Document Title	Specification of Update and Configuration Management
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	888

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R19-11

Document Change History

Date	Release	Changed by	Description
2019-11-28	R19-11	AUTOSAR Release Management	• Introduced UCM Master concept • Software Package state machine updated for processing while streaming • Reviewed UCM State Machine • Added new security analysis appendix • Changed Document Status from Final to published
2019-03-29	19-03	AUTOSAR Release Management	• Updating Package Management state machine • New requirements for robustness against reset • Improving specification item atomicity • Fixing errors in chapter Service Interfaces
2018-10-31	18-10	AUTOSAR Release Management	• Updated interaction other functional clusters like PER and EMO/SM • Introduction of vehicle package distribution
2018-03-29	18-03	AUTOSAR Release Management	• Extended and updated service interface • Introduction of Software Package • Introduction to securing update process

2017-10-27	17-10	AUTOSAR Release Management	• Initial release

Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview

This software specification contains the functional description and interfaces of the functional cluster Update and Configuration Management which belongs to the AUTOSAR Adaptive Platform Services. Update and Configuration Management has the responsibility of installing, updating and removing software on an AUTOSAR Adaptive Platform in a safe and secure way while not sacrificing the dynamic nature of the AUTOSAR Adaptive Platform.

The Update and Configuration Management functional cluster is responsible for:

- Version reporting of the software present in the AUTOSAR Adaptive Platform
- Receiving and buffering software updates
- Checking that enough resources are available to ensure a software update
- Performing software updates and providing log messages and progress information
- Validating the outcome of a software update
- Providing rollback functionality to restore a known functional state in case of failure

In addition to updating and changing software on the AUTOSAR Adaptive Platform, the Update and Configuration Management is also responsible for updates and changes to the AUTOSAR Adaptive Platform itself, including all functional clusters, the underlying POSIX OS and its kernel with the responsibilities defined above.

In order to allow flexibility in how Update and Configuration Management is used, it will expose its functionality via ara::com service interfaces, not direct APIs. This ensures that the user of the functional cluster Update and Configuration Management does not have to be located on the same ECU.

2 Acronyms and abbreviations

The glossary below includes acronyms and abbreviations relevant to the UCM module that are not included in the [1, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
DM	AUTOSAR Adaptive Diagnostic Management
UCM	Update and Configuration Management
UCM Master	UCM Master is distributing packages and coordinating an update campaign in a vehicle
Backend	Backend is a server hosting Software Packages
OTA Client	OTA Client is an Adaptive Application in communication with Backend Over The Air
Application Error	Errors returned by UCM
Boot options	Boot Manager Configuration

Some technical terms used in this document are already defined in the corresponding document mentioned in the table below. This is to avoid duplicate definition of the technical term. And to refer to the correct document.

Term	Description
Adaptive Application	see [1] AUTOSAR Glossary
Application	see [1] AUTOSAR Glossary
AUTOSAR Adaptive Platform	see [1] AUTOSAR Glossary
AUTOSAR Classic Platform	see [1] AUTOSAR Glossary
Electronic Control Unit	see [1] AUTOSAR Glossary
Adaptive Platform Foundation	see [1] AUTOSAR Glossary
Adaptive Platform Services	see [1] AUTOSAR Glossary
Manifest	see [1] AUTOSAR Glossary
Executable	see [1] AUTOSAR Glossary
Functional Cluster	see [1] AUTOSAR Glossary
Machine	see [1] AUTOSAR Glossary
Service	see [1] AUTOSAR Glossary
Service Interface	see [1] AUTOSAR Glossary
Service Discovery	see [1] AUTOSAR Glossary
Execution Management	see [2] AUTOSAR Execution Management
kRunning	see [2] AUTOSAR Execution Management
Software Cluster	see [1] AUTOSAR Glossary
Software Package	see [1] AUTOSAR Glossary
Vehicle Package	see [1] AUTOSAR Glossary

Table 2.1: Reference to Technical Terms

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_TR_Glossary

[2] Specification of Execution Management
AUTOSAR_SWS_ExecutionManagement

[3] General Specification of Adaptive Platform
AUTOSAR_SWS_General

[4] Specification of State Management
AUTOSAR_SWS_StateManagement

[5] Specification of Cryptography for Adaptive Platform
AUTOSAR_SWS_Cryptography

[6] Specification of Communication Management
AUTOSAR_SWS_CommunicationManagement

[7] Specification of Identity and Access Management
AUTOSAR_SWS_IdentityAndAccessManagement

[8] Requirements on Update and Configuration Management
AUTOSAR_RS_UpdateAndConfigManagement

[9] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[10] Explanation of Adaptive Platform Design
AUTOSAR_EXP_PlatformDesign

[11] Specification of Persistency
AUTOSAR_SWS_Persistency

[12] Requirements on Security Management for Adaptive Platform
AUTOSAR_RS_SecurityManagement

3.2 Related specification

See chapter 3.1.

3.3 Further applicable specification

AUTOSAR provides a general specification [3] which is also applicable for UCM. The specification SWS General shall be considered as additional and required specification for implementation of UCM.

4 Constraints and assumptions

4.1 Limitations

UCM is not responsible to initiate the update process. UCM realizes a service interface to achieve this operation. The user of this service interface is responsible to verify that the vehicle is in a safe state before executing a software update procedure on demand. It is also in the responsibility of the user to communicate with other AUTOSAR Adaptive Platforms or AUTOSAR Classic Platforms within the vehicle. Therefore management of software dependencies between different physical or virtual ECU software platforms is currently out of UCM's scope but will be managed by the UCM Master which will be introduced in the next release.

The UCM receives a locally available software package for processing. The software package is usually downloaded from the OEM backend. The download of the software packages has to be done by another application, i.e. UCM does not manage the connection to the OEM backend. Prior to triggering their processing, the software packages have to be transferred to UCM by using the provided `ara::com` interface.

The UCM update process is designed to cover updates on use case with single AUTOSAR Adaptive Platform. UCM can update Adaptive Applications, the AUTOSAR Adaptive Platform itself, including all functional clusters and the underlying OS. Distinction between different types of updates, such as safety critical updates vs infotainment updates, isn't addressed in this release. Currently such distinction shall be included into vendor specific meta-data.

The UCM is not responsible for enforcing authentication and access control to the provided interfaces. The document currently does not provide any mechanism for the confidentiality protection as well as measures against denial of service attacks. The assumption is that the platform preserves the integrity of parameters exchanged between UCM and its user.

The UCM do not support update of ECUs not supporting ARA::COM or UDS with aligned diagnostic flash sequence support.

This UCM Master specification release scope is limited to update, install or remove of Adaptive platform Software Clusters. It is planned to specify any modification of Classic platform (being FOTA or non FOTA compatible) and non-Autosar platform from release 20-11.

4.2 Applicability to car domains

No restrictions to applicability.

5 Dependencies to other functional clusters

The UCM functional cluster expose services to client applications via the ara::com middleware.

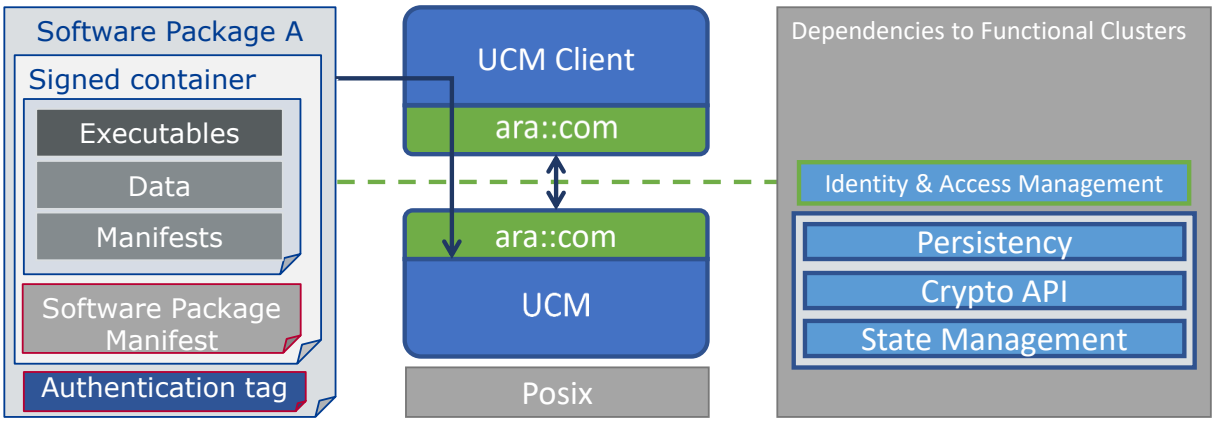

Figure 5.1: UCM dependencies to other Functional Clusters.

5.1 Interfaces to Adaptive State Management

Certain applications can conflict with the update process or the newly updated package, and they need to be stopped during the update process. This could be achieved by putting the machine to a safe Machine State, for example Update State, or by activating a combination of suitable Function Groups and its states. It is the responsibility of the platform integrator to define this state or Function Groups. The application accessing the UCM, should make sure that the platform is switched to this state (using interfaces from State Management [4]), before starting the update.

UCM uses State Management interface field parameter FunctionGroupState to monitor the restart of the updated software.

5.2 UCM service over ara::com

The UCM shall provide a service interface over ara::com using methods and fields.

5.3 Interfaces to Adaptive Crypto Interface

UCM uses Crypto Interface for AUTOSAR Adaptive Platform [5] to verify package integrity and authenticity and to decrypt confidential update data.

5.4 Interfaces to Identity and Access Management

Communication Management,[6] uses Identity and Access Management [7] to validate the authorization of requests made to UCM's service interface PackageManagement.

6 Requirements Tracing

The following tables reference the requirements specified in [8] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_SM_00001]	State Management shall coordinate and control multiple sets of Applications.	[SWS_UCM_00102] [SWS_UCM_00124]
[RS_UCM_00001]	UCM shall support installing new software on AUTOSAR Adaptive Platform	[SWS_UCM_00001] [SWS_UCM_00017] [SWS_UCM_00073] [SWS_UCM_00099] [SWS_UCM_00131] [SWS_UCM_00137] [SWS_UCM_00165] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183]
[RS_UCM_00002]	UCM shall support reporting version information for an AUTOSAR Adaptive Platform	[SWS_UCM_00004] [SWS_UCM_00038] [SWS_UCM_00039] [SWS_UCM_00040] [SWS_UCM_00071] [SWS_UCM_00077] [SWS_UCM_00078] [SWS_UCM_00079] [SWS_UCM_00105] [SWS_UCM_00112] [SWS_UCM_00130] [SWS_UCM_00131] [SWS_UCM_00174] [SWS_UCM_00175] [SWS_UCM_00176] [SWS_UCM_00177] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183] [SWS_UCM_01114] [SWS_UCM_CONSTR_00001]
[RS_UCM_00003]	UCM shall support updating installed software on Adaptive Platform	[SWS_UCM_00017] [SWS_UCM_00165]
[RS_UCM_00004]	UCM shall support uninstalling software on AUTOSAR Adaptive Platform	[SWS_UCM_00001] [SWS_UCM_00137] [SWS_UCM_00165]
[RS_UCM_00005]	UCM shall make sure that persistent data owned by uninstalled software is deleted	[SWS_UCM_00001] [SWS_UCM_00137]

Requirement	Description	Satisfied by
[RS_UCM_00006]	UCM shall verify Software Package authenticity and integrity using strong cryptographic techniques	[SWS_UCM_00028] [SWS_UCM_00038] [SWS_UCM_00039] [SWS_UCM_00040] [SWS_UCM_00077] [SWS_UCM_00078] [SWS_UCM_00079] [SWS_UCM_00136]
[RS_UCM_00007]	UCM shall check that software dependencies are fulfilled	[SWS_UCM_00026] [SWS_UCM_00027] [SWS_UCM_00120] [SWS_UCM_00128] [SWS_UCM_00136] [SWS_UCM_00161]
[RS_UCM_00008]	UCM shall support a recovery mechanism in case of failed update process	[SWS_UCM_00005] [SWS_UCM_00024] [SWS_UCM_00096] [SWS_UCM_00107] [SWS_UCM_00110] [SWS_UCM_00111] [SWS_UCM_00113] [SWS_UCM_00126] [SWS_UCM_00127] [SWS_UCM_00131] [SWS_UCM_00142] [SWS_UCM_00146] [SWS_UCM_00155] [SWS_UCM_00162] [SWS_UCM_00163] [SWS_UCM_00164] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183]
[RS_UCM_00010]	UCM shall support reporting of Software Packages downloaded for AUTOSAR Adaptive Platform	[SWS_UCM_00038] [SWS_UCM_00039] [SWS_UCM_00040] [SWS_UCM_00069] [SWS_UCM_00077] [SWS_UCM_00078] [SWS_UCM_00079] [SWS_UCM_00105] [SWS_UCM_00131] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183] [SWS_UCM_CONSTR_00001]

Requirement	Description	Satisfied by
[RS_UCM_00011]	UCM shall support reporting software versions which have been installed and will be activated when new versions are activated	[SWS_UCM_00030] [SWS_UCM_00038] [SWS_UCM_00039] [SWS_UCM_00040] [SWS_UCM_00077] [SWS_UCM_00078] [SWS_UCM_00079] [SWS_UCM_00105] [SWS_UCM_00131] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183] [SWS_UCM_CONSTR_00001]
[RS_UCM_00012]	UCM shall check the consistency of transferred Software Package	[SWS_UCM_00029] [SWS_UCM_00038] [SWS_UCM_00039] [SWS_UCM_00040] [SWS_UCM_00077] [SWS_UCM_00078] [SWS_UCM_00079] [SWS_UCM_00104] [SWS_UCM_00136]
[RS_UCM_00013]	UCM shall check that it has enough resources to receive, process and store the Software Package and associated data	[SWS_UCM_00007] [SWS_UCM_00008] [SWS_UCM_00010] [SWS_UCM_00087] [SWS_UCM_00088] [SWS_UCM_00091] [SWS_UCM_00092] [SWS_UCM_00098] [SWS_UCM_00136] [SWS_UCM_00140] [SWS_UCM_00145] [SWS_UCM_00156]
[RS_UCM_00014]	UCM shall check that correct amount of data has been transferred for the Software Package	[SWS_UCM_00136]
[RS_UCM_00015]	UCM shall remove all unneeded data after Software Package processing has finished	[SWS_UCM_00020] [SWS_UCM_00131] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183]
[RS_UCM_00017]	UCM shall support installing and updating the persistent data storage for an Adaptive Application	[SWS_UCM_00011] [SWS_UCM_00113]
[RS_UCM_00018]	UCM shall announce when an application has been installed, updated or uninstalled	[SWS_UCM_00021] [SWS_UCM_00131] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183]

Requirement	Description	Satisfied by
[RS_UCM_00019]	UCM shall support simultaneous transfers multiple Software Packages	[SWS_UCM_00007] [SWS_UCM_00008] [SWS_UCM_00010] [SWS_UCM_00031] [SWS_UCM_00075] [SWS_UCM_00087] [SWS_UCM_00088] [SWS_UCM_00091] [SWS_UCM_00092] [SWS_UCM_00093] [SWS_UCM_00098] [SWS_UCM_00140] [SWS_UCM_00141] [SWS_UCM_00145] [SWS_UCM_00148] [SWS_UCM_00156]
[RS_UCM_00020]	UCM shall support cancellation of an update or install operation	[SWS_UCM_00003] [SWS_UCM_00167]
[RS_UCM_00021]	UCM shall support atomic activation of installed or updated packages	[SWS_UCM_00022] [SWS_UCM_00025] [SWS_UCM_00094] [SWS_UCM_00131] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183]
[RS_UCM_00022]	UCM shall support logging of the update or installation process	[SWS_UCM_00041] [SWS_UCM_00042] [SWS_UCM_00043] [SWS_UCM_00131] [SWS_UCM_00143] [SWS_UCM_00170] [SWS_UCM_00171] [SWS_UCM_00172] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183]
[RS_UCM_00023]	UCM shall provide an interface to read progress of the update	[SWS_UCM_00018] [SWS_UCM_00131] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183]

Requirement	Description	Satisfied by
[RS_UCM_00024]	UCM shall provide an interface to read the state of UCM	[SWS_UCM_00019] [SWS_UCM_00044] [SWS_UCM_00080] [SWS_UCM_00081] [SWS_UCM_00082] [SWS_UCM_00083] [SWS_UCM_00084] [SWS_UCM_00085] [SWS_UCM_00086] [SWS_UCM_00131] [SWS_UCM_00147] [SWS_UCM_00149] [SWS_UCM_00150] [SWS_UCM_00151] [SWS_UCM_00152] [SWS_UCM_00153] [SWS_UCM_00154] [SWS_UCM_00166] [SWS_UCM_00168] [SWS_UCM_00169] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183]
[RS_UCM_00025]	UCM shall support efficient streaming of Software Package data	[SWS_UCM_00007] [SWS_UCM_00008] [SWS_UCM_00010] [SWS_UCM_00031] [SWS_UCM_00032] [SWS_UCM_00087] [SWS_UCM_00088] [SWS_UCM_00091] [SWS_UCM_00092] [SWS_UCM_00098] [SWS_UCM_00131] [SWS_UCM_00140] [SWS_UCM_00145] [SWS_UCM_00156] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183]
[RS_UCM_00026]	UCM shall process installation of new Software Packages, updates and removal of existing Software Packages sequentially	[SWS_UCM_00017] [SWS_UCM_00044] [SWS_UCM_00122]
[RS_UCM_00027]	UCM shall be able to safely recover from unexpected interruption.	[SWS_UCM_00157] [SWS_UCM_00158]
[RS_UCM_00028]	UCM shall support updating Functional Clusters	[SWS_UCM_00100]
[RS_UCM_00029]	UCM shall support updating the underlying Operating System	[SWS_UCM_00101]

Requirement	Description	Satisfied by
[RS_UCM_00030]	UCM shall be able to verify the updated software during activation	[SWS_UCM_00096] [SWS_UCM_00107] [SWS_UCM_00108] [SWS_UCM_00111] [SWS_UCM_00126] [SWS_UCM_00127] [SWS_UCM_00146] [SWS_UCM_00155] [SWS_UCM_00162] [SWS_UCM_00163] [SWS_UCM_00164]
[RS_UCM_00031]	UCM shall prevent installation of arbitrary previous version of an Adaptive Application or the Adaptive Platform	[SWS_UCM_00103]
[RS_UCM_00032]	UCM shall provide an interface to return UCM's action history	[SWS_UCM_00115] [SWS_UCM_00131] [SWS_UCM_00132] [SWS_UCM_00133] [SWS_UCM_00134] [SWS_UCM_00135] [SWS_UCM_00160] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183] [SWS_UCM_01177]
[RS_UCM_00033]	UCM Master shall support reporting version information of a complete vehicle	[SWS_UCM_01101] [SWS_UCM_01102] [SWS_UCM_01103] [SWS_UCM_01218] [SWS_UCM_01304]
[RS_UCM_00034]	UCM Master shall record all UCM Master's action history	[SWS_UCM_01247] [SWS_UCM_01248]

Requirement	Description	Satisfied by
[RS_UCM_00035]	UCM Master shall coordinate software update in a vehicle across multiple Electronic Control Units	[SWS_UCM_00178] [SWS_UCM_00210] [SWS_UCM_01006] [SWS_UCM_01007] [SWS_UCM_01008] [SWS_UCM_01009] [SWS_UCM_01010] [SWS_UCM_01106] [SWS_UCM_01111] [SWS_UCM_01204] [SWS_UCM_01205] [SWS_UCM_01206] [SWS_UCM_01207] [SWS_UCM_01208] [SWS_UCM_01209] [SWS_UCM_01211] [SWS_UCM_01212] [SWS_UCM_01213] [SWS_UCM_01214] [SWS_UCM_01215] [SWS_UCM_01216] [SWS_UCM_01217] [SWS_UCM_01218] [SWS_UCM_01219] [SWS_UCM_01220] [SWS_UCM_01221] [SWS_UCM_01222] [SWS_UCM_01223] [SWS_UCM_01224] [SWS_UCM_01225] [SWS_UCM_01226] [SWS_UCM_01227] [SWS_UCM_01228] [SWS_UCM_01229] [SWS_UCM_01230] [SWS_UCM_01231] [SWS_UCM_01232] [SWS_UCM_01233] [SWS_UCM_01234] [SWS_UCM_01235] [SWS_UCM_01236] [SWS_UCM_01237] [SWS_UCM_01238] [SWS_UCM_01239] [SWS_UCM_01240] [SWS_UCM_01241] [SWS_UCM_01242] [SWS_UCM_01243] [SWS_UCM_01244] [SWS_UCM_01245] [SWS_UCM_01246] [SWS_UCM_01303]

Requirement	Description	Satisfied by
[RS_UCM_00036]	UCM Master shall use platform communication services for interacting with UCM subordinates	[SWS_UCM_00009] [SWS_UCM_00173] [SWS_UCM_01002] [SWS_UCM_01005] [SWS_UCM_01007] [SWS_UCM_01008] [SWS_UCM_01009] [SWS_UCM_01010]
[RS_UCM_00037]	UCM Master shall ensure it is safe to perform any modification to the vehicle	[SWS_UCM_00179] [SWS_UCM_01004] [SWS_UCM_01106] [SWS_UCM_01108] [SWS_UCM_01109] [SWS_UCM_01110] [SWS_UCM_01111] [SWS_UCM_01112] [SWS_UCM_01113] [SWS_UCM_01115] [SWS_UCM_01222] [SWS_UCM_01223] [SWS_UCM_01224] [SWS_UCM_01226] [SWS_UCM_01228] [SWS_UCM_01229] [SWS_UCM_01230] [SWS_UCM_01231] [SWS_UCM_01234] [SWS_UCM_01235] [SWS_UCM_01237] [SWS_UCM_01238] [SWS_UCM_01240] [SWS_UCM_01244] [SWS_UCM_01245] [SWS_UCM_01246]
[RS_UCM_00038]	UCM Master shall interact with driver	[SWS_UCM_00180] [SWS_UCM_01105] [SWS_UCM_01107] [SWS_UCM_01116] [SWS_UCM_01206] [SWS_UCM_01208] [SWS_UCM_01211] [SWS_UCM_01222] [SWS_UCM_01223] [SWS_UCM_01224] [SWS_UCM_01228] [SWS_UCM_01230] [SWS_UCM_01231] [SWS_UCM_01234] [SWS_UCM_01235] [SWS_UCM_01237]
[RS_UCM_00039]	UCM Master shall prevent processing of compromised Vehicle Packages	[SWS_UCM_01001] [SWS_UCM_01221] [SWS_UCM_01301] [SWS_UCM_01302]

Requirement	Description	Satisfied by
[RS_UCM_00042]	UCM Master shall provide an interface to read the state of an update campaign	[SWS_UCM_01203] [SWS_UCM_01205]
[RS_UCM_00043]	UCM Master shall orchestrate a software update campaign according to the Vehicle Package's Manifest	[SWS_UCM_00179] [SWS_UCM_00180] [SWS_UCM_00210] [SWS_UCM_01001] [SWS_UCM_01003] [SWS_UCM_01006] [SWS_UCM_01115] [SWS_UCM_01116] [SWS_UCM_01201] [SWS_UCM_01207] [SWS_UCM_01209] [SWS_UCM_01212] [SWS_UCM_01228] [SWS_UCM_01230] [SWS_UCM_01301] [SWS_UCM_01302] [SWS_UCM_01303]

7 Functional specification

7.1 UCM

7.1.1 Technical Overview

One of the declared goals of AUTOSAR Adaptive Platform is the ability to flexibly update the software and its configuration through over-the-air updates. During the life-cycle of an AUTOSAR Adaptive Platform, UCM is responsible to perform software modifications on the machine and to retain consistency of the whole system.

The UCM Functional Cluster provides a service interface that exposes its functionality to retrieve AUTOSAR Adaptive Platform software information and consistently execute software updates. Since ara::com is used, the client using the UCM service interface can be located on the same AUTOSAR Adaptive Platform, but also remote clients are possible.

The service interface has been primarily designed with the goal to make it possible to use standard diagnostic services for downloading and installing software updates for the AUTOSAR Adaptive Platform. However, the methods and fields in the service interface are designed in such a way that they can be used in principle by any Adaptive Application. UCM does not impose any specific protocol on how data is transferred to the AUTOSAR Adaptive Platform and how package processing is controlled. In particular UCM does not expose diagnostic services.

As shown in Figure 7.1, wether the use case is an over-the-air update or garage update done through diagnostics, it is not visible to the UCM. The UCM Client abstracts the use case from the UCM and forwards the data stream and sequence control commands to the UCM. Later in this document the term UCM Client is used to cover both roles: Diagnostic Application and OTA Client.

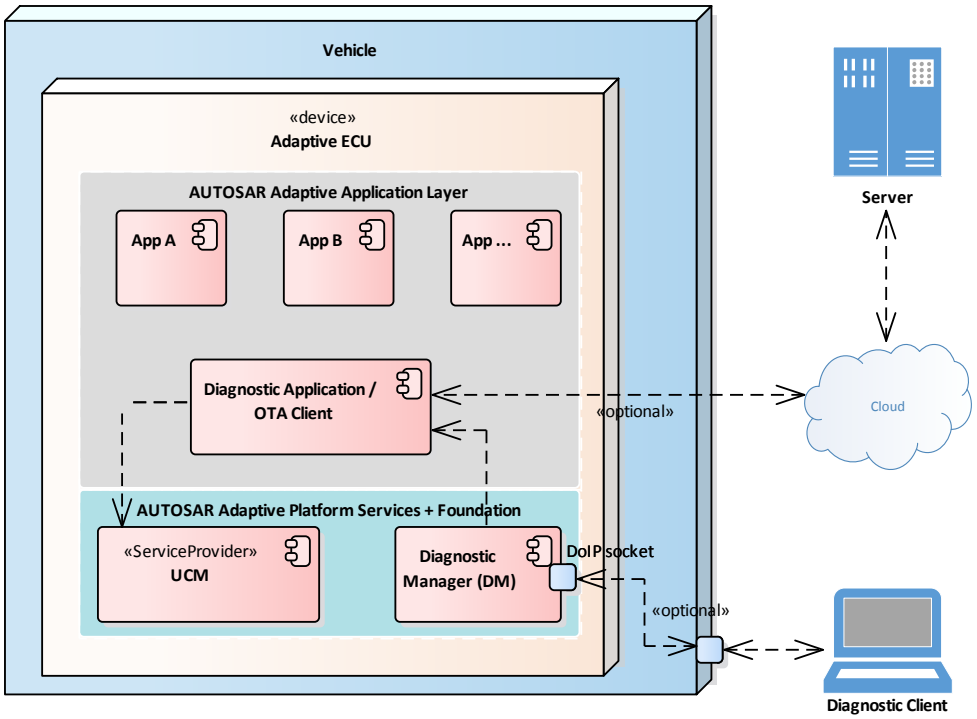

Figure 7.1: Architecture overview for diagnostic use case

7.1.1.1 Software Package Management

The UCM update sequence consists three different phases:

- Software Package transfer: A phase in which, one or several Software Packages are transferred from the UCM's Client Application to the internal buffer of the UCM. For further information see chapter 7.1.2.
- Software Package processing: A phase in which the UCM performs the operation (kInstall, kUpdate, kRemove) on the relevant SoftwareCluster. For further information see chapter 7.1.4.
- Activation: A phase in which the UCM checks the dependencies of the SoftwareClusters that have been involved in the operation, then activates them and finally check that all the SoftwareClusters can be executed properly (via State Management [4]) prior to finishing the update. For further information see chapter 7.1.6

7.1.1.1.1 Software Package

[SWS_UCM_00122] Software Package utilization ⌈The unit for deployment that the UCM shall take as input is called Software Package, see [1]. Each Software Package shall address a single SoftwareCluster.⌋ *(RS_UCM_00026)*

A SoftwareCluster can act in two roles:

- 'Sub'-SoftwareCluster : It is a SoftwareCluster without diagnostic target address, containing processes, executables and further elements
- 'Root'-SoftwareCluster : It is a SoftwareCluster with a diagnostic target address that may reference several other 'Sub'-SoftwareClusters, which thus form a logical group.

The two roles are expressed by reserved values of the attribute SoftwareCluster.category.

A Software Package has to be modelled as a so-called SoftwareCluster which describes the content of a Software Package that has to be uploaded to the AUTOSAR Adaptive Platform, see [9].

The term Software Package is used for the "physical", uploadable Software Package that is processed by UCM whereas the term SoftwareCluster is used for the modeling element. In the model, the content of a SoftwareCluster is define by references to all required model elements. The SoftwareCluster and the related model elements define the content of the manifest that is part of the Software Package. The Software Package format and the update scope are described in chapter "Content of a Software Package" as well as in [10].

7.1.1.1.2 Content of a Software Package

Each Software Package addresses a single SoftwareCluster and contains manifests, executables and further data (depending on the role of the SoftwareCluster) as example sketched in Figure 7.2.

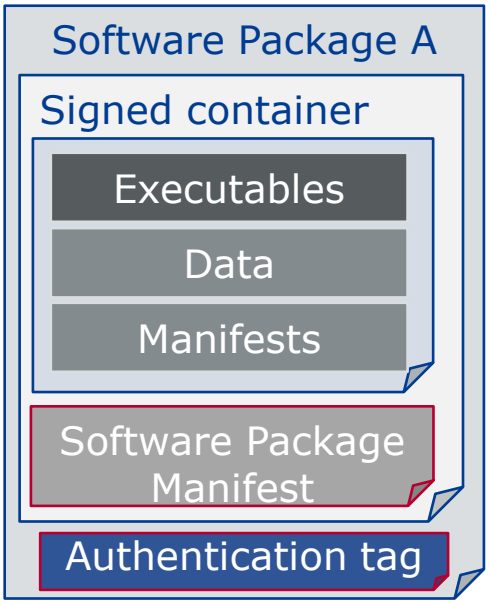

Figure 7.2: Software Package content description

A single Software Package is designed in a way that it could contain one or several executables of Adaptive Applications, kernel or firmware updates, or updated configuration and calibration data to be deployed on the AUTOSAR Adaptive Platform. An exemplary implementation of the adaptive workflow with Software Packages can be seen in chapter Methodology and Manifest in [10].

[SWS_UCM_00112]{DRAFT} **Software Cluster and version** ⌈SoftwareCluster's manifest shall include a name and a version following semantic versioning 2.0.0 (https://semver.org/). A time stamp shall be trailing the Major.Minor.Patch version.⌋ *(RS_UCM_00002)*

[SWS_UCM_CONSTR_00001] ⌈If any content of SoftwareCluster is modified by a Software Package, for instance an executable or persistent data, then the version number of SoftwareCluster indicated in the Software Package shall be higher.⌋ *(RS_UCM_00002, RS_UCM_00010, RS_UCM_00011)*

[SWS_UCM_00130] Software Cluster and version error ⌈If SoftwareCluster's manifest does not contain any version as specified in [SWS_UCM_00112], UCM shall raise the ApplicationError InvalidManifest.⌋ *(RS_UCM_00002)*

7.1.1.1.3 Applications Persisted Data

[SWS_UCM_00011]{DRAFT} **Updating persisted data** ⌈The UCM shall be able to create, update or remove any persistency data that is contained in the SoftwareCluster.⌋ *(RS_UCM_00017)*

Further details on the persistent data can be found in Persistency Specification [11].

[SWS_UCM_00113]{DRAFT} **Rollback of persisted data** ⌈The UCM shall be able to rollback changes done to persistent data during update process.⌋ *(RS_UCM_00017, RS_UCM_00008)*

7.1.1.2 Runtime dependencies

Both 'Sub' and 'Root' SoftwareCluster can have execution dependencies toward other SoftwareClusters.

Dependencies are described in the SoftwareCluster metamodel, see [9].

[SWS_UCM_00120]{DRAFT} **Runtime dependencies check** ⌈UCM shall check runtime dependencies before the activation of the new software version. This action is done in the context of Activate.⌋ *(RS_UCM_00007)*

The rationale is, if UCM has to process several Software Packages, then execution dependencies may not be fulfilled at all times during the Software Packages process but must be fulfilled before changes can be activated.

[SWS_UCM_00128]{DRAFT} ⌈If dependency check fails, UCM shall raise the ApplicationError MissingDependencies and change its state from kActivating to kReady.⌋*(RS_UCM_00007)*

7.1.1.3 Update scope and state management

Software Package processed by UCM can contain Adaptive Applications, updates to AUTOSAR Adaptive Platform itself or to the underlying OS. Update type depends on the content of the Software Package.

[SWS_UCM_00099]{DRAFT} **Update of Adaptive Application** ⌈UCM shall be able to update Adaptive Applications⌋*(RS_UCM_00001)*

[SWS_UCM_00100]{DRAFT} **Update of Functional Clusters** ⌈UCM shall be able to update all Functional Clusters, including UCM itself.⌋*(RS_UCM_00028)*

[SWS_UCM_00101]{DRAFT} **Update of Host** ⌈UCM shall be able to update the underlying OS hosting the AUTOSAR Adaptive Platform.⌋*(RS_UCM_00029)*

Definition of a safe state with respect to the system setup is the OEM responsibility. Based on the system setup and the application, the system might need to be switched into an **update** state, to free resource to speed up the update, to block normal usage of software which might cause interruptions to update process and to block using functionality which might be interrupted by the update sequence.

[SWS_UCM_00102]{DRAFT} **Update state** ⌈For the updates of processes associated with Machine State Function Group, UCM shall check that system is set to **update** state.⌋*(RS_SM_00001)*

In **update** state only the applications required for the Update process are executed. This way system is more robust, more resources are free and user is blocked from using applications, of which failure could cause safety risk to the user.

It is the responsibility of the UCM Client to request the transition to **update** state, using suitable interfaces of Adaptive State Management [4].

[SWS_UCM_00124]{DRAFT} **Verify State** ⌈As minimal check UCM shall check that processed Software Package is able to reach kRunning state. For checking if the updated software can reach the kRunning state, the machine or the related Function Group (depending on what is updated) shall be set into **verify** state.⌋*(RS_SM_00001)*

After the Dependency Check has been performed successfully, kVerifying state is set (see chapter 7.1.5 for more details). In this state, it is the responsibility of the UCM Client to request the transition to **verify** state, using suitable interfaces of Adaptive State Management [4]. Then, State Management [4] will return a successful state change only if all the relevant processes have reached the kRunning state. This gives a chance to perform a Rollback if some processes fails to reach the kRunning state.

Update of some components require a Machine reset to be performed. These components should be configured to be part of Machine State function group, as the update sequence of Machine State function group includes a Machine reset. `Execution Manager`, `State Manager`, `Communication Manager` and `UCM` itself are good examples which probably require a Machine reset to activate the update. Other such components could be applications involved in the update sequence or applications involved in safety monitoring. Further details on `Machine State function group` can be found in State Management [4].

7.1.2 Transferring Software Packages

To speed up the overall data transmission time, the package transfer is decoupled from the processing and activation process. This section describes requirements for initiation of a data transfer, the data transmission and ending of the data transmission.

Each `Software Package` gets its own state as soon as it is being transferred to `UCM`. The state machine in Fig. 7.3 specifies the lifecycle of a `Software Package` that is transferred to and processed by `UCM`. During this lifecycle, a `Software Package` is uniquely identified with an `id` that `UCM` provides to the client.

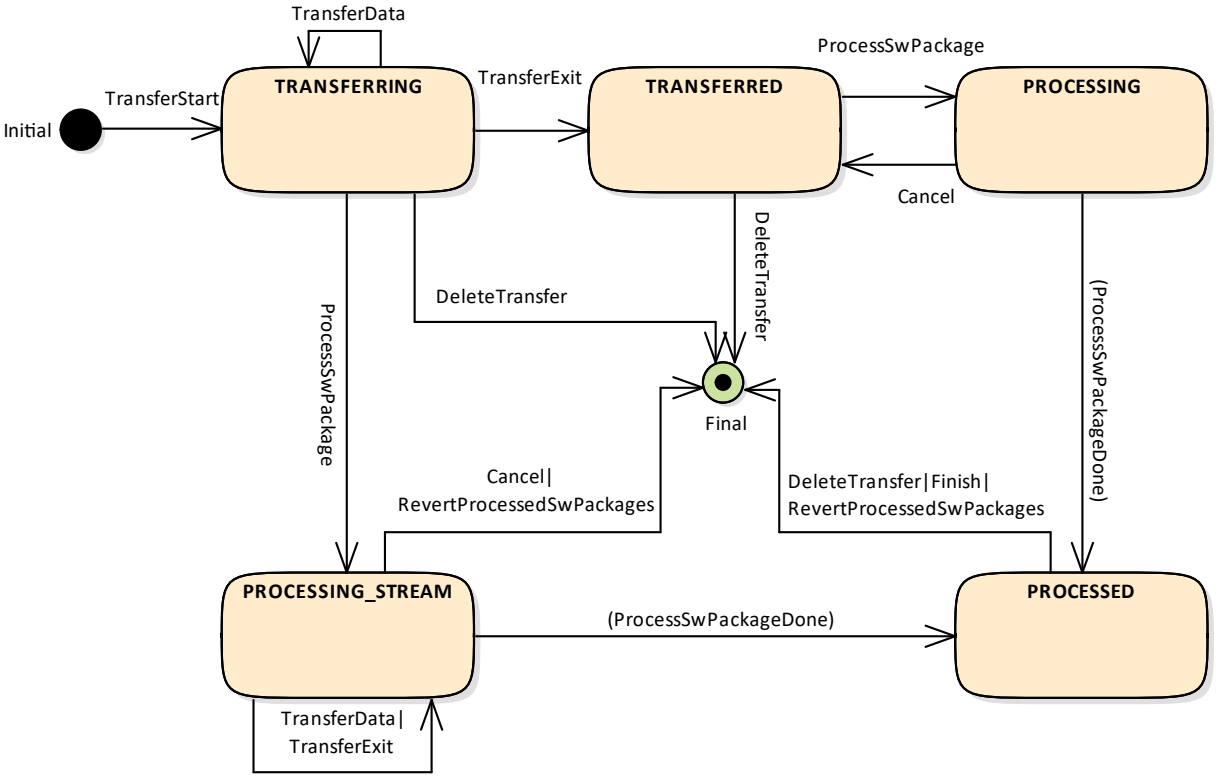

Figure 7.3: State Machine for transferring packages using service interface `PackageManagement`

[SWS_UCM_00007]{DRAFT} **Data transfer at any time** ⌈`UCM` shall provide support to transfer `Software Packages` at any time when `UCM` is running. Transferring is

decoupled from the UCM Package Management states.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

[SWS_UCM_00088]{DRAFT} **Preparation of data transfer** ⌈Data transfer shall be prepared with the method TransferStart. In the preparation step the number of bytes to be transferred is provided by the client and UCM assigns a id for the Software Package to be transferred.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

[SWS_UCM_00140]{DRAFT} **UCM insufficient memory** ⌈TransferStart method shall raise the ApplicationError InsufficientMemory if the UCM buffer has not enough resources to store the corresponding Software Package.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

[SWS_UCM_00008] Executing the data transfer ⌈After preparing of the data transfer, the transmission of the Software Package block-wise shall be supported by the method TransferData.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

[SWS_UCM_00145] Sequential order of data transfer ⌈The method TransferData shall support the parameter blockCounter that shall start with 0x01 and incremented by one for each subsequent block.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

[SWS_UCM_00010] End of data transfer ⌈After transmission of a Software Package is completed, the transmission can be finished with method TransferExit.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

[SWS_UCM_00156]{DRAFT} **Procurement of Checksum** ⌈During TransferExit, the client may also provide to the UCM the Checksums (e.g. Checksum of the Software Packages, Checksum of the Payload) needed by the UCM for performing the upcoming integrity checks.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

[SWS_UCM_00087] Insufficient amount of data transferred ⌈During TransferExit UCM shall check if all blocks of the Software Package have been transferred according to the size parameter of TransferStart. If not UCM shall return ApplicationError InsufficientData.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

[SWS_UCM_00092]{DRAFT} **Package consistency** ⌈During TransferExit UCM shall raise the ApplicationError PackageInconsistent if the package integrity check fails. This package integrity check may be realized by the UCM via a Package Checksum check or via other mechanisms.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

[SWS_UCM_00028]{DRAFT} **Package Authentication** ⌈UCM shall authenticate the Software Package.⌋*(RS_UCM_00006)*

Software Package contains authentication and integrity tags, which are used during update sequence to authenticate the source of the Software Package. Usage of hash algorithms and cryptographic signatures to validate the package authenticity is defined in [12].

[SWS_UCM_00098]{DRAFT} **Package Authentication failure** ⌈During TransferExit UCM shall raise the ApplicationError AuthenticationFailed, if the data authentication check fails.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

[SWS_UCM_00091]{DRAFT} **Successful data transfer** ⌈During TransferExit UCM shall not raise any ApplicationError if the transfer of data could be successfully finished.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

[SWS_UCM_00075] Multiple data transfers in parallel ⌈Handling of multiple data transfers in parallel shall be supported by UCM.⌋*(RS_UCM_00019)*

[SWS_UCM_00141]{DRAFT} **UCM insufficient memory for parallel data transfer** ⌈While a Software Package is being transferred, if UCM receives a subsequent TransferStart call targeting another Software Package, UCM shall make sure that the sum of the size of both Software Packages (the one being transferred and the one requested to be transferred) does not exceed the size of the UCM buffer. Otherwise, the TransferStart shall raise the ApplicationError InsufficientMemory and the newly requested transmission shall be rejected.⌋*(RS_UCM_00019)*

If UCM provide enough buffering resources for Software Packages, several packages could be transferred (in parallel) before they are processed one after the other. The processing (i.e. unpacking and actually applying changes to the AUTOSAR Adaptive Platform) of Software Packages described by the state kProcessing is further detailed in Sect. 7.1.4.

[SWS_UCM_00021] Deleting transferred Software Packages ⌈UCM shall provide a method DeleteTransfer that shall delete the targeted Software Package and free the resources reserved to store that Software Package.⌋*(RS_UCM_00018)*

[SWS_UCM_00093] Transfer sequence ⌈For each Software Package UCM shall ensure that TransferStart, TransferData and TransferExit had been used.⌋*(RS_UCM_00019)*

[SWS_UCM_00148]{DRAFT} **Transfer sequence order** ⌈Calling TransferExit without calling TransferData at least once shall raise the ApplicationError OperationNotPermitted.⌋*(RS_UCM_00019)*

[SWS_UCM_00069]{DRAFT} **Report information on Software Packages** ⌈UCM shall provide a method GetSwPackages of the interface service PackageManagement to provide the identifiers, names and versions of Software Packages of any state.⌋*(RS_UCM_00010)*

If Software Package is in kTransferring state, it is not possible to get versions or names as manifest could not be complete or accessible, therefore method GetSwPackages should return empty values except for identifiers at this particular state.

7.1.3 Processing of Software Packages from a stream

It is also possible to process a Software Package while the transfer is still ongoing. The following requirements apply for this use case.

[SWS_UCM_00165]{DRAFT} **Processing from stream** ⌈The UCM may support calling ProcessSwPackage directly from stream without waiting to receive the Software Package completely.⌋*(RS_UCM_00001, RS_UCM_00003, RS_UCM_00004)*

[SWS_UCM_00166]{DRAFT} **Processing from stream state** ⌈If UCM supports processing from stream and is in state kIdle or kReady, the method ProcessSwPackage for a Software Package in state kTransferring shall set this Software Package to state kProcessingStream.⌋*(RS_UCM_00024)*

[SWS_UCM_00167]{DRAFT} **Cancelling streamed packages** ⌈All temporary and processed data of a Software Package in state kProcessingStream shall be removed if Cancel is called.⌋*(RS_UCM_00020)*

[SWS_UCM_00168]{DRAFT} **Transferring while processing from stream** ⌈Software Package state shall remain in kProcessingStream when TransferData is called.⌋*(RS_UCM_00024)*

[SWS_UCM_00169]{DRAFT} **Finishing transfer while processing from stream** ⌈Software Package state shall remain in kProcessingStream when TransferExit is called until the Software Package is completely processed.⌋*(RS_UCM_00024)*

7.1.4 Processing Software Packages

In contrast to package transmission, only one Software Package can be processed at the same time to ensure consistency of the system. In the following, a software or package processing can involve any combination of an installation, update or removal of applications, configuration data, calibration data or manifests. It is up to the vendor-specific metadata inside a Software Package to describe the tasks UCM has to perform for its processing. For a removal, this might involve metadata describing which data needs to be deleted. Nevertheless, the communication sequence between the triggering application of the software modification and UCM is the same in any case. For an update of an existing application, the Software Package can contain only partial data, e.g. just an updated version of the execution manifest.

[SWS_UCM_00001]{DRAFT} **Starting the package processing** ⌈UCM shall provide a method ProcessSwPackage to process transferred Software Package. id corresponding to Software Package shall be provided for this method.⌋*(RS_UCM_00001, RS_UCM_00004, RS_UCM_00005)*

[SWS_UCM_00137]{DRAFT} **Processing several update Software Packages** ⌈UCM shall support processing of several Software Packages by calling method

ProcessSwPackage several times in sequence.⌋(*RS_UCM_00001, RS_UCM_00004, RS_UCM_00005*)

During package processing, the progress is provided.

[SWS_UCM_00018]{DRAFT} **Providing Progress Information** ⌈UCM shall provide a method GetSwProcessProgress to query the progress of executing the ProcessSwPackage method call for provided transferId. Parameter progress shall be set to a value representing the progress between 0% and 100% (0x00 ... 0x64).⌋(*RS_UCM_00023*)

[SWS_UCM_00029]{DRAFT} **Consistency Check of Manifest** ⌈UCM shall validate the content of the manifest against the schema defined for the meta-data(eg: for missing parameter or for value out of range of the parameter) and shall raise the ApplicationError InvalidManifest if it finds discrepancies there.⌋(*RS_UCM_00012*)

[SWS_UCM_00104]{DRAFT} **Consistency Check of processed Package** ⌈UCM shall raise the ApplicationError ProcessedSoftwarePackageInconsistent if integrity check of the processed Software Packages fails. This operation is realized by the UCM to verify that it did not corrupt any files during the processing. This integrity check may be realized by the UCM by checking the payload Checksum or by any other mechanisms.⌋(*RS_UCM_00012*)

[SWS_UCM_00003]{DRAFT} **Cancelling the package processing** ⌈UCM shall provide a method Cancel to cancel the running package processing. UCM shall then abort the current package processing task, undo any changes and free any reserved resources.⌋(*RS_UCM_00020*)

[SWS_UCM_00024]{DRAFT} **Revert all processed Software Packages** ⌈UCM shall provide a method RevertProcessedSwPackages to revert all changes done with ProcessSwPackage.⌋(*RS_UCM_00008*)

Depending on the capabilities of UCM and of the updated target, Cancel and RevertProcessedSwPackages is used to revert all the changes that have been applied by ProcessSwPackage. For example, if an application with large resource files is updated “in place” (i.e. in the same partition) then it might not be feasible to revert the update. In this case, to perform a rollback the triggering application could download a Software Package to restore a stable version of the application.

[SWS_UCM_00161]{DRAFT} **Check Software Package version compatibility against UCM version** ⌈At ProcessSwPackage, TransferData or TransferExit calls, UCM shall raise ApplicationError IncompatiblePackageVersion if the version for the Software Package transferred or to be processed is not compatible with the current version of UCM⌋(*RS_UCM_00007*)

The Software Package is generated by a tooling including a packager which version could not match with the UCM version, leading to manifest interpretation issues for instance.

7.1.5 Status Reporting

Once Software Packages are transferred to UCM, they are ready to be processed to finally apply changes to the AUTOSAR Adaptive Platform. In contrast to the transmission, the processing and activation tasks have to happen in a strict sequential order.

To give an overview of the update sequence, the global state of UCM is described in this section. The details of the processing and activation phases and the methods are specified in the 7.1.4 and 7.1.6.

The global state of UCM can be queried using the field CurrentStatus. The state machine for CurrentStatus is shown in Fig. 7.4.

[SWS_UCM_00019]{DRAFT} **Status Field of Package Management** ⌈The global state of UCM shall be provided using the field CurrentStatus⌋ *(RS_UCM_00024)*

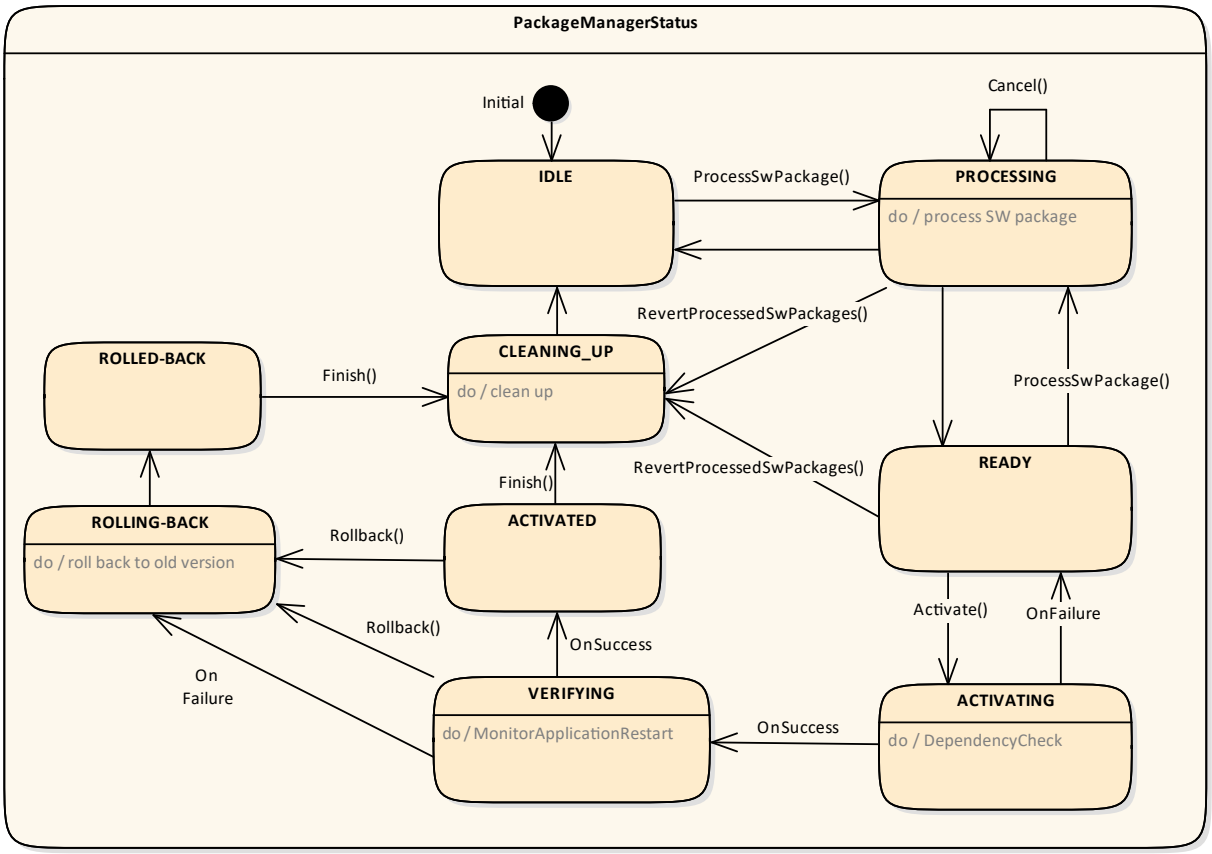

Figure 7.4: State Machine for the package processing using service interface: Package-Management

UCM supported method calls for each value of field CurrentStatus are shown in Fig. 7.4.

[SWS_UCM_00086]{DRAFT} **Unsupported method calls** ⌈Unsupported method calls shall raise the ApplicationError OperationNotPermitted.⌋ *(RS_UCM_00024)*

[SWS_UCM_00080]{DRAFT} **Idle state of Package Management** ⌈kIdle shall be the default state.⌋*(RS_UCM_00024)*

[SWS_UCM_00147]{DRAFT} **Return to the Idle state from Cleaning-up state** ⌈kIdle state shall be set when the Clean-up operation has been completed succesfully. Once ProcessSwPackage is performed successfully, UCM is managing two software configurations, active and inactive. UCM must go through kCleaningUp state to start a new update from kIdle state.⌋*(RS_UCM_00024)*

[SWS_UCM_00082]{DRAFT} **Exit from Processing state of Package Management** ⌈kProcessing state shall be exited when processing of called method ProcessSwPackage or RevertProcessedSwPackages has finished or after the processing of the package has been interrupted by calling Cancel. Following state reported by CurrentStatus is kCleaning-up in case of a RevertProcessedSwPackages call or kReady in case of a ProcessSwPackage completion.⌋*(RS_UCM_00024)*

[SWS_UCM_00150]{DRAFT} **Cancellation of a Software Package processing** ⌈ProcessSwPackage method shall raise the ApplicationError ProcessSwPackageCancelled if the Cancel method has been called during the processing of a Software Package.⌋*(RS_UCM_00024)*

[SWS_UCM_00149]{DRAFT} **Return to the Idle state from Processing state** ⌈kIdle state shall be set when ProcessSwPackage returns with error code ProcessSwPackageCanceled and if no other Software Packages were previously processed during this processing operation.⌋*(RS_UCM_00024)*

[SWS_UCM_00151]{DRAFT} **Entering the Ready state of Package Management after a Cancel call** ⌈If ProcessSwPackage has been cancelled, it shall return error code ProcessSwPackageCanceled and set state to kReady only if at least one other Software Package was previously processed during this processing operation.⌋*(RS_UCM_00024)*

[SWS_UCM_00081]{DRAFT} **Processing state of Package Management** ⌈kProcessing state shall be set only if ProcessSwPackage has been called. This shall only be possible, if CurrentStatus is reported as kIdle or kReady.⌋*(RS_UCM_00024)*

[SWS_UCM_00017]{DRAFT} **Sequential Software Package Processing** ⌈Once method ProcessSwPackage has been called by a client, further calls to the same method shall be rejected with ApplicationError ServiceBusy as long as CurrentStatus is different than kIdle or kReady.⌋*(RS_UCM_00001, RS_UCM_00003, RS_UCM_00026)*

[SWS_UCM_00083]{DRAFT} **Entering the Ready state of Package Management after a successful processing operation** ⌈kReady state shall be set after a Software Package processing has been completed successfully.⌋*(RS_UCM_00024)*

[SWS_UCM_00152]{DRAFT} **Entering the Ready state of Package Management after a missing dependency** ⌈kReady state shall be set when Activate fails due to an ApplicationError MissingDependencies.⌋*(RS_UCM_00024)*

[SWS_UCM_00084]{DRAFT} **Entering the Activating state of Package Management** ⌈kActivating shall be set when Activate is called. This triggers the dependency check and prepares the processed Software Package to be executed in the next restart of the machine or Function Group.⌋*(RS_UCM_00024)*

[SWS_UCM_00153]{DRAFT} **Action in Activating state of Package Management** ⌈When kActivating is set, the UCM shall perform a dependency check to ensure that all the Software Packages having dependencies toward each other have been processed successfully and shall return ApplicationError MissingDependencies if this check fails.⌋*(RS_UCM_00024)*

[SWS_UCM_00154]{DRAFT} **Entering the Verifying state of Package Management** ⌈kVerifying shall be set when ActivateReturnType is returned and no error has been raised in the context of the Activate call. This implies that the dependency check have been performed successfully (all dependencies are satisfied) and that the processed Software Package can now be executed.⌋*(RS_UCM_00024)*

In kVerifying, the machine has to be restarted in case a A/B partition is used. In case the A/B partition is not used, all affected Function Groups or the platform could be restarted. Immediately after the processed Software Package has been restarted, a system check has to be performed in order to make sure the machine is able to start up as expected. With this check it is verified that other safety relevant software like Functional Cluster Platform Health Manager is running and user can be protected from any issues caused by the update after the update has finished. To do so, one mechanism offered by the Adaptive Platform is to restart the processed Software Package into a Verify Function Group state (refer to requirement [SWS_UCM_00124] and State Management [4] specification).

[SWS_UCM_00085]{DRAFT} **Entering the Activated state of Package Management** ⌈kActivated state shall be set when the machine or all impacted Function Groups (the ones related to the processed Software Package) have been successfully restarted into **verify** Function Group state. Practically, this is done when the [4] Function GroupState field notifies that the Verify state associated with the processed Software Package has been reached (which means that all the updated processes have reached the kRunning state).⌋*(RS_UCM_00024)*

UCM monitors FunctionGroupStates from State Management [4] to conclude if activation was successful. kVerifying state gives the client controlling the update process a chance to perform verification test, though functionality in verify state can be limited. Client can also coordinate the results over several AUTOSAR Adaptive Platforms and still perform a Rollback if verification indicates the need for it.

If the system check is successful, the client can decide either to Rollback the current active processing so that the previous processed working software gets started, or to perform Finish so that the changes of processed software become permanent. By calling Finish a clean-up is initiated and in case of A/B partition, a swap between the partitions happens and the newly inactive partition becomes a copy of the newly active partition. In case Finish succeeds (including the clean-up), the current CurrentStatus changes to kIdle.

For Rollback the update software needs to be deactivated and possibly reactivated from original version. e.g. self-update of UCM. For this reason Rollback is also performed through two states, similarly as activation. Calling Rollback sets UCM into kRollingBack state where original software version is made executable and where original software is activated by the State Management [4], then UCM goes to kRolledBack state. In this state all the changes introduced during update process have been deactivated and can be cleaned by calling Finish.

[SWS_UCM_00126]{DRAFT} **Entering the RollingBack state after a Rollback call** ⌈The state kRollingBack shall be set when Rollback is called. This prepares the original software to be executed in the next restart of the machine or Function Group.⌋*(RS_UCM_00008, RS_UCM_00030)*

[SWS_UCM_00155]{DRAFT} **Entering the RollingBack state after a failure in the Verifying state** ⌈The state kRollingBack shall be set when a failure occurs in the Verifying state. Such failure could result from the [4] Function GroupState field notifying that the updated processes could not be executed successfully (i.e. when verify state is reported as not reached by [4]).⌋*(RS_UCM_00008, RS_UCM_00030)*

[SWS_UCM_00111]{DRAFT} **Entering the Rolled-back state** ⌈The state kRolled-Back shall be set when State Management FunctionGroupState field indicates that all the software updated have been restarted or shutdown.⌋*(RS_UCM_00008, RS_-UCM_00030)*

[SWS_UCM_00146]{DRAFT} **Entering the Cleaning-up state after a Finish call** ⌈The state kCleaning-up shall be set when Finish is called and the UCM starts to perform cleanup actions.⌋*(RS_UCM_00008, RS_UCM_00030)*

[SWS_UCM_00162]{DRAFT} **Entering the Cleaning-up state after a RevertProcessedSwPackages call** ⌈The state kCleaning-up shall be set when RevertProcessedSwPackages is called and the UCM starts to perform cleanup actions.⌋*(RS_-UCM_00008, RS_UCM_00030)*

[SWS_UCM_00163]{DRAFT} **Action in Cleaning-up state** ⌈When kCleaning-up state is set, the UCM shall clean up all data of the processed packages that are not needed anymore.⌋*(RS_UCM_00008, RS_UCM_00030)*

[SWS_UCM_00164]{DRAFT} **Cleaning up of Software Packages** ⌈In kCleaning-up state, the UCM may remove (from the UCM buffer for instance) the "physical" Software Package (e.g. zip file) that was used to transport the the SoftwareCluster to the UCM.⌋*(RS_UCM_00008, RS_UCM_00030)*

[SWS_UCM_00127]{DRAFT} **Finishing update sequence** ⌈kIdle shall be set when Finish is called and the clean-up has been successfully performed. This finishes the update sequence and next sequence can be started.⌋*(RS_UCM_00008, RS_UCM_-00030)*

7.1.6 Activation and Rollback

[SWS_UCM_00108]{DRAFT} **Execution of the update software** ⌈UCM shall only commit updates which have been successfully executed. As part of Activation sequence a context switch to updated software is performed and updated software is executed, before update sequence can be successfully Finished.⌋*(RS_UCM_00030)*

UCM should notify the activation or rollback of Software Packages to other Functional Clusters of the AUTOSAR Adaptive Platform. Vendor specific solution dictates to which modules this information is available, in which form and if this is done directly when change is done or when change is executed.

7.1.6.1 Activation

The SoftwareCluster state kPresent does not express whether a SoftwareCluster is currently executed or not.

[SWS_UCM_00107]{DRAFT} **Activated state** ⌈UCM state kActivated shall express that new version of updated SoftwareCluster is executed.⌋*(RS_UCM_00008, RS_UCM_00030)*

The state management on the level of execution is handled by the UCM's client controlling the update process.

UCM has to be able to update several SoftwareClusters for an update campaign. However, these SoftwareClusters could have dependencies not satisfied if updates are processed and activated one by one. Therefore, UCM splits the activation action from the general package processing.

[SWS_UCM_00026]{DRAFT} **Dependency Check** ⌈At activation (i.e. when Activate is called), UCM shall check that dependencies described in the SoftwareClusters are all fulfilled. Unfulfilled dependencies shall raise the ApplicationError MissingDependencies.⌋*(RS_UCM_00007)*

[SWS_UCM_00027]{DRAFT} **Delta Package activation** ⌈Minimum version of SoftwareCluster on which to apply delta shall be included into SoftwareCluster dependency model with role SoftwareCluster.dependsOn⌋*(RS_UCM_00007)*

[SWS_UCM_00025]{DRAFT} **Activation of SoftwareClusters** ⌈UCM shall offer method Activate to enable execution of any pending changes from the previously processed Software Packages. After Activate the new set of SoftwareClusters can be started. Activation covers all the processed Software Packages for all the clients.⌋*(RS_UCM_00021)*

[SWS_UCM_00022]{DRAFT} **Shared Activation of Software Packages** ⌈UCM shall activate all the processed Software Packages when Activate is called.⌋*(RS_UCM_00021)*

The activation method could either lead to a full system reset or restart of Function Groups impacted by the Software Package. When Software Package updates underlying OS, AUTOSAR Adaptive Platform or any Adaptive Application which is configured to be part of Machine State function group, the execution of updated software occurs through system reset. In other cases Function Group restarts can be used to execute the updated software. Meta-data of Software Package defines the activation method, but it can be overruled using an optional input argument indicating if a system reset or Function Group restart will occur.

The UCM does not trigger the restart of processed software. This needs to be performed by the client application. This is due to the fact that such restart might need to be synchronized between several Platforms/ECUs (e.g. during an update campaign where several dependent Software Packages from several ECUs have to be updated).

7.1.6.2 Rollback

[SWS_UCM_00005]{DRAFT} **Rollback to the software prior to Finish the update process** ⌈UCM shall provide a method Rollback to recover from an activation that went wrong.⌋*(RS_UCM_00008)*

Rollback can be called in the case of A/B partitions or UCM uses some other solution to maintain backups of updated or removed Software Packages.

[SWS_UCM_00110]{DRAFT} **Rolling-back the software update** ⌈During Rolling-Back UCM shall disable the changes done by the software update.⌋*(RS_UCM_00008)*

[SWS_UCM_00142]{DRAFT} **Prevent software from blocking the Rollback operation** ⌈While Rolling-Back, UCM can forcefully shutdown the newly processed software (i.e the one that needs to be the Rolled-back), if needed, in order to avoid this software blocking the Rollback operation.⌋*(RS_UCM_00008)*

7.1.6.3 Boot options

During update process the executed software is switched from original software to updated software and in case of rollback, from updated software to original version. Which version of software is executed is dependent on the UCM state and this is managed by the UCM. In case of platform and OS update the switch between software versions occurs through system reset and depending on the system design the Execution Management [2] might be started before UCM. In this case there can't be direct interface between UCM and Execution Management [2] to define which versions of software would be executed. Instead this would be controlled through persistent controls which are referred as Boot options in this document.

[SWS_UCM_00094]{DRAFT} **Management of executable software** ⌈UCM shall manage which version of software is available for the Execution Management [2] to launch.⌋*(RS_UCM_00021)*

During the kActivating state UCM modifies the Boot options so that in the next restart for the updated software the new versions will be executed. In the kRolling-Back state UCM modifies the Boot options so that in the next restart of the updated software the original versions will be executed. Successful exit from kActivating and kRollingBack states is triggered by the FunctionGroupState from State Management.

[SWS_UCM_00096]{DRAFT} **Entering the Rolled-back state** ⌈UCM shall switch from kRollingBack state into the kRolledBack state when FunctionGroupState from State Management [4] indicates that original software has successfully reached the Application state kRunning.⌋*(RS_UCM_00008, RS_UCM_00030)*

7.1.6.4 Finishing activation

[SWS_UCM_00020]{DRAFT} **Finishing the packages activation** ⌈UCM shall provide a method Finish to commit all the changes and clean up all temporary data of the packages processed.

UCM should also remove Software Packages, logs or any older versions of changed software to save storage space. It is up to implementer to remove or not the Software Packages.⌋*(RS_UCM_00015)*

For UCM to be able to free all unneeded resources while processing the Finish request, it is up to the vendor and platform specific implementation to make sure that obsolete versions of changed SoftwareClusters aren't executed anymore.

7.1.7 Robustness against reset

Failure during over-the-air updates could lead into corrupted or inconsistent software configuration and further updates might be blocked. For this reason UCM needs to be robust against interruptions like power downs.

[SWS_UCM_00157]{DRAFT} **Detection of reset** ⌈At start up UCM shall identify if uncontrolled reset occurred.⌋*(RS_UCM_00027)*

[SWS_UCM_00158]{DRAFT} **Cleanup of interrupted actions** ⌈After uncontrolled reset UCM shall perform cleanup and return the system into consistent state, from where the client can continue in a controlled manner.⌋*(RS_UCM_00027)*

7.1.7.1 Boot monitoring

Activation failure during OS and Platform-self updates can lead to a state in which the system is not able to reach a point where UCM and the client are able to function as expected and thus not able to execute the rollback. For these cases the system should include component which is responsible to monitor that the OS and platform will start

up correctly. In case of failure, the Boot monitoring component should trigger a reset or modify the boot options to trigger a rollback.

7.1.8 Logging and history

[SWS_UCM_00170]{DRAFT} **Log message retrieving** ⌈UCM shall provide a method GetLog to retrieve all log messages that have been stored by UCM.⌋*(RS_UCM_00022)*

[SWS_UCM_00143]{DRAFT} **Log level setting** ⌈UCM shall provide a method SetLogLevel to provide a log level for all subsequent log messages that are stored by UCM.⌋*(RS_UCM_00022)*

[SWS_UCM_00171]{DRAFT} **Log level changing** ⌈Calling SetLogLevel shall immediatly lead to change of loglevel, even in the middle of any UCM action.⌋*(RS_UCM_00022)*

[SWS_UCM_00172]{DRAFT} **Log messages removing** ⌈All log messages that are stored by UCM shall be removed within the Finish call.⌋*(RS_UCM_00022)*

[SWS_UCM_00115]{DRAFT} **History** ⌈UCM shall provide a method GetHistory to retrieve all actions that have been performed by UCM in a specific time window.⌋*(RS_UCM_00032)*

[SWS_UCM_00160]{DRAFT} **Processing results records** ⌈UCM shall save activation time and activation result of processed Software Packages in the history.⌋*(RS_UCM_00032)*

7.1.9 Version Reporting

[SWS_UCM_00004]{DRAFT} **Report software information** ⌈UCM shall provide a method GetSwClusterInfo of the interface service PackageManagement to provide the identifiers and versions of the SoftwareClusters that are in state kPresent.⌋*(RS_UCM_00002)*

[SWS_UCM_00030]{DRAFT} **Report changes** ⌈UCM shall provide a method GetSwClusterChangeInfo of the interface service PackageManagement to provide the identifiers and versions of the SoftwareCluster that are in state kAdded, kUpdated or kRemoved.⌋*(RS_UCM_00011)*

7.1.10 SoftwareCluster lifecycle

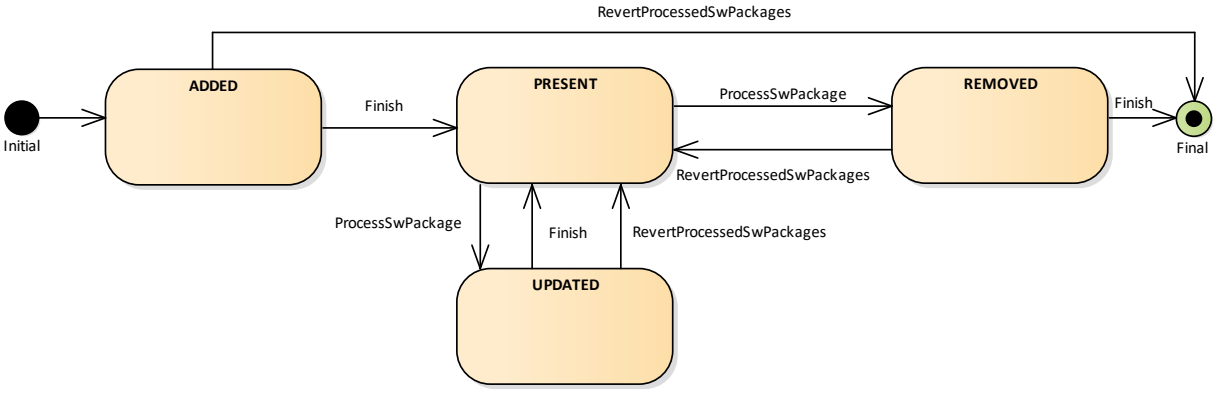

Figure 7.5: State Machine for a `SoftwareCluster`

The state machine in Fig. 7.5 describes the states of a `SoftwareCluster`. After processing a `Software Package` with a new `SoftwareCluster` that was not yet existing on the `AUTOSAR Adaptive Platform`, the new `SoftwareCluster` starts its lifecycle with state `kAdded`. After finishing update process with method `Finish`, it is in state `kPresent`. In another update process, by processing a `Software Package` with new data for the `SoftwareCluster`, it changes to `kUpdated` and returns to `kPresent` once update process has finished. If a `Software Package` is processed and it involves the deletion of an existing `SoftwareCluster` the state changes to `kRemoved`. `Finish` commits the change and the removed `SoftwareCluster` is not reported by `UCM` any more.

The state machine in Fig. 7.5 describes the states of a `SoftwareCluster`. After processing a `Software Package` with a new `SoftwareCluster` that was not yet existing on the `AUTOSAR Adaptive Platform`, the new `SoftwareCluster` starts its lifecycle with state `kAdded`. After finishing update process with method `Finish`, it is in state `kPresent`. In another update process, by processing a `Software Package` with new data for the `SoftwareCluster`, it changes to `kUpdated` and returns to `kPresent` once update process has finished. If a `Software Package` is processed and it involves the deletion of an existing `SoftwareCluster` the state changes to `kRemoved`. `Finish` commits the change and the removed `SoftwareCluster` is not reported by `UCM` any more.

7.1.11 Securing Software Updates

`UCM` provides service interface using `ara::com`. There is no authentication of the client in `UCM`'s update sequence.

For authentication of the `Software Package`, you can refer to 7.1.2

[SWS_UCM_00103]{DRAFT} **Update to older `Software Cluster` version than currently present** ⌈In order to avoid an attacker to install an old `Software Cluster` version having known security flaws, `UCM` shall prohibit its processing. In case

of such attempt, UCM TransferExit shall raise the ApplicationError OldVersion, keep within history this tentative and delete old Software Package.⌋*(RS_UCM_00031)*

[SWS_UCM_00105]{DRAFT} **UCM confidential information handling** ⌈The methods GetSwClusterInfo, GetSwClusterChangeInfo, GetLog, GetHistory and GetSwPackages shall only be called over secure communication channel providing confidentiality protection.⌋*(RS_UCM_00002, RS_UCM_00010, RS_UCM_00011)*

7.1.12 Functional cluster lifecycle

7.1.13 Shutdown behaviour

There are no requirements of shutdown behaviour from UCM functional cluster.

7.2 UCM Master

7.2.1 Technical Overview

UCM Master objective is to provide a standard Adaptive Autosar solution to safely and securely update a complete vehicle Over The Air or by a Diagnostic Tester.

UCM Master receives packages from Backend or Diagnostic tool, parses and interprets the Vehicle Package, transfers or streams Software Packages to suitable target (UCM subordinate or Diagnostic Application) and orchestrates the processing, activations and eventual rollbacks. All these actions are what is called an update campaign which UCM Master is coordinating.

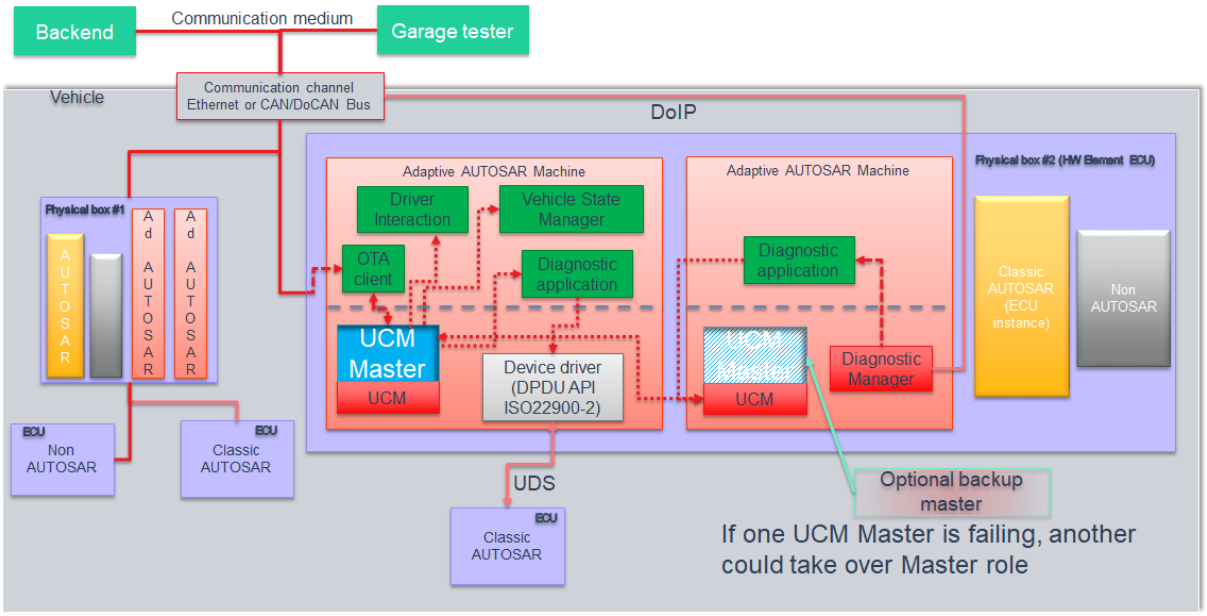

Figure 7.6: Example of UCM Master architecture overview within a vehicle

The UCM Master could be considered as a set of add-on features that could enrich any UCM instance. Therefore, as per the UCM APIs, the UCM Master APIs are part of the Adaptive Platform Services. UCM and UCM Master have separate service instances.

When communication is established between Backend and UCM Master by OTA Client, versions of installed Software Clusters in vehicle can be exchanged between these two components. This communication could be triggered by OTA Client with a scheduler or by Backend to push for instance an important security update to a fleet of vehicles. The computation to find new Software Clusters versions and resolution of dependencies between Software Clusters can be either done at UCM Master or Backend.

Vehicle Driver interface Adaptive Application is required if it is needed during an update campaign to interact with vehicle human driver through for instance Human-Machine Interface. Download of packages from a Backend could have various financial costs for the driver depending of communication types, so consent from driver could be suitable.

Vehicle State Manager Adaptive Application is required if it is needed during an update campaign to control the vehicle state for safety purposes. For instance, it could be required for safety to have standing still vehicle, shut-off engine, closed doors, etc. before starting an UCM activation or during its processing.

7.2.2 UCM Master general behaviour

The UCM Master runs as a separate service instance besides other UCM services. It uses these offered UCM services as a supervisor to allow performance of update campaigns. To achieve this behaviour, the UCM specification is extended. As an UCM Master generally acts as a UCM client, it uses the already specified UCM API. UCM Master aggregates UCM subordinates states and can report its status field to a Backend trough its OTA Client.

An UCM Master receives a Vehicle Package and transfers or streams Software Package(s) to the UCM subordinates for an AUTOSAR Adaptive Platform Software Cluster update. A Vehicle Package contains instructions for orchestrating updates between ECUs. The UCM Master provides information about ECUs in the vehicle, installed software and update campaign resolution.

[SWS_UCM_01001]{DRAFT} **UCM Master processes Vehicle Package** ⌈An UCM Master shall receive a Vehicle Package and transfers corresponding Software Package(s) to its UCM subordinates.⌋ *(RS_UCM_00039, RS_UCM_00043)*.

[SWS_UCM_01002]{DRAFT} **UCM Master shall provide UCM services** ⌈It is currently not foreseen to have a UCM acting only as a UCM Master, which would imply that this UCM Master would require another UCM instance to perform the update of its AUTOSAR Adaptive Platform.⌋ *(RS_UCM_00036)*.

[SWS_UCM_01003]{DRAFT} **UCM Master checks states of UCM subordinates** ⌈An UCM Master shall check the populated status of its UCM subordinates to make sure no interfering update is currently ongoing.⌋ *(RS_UCM_00043)*

UCM Master should for instance make sure that there is no ongoing diagnostic updates before starting an update campaign by checking the reported state(s) of the UCM subordinate(s) to be idle.

[SWS_UCM_01004]{DRAFT} **Only one UCM Master shall be active per network domain** ⌈As UCM Master is distributing Software Packages and coordinating UCM subordinates, no other UCM Master shall be active within a network domain in order to avoid any interferences and guaranty success of an update campaign.⌋ *(RS_UCM_00037)*

7.2.3 UCM identification

For UCM Master to distribute Software Packages to other UCM subordinates, UCM Master has to identify UCM subordinates in vehicle. This identification could be at boot or later but at least before any communication with Backend are engaged. Each UCM has a unique identifier in Vehicle Package ucmModuleInstantiation called identifier to help UCM Master transferring packages to targeted UCMs. To get such identifier, UCM Master will perform first a service discovery through ara::com to get all UCMs service instances available. Then UCM Master will call GetId method for each UCM subordinates returning each corresponding ucmModuleInstantiation identifiers.

[SWS_UCM_00009]{DRAFT} **UCM exposing its identifier** ⌈UCM shall provide a method GetId returning its ucmModuleInstantiation identifier.⌋*(RS_UCM_00036)*

If an ECU hosting UCM subordinate is replaced physically, it will register its services to the registry at boot up and UCM Master will be able to communicate with UCM subordinate(s).

[SWS_UCM_01005]{DRAFT} **UCM Master is discovering UCMs in vehicle** ⌈UCM Master shall continuously look for UCM service instances (use of StartFindService() call).⌋*(RS_UCM_00036)*

If a UCM Master is failing, another inactive UCM Master could be used or activated by OTA Client.

Default (at boot) Master/Subordinate hierarchy or priority could be optionally overwritten for each campaign based on Vehicle Package content at the condition OTA Client could properly parse Vehicle Packages.

7.2.4 UCM Master Software Packages transfer or streaming

UCM Master has generally same transfer API as UCM in order to simplify implementation and reuse code as much as possible (could be shared library between UCM and UCM Master).

[SWS_UCM_01006]{DRAFT} **Vehicle Package transfer to UCM Master** ⌈UCM Master shall provide method transferVehiclePackage via ARA::COM to OTA Client.⌋*(RS_UCM_00035, RS_UCM_00043)* It is necessary to distinguish Vehicle Package (UCM Master specific) from Software Packages transfer.

[SWS_UCM_01007]{DRAFT} **Start transfer of a Vehicle Package or Software Packageto UCM Master** ⌈UCM Master shall provide method transferStart via ARA::COM to OTA Client.⌋*(RS_UCM_00035, RS_UCM_00036)*

[SWS_UCM_01008]{DRAFT} **Transfer data of a Vehicle Package to UCM Master** ⌈UCM Master shall provide method transferData via ARA::COM to OTA Client.⌋*(RS_UCM_00035, RS_UCM_00036)*

[SWS_UCM_01009]{DRAFT} **Exit the transfer of a Vehicle Package to UCM Master** ⌈UCM Master shall provide method transferExit via ARA::COM to OTA Client.⌋*(RS_UCM_00035, RS_UCM_00036)*

[SWS_UCM_01010]{DRAFT} **Delete a Vehicle Package transferred to UCM Master** ⌈UCM Master shall provide method deleteTransfer via ARA::COM to OTA Client.⌋*(RS_UCM_00035, RS_UCM_00036)*

7.2.5 Adaptive Applications interacting with UCM Master

In order to have interoperability between several vendors platforms, Adaptive Applications interacting with UCM Master via ara::com like OTA Client, Vehicle State Manager or Vehicle Driver Interface have their APIs specified. However, their detailed behaviours are out of scope for this specification document.

7.2.5.1 OTA Client

OTA Client is an Adaptive Application that sets communication channel between Backend and UCM Master. It uses the UCM Master as a service provider via ARA::COM. The communication between Backend and OTA Client is abstracted and details like protocol are out of scope for this specification document. OTA Client shall make sure Backend is providing the right information and packages to the vehicle by identifying the vehicle, by for instance sending VIN to Backend.

[SWS_UCM_01101]{DRAFT} **Provide information of installed Software Clusters in vehicle** ⌈UCM Master shall provide a method GetSwClusterInfo to return information of all Software Cluster present in the vehicle.⌋*(RS_UCM_00033)*

UCM Master can aggregate Software Cluster information from several UCMs within a vehicle and returns the result to a Backend which can compute if there is any new Software Cluster available and decide to send to UCM Master through OTA Client a Vehicle Package.

[SWS_UCM_01102]{DRAFT} **Get information of available Software Clusters in Backend** ⌈UCM Master shall provide a method SwPackageInventory which argument contains information Software Clusters present in Backend for the vehicle.⌋*(RS_UCM_00033)*

[SWS_UCM_01103]{DRAFT} **Inform Backend of needed Software Clusters for an update** ⌈After UCM Master receives with SwPackageInventory call the information of available Software Clusters present in Backend for the vehicle, UCM Master shall compute based on its own internal information of present Software Clusters in vehicle what are the new Software Clusters available at Backend and return to it.⌋*(RS_UCM_00033)*

7.2.5.2 Vehicle Driver Interface

Vehicle driver interface could be required by legal constrains or communication cost consideration. To support mandatory safety and security critical updates, driver interaction can be used for:

- Requesting transfer, processing or activation permission from vehicle driver
- Notifying vehicle driver of safety and security measures he has to apply to the vehicle in order to proceed to next step into the update campaign

[SWS_UCM_01105]{DRAFT} **Interaction of `UCM Master` with Vehicle Driver** ⌈Vehicle Driver Interface `Adaptive Application` shall provide to `UCM Master` a method `DriverNotification` over ARA::COM in order for `UCM Master` to inform the driver at what state is the update, if approval is required or what safety measures shall be applied to the vehicle in order to continue towards next update state.⌋*(RS_UCM_00038)*

After a call of `DriverNotification`, `UCM Master` waits for an approval before going to next step in the update campaign.

[SWS_UCM_01106]{DRAFT} **Exclusive use of Vehicle Driver Interface** ⌈Vehicle Driver Interface shall ensure that `UCM Master` is the exclusive user of the `DriverNotification` method at any time during a software update campaign.⌋*(RS_UCM_00035, RS_UCM_00037)*

For example, the integrator may restrict the access to Vehicle Driver Interface in configuring the Identity and Access Management functional cluster accordingly.

[SWS_UCM_01107]{DRAFT} **`UCM Master` provides progress information to Vehicle Driver** ⌈`UCM Master` shall provide to Vehicle Driver Interface `Adaptive Application` methods `GetSwTransferProgress` and `GetSwProcessProgress` in order for `UCM Master` to inform progress of respectively update campaign's transfer and processing.⌋*(RS_UCM_00038)*

[SWS_UCM_01108]{DRAFT} **Unsupported safety policy by Vehicle driver interface** ⌈In the case method `DriverNotification` is called with an unsupported safety policy argument, Vehicle driver interface shall raise the `ApplicationError` `notSupportedSafetyPolicy`.⌋*(RS_UCM_00037)*

7.2.5.3 Vehicle State Manager

Vehicle State Manager is collecting states from the several vehicle `ECU`s and provides to `UCM Master` a field to subscribe, a judgement against the safety policy referred in the `Vehicle Package`. If the safety policy is not met, the `UCM Master` can for instance decide to:

- Inform vehicle driver that the safety conditions are not met to continue the update
- postpone, pause or cancel the update until policy is met

[SWS_UCM_01109]{DRAFT} **Vehicle State Manager shall provide to `UCM Master` a safety state** ⌈`UCM Master` shall provide to Vehicle State Manager `Adaptive Application` a field `SafeToUpdate`.⌋*(RS_UCM_00037)*

[SWS_UCM_01110]{DRAFT} **`UCM Master` shall be able to set the safety policy to be computed by Vehicle State Manager** ⌈Vehicle State Manager `Adaptive Application` shall provide to `UCM Master` a method `ApplyPolicy` via ARA::COM setting suitable safety policy to be computed by Vehicle State Manager.⌋*(RS_UCM_00037)*

[SWS_UCM_01111]{DRAFT} **Exclusive use of Vehicle State Manager** ⌈Vehicle State Manager shall ensure that `UCM Master` is the exclusive user of the `ApplyPolicy` method at any time during a software update campaign.⌋*(RS_UCM_00035, RS_UCM_00037)*

For example, the integrator may restrict the access to Vehicle State Manager in configuring the Identity and Access Management functional cluster accordingly.

[SWS_UCM_01112]{DRAFT} **Unsupported safety policy by Vehicle State Manager** ⌈In the case method `ApplyPolicy` is called with an unsupported safety policy argument, Vehicle State Manager shall raise the `ApplicationError` `notSupportedSafetyPolicy`.⌋*(RS_UCM_00037)*

[SWS_UCM_01113]{DRAFT} **Switching vehicle into update mode** ⌈Vehicle State Manager shall change vehicle's state and its `ECUs` in the right update mode in order to avoid any timeout issues during update.⌋*(RS_UCM_00037)* This vehicle state change could be triggered based on UCM Master State Machine.

7.2.6 Status reporting

UCM Master supports a mechanism to provide the state of an update campaign typically to OTA Client, Vehicle Driver Application and Vehicle State Manager.

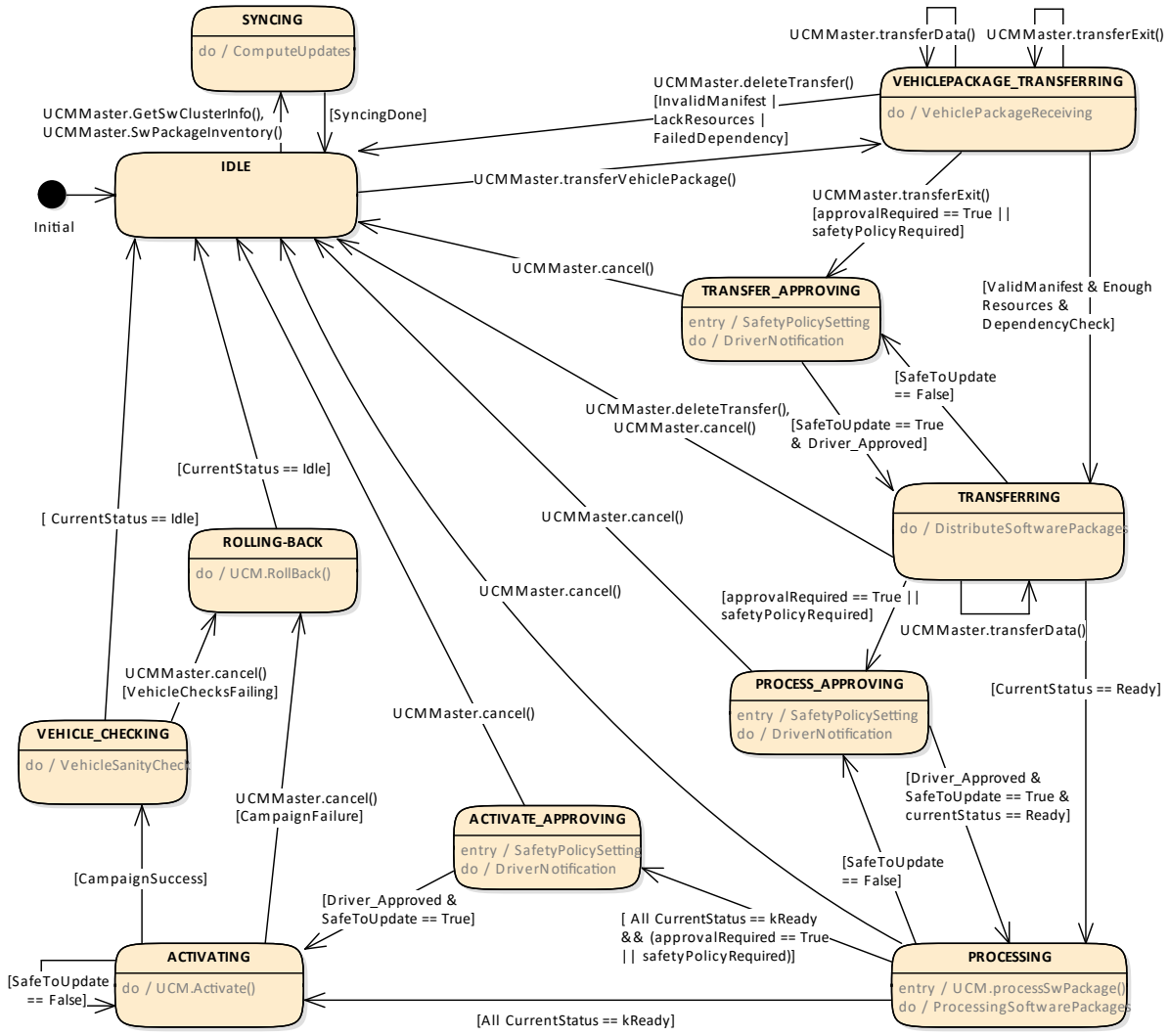

Figure 7.7: Campaign State Machine

[SWS_UCM_01201]{DRAFT} **Sequential orchestration of campaigns** ⌈UCM Master shall orchestrate at most a single campaign at any one time.⌋*(RS_UCM_00043)*

[SWS_UCM_01203]{DRAFT} **CampaignState field** ⌈UCM Master shall provide the state of a campaign over the CampaignState field of the UCM Master ProvidedPort.⌋*(RS_UCM_00042)* There is an overview of the campaign state machine in Fig. 7.7 detailing UCM Master campaign states and transitions.

7.2.6.1 States

[SWS_UCM_01204]{DRAFT} **Initial state** ⌈kIdle shall be the initial state at UCM Master startup if no recovery is required.⌋*(RS_UCM_00035)*

[SWS_UCM_01205]{DRAFT} **UCM Master internal state persistency** ⌈UCM Master shall persist its state to be able to resume on-going update campaign after an intended or unintended reboot.⌋*(RS_UCM_00035, RS_UCM_00042)*

[SWS_UCM_01206]{DRAFT} **Trigger on kTransferApproving state** ⌈On transition to kTransferApproving state, UCM Master shall request if required (campaign orchestration) approval for transferring (DriverNotification) from the Vehicle Driver Application and request if required (campaign orchestration) safety policy enforcement for transferring (ApplyPolicy) from Vehicle State Manager.⌋*(RS_UCM_00035, RS_UCM_00038)*

[SWS_UCM_01207]{DRAFT} **Trigger on kTransferring state** ⌈On transition to kTransferring state and if all UCM subordinates part of the campaign are in kIdle state, UCM Master shall start or resume transferring (TransferStart and TransferData as well as TransferExit if no streaming required) the software packages to the UCM subordinates according to the campaign orchestration.⌋*(RS_UCM_00035, RS_UCM_00043)*

[SWS_UCM_01208]{DRAFT} **Trigger on kProcessApproving state** ⌈On transition to kProcessApproving state, UCM Master shall request if required (campaign orchestration) approval for processing (DriverNotification) from the Vehicle Driver Application and request if required (campaign orchestration) safety policy enforcement for processing (ApplyPolicy) from Vehicle State Manager.⌋*(RS_UCM_00035, RS_UCM_00038)*

[SWS_UCM_01209]{DRAFT} **Trigger on kProcessing state** ⌈On transition to kProcessing state, UCM Master shall start or resume processing the software packages (ProcessSwPackage) ready for processing according to the campaign orchestration.⌋*(RS_UCM_00035, RS_UCM_00043)*

[SWS_UCM_00210]{DRAFT} **Transferring of software packages on kProcessApproving or kProcessing state** ⌈If UCM Master is in kProcessApproving or kProcessing state, UCM Master shall transfer software packages to the UCM subordinates according to the campaign orchestration.⌋*(RS_UCM_00035, RS_UCM_00043)*

[SWS_UCM_01211]{DRAFT} **Trigger on kActivateApproving state** ⌈On transition to kActivateApproving state, UCM Master shall request if required (campaign orchestration) approval for activating (DriverNotification) from the Vehicle Driver Application and request if required (campaign orchestration) safety policy enforcement for activating (ApplyPolicy) from Vehicle State Manager.⌋*(RS_UCM_00035, RS_UCM_00038)*

[SWS_UCM_01212]{DRAFT} **Trigger on `kActivating` state** ⌈On transition to `kActivating` state, `UCM Master` shall activate the software (`Activate`) according to the campaign orchestration.⌋*(RS_UCM_00035, RS_UCM_00043)*

[SWS_UCM_01213]{DRAFT} **Trigger on `kVehicleChecking` state** ⌈On transition to `kVehicleChecking` state, `UCM Master` shall first perform checks (OEM specific) to assess the post-activation state of the vehicle.⌋*(RS_UCM_00035)*

`UCM Master` may be responsible for performing post-activation checks, interfacing with an application performing such checks, confirming backend is still reachable and further updates are still possible.

[SWS_UCM_01214]{DRAFT} **Final action on `kVehicleChecking` state** ⌈If `UCM Master` is in `kVehicleChecking` state and the post-activation checks (OEM specific) are successful, `UCM Master` shall secondly commit (`Finish`) the software on all UCM subordinates part of the campaign.⌋*(RS_UCM_00035)*

[SWS_UCM_01215]{DRAFT} **Trigger on `kRollingBack` state** ⌈On transition to `kRollingBack` state, `UCM Master` shall first rollback (`RollingBack`) the software on all UCM subordinates part of the campaign.⌋*(RS_UCM_00035)*

[SWS_UCM_01216]{DRAFT} **Final action on `kRollingBack` state** ⌈If `UCM Master` is in `kRollingBack` state and the rollback of software on all UCM subordinates is successful (successful `RollingBack` and transition from `kRollingBack` to `kRolledBack`), `UCM Master` shall secondly commit (`Finish`) the software on all UCM subordinates part of the campaign.⌋*(RS_UCM_00035)*

[SWS_UCM_01217]{DRAFT} **Monitoring of UCM subordinates** ⌈`UCM Master` shall monitor the state of the UCM subordinates during a campaign.⌋*(RS_UCM_00035)*

7.2.6.2 States Transitions

[SWS_UCM_01218]{DRAFT} **Transition from `kIdle` state to `kSyncing` state** ⌈If `UCM Master` is in `kIdle` state, `UCM Master` shall enter the `kSyncing` state on a request to `GetSwClusterInfo` or `SwPackageInventory`.⌋*(RS_UCM_00035, RS_UCM_00033)*

[SWS_UCM_01219]{DRAFT} **Transition from `kSyncing` state to `kIdle` state** ⌈If `UCM Master` is in `kSyncing` state, `UCM Master` shall enter the `kIdle` state on completion of `GetSwClusterInfo` or `SwPackageInventory`.⌋*(RS_UCM_00035)*

[SWS_UCM_01220]{DRAFT} **Transition from `kIdle` state to `kVehiclePackageTransferring` state** ⌈If `UCM Master` is in `kIdle` state, `UCM Master` shall enter the `kVehiclePackageTransferring` state on successful completion of `TransferVehiclePackage`.⌋*(RS_UCM_00035)*

[SWS_UCM_01221]{DRAFT} **Transition from `kVehiclePackageTransferring` state to `kIdle` state** ⌈If `UCM Master` is in `kVehiclePackageTransferring` state,

UCM Master shall enter the kIdle state on unsuccessful completion of TransferExit (Vehicle Package) or successful completion of DeleteTransfer (Vehicle Package).⌋*(RS_UCM_00035, RS_UCM_00039)*

[SWS_UCM_01222]{DRAFT} **Transition from kVehiclePackageTransferring state to kTransferring state** ⌈If UCM Master is in kVehiclePackageTransferring state, UCM Master shall enter the kTransferring state on successful completion of TransferExit (Vehicle Package) if no driver approval or safety policy enforcement is required for transferring.⌋*(RS_UCM_00035, RS_UCM_00037, RS_UCM_00038)*

[SWS_UCM_01223]{DRAFT} **Transition from kVehiclePackageTransferring state to kTransferApproving state** ⌈If UCM Master is in kVehiclePackageTransferring state, UCM Master shall enter the kTransferApproving state on successful completion of TransferExit (Vehicle Package) and driver approval or safety policy enforcement is required for transferring.⌋*(RS_UCM_00035, RS_UCM_00037, RS_UCM_00038)*

[SWS_UCM_01224]{DRAFT} **Transition from kTransferApproving state to kTransferring state** ⌈If UCM Master is in kTransferApproving state, UCM Master shall enter the kTransferring state on a positive feedback (return value of DriverNotification is true) from the Vehicle Driver Application if driver approval is required and on a positive feedback (value of SafeToUpdate field is true) from the Vehicle State Manager if safety policy enforcement is required.⌋*(RS_UCM_00035, RS_UCM_00037, RS_UCM_00038)*

[SWS_UCM_01225]{DRAFT} **Transition from kTransferApproving state to kIdle state** ⌈If UCM Master is in kTransferApproving state, UCM Master shall enter the kIdle state on successful cancellation request (Cancel) and completion.⌋*(RS_UCM_00035)*

[SWS_UCM_01226]{DRAFT} **Transition from kTransferring state to kTransferApproving state** ⌈If UCM Master is in kTransferring state, UCM Master shall enter the kTransferApproving state on a negative feedback (value of SafeToUpdate field is false) from the Vehicle State Manager if safety policy enforcement is required.⌋*(RS_UCM_00035, RS_UCM_00037)*

[SWS_UCM_01227]{DRAFT} **Transition from kTransferring state to kIdle state** ⌈If UCM Master is in kTransferring state, UCM Master shall enter the kIdle state on successful cancellation request (Cancel) and completion.⌋*(RS_UCM_00035)*

[SWS_UCM_01228]{DRAFT} **Transition from kTransferring state to kProcessing state** ⌈If UCM Master is in kTransferring state and any software packages are ready for processing (successful completion of TransferExit or processing started by ProcessSwPackage call) according to the campaign orchestration, UCM Master shall enter the kProcessing state if no driver approval of safety policy enforcement is required.⌋*(RS_UCM_00035, RS_UCM_00037, RS_UCM_00038, RS_UCM_00043)*

[SWS_UCM_01229]{DRAFT} **SafetyPolicy while processing stream** ⌈In the case there is transition from `kTransferring` state to `kProcessing` state, the SafetyPolicy for `kProcessing` state shall apply even though there are `Software Packages` transferring.⌋*(RS_UCM_00035, RS_UCM_00037)* Integrator should make sure in this use case that safety policy for Processing will also cover safety approach of transferring.

[SWS_UCM_01230]{DRAFT} **Transition from `kTransferring` state to `kProcessApproving` state** ⌈If `UCM Master` is in `kTransferring` state and any software packages are ready for processing (successful completion of `TransferExit` or streaming required) according to the campaign orchestration, `UCM Master` shall enter the `kProcessing` state if driver approval of safety policy enforcement is required.⌋*(RS_UCM_00035, RS_UCM_00037, RS_UCM_00038, RS_UCM_00043)*

[SWS_UCM_01231]{DRAFT} **Transition from `kProcessApproving` state to `kProcessing` state** ⌈If `UCM Master` is in `kProcessApproving` state, `UCM Master` shall enter the `kProcessing` state on a positive feedback (return value of `DriverNotification` is `true`) from the `Vehicle Driver Application` if driver approval is required and on a positive feedback (value of `SafeToUpdate` field is `true`) of the `Vehicle State Manager` if safety policy enforcement is required.⌋*(RS_UCM_00035, RS_UCM_00037, RS_UCM_00038)*

[SWS_UCM_01232]{DRAFT} **Transition from `kProcessApproving` state to `kIdle` state** ⌈If `UCM Master` is in `kProcessApproving` state, `UCM Master` shall enter the `kIdle` state on successful cancellation request (`Cancel`) and completion.⌋*(RS_UCM_00035)*

[SWS_UCM_01233]{DRAFT} **Transition from `kProcessing` state to `kProcessApproving` state** ⌈If `UCM Master` is in `kProcessing` state, `UCM Master` shall enter the `kProcessApproving` state on a negative feedback (value of `SafeToUpdate` field is `false`) from the `Vehicle State Manager` if safety policy enforcement is required.⌋*(RS_UCM_00035)*

[SWS_UCM_01234]{DRAFT} **Transition from `kProcessing` state to `kActivating` state** ⌈If `UCM Master` is in `kProcessing` state and all software packages of the campaign have been successfully (successful `ProcessSwPackage`) processed and all UCM subordinates part to the campaign are in the `kReady` state, `UCM Master` shall enter the `kActivating` state if no driver approval or safety policy enforcement is required.⌋*(RS_UCM_00035, RS_UCM_00037, RS_UCM_00038)*

[SWS_UCM_01235]{DRAFT} **Transition from `kProcessing` state to `kActivateApproving` state** ⌈If `UCM Master` is in `kProcessing` state and all software packages of the campaign have been successfully (successful `ProcessSwPackage`) processed and all UCM subordinates part to the campaign are in the `kReady` state, `UCM Master` shall enter the `kActivateApproving` state if driver approval or safety policy enforcement is required.⌋*(RS_UCM_00035, RS_UCM_00037, RS_UCM_00038)*

[SWS_UCM_01236]{DRAFT} **Transition from `kProcessing` state to `kIdle` state** ⌈If UCM Master is in kProcessing state, UCM Master shall enter the kIdle state on successful cancellation request (Cancel) and completion.⌋ *(RS_UCM_00035)*

[SWS_UCM_01237]{DRAFT} **Transition from `kActivateApproving` state to `kActivating` state** ⌈If UCM Master is in kActivateApproving state, UCM Master shall enter the kActivating state on a positive feedback (return value of DriverNotification is true) from the Vehicle Driver Application if driver approval is required and on a positive feedback (value of SafeToUpdate field is true) from the Vehicle State Manager if safety policy enforcement is required.⌋ *(RS_UCM_00035, RS_UCM_00037, RS_UCM_00038)*

[SWS_UCM_01238]{DRAFT} **Transition from `kActivateApproving` state to `kIdle` state** ⌈If UCM Master is in kActivateApproving state, UCM Master shall enter the kIdle state on successful cancellation request (Cancel) and completion.⌋ *(RS_UCM_00035, RS_UCM_00037)*

[SWS_UCM_01239]{DRAFT} **Transition from `kActivating` state to `kRollingBack` state** ⌈If UCM Master is in kActivating state, UCM Master shall enter the kRollingBack state if any UCM subordinates part of the campaign unsuccessfully (unsuccessful Activate and transition from kVerifying to kRollingBack) completed activation.⌋ *(RS_UCM_00035)*

[SWS_UCM_01240]{DRAFT} **Transition from `kActivating` state to `kVehicleChecking` state** ⌈If UCM Master is in kActivating state, UCM Master shall enter the kRollingBack state if all UCM subordinates part of the campaign successfully (successful Activate and transition from kVerifying to kActivated) completed activation.⌋ *(RS_UCM_00035, RS_UCM_00037)*

[SWS_UCM_01241]{DRAFT} **Transition from `kVehicleChecking` state to `kRollingBack` state** ⌈If UCM Master is in kVehicleChecking state and the post-activation checks (OEM specific) are unsuccessful, UCM Master shall enter the kRollingBack state.⌋ *(RS_UCM_00035)*

[SWS_UCM_01242]{DRAFT} **Transition from `kVehicleChecking` state to `kIdle` state** ⌈If UCM Master is in kVehicleChecking state and all UCM subordinates part of the campaign transitioned from kCleaningUp to kIdle, UCM Master shall enter the kIdle state.⌋ *(RS_UCM_00035)*

[SWS_UCM_01243]{DRAFT} **Transition from `kRollingBack` state to `kIdle` state** ⌈If UCM Master is in kRollingBack state and all UCM subordinates part of the campaign transitioned from kCleaningUp to kIdle, UCM Master shall enter the kIdle state.⌋ *(RS_UCM_00035)*

[SWS_UCM_01244]{DRAFT} **Cancellation of an update campaign shall be possible** ⌈UCM Master shall provide method Cancel to any of its client to cancel from kTransferring, kTransferApproving, kProcessApproving or kProcessing).⌋ *(RS_UCM_00035, RS_UCM_00037)*

Cancel method could be used at garage to unlock a blocked update. Details on action by UCM Master, like cleaning up the several UCMs, changing AUTOSAR Adaptive Platform states, etc. are implementation specific.

[SWS_UCM_01245]{DRAFT} **Cancellation during activation shall be possible** ⌈UCM Master shall provide method Cancel to any of its client to cancel from kActivating or kActivateApproving).⌋*(RS_UCM_00035, RS_UCM_00037)*

In case an update campaign was cancelled, a new update campaign could use again the already transferred Software Packages. UCM Master could list transferred Software Packages by calling the UCM subordinates with getSwPackages.

[SWS_UCM_01246]{DRAFT} **Unreachable UCM during update campaign** ⌈In case a UCM is not reachable by UCM Master during an update campaign (from kTransferring, kTransferApproving, kProcessApproving or kProcessing), UCM Master shall cancel and go back to kIdle.⌋*(RS_UCM_00035, RS_UCM_00037)*

7.2.7 Campaign Reporting

After campaign is finished (finish method has been sent to all UCM subordinates), UCM Master should report to Backend server status of the vehicle, with for instance updated information of Software Clusters present in vehicle.

[SWS_UCM_01247]{DRAFT} **Method to read History Report** ⌈UCM Master shall provide a method GetHistory to retrieve all actions that have been performed by UCM Master in a specific time window.⌋*(RS_UCM_00034)*

[SWS_UCM_01248]{DRAFT} **Content of History Report** ⌈UCM Master shall save activation time and activation result of processed Vehicle Packages in the history.⌋*(RS_UCM_00034)*

7.2.8 Content of Vehicle Package

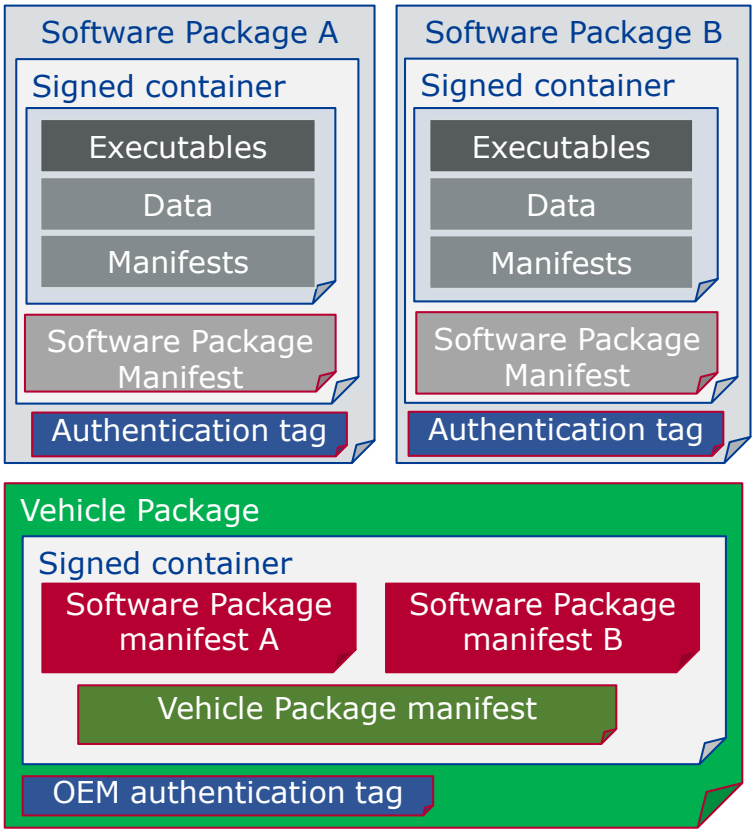

Figure 7.8: Vehicle package overview

A Vehicle Package is typically assembled by an OEM Backend. A Vehicle Package has to be modelled as a so-called VehiclePackage which describes the content of the Vehicle Package. It contains a collection of Software Package Manifests extracted from Backend packages stored in the Backend database. These Software Packages have to be modelled as a so-called SoftwarePackage which describes the content of the Software Package. A Vehicle Package contains only one Vehicle Package Manifest.

It is possible that within an update campaign, several Machine or ECUs need to be updated/installed/removed by groups. Some Software Clusters could require reboot of Machine or ECU, some just a restart of Adaptive Application or nothing (waiting passively for next reboot) to get activated. To optimize a campaign or fulfil dependencies, it could be required to activate Software Clusters one after the other or several at once. To support all possible campaigns, the Vehicle Package includes a model describing this coordination. It also contains a way to identify the several involved UCMs for packages distribution within the vehicle and potentially overwriting default UCM Master for this specific campaign.

You can find below for information purpose a description of the information that must be contained in Vehicle Package manifest:

- Dependencies: dependencies between Software Clusters that will overrule the already defined dependencies in Software Package Manifest. Typically used by vehicle systems integrator to add dependencies related to vehicle systems that Software Package supplier is not aware of. Modelled by inheritance of SoftwareActivationDependency, also used by SoftwareCluster.
- Repository: uri, repository or diagnostic address, for history, tracking and security purposes
- Vehicle description: vehicle description
- Vehicle Driver notifications: it might be needed to ask vehicle driver if UCM Master can start transferring Software Packages, processing it and activating it but also inform him of the necessary safety requirements if applicable.
- Safety policy: safety policy index to be used as argument to subscribe a field to vehicle safety manager. With this field, UCM Master will be informed at any time of campaign if vehicle safety is met or not.
- UCM Master identifiers list: defines backup UCM Masters
- Campaign orchestration: You can refer to [9] for more details. This campaign model allows to group activation of several UCMs and group Software Packages processing and transferring.

[SWS_UCM_01301]{DRAFT} **Vehicle Package authentication** ⌈Vehicle Package shall be authenticated by UCM Master before any transfer of Software Packages.⌋*(RS_UCM_00039, RS_UCM_00043)*

[SWS_UCM_01302]{DRAFT} **Vehicle Package authentication failure** ⌈In case Vehicle Package authentication fails at transferExit call, UCM Master shall raise the ApplicationError AuthenticationFailed.⌋*(RS_UCM_00039, RS_UCM_00043)*

[SWS_UCM_01303]{DRAFT} **Dependencies between Software Packages** ⌈UCM Master shall check dependencies based on Vehicle Package Manifests and Software Packages Manifests before an transfer of Software Packages.⌋ *(RS_UCM_00035, RS_UCM_00043)*

7.2.9 Vehicle update security and confidentiality

The methods GetSwClusterInfo, SwPackageInventory and GetHistory could use private or confidential information.

[SWS_UCM_01304]{DRAFT} **Confidential information protection** ⌈The methods GetSwClusterInfo, SwPackageInventory and GetHistory shall only be called over secure communication channel providing confidentiality protection.⌋*(RS_UCM_00033)*

8 API specification

There are no APIs defined in this release.

9 Service Interfaces

9.1 Type definitions

This chapter lists all types provided by the UCM.

9.1.1 UCMIdentifierType

[SWS_UCM_00173]{DRAFT} **UCMIdentifierType table** ⌈

Name	UCMIdentifierType
Kind	STRING
Derived from	-
Description	UCM Module Instantiation Identifier.

Table 9.1: Implementation Data Type - UCMIdentifierType

⌋(*RS_UCM_00036*)

9.1.2 TransferIdType

[SWS_UCM_00031]{DRAFT} **TransferIdType table** ⌈

Name	TransferIdType
Kind	ARRAY
Array size	16
Subelements	`uint8_t`
Derived from	-
Description	Represents a handle identifier used to reference a particular transfer request.

Table 9.2: Implementation Data Type - TransferIdType

⌋(*RS_UCM_00019*, *RS_UCM_00025*)

9.1.3 SwNameType

[SWS_UCM_00071]{DRAFT} **SwNameType table** ⌈

Name	SwNameType
Kind	STRING
Derived from	-
Description	SoftwareCluster (SoftwarePackage) name.

Table 9.3: Implementation Data Type - SwNameType

⌋(*RS_UCM_00002*)

9.1.4 SwNameVectorType

[SWS_UCM_00174]{DRAFT} **SwNameVectorType table** ⌈

Name	SwNameVectorType
Kind	VECTOR
Subelements	SwNameType
Derived from	-
Description	Represents a dynamic size array of Software Cluster names.

Table 9.4: Implementation Data Type - SwNameVectorType

⌋(*RS_UCM_00002*)

9.1.5 StrongRevisionLabelString

[SWS_UCM_00175]{DRAFT} **StrongRevisionLabelString table** ⌈

Name	StrongRevisionLabelString
Kind	STRING
Derived from	-
Description	SoftwareCluster (SoftwarePackage) version.

Table 9.5: Implementation Data Type - StrongRevisionLabelString

⌋(*RS_UCM_00002*)

9.1.6 SwNameVersionType

[SWS_UCM_00176]{DRAFT} **SwNameVersionType table** ⌈

Name	SwNameVersionType
Kind	STRUCTURE
Subelements	Name `SwNameType` Version `StrongRevisionLabelString`
Derived from	-
Description	Represents the information of a Software Package (Software Cluster) name and version.

Table 9.6: Implementation Data Type - SwNameVersionType

⌋(*RS_UCM_00002*)

9.1.7 SwNameVersionVectorType

[SWS_UCM_00177]{DRAFT} **SwNameVersionVectorType table** ⌈

Name	SwNameVersionVectorType
Kind	VECTOR
Subelements	`SwNameVersionType`
Derived from	-
Description	Represents a dynamic size array of Software Name and Version

Table 9.7: Implementation Data Type - SwNameVersionVectorType

⌋(*RS_UCM_00002*)

9.1.8 ByteVectorType

[SWS_UCM_00032]{DRAFT} **ByteVectorType table** ⌈

Name	ByteVectorType
Kind	VECTOR
Subelements	`uint8_t`
Derived from	-
Description	Byte vector representing raw data.

Table 9.8: Implementation Data Type - ByteVectorType

⌋(*RS_UCM_00025*)

9.1.9 SwPackageStateType

[SWS_UCM_00038]{DRAFT} **SwPackageStateType table** ⌈

Name	SwPackageStateType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Represents the state of a Software Package on the Platform.	
Range / Symbol	**Limit**	**Description**
kTransferring	0x00	Software package is being transferred, i.e. not completely received.
kTransferred	0x01	Software package is completely transferred and ready to be processed.
kProcessing	0x02	Software package is currently being processed.
kProcessed	0x03	Software package processing finished.
kProcessingStream	0x04	Software package is being processed from a stream.

Table 9.9: Implementation Data Type - SwPackageStateType

⌋(*RS_UCM_00002*, *RS_UCM_00006*, *RS_UCM_00010*, *RS_UCM_00011*, *RS_UCM_00012*)

9.1.10 SwPackageInfoType

[SWS_UCM_00039]{DRAFT} **SwPackageInfoType table** ⌈

Name	SwPackageInfoType
Kind	STRUCTURE
Subelements	Name `SwNameType` Version `StrongRevisionLabelString` TransferID `TransferIdType` ConsecutiveBytesReceived `uint64_t` ConsecutiveBlocksReceived `uint64_t` State `SwPackageStateType`
Derived from	-
Description	Represents the information of a Software Package.

Table 9.10: Implementation Data Type - SwPackageInfoType

⌋(*RS_UCM_00002*, *RS_UCM_00006*, *RS_UCM_00010*, *RS_UCM_00011*, *RS_UCM_00012*)

9.1.11 SwPackageInfoVectorType

[SWS_UCM_00040]{DRAFT} **SwPackageInfoVectorType table** ⌈

Name	SwPackageInfoVectorType
Kind	VECTOR
Subelements	SwPackageInfoType
Derived from	-
Description	Represents a dynamic size array of Software Packages

Table 9.11: Implementation Data Type - SwPackageInfoVectorType

⌋(*RS_UCM_00002*, *RS_UCM_00006*, *RS_UCM_00010*, *RS_UCM_00011*, *RS_UCM_00012*)

9.1.12 SwClusterStateType

[SWS_UCM_00077]{DRAFT} **SwClusterStateType table** ⌈

Name	SwClusterStateType	
Kind	TYPE_REFERENCE	
Derived from	uint8_t	
Description	Represents the state of a SoftwareCluster on the adaptive platform.	
Range / Symbol	***Limit***	***Description***
kPresent	0x00	State of a SoftwareCluster that is installed on the adaptive platform and installation has finished.
kAdded	0x01	State of a SoftwareCluster that has been newly installed.
kUpdated	0x02	State of a SoftwareCluster that has been updated.
kRemoved	0x03	State of a SoftwareCluster that has been removed.

Table 9.12: Implementation Data Type - SwClusterStateType

⌋(*RS_UCM_00002*, *RS_UCM_00006*, *RS_UCM_00010*, *RS_UCM_00011*, *RS_UCM_00012*)

9.1.13 SwClusterInfoType

[SWS_UCM_00078]{DRAFT} **SwClusterInfoType table** ⌈

Name	SwClusterInfoType
Kind	STRUCTURE
Subelements	Name SwNameType Version StrongRevisionLabelString State SwClusterStateType
Derived from	-

▽

△

Description	Represents the information of a SoftwareCluster.

Table 9.13: Implementation Data Type - SwClusterInfoType

⌋(*RS_UCM_00002*, *RS_UCM_00006*, *RS_UCM_00010*, *RS_UCM_00011*, *RS_UCM_00012*)

9.1.14 SwClusterInfoVectorType

[SWS_UCM_00079]{DRAFT} **SwClusterInfoVectorType table** ⌈

Name	SwClusterInfoVectorType
Kind	VECTOR
Subelements	`SwClusterInfoType`
Derived from	-
Description	Represents a dynamic size array of SoftwareClusters

Table 9.14: Implementation Data Type - SwClusterInfoVectorType

⌋(*RS_UCM_00002*, *RS_UCM_00006*, *RS_UCM_00010*, *RS_UCM_00011*, *RS_UCM_00012*)

9.1.15 LogLevelType

[SWS_UCM_00041]{DRAFT} **LogLevelType table** ⌈

Name	LogLevelType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Represents the severity of the log messages.	
Range / Symbol	***Limit***	***Description***
kOff	0x00	Logging is deactivated.
kFatal	0x01	Only fatal messages are logged.
kError	0x02	Only messages up to error level are logged.
kWarning	0x03	Only messages up to warning level are logged.
kInfo	0x04	Only messages up to info level are logged.
kDebug	0x05	Only messages up to debug level are logged.
kVerbose	0x06	Only messages up to verbose level are logged.

Table 9.15: Implementation Data Type - LogLevelType

⌋(*RS_UCM_00022*)

9.1.16 LogEntryType

[SWS_UCM_00042]{DRAFT} **LogEntryType table** ⌈

Name	LogEntryType
Kind	STRUCTURE
Subelements	LogLevel `LogLevelType` Message `LogMessageType`
Derived from	-
Description	Represents a single log message with a log level.

Table 9.16: Implementation Data Type - LogEntryType

⌋(*RS_UCM_00022*)

9.1.17 LogVectorType

[SWS_UCM_00043]{DRAFT} **LogVectorType table** ⌈

Name	LogVectorType
Kind	VECTOR
Subelements	`LogEntryType`
Derived from	-
Description	Represents a list of log messages.

Table 9.17: Implementation Data Type - LogVectorType

⌋(*RS_UCM_00022*)

9.1.18 PackageManagerStatusType

[SWS_UCM_00044]{DRAFT} **PackageManagerStatusType table** ⌈

Name	PackageManagerStatusType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Represents the state of UCM.	
Range / Symbol	***Limit***	***Description***
kIdle	0x00	UCM is ready to start processing if software packages are present.
kReady	0x01	UCM has processed one or several packages and waits for additional packages, activation or reversion of processed packages.
kProcessing	0x02	UCM is currently in the middle of processing a Software Package, i.e. a client has called ProcessSwPackage.

▽

△

kActivating	0x03	UCM is performing the dependency check and preparing the activation of the processed Software packages.
kActivated	0x04	Software changes introduced with processed Software Packages has been activated and executed.
kRollingBack	0x05	UCM is reverting changes introduced with processed packages.
kRolledBack	0x06	Software changes introduced with processed Software Packages has been deactivated and original software is executed.
kCleaningUp	0x07	Making sure that the system is in a clean state.
kVerifying	0x08	UCM (via State Management) is checking that the processed packages have been properly restarted.

Table 9.18: Implementation Data Type - PackageManagerStatusType

⌋(*RS_UCM_00024*, *RS_UCM_00026*)

9.1.19 ActionType

[SWS_UCM_00132]{DRAFT} **ActionType table** ⌈

Name	ActionType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Represents the UCM action.	
Range / Symbol	***Limit***	***Description***
kUpdate	0x00	Update of a SoftwareCluster.
kInstall	0x01	Installation of a new SoftwareCluster.
kRemove	0x02	Removal of a SoftwareCluster.

Table 9.19: Implementation Data Type - ActionType

⌋(*RS_UCM_00032*)

9.1.20 ResultType

[SWS_UCM_00133]{DRAFT} **ResultType table** ⌈

Name	ResultType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Represents the result of UCM action.	
Range / Symbol	***Limit***	***Description***
kSuccessfull	0x00	UCM's action was successful.

▽

△

kFailed	0x01	UCM's action failed.

Table 9.20: Implementation Data Type - ResultType

⌋(*RS_UCM_00032*)

9.1.21 GetHistoryType

[SWS_UCM_00134]{DRAFT} **GetHistoryType table** ⌈

Name	GetHistoryType
Kind	STRUCTURE
Subelements	Time `uint64_t` Name `SwNameType` Version `StrongRevisionLabelString` Action `ActionType` Resolution `ResultType`
Derived from	-
Description	Time refers to the activation time of the software cluster. It is represented in milliseconds of UCM's action resolution since 01.01.1970 (UTC).

Table 9.21: Implementation Data Type - GetHistoryType

⌋(*RS_UCM_00032*)

9.1.22 GetHistoryVectorType

[SWS_UCM_00135]{DRAFT} **GetHistoryType table** ⌈

Name	GetHistoryVectorType
Kind	VECTOR
Subelements	`GetHistoryType`
Derived from	-
Description	Represents a list of UCM actions

Table 9.22: Implementation Data Type - GetHistoryVectorType

⌋(*RS_UCM_00032*)

9.1.23 CampaignStateType

[SWS_UCM_01177]{DRAFT} **CampaignStateType table** ⌈

Name	CampaignStateType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Represents the status of Campaign.	
Range / Symbol	**Limit**	**Description**
kIdle	0x00	UCM Master is ready to start a software update campaign.
kSyncing	0x01	UCM master is providing the list of installed SWCLs (GetSwCluster Info) or computing the list of SWCLs to install (SwPackageInventory).
kVehiclePackageTransferring	0x02	A vehicle package is being transferred to UCM Master.
kTransferApproving	0x03	A driver approval and/or a safety policy application for transferring is pending.
kTransferring	0x04	UCM Master is transferring software packages to the UCM subordinates.
kProcessApproving	0x05	A driver approval and/or a safety policy application for processing is pending.
kProcessing	0x06	The processing of software packages on UCM subordinates is ongoing. The transferring of software packages may still occur.
kActivateApproving	0x07	A driver approval and/or a safety policy application for activation is pending.
kActivating	0x08	The activation of SWCLs on UCM subordinates is ongoing.
kVehicleChecking	0x09	UCM Master is performing post-activation checks (OEM specific).
kRollingBack	0x10	UCM Master is rolling-back the activated SWCLs on the UCM subordinates.

Table 9.23: Implementation Data Type - CampaignStateType

⌋(*RS_UCM_00032*)

9.1.24 SafetyPolicyType

[SWS_UCM_01114]{DRAFT} **SafetyPolicyType table** ⌈

Name	SafetyPolicyType
Kind	STRING
Derived from	-
Description	The type of the Safety Policy.

Table 9.24: Implementation Data Type - SafetyPolicyType

⌋(*RS_UCM_00002*)

9.2 Service Interfaces

9.2.1 Provided Service Interfaces

This chapter lists all provided service interfaces of the UCM.

9.2.1.1 Package Management

Port

[SWS_UCM_00073]{DRAFT} **ProvidedPort PackageManagement** ⌈

Name	PackageManagement		
Kind	ProvidedPort	***Interface***	`PackageManagement`
Description			
Variation			

Table 9.25: Port - PackageManagement

⌋(*RS_UCM_00001*)

Service Interface

[SWS_UCM_00131]{DRAFT} **ProvidedInterface PackageManagement** ⌈

Name	PackageManagement
NameSpace	ara::ucm::pkgmgr

Table 9.26: Service Interfaces - PackageManagement

Fields

Name	CurrentStatus
Description	The current status of UCM.
Type	`PackageManagerStatusType`
HasGetter	true
HasNotifier	true
HasSetter	false

Table 9.27: Service Interface PackageManagement - Field: CurrentStatus

Methods

Name	GetSwClusterInfo	
Description	This method returns a list of SoftwareClusters that are in state kPresent.	
FireAndForget	false	
Parameter	SwInfo	
	Description	List of installed SoftwareClusters that are in state kPresent.
	Type	`SwClusterInfoVectorType`
	Variation	
	Direction	OUT

▽

△

Table 9.28: Service Interface PackageManagement - Method: GetSwClusterInfo

Name	GetSwClusterChangeInfo	
Description	This method returns a list pending changes to the set of SoftwareClusters on the adaptive platform. The returned list includes all SoftwareClusters that are to be added, updated or removed. The list of changes is extended in the course of processing Software Packages.	
FireAndForget	false	
Parameter	SwInfo	
	Description	List of SoftwareClusters that are in state kAdded,kUpdated or kRemoved.
	Type	`SwClusterInfoVectorType`
	Variation	
	Direction	OUT

Table 9.29: Service Interface PackageManagement - Method: GetSwClusterChangeInfo

Name	GetSwPackages	
Description	This method returns the Software Packages that available in UCM.	
FireAndForget	false	
Parameter	Packages	
	Description	List of Software Packages.
	Type	`SwPackageInfoVectorType`
	Variation	
	Direction	OUT

Table 9.30: Service Interface PackageManagement - Method: GetSwPackages

Name	TransferStart	
Description	Start the transfer of a Software Package. The size of the Software Package to be transferred to UCM must be provided. UCM will generate a Transfer ID for subsequent calls to TransferData, TransferExit, ProcessSwPackage, DeleteTransfer.	
FireAndForget	false	
Parameter	size	
	Description	Size (in bytes) of the Software Package to be transferred.
	Type	`uint64_t`
	Variation	
	Direction	IN
Parameter	id	
	Description	Return TransferId.
	Type	`TransferIdType`

▽

△

	Variation	
	Direction	OUT
Application Errors	`Insuffi-cientMemory`	Insufficient memory to perform operation.

Table 9.31: Service Interface PackageManagement - Method: TransferStart

Name	TransferData	
Description	Block-wise transfer of a Software Package to UCM.	
FireAndForget	false	
Parameter	id	
	Description	Transfer ID.
	Type	`TransferIdType`
	Variation	
	Direction	IN
Parameter	data	
	Description	Data block of the Software Package.
	Type	`ByteVectorType`
	Variation	
	Direction	IN
Parameter	blockCounter	
	Description	Block counter value of the current block.
	Type	`uint64_t`
	Variation	
	Direction	IN
Application Errors	`Incorrect-Block`	The the same block number is received twice.
Application Errors	`Incorrect-Size`	The size of the Software Package exceeds the provided size in TransferStart.
Application Errors	`Insuffi-cientMemory`	Insufficient memory to perform operation.
Application Errors	`Invalid-TransferId`	The Transfer ID is invalid.
Application Errors	`PackageIn-consistent`	Package integrity check failed.
Application Errors	`Opera-tionNotPer-mitted`	The operation is not supported in the current context.
Application Errors	`Authentica-tionFailed`	Software Package authentication failed.
Application Errors	`Incompati-blePackageV-ersion`	The version of the Software Package to be processed is not compatible with the current version of UCM.

▽

△

Application Errors	`BlockInconsistent`	Consistency check for transferred block failed.
Application Errors	`TransferInterrupted`	Transfer has been interrupted.

Table 9.32: Service Interface PackageManagement - Method: TransferData

Name	TransferExit	
Description	Finish the transfer of a Software Package to UCM.	
FireAndForget	false	
Parameter	id	
	Description	Transfer ID of the currently running request.
	Type	`TransferIdType`
	Variation	
	Direction	IN
Application Errors	`InsufficientData`	TransferExit has been called but total transferred data size does not match expected data size provided with TransferStart call.
Application Errors	`PackageInconsistent`	Package integrity check failed.
Application Errors	`AuthenticationFailed`	Software Package authentication failed.
Application Errors	`OldVersion`	Software Package version is too old.
Application Errors	`InvalidTransferId`	The Transfer ID is invalid.
Application Errors	`OperationNotPermitted`	The operation is not supported in the current context.
Application Errors	`IncompatiblePackageVersion`	The version of the Software Package to be processed is not compatible with the current version of UCM.

Table 9.33: Service Interface PackageManagement - Method: TransferExit

Name	DeleteTransfer	
Description	Delete a transferred Software Package.	
FireAndForget	false	
Parameter	id	
	Description	Transfer ID of the currently running request.
	Type	`TransferIdType`
	Variation	
	Direction	IN
Application Errors	`GeneralReject`	General reject.
Application Errors	`GeneralMemoryError`	A general memory error occured.

▽

△

Application Errors	`InvalidTransferId`	The Transfer ID is invalid.
Application Errors	`OperationNotPermitted`	The operation is not supported in the current context.

Table 9.34: Service Interface PackageManagement - Method: DeleteTransfer

Name	ProcessSwPackage	
Description	Process a previously transferred Software Package.	
FireAndForget	false	
Parameter	id	
	Description	The Transfer ID of this Software Package.
	Type	`TransferIdType`
	Variation	
	Direction	IN
Application Errors	`ServiceBusy`	Another processing is already ongoing and therefore the current processing request has to be rejected.
Application Errors	`InvalidManifest`	Package manifest could not be read.
Application Errors	`ProcessedSoftwarePackageInconsistent`	The processed Software Package integrity check has failed.
Application Errors	`InsufficientMemory`	Insufficient memory to perform operation.
Application Errors	`InvalidTransferId`	The Transfer ID is invalid.
Application Errors	`OperationNotPermitted`	The operation is not supported in the current context.
Application Errors	`ProcessSwPackageCancelled`	The processing operation has been interrupted by a Cancel() call.
Application Errors	`AuthenticationFailed`	Software Package authentication failed.
Application Errors	`IncompatiblePackageVersion`	The version of the Software Package to be processed is not compatible with the current version of UCM.

Table 9.35: Service Interface PackageManagement - Method: ProcessSwPackage

Name	RevertProcessedSwPackages	
Description	Revert the changes done by processing (ProcessSwPackage) of one or several software packages.	
FireAndForget	false	
Application Errors	`NothingToRevert`	RevertProcessedSwPackages has been called without prior processing of a Software Package.
Application Errors	`NotAbleToRevertPackages`	RevertProcessedSwPackages failed.

▽

△

Application Errors	`OperationNotPermitted`	The operation is not supported in the current context.

Table 9.36: Service Interface PackageManagement - Method: RevertProcessedSwPackages

Name	GetSwProcessProgress	
Description	Get the progress (0 - 100%) of the currently processed Software Package.	
FireAndForget	false	
Parameter	id	
	Description	The Transfer ID of the Software Package.
	Type	`TransferIdType`
	Variation	
	Direction	IN
Parameter	progress	
	Description	The progress of the current package processing (0% - 100%). 0x00 ... 0x64, 0xFF for "'No information available"'
	Type	`uint8_t`
	Variation	
	Direction	OUT
Application Errors	`InvalidTransferId`	The Transfer ID is invalid.

Table 9.37: Service Interface PackageManagement - Method: GetSwProcessProgress

Name	Cancel	
Description	This method aborts an ongoing processing of a Software Package.	
FireAndForget	false	
Parameter	id	
	Description	The Transfer ID.
	Type	`TransferIdType`
	Variation	
	Direction	IN
Application Errors	`CancelFailed`	Cancel failed.
Application Errors	`OperationNotPermitted`	The operation is not supported in the current context.
Application Errors	`InvalidTransferId`	The Transfer ID is invalid.

Table 9.38: Service Interface PackageManagement - Method: Cancel

Name	Rollback	
Description	Rollback the system to the state before the packages were processed.	
FireAndForget	false	
Application Errors	`NothingToRollback`	Rollback cannot be performed due to no rollback data available.
Application Errors	`NotAbleToRollback`	Rollback failed.
Application Errors	`OperationNotPermitted`	The operation is not supported in the current context.

Table 9.39: Service Interface PackageManagement - Method: Rollback

Name	Activate	
Description	This method activates the processed components.	
FireAndForget	false	
Parameter	preActivate	
	Description	The ordered collection of the SoftwarePackage shortnames that are supposed to be pre-activated.
	Type	`SwNameVectorType`
	Variation	
	Direction	IN
Parameter	verify	
	Description	The ordered collection of the SoftwarePackage shortnames that are supposed to be verified.
	Type	`SwNameVectorType`
	Variation	
	Direction	IN
Application Errors	`PreActivationFailed`	Error during preActivation step.
Application Errors	`VerificationFailed`	Error during verification step.
Application Errors	`ErrorNoValidProcessing`	Activate cannot be performed because previous processing is invalid.
Application Errors	`MissingDependencies`	Activate cannot be performed because of missing dependencies.
Application Errors	`OperationNotPermitted`	The operation is not supported in the current context.

Table 9.40: Service Interface PackageManagement - Method: Activate

Name	Finish
Description	This method finishes the processing for the current set of processed Software Packages. It does a cleanup of all data of the processing including the sources of the Software Packages.
FireAndForget	false

▽

△

Application Errors	`GeneralReject`	General reject.
Application Errors	`OperationNotPermitted`	The operation is not supported in the current context.

Table 9.41: Service Interface PackageManagement - Method: Finish

Name	SetLogLevel	
Description	This method sets the log level.	
FireAndForget	false	
Parameter	logLevel	
	Description	The new log level to be used.
	Type	`LogLevelType`
	Variation	
	Direction	IN

Table 9.42: Service Interface PackageManagement - Method: SetLogLevel

Name	GetLog	
Description	Getter method to poll for the log messages.	
FireAndForget	false	
Parameter	log	
	Description	The log messages.
	Type	`LogVectorType`
	Variation	
	Direction	OUT

Table 9.43: Service Interface PackageManagement - Method: GetLog

Name	GetHistory	
Description	Getter method to retrieve all actions that have been performed by UCM.	
FireAndForget	false	
Parameter	timestampGE	
	Description	Earliest timestamp (inclusive)
	Type	`uint64_t`
	Variation	
	Direction	IN
Parameter	timestampLT	
	Description	Latest timestamp (exclusive)

▽

△

	Type	`uint64_t`
	Variation	
	Direction	IN
Parameter	history	
	Description	The history of all actions that have been performed by UCM.
	Type	`GetHistoryVectorType`
	Variation	
	Direction	OUT

Table 9.44: Service Interface PackageManagement - Method: GetHistory

Name	GetId	
Description	Get the UCM Instance Identifier.	
FireAndForget	false	
Parameter	id	
	Description	UCM Module Instantiation Identifier.
	Type	`UCMIdentifierType`
	Variation	
	Direction	OUT

Table 9.45: Service Interface PackageManagement - Method: GetId

⌋(*RS_UCM_00001*, *RS_UCM_00002*, *RS_UCM_00008*, *RS_UCM_00010*, *RS_UCM_00011*, *RS_UCM_00015*, *RS_UCM_00018*, *RS_UCM_00021*, *RS_UCM_00022*, *RS_UCM_00023*, *RS_UCM_00024*, *RS_UCM_00025*, *RS_UCM_00032*)

9.2.1.2 Vehicle Package Management

Port

[SWS_UCM_00178]{DRAFT} **ProvidedPort VehiclePackageManagement** ⌈

Name	VehiclePackageManagement		
Kind	ProvidedPort	***Interface***	`VehiclePackageManagement`
Description			
Variation			

Table 9.46: Port - VehiclePackageManagement

⌋(*RS_UCM_00035*)

Service Interface

[SWS_UCM_00181]{DRAFT} **ProvidedInterface VehiclePackageManagement** ⌈

Name	VehiclePackageManagement
NameSpace	ara::ucm::pkgmgr

Table 9.47: Service Interfaces - VehiclePackageManagement

Fields

Name	CampaignState
Description	The current status of Campaign.
Type	`CampaignStateType`
HasGetter	true
HasNotifier	true
HasSetter	false

Table 9.48: Service Interface VehiclePackageManagement - Field: CampaignState

Methods

Name	SwPackageInventory	
Description		
FireAndForget	false	
Parameter	AvailableSoftwarePackages	
	Description	List of available Software Packages in Backend corresponding to VIN.
	Type	`SwNameVersionVectorType`
	Variation	
	Direction	IN
Parameter	RequiredSoftwarePackages	
	Description	List of Software Packages to be sent to UCM Master.
	Type	`SwNameVersionVectorType`
	Variation	
	Direction	OUT

Table 9.49: Service Interface VehiclePackageManagement - Method: SwPackageInventory

Name	GetSwClusterInfo	
Description	This method returns a list of SoftwareClusters that are in state kPresent.	
FireAndForget	false	
Parameter	SwInfo	
	Description	List of installed SoftwareClusters that are in state kPresent.
	Type	`SwClusterInfoVectorType`

▽

△

	Variation	
	Direction	OUT

Table 9.50: Service Interface VehiclePackageManagement - Method: GetSwClusterInfo

Name	TransferStart	
Description		
FireAndForget	false	
Parameter	SoftwarePackageName	
	Description	Software Package Short Name of the Software Package to be transferred.
	Type	`SwNameType`
	Variation	
	Direction	IN
Parameter	id	
	Description	Return TransferId.
	Type	`TransferIdType`
	Variation	
	Direction	OUT
Application Errors	`Insuffi-cientMemory`	Insufficient memory to perform operation.

Table 9.51: Service Interface VehiclePackageManagement - Method: TransferStart

Name	TransferVehiclePackage	
Description	Start the transfer of a Software Package. The size of the Software Package to be transferred to UCM must be provided. UCM will generate a Transfer ID for subsequent calls to TransferData, TransferExit, ProcessSwPackage, DeleteTransfer.	
FireAndForget	false	
Parameter	size	
	Description	Size (in bytes) of the Software Package to be transferred.
	Type	`uint64_t`
	Variation	
	Direction	IN
Parameter	id	
	Description	Return TransferId.
	Type	`TransferIdType`
	Variation	
	Direction	OUT

▽

△

Application Errors	Insuffi-cientMemory	Insufficient memory to perform operation.

Table 9.52: Service Interface VehiclePackageManagement - Method: TransferVehicle Package

Name	TransferData	
Description	Block-wise transfer of a Software Package to UCM.	
FireAndForget	false	
Parameter	id	
	Description	Transfer ID.
	Type	TransferIdType
	Variation	
	Direction	IN
Parameter	data	
	Description	Data block of the Software Package.
	Type	ByteVectorType
	Variation	
	Direction	IN
Parameter	blockCounter	
	Description	Block counter value of the current block.
	Type	uint64_t
	Variation	
	Direction	IN
Application Errors	Incorrect-Block	The the same block number is received twice.
Application Errors	Incorrect-Size	The size of the Software Package exceeds the provided size in TransferStart.
Application Errors	Insuffi-cientMemory	Insufficient memory to perform operation.
Application Errors	Invalid-TransferId	The Transfer ID is invalid.
Application Errors	PackageIn-consistent	Package integrity check failed.
Application Errors	Opera-tionNotPer-mitted	The operation is not supported in the current context.
Application Errors	Authentica-tionFailed	Software Package authentication failed.
Application Errors	Incompati-blePackageV-ersion	The version of the Software Package to be processed is not compatible with the current version of UCM.
Application Errors	BlockIncon-sistent	Consistency check for transferred block failed.

▽

△

Application Errors	`TransferInterrupted`	Transfer has been interrupted.

Table 9.53: Service Interface VehiclePackageManagement - Method: TransferData

Name	TransferExit	
Description	Finish the transfer of a Software Package to UCM.	
FireAndForget	false	
Parameter	id	
	Description	Transfer ID of the currently running request.
	Type	`TransferIdType`
	Variation	
	Direction	IN
Application Errors	`InsufficientData`	TransferExit has been called but total transferred data size does not match expected data size provided with TransferStart call.
Application Errors	`PackageInconsistent`	Package integrity check failed.
Application Errors	`AuthenticationFailed`	Software Package authentication failed.
Application Errors	`OldVersion`	Software Package version is too old.
Application Errors	`InvalidTransferId`	The Transfer ID is invalid.
Application Errors	`OperationNotPermitted`	The operation is not supported in the current context.
Application Errors	`IncompatiblePackageVersion`	The version of the Software Package to be processed is not compatible with the current version of UCM.

Table 9.54: Service Interface VehiclePackageManagement - Method: TransferExit

Name	DeleteTransfer	
Description	Delete a transferred Software Package.	
FireAndForget	false	
Parameter	id	
	Description	Transfer ID of the currently running request.
	Type	`TransferIdType`
	Variation	
	Direction	IN
Application Errors	`GeneralReject`	General reject.
Application Errors	`GeneralMemoryError`	A general memory error occured.
Application Errors	`InvalidTransferId`	The Transfer ID is invalid.

▽

△

Application Errors	`OperationNotPermitted`	The operation is not supported in the current context.

Table 9.55: Service Interface VehiclePackageManagement - Method: DeleteTransfer

Name	GetSwTransferProgress	
Description	Get the progress (0 - 100%) of the currently package transferring.	
FireAndForget	false	
Parameter	progress	
	Description	The progress of the current package transferring (0% - 100%). 0x00 ... 0x64, 0xFF for "'No information available"'
	Type	`uint8_t`
	Variation	
	Direction	OUT

Table 9.56: Service Interface VehiclePackageManagement - Method: GetSwTransfer Progress

Name	GetSwProcessProgress	
Description	Get the progress (0 - 100%) of the currently processing.	
FireAndForget	false	
Parameter	progress	
	Description	The progress of the current package processing (0% - 100%). 0x00 ... 0x64, 0xFF for "'No information available"'
	Type	`uint8_t`
	Variation	
	Direction	OUT

Table 9.57: Service Interface VehiclePackageManagement - Method: GetSwProcess Progress

Name	Cancel	
Description	This method aborts an ongoing processing of a Software Package.	
FireAndForget	false	
Parameter	id	
	Description	The Transfer ID.
	Type	`TransferIdType`
	Variation	
	Direction	IN
Application Errors	`CancelFailed`	Cancel failed.

▽

△

Application Errors	`OperationNotPermitted`	The operation is not supported in the current context.
Application Errors	`InvalidTransferId`	The Transfer ID is invalid.

Table 9.58: Service Interface VehiclePackageManagement - Method: Cancel

Name	GetHistory	
Description	Getter method to retrieve all actions that have been performed by UCM.	
FireAndForget	false	
Parameter	timestampGE	
	Description	Earliest timestamp (inclusive)
	Type	`uint64_t`
	Variation	
	Direction	IN
Parameter	timestampLT	
	Description	Latest timestamp (exclusive)
	Type	`uint64_t`
	Variation	
	Direction	IN
Parameter	history	
	Description	The history of all actions that have been performed by UCM.
	Type	`GetHistoryVectorType`
	Variation	
	Direction	OUT

Table 9.59: Service Interface VehiclePackageManagement - Method: GetHistory

⌋(*RS_UCM_00001*, *RS_UCM_00002*, *RS_UCM_00008*, *RS_UCM_00010*, *RS_UCM_00011*, *RS_UCM_00015*, *RS_UCM_00018*, *RS_UCM_00021*, *RS_UCM_00022*, *RS_UCM_00023*, *RS_UCM_00024*, *RS_UCM_00025*, *RS_UCM_00032*)

9.2.2 Required Service Interfaces

This chapter lists all required service interfaces of the `UCM Master`.

9.2.2.1 Vehicle Driver Application

Port

[SWS_UCM_00180]{DRAFT} **RequiredPort VehicleDriverApplication** ⌈

Name	VehicleDriverApplication		
Kind	RequiredPort	***Interface***	VehicleDriverApplication
Description			
Variation			

Table 9.60: Port - VehicleDriverApplication

⌋(*RS_UCM_00038*, *RS_UCM_00043*)

Service Interface

[SWS_UCM_00182]{DRAFT} **RequiredInterface VehicleDriverApplication** ⌈

Name	VehicleDriverApplication

Table 9.61: Service Interfaces - VehicleDriverApplication

Methods

Name	DriverNotification	
Description		
FireAndForget	false	
Parameter	CampaignState	
	Description	
	Type	CampaignStateType
	Variation	
	Direction	IN
Parameter	ApprovalRequiredFlag	
	Description	
	Type	bool
	Variation	
	Direction	IN
Parameter	SafetyPolicy	
	Description	
	Type	SafetyPolicyType
	Variation	
	Direction	IN
Parameter	ApprovalStatus	
	Description	
	Type	bool

▽

△

	Variation	
	Direction	OUT
Application Errors	`notSupport-edPolicy`	Policy not supported.

Table 9.62: Service Interface VehicleDriverApplication - Method: DriverNotification

⌋(*RS_UCM_00001*, *RS_UCM_00002*, *RS_UCM_00008*, *RS_UCM_00010*, *RS_-UCM_00011*, *RS_UCM_00015*, *RS_UCM_00018*, *RS_UCM_00021*, *RS_UCM_-00022*, *RS_UCM_00023*, *RS_UCM_00024*, *RS_UCM_00025*, *RS_UCM_00032*)

9.2.2.2 Vehicle State Manager

Port

[SWS_UCM_00179]{DRAFT} **RequiredPort VehicleStateManager** ⌈

Name	VehicleStateManager		
Kind	RequiredPort	***Interface***	`VehicleStateManager`
Description			
Variation			

Table 9.63: Port - VehicleStateManager

⌋(*RS_UCM_00037*, *RS_UCM_00043*)

Service Interface

[SWS_UCM_00183]{DRAFT} **RequiredInterface VehicleStateManager** ⌈

Name	VehicleStateManager

Table 9.64: Service Interfaces - VehicleStateManager

Fields

Name	SafeToUpdate
Description	Vehicle State Manager Application returns a field for UCM Master to know before/during/after update campaign what is vehicle state considering a safety policy described in the vehicle package.
Type	`bool`
HasGetter	true
HasNotifier	true

▽

△

HasSetter	false

Table 9.65: Service Interface VehicleStateManager - Field: SafeToUpdate

Methods

Name	ApplyPolicy	
Description		
FireAndForget	false	
Parameter	SafetyPolicy	
	Description	
	Type	SafetyPolicyType
	Variation	
	Direction	IN
Parameter	SafeToUpdate	
	Description	
	Type	bool
	Variation	
	Direction	OUT
Application Errors	notSupportedPolicy	Policy not supported.

Table 9.66: Service Interface VehicleStateManager - Method: ApplyPolicy

⌋(*RS_UCM_00001*, *RS_UCM_00002*, *RS_UCM_00008*, *RS_UCM_00010*, *RS_UCM_00011*, *RS_UCM_00015*, *RS_UCM_00018*, *RS_UCM_00021*, *RS_UCM_00022*, *RS_UCM_00023*, *RS_UCM_00024*, *RS_UCM_00025*, *RS_UCM_00032*)

9.3 Application Errors

9.3.1 Application Error Domain

9.3.1.1 UCMErrorDomain

This section lists all application errors of the UCM.

[SWS_UCM_00136]{DRAFT} **UCMErrorDomain** ⌈

Name	*Code*	*Description*
InsufficientMemory	1	Insufficient memory to perform operation.
IncorrectBlock	2	The the same block number is received twice.
IncorrectSize	3	The size of the Software Package exceeds the provided size in TransferStart.
InvalidTransferId	4	The Transfer ID is invalid.
OperationNotPermitted	5	The operation is not supported in the current context.
InsufficientData	6	TransferExit has been called but total transferred data size does not match expected data size provided with TransferStart call.
PackageInconsistent	7	Package integrity check failed.
AuthenticationFailed	8	Software Package authentication failed.
OldVersion	9	Software Package version is too old.
GeneralReject	10	General reject.
GeneralMemoryError	11	A general memory error occured.
ServiceBusy	12	Another processing is already ongoing and therefore the current processing request has to be rejected.
InvalidManifest	13	Package manifest could not be read.
NothingToRevert	14	RevertProcessedSwPackages has been called without prior processing of a Software Package.
NotAbleToRevertPackages	15	RevertProcessedSwPackages failed.
CancelFailed	16	Cancel failed.
NothingToRollback	17	Rollback cannot be performed due to no rollback data available.
NotAbleToRollback	18	Rollback failed.
PreActivationFailed	19	Error during preActivation step.
ErrorNoValidProcessing	20	Activate cannot be performed because previous processing is invalid.
MissingDependencies	21	Activate cannot be performed because of missing dependencies.
ProcessSwPackageCancelled	22	The processing operation has been interrupted by a Cancel() call.
ProcessedSoftwarePackageInconsistent	23	The processed Software Package integrity check has failed.
IncompatiblePackageVersion	24	The version of the Software Package to be processed is not compatible with the current version of UCM.
BlockInconsistent	25	Consistency check for transferred block failed.
TransferInterrupted	26	Transfer has been interrupted.
VerificationFailed	27	Error during verification step.

Table 9.67: Application Errors of UCMErrorDomain

⌋(*RS_UCM_00006*, *RS_UCM_00007*, *RS_UCM_00012*, *RS_UCM_00013*, *RS_UCM_00014*)

9.3.1.2 VehicleStateManagerErrorDomain

This section lists all application errors of the Vehicle State Manager.

[SWS_UCM_01115]{DRAFT} **VehicleStateManagerErrorDomain** ⌈

Name	Code	Description
notSupportedPolicy	1	Policy not supported.

Table 9.68: Application Errors of VehicleStateManagerErrorDomain

⌋(*RS_UCM_00037*, *RS_UCM_00043*)

9.3.1.3 VehicleDriverApplicationErrorDomain

This section lists all application errors of the Vehicle Driver Application.

[SWS_UCM_01116]{DRAFT} **VehicleDriverApplicationErrorDomain** ⌈

Name	Code	Description
notSupportedPolicy	1	Policy not supported.

Table 9.69: Application Errors of VehicleDriverApplicationErrorDomain

⌋(*RS_UCM_00038*, *RS_UCM_00043*)

10 Sequence diagrams

10.1 Update process

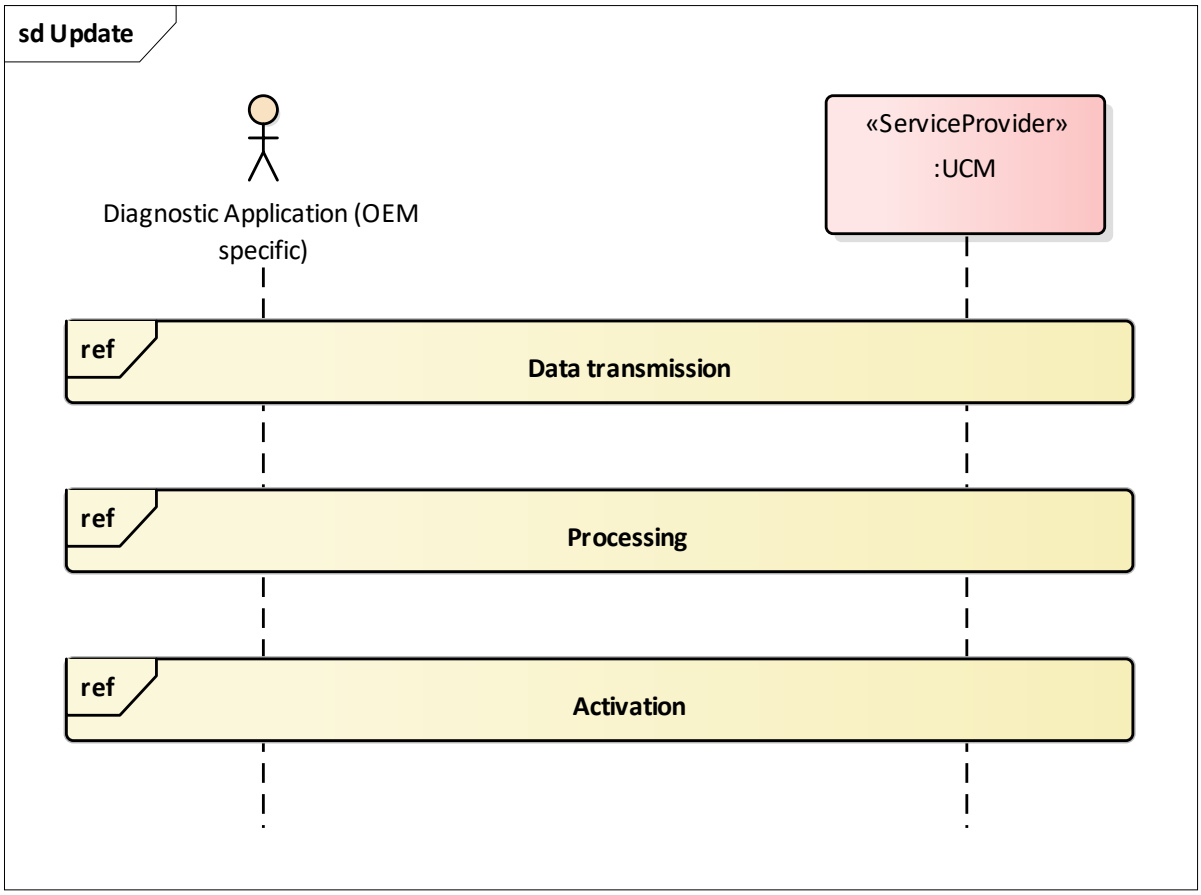

Figure 10.1: Sequence diagram showing the update process

10.2 Data transmission

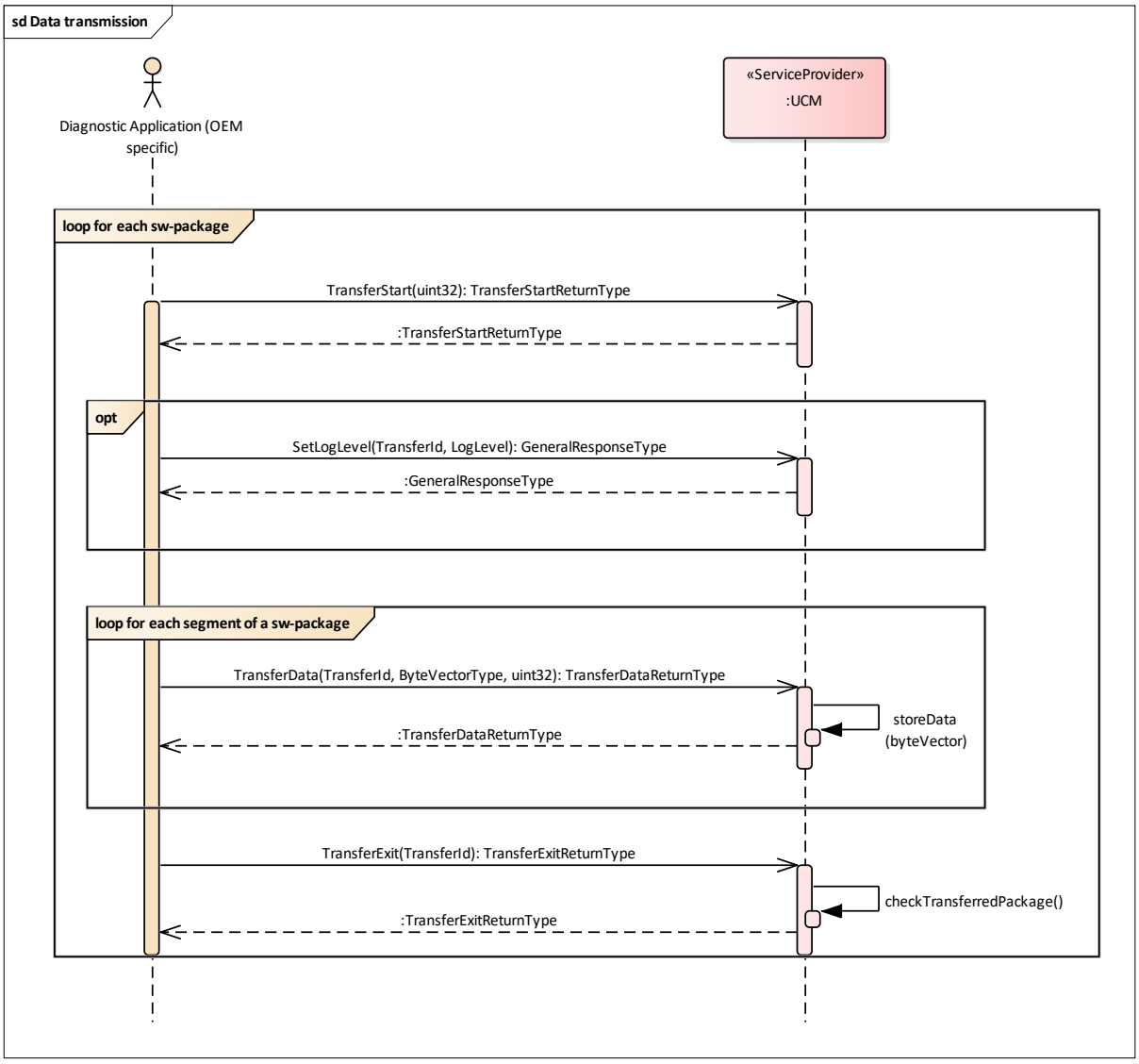

Figure 10.2: Sequence diagram showing the data transmission

10.3 Package processing

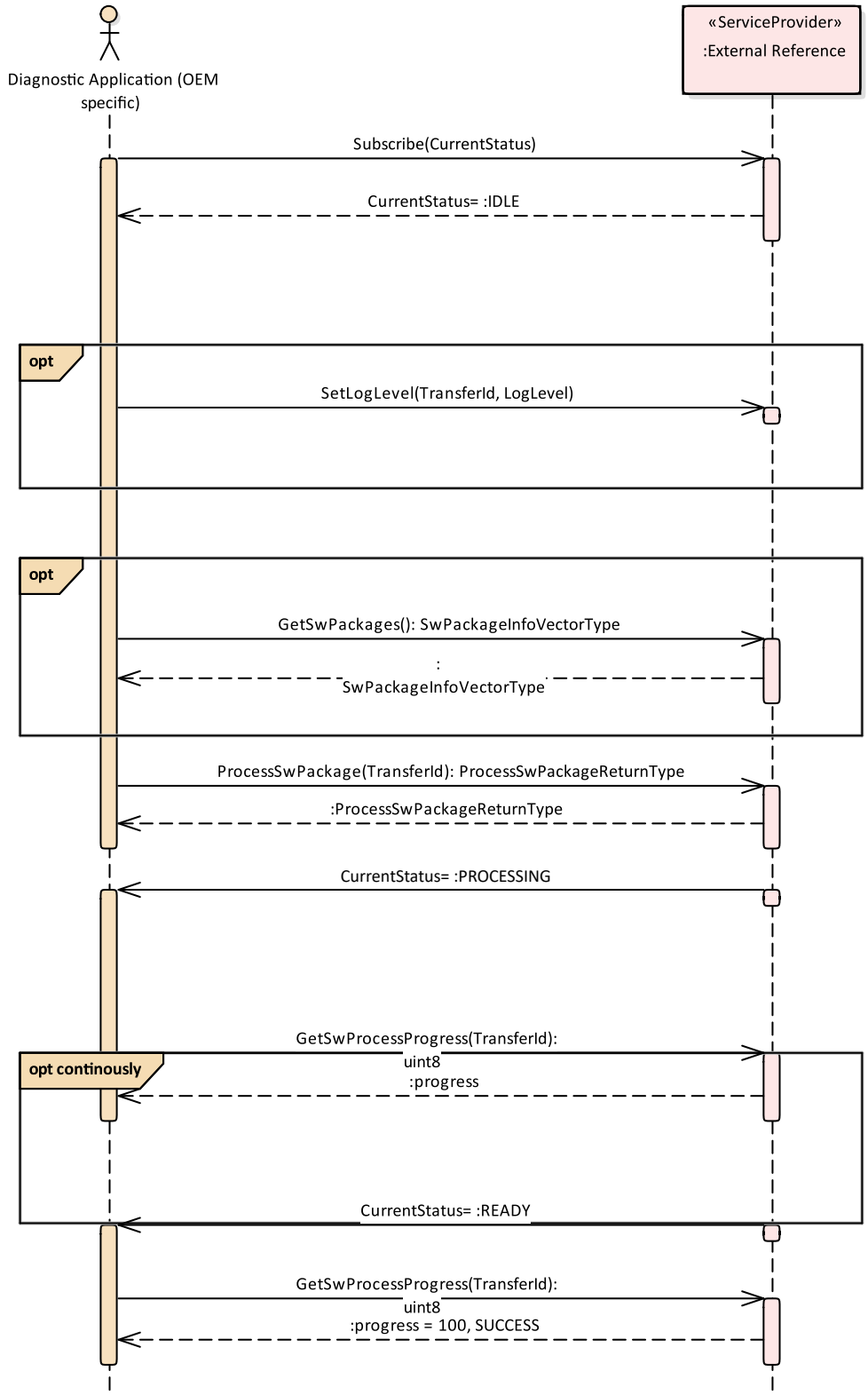

Figure 10.3: Sequence diagram showing the package processing

10.4 Activation

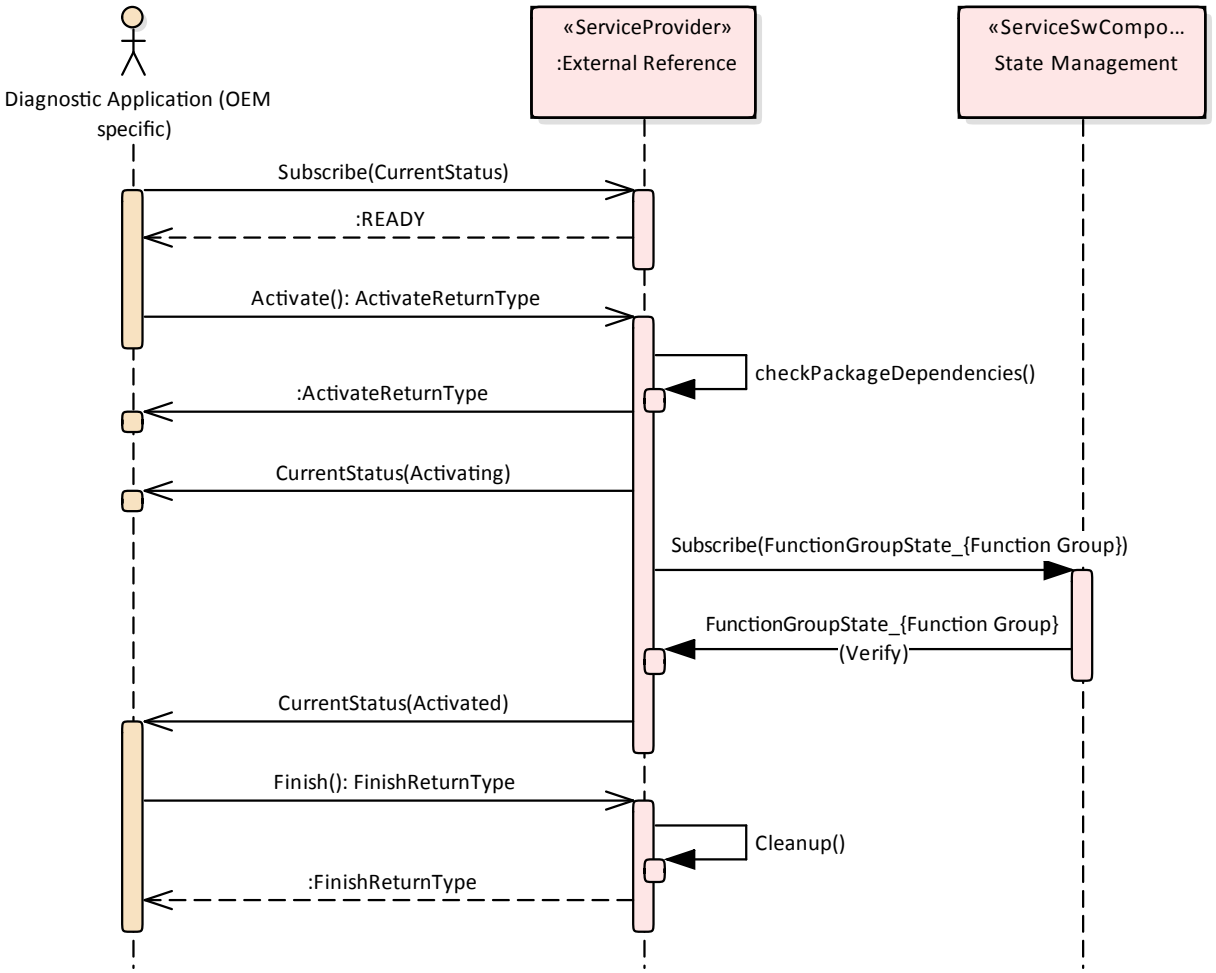

Figure 10.4: Sequence diagram showing the activation process

10.5 UCM Master simplified vehicle update

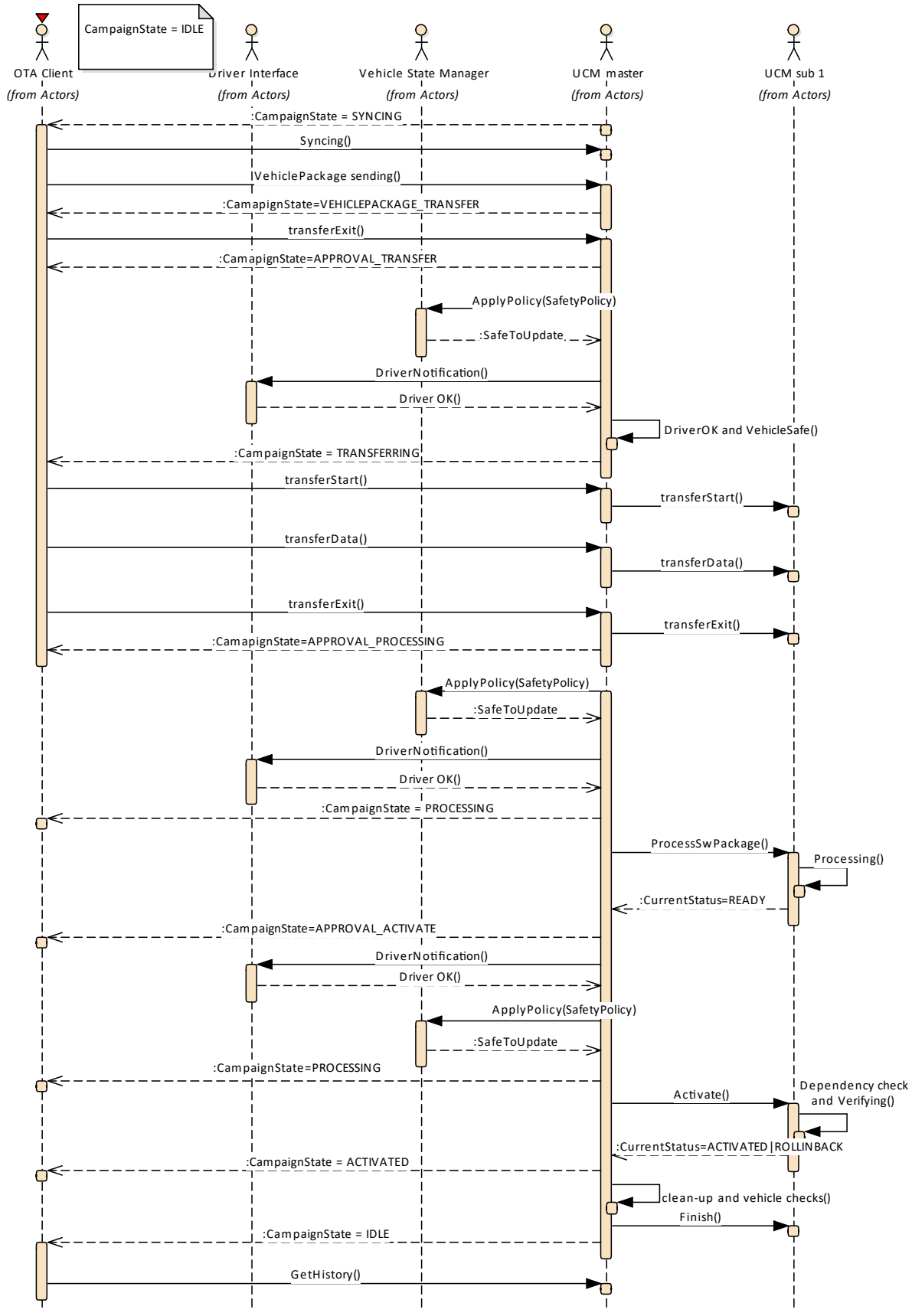

Figure 10.5: Sequence diagram showing vehicle update

A Not applicable requirements

none

B 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	***Identifiable*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*
Subclasses	ARPackage, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractServiceInstance*, *AbstractSignalBasedToISignalTriggeringMapping*, *AdaptiveModuleInstantiation*, AdaptiveSwcInternal Behavior, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartitionMapping, AsynchronousServerCallResultPoint, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, Autosar OperationArgumentInstance, AutosarVariableInstance, BswInternalTriggeringPoint, BswModule Dependency, *BuildActionEntity*, BuildActionEnvironment, CanTpAddress, CanTpChannel, CanTpNode, Chapter, CheckpointTransition, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *Communication Connector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, Coupling Port, *CouplingPortStructuralElement*, CryptoKeySlot, *CryptoServiceMapping*, DataPrototypeGroup, Data Transformation, DdsRpcServiceDeployment, DependencyOnArtifact, DeterministicClientResourceNeeds, *DiagEventDebounceAlgorithm*, DiagnosticConnectedIndicator, DiagnosticDataElement, Diagnostic FunctionInhibitSource, DiagnosticMasterToSlaveEventMapping, *DiagnosticRoutineSubfunction*, Dlt Argument, DltLogChannel, DltMessage, DoIpInterface, DoIpLogicAddress, E2EProfileConfiguration, EC UMapping, *EOCExecutableEntityRefAbstract*, EcuPartition, EcucContainerValue, *EcucDefinition Element*, EcucDestinationUriDef, EcucEnumerationLiteralDef, EcucQuery, EcucValidationCondition, End2EndEventProtectionProps, EndToEndProtection, EventMapping, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMap Element, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FieldMapping, FireAndForget Mapping, FlatInstanceDescriptor, FlexrayArTpNode, FlexrayTpConnectionControl, FlexrayTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HealthChannel*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPin Group, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, InterfaceMapping, InternalTriggeringPoint, J1939SharedAddressCluster, J1939TpNode, Keyword, Life CycleState, LinScheduleTable, LinTpNode, Linker, MacMulticastGroup, McDataInstance, Memory Section, MethodMapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, Network Endpoint, *NmCluster*, *NmNode*, NvBlockDescriptor, *PackageableElement*, ParameterAccess, PduTo FrameMapping, PduTriggering, PerInstanceMemory, PersistencyFileProxy, PersistencyKeyValuePair, *PhmActionItem*, PhmActionList, PhmLogicalExpression, PhmRule, *PhmSupervision*, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, PossibleErrorReaction, ProcessDesignToMachineDesignMapping, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKeySlotMapping, RawDataStream MethodDeployment, ResourceConsumption, ResourceGroup, *RestAbstractEndpoint*, RestElementDef, RestResourceDef, RootSwClusterDesignComponentPrototype, RootSwComponentPrototype, RootSw CompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RunnableEntityGroup, *SdgAttribute*, SdgClass, Sec OcJobMapping, SecOcJobRequirement, *SecureComProps*, SecureCommunicationAuthenticationProps, *SecureCommunicationDeployment*, SecureCommunicationFreshnessProps, *ServerCallPoint*, *Service EventDeployment*, *ServiceFieldDeployment*, ServiceInstanceToSignalMapping, *ServiceInterfaceElement Mapping*, ServiceInterfaceElementSecureComConfig, ServiceInterfaceMapping, *ServiceMethod Deployment*, *ServiceNeeds*, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SoftwarePackageStep, SomeipEventGroup, SomeipProvidedEventGroup, SomeipTp Channel, *SpecElementReference*, *StackUsage*, StartupConfig, StaticSocketConnection, StructuredReq, SupervisionCheckpoint, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwcTo ApplicationPartitionMapping, SwcToEcuMapping, SwcToImplMapping, SystemMapping, SystemMemory ▽

▽

△

Class	***Identifiable*** (abstract)			
	△ Usage, TcpOptionFilterList, *TimeBaseResource*, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, TlsCryptoCipherSuite, TlsJobMapping, Topic1, Tp Address, TraceableTable, TraceableText, *TracedFailure*, *TransformationProps*, TransformationPropsTo ServiceInterfaceElementMapping, TransformationTechnology, Trigger, UcmDescription, UcmStep, VariableAccess, VariationPointProxy, VehicleRolloutStep, ViewMap, VlanConfig, WaitPoint			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
adminData	AdminData	0..1	aggr	This represents the administrative data for the identifiable object. **Tags:**xml.sequenceOffset=-40
annotation	Annotation	*	aggr	Possibility to provide additional notes while defining a model element (e.g. the ECU Configuration Parameter Values). These are not intended as documentation but are mere design notes. **Tags:**xml.sequenceOffset=-25
category	CategoryString	0..1	attr	The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints. **Tags:**xml.sequenceOffset=-50
desc	MultiLanguageOverview Paragraph	0..1	aggr	This represents a general but brief (one paragraph) description what the object in question is about. It is only one paragraph! Desc is intended to be collected into overview tables. This property helps a human reader to identify the object in question. More elaborate documentation, (in particular how the object is built or used) should go to "introduction". **Tags:**xml.sequenceOffset=-60
introduction	DocumentationBlock	0..1	aggr	This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock. **Tags:**xml.sequenceOffset=-30
uuid	String	0..1	attr	The purpose of this attribute is to provide a globally unique identifier for an instance of a meta-class. The values of this attribute should be globally unique strings prefixed by the type of identifier. For example, to include a DCE UUID as defined by The Open Group, the UUID would be preceded by "DCE:". The values of this attribute may be used to support merging of different AUTOSAR models. The form of the UUID (Universally Unique Identifier) is taken from a standard defined by the Open Group (was Open Software Foundation). This standard is widely used, including by Microsoft for COM (GUIDs) and by many companies for DCE, which is based on CORBA. The method for generating these 128-bit IDs is published in the standard and the effectiveness and uniqueness of the IDs is not in practice disputed. If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003". The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp. **Tags:**xml.attribute=true

Table B.1: Identifiable

Class	***SoftwareActivationDependency*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::UploadableSoftwarePackage			
Note	This meta-class acts as an abstract base class for the formalization of dependencies in the context of software activation on the AUTOSAR adaptive platform. **Tags:**atp.Status=draft			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Subclasses	SoftwareCluster, SoftwareClusterDesign, VehiclePackage			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
conflictsTo	SoftwareActivation DependencyFormula	0..1	aggr	This aggregation handles conflicts. If it yields true then the SoftwareActivationDependency shall not be installed. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=conflictsTo atp.Status=draft
dependsOn	SoftwareActivation DependencyFormula	0..1	aggr	This aggregation can be taken to identify a dependency for the enclosing SoftwareActivationDependency. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=dependsOn atp.Status=draft

Table B.2: SoftwareActivationDependency

Class	**SoftwareCluster**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::UploadableSoftwarePackage			
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.ManifestKind=SoftwareDistribution atp.Status=draft atp.recommendedPackage=SoftwareClusters			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *SoftwareActivationDependency*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
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 atp.Status=draft
containedFibex Element	FibexElement	*	ref	This allows for referencing FibexElements that need to be considered in the context of a SoftwareCluster. **Tags:**atp.Status=draft
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 atp.Status=draft

▽

△

Class	**SoftwareCluster**			
contained Process	Process	*	ref	This reference represent the processes contained in the enclosing SoftwareCluster. **Tags:**atp.Status=draft
design	SoftwareClusterDesign	*	ref	This reference represents the identification of all Software ClusterDesigns applicable for the enclosing Software Cluster. **Stereotypes:** atpUriDef **Tags:**atp.Status=draft
diagnostic Address	SoftwareCluster DiagnosticAddress	*	aggr	This aggregation represents the collection of diagnostic addresses that apply for the SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=diagnosticAddress atp.Status=draft
diagnostic Extract	DiagnosticContribution Set	0..1	ref	This reference represents the definition of the diagnostic extract applicable to the referencing SoftwareCluster **Tags:**atp.Status=draft
license	Documentation	*	ref	This attribute allows for the inclusion of the 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. **Tags:**atp.Status=draft
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 atp.Status=draft
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. **Tags:**atp.Status=draft
subSoftware Cluster	SoftwareCluster	*	ref	This reference is used to identify the sub-Software Clusters of an "umbrella" SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=subSoftwareCluster atp.Status=draft
typeApproval	String	0..1	attr	This attribute carries the homologation information that may be specific for a given country.
vendorId	PositiveInteger	1	attr	Vendor ID of this Implementation according to the AUTOSAR vendor list.
vendor Signature	CryptoService Certificate	1	ref	This reference identifies the certificate that represents the vendor's signature. **Tags:**atp.Status=draft
version	StrongRevisionLabel String	1	attr	This attribute can be used to describe a version information for the enclosing SoftwareCluster.

Table B.3: SoftwareCluster

Class	**SoftwarePackage**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::UploadableSoftwarePackage			
Note	This meta-class represents the ability to formalize the content of a software package. **Tags:** atp.ManifestKind=SoftwareDistribution atp.Status=draft atp.recommendedPackage=SoftwarePackages			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
actionType	SoftwarePackageAction TypeEnum	1	attr	This attribute defines the action to be taken in the step of processing the enclosing SoftwarePackage.
activationAction	SoftwarePackage ActivationActionEnum	1	attr	This attribute governs the action to be taken after the installation of the SoftwareCluster completed.
compressed Software PackageSize	PositiveInteger	1	attr	This size represents the size of the compressed Software Package.
isDeltaPackage	Boolean	1	attr	This attribute denotes whether the SoftwarePackage is only able to update but not for initial installation.
maximum SupportedUcm Version	RevisionLabelString	1	attr	This attribute identifies the maximum supported version of the UCM for this SoftwarePackage.
minimum SupportedUcm Version	RevisionLabelString	1	attr	This attribute identifies the minimum supported version of the UCM for this SoftwarePackage.
packagerId	PositiveInteger	1	attr	This attribute identifies Id of the organization that provides the packager generating the SoftwarePackage.
packager Signature	CryptoService Certificate	1	ref	This reference identifies the certificate that represents the packager's signature. **Tags:**atp.Status=draft
postVerification Reboot	Boolean	1	attr	Reboot the platform after the verification of the activated software.
preActivate	ModeDeclaration	*	iref	The referenced function group states shall be established for the switch between the already installed and the activated software. **Tags:**atp.Status=draft
preActivation Reboot	Boolean	1	attr	Reboot the platform before the switch to the activated software.
softwareCluster	SoftwareCluster	1	ref	This reference identifies the SoftwareCluster that belongs to the SoftwarePackage. The nature of this relation is actually more like an aggregation than a reference. But the relation is still modelled as a reference because two ARElements cannot aggregate each other. **Tags:**atp.Status=draft
uncompressed SoftwareCluster Size	PositiveInteger	1	attr	This attribute gives an indication about the storage that has to be available on the target.
verify	ModeDeclaration	*	iref	The referenced function group states shall be established for the verification of the activated software. **Tags:**atp.Status=draft

Table B.4: SoftwarePackage

Class	**VehiclePackage**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::UploadableSoftwarePackage			
Note	This meta-class represents the ability to define a vehicle package for executing an update campaign. **Tags:** atp.ManifestKind=SoftwareDistribution atp.Status=draft atp.recommendedPackage=VehiclePackages			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *SoftwareActivationDependency*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
driver Notification	VehicleDriver Notification	*	aggr	This aggregation provides the ability to configure the necessary driver notifications. **Tags:**atp.Status=draft
packager Signature	CryptoService Certificate	1	ref	This reference identifies the certificate that represents the packager's signature. **Tags:**atp.Status=draft
repository	UriString	0..1	attr	This attribute identifies the repository where the Vehicle Package is stored.
rollout Qualification (ordered)	VehicleRolloutStep	*	aggr	This represents the rollout qualification. **Tags:**atp.Status=draft
ucm	UcmDescription	*	aggr	This aggregation represents the UcmDescriptions to be considered in the context of the VehiclePackage. **Tags:**atp.Status=draft
ucmMaster Fallback (ordered)	UcmDescription	*	ref	This reference lists the fallback order of Ucms that can take over the master role if the master goes down. **Tags:**atp.Status=draft
vehicle Description	Documentation	0..1	ref	This reference identifies the vehicle description. **Tags:**atp.Status=draft

Table B.5: VehiclePackage

Class	**UcmModuleInstantiation**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Ucm			
Note	This meta-class represents the ability to define a definition of a UCM instantiation. **Tags:**atp.Status=draft			
Base	*ARObject*, *AdaptiveModuleInstantiation*, *Identifiable*, *MultilanguageReferrable*, *NonOsModule Instantiation*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
identifier	String	1	attr	This represents the identification of a UCM.

Table B.6: UcmModuleInstantiation

C Interfaces to other Functional Clusters (informative)

C.1 Overview

AUTOSAR decided not to standardize interfaces which are exclusively used between Functional Clusters (on platform-level only), to allow efficient implementations, which might depend e.g. on the used Operating System.

This chapter provides informative guidelines how the interaction between Functional Clusters looks like, by clustering the relevant requirements of this document. In addition, the standardized public interfaces which are accessible by user space applications (see chapter 8) can also be used for interaction between Functional Clusters.

The goal is to provide a clear understanding of Functional Cluster boundaries and interaction, without specifying syntactical details. This ensures compatibility between documents specifying different Functional Clusters and supports parallel implementation of different Functional Clusters. Details of the interfaces are up to the platform provider.

C.2 Interfaces Tables

C.2.1 UCM update notification

UCM shall provide the notification to other Functional Clusters that changes have been done to the software. This enables other functional clusters to check if updated manifests have changes relevant for the concerned Functional Cluster. This can be done through the field CurrentStatus provided by the UCM service.

D Packages distribution within vehicle detailed sequence examples

D.1 Collect information of present Software Clusters in vehicle

From a regular basis, UCM master and UCM can collect information of present Software Clusters from the other AUTOSAR Adaptive Platforms of the vehicle in order to be used later when communicating with Backend and then determine if there are new actions (update, remove, install) required.

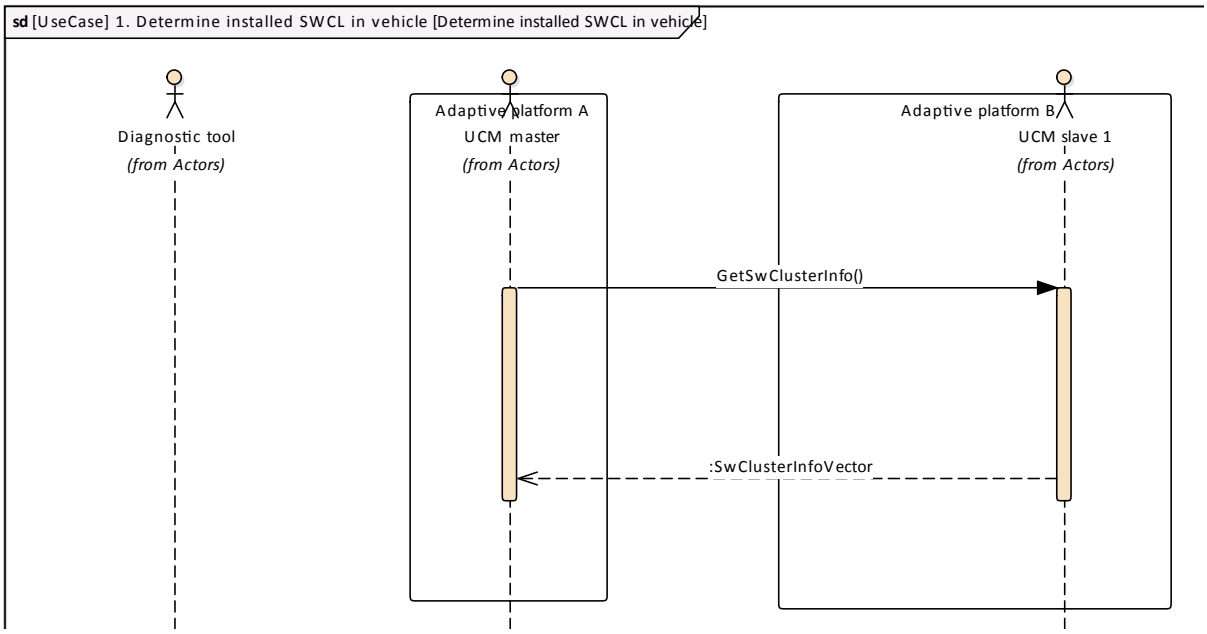

Figure D.1: Collect information of Software Clusters present in vehicle from several AUTOSAR Adaptive Platforms

D.2 Action computation

In order to find out if there is a new update available from Backend or the need to install or remove a Software Cluster, vehicle and Backend have to share their current status and either Backend or vehicle have to compute what UCM Master actions are needed.

Backend will have the possibility to push a package into the vehicle when communication is established, for instance for security purpose.

Communication trial between Backend and UCM master can be done on driver's request or from a scheduler.

D.2.1 Pull package from Backend into vehicle

Case where vehicle is computing the difference between Software Clusters versions that are present in vehicle and the ones available in Backend.

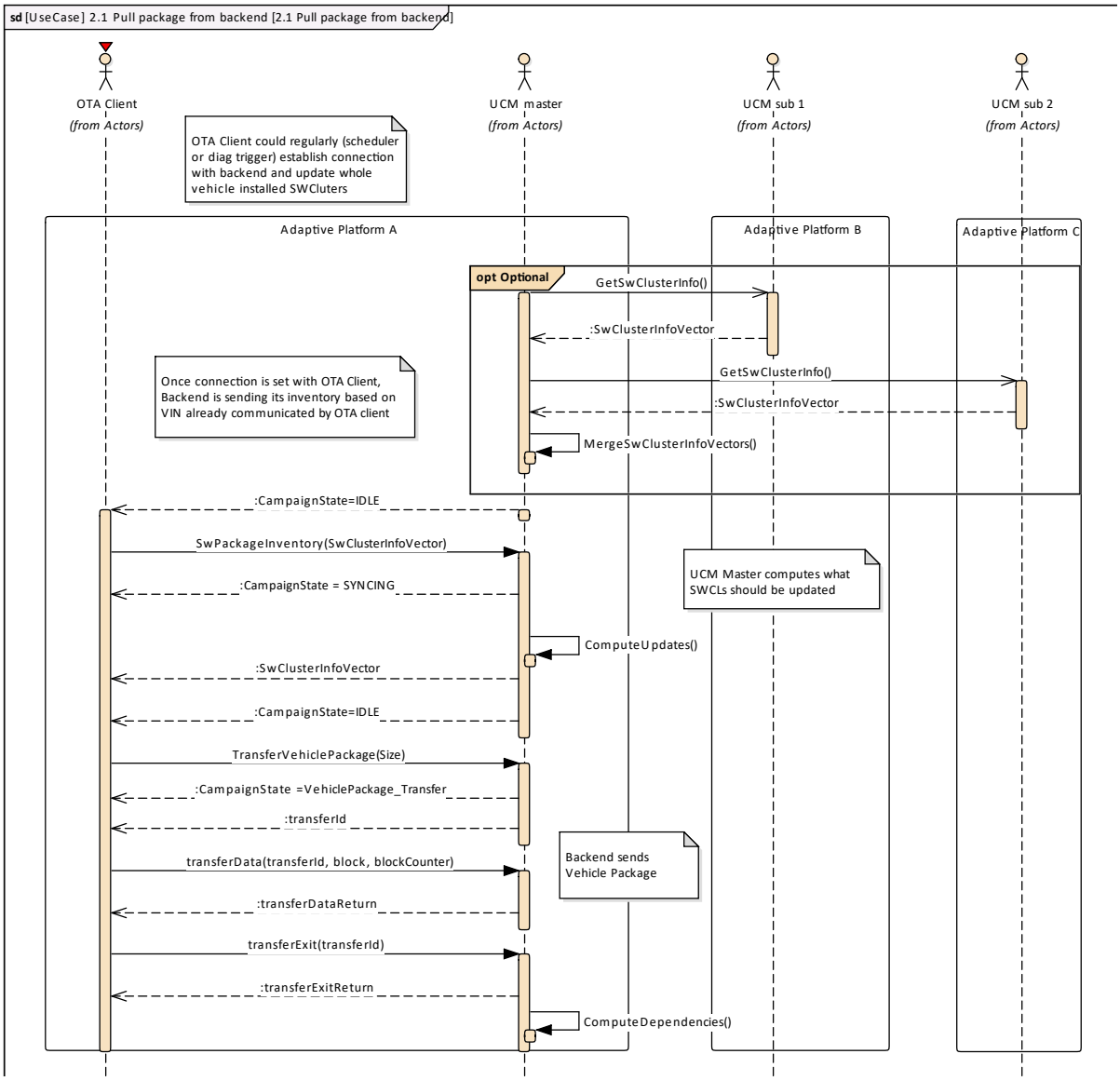

Figure D.2: Pull package from backend

D.2.2 Push package from backend into vehicle

Case where Backend is computing the difference between Software Clusters versions that are present in vehicle and the ones available in Backend.

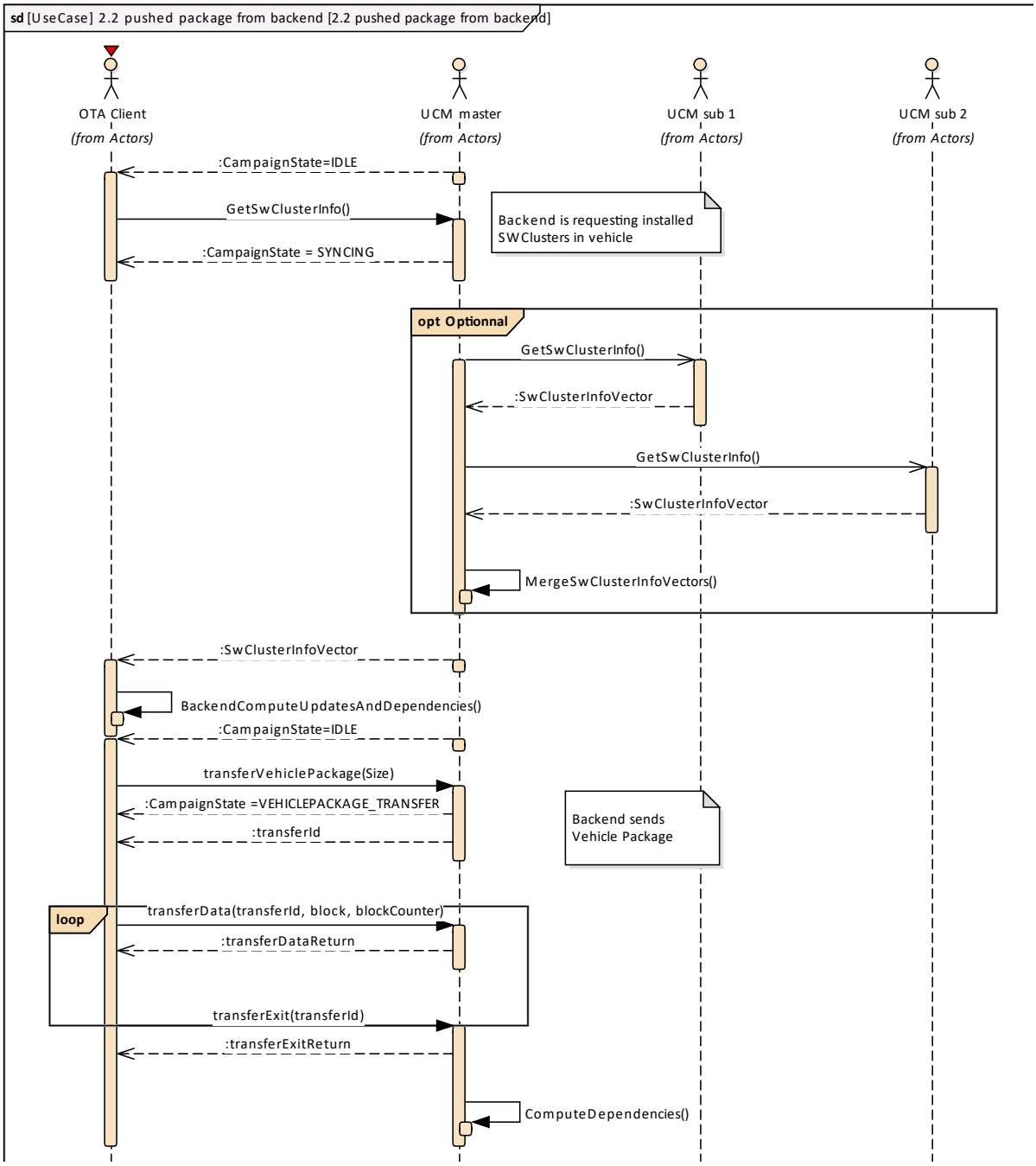

Figure D.3: Push package from backend

D.3 Packages transfer from backend into targeted UCM

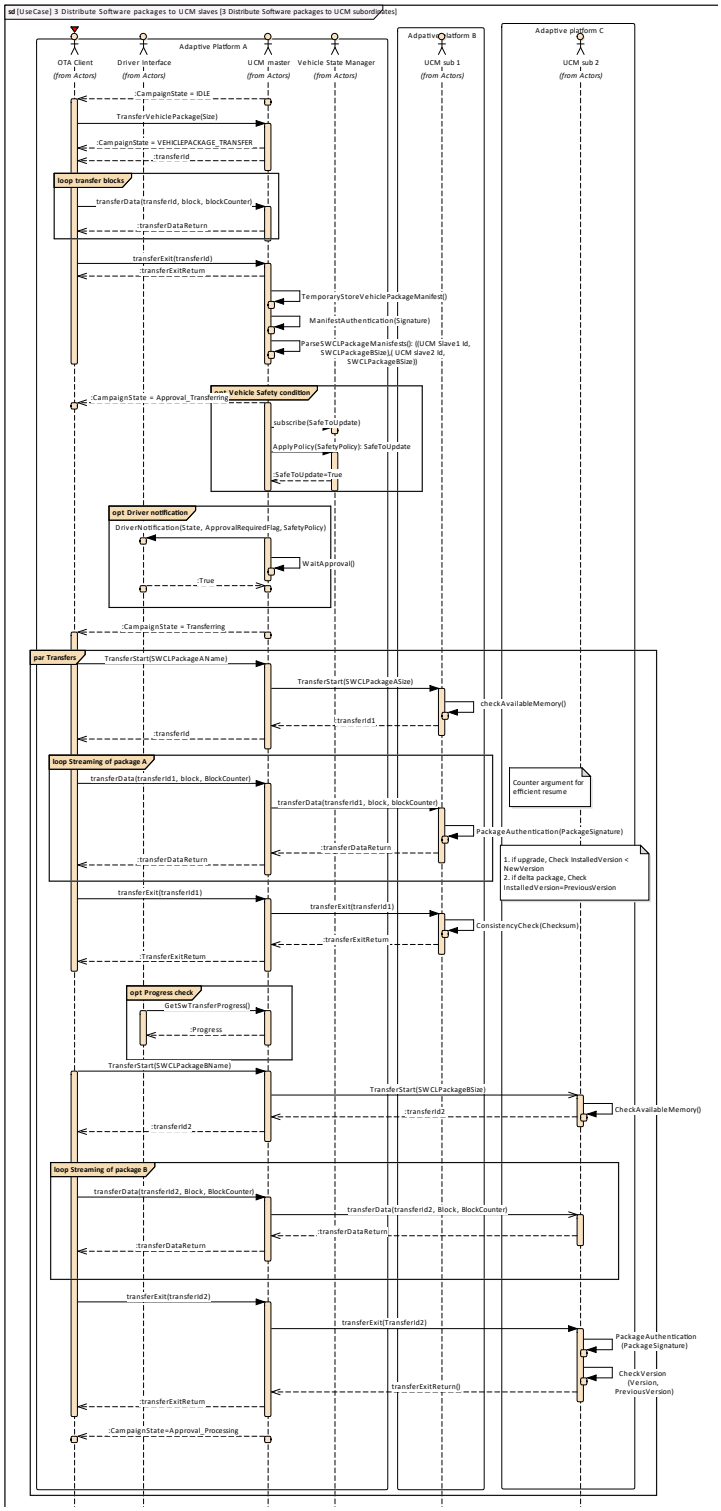

Figure D.4: Stream packages blocks from backend into targeted UCM

D.4 Package processing

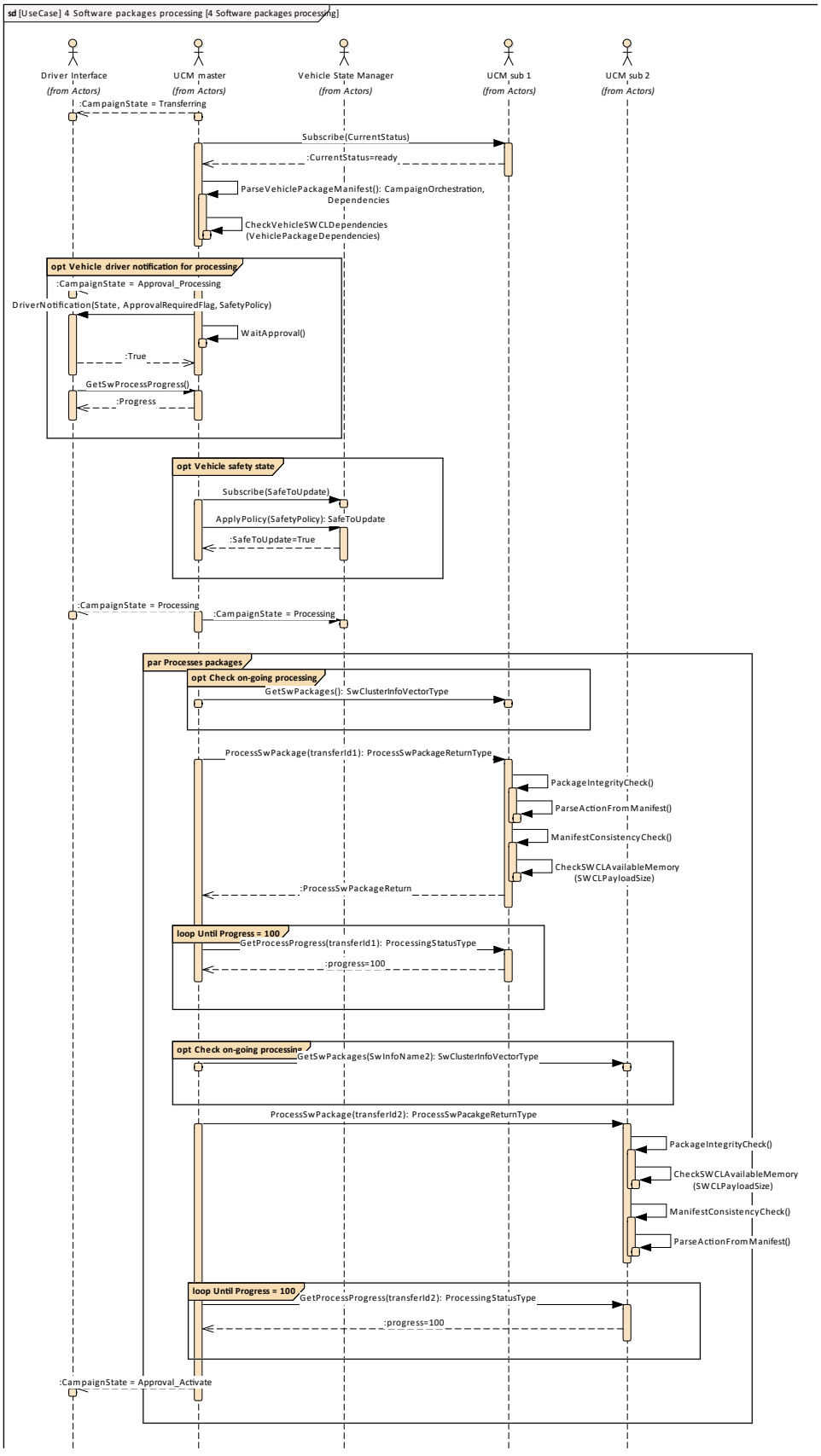

Figure D.5: Packages processing by UCMs

D.5 Package activation

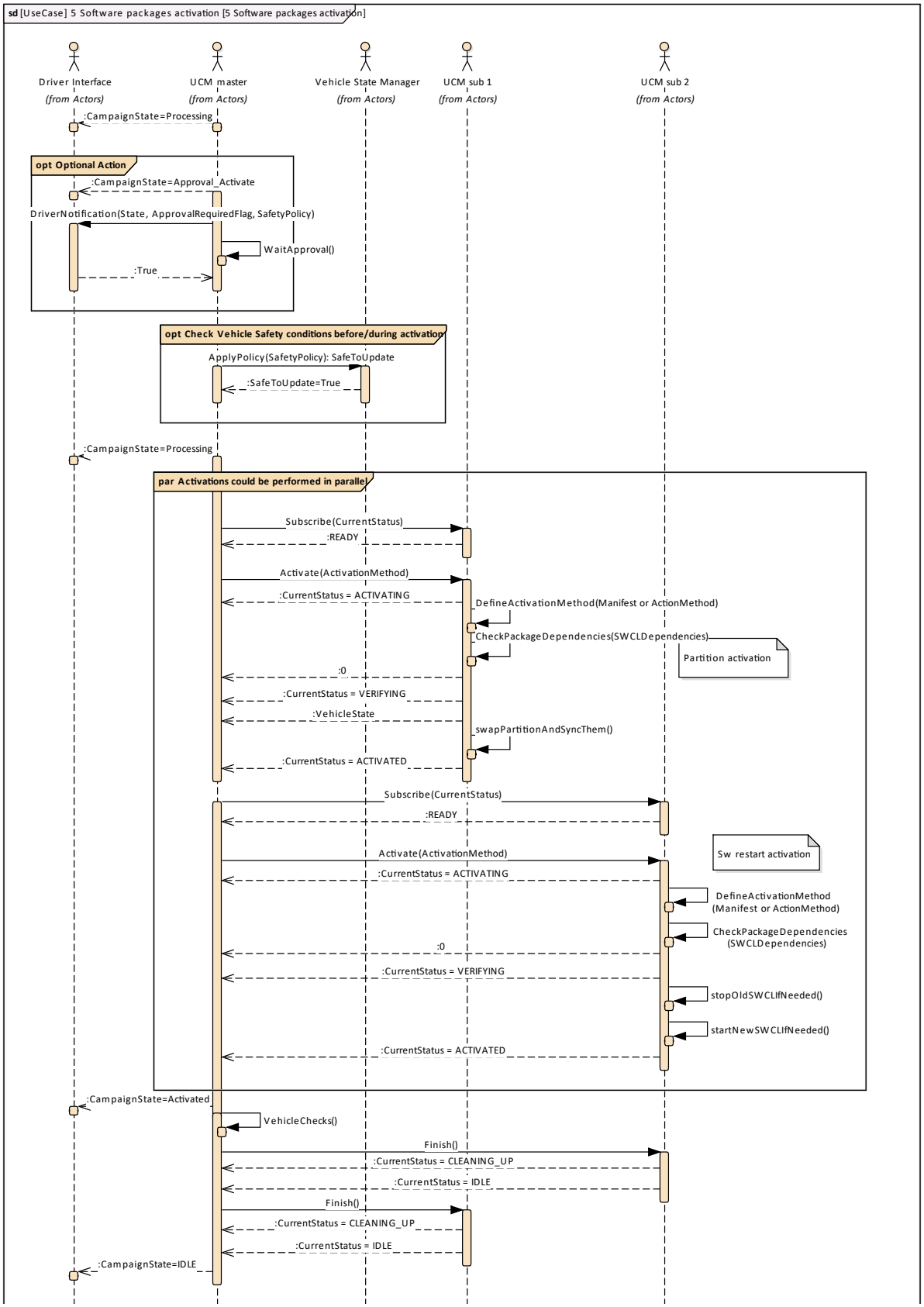

Figure D.6: Packages activation by UCMs

D.6 Package rollback

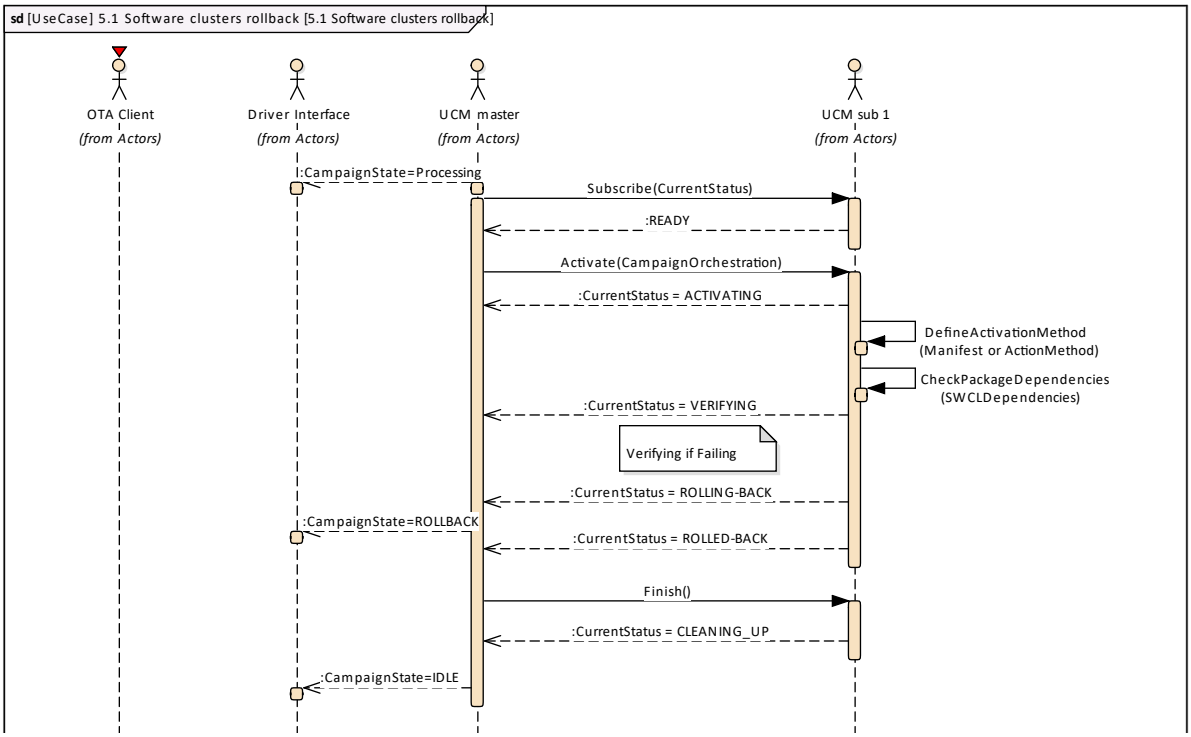

Figure D.7: Packages rollback by UCMs

D.7 Campaign reporting

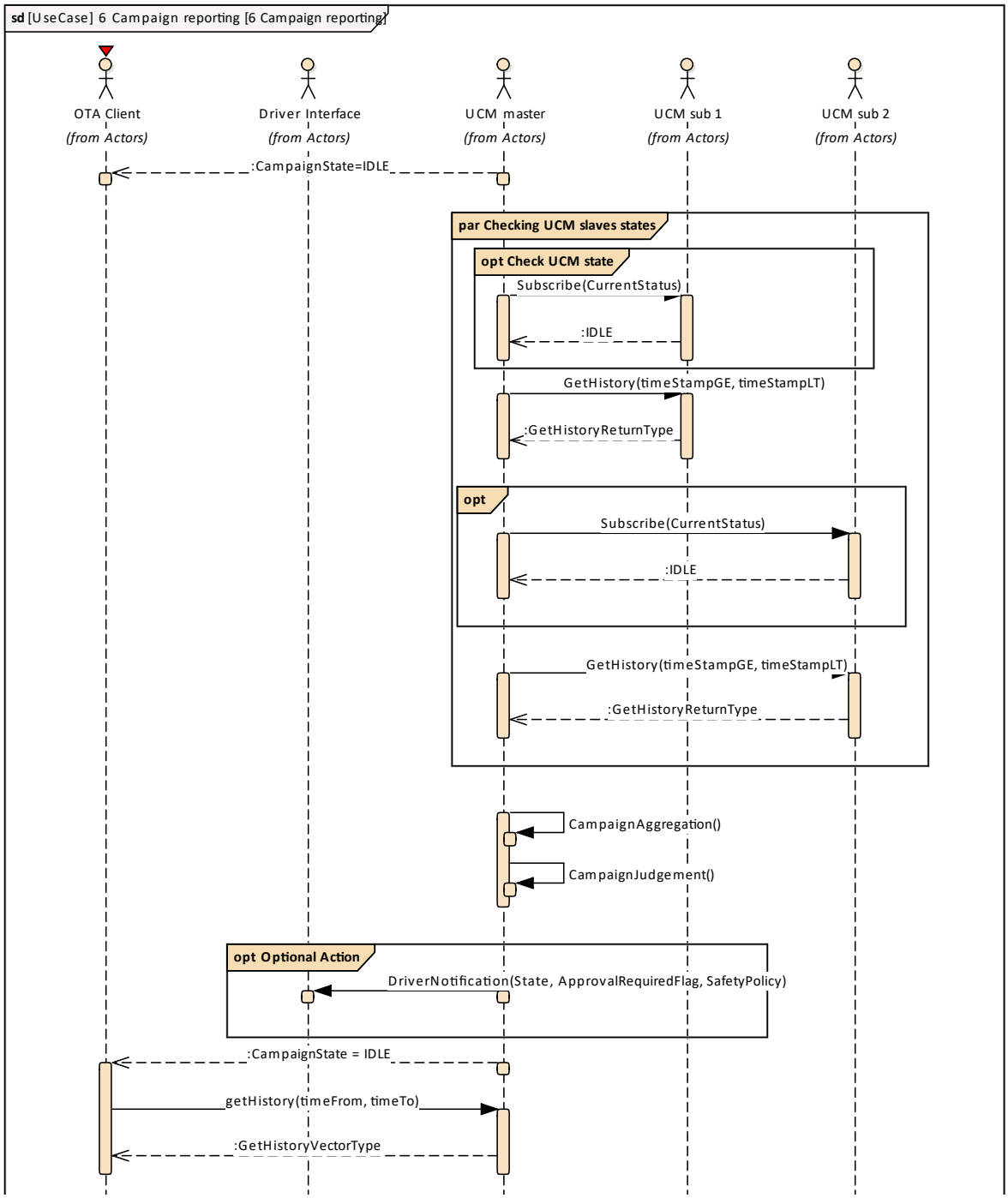

Figure D.8: Campaign reporting to backend

E Security Analysis of Installation and Update

This chapter presents a summary for the security analysis of the UCM. Some of the threats could not be addressed by specifying AUTOSAR requirements. The main reason for not specifying the countermeasures is to allow vendors to flexibly decide on the solution that fits their setup. Here we aim to raise awareness and provide advice on the selected topics:

E.1 Securing Software Package

UCM is responsible for applying changes of the platform and applications contained in the Software Packages it receives. Therefore, integrity and authenticity of Software Packages are critical to protect system integrity. It shall be ensured that the Software Packages are neither illegitimately altered nor issued by unauthorized parties. This can be achieved by applying cryptographic techniques such as digital signatures. The period that Software Package resides in UCM before being activated shall not be neglected. It provides a window of opportunity for an attacker to tamper with the Software Package after the authentication is done at TransferExit.

Information disclosure is another security threat category that might be applicable to Software Packages. Packages that contain sensitive information, such as intellectual properties or cryptographic keys, require confidentiality protection in addition to integrity and authenticity when being persisted or transmitted over a communication channel.

Another aspect of protecting Software Update Packages is their freshness. An attacker may try to manipulate the system by downgrading the software via replaying an authentic but older Software Update Package. In this regard, the platform shall ensure that only newer packages (i.e. packages that contain newer version of installed SWCL) can be installed.

E.2 Securing Calls to UCM

UCM provides a very critical functionality in the platform that allows modifying applications and platform components. In that sense, it is critical to prevent unauthorized access to UCM, meaning only legitimate callers should be allowed to reach the UCM service interface. This is primarily enforced in the communication layer supported by the Identity and Access Management. Additionally, the calls to the UCM interface shall be protected against altering, e.g. changing API arguments. When the service and client reside on the same machine, the security relies on the integrity of the operating system and the platform. In case, the service and the client are running on different machines, a secure communication, assuring authenticity and integrity of communication, is additionally required.

Moreover, some API methods of the UCM interface returns sensitive information about the platform. This subset (GetSwClusterInfo, GetSwClusterChangeInfo, GetLog, GetHistory, GetSwPackages) shall be protected against information disclosure and should only be reachable over a channel that provides confidentiality.

A similar reasoning is applicable for securing the communication between UCM Master and its clients. Regarding protection against information disclosure, GetSwClusterInfo, SwPackageInventory and GetHistory for UCM Master shall only be called over confidential channels.

E.3 Suppressing Call to UCM

Multiple scenarios can be envisioned where an attacker targets suppressing the calls to UCM. The attack could block the calls to or the response from UCM. In both cases the caller of the service may assume that UCM is not responding and retries its request. This would lead to undesired overhead on the system. For such scenarios, it is recommended that both UCM and the UCM Client consider reporting security events when same calls repeatedly received at UCM or calls repeatedly fail at the caller side. This information could potentially be picked up by Intrusion Detection Systems or Anomaly Detection Systems.

E.4 Resource Starvation

According to the current specification, the available resources for transferring a Software Package is only checked when TransferStart is called but not reserved. This means, while the transfer is ongoing, the system storage can be exhausted by other processes using the same storage media. This scenario is also applicable to UCM Master when receiving data from its client. A similar case is possible for processing of Software Package, as the resources are only checked at the beginning but not reserved. In this regard, a solution could be to reserve the necessary resources for the Software Package transfer or processing from the beginning to prevent attacks aiming at such scenarios.

At the same time, reserving the resources might provide opportunity to the attacker in other scenarios. The specification allows transferring multiple Software Packages in parallel. Consequently, a misbehaving or compromised client can open unlimited number of transfer sessions causing UCM to run out of resources. To cope with this scenario, a threshold for the number of parallel transfer sessions can be defined.

E.5 Zombie Sessions

The AUTOSAR specification does not enforce any expiry time for the established transfer sessions. As a result, the resources that are hold by an ongoing session will not

be released no matter how long time it takes. At the same time, in certain cases it may take a long time for larger software packages to be transferred to UCM or UCM Master, especially when they are received from external sources with weak connectivity on-the-fly. However, a timeout may be considered for such a transfer to prevent attackers from mounting denial of service attacks by long term allocation of resources.

F History of Specification Items

Please note that the lists in this chapter also include specification items that have been removed from the specification in a later version. These specification items do not appear as hyperlinks in the document.

F.1 Specification Item History of this document compared to AUTOSAR R19-03.

F.1.1 Added Traceables in R19-11

Number	Heading
[SWS_UCM_00009]	UCM exposing its identifier
[SWS_UCM_00105]	UCM confidential information handling
[SWS_UCM_00161]	Check Software Package version compatibility against UCM version
[SWS_UCM_00162]	Entering the Cleaning-up state after a RevertProcessedSwPackages call
[SWS_UCM_00163]	Action in Cleaning-up state
[SWS_UCM_00164]	Cleaning up of Software Packages
[SWS_UCM_00165]	Processing from stream
[SWS_UCM_00166]	Processing from stream state
[SWS_UCM_00167]	Cancelling streamed packages
[SWS_UCM_00168]	Transferring while processing from stream
[SWS_UCM_00169]	Finishing transfer while processing from stream
[SWS_UCM_00170]	Log message retrieving
[SWS_UCM_00171]	Log level changing
[SWS_UCM_00172]	Log messages removing
[SWS_UCM_00173]	UCMIdentifierType table
[SWS_UCM_00174]	SwNameVectorType table
[SWS_UCM_00175]	StrongRevisionLabelString table
[SWS_UCM_00176]	SwNameVersionType table
[SWS_UCM_00177]	SwNameVersionVectorType table
[SWS_UCM_00178]	ProvidedPort VehiclePackageManagement
[SWS_UCM_00179]	RequiredPort VehicleStateManager
[SWS_UCM_00180]	RequiredPort VehicleDriverApplication
[SWS_UCM_00181]	ProvidedInterface VehiclePackageManagement
[SWS_UCM_00182]	RequiredInterface VehicleDriverApplication
[SWS_UCM_00183]	RequiredInterface VehicleStateManager

▽

△

Number	Heading
[SWS_UCM_00210]	Transferring of software packages on `kProcessApproving` or `kProcessing` state
[SWS_UCM_01001]	`UCM Master` processes `Vehicle Package`
[SWS_UCM_01002]	`UCM Master` shall provide `UCM services`
[SWS_UCM_01003]	`UCM Master` checks states of `UCM` subordinates
[SWS_UCM_01004]	Only one `UCM Master` shall be active per network domain
[SWS_UCM_01005]	`UCM Master` is discovering `UCM`s in vehicle
[SWS_UCM_01006]	`Vehicle Package` transfer to `UCM Master`
[SWS_UCM_01007]	Start transfer of a `Vehicle Package` or `Software Package`to `UCM Master`
[SWS_UCM_01008]	Transfer data of a `Vehicle Package` to `UCM Master`
[SWS_UCM_01009]	Exit the transfer of a `Vehicle Package` to `UCM Master`
[SWS_UCM_01010]	Delete a `Vehicle Package` transferred to `UCM Master`
[SWS_UCM_01101]	Provide information of installed `Software Clusters` in vehicle
[SWS_UCM_01102]	Get information of available `Software Clusters` in `Backend`
[SWS_UCM_01103]	Inform `Backend` of needed `Software Clusters` for an update
[SWS_UCM_01105]	Interaction of `UCM Master` with Vehicle Driver
[SWS_UCM_01106]	Exclusive use of Vehicle Driver Interface
[SWS_UCM_01107]	`UCM Master` provides progress information to Vehicle Driver
[SWS_UCM_01108]	Unsupported safety policy by Vehicle driver interface
[SWS_UCM_01109]	Vehicle State Manager shall provide to `UCM Master` a safety state
[SWS_UCM_01110]	`UCM Master` shall be able to set the safety policy to be computed by Vehicle State Manager
[SWS_UCM_01111]	Exclusive use of Vehicle State Manager
[SWS_UCM_01112]	Unsupported safety policy by Vehicle State Manager
[SWS_UCM_01113]	Switching vehicle into update mode
[SWS_UCM_01114]	SafetyPolicyType table
[SWS_UCM_01115]	VehicleStateManagerErrorDomain
[SWS_UCM_01116]	VehicleDriverApplicationErrorDomain
[SWS_UCM_01177]	CampaignStateType table
[SWS_UCM_01201]	Sequential orchestration of campaigns
[SWS_UCM_01203]	CampaignState field
[SWS_UCM_01204]	Initial state
[SWS_UCM_01205]	`UCM Master` internal state persistency
[SWS_UCM_01206]	Trigger on `kTransferApproving` state
[SWS_UCM_01207]	Trigger on `kTransferring` state
[SWS_UCM_01208]	Trigger on `kProcessApproving` state
[SWS_UCM_01209]	Trigger on `kProcessing` state

▽

△

Number	Heading
[SWS_UCM_01211]	Trigger on `kActivateApproving` state
[SWS_UCM_01212]	Trigger on `kActivating` state
[SWS_UCM_01213]	Trigger on `kVehicleChecking` state
[SWS_UCM_01214]	Final action on `kVehicleChecking` state
[SWS_UCM_01215]	Trigger on `kRollingBack` state
[SWS_UCM_01216]	Final action on `kRollingBack` state
[SWS_UCM_01217]	Monitoring of UCM subordinates
[SWS_UCM_01218]	Transition from `kIdle` state to `kSyncing` state
[SWS_UCM_01219]	Transition from `kSyncing` state to `kIdle` state
[SWS_UCM_01220]	Transition from `kIdle` state to `kVehiclePackageTransferring` state
[SWS_UCM_01221]	Transition from `kVehiclePackageTransferring` state to `kIdle` state
[SWS_UCM_01222]	Transition from `kVehiclePackageTransferring` state to `kTransferring` state
[SWS_UCM_01223]	Transition from `kVehiclePackageTransferring` state to `kTransferApproving` state
[SWS_UCM_01224]	Transition from `kTransferApproving` state to `kTransferring` state
[SWS_UCM_01225]	Transition from `kTransferApproving` state to `kIdle` state
[SWS_UCM_01226]	Transition from `kTransferring` state to `kTransferApproving` state
[SWS_UCM_01227]	Transition from `kTransferring` state to `kIdle` state
[SWS_UCM_01228]	Transition from `kTransferring` state to `kProcessing` state
[SWS_UCM_01229]	SafetyPolicy while processing stream
[SWS_UCM_01230]	Transition from `kTransferring` state to `kProcessApproving` state
[SWS_UCM_01231]	Transition from `kProcessApproving` state to `kProcessing` state
[SWS_UCM_01232]	Transition from `kProcessApproving` state to `kIdle` state
[SWS_UCM_01233]	Transition from `kProcessing` state to `kProcessApproving` state
[SWS_UCM_01234]	Transition from `kProcessing` state to `kActivating` state
[SWS_UCM_01235]	Transition from `kProcessing` state to `kActivateApproving` state
[SWS_UCM_01236]	Transition from `kProcessing` state to `kIdle` state
[SWS_UCM_01237]	Transition from `kActivateApproving` state to `kActivating` state
[SWS_UCM_01238]	Transition from `kActivateApproving` state to `kIdle` state
[SWS_UCM_01239]	Transition from `kActivating` state to `kRollingBack` state
[SWS_UCM_01240]	Transition from `kActivating` state to `kVehicleChecking` state
[SWS_UCM_01241]	Transition from `kVehicleChecking` state to `kRollingBack` state
[SWS_UCM_01242]	Transition from `kVehicleChecking` state to `kIdle` state
[SWS_UCM_01243]	Transition from `kRollingBack` state to `kIdle` state
[SWS_UCM_01244]	Cancellation of an update campaign shall be possible
[SWS_UCM_01245]	Cancellation during activation shall be possible
[SWS_UCM_01246]	Unreachable UCM during update campaign

▽

△

Number	Heading
[SWS_UCM_01247]	Method to read History Report
[SWS_UCM_01248]	Content of History Report
[SWS_UCM_01301]	`Vehicle Package` authentication
[SWS_UCM_01302]	`Vehicle Package` authentication failure
[SWS_UCM_01303]	Dependencies between `Software Packages`
[SWS_UCM_01304]	Confidential information protection
[SWS_UCM_CONSTR_00001]	

Table F.1: Added Traceables in R19-11

F.1.2 Changed Traceables in R19-11

Number	Heading
[SWS_UCM_00003]	Cancelling the package processing
[SWS_UCM_00017]	Sequential `Software Package` Processing
[SWS_UCM_00018]	Providing Progress Information
[SWS_UCM_00027]	Delta Package activation
[SWS_UCM_00071]	SwNameType table
[SWS_UCM_00081]	Processing state of Package Management
[SWS_UCM_00082]	Exit from Processing state of Package Management
[SWS_UCM_00102]	Update state
[SWS_UCM_00103]	Update to older `Software Cluster` version than currently present
[SWS_UCM_00104]	Consistency Check of processed Package
[SWS_UCM_00111]	Entering the Rolled-back state
[SWS_UCM_00112]	`Software Cluster` and version
[SWS_UCM_00126]	Entering the RollingBack state after a Rollback call
[SWS_UCM_00130]	`Software Cluster` and version error
[SWS_UCM_00146]	Entering the Cleaning-up state after a Finish call
[SWS_UCM_00149]	Return to the Idle state from Processing state
[SWS_UCM_00151]	Entering the Ready state of Package Management after a Cancel call
[SWS_UCM_00155]	Entering the RollingBack state after a failure in the Verifying state

Table F.2: Changed Traceables in R19-11

F.1.3 Deleted Traceables in R19-11

Number	Heading
[SWS_UCM_00012]	Log message retrieving
[SWS_UCM_00114]	ActivateOptionType table
[SWS_UCM_00144]	Log error

Table F.3: Deleted Traceables in R19-11