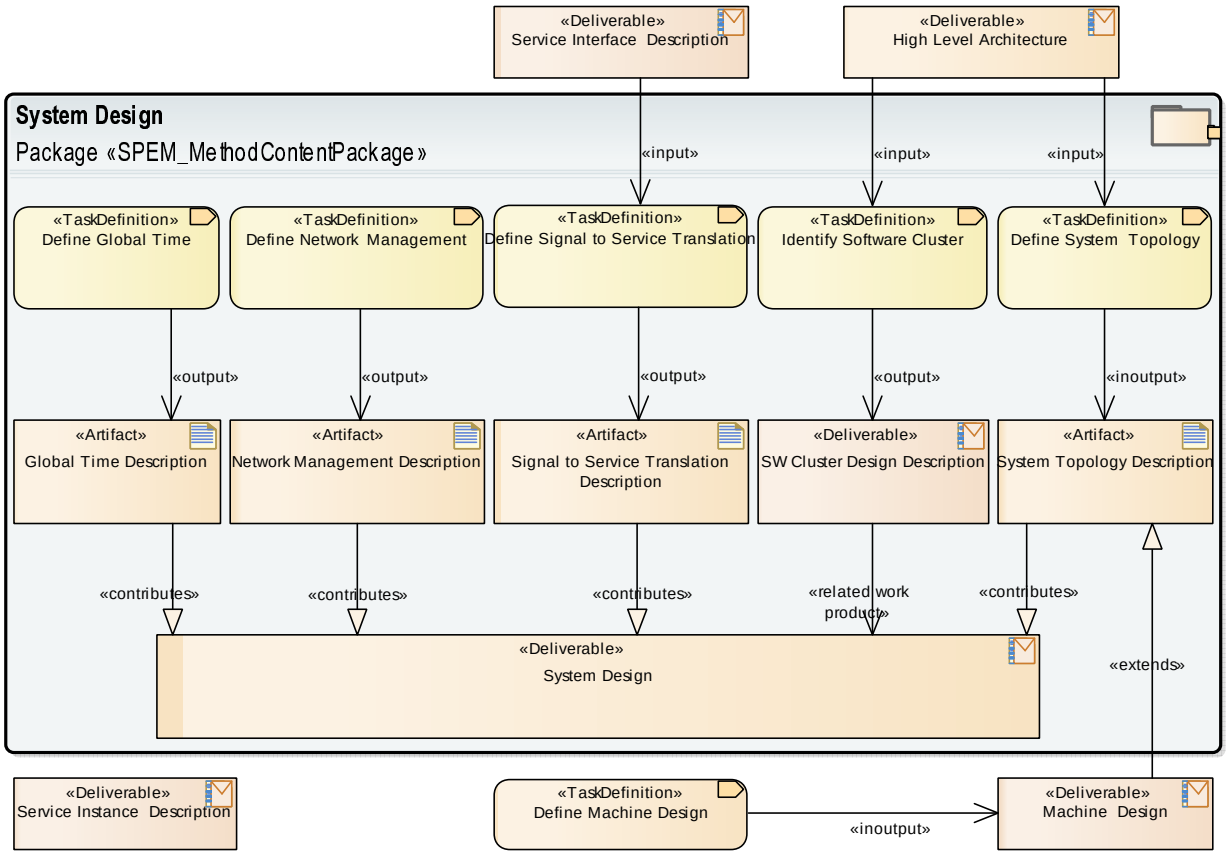

Figure 2.8: System Design

The `System Design` consists of

1. `System Topology Description` see 2.2.1.1
2. `SW Cluster Design Description` see 2.2.1.2
3. `Global Time Description` see 2.2.1.3
4. `Network Management Description` see 2.2.1.4
5. `Signal to Service Translation Description` see 2.2.1.5

2.2.1 System Design Contributions

2.2.1.1 System Topology Description

System Topology Description as outcome of task Define System Topology with the intention to refine the Vehicle Software Architecture and to define sub-systems with the required functionalities and their communication needs. This includes:

- Specify groups for distributed but coherent functionality. These function groups will be detailed as SW Cluster Design Descriptions during Software Cluster Design.

  In AP the Function Group Description defines a set of cohesive executable instances (aka application processes) likely having run-time interdependencies.

- Specify the network connectivity for the communication clusters of the required communication technologies like CAN or Ethernet for CP and SOME/IP for AP. [4]

  For Ethernet (and of course also SOME/IP) the Network Endpoint Description defines the network addresses (as Internet Protocol version 4 or version 6 address and/or Media-Access-Control (MAC) multicast address) for the individual communication connectors of ECUs (or to be more precise of the CP-ECU-Instances and AP-Machines) taking part in a communication cluster.

- Specify the network of services as Service Topology Description. This includes the specification of service interfaces (see chapter 2.3), the definition of technology specific services [5] [6], the instantiation of service providers, the allocation of service instances to the CP-ECU-Instances and AP-Machines and finally also the allocation of service instances to network endpoints.

  Optionally also the instantiation of service consumers and the connectivity between service providers and consumers via provided and required service instances may be specified.

  Please be aware that this is abstracted from the actual functionality of service providers and consumers but constrains the technology binding of their implementation.

[4] A physical channel is the transmission medium that is used to send and receive information between communicating ECUs. This element represents a physical connection (in case of CAN, FlexRay, LIN) or a logical connection (VLAN in case of Ethernet) between communicating devices.

[5] Services are based on technology independent interfaces but require technology specific IDs to be used in service oriented communication. Therefore service (instances) need to be explicitly specified with a technology binding for SOME/IP or DDS etc..

[6] The demand for service providers and consumers and their connectivity is visible as ports and connectors in Abstract Platform Specification

2.2.1.2 SW Cluster Design Description

Some initial SW Cluster Design Description as outcome of task Identify Software Cluster with the intention to define the major building blocks in the subsystem software architecture. See chapter 2.9 Software Cluster Design for the actual elaboration of SW Cluster Design Description.

We shall/may group software components into software clusters with the intention to deploy the software clusters on an ECU. For CP this is an optional grouping to facilitate the mapping of SWCs to ECU-Instances. For AP this is a mandatory grouping of SWCs taking part in the build of executables to be deployed on a specific Machine.

See also chapter Software Distribution in [6, Manifest Specification] and chapter Software Cluster in [10, System Template] .

2.2.1.3 Global Time Description

Global Time Description as outcome of task Define Global Time with the intention to take part in the vehicle-wide time synchronization of ECUs. [7]

It may be necessary that several ECUs in a vehicle E/E system need to act in concert while executing a (distributed) vehicle function. To achieve this the global time functional clusters across all connected CP-ECU-Instances and AP-Machines need to synchronize to the same time bases.

Please see in the schematic figure 2.9 Synchronize functional clusters across CP-ECU-Instances and AP-Machines how the involved global time functional clusters (labeled **X** in diagram) may exchange synchronization messages through the communication infrastructure.

Global Time Description for AP describes how the Machines in scope take part in the Global Time Synchronization of a vehicle. This happens via Ethernet. For CP the Global Time Synchronization may happen via CAN or Ethernet.

See also chapter Time Synchronization Deployment in [6, Manifest Specification] and chapter Global Time Synchronization in [10, System Template].

[7] A very trivial example for this requirement is the activation of turn indicators in a car. These are rarely connected to a single ECU (which could take care of synchronously flashing the turn indicators) but their synchronized execution is still essential for the vehicle operation.

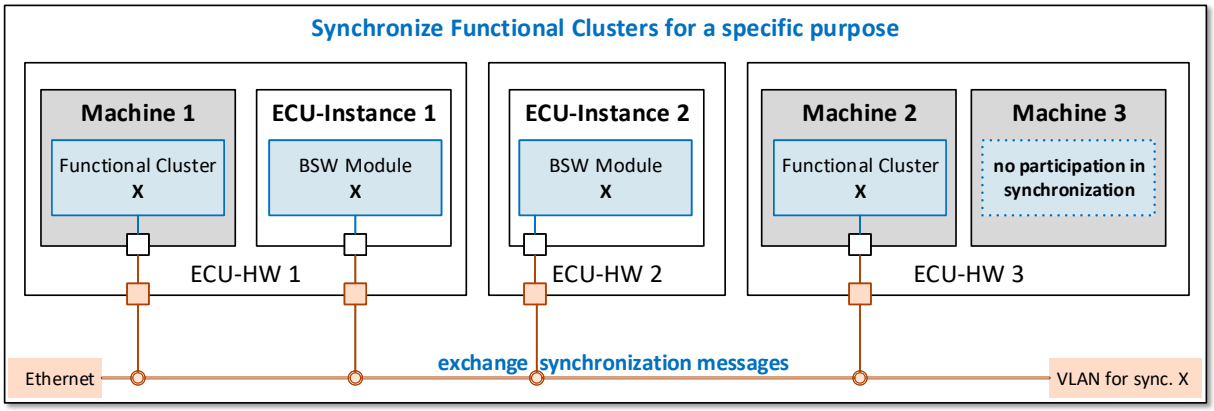

Figure 2.9: Synchronize functional clusters across CP-ECU-Instances and AP-Machines

2.2.1.4 Management Description

Network Management Description as outcome of task Define Network Management with the intention to take part in the vehicle-wide wake-up and sleep behavior of Networks (and network-connectors of ECUs).

Please see in the schematic figure 2.9 Synchronize functional clusters across CP-ECU-Instances and AP-Machines how the involved network management functional clusters (labeled **X** in diagram) may exchange synchronization messages through the communication infrastructure.

Typically an OEM provides the network configuration with all configuration settings that are relevant for the network management functional clusters across all connected CP-ECU-Instances and AP-Machines in a vehicle.

Network Management Description for AP describes how the Machines in scope take part in the overall network management in a vehicle. [8]

See also TPS_MANI_03166 and TPS_MANI_03226 and related explanations in [6, Manifest Specification] and chapter Network Management in [10, System Template].

[8] AUTOSAR Adaptive Network Management (NM) is based on periodic NM messages between nodes in the network. The reception of NM messages indicates that the sender of this message wants to keep a partial network awake. If any node is ready to go to sleep mode, it stops sending NM messages, but as long as NM messages from other nodes are received, it postpones the transition to sleep mode.

2.2.1.5 Signal to Service Translation Description

Signal to Service Translation Description as outcome of task Define Signal to Service Translation with the intention to define how SOME/IP serialized communication of AP can be translated into signal based communication of CP and vice versa.

See chapter 2.2.4 Signal to Service Translation for details.

The above mentioned tasks may be executed - as per demand - in arbitrary order.

2.2.2 System Design Usage Scenario (top-down)

Figure 2.10 shows the tailored task and work product definitions of System Design in a top-down usage scenario targeting at a vehicle wide system description. This may start from scratch or from some base line system description.

Elements are:

- Define (complete) System Topology with
  - Decompose System into Sub-Systems
  - Define Function Groups
  - Define Network Endpoints
  - Define Service Topology

  (see overview in list item 2.2.1.1 on page 31)
- Identify Software Clusters for function groups:
  Define the major building blocks in the sub-system software architecture (see overview in list item 2.2.1.2 on page 32) for software entities and sub-systems identified in Vehicle Software Architecture, see also [TR_AMETH_00203].
- Define system-wide Global Time:
  Define the vehicle-wide time synchronization of ECUs (see overview in list item 2.2.1.3 on page 32) .
  This enables the synchronization of the global time functional clusters across all connected CP-ECU-Instances and AP-Machines to same time bases.
- Define system-wide Network Management:
  Define the vehicle-wide wake-up and sleep behavior of Networks (see overview in list item 2.2.1.4 on page 33).
  This enables the network management functional clusters across all connected CP-ECU-Instances and AP-Machines to control the network-connectors as per expected wake-up and sleep behavior.

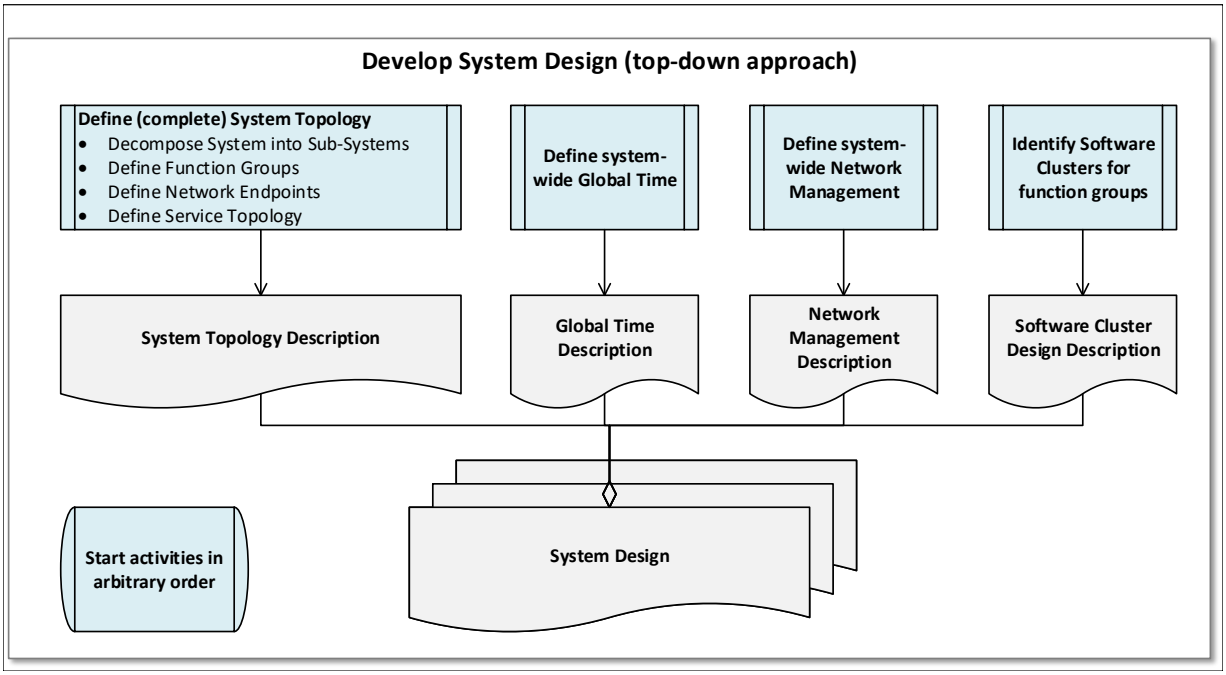

Figure 2.10: How to elaborate System Design (top-down approach)

The top-down approach leads to the following work products tailored for `System`-scope:

- `System Topology Description`
  as outcome of `Define (complete) System Topology` considering sub-systems, function groups, network endpoints and the service topology as per current system design scope.
- `Global Time Description`
  as outcome of `Define system-wide Global Time`
- `Network Management Description`
  as outcome of `Define system-wide Network Management`
- `SW Cluster Design Description`s
  as outcome of `Identify Software Clusters for function groups` identifying and describing the software clusters implementing the function groups as per current system design scope.

2.2.3 System Design Usage Scenario (bottom-up)

Figure 2.11 shows the tailored task and work product definitions of `System Design` in a bottom-up usage scenario targeting at a `Contribution to System Design` for a specific `AP-Machine`.

Elements are:

- Define the System Topology Contribution for a Machine
  with the intention to Define the Sub-System for a Machine and Contribute Network Endpoints for a Machine
  eventually in combination with Define the Machine Design Locally (see chapter 2.7.1 Machine Design Usage Scenario)
- Outline the Sub-System in scope of a specific Machine
  to identify the SoftwareClusters in scope.

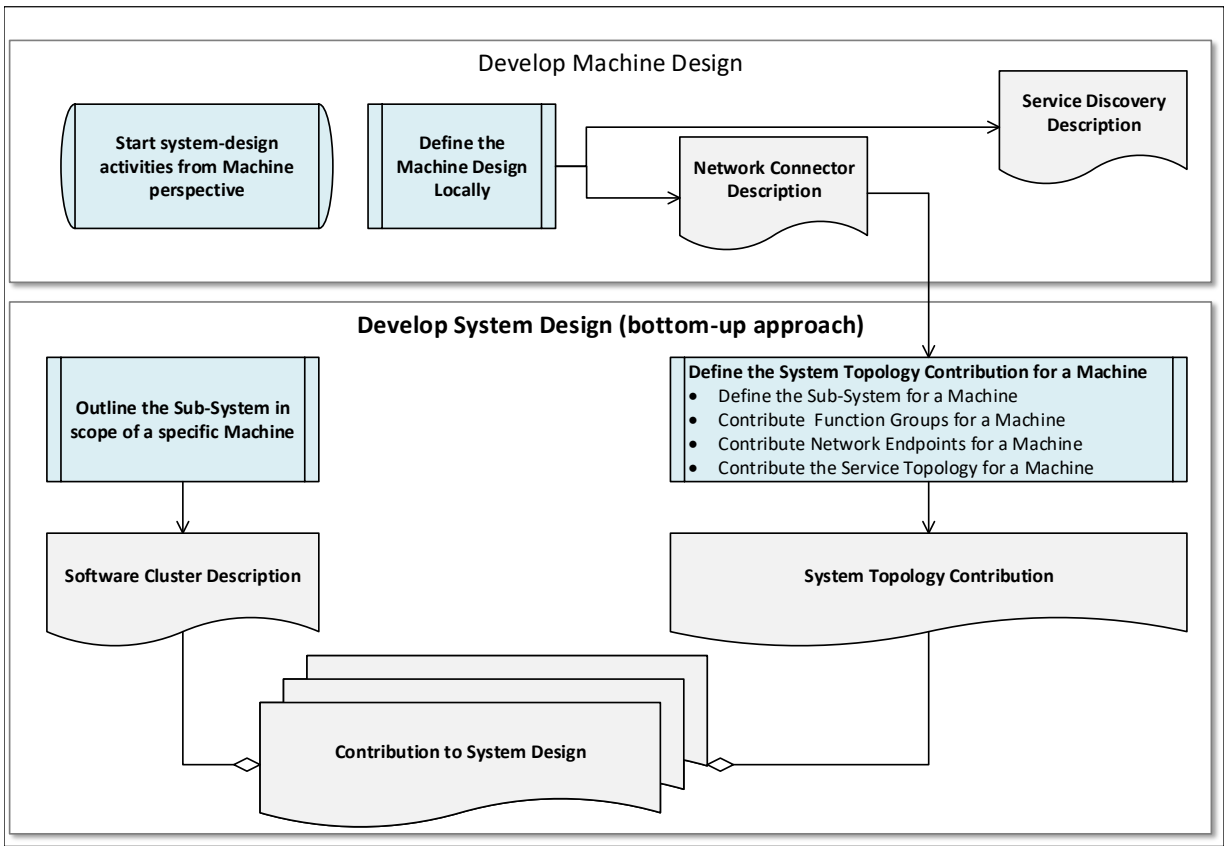

Figure 2.11: How to elaborate System Design (bottom-up approach)

The bottom-up approach leads to the following work products tailored for Machine-scope:

- Software Cluster Descriptions
  as outcome of Outline the Sub-System in scope of a specific Machine identifying the software clusters implementing the function groups as per current machine scope.
- System Topology Contribution
  as outcome of Define the System Topology Contribution for a Machine considering function groups, network endpoints and the service topology as per current machine scope.

2.2.4 Signal to Service Translation

Figure 2.12 Signal to Service Translation outlines the activity of designing the service oriented communication between Classic and Adaptive Platform software entities, if (and only if) the CP side does not directly and completely fulfill the SOME/IP message format.

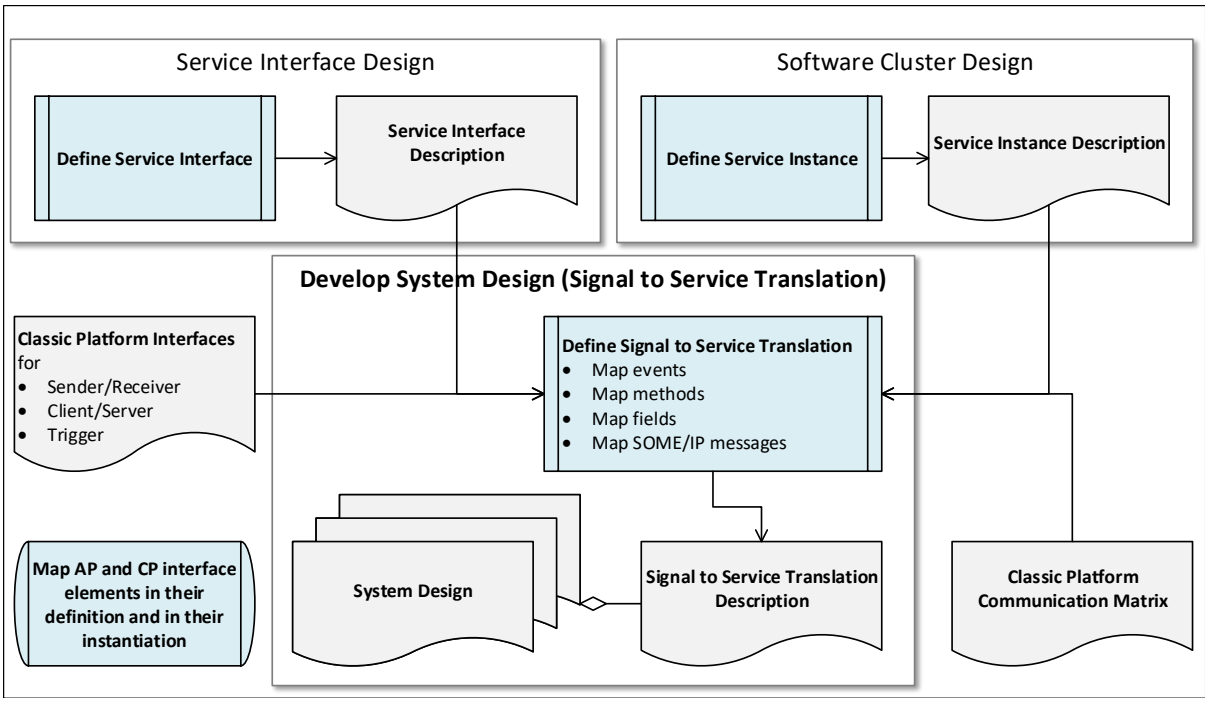

Figure 2.12: Signal to Service Translation

Remarks:

- The background of this activity is the request to enable service oriented communication between applications of a Classic Platform (CP) ECU-Instance and those of an Adaptive Platform (AP) Machine via SOME/IP.
- If CP communicates as per SOME/IP no signal to service translation is necessary. This is the case if
  - CP communicates directly via Ethernet and fulfills the SOME/IP message format.
  - a Gateway does the necessary translation (e.g. from CAN) to the SOME/IP message format.
- Otherwise a signal to service translation as per [TR_AMETH_00207], [TR_AMETH_00208] and [TR_AMETH_00210] has to do the job:
  - Unfortunately CP does not support service interfaces. Thus, the equivalent to an AP service interface may be composed of different types of CP Port-Interfaces like Sender/Receiver, Client/Server or Trigger interfaces.

– A signal to service translation specification needs to cover the mapping of AP and CP interface elements as well as the mapping of data instances (i.e. SOME/IP messages from AP service instances and CP signals).

[TR_AMETH_00207] Design communication between Classic Platform (CP) ECU-Instances and Adaptive Platform (AP) Machines

Status: DRAFT
Upstream requirements: RS_METH_00202

⌈Adaptive Software communicates in a service oriented manner. However, a typical vehicle will be equipped with ECU-HW that hosts CP ECU-Instances and AP Machines in any combination.

Thus, it is very likely that ECU-Instances and Machines need to communicate:

- If the ECU-Instance implements SOME/IP the service oriented communication with Machines can be achieved.
  - This works directly if both sides implement compatible SOME/IP messages on bus level.
  - In case of incompatible SOME/IP messages a Gateway is required to achieve compatibility on bus level
- If the ECU-Instance communicates in a signal based way (e.g. via CAN or Ethernet in a non-SOME/IP use case) a Signal to Service Translation Description is needed.
  - Both sides use different PortInterface types, the artifact Signal to Service Translation Description documents this communication scenario.
  - The context includes a PDU or Ethernet-socket on CP side and a Service Instance on AP side.

⌋

[TR_AMETH_00208] Design Signal to Service translation between AUTOSAR Classic Platform (CP) and Adaptive Platform (AP)

Status: DRAFT
Upstream requirements: RS_METH_00200, RS_METH_00202

⌈

In order to describe the communication between AP and CP even in use cases where the CP side does not fully comply to SOME/IP, the activity Define Signal to Service Translation describes the mapping of the elements of one or more CP PortInterfaces to the elements of a single AP Service Interface in scope of an individual Service Instance:

- map AP method(s):
  - map a CP Client/Server Operation to a method of the Service Interface.
  - map a CP Trigger to a "Fire and Forget" method of the Service Interface.
- map AP event(s):
  - map a CP Sender/Receiver Data Prototype to an event of the Service Interface.
- map AP field(s):
  - map CP Client/Server Operations to field-getter/setter methods of the Service Interface
  - map a CP Sender/Receiver Data Prototype to a field-notifier of the Service Interface.
- map AP SOME/IP message(s) as per service instance:
  - consider the service instance ID as context in service discovery
  - map event messages to ISignal triggerings
  - map method messages to ISignal triggerings

The resulting artifact Signal to Service Translation Description may be used to generate translation code.⌋

[TR_AMETH_00210] Map signals to services

Status: DRAFT
Upstream requirements: RS_METH_00200, RS_METH_00202

⌈For a Signal to Service Translation Description all Service Interface elements of a Service Instance Description are mapped to the corresponding items of a signal-based communication protocol like CAN or FlexRay.

- for methods see TPS_MANI_03125 in [6, Manifest Specification]
- for events see TPS_MANI_03124 in [6, Manifest Specification]
- for fields see TPS_MANI_03126 in [6, Manifest Specification]
- for the concrete instance context see TPS_MANI_03000 in [6, Manifest Specification]

⌋

2.3 Service Interface Design

Figure 2.13 gives an overview of the tasks and work products in scope of `Service Interface (Design) Description`. Please find the detailed definitions of these tasks and work products in Adaptive Methodology Library (see chapter 3.8).

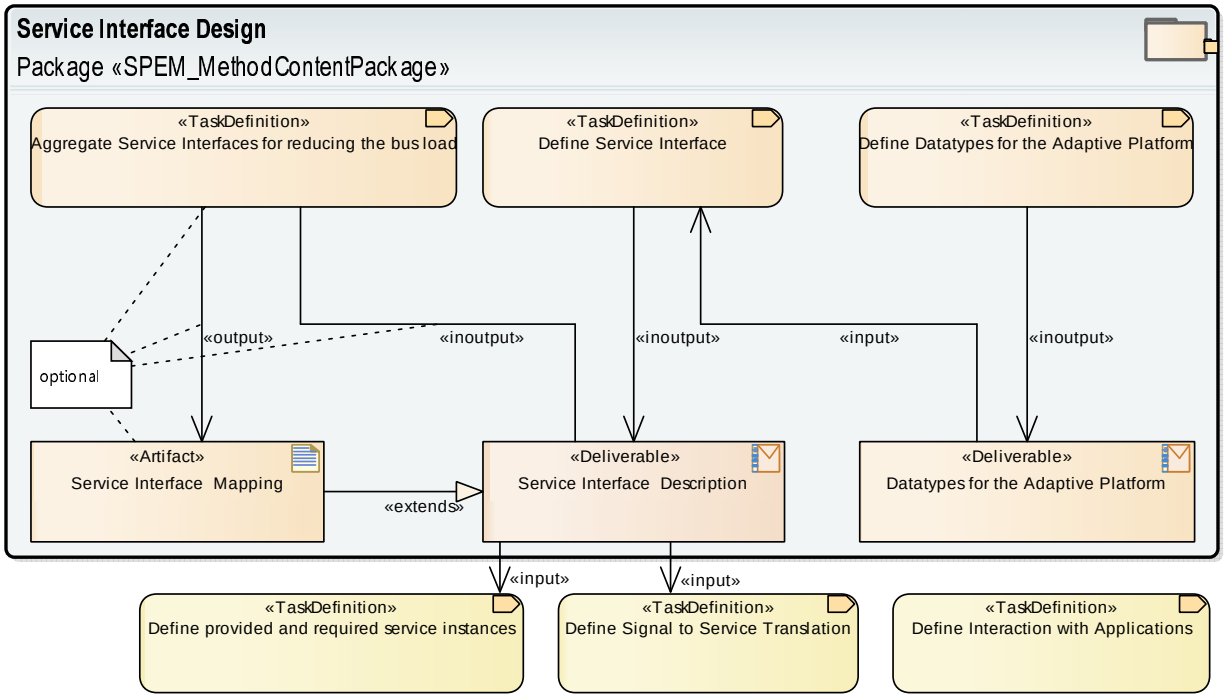

Figure 2.13: Service Interface Design

The resulting work products are

- `Service Interface Description` as outcome of `Define Service Interface`
  eventually enhanced by `Service Interface Mapping` as outcome of `Aggregate Service Interfaces for reducing the bus load`
- `Datatypes for the Adaptive Platform` as outcome of `Define Datatypes for the Adaptive Platform`

[TR_AMETH_00008] Develop Service Interfaces for Adaptive Software

Status: DRAFT
Upstream requirements: RS_METH_00201

⌈All service interfaces used in `Adaptive Software` need to be defined as `Service Interface Description` with event, method and field details.

They are the basis for the header file generation. Therefore, it is also possible to define name spaces within a service interface, which has a direct influence on the generated code.⌋

[TR_AMETH_00009] Aggregating service interfaces for reducing the bus load

Status: DRAFT
Upstream requirements: RS_METH_00201

⌈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.⌋

[TR_AMETH_00007] Definition of data types for the Adaptive Platform

Status: DRAFT
Upstream requirements: RS_METH_00201

⌈`Datatypes 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 `Datatypes for the Adaptive Platform` include C++ implementation data types (supporting vectors, strings, maps etc.).⌋

For more information on data types as specified for the Classic Platform and the extensions for the Adaptive Platform, see [6, Manifest Specification] and [11, Software Component Template].

2.3.1 Service Interface Design Usage Scenario

Figure 2.14 shows the usage scenario of how to obtain `Service Interface Descriptions`

- either in create/change use cases based on the tailored tasks:
  - `Define (or re-use) Service Interface`
  - `Define (or re-use) Datatypes for a Service Interface`
  - this may be done in the context of system and/or application design activities in top-down approach
- either in re-use use cases based on the tailored tasks:
  - `Re-use Service Interface`
  - `(Define or) re-use Datatypes for a Service Interface`

  - this may be done in the context of system and/or application design activities in bottom-up approach

- resulting in tailored work products
  - Service Interface Description
  - Data Type Descriptions for Service Interfaces

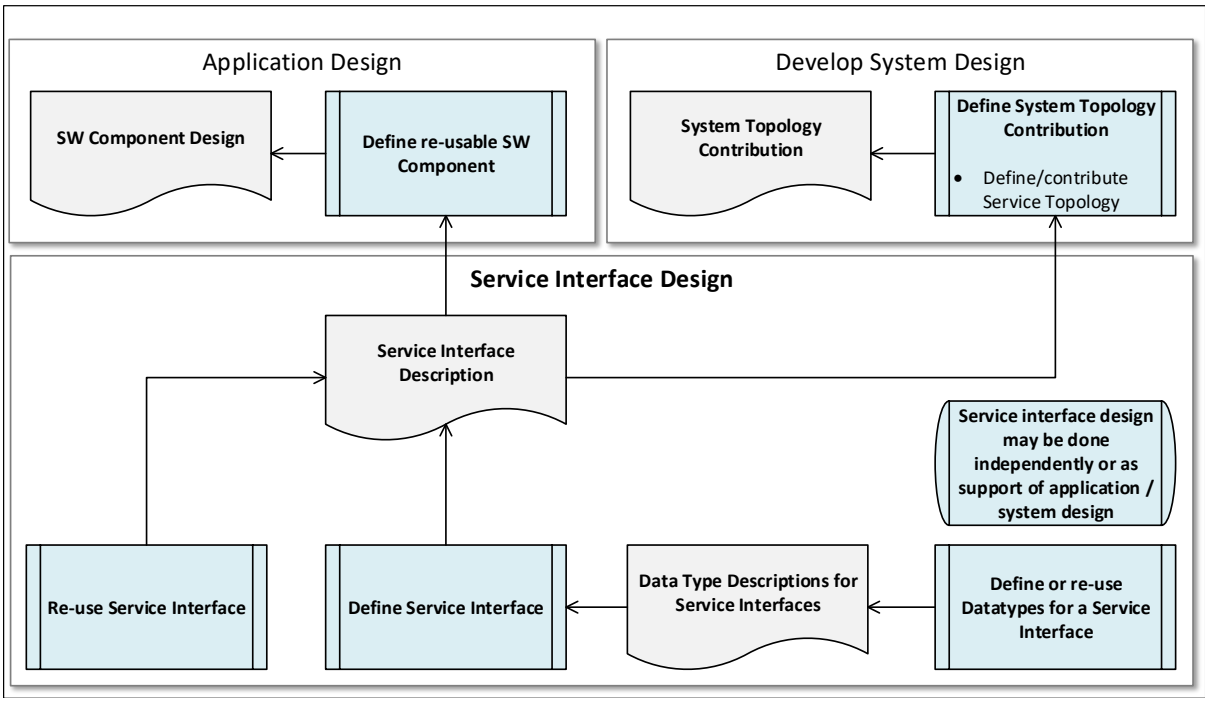

Figure 2.14: How to elaborate Service Interface Design

2.4 Application Design

Figure 2.15 gives an overview of the tasks and work products in scope of Application Design. Please find the detailed definitions of these tasks and work products in Adaptive Methodology Library (see chapter 3.3).

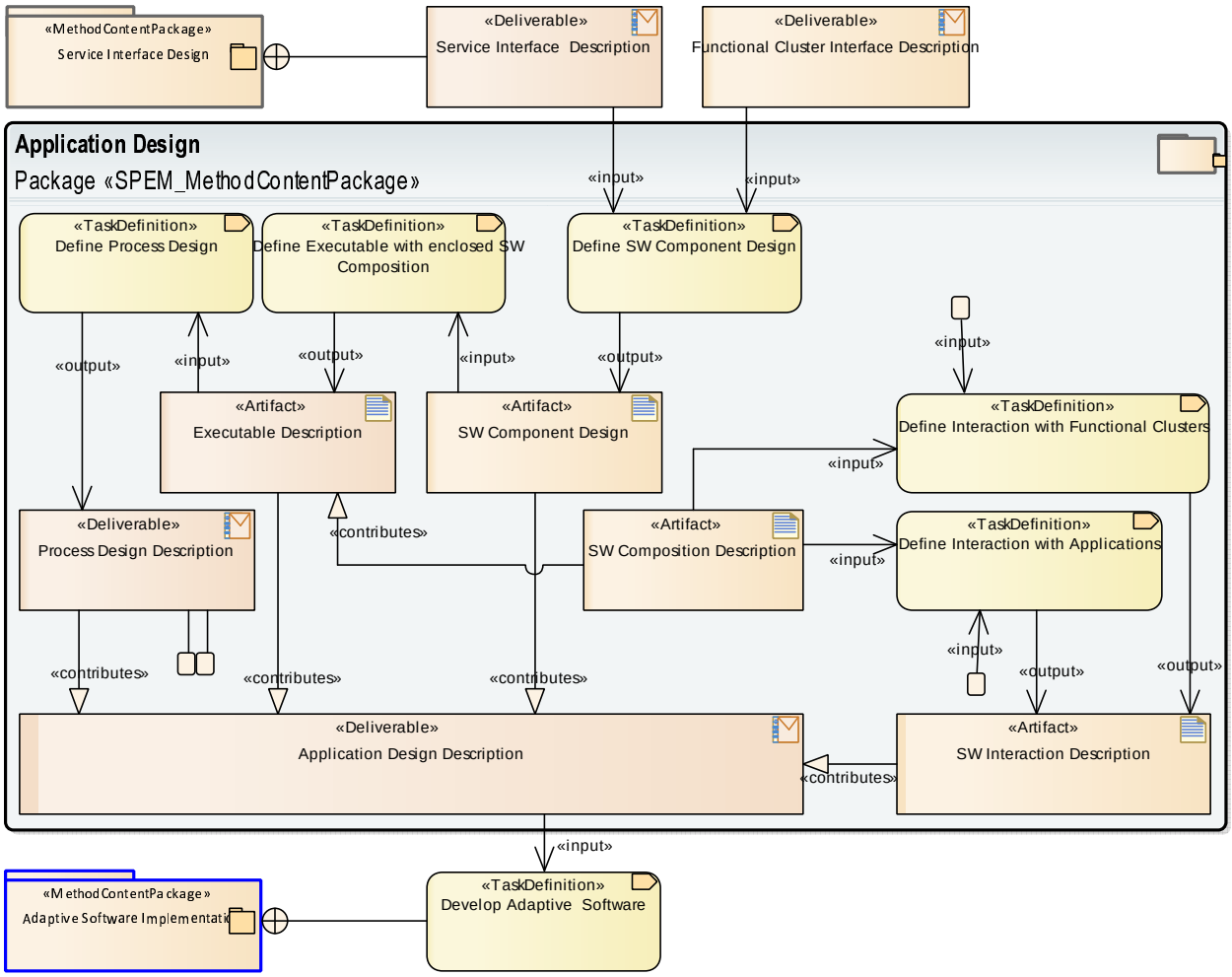

Figure 2.15: Application Design

The resulting Application Design Description consists of

- SW Component Design as outcome of task Define SW Component Design.
  The SW Component Design defines – among other aspects not to be detailed here – interaction endpoints in terms of SWC Ports instantiating specific (application-level or platform-level) PortInterfaces.
- Executable Description including a SW Composition Description as outcome of task Define Executable with enclosed SW Composition. Dependencies are:
  - all addressed SW Component Designs are available.

- Process Design Description as outcome of task Define Process Design with the intention to define a design time proxy for the actual OS process instantiating (i.e. starting) a specific executable.
  Dependencies are:
  - Executable Description is available.
- SW Interaction Description with the at design-time known (and integration relevant) interaction configuration for the SWC Port instances existing in an executable instance. Containing
  - application-level interactions as outcome of task Define Interaction with Applications.
  - platform-level interactions as outcome of task Define Interaction with Functional Clusters.

    For diagnostics this may include/anticipate the Diagnostic Design related SW to Diagnostics Interaction Description contributions (see also Figure 2.20 Diagnostic Design).

  Dependencies are:
  - Process Design Description and related Executable Description with SW Composition Description are available.
  - SW Component Designs include the application-level and platform-level Ports to be configured as interaction endpoints.

The above mentioned tasks have inter-dependencies and may be executed - as per demand - in a suitable order.

Figure 2.16 gives an overview of the AUTOSAR Adaptive Platform software layers. We distinct between application-level and platform-level software. Please be aware that the activities defined in this chapter to describe the Application Design may also apply to platform-level software, especially if platform-level software supports the Port / PortInterface concept.

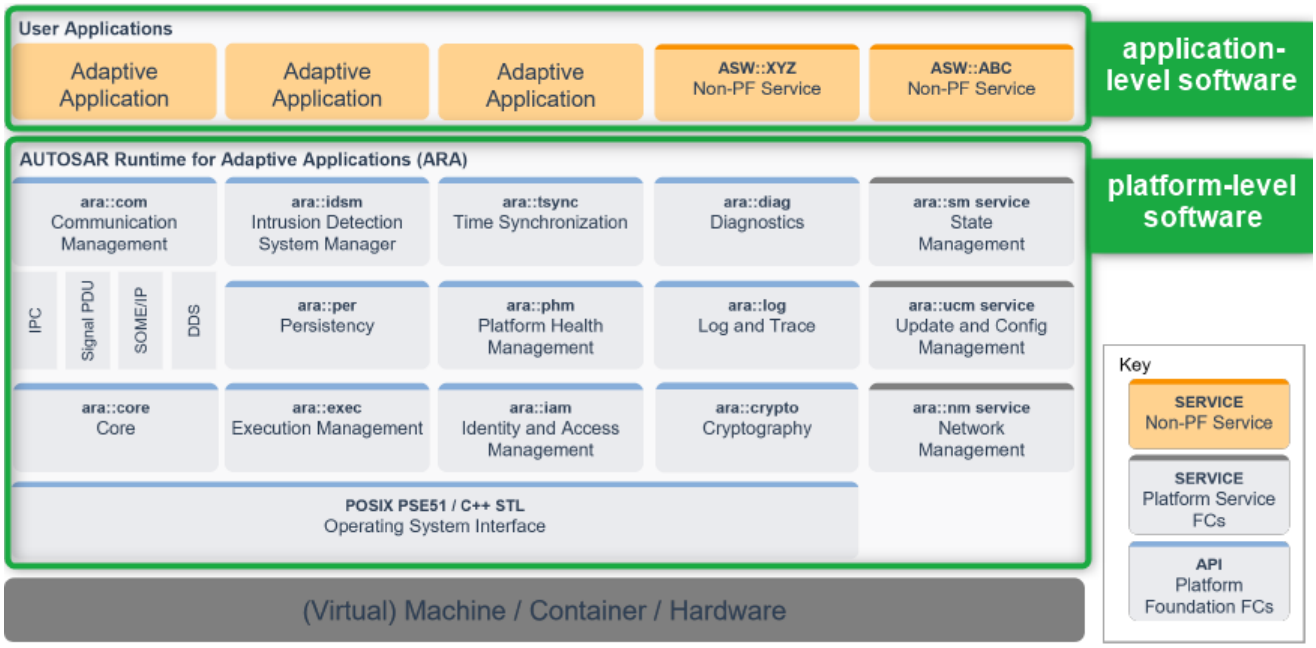

Figure 2.16: AP Software Layers

[TR_AMETH_00010] Application-level Software

Status: DRAFT
Upstream requirements: RS_METH_00202

⌈An `Adaptive Software` of category application-level is a collection of `Executable`s.

Any application-level `Executable` may compose one or more software components.⌋

[TR_AMETH_00011] Design of the software components

Upstream requirements: RS_METH_00202

⌈Based on the service interfaces, the development of adaptive application software starts with the design of the software components.

The software components may have a hierarchical structure.

Any software component design defines its interaction endpoints via required and/or provided `Port`s instantiating service interfaces (for application-level `Port`s) and functional cluster interfaces (for platform-level `Port`s).⌋

[TR_AMETH_00035] Platform-level Software

Status: DRAFT
Upstream requirements: RS_METH_00207, RS_METH_00041

⌈An `Adaptive Software` of category platform-level is a collection of `Executable`s.

A platform-level 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.⌋

2.4.1 Application Design Usage Scenario (top-down)

Figure 2.17 shows the usage scenario of how to elaborate the Application Design, targeting at complete executables and their instantiation as Process Design Description (top-down approach) based on the tailored tasks:

- Define Executable with enclosed SW Composition with
  - Define or re-use SW Components
    This includes the specification of SWC interaction endpoints in terms of Ports instantiating specific (application-level or platform-level) PortInterfaces.
    Inputs are Functional Cluster Interface Descriptions and Service Interface Descriptions, deliverable is a SW Component Description.
  - Define Executable
    This includes the instantiation of SWCs in the dedicated SW composition of an executable.
    Inputs are SW Component Descriptions, deliverable is the Executable Description along with the related SW Composition Description
- Define Executable Run-time Behavior with
  - Define Process Design as design time proxy for the actual OS process instantiating (i.e. starting) a specific executable.
    Input is the Executable Description, deliverable is a Process Design.
- Define SW Interaction for the SWC Port instances existing in an executable instance, with
  - Define Interaction with Application SW
  - Define Interaction with Functional Clusters
  - Inputs come from Application Design, deliverable is a SW Interaction Description that extends the Application Design.

The above mentioned activities have predecessor-dependencies and need to be executed as per order of the bullet points.

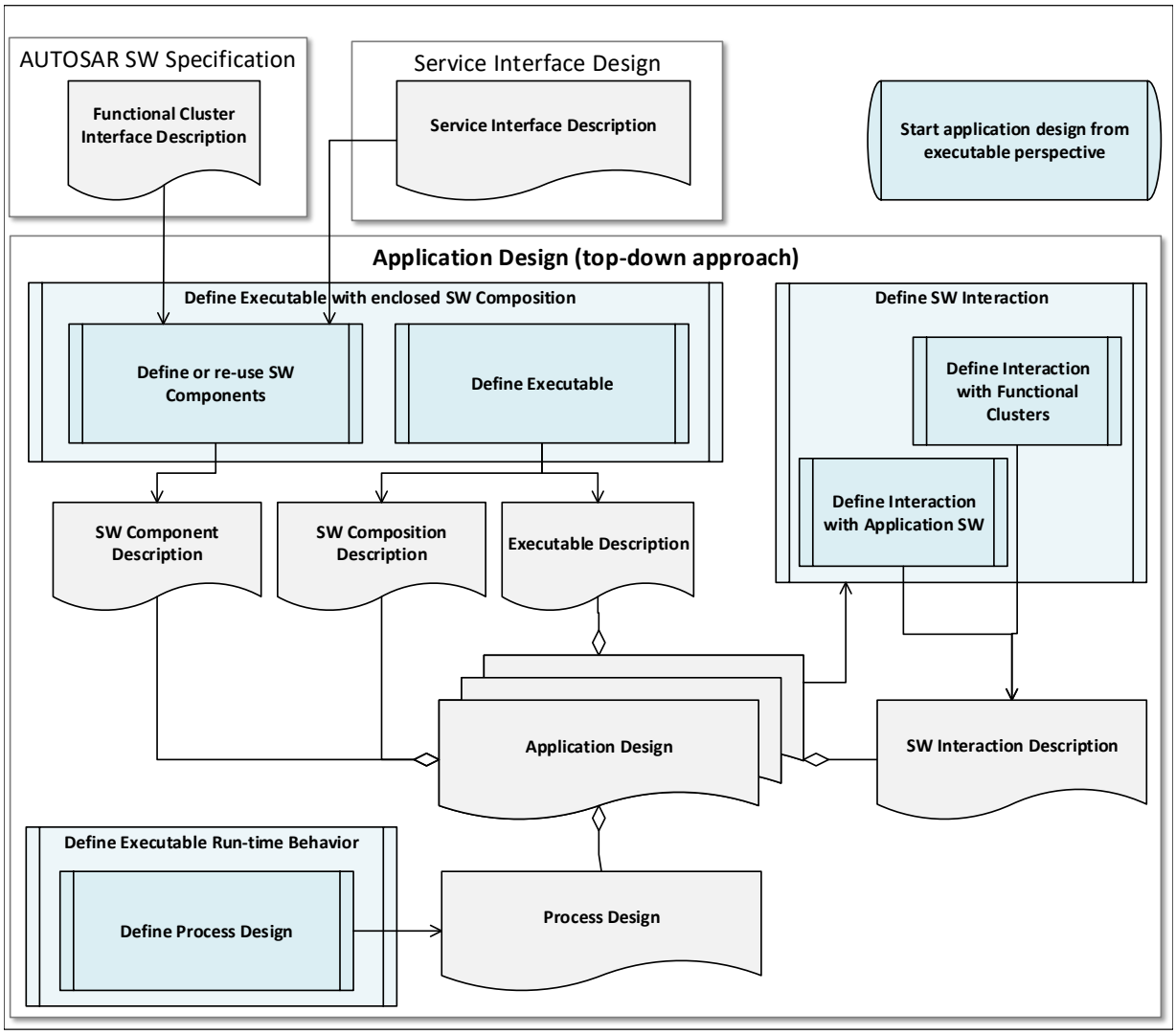

Figure 2.17: How to elaborate Application Design (top-down approach)

2.4.2 Application Design Usage Scenario (bottom-up)

Figure 2.18 shows the usage scenario of how to elaborate a `Contribution to Application Design` based on the tailored task:

- `Define SW Components for re-use` with `Define re-usable SW Component`
  Inputs are `Functional Cluster Interface Description`s and `Service Interface Description`s, deliverable is a `SW Component Description`.

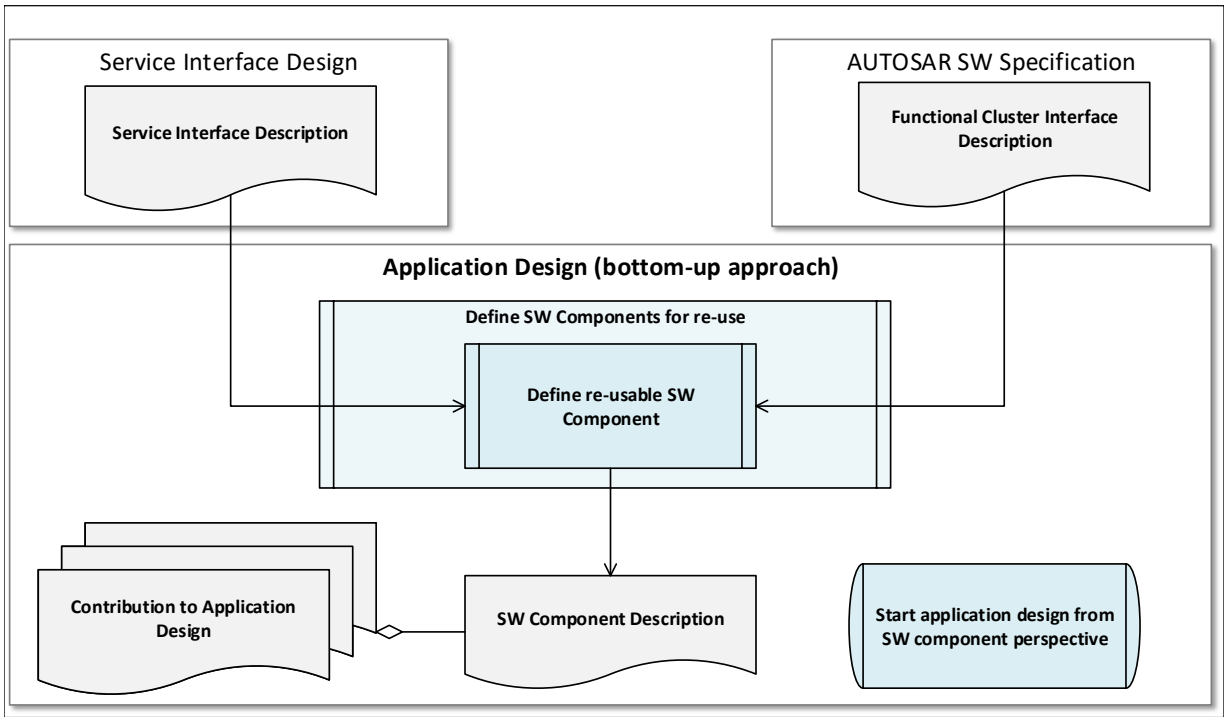

Figure 2.18: How to elaborate Application Design (bottom-up approach)

2.5 Implementation of Adaptive Software

Figure 2.19 gives an overview of the tasks and work products in scope of application-level and platform-level Adaptive Software Implementation. Please find the detailed definitions of these tasks and work products in Adaptive Methodology Library (see chapter 3.4).

[TR_AMETH_00002] Develop Adaptive Software

Status: DRAFT
Upstream requirements: RS_METH_00202, RS_METH_00032

⌈The development of application-level and/or platform-level Adaptive Software can start when the adaptive (service) interfaces have been defined. This software development may include several sub-activities like analysis, design, implementation or test.

The most important outcome of this activity are either source-code or object-code artifacts, depending on whether or not the developer knows the Adaptive Software Build Configuration beforehand.

Finally the adaptive software developer forwards the resulting Adaptive Software Implementation to an integrator.⌋

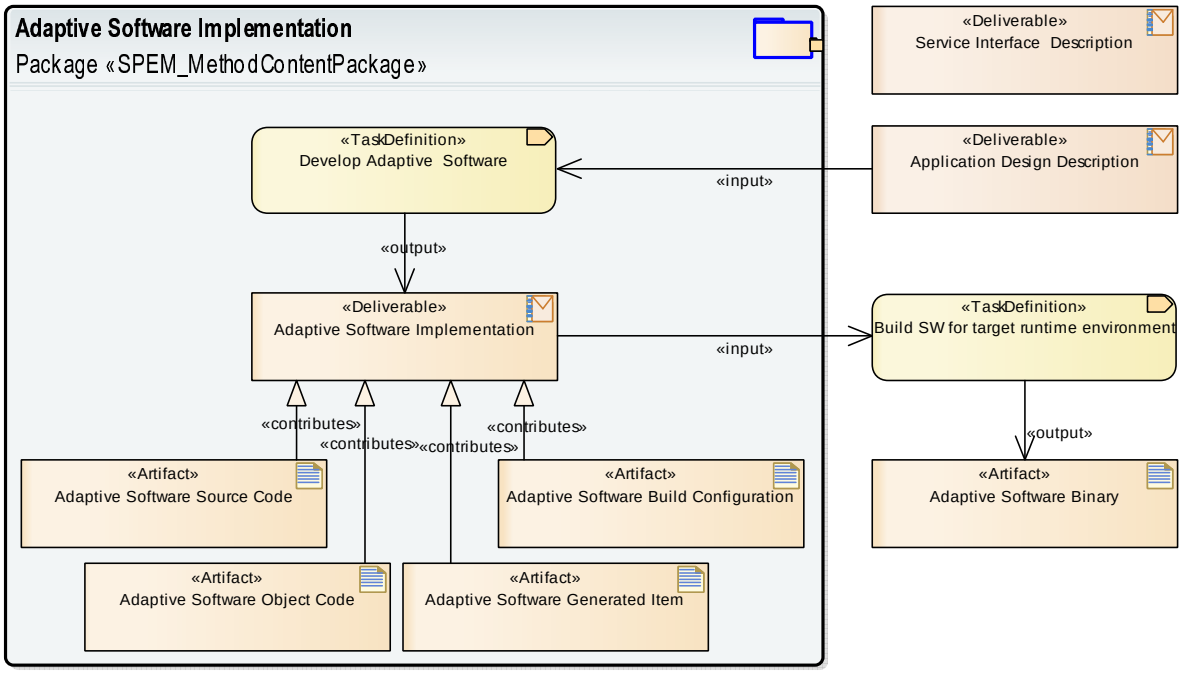

Figure 2.19: Adaptive Software Implementation

Develop Adaptive Software and Adaptive Software Implementation details:

- Input to the actual software development are Adaptive Software Generated Items (see [TR_AMETH_00012]). This is necessary for (but not restricted to) service proxies and skeletons implementing COM Ports and related PortInterfaces.

  See also chapter 2.3 Service Interface Design.

- The actual software development (see [TR_AMETH_00013]) depends on the availability of Adaptive Software Build Configuration:
  - Of course the Adaptive Software Source Code needs to be developed first.
  - If the integrator provides the Adaptive Software Build Configuration (see [TR_AMETH_00014]) Adaptive Software Object Code can be delivered (with no need to expose the Adaptive Software Source Code itself).
  - Else Adaptive Software Source Code is delivered (see [TR_AMETH_00015]).
- Please be aware that application-level and platform-level Adaptive Software need a main function and a execution manifest (see also [TR_AMETH_00020]).

[TR_AMETH_00012] Generation of the header files for service interfaces

Status: DRAFT
Upstream requirements: RS_METH_00202, RS_METH_00066

⌈This step is independent of the design of the software component and its concrete ports: the header files are generated for any number of service interfaces in scope, to be used on demand for the development of software components.

The resulting Adaptive Software Generated Items include service proxies (for requiring COM Ports) and service skeletons (for providing COM Ports). This includes platform independent header files and optionally also platform specific implementations.⌋

[TR_AMETH_00013] Implementation and compilation of software components

Status: DRAFT
Upstream requirements: RS_METH_00202, RS_METH_00015, RS_METH_00066, RS_METH_00042

⌈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 Adaptive Software Build Configuration is known or not known and therefore if source code or object code is delivered by the application developer.

[TR_AMETH_00014] Development with knowledge of the Adaptive Software Build Configuration

Status: DRAFT
Upstream requirements: RS_METH_00202, RS_METH_00077

⌈In this approach, the integrator hands over the Adaptive Software Build 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.⌋

[TR_AMETH_00015] Development without knowledge of the Adaptive Software Build Configuration

Status: DRAFT
Upstream requirements: RS_METH_00202, RS_METH_00077

⌈For this use case, the application developer is not aware of the Adaptive Software Build Configuration and needs to deliver source code to the integrator.

The integrator then takes care for the compilation of the the software component source code.⌋

[TR_AMETH_00020] Development of platform-level Adaptive Software Object Code

Status: DRAFT
Upstream requirements: RS_METH_00207, RS_METH_00041

⌈The platform modules, which consist of an executable, need to be developed. Similar as application-level software, they are later instantiated in terms of an Execution Manifest and then deployed on the machine.

For each executable the corresponding main function needs to be developed as well.⌋

2.6 Diagnostic Design

Figure 2.20 gives an overview of the tasks and work products in scope of Diagnostic Design. Please find the detailed definitions of these tasks and work products in Adaptive Methodology Library (see chapter 3.5).

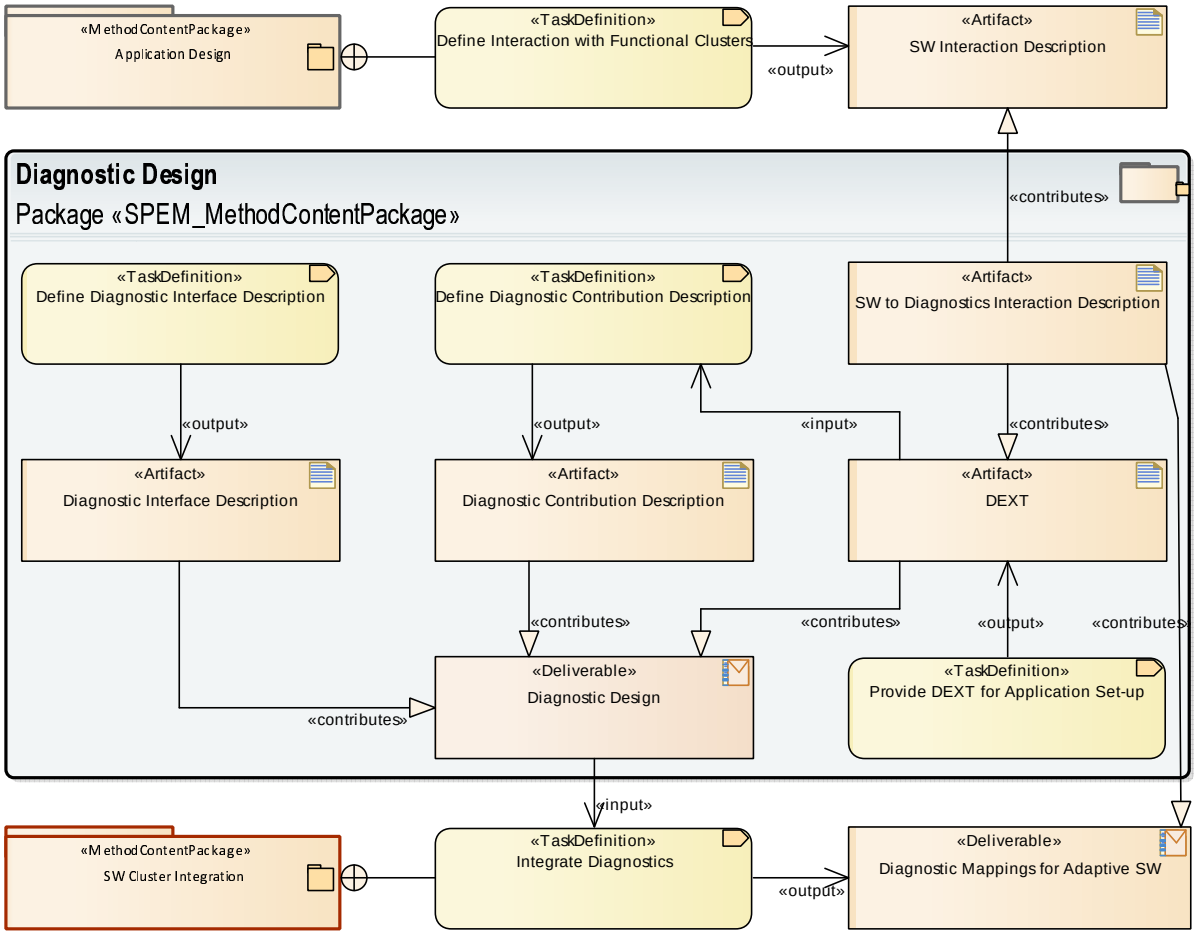

Figure 2.20: Diagnostic Design

This activity associates given diagnostic information (diagnostic data, diagnostic enable conditions, diagnostic events, diagnostic operation cycles) with the software structure (application design information for components, ports etc.) targeting at a particular diagnostic manager for a specific machine.

The configuration of diagnostics on the *AUTOSAR adaptive platform* will typically be done by creating a Diagnostic Extract (`DEXT`) as per [12, Diagnostic Extract Template] that is also used on the *AUTOSAR classic platform*. Therefore, concepts within the Diagnostic Extract are similarly applicable to models on both platforms uniformly:
It can even be safely expected that a given `DEXT` can be divided into parts applying to `CP ECU-Instance`s and parts applying to `AP Machine`s that all belong to the same vehicle.

In order to exemplify the approach, the diagram depicted in Figure 2.21 describes a very simplistic situation where two different `Port`s typed by possibly two different

diagnostic PortInterfaces exposed by an adaptive SWC are queried by the Diagnostic Manager with the purpose of accessing the related (i.e. mapped) diagnostic data identifier (aka DID).
Figure 2.21 Example data exchange for diagnostic purpose shows a diagnostic data to port mapping (from Diagnostic Mappings for Adaptive SW) that formalizes the "connection" between both ends of the communication. [9]

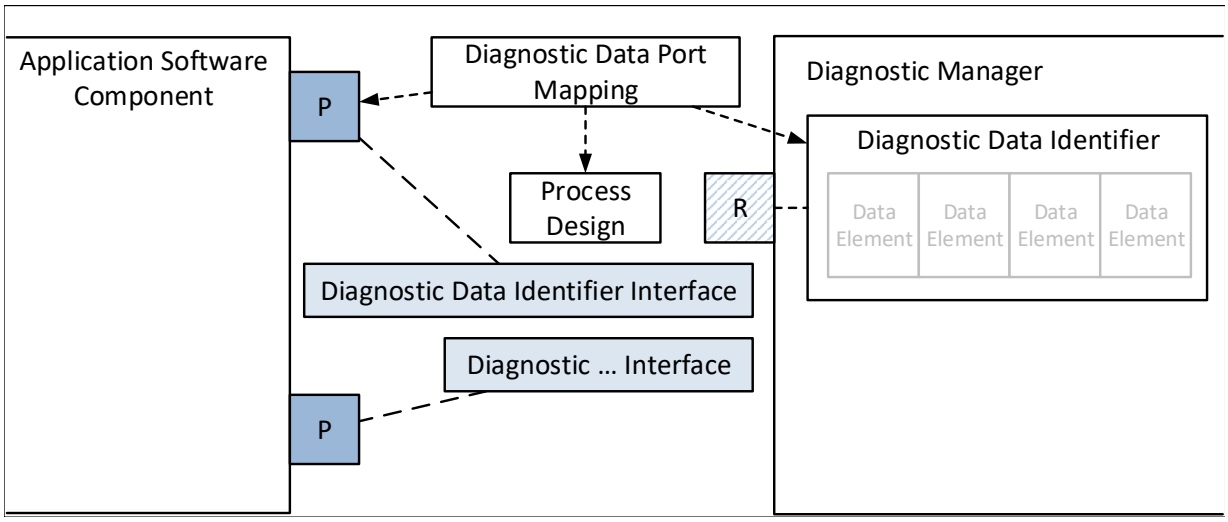

Figure 2.21: Example data exchange for diagnostic purpose

Activities to obtain a Diagnostic Design:

- Define Diagnostic Interface Description results in Diagnostic Interface Descriptions to be used in Application Design during Define SW Component Design to specify the diagnostic Ports of adaptive SWCs.
- Provide DEXT for Application Set-up includes all activities to specify diagnostic resources (like e.g. diagnostic data identifiers) and add them to DEXT.
- An adaptive SWC may be multiply instantiated in an Executable, and an adaptive Executable may be multiply instantiated in Processes.

  In consequence each diagnostic Port instance has an unambiguous Instance Specifier in the context of an Executable Description and needs a dedicated mapping in SW to Diagnostics Interaction Description:

  – This mapping needs to be associated to an OS Process as per Process Design Description which may be done at design time, but shall be finalized at integration time during Integrate Diagnostics (see Figure 2.21 Example data exchange for diagnostic purpose).

[9] While diagnostics for CP uses the different types of CP PortInterfaces like Sender/Receiver or Client/Server interfaces, diagnostics for AP introduces dedicated diagnostic PortInterfaces tailored to the respective diagnostic use case.
Please be aware that the diagnostic mapping completely specifies the port on the diagnostic manager side which may be virtualized or generated along with the diagnostic manager implementation.

  - DEXT includes SW to Diagnostics Interaction Descriptions that may have been specified during Application Design in Define Interaction with Functional Clusters, but shall be finalized during Software Cluster Integration by completing SW to Diagnostics Interaction Description to obtain Diagnostic Mappings for Adaptive SW (see Figure 2.30 Software Cluster Integration (part 2)).

- Provide DEXT for Application Set-up results in the applicable DEXT with contents collected at Application Design and/or Diagnostic Design and/or Software Cluster Design.

  Please be aware that a DEXT may hold the diagnostic configuration for multiple Executables, Diagnostic Managers and SoftwareClusters. Provide DEXT for Application Set-up ensures that the diagnostics demand of a specific Application Design Description is completely covered.

- Define Diagnostic Contribution Description results in Diagnostic Contribution Description specifying what content of a DEXT shall be used in the Diagnostic Manager tailored for the Executables in a specific SoftwareCluster.

A diagnostic mapping is a formal model for the relation between the adaptive diagnostic manager (module) and specific diagnostics-related endpoints in the application software (see also Figure 2.21 Example data exchange for diagnostic purpose) and may be elaborated

1. during Application Design
   in the activity Define Interaction with Diagnostics (as part of Define Interaction with Functional Clusters) resulting in an early form of Diagnostic Mappings for SW to Diagnostics Interaction Description.

   The expectations from [TR_AMETH_00212] and [TR_AMETH_00213] may be fulfilled at that point of time (depending on the availability of the related Diagnostic Resources in DEXT).

2. during Diagnostic Design
   in the activity Define Diagnostic Mappings (as part of Provide DEXT for Application Set-up) resulting in consolidated Diagnostic Mappings

   The expectations from [TR_AMETH_00212] shall be fulfilled and
   the expectations from [TR_AMETH_00213] may be fulfilled at that point of time.

3. during Software Cluster Integration
   in the activity Finalize Diagnostic Mappings (as part of Integrate Diagnostics) resulting in finalized Diagnostic Mappings based on SW to Diagnostics Interaction Description

   The expectations from [TR_AMETH_00212] and [TR_AMETH_00213] shall be fulfilled at that point of time.

[TR_AMETH_00212] Design a diagnostic mapping

Status: DRAFT
Upstream requirements: RS_METH_00207, RS_METH_00201, RS_METH_00016

⌈This activity covers most necessary tasks to perform the diagnostic mapping (only the association of the corresponding Process Designs will or may be done later by an integrator).

These tasks are in detail:

- Map Diagnostic Clear Condition to Port(s)
- Map Diagnostic Data to Port(s)
- Map Diagnostic Enable Condition to Port(s)
- Map Diagnostic Event to Port(s)
- Map Diagnostic Generic Service to Port(s)
- Map Diagnostic Indicator to Port(s)
- Map Diagnostic Memory Destination to Port(s)
- Map Diagnostic Operation Cycle to Port(s)
- Map Diagnostic Security Level to Port(s)
- Map Diagnostic SW Service to Port(s)

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

- The DEXT as Diagnostic (Machine) Extract that contains the required diagnostic resources.
- Executable Description with the SW Composition Descriptions specifying the included software components and their ports.

This step results in partly filled in artifact Diagnostic Mappings for Adaptive SW which is represented at design time by SW to Diagnostics Interaction Description.⌋

2.7 Machine Design

Figure 2.22 gives an overview of the tasks and work products in scope of Machine Design. Please find the detailed definitions of these tasks and work products in Adaptive Methodology Library (see chapter 3.6).

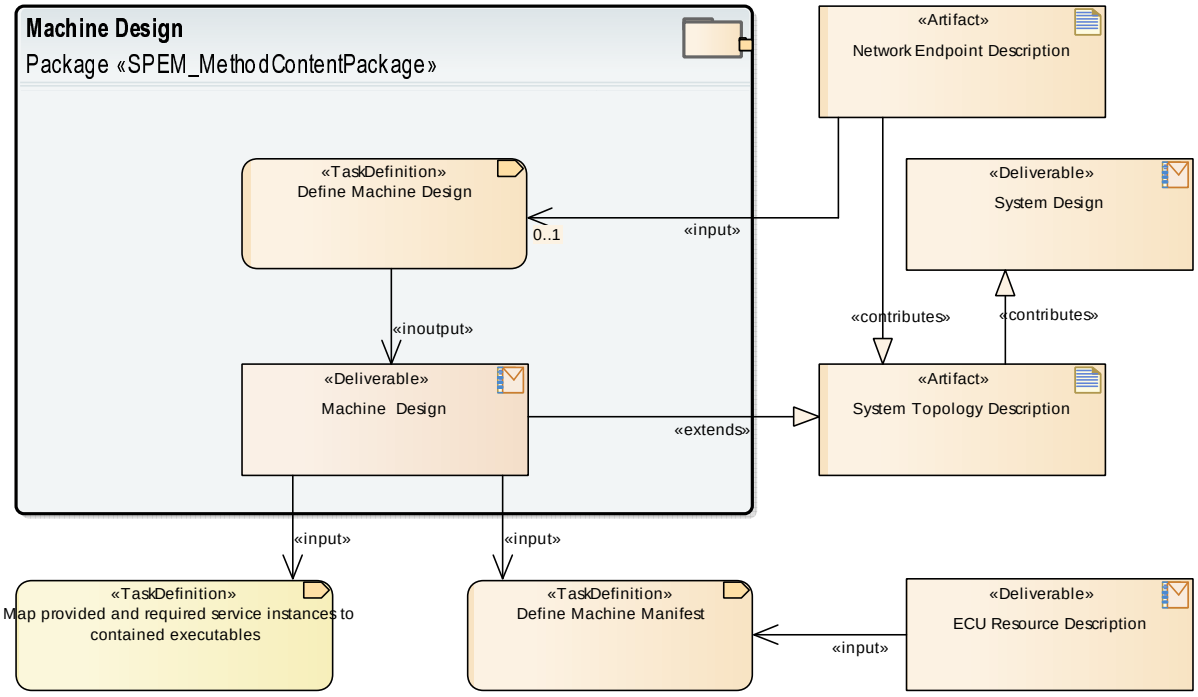

Figure 2.22: Machine Design

The resulting Machine Design as outcome of Define Machine Design uses or anticipates network endpoint definitions from System Design

[TR_AMETH_00003] Configuration of the Machine

Status: DRAFT
Upstream requirements: RS_METH_00204, RS_METH_00203

⌈A Machine has a configuration that is tailored as per ECU Resource Description of a concrete ECU-HW. Due to this the activities Define Machine Design and Define Machine Manifest are in scope of Tier 1 company integrators.

The provisioning of Network Endpoint Description - which is in the scope of a communication designer of an OEM - is a precondition for this. This happens in an early design phase, the resulting Machine Design represents requirements regarding the network communication for a prospective Machine.

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

- The first step is the configuration of the communication structure of a prospective Machine and will be

  - either performed by a communication designer of an OEM as part of the (system) design phase,

  - or performed by an integrator of a Tier 1 company based on the Network Endpoint Description shared by an OEM.

  Result is a Machine Design.

- The second step covers activities and tasks for the configuration of a Machine to be deployed on a real ECU-HW and will be performed by an integrator of a Tier 1 company.
  The resulting configuration is then part of the Machine Manifest.

⌋

[10]

[TR_AMETH_00021] Define and configure the network communication for a Machine

Status: DRAFT
Upstream requirements: RS_METH_00204, RS_METH_00203

⌈This activity will cover the definition and configuration of the network communication for a prospective Machine resulting in a Machine Design with:

- Configuration of the network connections with the network endpoint IP (version 4 or version 6) addresses.
- Configuration of the service discovery message exchange with the multicast IP addresses and UDP ports.

⌋

2.7.1 Machine Design Usage Scenario

- either in the context of an existing System Design (top-down approach) based on the Network Endpoint Descriptions defined in System Topology Description as per tailored task:
  - Define Machine Design as per System Topology
- or locally (bottom-up approach) – eventually along with a system topology contribution – based on the tailored task:
  - Define Machine Design Locally

[10] see also chapter 2.8 Machine Manifest

resulting in tailored work products

- Network Connector Description and Service Discovery Description as Machine Design
- optionally also System Topology Contribution in System Design

2.8 Machine Manifest

Figure 2.23 gives an overview of the tasks and work products in scope of `Machine Manifest`. Please find the detailed definitions of these tasks and work products in Adaptive Methodology Library (see chapter 3.7).

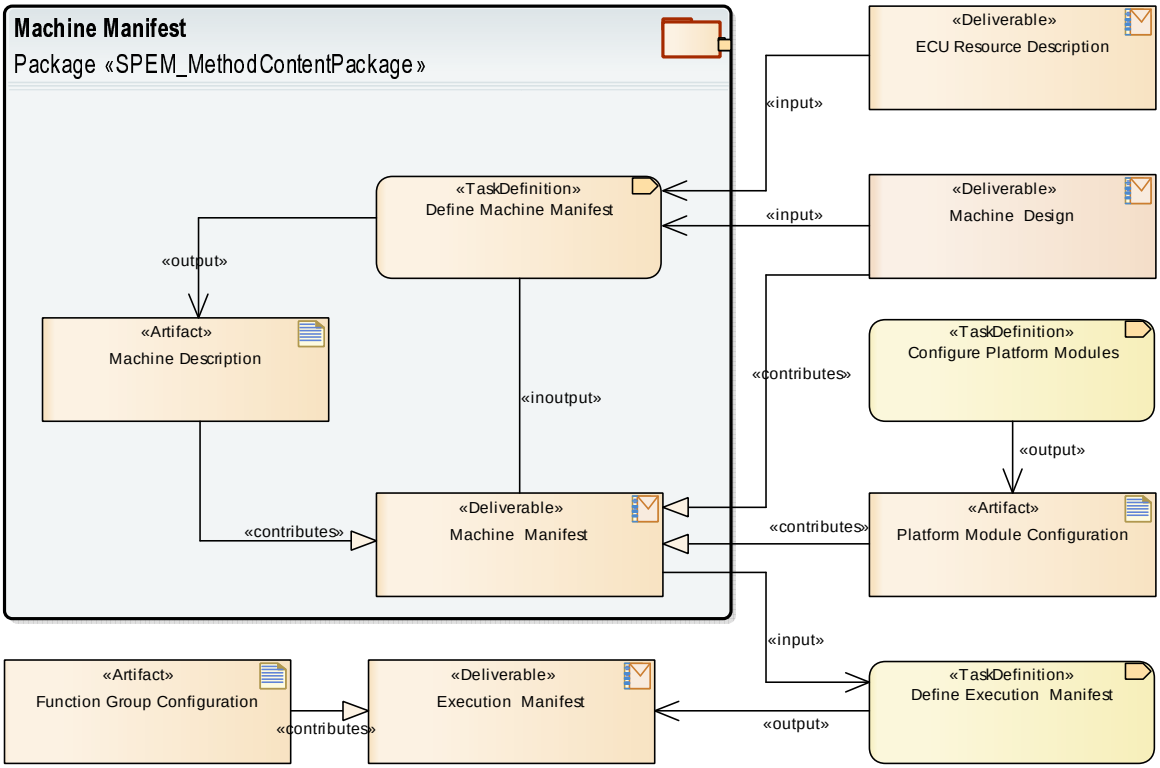

Figure 2.23: Machine Manifest

A major input for `Define Machine Manifest` is the configuration of the communication structure available in `Machine Design` which was prepared during Machine Design (see chapter 2.7), there [TR_AMETH_00003] defines this work split.

The resulting `Machine Manifest` includes

- `Machine Description` describing the available `ECU-HW` resources, see also [TR_AMETH_00019] and [TR_AMETH_00217].
- `Machine Design` as outcome of `Define Machine Design`, see chapter 2.7 Machine Design.
- `Platform Module Configuration`s as outcome of `Configure Platform Modules`, see also [TR_AMETH_00023], [TR_AMETH_00214], [TR_AMETH_00215] and chapter 2.10.3 Create Platform Module Configuration.
- The configuration of what processes run on a specific `Machine` is part of `Execution Manifest`, see chapter 2.10.2 Create Execution Manifest.

[TR_AMETH_00019] Description of the ECU-HW resources available for the Adaptive Platform

Status: DRAFT
Upstream requirements: RS_METH_00207, RS_METH_00041

⌈As a first preparatory step, the available hardware elements for a particular Adaptive Platform instance need to be specified.

Major configuration entity for this is the ECU Resource Description describing the available hardware elements like processing units, memories, sensors, actuators or pins. The ECU Resource Description should come from the ECU-HW provider.⌋

[TR_AMETH_00034] Select the Operating System for Adaptive Platform

Status: DRAFT
Upstream requirements: RS_METH_00207, RS_METH_00041

⌈An operating system (OS) needs to be selected and provided for a particular Adaptive Platform. It might be necessary to tailor the OS for the specific ECU-HW and Machine.

The OS for the Adaptive Platform is a platform module not having an Execution Manifest. Note, that its development work flow will differ from the work flow of platform-level software.⌋

[TR_AMETH_00217] Definition of resources

Status: DRAFT
Upstream requirements: RS_METH_00204, RS_METH_00203

⌈The configuration of a Machine may include the specification of resources as Machine Description

Based on the ECU Resource Description of the target ECU-HW, available hardware resources for a specific Machine can be described and added to the Machine Manifest.⌋

[TR_AMETH_00023] Configuration of the operating system

Status: DRAFT
Upstream requirements: RS_METH_00204, RS_METH_00203

⌈The operating system Platform Module Configuration specifies a specific instantiation of the operating system with resource groups, the supported timer granularity etc..⌋

[TR_AMETH_00214] Configuration of Platform Services

Status: DRAFT
Upstream requirements: RS_METH_00204, RS_METH_00203

⌈The configuration of a Machine includes the machine-specific Platform Module Configurations of Adaptive Platform Services (like Network Management, DoIP).⌋

[TR_AMETH_00215] Configuration of Platform Foundation Modules

Status: DRAFT
Upstream requirements: RS_METH_00204, RS_METH_00203

⌈Beside the configuration of the Operating System, the configuration of a `Machine` also includes the machine-specific `Platform Module Configuration`s of the Adaptive Platform Foundation Modules (like Log & Trace).⌋

2.9 Software Cluster Design

Figure 2.24 and Figure 2.26 give an overview of the tasks and work products in scope of SW Cluster Design Description. Please find the detailed definitions of these tasks and work products in Adaptive Methodology Library (see chapter 3.9).

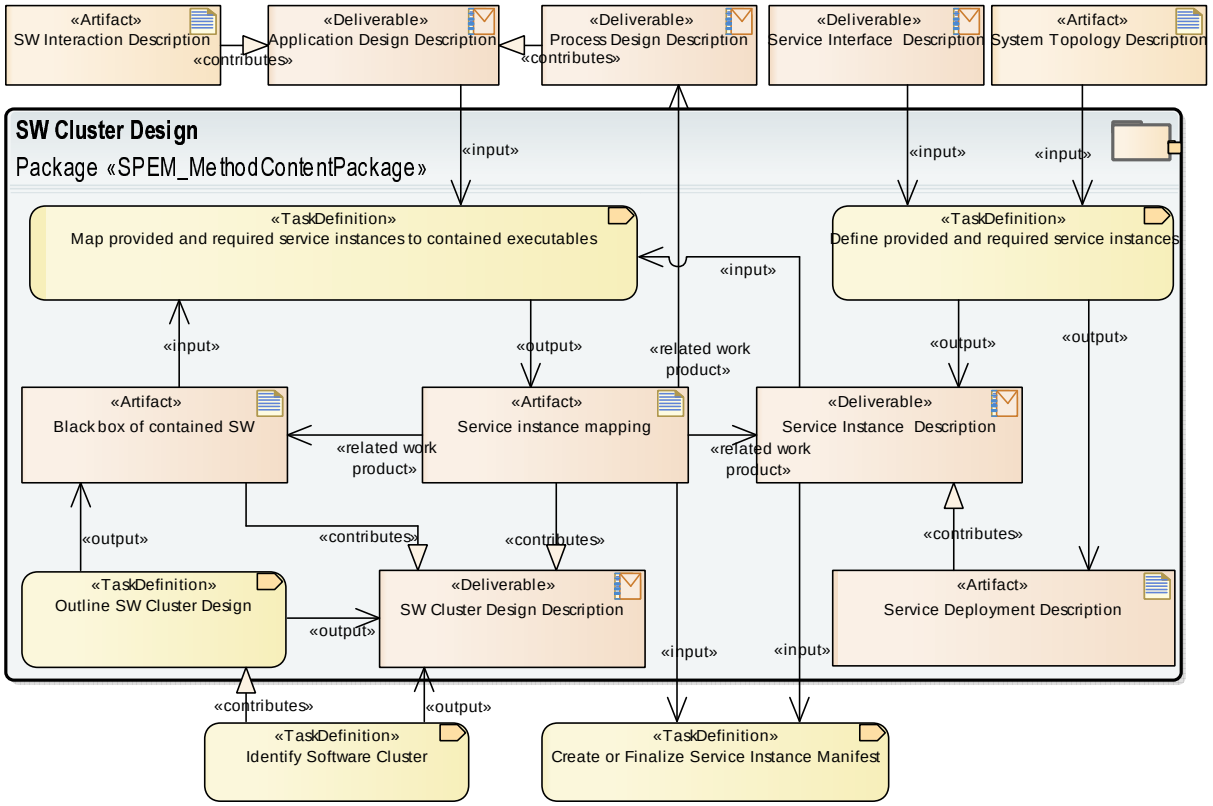

Figure 2.24: Software Cluster Design (Part 1)

The nature of the *AUTOSAR adaptive platform* as a platform for deploying software units in the field requires upfront design-support for such software.

As an example, such a software unit could be a self-contained driving function.

This requires support for a design process for application-level software communicating with other application level software inside or across driving functions.

We use SW Cluster Design Description for the design of software that might represent such a driving function.

Please note that SW Cluster Design Description supports an arbitrary granularity of software and may therefore also cover multiple driving functions.

Please note also that the conversion of a SW Cluster Design Description to a Software Cluster Description is not formalized by AUTOSAR. This step can be done by a tool at the discretion of the integrator. In some cases this conversion may be done relatively early in the development project, while other projects may keep the SW Cluster Design Description around for a longer period in time.

The SW Cluster Design Description covers various design aspects of a deployable software entity that may be specified in top-down or bottom-up work flows:

- In a top-down approach, the SW Cluster Design Description considers a part of the Vehicle Software Architecture, which was (de)composed into a building block of a sub-system software architecture during System Design in the activities Define System Topology and Identify Software Cluster as shown in Figure 2.4 Extract sub-systems from Vehicle Software Architecture:
  - Define System Topology considers the decomposition into subsystems along with the definition of the service topology.
  - Identify Software Cluster considers the SW Cluster Design Description content available at system design time.

  Based on this information, the elaboration of SW Cluster Design Description starts and identifies the included application- and platform-level software entities to be detailed in Application Design.
- In a bottom-up approach, some independently developed Application Design Descriptions (see Figure 2.15 Application Design in chapter 2.4 Application Design) may be input for the set-up of a SW Cluster Design Description.
- In both cases, Define provided and required service instances for the functionality in scope (along with the related Service Interface Descriptions) is a good starting point for the development.

2.9.1 Outline SW Cluster Design

The SW Cluster Design Description represents the formalized response to requirements, which have initially been formulated by an OEM, and may be enriched as the development of the software progresses.

- Purpose of any SW Cluster Design Description is the design of an installation change on a target Machine: an installable entity shall be added, updated or removed. This includes (but is not restricted to):
  - front-load the definitions of SW Clusters
  - support consistency
  - define building blocks for the future deployment
- Granularity and partitioning of installable entities:
  - In simple cases, a SW Cluster Design Description covers exactly one (consistent) installable entity.

  - in more complex cases, a SW Cluster Design Description has a nested structure reflecting a set of installable entities to be installed consistently. Uses cases are:

    – A distributed development scenario (nesting as per involved parties).
    – A piece-wise update logistics (nesting defines building blocks).

- The relation to High Level Architecture may be orthogonal:
  - e.g. a driving function may have multiple SW cluster designs
  - The concept of SW Cluster Design Description applies to platform level and application level software in any combination.

Please note that SW Cluster Design Description is not intended to be uploaded to the target platform. It is just an early form of the final Software Cluster Description that does get uploaded.

Outline SW Cluster Design covers the basic set-up of a SW Cluster Design Description:

- Specify the SW Cluster Design Description container itself.
- Specify the Black box of contained SW, identifying the communication endpoints in SW Cluster Design Description (see also Figure 2.25 Service instance mapping), likely before any Application Design activities are started.
- Dependencies between SW Cluster Design Descriptions allow to describe in an early design phase what needs to be installed together.

  Since SW Cluster Design Description does not include a version-tag, these dependencies just consider software clusters in general.
- optionally Associate Diagnostic Address and Contribution
- optionally Associate content elements currently known.

  This may include Machine Design for the intended target machine as context for the future uploadable software package in an early phases of a development project.

The detailed content is added at a later point of time, in any order, as per the individual design work flow.

2.9.2 Define (and map) provided and required service instances

Usually, Define provided and required service instances is the starting point for the development of software clusters.

Based on the necessary input of Service Interface Descriptions, this results in Service Instance Descriptions (see [TR_AMETH_00005]) based on Service Deployment Descriptions (see [TR_AMETH_00027]).

Black box of contained SW outlines the communication endpoints as delegation ports (representing the exposed ports of the enclosed software of SW Cluster Design Description) at an early design stage – typically before the actual Application Design Description is available.

In order to associate a Service Instance Description to a SW Cluster Design Description, we need to make it effective at a specific communication endpoint of the enclosed software.

- To achieve this, Black box of contained SW specifies an individual delegation port per Service Instance Description addressed in the SW Cluster Design Description.
- Only when Application Design Description and Process Design Descriptions are available, Map provided and required service instances to contained executables allows to finalize the Service instance mapping.
- Figure 2.25 Service instance mapping gives an overview on this.
- Service Instance Descriptions and Service instance mapping will be used to create the Service Instance Manifest during Software Cluster Integration.

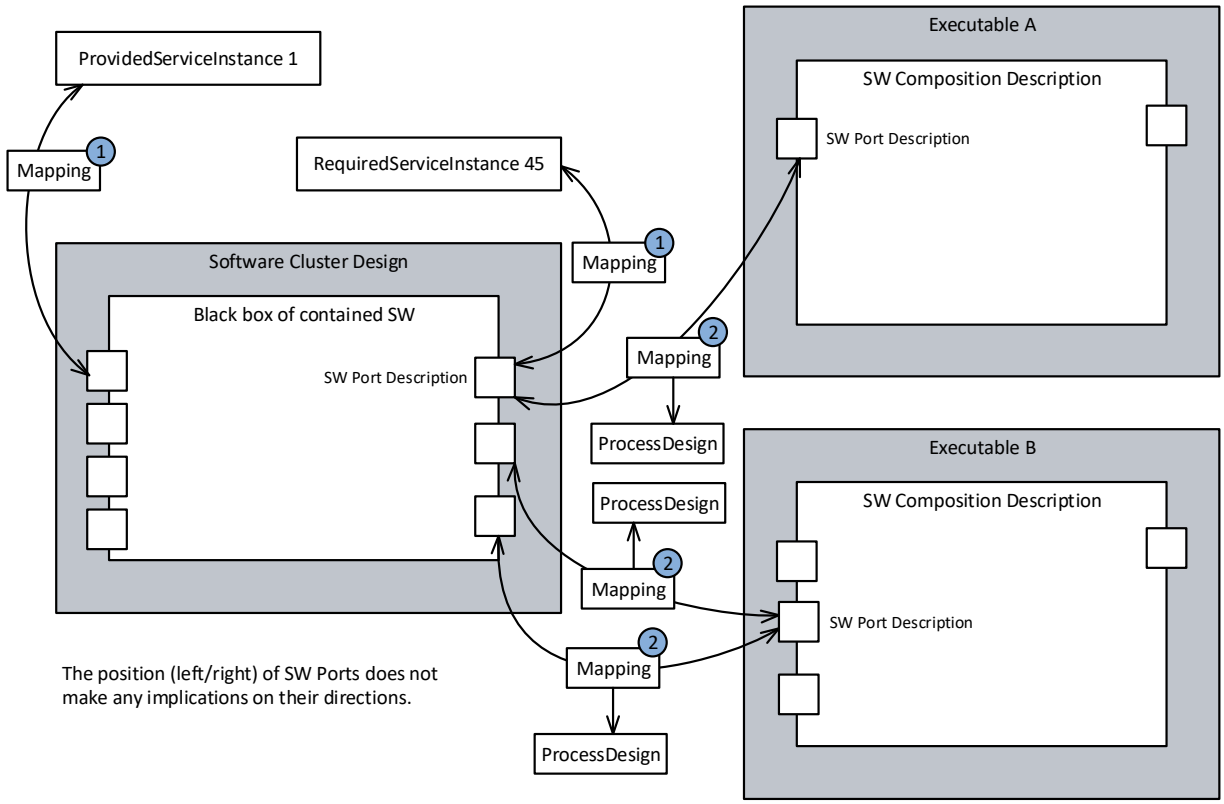

Figure 2.25: Service instance mapping

[TR_AMETH_00027] Configuration of Service Interface Deployment

Status: DRAFT
Upstream requirements: RS_METH_00206, RS_METH_00203

⌈The software cluster responsible specifies in Service Deployment Description how the service interfaces shall be deployed. This includes the (vehicle-wide consolidated) properties describing the individual transport layer binding of the service interface.

E.g. for SOME/IP deployment, an ID for each service interface is defined. This ID needs to be unambiguous in the system context. Additionally method-IDs, event-IDs as well as event groups are defined unambiguously in the scope of the individual SOME/IP service interface deployment.⌋

[TR_AMETH_00005] Configuration of the service instances

Status: DRAFT
Upstream requirements: RS_METH_00206, RS_METH_00203

⌈The software cluster responsible specifies in Service Instance Description how a service interface shall be instantiated as provided or required services. This extends the (vehicle-wide consolidated) properties describing the individual transport layer binding of the service interface from Service Deployment Description.

E.g. for SOME/IP deployment, an ID for each service instance is defined. This service **instance** ID needs to be unambiguous in SOME/IP service **interface** ID context (which in turn is unambiguous in the overall vehicle context).⌋

2.9.3 Associate content elements

One of the most prominent contents of a SW Cluster Design Description is the reference to the executable software via Process Design Descriptions.

- The Process Design Description is a design-level representation of a Process, i.e. an instance of the corresponding executable (software image) on the target Machine.
- Please be aware that Application Design Description with the Executable Description and Process Design Description are deliverables of Application Design.

An important aspect of a SW Cluster Design Description is the question what diagnostic extract shall be applied.

- In an early stage of the development process, it is intentionally made possible to reference multiple Diagnostic Designs in order to support the decentralized (e.g. partly done by OEM and partly done by supplier) configuration of the diagnostics stack.

- SW Cluster Design Description contains one physical diagnostic address and any number of functional diagnostic addresses. This information typically comes from OEM as part of the contracting.
- Associate Diagnostic Address and Contribution (considering the applicable DEXT content as per Diagnostic Contribution Description from Diagnostic Design, see also chapter 2.6 Diagnostic Design) may be done during or after Outline SW Cluster Design.

Besides that Associate content elements takes care that all required uploadable elements in scope of SW Cluster Design Description (like Machine Design, Process Design Description etc.) are registered in Associated uploadable elements.

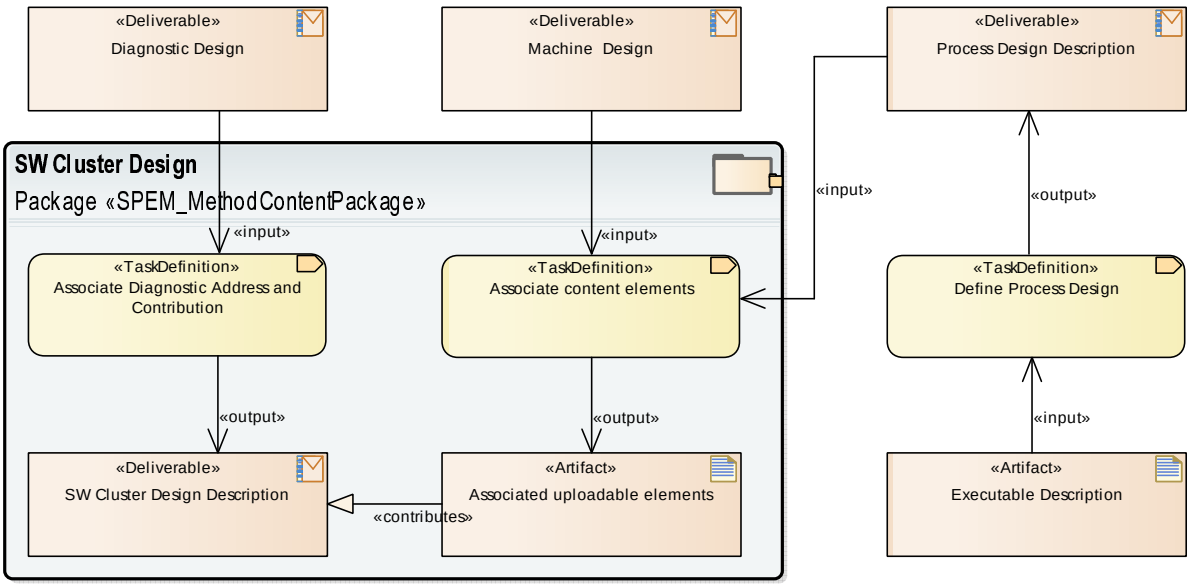

Figure 2.26: Software Cluster Design (Part 2)

2.9.4 Software Cluster Design Usage Scenario (top-down)

Figure 2.27 shows the tailored task and work product definitions of SW Cluster Design in a top-down usage scenario starting with the information available with System Design.

Elements are:

- Define Service Instances considering the service topology of provided and required services known at system design time in System Topology Description.

  This is a major design decision:

  - what service instances identified by an OEM at system-design time shall be implemented in the currently designed software cluster.
  - what additional supporting service instances are needed for the software cluster implementation.

- Define SW Cluster with Communication Endpoints considering (besides the sub-system topology known at system design time in System Topology Description) the Software Cluster Descriptions identified at system design time.

  This decouples the software cluster design from the actual software design:

  - Delegation ports listed in a software cluster black box (see also Figure 2.25 Service instance mapping) identify the communication endpoints.
  - One or more applications (taking part in the currently designed software cluster) shall implement these communication endpoints (and only these communication endpoints).

- Map Service Instance links the software cluster design to the related application designs (see also Figure 2.25).

- Associate Content along with Associate Diagnostic Address and Contribution completes the SW Cluster Design

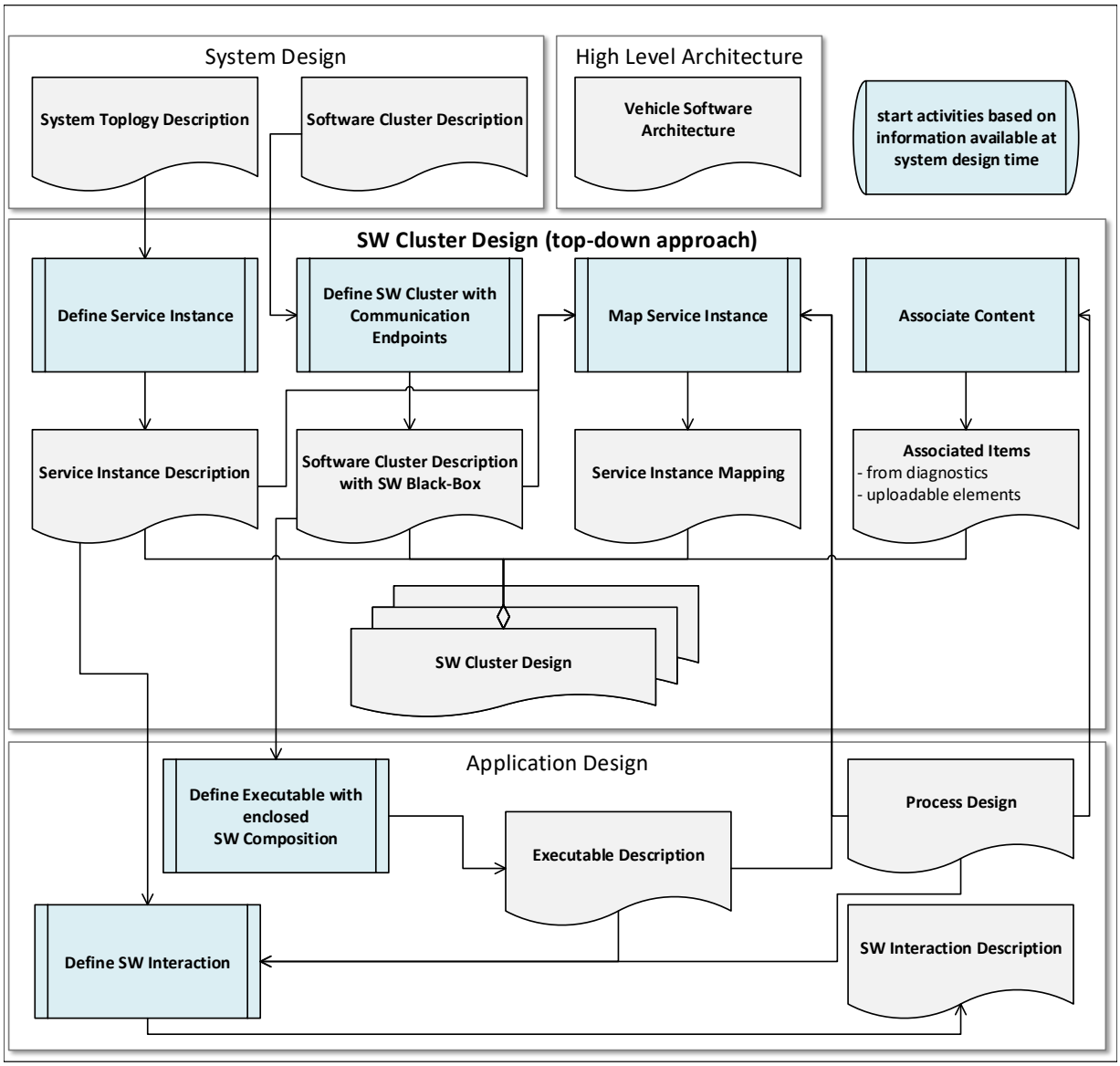

Figure 2.27: How to elaborate Software Cluster Design (top-down approach)

The top-down approach results in work products tailored for software cluster scope:

- Service Instance Descriptions (along with the Service Deployment Descriptions in scope) to be deployed as part of the currently designed SW Cluster.
- Software Cluster Description with SW Black-Box
- Service Instance Mapping combining Service Instance Description and Software Cluster Description with SW Black-Box with concrete port instances from Application Design.
- Associated Items considering Associated uploadable elements

Please be aware of the dependencies between the work flows for Application Design and Software Cluster Design:

- Application Design can start only when Software Cluster Description with SW Black-Box specifies the communication endpoints for Define Executable with enclosed SW Composition.
- Define SW Interaction in Application Design preferably uses Service Instance Description.
- Map Service Instance requires the detailed Executable Descriptions and Process Designs from Application Design.

2.9.5 Software Cluster Design Usage Scenario (bottom-up)

Figure 2.28 shows the tailored task and work product definitions of SW Cluster Design Design in a bottom-up usage scenario starting with the information available with Application Design.

Elements are:

- Define Service Instance considering the provided and required service instances identified during Application Design and used there in Define SW Interaction.
- Define SW Cluster with Communication Endpoints considering the communication endpoints identified during Application Design in Define Executable with enclosed SW Composition and Define Executable Run-time Behavior and used there in Define SW Interaction.

  The information for Map Service Instance is available at that point of time.
- Associate Content for Associate content elements and Associate Diagnostic Address and Contribution as per availability during Application Design.

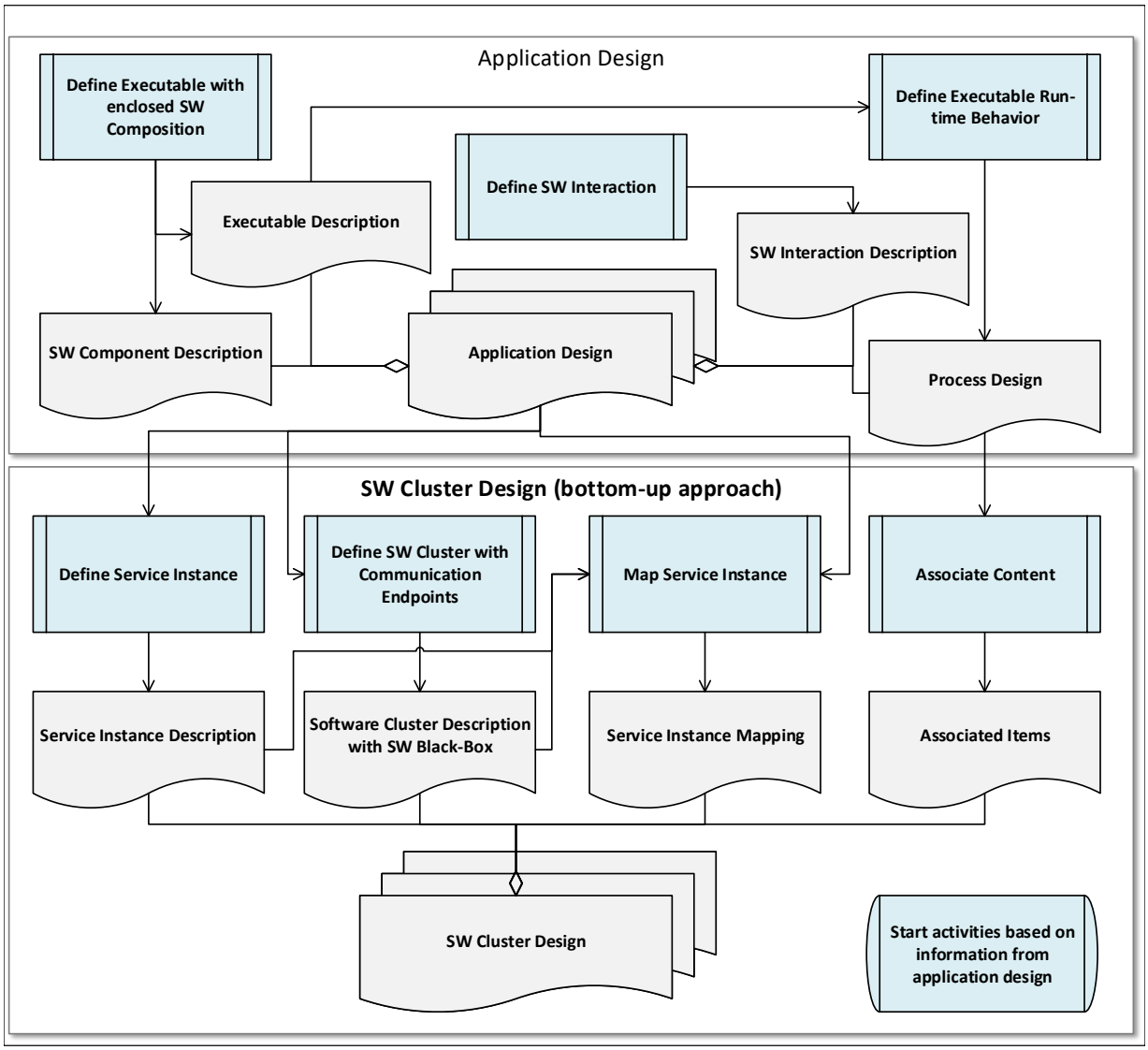

Figure 2.28: How to elaborate Software Cluster Design (bottom-up approach)

The bottom-up approach results in in work products tailored for software cluster scope:

- `Service Instance Description` and `Service Deployment Description`
- `Software Cluster Description with SW Black-Box`
- `Service instance mapping`
- `Associated Items` for `Associated uploadable elements`

2.10 Software Cluster Integration

Figure 2.29 and 2.30 give an overview of the tasks and work products in scope of SoftwareCluster Integration. Please find the detailed definitions of these tasks and work products in Adaptive Methodology Library (see chapter 3.10).

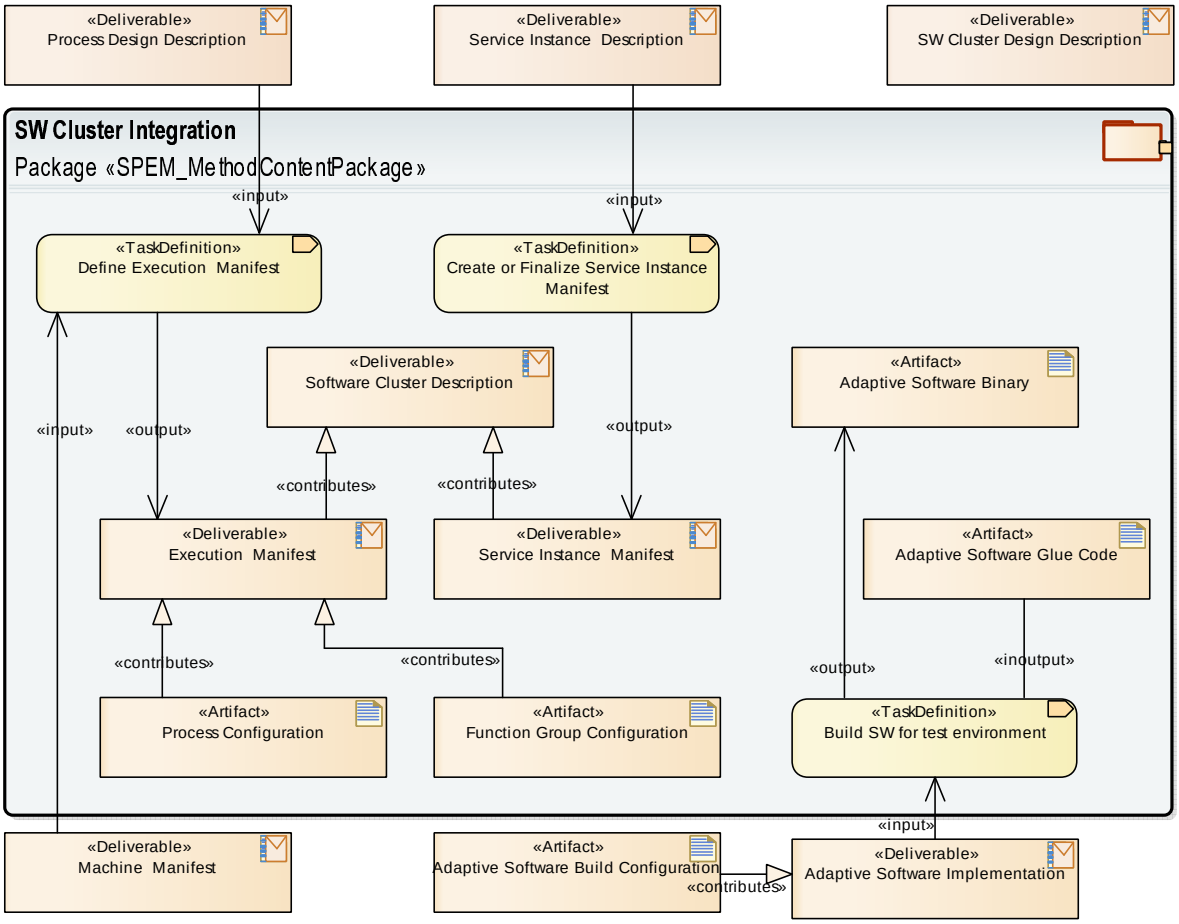

Figure 2.29: Software Cluster Integration (part 1)

One of the key features of the *AUTOSAR adaptive platform* is the ability to extend the software on a given Machine without having to re-flash the entire ECU-HW. Instead, software packages are uploaded to the Machine and installed via UCM.

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

Please be aware that an uploadable software package consists not only of the (executable) software itself, but also of manifest content required to install and run this software on the target Machine in the context of the previously installed AUTOSAR adaptive platform software environment.

The major activities of SoftwareCluster Integration are:

- Build the executable software to obtain the related Adaptive Software Binarys (see below, chapter Build SW for test environment)
- Configure the SoftwareCluster to obtain the Software Cluster Description for the included application-level and platform-level Adaptive Software entities.

  With:

  – Define Execution Manifest resulting in Execution Manifest (see below, chapter Create Execution Manifest)

  – Configure Platform Modules resulting in Platform Module Configurations if the SoftwareCluster includes platform-level Adaptive Software (see below, chapter Create Platform Module Configuration)

  – Create or Finalize Service Instance Manifest resulting in Service Instance Manifests if the SoftwareCluster includes application-level Adaptive Software (see below, chapter Create or Finalize Service Instance Manifest)

  – Integrate Diagnostics resulting in Diagnostic Mappings for Adaptive SW (see below, chapter Integrate Diagnostics)

  Major inputs come from SW Cluster Design Description (see chapter 2.9 Software Cluster Design).

2.10.1 Build SW for test environment

The integration time activity Build SW for test environment (see also [TR_AMETH_00205]) applies to application-level and platform-level Adaptive Software.

Build SW for test environment anticipates and enables the deployment time activity Build SW for target runtime environment (see 2.11Software Packaging):

- Major inputs come from Application Design Description (see 2.4 Application Design) and Adaptive Software Implementation (see 2.5 Implementation of Adaptive Software) with Adaptive Software Source Codes and/or Adaptive Software Object Codes, eventually some Adaptive Software Generated Items and a Adaptive Software Build Configuration.
- An integrator adds the necessary information required to build the (adaptive) Executable as per Executable Description from Application Design Description (see [TR_AMETH_00018]).

This includes the development or finalization of serialization properties for COM (see [TR_AMETH_00016]) and the generation of glue-code e.g. for service proxies and skeletons (see [TR_AMETH_00017]).

- Resulting work products are: Adaptive Software Binarys per executable and/or library specified in Software Cluster Description and the related Diagnostic Manager Binary.

[TR_AMETH_00205] Integrate Software

Status: DRAFT
Upstream requirements: RS_METH_00202, RS_METH_00032

⌈An integrator will take source-code and/or object-code files delivered as Adaptive Software Implementation by the (contracted) software development organization and will bind them together in order to build an (Adaptive) Executable targeting at a specific Machine.

This activity does not include instantiation, i.e., the binding of an Executable to the context of a Process.⌋

[11]

[TR_AMETH_00016] Development of serialization properties

Status: DRAFT
Upstream requirements: RS_METH_00006, RS_METH_00077, RS_METH_00066

⌈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 as Adaptive Software Generated Item during software development or as Adaptive Software Glue Code during integration.⌋

[TR_AMETH_00017] Implementation of service proxies and skeletons

Status: DRAFT
Upstream requirements: RS_METH_00207

⌈The service proxy and skeleton signatures will be generated as header files for Service Interfaces before the actual software development begins and are (likely) available as Adaptive Software Generated Items and used in Adaptive Software Object Code.

[11] Please be aware that an individual Process configured via Process Configuration can start exactly one Executable, but an Executable may be started/instantiated by any number of Processes in arbitrary order.

The implementation for service proxies and skeletons shall (preferably) be generated during integration time as Adaptive Software Glue Code considering the applicable serialization properties.⌋

[TR_AMETH_00018] Building the (adaptive) Executable

Status: DRAFT
Upstream requirements: RS_METH_00202, RS_METH_00066, RS_METH_00042

⌈The (adaptive) Executable can be built based on application-level or platform-level Adaptive Software Object Code together with the respective Main Function being part of Adaptive Software Source Code.

Additionally, the Adaptive Software Glue Code – e.g. with serialization source code, service proxy and skeleton implementations etc. – and all necessary libraries are generated, compiled and linked to an Executable.⌋

2.10.2 Create Execution Manifest

The integration time activity Define Execution Manifest applies to application-level and platform-level Adaptive Software.

- Major inputs come from SW Cluster Design Description (see chapter 2.9 Software Cluster Design)
- An integrator collects and specifies the information required to enable the execution manager to consistently start and stop the Executables of the Function Groups identified in a SoftwareCluster, see [TR_AMETH_00004], [TR_AMETH_00022], [TR_AMETH_00024], [TR_AMETH_00025] and [TR_AMETH_00026].
- Resulting work products are: Execution Manifest with Function Group Configuration and Process Configuration

[TR_AMETH_00004] Creation of the Execution Manifest

Status: DRAFT
Upstream requirements: RS_METH_00203

⌈The integrator activity Define Execution Manifest specifies in the Execution Manifest how the Executables of an Adaptive Software are instantiated by means of OS Processes.

- Instantiation means to bind an (adaptive) Executable to the context of specific Processes of the operating system.
- A major part of the Execution Manifest is the Process Configuration defining the applicable start-up configurations, association to a Function Group and further dependencies.

- Different Processes may start same Executable with different start-up configurations depending on the applicable Function Group states.
- Further on, the Execution Manifest may also define dependencies between Processes and their states.

⌋

[TR_AMETH_00022] Definition of Function Group states

Status: DRAFT
Upstream requirements: RS_METH_00204, RS_METH_00203

⌈The Function Group Configuration defines Function Groups as state-machines for a set of cohesive Executable instances (i.e. Processes) likely having run-time interdependencies.

A Function Group itself does not depend on Processes, instead each Process specifies it's association to exactly one Function Group (i.e. the binding of Process to Function Group is exclusive).

Processes for platform-level Adaptive Software may use a standardized Function Group named "MachineFG" for dependencies on current Machine state, see [SWS_EM_CONSTR_02556] in [13] for details.⌋

[TR_AMETH_00024] Instantiation of an (adaptive) Executable

Status: DRAFT
Upstream requirements: RS_METH_00203, RS_METH_00077

⌈Define the instantiation of an (adaptive) Executable for a specific purpose on a specific Machine in terms of a Process Configuration.

An executable may be instantiated several times with different start-up behavior in arbitrary order. Multiple processes are needed if multiple instances of same executable shall run concurrently.⌋

[TR_AMETH_00025] Definition of the Process start-up behavior

Status: DRAFT
Upstream requirements: RS_METH_00203

⌈For each Process the start-up behavior can depend on one or more states of same Function Group.

Therefore, the Process might have a different start-up behavior in one function group state compared to a second function group state. This behavior can e.g. vary in terms of the scheduling priority or the execution dependencies to other processes.

If an (adaptive) Executable shall run in multiple Processes multiple Function Groups may be effective⌋

[TR_AMETH_00026] Definition of `Execution Manifest`

Status: DRAFT
Upstream requirements: RS_METH_00203

⌈The `Execution Manifest` specifies the `Process Configuration` with:

- the start-up configuration of the associated `Executable`
- the association to exactly one `Function Group` with the dependencies on its `Function Group` states
- dependencies to other processes
- timeouts and resource demand
- scheduling policies

⌋

[TR_AMETH_00216] Map `Process`es to a particular `Machine`

Status: DRAFT
Upstream requirements: RS_METH_00204, RS_METH_00203

⌈The `Process Configuration` is `Machine`-specific, in consequence the `Execution Manifest` shall include a `Process` to `Machine` mapping.⌋

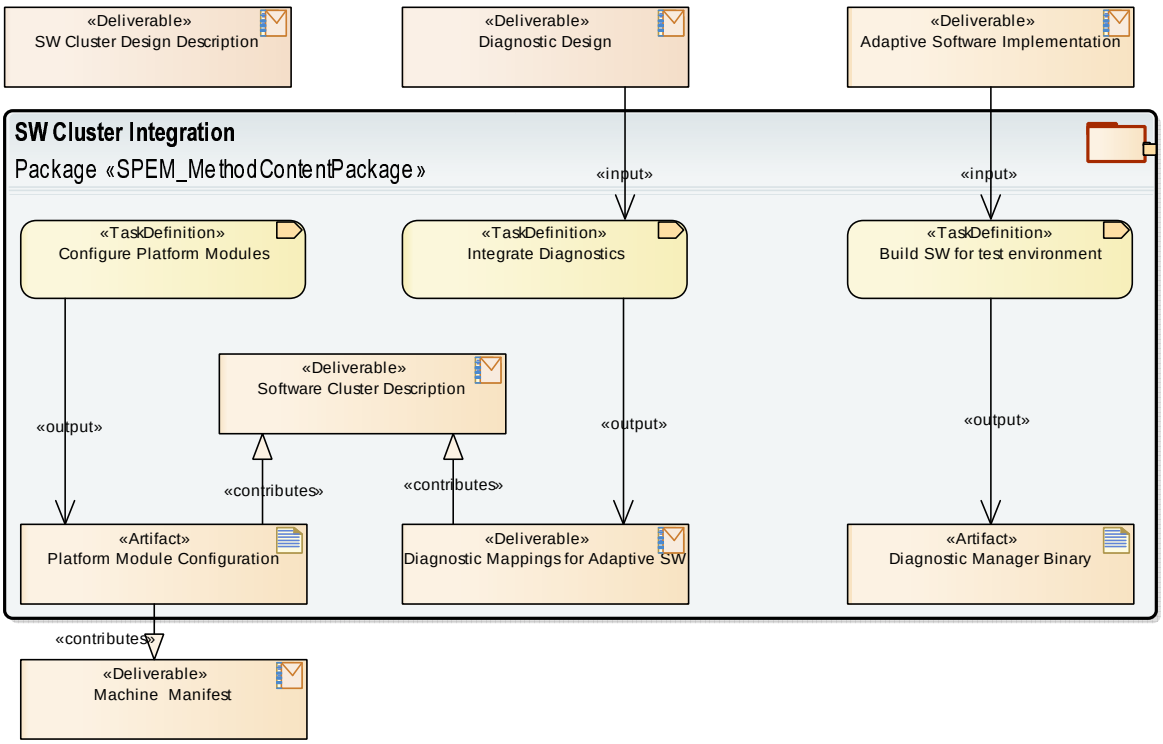

Figure 2.30: Software Cluster Integration (part 2)

2.10.3 Create Platform Module Configuration

The integration time activity `Configure Platform Modules` applies to platform-level `Adaptive Software` and adds `Platform Module Configuration`s to the target `Machine Manifest`. `Configure Platform Modules` is therefore closely related to `Define Machine Manifest` (see 2.8 Machine Manifest).

Various `SoftwareCluster`s for platform-level `Adaptive Software` require a `Machine`-specific `Platform Module Configuration`:

- COM = Communication Management
- CRYPTO
- DoIP = Diagnostics over Internet Protocol
- GTS = (Global) Time Synchronization
- DLT = Log & Trace
- IAM = Identity Access Manager
- IDS = Intrusion Detection System
- NM = Network Management
- OS = Operating System
- PER = Persistency
- PHM = Platform Health Management
- UCM = Update and Configuration Management

which is then associated to the `Software Cluster Description`.

The content of `Platform Module Configuration` is platform module specific and represents the run-time configuration of the platform module.

2.10.4 Create or Finalize Service Instance Manifest

The integration time activity `Create or Finalize Service Instance Manifest` configures the communication endpoints of application-level (and eventually also platform-level) `Adaptive Software`.

- Major inputs come from `SW Cluster Design Description` (see chapter 2.9 Software Cluster Design) with `Service Instance Description`s and `Service instance mapping`s configured as per information available at design time.
- An integrator specifies the `Service Instance Manifest` with

- finalize the Service Instance ID in Service Instance Descriptions as per applicable Service Topology Description from System Topology Description (see also chapter 2.2 System Design).
- enhance the Service instance mappings with the OS Process as per Process Configuration previously matching the mapped Process Design Description (see also chapter 2.10.2 Create Execution Manifest).

[TR_AMETH_00028] Configuration of Service Instances

Status: DRAFT
Upstream requirements: RS_METH_00206, RS_METH_00203

⌈Based on the System Topology Description (preferably shared by OEM at system design time) an integrator defines instances of the deployed service interfaces and decides whether a service instance is provided or consumed.

In order to set up the service-oriented communication Service Instance Manifest includes service discovery properties for search or offer criteria.

E.g. for SOME/IP, an ID for each provided service instance is defined. This ID shall be unique in the vehicle (or if over-the-air communication is involved even beyond the vehicle). For required service instances SOME/IP allows to specify a required service instance ID (which of course should be provided somewhere).⌋

[TR_AMETH_00029] Mapping of Service Instances to a Machine

Status: DRAFT
Upstream requirements: RS_METH_00206, RS_METH_00203, RS_METH_00078

⌈Provided and required Service Instances are allocated to a Machine that will host the Executables implementing them.

This Service Instance to Machine mapping actually considers the applicable communication connector of the Machine, which includes technology specific properties of communication endpoints.
E.g. for SOME/IP, the TCP and IP configuration for the client and the server are described.⌋

[TR_AMETH_00033] Mapping of Service Instances to Ports of Adaptive Software

Status: DRAFT
Upstream requirements: RS_METH_00206, RS_METH_00203, RS_METH_00078

⌈Service instances need to be mapped to their representation in the application-level (and eventually also platform-level) Adaptive Software via the Service Instance to

Port Prototype mapping, which specifies also the OS Process of the Executable exposing the Port.

This mapping decouples the service implementation from its configuration and allows the run-time tailoring of deployed Adaptive Software to a concrete service topology.⌋

2.10.5 Integrate Diagnostics

Associate Diagnostic Mapping with Process Design: It may be necessary that different instances of a particular application software require different diagnostic mappings. Therefore, a relation between a particular diagnostic mapping and a particular Process (Design) needs to be established.

These mappings may be prepared during Diagnostic Design related activities (see chapter 2.6 Diagnostic Design) and finalized at Integrate Diagnostics, considering

- Map Diagnostic Clear Condition to Port(s)
- Map Diagnostic Enable Condition to Port(s)
- Map Diagnostic Event to Port(s)
- Map Diagnostic Generic Service to Port(s)
- Map Diagnostic Indicator to Port(s)
- Map Diagnostic Memory Destination to Port(s)
- Map Diagnostic Operation Cycle to Port(s)
- Map Diagnostic Security Level to Port(s)
- Map Diagnostic Service Data Identifier to Port(s)
- Map Diagnostic SW Service to Port(s)

resulting in now completely filled in Diagnostic Mappings for Adaptive SW.

[TR_AMETH_00213] Relate diagnostic mappings to instances of Executables

Status: DRAFT
Upstream requirements: RS_METH_00207, RS_METH_00201, RS_METH_00016

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

This assignment may be independent of the step of designing diagnostic mappings and may be done in a final extra step (Associate Diagnostic Mapping with Process Design) in Integrate Diagnostics.

This step takes the partly filled in artifact Diagnostic Mappings for Adaptive SW and the artifact Process Design Description as inputs and results in a completely filled in Diagnostic Mappings for Adaptive SW.

2.11 Software Packaging

Figure 2.31 gives an overview of the tasks and work products in scope of Software Packaging. Please find the detailed definitions of these tasks and work products in Adaptive Methodology Library (see chapter 3.11).

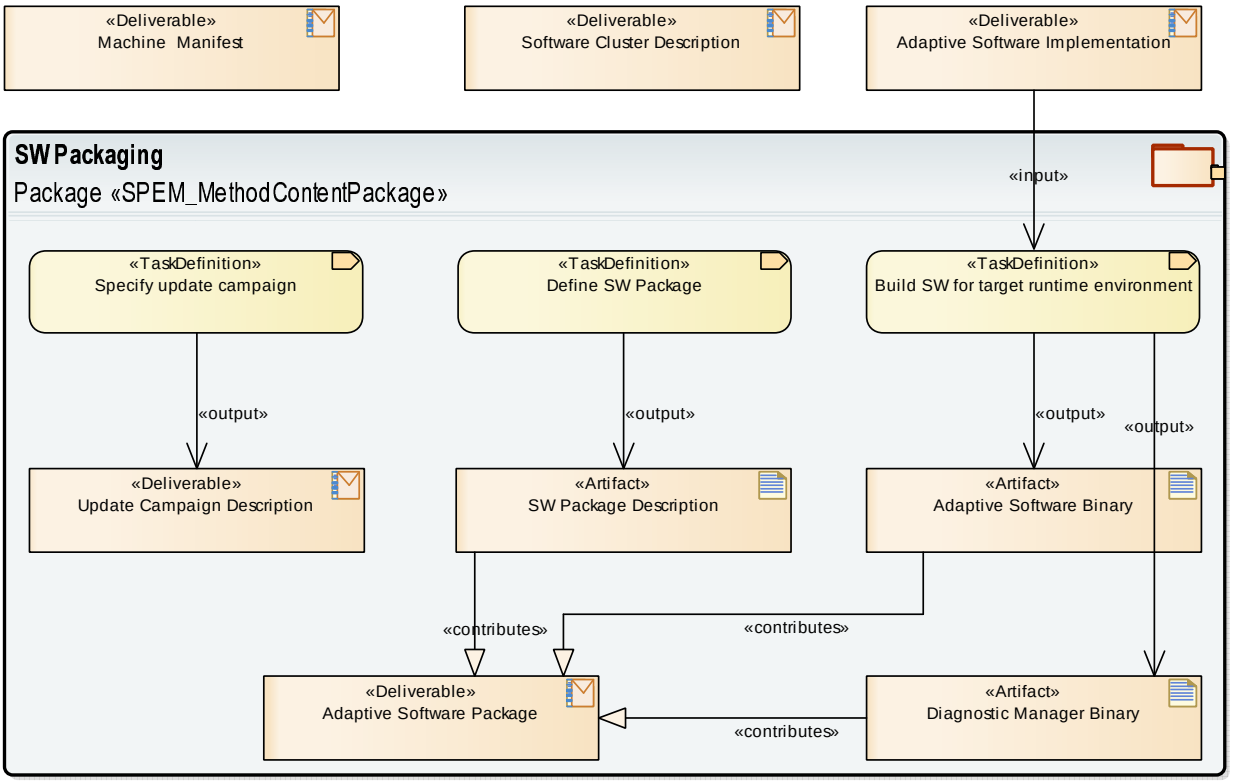

Figure 2.31: Software Packaging

The existence of the `SoftwareCluster` (as outcome of Software Cluster Integration) by itself is not sufficient for installation. Actually, the `SoftwareCluster` gets wrapped into an `Adaptive Software Package` that comes with an own manifest format (the `SW Package Description`) that is at least partly standardized.

The difference between the semantics of a `SoftwareCluster` and the semantics of `Adaptive Software Package` is that a `SoftwareCluster` focuses on the structure of the software itself while the `Adaptive Software Package` is created to handle the logistics aspect of the software installation.

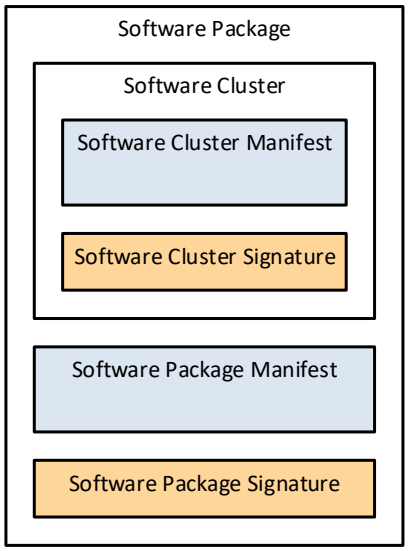

Figure 2.32: Conceptual relation of `SoftwareCluster` and `Software Package`

Major activities are:

- `Define SW Package` resulting in `Adaptive Software Package` with `SW Package Description`, `Adaptive Software Binary` and `Diagnostic Manager Binary`

  see also [TR_AMETH_00006], [TR_AMETH_00206], [TR_AMETH_00218], [TR_AMETH_00219], [TR_AMETH_00222]

- `Specify update campaign` resulting in `Update Campaign Description`

  see also [TR_AMETH_00220], [TR_AMETH_00221]

- perform the actual software update

  see also [TR_AMETH_00031], [TR_AMETH_00223], [TR_AMETH_00224], [TR_AMETH_00225]

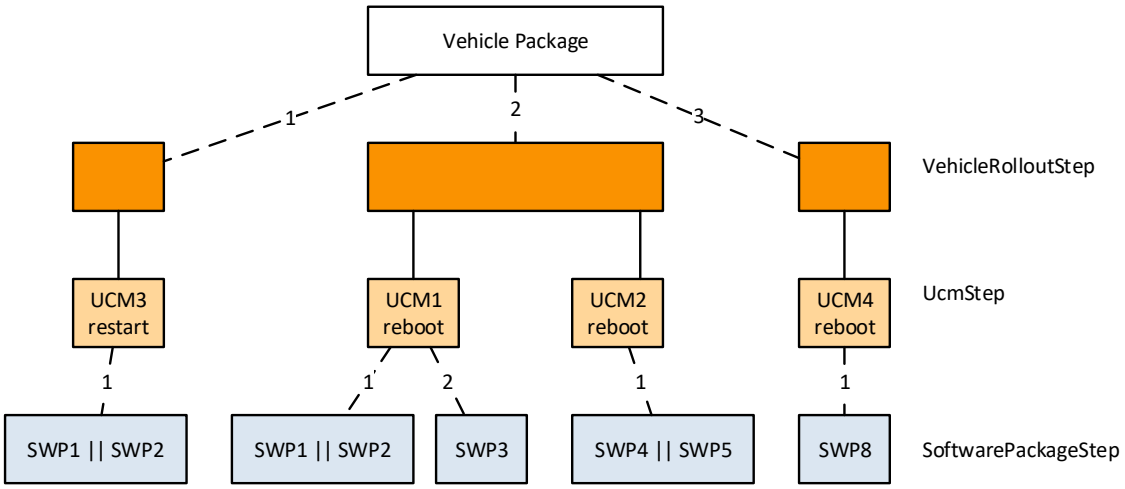

Figure 2.33: Conceptual view on an update campaign

[TR_AMETH_00224] Management of Adaptive Software Packages

Status: DRAFT
Upstream requirements: RS_METH_00205

⌈Once SW Package Descriptions and SW package payload like Adaptive Software Binarys and Diagnostic Manager Binary have been created, they are generally ready to be deployed via Adaptive Software Package to dedicated adaptive Machines in the field.

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

The management of this repository of the Adaptive Software Packages may be supported by means of databases.

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

[TR_AMETH_00031] Setting up an initial machine

Status: DRAFT
Upstream requirements: RS_METH_00205, RS_METH_00204

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

1. Install the selected Operating System on the selected target (machine).

2. Install all necessary Platform modules on top of the installed OS in order to be able to perform the upload and the installation of additional application software by means of Adaptive Software Packages.

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

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

⌋

2.11.1 Define SW Package

[TR_AMETH_00006] Deployment of the application software

Status: DRAFT
Upstream requirements: RS_METH_00205

⌈Software is deployed to a Machine (i.e. to a particular Adaptive AUTOSAR Platform instance), by means of Adaptive Software Packages. This means that:

1. associated software artifacts need to be compiled, built and added as SW package payload like Adaptive Software Binarys and Diagnostic Manager Binary.
2. Adaptive Software Packages are provided by an OEM-specific Back-end server in order to be accessible by the machines in the field.

⌋

[TR_AMETH_00206] Create a Adaptive Software Package

Status: DRAFT
Upstream requirements: RS_METH_00205

⌈The following activities/tasks are needed in order to obtain a Adaptive Software Package:

- Create a SW Package Description

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

⌋

[TR_AMETH_00218] Create an initial SW Package Description

Status: DRAFT
Upstream requirements: RS_METH_00205

⌈The main input for this step are the requirements of the OEM given by means of SW Cluster Design Description. Thus, this task is about to create an new SW Package Description and to transfer the structure and the entries of the given SW Cluster Design Description into the newly created SW Package Description.⌋

[TR_AMETH_00219] Collect all software artifacts that belong to a SoftwareCluster, structure and model them

Status: DRAFT
Upstream requirements: RS_METH_00205

⌈On base of the SW Cluster Design Description of the newly created SW Package Description, this step includes the following sub-tasks:

- Identify necessary (software) artifacts
  - Identify necessary (software) artifacts in order to build the Adaptive Software Package, also with respect to their versions
  - Check, whether there are deviations between the required and actual sets of *Sub Software Cluster*s (by means of the aggregated artifacts and versions), if necessary solve them and re-model the SW Package Description accordingly
  - Check, whether there are discrepancies between the required and actual set of the (root) Software Cluster Description (by means of its aggregated *Sub Software Clusters* and versions)
- Collect belonging (software) artifacts of *Sub Software Cluster*s
  - Collect belonging (software) artifacts of *Sub Software Cluster*s into separate baskets (*Sub Software Clusters*) in order to prepare the final step of creating the SW Package Description

- Execute a receiving inspection (optional)
- Store incoming artifacts into a repository

⌋

[TR_AMETH_00222] Create the Adaptive Software Package

Status: DRAFT
Upstream requirements: RS_METH_00205

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

Thus, this activity handles the compilation of Software Cluster Description and SW Package Description into a Adaptive Software Package.

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

[TR_AMETH_00223] Manage the data base of Software Cluster Descriptions (of any category)

Status: DRAFT
Upstream requirements: RS_METH_00205

⌈A general activity may be the management of the data base of SoftwareClusters with respect to all their versions, dependencies and further aspects.

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

2.11.2 Specify update campaign

[TR_AMETH_00220] Model dependencies between SoftwareClusters of any category

Status: DRAFT
Upstream requirements: RS_METH_00205

⌈Dependencies between SoftwareClusters of the same or different categories may already be given by the requirements of an OEM by means of a SW Cluster Design Description. Dependencies to SoftwareClusters are specified by means of their identification (name) and version.

Because dependencies may change during the development process, the activity needs to handle this case. Therefore, this task describes the handling of dependencies by at least the following sub-tasks:

- Check, whether the dependencies between Software Cluster Descriptions of the same or different categories, given by the respective SW Cluster Design Description are still valid
- Determine changes between the actual and required dependencies between Software Cluster Descriptions of any category
- If necessary, re-model the SW Package Description in accordance with the outcomes of the two tasks above

⌋

[TR_AMETH_00221] Develop installation instructions

Status: DRAFT
Upstream requirements: RS_METH_00205

⌈Installation instruction control the behavior of the UCM during the update of Adaptive Software Packages. Installation instructions can either be 'add/update' meaning to install a package or 'remove' to express that a package shall be uninstalled and deleted from the machine. Installation instructions are defined per Software Cluster Description, independent of its category. For details, see [14, Specification of Update and Configuration Management].

Thus, this task may includes the sub-tasks:

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

The particular installation instructions are part of the SW Package Description.⌋

[TR_AMETH_00225] Provision of Adaptive Software Packages for machines in the field

Status: DRAFT
Upstream requirements: RS_METH_00205

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

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

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 High Level Architecture

This chapter contains the definition of tasks and work products related to High Level Architecture (see chapter 2.1).

3.1.1 Tasks

3.1.1.1 Define High Level Architecture

Task Definition	**Define High Level Architecture**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::High Level Architecture		
Brief Description	Define the High Level Architecture of a vehicle E/E system		
Description	One or more E/E system architects evaluate and specify vehicle functions, features and requirements necessary for a specific vehicle E/E project or project family and outline the future implementation on an abstract and platform-spanning level. This includes: • Develop Function Architecture • Develop Abstract Platform Specification • Develop Vehicle Software Architecture • Develop Vehicle Hardware Architecture		
Relation Type	***Related Element***	***Mult.***	***Note***

Table 3.1: Define High Level Architecture

3.1.1.2 Develop Function Architecture

Task Definition	**Develop Function Architecture**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::High Level Architecture		
Brief Description	Describe vehicle functions disregarding their future implementation.		
Description	An E/E system architect evaluates and specifies vehicle functions, features and requirements necessary for a specific vehicle E/E project or project family. See also [TR_AMETH_00201]		
Relation Type	***Related Element***	***Mult.***	***Note***
Produces	Function Architecture	1	

Table 3.2: Develop Function Architecture

3.1.1.3 Develop Abstract Platform Specification

Task Definition	**Develop Abstract Platform Specification**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::High Level Architecture		
Brief Description	Describe a platform independent component model for vehicle functions and their interaction		
Description	An E/E system architect evaluates vehicle functions (as defined in Function Architecture) and derives a platform independent (and likely coarse grained) component model outlining the future implementation of vehicle functions and their interaction. This activity defines an abstract view on the overall system of a vehicle E/E project.		
Relation Type	***Related Element***	***Mult.***	***Note***
Produces	Abstract Platform Specification	1	

Table 3.3: Develop Abstract Platform Specification

3.1.1.4 Develop Vehicle Software Architecture

Task Definition	**Develop Vehicle Software Architecture**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::High Level Architecture		
Brief Description	Provide a dedicated view of all AUTOSAR software entities and their communication relations within the vehicle E/E system		
Description	A software architect analyzes • either the Function Architecture (defining vehicle functions, features and requirements necessary for a specific vehicle E/E project or project family) • either the Abstract Platform Specification (defining a platform independent component model for vehicle functions and their interaction) and derives a corresponding Vehicle Software Architecture considering all AUTOSAR software entities and their interactions. See also [TR_AMETH_00202]		
Relation Type	***Related Element***	***Mult.***	***Note***
Produces	Vehicle Software Architecture	1	

Table 3.4: Develop Vehicle Software Architecture

3.1.1.5 Develop Vehicle Hardware Architecture

Task Definition	**Develop Vehicle Hardware Architecture**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::High Level Architecture		
Brief Description	Define the ECU topology for a vehicle E/E system		
Description	An E/E system architect defines the ECUs and communication infrastructure for a vehicle E/E system. These ECUs are designed to host the software entities from Vehicle Software Architecture and consider the required HW resources for AP (Machines), CP (ECU-Instances) and communication infrastructure.		
Relation Type	***Related Element***	***Mult.***	***Note***
Produces	Vehicle Hardware Architecture	1	

Table 3.5: Develop Vehicle Hardware Architecture

3.1.2 Work Products

3.1.2.1 High Level Architecture

Deliverable	**High Level Architecture**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::High Level Architecture		
Brief Description	Definition of the High Level Architecture of a vehicle E/E system		
Description	The High Level Architecture considers vehicle functions, features and requirements necessary for a specific vehicle E/E project or project family and outlines the future implementation on an abstract and platform-spanning level. Elements are: • Function Architecture • Abstract Platform Specification • Vehicle Software Architecture • Vehicle Hardware Architecture		
Kind	ARXML and non-AUTOSAR artifacts		
Extended By	Abstract Platform Specification, Function Architecture, Vehicle Hardware Architecture, Vehicle Software Architecture		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumed by	Define System Topology	1	
Consumed by	Identify Software Cluster	1	

Table 3.6: High Level Architecture

3.1.2.2 Function Architecture

Artifact	**Function Architecture**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::High Level Architecture		
Brief Description	Description of vehicle functions disregarding their future implementation.		
Description	The Function Architecture consists of a function network representing functionalities that are needed to execute particular vehicle functions. These functionalities may be realized in software or hardware or as a mix of both. See also [TR_AMETH_00201]		
Kind	non-AUTOSAR artifacts		
Extends	High Level Architecture		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Develop Function Architecture	1	

Table 3.7: Function Architecture

3.1.2.3 Abstract Platform Specification

Artifact	**Abstract Platform Specification**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::High Level Architecture		
Brief Description	Description of a platform independent component model for vehicle functions and their interaction		
Description	The Abstract Platform Specification identifies • abstract components disregarding their future implementation as AUTOSAR AP / CP / non-AUTOSAR software entities or as hardware entities. • communication endpoints based on abstract PortInterface descriptions (see [TR_AMETH_00001])		
Kind	ARXML		
Extends	High Level Architecture		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Develop Abstract Platform Specification	1	

Table 3.8: Abstract Platform Specification

3.1.2.4 Vehicle Software Architecture

Artifact	**Vehicle Software Architecture**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::High Level Architecture		
Brief Description	Description of the AUTOSAR component model for vehicle functions and their interaction		
Description	The Vehicle Software Architecture provides a dedicated view of all AUTOSAR software entities and their communication relation within the vehicle E/E system. With: • AUTOSAR software components of the Adaptive Platform (AP-SWCs), • AUTOSAR software components of the Classic Platform (CP-SWCs), • AUTOSAR software compositions with arbitrary combinations of AP- and/or CP- SWCs, • the communication between software components. See also [TR_AMETH_00202]		
Kind	ARXML		
Extends	High Level Architecture		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Develop Vehicle Software Architecture	1	

Table 3.9: Vehicle Software Architecture

3.1.2.5 Vehicle Hardware Architecture

Artifact	**Vehicle Hardware Architecture**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::High Level Architecture		
Brief Description	Description of the ECU topology for a vehicle E/E system		
Description	The Vehicle Hardware Architecture defines the ECUs and communication infrastructure for an vehicle E/E system. These ECUs are designed to host the software entities from Vehicle Software Architecture and consider the required HW resources for AP (Machines), CP (ECU-Instances) and communication infrastructure.		
Kind	ARXML and non-AUTOSAR artifacts		
Extends	High Level Architecture		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Develop Vehicle Hardware Architecture	1	

Table 3.10: Vehicle Hardware Architecture

3.2 System Design

This chapter contains the definition of tasks and work products related to System Design (see chapter 2.2).

3.2.1 Tasks

3.2.1.1 Define Global Time

Task Definition	**Define Global Time**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::System Design		
Brief Description	Define the vehicle-wide time synchronization of ECUs.		
Description	It may be necessary that several ECUs in a vehicle E/E system need to act in concert while executing a (distributed) vehicle function. To achieve this the global time functional clusters across all connected CP-ECU-Instances and AP-Machines need to synchronize to same time bases.		
Relation Type	***Related Element***	***Mult.***	***Note***
Produces	Global Time Description	1	

Table 3.11: Define Global Time

3.2.1.2 Define Network Management

Task Definition	**Define Network Management**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::System Design		
Brief Description	Define the vehicle-wide wake-up and sleep behavior of Networks		
Description	Typically an OEM provides the network configuration with all configuration settings that are relevant for the network management functional clusters across all connected CP-ECU-Instances and AP-Machines in a vehicle.		
Relation Type	***Related Element***	***Mult.***	***Note***
Produces	Network Management Description	1	

Table 3.12: Define Network Management

3.2.1.3 Identify Software Cluster

Task Definition	**Identify Software Cluster**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::System Design		
Brief Description	Define software clusters as major building blocks in the (sub)system software architecture		
Description	Identify and outline software clusters as per information available at system design time. We shall/may group software components into software clusters with the intention to deploy the software clusters on an ECU. • For CP this is an optional grouping to facilitate the mapping of SWCs to ECU-Instances. • For AP this is a mandatory grouping of SWCs taking part in the build of executables to be deployed on a specific Machine.		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	High Level Architecture	1	

▽

△

Task Definition	**Identify Software Cluster**		
Produces	SW Cluster Design Description	1	

Table 3.13: Identify Software Cluster

3.2.1.4 Define System Topology

Task Definition	**Define System Topology**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::System Design		
Brief Description	Define sub-systems with the required functionalities and their communication needs		
Description	Refine the Vehicle Software Architecture and define sub-systems with the required functionalities and their communication needs. This includes: • Specify groups for distributed but coherent functionality. • Specify the network connectivity for the involved communication clusters • Specify the network of services		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	High Level Architecture	1	
In/out	Function Group Description	1	
In/out	Network Endpoint Description	1	
In/out	Service Topology Description	1	
In/out	System Topology Description	1	

Table 3.14: Define System Topology

3.2.1.5 Define Signal to Service Translation

Task Definition	**Define Signal to Service Translation**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::System Design		
Brief Description	Define the communication between AP and CP SWCs via signal/service translation		
Description	The background of this activity is the request to enable service oriented communication across AUTOSAR platforms even in use cases where the Classic Platform (CP) does not fully comply to SOME/IP. In such cases the signal based communication defined for CP needs to be translated into SOME/IP messages (and vice versa, as required for Adaptive Platform (AP)) See also [TR_AMETH_00207] and [TR_AMETH_00208]		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Service Interface Description	1	

▽

△

Task Definition	**Define Signal to Service Translation**		
Produces	Signal to Service Translation Description	1	

Table 3.15: Define Signal to Service Translation

3.2.2 Work Products

3.2.2.1 System Design

Deliverable	**System Design**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::System Design		
Brief Description	The System Design represents a - preferably platform specific - sub-system derived from High Level Architecture.		
Description	Elements are: • Global Time Description • Network Management Description • System Topology Description, enhanced by intial versions of SW Cluster Design Descriptions • Signal to Service Translation Description		
Kind	ARXML		
Extended By	Global Time Description, Network Management Description, Signal to Service Translation Description, System Topology Description		
Relation Type	*Related Element*	*Mult.*	*Note*

Table 3.16: System Design

3.2.2.2 Global Time Description

Artifact	**Global Time Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::System Design		
Brief Description	Definition of the vehicle-wide time synchronization of ECUs.		
Description	It may be necessary that several ECUs in a vehicle E/E system need to act in concert while executing a (distributed) vehicle function. To achieve this the global time functional clusters across all connected CP-ECU-Instances and AP-Machines need to synchronize to same time bases.		
Kind	ARXML		
Extends	System Design		
Relation Type	*Related Element*	*Mult.*	*Note*
Produced by	Define Global Time	1	

Table 3.17: Global Time Description

3.2.2.3 Network Management Description

Artifact	**Network Management Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::System Design		
Brief Description	Definition of the vehicle-wide wake-up and sleep behavior of Networks		
Description	Typically an OEM provides the network configuration with all configuration settings that are relevant for the network management functional clusters across all connected CP-ECU-Instances and AP-Machines in a vehicle.		
Kind	ARXML		
Extends	System Design		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Define Network Management	1	

Table 3.18: Network Management Description

3.2.2.4 System Topology Description

Artifact	**System Topology Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::System Design		
Brief Description	Definition of sub-systems with the required functionalities and their communication needs		
Description	Definition of sub-systems with the required functionalities and their communication needs. This includes: • Function Group Description • Network Endpoint Description • Service Topology Description		
Kind	ARXML		
Extended By	Function Group Description, Machine Design, Network Endpoint Description, Service Topology Description		
Extends	System Design		
Relation Type	***Related Element***	***Mult.***	***Note***
In/out	Define System Topology	1	
Consumed by	Define provided and required service instances	1	

Table 3.19: System Topology Description

Artifact	**Function Group Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::System Design		
Brief Description	Contribution of Function Group Descriptions to System Topology		
Description	This extends the System Topology Description with the functional grouping of distributed but coherent functionality.		
Kind	ARXML		
Extends	System Topology Description		
Relation Type	***Related Element***	***Mult.***	***Note***
In/out	Define System Topology	1	

Table 3.20: Function Group Description

Artifact	**Network Endpoint Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::System Design		
Brief Description	Contribution of Network Endpoint Descriptions to System Topology		
Description	This extends the System Topology Description with the details about network connectivity.		
Kind	ARXML		
Extends	System Topology Description		
Relation Type	***Related Element***	***Mult.***	***Note***
In/out	Define System Topology	1	
Consumed by	Define Machine Design	1	

Table 3.21: Network Endpoint Description

Artifact	**Service Topology Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::System Design		
Brief Description	Contribution of Service Topology Descriptions to System Topology		
Description	This extends the System Topology Description with the details about (technology specific) services and their instantiation.		
Kind	ARXML		
Extends	System Topology Description		
Relation Type	***Related Element***	***Mult.***	***Note***
In/out	Define System Topology	1	
Consumed by	Define provided and required service instances	1	

Table 3.22: Service Topology Description

3.2.2.5 Signal to Service Translation Description

Artifact	**Signal to Service Translation Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::System Design		
Brief Description	Definition of the communication between AP and CP SWCs via signal/service translation		
Description	This is related to service oriented communication across AUTOSAR platforms in use cases where the Classic Platform (CP) does not fully comply to SOME/IP. In such cases the Signal to Service Translation Description defines how the signal based communication defined for CP shall be translated into SOME/IP messages (and vice versa, as required for Adaptive Platform (AP)) in the context of an individual Service Instance. This includes the mapping of elements of AP-based ServiceInterfaces to elements of corresponding elements of CP-based SenderReceiverInterfaces, ClientServerInterfaces and TriggerInterfaces considering the context with a PDU or Ethernet-socket on CP side and a Service Instance on AP side See also [TR_AMETH_00207] and [TR_AMETH_00208]		
Kind	ARXML		
Extends	System Design		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Define Signal to Service Translation	1	

Table 3.23: Signal to Service Translation Description

3.3 Application Design

This chapter contains the definition of tasks and work products related to Application Design (see chapter 2.4).

3.3.1 Tasks

3.3.1.1 Define Executable with enclosed SW Composition

Task Definition	**Define Executable with enclosed SW Composition**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Application Design		
Brief Description	Describe an Executable containing one or more software components		
Description	The executable can contain an arbitrary hierarchy of composition and atomic software components. • The atomic adaptive software components contain the actual functionality of the executable. • Executables can be of category application-level or platform-level. Executables may have a flat software composition. In this case the executable contains just one atomic adaptive software component. Executables may have a hierarchical software composition. In this case the executable contains a nested set-up of composition and atomic software components. This includes the possibility of multiple instantiation of software components on all nesting levels.		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	SW Component Design	1	
Produces	Executable Description	1	

Table 3.24: Define Executable with enclosed SW Composition

3.3.1.2 Define Interaction with Applications

Task Definition	**Define Interaction with Applications**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Application Design		
Brief Description	Describe the (service oriented) communication endpoints of Executable instances		
Description	The communication endpoint context consists of • the port instance inside the SW composition of the Executable • the Process Design representing the Executable instance		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Process Design Description	1	
Consumes	SW Composition Description	1	
Produces	SW Interaction Description	1	

Table 3.25: Define Interaction with Applications

3.3.1.3 Define Interaction with Functional Clusters

Task Definition	**Define Interaction with Functional Clusters**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Application Design		
Brief Description	Describe how Executables access functional clusters		
Description	The functional cluster access point context consists of • the port instance inside the SW composition of the Executable • the Process Design representing the Executable instance		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Process Design Description	1	
Consumes	SW Composition Description	1	
Produces	SW Interaction Description	1	

Table 3.26: Define Interaction with Functional Clusters

3.3.1.4 Define SW-Component Design

Task Definition	**Define SW Component Design**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Application Design		
Brief Description	Describe an (adaptive) software component with its ports		
Description	Adaptive software components communicate via service interfaces. The related ports follow the standardized service proxy/skeleton patterns: • a RPort is used, if the software component requires a service interface. • a PPort is used, if the software component provides a service interface. Adaptive software components may communicate with functional clusters via dedicated interfaces and ports. Besides that adaptive software components may use the functional cluster API as specified in SWS.		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Functional Cluster Interface Description	1	
Consumes	Service Interface Description	1	
Produces	SW Component Design	1	

Table 3.27: Define SW Component Design

3.3.1.5 Define Process Design

Task Definition	**Define Process Design**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Application Design		
Brief Description	Define design time proxies for OS processes		
Description	Executables are started by processes. Since processes will be defined only at integration time we need a process proxy at design time: the Process Design. The Process Design allows to distinguish different instances of Executables at design time (e.g. at map service instances to port instances).		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Executable Description	1	
Produces	Process Design Description	1	

Table 3.28: Define Process Design

3.3.2 Work Products

3.3.2.1 Application Design Description

Deliverable	**Application Design Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Application Design		
Brief Description	Container of elements related to Application Design		
Description	Elements related to Application Design are • Executable Description – Functional Cluster Port Description – SW Port Description – SW Composition Description • Process Design Description		
Kind	ARXML		
Extended By	DEXT, Diagnostic Contribution Description, Executable Description, Function Group Configuration, Platform Module Configuration, Process Design Description, SW Component Design, SW Interaction Description, SW to Diagnostics Interaction Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumed by	Associate content elements	1	
Consumed by	Develop Adaptive Software	1	
Consumed by	Map provided and required service instances to contained executables	1	

Table 3.29: Application Design Description

3.3.2.2 Functional Cluster Interface Description

Deliverable	**Functional Cluster Interface Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Application Design		
Brief Description	Description of the standardized functional cluster interfaces		
Description	The SWS of the functional clusters define • the available dedicated interfaces to be used in ports of software components. • the API to be used on C++ level		
Kind	ARXML and SWS		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumed by	Define SW Component Design	1	

Table 3.30: Functional Cluster Interface Description

3.3.2.3 Process Design Description

Deliverable	**Process Design Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Application Design		
Brief Description	Description of a Proxy for a Process at design time		
Description	This element stands in as a proxy for a Process at the time when it does not yet exist (i.e., at design time - before an Integrator has defined the actual Process). The Process Design allows to distinguish different instances of Executables at design time.		
Kind	ARXML		
Extends	Application Design Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Define Process Design	1	
Consumed by	Associate content elements	1	
Consumed by	Define Execution Manifest	1	
Consumed by	Define Interaction with Applications	1	
Consumed by	Define Interaction with Functional Clusters	1	

Table 3.31: Process Design Description

3.3.2.4 Executable Description

Artifact	**Executable Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Application Design		
Brief Description	Description of an Executable containing one or more software components		
Description	The executable can contain an arbitrary hierarchy of composition and atomic software components. The atomic adaptive software components contain the actual functionality of the executable. Executables can be of category application-level or platform-level.		
Kind	ARXML		
Extended By	SW Composition Description		
Extends	Application Design Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Define Executable with enclosed SW Composition	1	
Consumed by	Define Process Design	1	

Table 3.32: Executable Description

3.3.2.5 SW Component Design

Artifact	**SW Component Design**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Application Design		
Brief Description	Describe an (adaptive) software component with its ports		
Description	The description of an Adaptive software components identifies ports: • Ports for communication via service interfaces follow the standardized service proxy/skeleton patterns: – a RPort is used, if the software component requires a service interface. – a PPort is used, if the software component provides a service interface. • Adaptive software components may communicate with functional clusters via dedicated interfaces and ports. • Besides that adaptive software components may use the functional cluster API as specified in SWS.		
Kind	ARXML		
Extends	Application Design Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Define SW Component Design	1	
Consumed by	Define Executable with enclosed SW Composition	1	

Table 3.33: SW Component Design

3.3.2.6 SW Composition Description

Artifact	**SW Composition Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Application Design		
Brief Description	Description of a (hierarchical) software component for the Adaptive Platform		
Description	Executables may have a flat software composition. In this case the executable contains just one atomic adaptive software component. Executables may have a hierarchical software composition. In this case the executable contains a nested set-up of composition and atomic software component. This includes the possibility of multiple instantiation of software components on all nesting levels.		
Kind	ARXML		
Extends	Executable Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumed by	Define Interaction with Applications	1	
Consumed by	Define Interaction with Functional Clusters	1	

Table 3.34: SW Composition Description

3.3.2.7 SW Interaction Description

Artifact	**SW Interaction Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Application Design		
Brief Description	Describe how Executables access ports of application-level and platform-level software		
Description	This considers • the (service oriented) communication ports of Executables • the access points to functional clusters In both cases the the port context consists of • the port instance inside the SW composition of the Executable • the Process Design representing the Executable instance		
Kind	ARXML		
Extended By	SW to Diagnostics Interaction Description		
Extends	Application Design Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Define Interaction with Applications	1	
Produced by	Define Interaction with Functional Clusters	1	

Table 3.35: SW Interaction Description

3.4 Adaptive Software Implementation

This chapter contains the definition of tasks and work products related to Implementation of Adaptive Software (see chapter 2.5).

3.4.1 Tasks

3.4.1.1 Develop Adaptive Software

Task Definition	**Develop Adaptive Software**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Adaptive Software Implementation		
Brief Description	Activities related to the development of application-level and/or platform-level Adaptive Software		
Description	The development of application-level and/or platform-level Adaptive Software can start when the adaptive (service) interfaces have been defined. This software development may include several sub-activities like analysis, design, implementation or test. The most important outcome of this activity are either source-code or object-code artifacts, depending on whether or not the developer knows the Adaptive Software Build Configuration beforehand. See also [TR_AMETH_00002], [TR_AMETH_00012], [TR_AMETH_00013], [TR_AMETH_00014], [TR_AMETH_00015] and , [TR_AMETH_00020]		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Application Design Description	1	
Produces	Adaptive Software Implementation	1	

Table 3.36: Develop Adaptive Software

3.4.2 Work Products

3.4.2.1 Adaptive Software Implementation

Deliverable	**Adaptive Software Implementation**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Adaptive Software Implementation		
Brief Description	Container of elements related to Adaptive Software Implementation		
Description	Elements related to Adaptive Software Implementation are • Adaptive Software Source Code • Adaptive Software Object Code • Adaptive Software Generated Item • Adaptive Software Build Configuration		
Kind			
Extended By	Adaptive Software Build Configuration, Adaptive Software Generated Item, Adaptive Software Object Code, Adaptive Software Source Code		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Develop Adaptive Software	1	
Consumed by	Build SW for target runtime environment	1	
Consumed by	Build SW for test environment	1	

Table 3.37: Adaptive Software Implementation

3.4.2.2 Adaptive Software Source Code

Artifact	**Adaptive Software Source Code**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Adaptive Software Implementation		
Brief Description	Source Code		
Description	This includes the SW component source code and a main function per Executable. 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		
Extends	Adaptive Software Implementation		
Relation Type	***Related Element***	***Mult.***	***Note***

Table 3.38: Adaptive Software Source Code

3.4.2.3 Adaptive Software Object Code

Artifact	**Adaptive Software Object Code**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Adaptive Software Implementation		
Brief Description	Object Code (i.e. the compiled source code)		
Description	This is the compiler result. In case the integrator is not responsible for the compilation of the software components, the object code will be delivered.		
Kind	Object Code		
Extends	Adaptive Software Implementation		
Relation Type	***Related Element***	***Mult.***	***Note***

Table 3.39: Adaptive Software Object Code

3.4.2.4 Adaptive Software Generated Item

Artifact	**Adaptive Software Generated Item**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Adaptive Software Implementation		
Brief Description	Generated Code		
Description	This includes generated Header Files (and implementations) for Adaptive Interfaces For service interfaces this is: proxy/skeleton header and implementation. This may include also SOME/IP Serialization Source Code.		
Kind	Generated Code		
Extends	Adaptive Software Implementation		
Relation Type	***Related Element***	***Mult.***	***Note***

Table 3.40: Adaptive Software Generated Item

3.4.2.5 Adaptive Software Build Configuration

Artifact	**Adaptive Software Build Configuration**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Adaptive Software Implementation		
Brief Description	Build scripts		
Description	The Build Chain Configuration contains the used compiler and linker settings. These settings are platform implementation specific.		
Kind	Build scripts		
Extends	Adaptive Software Implementation		
Relation Type	***Related Element***	***Mult.***	***Note***

Table 3.41: Adaptive Software Build Configuration

3.5 Diagnostic Design

This chapter contains the definition of tasks and work products related to Diagnostic Design (see chapter 2.6).

3.5.1 Tasks

3.5.1.1 Define Diagnostic Contribution Description

Task Definition	**Define Diagnostic Contribution Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Diagnostic Design		
Brief Description	Describe the Diagnostic Contribution applicable for a specific Software Cluster		
Description	Specify what content of a DEXT shall be used in the Diagnostic Manager tailored for the Executables and Processes in a specific Software Cluster.		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	DEXT	1	
Produces	Diagnostic Contribution Description	1	

Table 3.42: Define Diagnostic Contribution Description

3.5.1.2 Define Diagnostic Interface Description

Task Definition	**Define Diagnostic Interface Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Diagnostic Design		
Brief Description	Description of Diagnostic Interfaces in scope of adaptive (application) software		
Description	Covers the below listed Diagnostic Interface kinds: • DiagnosticConditionInterface • DiagnosticDataIdentifierInterface. • DiagnosticDTCInformationInterface. • DiagnosticEventInterface • DiagnosticIndicatorInterface • DiagnosticMonitorInterface • DiagnosticOperationCycleInterface • DiagnosticSecurityLevelInterface • DiagnosticServiceValidationInterface to be used in the diagnostic ports of adaptive (application) software.		
Relation Type	***Related Element***	***Mult.***	***Note***
Produces	Diagnostic Interface Description	1	

Table 3.43: Define Diagnostic Interface Description

3.5.1.3 Provide DEXT for Application Set-up

Task Definition	**Provide DEXT for Application Set-up**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Diagnostic Design		
Brief Description	Collect the diagnostic resources and mappings for adaptive (application) software		
Description	Collect the applicable DEXT content specified during Application Design and/or Diagnostic Design and/or Software Cluster Design and/or Software Cluster Integration. This includes all activities to • specify diagnostic resources (like e.g. diagnostic data identifiers / enable conditions / events / operation cycles) and add them to DEXT. • specify diagnostic mappings of diagnostic resources to adaptive software port instances (see also [TR_AMETH_00212]).		
Relation Type	***Related Element***	***Mult.***	***Note***
Produces	DEXT	1	

Table 3.44: Provide DEXT for Application Set-up

3.5.2 Work Products

3.5.2.1 Diagnostic Design

Deliverable	**Diagnostic Design**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Diagnostic Design		
Brief Description	Collection of all artifacts in scope of a specific Diagnostic Design		
Description	Diagnostic-Design includes • Diagnostic-Interface-Descriptions • the Diagnostic Extract (DEXT) specifying diagnostic resources and diagnostic mappings • the Diagnostic-Contribution-Description for a concrete tailoring of DEXT for the Diagnostic Manager of a specific Software Cluster.		
Kind	ARXML		
Extended By	DEXT, Diagnostic Contribution Description, Diagnostic Interface Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumed by	Associate Diagnostic Address and Contribution	1	
Consumed by	Integrate Diagnostics	1	

Table 3.45: Diagnostic Design

3.5.2.2 Diagnostic Extract (DEXT)

Artifact	**DEXT**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Diagnostic Design		
Brief Description	Collection of diagnostic descriptions as defined in TPS DEXT and TPS Manifest		
Description	Specify • diagnostic properties and • diagnostic resources (like e.g. diagnostic data identifiers / enable conditions / events / operation cycles) and • diagnostic mappings of diagnostic resources to adaptive software port instances (represented at design-time by SW-to-Diagnostics-Interaction-Description as an early form of Diagnostic-Mappings-for-Adaptive-SW to be finalized at Software-Cluster-Integration)		
Kind	ARXML		
Extended By	SW to Diagnostics Interaction Description		
Extends	Application Design Description, Diagnostic Design		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Provide DEXT for Application Set-up	1	
Consumed by	Define Diagnostic Contribution Description	1	

Table 3.46: DEXT

3.5.2.3 Diagnostic Contribution Description

Artifact	**Diagnostic Contribution Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Diagnostic Design		
Brief Description	Collection of DEXT-elements applicable for a specific Software Cluster		
Description	This is a collection of references to DEXT-elements in scope of the Diagnostic Manager of a specific Software Cluster		
Kind	ARXML		
Extends	Application Design Description, Diagnostic Design		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Define Diagnostic Contribution Description	1	

Table 3.47: Diagnostic Contribution Description

3.5.2.4 Diagnostic Interface Description

Artifact	**Diagnostic Interface Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Diagnostic Design		
Brief Description	Description of diagnostic interfaces		
Description	Specify diagnostic interfaces of kind • DiagnosticConditionInterface • DiagnosticDataIdentifierInterface. • DiagnosticDTCInformationInterface. • DiagnosticEventInterface • DiagnosticIndicatorInterface • DiagnosticMonitorInterface • DiagnosticOperationCycleInterface • DiagnosticSecurityLevelInterface • DiagnosticServiceValidationInterface to be used in the diagnostic ports of adaptive (application) software.		
Kind	ARXML		
Extends	Diagnostic Design		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Define Diagnostic Interface Description	1	

Table 3.48: Diagnostic Interface Description

3.5.2.5 SW to Diagnostics Interaction Description

Artifact	**SW to Diagnostics Interaction Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Diagnostic Design		
Brief Description	Describe how Executables access diagnostic ports of adaptive (application) software		
Description	This considers diagnostic ports in (adaptive) Executables (i.e. ports instantiating diagnostic interfaces). The complete port instance context consists of • the port instance inside the SW composition of the Executable • the Process Design representing the Executable instance		
Kind	ARXML		
Extends	Application Design Description, DEXT, Diagnostic Mappings for Adaptive SW, SW Interaction Description		
Relation Type	***Related Element***	***Mult.***	***Note***

Table 3.49: SW to Diagnostics Interaction Description

3.6 Machine Design

This chapter contains the definition of tasks and work products related to Machine Design (see chapter 2.7).

3.6.1 Tasks

3.6.1.1 Define Machine Design

Task Definition	**Define Machine Design**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Machine Design		
Brief Description	Description of a design time proxy for a Machine		
Description	Define and configure the network communication of a prospective machine • consider the network connections • Configure the service discovery See also [TR_AMETH_00021]		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Network Endpoint Description	1	
In/out	Machine Design	1	

Table 3.50: Define Machine Design

3.6.2 Work Products

3.6.2.1 Machine Design

Deliverable	**Machine Design**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Machine Design		
Brief Description	Description of a design time proxy for a Machine		
Description	Define and configure the network communication of a prospective machine • consider the network connections • Configure the service discovery See also [TR_AMETH_00021]		
Kind	ARXML		
Extends	Machine Manifest, System Topology Description		
Relation Type	***Related Element***	***Mult.***	***Note***
In/out	Define Machine Design	1	
Consumed by	Associate content elements	1	
Consumed by	Define Machine Manifest	1	
Consumed by	Map provided and required service instances to contained executables	1	

Table 3.51: Machine Design

3.6.2.2 ECU Resource Description

Deliverable	**ECU Resource Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Machine Design		
Brief Description	Definition of the resources available on an ECU-HW.		
Description	Definition of the resources available on an ECU-HW. It mainly contains a description of hardware elements/entities (like physical memory sections or peripherals, pins, hardware connections). 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. The ECU Resource Description is a deliverable of the ECU-HW manufacturer/provider.		
Kind	ARXML		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumed by	Define Machine Manifest	1	

Table 3.52: ECU Resource Description

3.7 Machine Manifest

This chapter contains the definition of tasks and work products related to Machine Manifest (see chapter 2.8).

3.7.1 Tasks

3.7.1.1 Define Machine Manifest

Task Definition	**Define Machine Manifest**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Machine Manifest		
Brief Description	Create a Machine Manifest		
Description	Describe the configuration content of a Machine, which do not depend on deployment information of applications or service instances. This considers the configuration for the communication from Machine Design, links to ECU resources and adds platform module configurations. See also [TR_AMETH_00214], [TR_AMETH_00215], [TR_AMETH_00217], [TR_AMETH_00019], [TR_AMETH_00023]		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	ECU Resource Description	1	
Consumes	Machine Design	1	
In/out	Machine Manifest	1	
Produces	Machine Description	1	

Table 3.53: Define Machine Manifest

3.7.2 Work Products

3.7.2.1 Machine Manifest

Deliverable	**Machine Manifest**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Machine Manifest		
Brief Description	Machine Manifest		
Description	The Machine Manifest considers the configuration for the communication from Machine Design, links to ECU resources and adds platform module configurations. See also [TR_AMETH_00214], [TR_AMETH_00215], [TR_AMETH_00217], [TR_AMETH_00019], [TR_AMETH_00023]		
Kind	ARXML		
Extended By	Machine Design, Machine Description, Platform Module Configuration		
Relation Type	***Related Element***	***Mult.***	***Note***
In/out	Define Machine Manifest	1	
Consumed by	Define Execution Manifest	1	

Table 3.54: Machine Manifest

3.7.2.2 Machine Description

Artifact	**Machine Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Machine Manifest		
Brief Description	Machine Description		
Description	Description of deployment content for the configuration of the machine, independent of any service instances or applications.		
Kind	ARXML		
Extends	Machine Manifest		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Define Machine Manifest	1	

Table 3.55: Machine Description

3.8 Service Interface Design

This chapter contains the definition of tasks and work products related to Service Interface Design) (see chapter 2.3).

3.8.1 Tasks

3.8.1.1 Aggregatie Service Interfaces (for reducing the bus load)

Task Definition	**Aggregate Service Interfaces for reducing the bus load**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Service Interface Design		
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***	***Mult.***	***Note***
In/out	Service Interface Description	1	
Produces	Service Interface Mapping	1	

Table 3.56: Aggregate Service Interfaces for reducing the bus load

3.8.1.2 Define Data Types (for the Adaptive Platform)

Task Definition	**Define Datatypes for the Adaptive Platform**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Service Interface Design		
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 (AP) 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 of application data types, (C++) implementation data types and the necessary data type mapping sets.		
Relation Type	***Related Element***	***Mult.***	***Note***
In/out	Datatypes for the Adaptive Platform	1	

Table 3.57: Define Datatypes for the Adaptive Platform

3.8.1.3 Define Service Interface

Task Definition	**Define Service Interface**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Service Interface Design		
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***	***Mult.***	***Note***
Consumes	Datatypes for the Adaptive Platform	1	
In/out	Service Interface Description	1	

Table 3.58: Define Service Interface

3.8.2 Work Products

3.8.2.1 Datatypes for the Adaptive Platform

Deliverable	**Datatypes for the Adaptive Platform**
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Service Interface Design
Brief Description	Definition of data types for the Adaptive Platform
Description	Data types, which are required for the Adaptive Platform (AP) Instance. Some of these may be defined as platform data types by AUTOSAR. The AP Data Types are used for specifying data elements in service interfaces.
Kind	ARXML

▽

△

Deliverable	**Datatypes for the Adaptive Platform**		
Relation Type	***Related Element***	***Mult.***	***Note***
In/out	Define Datatypes for the Adaptive Platform	1	
Consumed by	Define Service Interface	1	

Table 3.59: Datatypes for the Adaptive Platform

3.8.2.2 Service Interface Description

Deliverable	**Service Interface Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Service Interface Design		
Brief Description	Definition of adaptive service interfaces with events, methods and fields.		
Description	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	ARXML		
Extended By	Service Interface Mapping		
Relation Type	***Related Element***	***Mult.***	***Note***
In/out	Aggregate Service Interfaces for reducing the bus load	1	
In/out	Define Service Interface	1	
Consumed by	Define SW Component Design	1	
Consumed by	Define Signal to Service Translation	1	
Consumed by	Define provided and required service instances	1	

Table 3.60: Service Interface Description

3.8.2.3 Service Interface (Element) Mapping

Artifact	**Service Interface Mapping**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::Service Interface Design		
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	ARXML		
Extends	Service Interface Description		
Relation Type	***Related Element***	***Mult.***	***Note***

▽

△

Artifact	**Service Interface Mapping**		
Produced by	Aggregate Service Interfaces for reducing the bus load	1	

Table 3.61: Service Interface Mapping

3.9 Software Cluster Design

This chapter contains the definition of tasks and work products related to Software Cluster Design (see chapter 2.9).

3.9.1 Tasks

3.9.1.1 Define provided and required service instances

Task Definition	**Define provided and required service instances**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Design		
Brief Description	Define the service instances and configure their search or offer criteria		
Description	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, • there can be required service instances (client) in case a service is required. The configuration of service instances includes • define the transport layer (e.g. SOME/IP) and configure the binding of a service interface to this transport layer (for the service interface and its elements, i.e. events, methods and fields). • define the transport layer specific service identification and configure the service discovery, including – search criteria for required service instances (for SOME/IP, the required service instance IDs and required service interface version needs to be defined; also, required event groups can/ shall be specified). – offer criteria for provided service instances (for SOME/IP, the provided service instance IDs need to defined). The service deployment and instance IDs need to be defined system-wide unambiguously. In consequence the service deployment and instance configuration may be prepared during software cluster design and finalized during software cluster integration.		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Service Interface Description	1	
Consumes	Service Topology Description	1	
Consumes	System Topology Description	1	

▽

△

Task Definition	**Define provided and required service instances**		
Produces	Service Deployment Description	1	
Produces	Service Instance Description	1	

Table 3.62: Define provided and required service instances

3.9.1.2 Map provided and required service instances to contained executables

Task Definition	**Map provided and required service instances to contained executables**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Design		
Brief Description	Define mapping of service instance to a port in executable instance context		
Description	This mapping is needed in order to ensure an unambiguous 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). For software cluster design this can be prepared by associating the service instance to a delegation port of the software cluster's Black box of contained SW. Only when the application design is available this can be finnalized with a mapping of service instance to the corresponding application port in process (design) context.		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Application Design Description	1	
Consumes	Black box of contained SW	1	
Consumes	Machine Design	1	
Consumes	Service Instance Description	1	
Produces	Service instance mapping	1	

Table 3.63: Map provided and required service instances to contained executables

3.9.1.3 Outline SW Cluster Design

Task Definition	**Outline SW Cluster Design**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Design		
Brief Description	Outline SW Cluster Design		
Description	Outline SW Cluster Design covers the basic set-up of a SW Cluster Design Description: • Specify the SW Cluster Design Description container itself. • Specify the Black box of contained SW identifying the communication endpoints in SW Cluster Design Description likely before any Application Design activities are started. • Dependencies between SW Cluster Design Description allow to describe in early design phase what needs to be installed together. • optionally Associate Diagnostic Address and Contribution • optionally Associate content elements currently known. The detailed content may be added in random order at a later point of time as per individual design work flow.		
Relation Type	***Related Element***	***Mult.***	***Note***
Produces	Black box of contained SW	1	
Produces	SW Cluster Design Description	1	

Table 3.64: Outline SW Cluster Design

3.9.1.4 Associate content elements

Task Definition	**Associate content elements**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Design		
Brief Description	Associate content elements to a software cluster		
Description	One of the most prominent contents of a SW Cluster Design Description is the reference to the executable software via Process Design Descriptions. An important aspect of a SW Cluster Design Description is the question what diagnostic extract shall be applied (see task Associate Diagnostic Address and Contribution). Besides that Associate content elements takes care that all required uploadable elements in scope of SW Cluster Design Description (like Machine Design, Process Design Description etc.) are registered.		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Application Design Description	1	
Consumes	Machine Design	1	
Consumes	Process Design Description	1	
Produces	Associated uploadable elements	1	

Table 3.65: Associate content elements

3.9.1.5 Associate Diagnostic Address and Contribution

Task Definition	**Associate Diagnostic Address and Contribution**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Design		
Brief Description	Associate Diagnostic Address and Contribution		
Description	An important aspect of a SW Cluster Design Description is the question what diagnostic extract and what diagnostic address shall be applied. See task Associate content elements for other items associated to the software cluster.		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Diagnostic Design	1	
Produces	SW Cluster Design Description	1	

Table 3.66: Associate Diagnostic Address and Contribution

3.9.2 Work Products

3.9.2.1 SW Cluster Design Description

Deliverable	**SW Cluster Design Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Design		
Brief Description	Software Cluster Design Description		
Description	The software cluster was identfied during system-design as a building block in the software topology of a sub-system. This may cover one or more application-level and/or platform-level software entities. The software entities that can be deployed in the field typically represent some sort of more or less self-contained driving function. We use SW Cluster Design Description for the design of software that might represent such a driving function. Please note that SW Cluster Design Description supports an arbitrary complexity of software and may therefore cover multiple driving functions. The SW Cluster Design Description covers various design aspects of a deployable software entity that may be specified in arbitrary work flows.		
Kind	ARXML		
Extended By	Associated uploadable elements, Black box of contained SW, Service instance mapping		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Associate Diagnostic Address and Contribution	1	
Produced by	Identify Software Cluster	1	
Produced by	Outline SW Cluster Design	1	

Table 3.67: SW Cluster Design Description

3.9.2.2 Associated uploadable elements

Artifact	**Associated uploadable elements**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Design		
Brief Description	Content elements associated to a software cluster		
Description	One of the most prominent contents of a SW Cluster Design Description is the reference to the executable software via Process Design Descriptions. An important aspect of a SW Cluster Design Description is the question what diagnostic extract shall be applied (see task Associate Diagnostic Address and Contribution). Besides that Associate content elements takes care that all required uploadable elements in scope of SW Cluster Design Description (like Machine Design, Process Design Description etc.) are registered.		
Kind	ARXML		
Extends	SW Cluster Design Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Associate content elements	1	

Table 3.68: Associated uploadable elements

3.9.2.3 Black box of contained SW

Artifact	**Black box of contained SW**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Design		
Brief Description	Black box of the SW cluster's ports exposed to the outside world.		
Description	Black box of contained SW outlines the communication endpoints as delegation ports (representing the exposed ports of the enclosed software of SW Cluster Design Description) on an early design level - typically before the actual Application Design Description is available.		
Kind	ARXML		
Extends	SW Cluster Design Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Outline SW Cluster Design	1	
Consumed by	Map provided and required service instances to contained executables	1	

Table 3.69: Black box of contained SW

3.9.2.4 Service Deployment Description

Artifact	**Service Deployment Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Design		
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. See also [TR_AMETH_00027]		
Kind	ARXML		
Extends	Service Instance Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Define provided and required service instances	1	

Table 3.70: Service Deployment Description

3.9.2.5 Service Instance Description

Deliverable	**Service Instance Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Design		
Brief Description	Definition and configuration of the service instances		
Description	Required as well as provided service instances are defined and configured as per transport layer. For the configuration, the search criteria for required service instances and offer criteria for provided service instances are specified. See also [TR_AMETH_00005]		
Kind	ARXML		
Extended By	Service Deployment Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Define provided and required service instances	1	
Consumed by	Create or Finalize Service Instance Manifest	1	
Consumed by	Map provided and required service instances to contained executables	1	

Table 3.71: Service Instance Description

3.9.2.6 Service Instance Mapping

Artifact	**Service instance mapping**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Design		
Brief Description	Binding of service instances to ports in instance context		
Description	This mapping is needed in order to ensure an unambiguous 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). For software cluster design this can be prepared by associating the service instance to a delegation port of the software cluster's Black box of contained SW. Only when the application design is available this can be finnalized with a mapping of service instance to the corresponding application port in process (design) context.		
Kind	ARXML		
Extends	SW Cluster Design Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Map provided and required service instances to contained executables	1	
Consumed by	Create or Finalize Service Instance Manifest	1	

Table 3.72: Service instance mapping

3.10 Software Cluster Integration

This chapter contains the definition of tasks and work products related to method content package Software Cluster Integration (see chapter 2.10).

3.10.1 Tasks

3.10.1.1 Build SW for test environment

Task Definition	**Build SW for test environment**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration		
Brief Description	Build software to run in a test environment		
Description	During development, it can be useful to run the software in a test environment. This environment will typically differ from the target environment, which allows testing at an earlier stage, but might require additional pieces of Adaptive Software Glue Code.		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Adaptive Software Implementation	1	
In/out	Adaptive Software Glue Code	1	
Produces	Adaptive Software Binary	1	

△

Task Definition	**Build SW for test environment**		
Produces	Diagnostic Manager Binary	1	

Table 3.73: Build SW for test environment

3.10.1.2 Define Execution Manifest

Task Definition	**Define Execution Manifest**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration		
Brief Description	Provide information that is needed to deploy an application onto the AUTOSAR adaptive platform		
Description	Associate Process with Process Design: Create the reference from an actual Process to its Process Design placeholder, which was used during the design phase. • Define the instantiation of executables. An executable can be instantiated several times (e.g. with different startup parameters) resulting in different processes. • Define Function Groups as state-machines for a set of cohesive Executable instances (i.e. Processes) likely having run-time interdependencies. • Define Execution Dependencies to function group states, for platform-level software to Machine FG and between processes - e.g. the currently considered process shall start/stop depending on the state of another process (to enforce a start-up or shut-down sequence) • Define Startup Configuration: Besides execution dependencies this considers program arguments, environment variable settings etc. • Create securityEvent references to report a SecurityEvent from a Process		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Machine Manifest	1	
Consumes	Process Design Description	1	
Produces	Execution Manifest	1	

Table 3.74: Define Execution Manifest

3.10.1.3 Configure Platform Modules

Task Definition	**Configure Platform Modules**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration		
Brief Description	Configure the platform modules of the adaptive stack		
Description	• Configure the operating system, e.g. the resource groups and the timer granularity can be defined. • Define the Machine-specific configuration settings for the platform modules mentioned in the chapter Create Platform Module Configuration in the adaptive methodology document		
Relation Type	***Related Element***	***Mult.***	***Note***
Produces	Platform Module Configuration	1	

Table 3.75: Configure Platform Modules

3.10.1.4 Create or Finalize Service Instance Manifest

Task Definition	**Create or Finalize Service Instance Manifest**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration		
Brief Description	Complete the service instance manifest		
Description	The service deployment and instance IDs need to be defined system-wide unambiguously. Therefore, the service deployment and instance configuration may be prepared during software cluster design, and must be finalized during software cluster integration.		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Service Instance Description	1	
Consumes	Service instance mapping	1	
Produces	Service Instance Manifest	1	

Table 3.76: Create or Finalize Service Instance Manifest

3.10.1.5 Integrate Diagnostics

Task Definition	**Integrate Diagnostics**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration		
Brief Description	Associate Diagnostic Mapping with Process Design		
Description	It may be necessary that different instances of a particular application software require different diagnostic mappings. Therefore, a relation between a particular diagnostic mapping and a particular Process (Design) needs to be established. These mappings may be prepared during Diagnostic Design activities and must be finalized at Integrate Diagnostics, considering the tasks to design a diagnostic mapping defined in [TR_AMETH_00212] . See also the adaptive methodology document, chapter Diagnostic Design.		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Diagnostic Design	1	
Produces	Diagnostic Mappings for Adaptive SW	1	

Table 3.77: Integrate Diagnostics

3.10.2 Work Products

3.10.2.1 Software Cluster Description

Deliverable	**Software Cluster Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration		
Brief Description	The Description for a SoftwareCluster made up of manifests and executables		
Description	A SoftwareCluster may contain Executable(s), Execution Manifest(s), Service Instance Manifest(s), Machine Manifest(s), Diagnostic Mapping(s) and other development artifacts. The term SoftwareCluster refers to a set of installed software entities (manifests, data and processes that run executables), which form a logical group and which are addressable by the diagnostic management by a shared diagnostic address. From an UCM (Update and Configuration Management) point of view, the term SoftwareCluster identifies a bundle of software artifacts that are uploaded together in order to be installed by the UCM. It should be mentioned, that a SoftwareCluster may be structured into sub-blocks in order to mimic the CP diagnostic workflow, where blocks are the smallest parts of update and to enable the execution of update campaigns. Both definitions match in the sense that the bundle of software uploaded are needed to form the set of installed software entities addressed by the same diagnostic address.		
Kind			
Extended By	Diagnostic Mappings for Adaptive SW, Execution Manifest, Platform Module Configuration, Service Instance Manifest		
Relation Type	***Related Element***	***Mult.***	***Note***

Table 3.78: Software Cluster Description

3.10.2.2 Adaptive Software Binary

Artifact	**Adaptive Software Binary**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Packaging		
Brief Description	Executable containing one or more software components		
Description	The executable can contain an arbitrary hierarchy of software components. The software components contain the functionality of the executable. Executables can be of category application-level or platform-level.		
Kind			
Extends	Adaptive Software Package		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Build SW for target runtime environment	1	
Produced by	Build SW for test environment	1	

Table 3.79: Adaptive Software Binary

3.10.2.3 Diagnostic Manager Binary

Artifact	**Diagnostic Manager Binary**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration		
Brief Description			
Description			
Kind			
Extends	Adaptive Software Package		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Build SW for target runtime environment	1	
Produced by	Build SW for test environment	1	

Table 3.80: Diagnostic Manager Binary

3.10.2.4 Diagnostic Mappings for Adaptive SW

Deliverable	**Diagnostic Mappings for Adaptive SW**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration		
Brief Description			
Description	Map parts of the diagnostic protocol to one or more service port instances of a particular executable instance. The diagnostic mappings are defined in [TR_AMETH_00212] Design a diagnostic mapping		
Kind			
Extended By	SW to Diagnostics Interaction Description		
Extends	Software Cluster Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Integrate Diagnostics	1	

Table 3.81: Diagnostic Mappings for Adaptive SW

3.10.2.5 Execution Manifest

Deliverable	**Execution Manifest**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration		
Brief Description	Contains information required to deploy and execute software on a machine		
Description	An execution manifest specifies all the properties of one process that executes on a specific machine. The execution manifest references the modes of the process in the startup configuration Process: The process is the top-level element of the Execution Manifest and references an executable. It is the unit of deployment on the AUTOSAR adaptive platform and refers to a POSIX process. Mode-dependent Startup Configuration: Startup configuration for one process and depending on the machine mode. Define Execution Dependencies: 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.		
Kind			
Extended By	Function Group Configuration, Process Configuration		
Extends	Software Cluster Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Define Execution Manifest	1	

Table 3.82: Execution Manifest

3.10.2.6 Service Instance Manifest

Deliverable	**Service Instance Manifest**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration		
Brief Description	Definition and configuration of the service instances		
Description	Required as well as provided service instances are defined and configured as per transport layer. For the configuration, the search criteria for required service instances and offer criteria for provided service instances are specified.		
Kind			
Extends	Software Cluster Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Create or Finalize Service Instance Manifest	1	

Table 3.83: Service Instance Manifest

3.10.2.7 Function Group Configuration

Artifact	**Function Group Configuration**
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration
Brief Description	
Description	The configuration of Function Groups

▽

△

Artifact	**Function Group Configuration**		
Kind			
Extends	Application Design Description, Execution Manifest		
Relation Type	***Related Element***	***Mult.***	***Note***

Table 3.84: Function Group Configuration

3.10.2.8 Adaptive Software Glue Code

Artifact	**Adaptive Software Glue Code**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration		
Brief Description			
Description			
Kind			
Relation Type	***Related Element***	***Mult.***	***Note***
In/out	Build SW for test environment	1	

Table 3.85: Adaptive Software Glue Code

3.10.2.9 Platform Module Configuration

Artifact	**Platform Module Configuration**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration		
Brief Description			
Description	Includes • Configuration of the operating system, e.g. the resource groups and the timer granularity can be defined. • Machine-specific configuration settings for the Log and Trace functional cluster. • Machine-specific configuration settings for DoIP. • Machine-specific configuration settings for the NM module.		
Kind			
Extends	Application Design Description, Machine Manifest, Software Cluster Description		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Configure Platform Modules	1	

Table 3.86: Platform Module Configuration

3.10.2.10 Process Configuration

Artifact	**Process Configuration**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Cluster Integration		
Brief Description			
Description	The process is the top-level element of the Execution Manifest and references an executable. It is the unit of deployment on the AUTOSAR adaptive platform and refers to a POSIX process. This includes Process-to-Machine-Mapping		
Kind			
Extends	Execution Manifest		
Relation Type	***Related Element***	***Mult.***	***Note***

Table 3.87: Process Configuration

3.11 Software Packaging

This chapter contains the definition of tasks and work products related to method content package Software Distribution (see chapter 2.11).

3.11.1 Tasks

3.11.1.1 Build SW for target runtime environment

Task Definition	**Build SW for target runtime environment**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Packaging		
Brief Description			
Description	The software components are linked together with the serialization code and necessary middleware libraries. Together with the main function, the executable is built.		
Relation Type	***Related Element***	***Mult.***	***Note***
Consumes	Adaptive Software Implementation	1	
Produces	Adaptive Software Binary	1	
Produces	Diagnostic Manager Binary	1	

Table 3.88: Build SW for target runtime environment

3.11.1.2 Define SW Package

Task Definition	**Define SW Package**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Packaging		
Brief Description			
Description	Software is deployed by means of a Software Package (see [TR_AMETH_00006]) See also chapter Define Software Package in the adaptive methodology document, especially [TR_AMETH_00206] Create Adaptive Software Package, [TR_AMETH_00218] Create an initial SW Package Description and [TR_AMETH_00219] Collect all software artifacts that belong to a SoftwareCluster, structure and model them.		
Relation Type	***Related Element***	***Mult.***	***Note***
Produces	SW Package Description	1	

Table 3.89: Define SW Package

3.11.1.3 Specify Update Campaign

Task Definition	**Specify update campaign**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Packaging		
Brief Description			
Description	See chapter Specify update campaign in the adaptive methodology document, especially [TR_AMETH_00220] Model dependencies between SoftwareClusters of any category and [TR_AMETH_00221] Develop installation Instructions.		
Relation Type	***Related Element***	***Mult.***	***Note***
Produces	Update Campaign Description	1	

Table 3.90: Specify update campaign

3.11.2 Work Products

3.11.2.1 Adaptive Software Package

Deliverable	**Adaptive Software Package**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Packaging		
Brief Description			
Description			
Kind			
Extended By	Adaptive Software Binary, Diagnostic Manager Binary, SW Package Description		
Relation Type	***Related Element***	***Mult.***	***Note***

Table 3.91: Adaptive Software Package

3.11.2.2 SW Package Description

Artifact	**SW Package Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Packaging		
Brief Description			
Description	Container to deploy software artifacts to a machine, consisting of one or more Software Clusters (including their Software Cluster Manifests) and the Software Package Manifest. See chapter Software Packaging in the adaptive methodology document.		
Kind			
Extends	Adaptive Software Package		
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Define SW Package	1	

Table 3.92: SW Package Description

3.11.2.3 Update Campaign Description

Deliverable	**Update Campaign Description**		
Package	AUTOSAR Root::M2::Methodology::AdaptiveMethodology::SW Packaging		
Brief Description			
Description	An update campaign is a coordinated set of changes (add, update, remove) to Adaptive Software Packages for a whole vehicle. See chapter Vehicle Package in TPS ManifestSpecification		
Kind			
Relation Type	***Related Element***	***Mult.***	***Note***
Produced by	Specify update campaign	1	

Table 3.93: Update Campaign Description

A Mentioned Class Tables

For the sake of completeness, this chapter contains a set of class tables representing meta-classes mentioned in the context of this document but which are not contained directly in the scope of describing specific meta-model semantics.

Class	**Executable**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::ApplicationStructure			
Note	This meta-class represents an executable program. **Tags:** atp.recommendedPackage=Executables			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
buildType	BuildTypeEnum	0..1	attr	This attribute describes the buildType of a module and/or platform implementation.
implementation Props	Executable ImplementationProps	*	aggr	This aggregation contains the collection of implementation-specific properties necessary to properly build the enclosing Executable.
minimumTimer Granularity	TimeValue	0..1	attr	This attribute describes the minimum timer resolution (TimeValue of one tick) that is required by the Executable.
reporting Behavior	ExecutionState ReportingBehavior Enum	0..1	attr	this attribute controls the execution state reporting behavior of the enclosing Executable.
rootSw Component Prototype	RootSwComponent Prototype	0..1	aggr	This represents the root SwCompositionPrototype of the Executable. This aggregation is required (in contrast to a direct reference of a SwComponentType) in order to support the definition of instanceRefs in Executable context.
traceSwitch Configuration	TraceSwitch Configuration	*	aggr	Configuration of the MsgId based trace switch **Tags:** atp.Status=draft
version	StrongRevisionLabel String	0..1	attr	Version of the executable.

Table A.1: Executable

Class	**Machine**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SubSystemDesign::MachineManifest			
Note	Machine that represents an Adaptive Autosar Software Stack. **Tags:** atp.recommendedPackage=Machines			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeployment Element*, *UploadablePackageElement*			
Aggregated by	ARPackage.element, *AtpClassifier*.atpFeature			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
default Application Timeout	EnterExitTimeout	0..1	aggr	This aggregation defines a default timeout in the context of a given Machine with respect to the launching and termination of applications.

△

Class	**Machine**			
environment Variable	TagWithOptionalValue	*	aggr	This aggregation represents the collection of environment variables that shall be added to the environment defined on the level of the enclosing Machine. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=environmentVariable
machineDesign	MachineDesign	0..1	ref	Reference to the MachineDesign this Machine is implementing.
module Instantiation	AdaptiveModule Instantiation	*	aggr	Configuration of Adaptive Autosar module instances that are running on the machine. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=moduleInstantiation.shortName
processor	Processor	*	aggr	This represents the collection of processors owned by the enclosing machine.
secure Communication Deployment	SecureCommunication Deployment	*	aggr	Target-configuration of secure communication protocol configuration settings to crypto module entities. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=secureCommunication Deployment.shortName
trustedPlatform Executable LaunchBehavior	TrustedPlatform ExecutableLaunch BehaviorEnum	0..1	attr	This attribute controls the behavior of how authentication affects the ability to launch for each Executable.

Table A.2: Machine

Class	***PortInterface*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Abstract base class for an interface that is either provided or required by a port of a software component.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*AbstractRawDataStreamInterface*, *AbstractSynchronizedTimeBaseInterface*, ClientServerInterface, *CryptoInterface*, *DataInterface*, *DiagnosticPortInterface*, FirewallStateSwitchInterface, *IdsmAbstractPort Interface*, LogAndTraceInterface, ModeSwitchInterface, NetworkManagementPortInterface, *Persistency Interface*, *PlatformHealthManagementInterface*, ServiceInterface, *StateManagementPortInterface*, TriggerInterface			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
namespace (ordered)	SymbolProps	*	aggr	This represents the SymbolProps used for the definition of a hierarchical namespace applicable for the generation of code artifacts out of the definition of a ServiceInterface. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=namespace.shortName

Table A.3: PortInterface

Class	**Process**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ExecutionManifest
Note	This meta-class provides information required to execute the referenced `Executable`. **Tags:** atp.recommendedPackage=Processes

▽

△

Class	**Process**			
Base	*ARElement*, *ARObject*, *AbstractExecutionContext*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *Uploadable PackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
design	ProcessDesign	0..1	ref	This reference represents the identification of the design-time representation for the Process that owns the reference.
executable	Executable	*	ref	Reference to executable that is executed in the process. **Stereotypes:** atpUriDef
functionCluster Affiliation	String	0..1	attr	This attribute specifies which functional cluster the Process is affiliated with.
numberOf RestartAttempts	PositiveInteger	0..1	attr	This attribute defines how often a process shall be restarted if the start fails. numberOfRestartAttempts = "0" OR Attribute not existing, start once numberOfRestartAttempts = "1", start a second time
preMapping	Boolean	0..1	attr	This attribute describes whether the executable is preloaded into the memory.
processState Machine	ModeDeclarationGroup Prototype	0..1	aggr	Set of Process States that are defined for the process. This attribute is used to support the modeling of execution dependencies that utilize the condition of process state. Please note that the process states may not be modeled arbitrarily at any stage of the AUTOSAR workflow because the supported states are standardized in the context of the SWS Execution Management [13].
stateDependent StartupConfig	StateDependentStartup Config	*	aggr	Applicable startup configurations.

Table A.4: Process

Class	**SoftwareCluster**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
Note	This meta-class represents the ability to define an uploadable software-package, i.e. the SoftwareCluster shall contain all software and configuration for a given purpose. **Tags:** atp.recommendedPackage=SoftwareClusters			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
artifact Checksum	ArtifactChecksum	*	aggr	This aggregation carries the checksums for artifacts contained in the enclosing SoftwareCluster. Please note that the value of these checksums is only applicable at the time of configuration. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=artifactChecksum.shortName, artifact Checksum.uri
artifactLocator	ArtifactLocator	*	aggr	This aggregation represents the artifact locations that are relevant in the context of the enclosing SoftwareCluster
claimed FunctionGroup	ModeDeclarationGroup Prototype	*	ref	Each SoftwareCluster can reserve the usage of a given functionGroup such that no other SoftwareCluster is allowed to use it

▽

△

Class	**SoftwareCluster**			
conflictsTo	SoftwareCluster DependencyFormula	0..1	aggr	This aggregation handles conflicts. If it yields true then the SoftwareCluster shall not be installed. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=conflictsTo
contained ARElement	ARElement	*	ref	This reference represents the collection of model elements that cannot derive from UploadablePackage Element and that contribute to the completeness of the definition of the SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=containedARElement
containedFibex Element	FibexElement	*	ref	This allows for referencing FibexElements that need to be considered in the context of a SoftwareCluster.
contained Package Element	UploadablePackage Element	*	ref	This reference identifies model elements that are required to complete the manifest content. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=containedPackageElement
contained Process	Process	*	ref	This reference represent the processes contained in the enclosing SoftwareCluster.
dependsOn	SoftwareCluster DependencyFormula	0..1	aggr	This aggregation can be taken to identify a dependency for the enclosing SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=dependsOn
design	SoftwareClusterDesign	*	ref	This reference represents the identification of all Software ClusterDesigns applicable for the enclosing Software Cluster. **Stereotypes:** atpUriDef
diagnostic Deployment Props	SoftwareCluster DiagnosticDeployment Props	0..1	ref	This reference identifies the applicable SoftwareCluster DiagnosticDeploymentProps that are applicable for the referencing SoftwareCluster.
installation Behavior	SoftwareCluster InstallationBehavior Enum	0..1	attr	This attribute controls the behavior of the SoftwareCluster in terms of installation.
license	Documentation	*	ref	This attribute allows for the inclusion of the full text of a license of the enclosing SoftwareCluster. In many cases open source licenses require the inclusion of the full license text to any software that is released under the respective license.
module Instantiation	AdaptiveModule Instantiation	*	ref	This reference identifies AdaptiveModuleInstantiations that need to be included with the SoftwareCluster in order to establish infrastructure required for the installation of the SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=moduleInstantiation
releaseNotes	Documentation	0..1	ref	This attribute allows for the explanations of changes since the previous version. The list of changes might require the creation of multiple paragraphs of test.
typeApproval	String	0..1	attr	This attribute carries the homologation information that may be specific for a given country.
vendorId	PositiveInteger	0..1	attr	Vendor ID of this Implementation according to the AUTOSAR vendor list.
vendor Signature	CryptoService Certificate	0..1	ref	This reference identifies the certificate that represents the vendor's signature.
version	StrongRevisionLabel String	0..1	attr	This attribute can be used to describe a version information for the enclosing SoftwareCluster.

Table A.5: SoftwareCluster

B Change History

B.1 Change History of this document according to AUTOSAR Release R17-03

B.1.1 Added Specification Items in 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 `Adaptive Software Build Configuration`
[TR_AMETH_00015]	Development without knowledge of the `Adaptive Software Build Configuration`
[TR_AMETH_00016]	Development of serialization properties
[TR_AMETH_00017]	Implementation of service proxies and skeletons
[TR_AMETH_00018]	Building the `(Adaptive) Executable`
[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 `(Adaptive) Executable`
[TR_AMETH_00025]	Definition of startup behavior of a process
[TR_AMETH_00026]	Definition 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

B.2 Change History of this document according to AUTOSAR Release R17-10

B.2.1 Added Specification Items in 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

B.2.2 Changed Specification Items in 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

B.2.3 Deleted Specification Items in 17-10

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

B.3 Change History of this document according to AUTOSAR Release R18-03

B.3.1 Added Specification Items in 18-03

Number	Heading
[TR_AMETH_00211]	Pool Executables together to form ExecutableGroups
[TR_AMETH_00212]	Design a diagnostic mapping
[TR_AMETH_00213]	Relate diagnostic mappings to instances of Executables
[TR_AMETH_00214]	Configuration of Platform Services

[TR_AMETH_00215]	Configuration of Platform Foundation Modules
[TR_AMETH_00216]	Map Processes to a particular machine
[TR_AMETH_00217]	Definition of resources
[TR_AMETH_00218]	Create an initial `Software Package Manifest`
[TR_AMETH_00219]	Collect all software artifacts that belong to a `Software Cluster`, structure and model them
[TR_AMETH_00220]	Model dependencies between `Software Cluster`s of any category
[TR_AMETH_00221]	Develop installation instructions
[TR_AMETH_00222]	Create the `Software Package`
[TR_AMETH_00223]	Manage the data base of `Software Cluster`s (of any category)
[TR_AMETH_00224]	Management of `Software Package`s
[TR_AMETH_00225]	Provision of `Software Package`s for machines in the field
[TR_AMETH_00226]	Documentation of work products

B.3.2 Changed Specification Items in 18-03

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

B.3.3 Deleted Specification Items in 18-03

Number	Heading
[TR_AMETH_00032]	Deploying the Software Package

B.4 Change History of this document according to AUTOSAR Release R18-10

B.4.1 Added Specification Items in 18-10

none

B.4.2 Changed Specification Items in 18-10

Number	Heading
[TR_AMETH_00004]	Creation of the `Execution Manifest`
[TR_AMETH_00020]	Development of `Platform Object Code`
[TR_AMETH_00026]	Definition of `Execution Manifest`

▽

△

Number	Heading
[TR_AMETH_00031]	Setting up an initial machine
[TR_AMETH_00034]	Select the Operating System for Adaptive Platform

Table B.1: Changed Specification Items in 18-10

B.4.3 Deleted Specification Items in 18-10

Number	Heading
[TR_AMETH_00211]	Pool Executables together to form ExecutableGroups

Table B.2: Deleted Specification Items in 18-10

B.5 Change History of this document according to AUTOSAR Release R19-03

B.5.1 Added Specification Items in 19-03

none

B.5.2 Changed Specification Items in 19-03

none

B.5.3 Deleted Specification Items in 19-03

none

B.6 Change History of this document according to AUTOSAR Release R19-11

B.6.1 Added Specification Items in 19-11

none

B.6.2 Changed Specification Items in 19-11

Number	Heading
[TR_AMETH_00001]	Disentangle service interface handling
[TR_AMETH_00002] [TR_AMETH_00003] [TR_AMETH_00004] [TR_AMETH_00006]	editorial changes, fix tech-term references
[TR_AMETH_00008]	Disentangle service interface handling
[TR_AMETH_00018] [TR_AMETH_00021]	editorial changes, fix tech-term references
[TR_AMETH_00022]	remove machine state (which was replaced by function group states)
[TR_AMETH_00024]	editorial changes, fix tech-term references
[TR_AMETH_00025]	remove machine state (which was replaced by function group states)
[TR_AMETH_00034] [TR_AMETH_00201] [TR_AMETH_00202] [TR_AMETH_00204]	editorial changes, fix tech-term references
[TR_AMETH_00205]	editorial changes, replace outdated term 'executable group' by 'adaptive executable'.
[TR_AMETH_00207] [TR_AMETH_00208] [TR_AMETH_00209]	editorial changes, fix tech-term references
[TR_AMETH_00212]	editorial changes, consider all currently defined diagnostic mappings
[TR_AMETH_00216] [TR_AMETH_00213] [TR_AMETH_00219] [TR_AMETH_00220] [TR_AMETH_00221] [TR_AMETH_00222] [TR_AMETH_00223]	editorial changes, fix tech-term references

Table B.3: Changed Specification Items in 19-11

B.6.3 Deleted Specification Items in 19-11

none

B.7 Change History of this document according to AUTOSAR Release R20-11

B.7.1 Added Specification Items in R20-11

none

B.7.2 Changed Specification Items in R20-11

none

B.7.3 Deleted Specification Items in R20-11

Number	Heading
[TR_AMETH_00209]	Define a signal-based `Service Interface`

Table B.4: Deleted Specification Items in R20-11

B.8 Change History of this document according to AUTOSAR Release R21-11

B.8.1 Added Specification Items in R21-11

Number	Heading
[TR_AMETH_00251]	Variant handling

Table B.5: Added Specification Items in R21-11

B.8.2 Changed Specification Items in R21-11

Number	Heading
[TR_AMETH_00001]	Identify Abstract Port Interfaces
[TR_AMETH_00002]	Develop Adaptive Software
[TR_AMETH_00003]	Configuration of the Machine
[TR_AMETH_00004]	Creation of the `Execution 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]	Develop Service Interfaces for Adaptive Software
[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 interfaces
[TR_AMETH_00013]	Implementation and compilation of software components

▽

△

Number	Heading
[TR_AMETH_00014]	Development with knowledge of the Adaptive Software Build Configuration
[TR_AMETH_00015]	Development without knowledge of the Adaptive Software Build Configuration
[TR_AMETH_00016]	Development of serialization properties
[TR_AMETH_00017]	Implementation of service proxies and skeletons
[TR_AMETH_00018]	Building the (adaptive) Executable
[TR_AMETH_00019]	Description of the ECU-HW resources available for the Adaptive Platform
[TR_AMETH_00020]	Development of platform-level Adaptive Software Object Code
[TR_AMETH_00021]	Define and configure the network communication for a Machine
[TR_AMETH_00022]	Definition of Function Group states
[TR_AMETH_00023]	Configuration of the operating system
[TR_AMETH_00024]	Instantiation of an (adaptive) Executable
[TR_AMETH_00025]	Definition of the Process start-up behavior
[TR_AMETH_00026]	Definition of Execution Manifest
[TR_AMETH_00027]	Configuration of Service Interface Deployment
[TR_AMETH_00028]	Configuration of Service Instances
[TR_AMETH_00029]	Mapping of Service Instances to a Machine
[TR_AMETH_00031]	Setting up an initial machine
[TR_AMETH_00033]	Mapping of Service Instances to Ports of Adaptive Software
[TR_AMETH_00034]	Select the Operating System for Adaptive Platform
[TR_AMETH_00035]	Platform-level Software
[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 Vehicle Software Architecture
[TR_AMETH_00203]	Derive Sub-Systems
[TR_AMETH_00204]	Develop the System
[TR_AMETH_00205]	Integrate Software
[TR_AMETH_00206]	Create a Adaptive Software Package
[TR_AMETH_00207]	Design communication between Classic Platform (CP) ECU-Instances and Adaptive Platform (AP) Machines
[TR_AMETH_00208]	Design Signal to Service translation between AUTOSAR Classic Platform (CP) and Adaptive Platform (AP)
[TR_AMETH_00210]	Map signals to services
[TR_AMETH_00212]	Design a diagnostic mapping
[TR_AMETH_00213]	Relate diagnostic mappings to instances of Executables
[TR_AMETH_00214]	Configuration of Platform Services
[TR_AMETH_00215]	Configuration of Platform Foundation Modules
[TR_AMETH_00216]	Map Processes to a particular Machine

▽

△

Number	Heading
[TR_AMETH_00217]	Definition of resources
[TR_AMETH_00218]	Create an initial SW Package Description
[TR_AMETH_00219]	Collect all software artifacts that belong to a SoftwareCluster, structure and model them
[TR_AMETH_00220]	Model dependencies between SoftwareClusters of any category
[TR_AMETH_00221]	Develop installation instructions
[TR_AMETH_00222]	Create the Adaptive Software Package
[TR_AMETH_00223]	Manage the data base of Software Cluster Descriptions (of any category)
[TR_AMETH_00224]	Management of Adaptive Software Packages
[TR_AMETH_00225]	Provision of Adaptive Software Packages for machines in the field

Table B.6: Changed Specification Items in R21-11

B.8.3 Deleted Specification Items in R21-11

none

B.9 Change History of this document according to AUTOSAR Release R22-11

B.9.1 Added Specification Items in R22-11

none

B.9.2 Changed Specification Items in R22-11

none

B.9.3 Deleted Specification Items in R22-11

none

B.10 Change History of this document according to AUTOSAR Release R23-11

B.10.1 Added Specification Items in R23-11

none

B.10.2 Changed Specification Items in R23-11

Number	Heading
[TR_AMETH_00026]	Definition of `Execution Manifest`

Table B.7: Changed Specification Items in R23-11

B.10.3 Deleted Specification Items in R23-11

none

B.11 Change History of this document according to AUTOSAR Release R24-11

B.11.1 Added Specification Items in R24-11

none

B.11.2 Changed Specification Items in R24-11

none

B.11.3 Deleted Specification Items in R24-11

none