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	R20-11

Document Change History

Date	Release	Changed by	Description
2020-11-30	R20-11	AUTOSAR Release Management	• Classic Plaftorm update specification for UCM Master • Refactored UCM Master API • Simplified UCM Master State Machine • Detailed campaign history information
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

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
VCI	Vehicle Communication Interface
MVCI	Modular Vehicle Communication Interface
D-PDU API	Diagnostic Protocol Data Unit Application Programming Interface
RDF	Root Description File
MDF	Module Description File

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
MachineFG	see [2] AUTOSAR Execution Management
State Management	see [3] AUTOSAR State Management
Function Group	see [3] AUTOSAR State Management
Communication Management	see [4] AUTOSAR Communication 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] Specification of State Management
AUTOSAR_SWS_StateManagement

[4] Specification of Communication Management
AUTOSAR_SWS_CommunicationManagement

[5] General Requirements specific to Adaptive Platform
AUTOSAR_RS_General

[6] Specification of Cryptography for Adaptive Platform
AUTOSAR_SWS_Cryptography

[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] Specification of Platform Health Management for Adaptive Platform
AUTOSAR_SWS_PlatformHealthManagement

3.2 Related specification

See chapter 3.1.

3.3 Further applicable specification

AUTOSAR provides a general specification [5] which is also applicable for UCM. The specification RS General shall be considered as additional and required specification for implementation of UCM.

4 Constraints and assumptions

4.1 Known 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 updatable 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.

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.

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.

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.

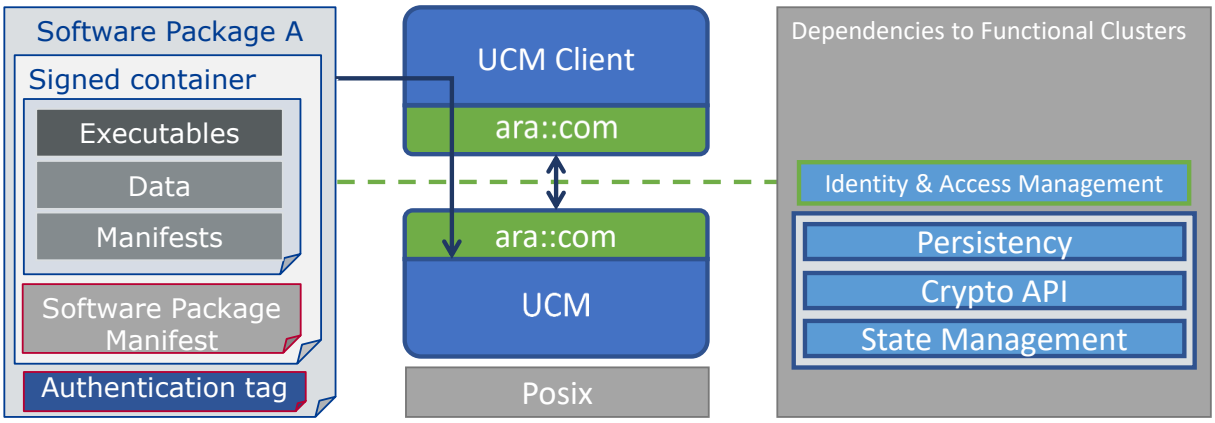

Figure 5.1: UCM dependencies to other Functional Clusters.

5.1 Interfaces to Adaptive State Management

UCM relies on State Management and its provided UpdateRequest Service Interface to perform the necessary Function Group state changes needed to activate the newly installed, updated or removed software.

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, 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 Adaptive Application accessing the UCM, should make sure that the platform is switched to this state (using interfaces from State Management), before starting the update.

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 [6] to verify package integrity and authenticity and to decrypt confidential update data.

5.4 Interfaces to Identity and Access Management

Identity and Access Management [7] controls the UCM's Clients access 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_EM_00014]	Execution Management shall support a Trusted Platform.	[SWS_UCM_00202]
[RS_SM_00001]	State Management shall coordinate and control multiple sets of Applications.	[SWS_UCM_00242]
[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] [SWS_UCM_00240]
[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_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_00185] [SWS_UCM_00186] [SWS_UCM_00187] [SWS_UCM_00190] [SWS_UCM_01114] [SWS_UCM_CONSTR_00001] [SWS_UCM_CONSTR_00002]
[RS_UCM_00003]	UCM shall support updating installed software on Adaptive Platform	[SWS_UCM_00017] [SWS_UCM_00165] [SWS_UCM_00257]

Requirement	Description	Satisfied by
[RS_UCM_00004]	UCM shall support uninstalling software on AUTOSAR Adaptive Platform	[SWS_UCM_00001] [SWS_UCM_00137] [SWS_UCM_00165] [SWS_UCM_00184]
[RS_UCM_00005]	UCM shall make sure that persistent data owned by uninstalled software is deleted	[SWS_UCM_00001] [SWS_UCM_00137]
[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] [SWS_UCM_00200] [SWS_UCM_00209] [SWS_UCM_00230] [SWS_UCM_00250]
[RS_UCM_00007]	UCM shall check that software dependencies are fulfilled	[SWS_UCM_00026] [SWS_UCM_00027] [SWS_UCM_00120] [SWS_UCM_00136] [SWS_UCM_00161] [SWS_UCM_00201] [SWS_UCM_00231] [SWS_UCM_00232] [SWS_UCM_00260]
[RS_UCM_00008]	UCM shall support a recovery mechanism in case of failed update process	[SWS_UCM_00005] [SWS_UCM_00024] [SWS_UCM_00107] [SWS_UCM_00110] [SWS_UCM_00111] [SWS_UCM_00126] [SWS_UCM_00127] [SWS_UCM_00131] [SWS_UCM_00146] [SWS_UCM_00155] [SWS_UCM_00162] [SWS_UCM_00163] [SWS_UCM_00164] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183] [SWS_UCM_00264]

Requirement	Description	Satisfied by
[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_00131] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183] [SWS_UCM_CONSTR_00001] [SWS_UCM_CONSTR_00002]
[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_00131] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183] [SWS_UCM_00185] [SWS_UCM_00186] [SWS_UCM_00187] [SWS_UCM_00191] [SWS_UCM_00192] [SWS_UCM_00193] [SWS_UCM_00194] [SWS_UCM_00195] [SWS_UCM_00196] [SWS_UCM_00197] [SWS_UCM_00198] [SWS_UCM_00199] [SWS_UCM_CONSTR_00001] [SWS_UCM_CONSTR_00002]
[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] [SWS_UCM_00207] [SWS_UCM_00209] [SWS_UCM_00213] [SWS_UCM_01306]

Requirement	Description	Satisfied by
[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_00092] [SWS_UCM_00098] [SWS_UCM_00136] [SWS_UCM_00140] [SWS_UCM_00145] [SWS_UCM_00206] [SWS_UCM_00217] [SWS_UCM_00243] [SWS_UCM_01011] [SWS_UCM_01012]
[RS_UCM_00014]	UCM shall check that correct amount of data has been transferred for the Software Package	[SWS_UCM_00136] [SWS_UCM_00204] [SWS_UCM_00205] [SWS_UCM_00211] [SWS_UCM_00243]
[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_00184]
[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] [SWS_UCM_00259]

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_00092] [SWS_UCM_00093] [SWS_UCM_00098] [SWS_UCM_00140] [SWS_UCM_00145] [SWS_UCM_00148] [SWS_UCM_00203] [SWS_UCM_00204] [SWS_UCM_00205] [SWS_UCM_00206] [SWS_UCM_00208] [SWS_UCM_00212] [SWS_UCM_00214] [SWS_UCM_00215] [SWS_UCM_00216]
[RS_UCM_00020]	UCM shall support cancellation of an update or install operation	[SWS_UCM_00003] [SWS_UCM_00167] [SWS_UCM_00233] [SWS_UCM_00234] [SWS_UCM_00235] [SWS_UCM_00236] [SWS_UCM_00237] [SWS_UCM_00238] [SWS_UCM_00239]
[RS_UCM_00021]	UCM shall support atomic activation of installed or updated Software Clusters	[SWS_UCM_00022] [SWS_UCM_00025] [SWS_UCM_00094] [SWS_UCM_00131] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183] [SWS_UCM_00241] [SWS_UCM_00259] [SWS_UCM_00260]
[RS_UCM_00022]	UCM shall support logging of the update or installation process	[SWS_UCM_00131] [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] [SWS_UCM_00220]

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_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] [SWS_UCM_00258]
[RS_UCM_00025]	UCM shall support receiving 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_00092] [SWS_UCM_00098] [SWS_UCM_00131] [SWS_UCM_00140] [SWS_UCM_00145] [SWS_UCM_00165] [SWS_UCM_00166] [SWS_UCM_00167] [SWS_UCM_00168] [SWS_UCM_00169] [SWS_UCM_00181] [SWS_UCM_00182] [SWS_UCM_00183] [SWS_UCM_00217] [SWS_UCM_00219] [SWS_UCM_00243]

Requirement	Description	Satisfied by
[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] [SWS_UCM_00184] [SWS_UCM_00218] [SWS_UCM_00219] [SWS_UCM_00240] [SWS_UCM_00257] [SWS_UCM_00258] [SWS_UCM_00261] [SWS_UCM_00262] [SWS_UCM_00263]
[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] [SWS_UCM_00245]
[RS_UCM_00029]	UCM shall support updating the underlying Operating System	[SWS_UCM_00101] [SWS_UCM_00245]
[RS_UCM_00030]	UCM shall be able to verify the updated software during activation	[SWS_UCM_00107] [SWS_UCM_00111] [SWS_UCM_00126] [SWS_UCM_00127] [SWS_UCM_00146] [SWS_UCM_00155] [SWS_UCM_00162] [SWS_UCM_00163] [SWS_UCM_00164] [SWS_UCM_00260] [SWS_UCM_00264]
[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_00190] [SWS_UCM_01177] [SWS_UCM_01178]
[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_01120] [SWS_UCM_01218] [SWS_UCM_01304]

Requirement	Description	Satisfied by
[RS_UCM_00034]	UCM Master shall record all UCM Master's action history	[SWS_UCM_00251] [SWS_UCM_00252] [SWS_UCM_00253] [SWS_UCM_00254] [SWS_UCM_00255] [SWS_UCM_00256] [SWS_UCM_01247] [SWS_UCM_01248] [SWS_UCM_01266] [SWS_UCM_01267] [SWS_UCM_01268] [SWS_UCM_01269]
[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_01013] [SWS_UCM_01119] [SWS_UCM_01121] [SWS_UCM_01122] [SWS_UCM_01123] [SWS_UCM_01124] [SWS_UCM_01125] [SWS_UCM_01126] [SWS_UCM_01127] [SWS_UCM_01128] [SWS_UCM_01129] [SWS_UCM_01130] [SWS_UCM_01131] [SWS_UCM_01132] [SWS_UCM_01133] [SWS_UCM_01134] [SWS_UCM_01204] [SWS_UCM_01205]

Requirement	Description	Satisfied by
		[SWS_UCM_01207] [SWS_UCM_01209] [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_01227] [SWS_UCM_01228] [SWS_UCM_01229] [SWS_UCM_01234] [SWS_UCM_01236] [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_01270] [SWS_UCM_01271] [SWS_UCM_01303] [SWS_UCM_01305] [SWS_UCM_CONSTR_00003] [SWS_UCM_CONSTR_00005] [SWS_UCM_CONSTR_00006] [SWS_UCM_CONSTR_00009] [SWS_UCM_CONSTR_00010] [SWS_UCM_CONSTR_00011]
[RS_UCM_00036]	UCM Master shall use platform communication services for interacting with UCM subordinates	[SWS_UCM_00009] [SWS_UCM_00173] [SWS_UCM_01005] [SWS_UCM_01007] [SWS_UCM_01008] [SWS_UCM_01009] [SWS_UCM_01010] [SWS_UCM_01015] [SWS_UCM_01016]

Requirement	Description	Satisfied by
[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_01109] [SWS_UCM_01110] [SWS_UCM_01117] [SWS_UCM_01222] [SWS_UCM_01228] [SWS_UCM_01229] [SWS_UCM_01234] [SWS_UCM_01240] [SWS_UCM_01244] [SWS_UCM_01245] [SWS_UCM_01246] [SWS_UCM_CONSTR_00003] [SWS_UCM_CONSTR_00004] [SWS_UCM_CONSTR_00005] [SWS_UCM_CONSTR_00006] [SWS_UCM_CONSTR_00007] [SWS_UCM_CONSTR_00008] [SWS_UCM_CONSTR_00009]
[RS_UCM_00038]	UCM Master shall interact with driver	[SWS_UCM_00180] [SWS_UCM_01105] [SWS_UCM_01107] [SWS_UCM_01117] [SWS_UCM_01118] [SWS_UCM_01120] [SWS_UCM_01222] [SWS_UCM_01228] [SWS_UCM_01234]
[RS_UCM_00039]	UCM Master shall prevent processing of compromised Vehicle Packages	[SWS_UCM_00200] [SWS_UCM_01001] [SWS_UCM_01221] [SWS_UCM_01301] [SWS_UCM_01302]
[RS_UCM_00042]	UCM Master shall provide an interface to read the state of an update campaign	[SWS_UCM_01017] [SWS_UCM_01203] [SWS_UCM_01205] [SWS_UCM_01265]

Requirement	Description	Satisfied by
[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_01014] [SWS_UCM_01015] [SWS_UCM_01016] [SWS_UCM_01201] [SWS_UCM_01207] [SWS_UCM_01209] [SWS_UCM_01212] [SWS_UCM_01228] [SWS_UCM_01301] [SWS_UCM_01302] [SWS_UCM_01303] [SWS_UCM_01305]

7 Functional specification

7.1 UCM

7.1.1 Software Cluster lifecycle

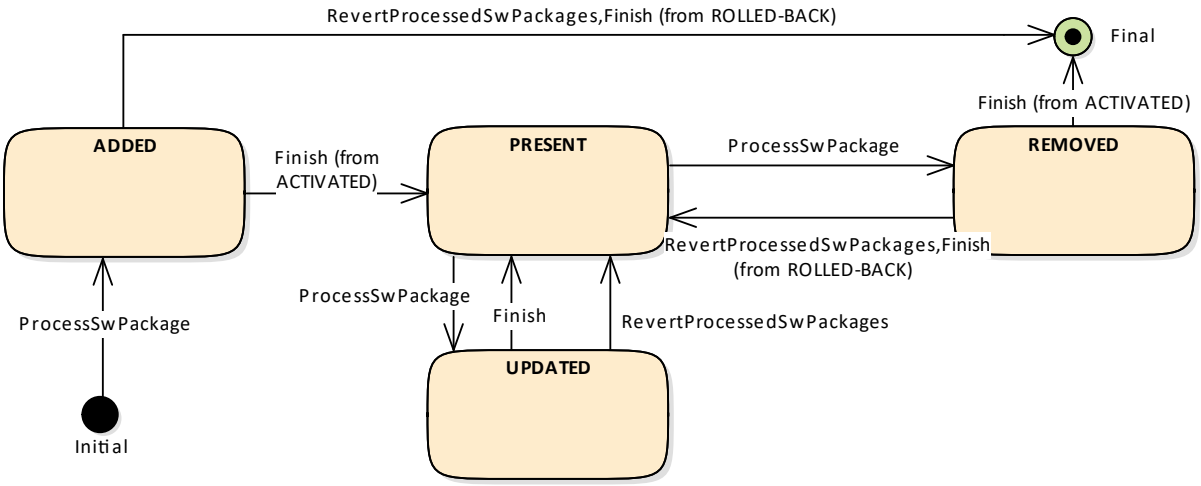

Figure 7.1: State Machine for a `Software Cluster`

The state machine in Fig. 7.1 describes the life-cycle states of a `Software Cluster`. These states are reported with `GetSwClusterInfo` method.

[SWS_UCM_00191]{DRAFT} **`Software Cluster` life-cycle state `kAdded`** ⌈A `Software Cluster` state shall be kAdded after the `Software Cluster` is successfully processed with `ProcessSwPackage` method call on the `AUTOSAR Adaptive Platform` and if it was not previously present in the `AUTOSAR Adaptive Platform`.⌋*(RS_UCM_00011)*

[SWS_UCM_00192]{DRAFT} **`Software Cluster` life-cycle state transition from `kAdded` to `kPresent`** ⌈A `Software Cluster` state shall change from `kAdded` to `kPresent` after a successful activation of a newly added `Software Cluster` with `Finish` method call.⌋*(RS_UCM_00011)*

[SWS_UCM_00193]{DRAFT} **`Software Cluster` life-cycle state transition from `kUpdated` to `kPresent`** ⌈A `Software Cluster` state shall change from `kUpdated` to `kPresent` after a successful activation of the updated `Software Cluster` with `Finish` method call.⌋*(RS_UCM_00011)*

[SWS_UCM_00194]{DRAFT} **`Software Cluster` life-cycle state transition from `kRemoved` to `kPresent`** ⌈A `Software Cluster` state shall change from `kRemoved` to `kPresent` after a successful call to `RevertProcessedSwPackages` method (in case the `Software Cluster` was previously requested to be removed by `ProcessSwPackage` method call) or `Finish` method (in case a `Software Cluster` being removed has to be rolled back after a failing activation).⌋*(RS_UCM_00011)*

[SWS_UCM_00195]{DRAFT} **Software Cluster life-cycle state kUpdated** ⌈A Software Cluster state shall be kUpdated after a successful processing of the updated Software Cluster with ProcessSwPackage method call.⌋*(RS_UCM_00011)*

[SWS_UCM_00196]{DRAFT} **Software Cluster life-cycle state kRemoved** ⌈A Software Cluster state shall be kRemoved after successful completion of method ProcessSwPackage which involves the removal of the existed Software Cluster.⌋*(RS_UCM_00011)*

[SWS_UCM_00197]{DRAFT} **End of Software Cluster life-cycle state from state kAdded** ⌈A Software Cluster shall reach the end of its life-cycle from kAdded after a successful removal of a newly added Software Cluster with RevertProcessedSwPackages method call (in case the Software Cluster was previously requested to be added by ProcessSwPackage method call) or Finish method call (in case the newly added Software Cluster has to be rolled back after a failing activation).⌋*(RS_UCM_00011)*

[SWS_UCM_00198]{DRAFT} **End of Software Cluster life-cycle state from state kRemoved** ⌈A Software Cluster shall reach the end of its life-cycle if it is successfully removed with a Finish method call and the Software Cluster is in state kRemoved.⌋*(RS_UCM_00011)*

[SWS_UCM_00199]{DRAFT} **Reporting of Software Cluster reaching end of life-cycle** ⌈Any Software Cluster reaching the end of its life-cycle shall not be reported by UCM any more.⌋*(RS_UCM_00011)*

7.1.2 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 7.2, 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.

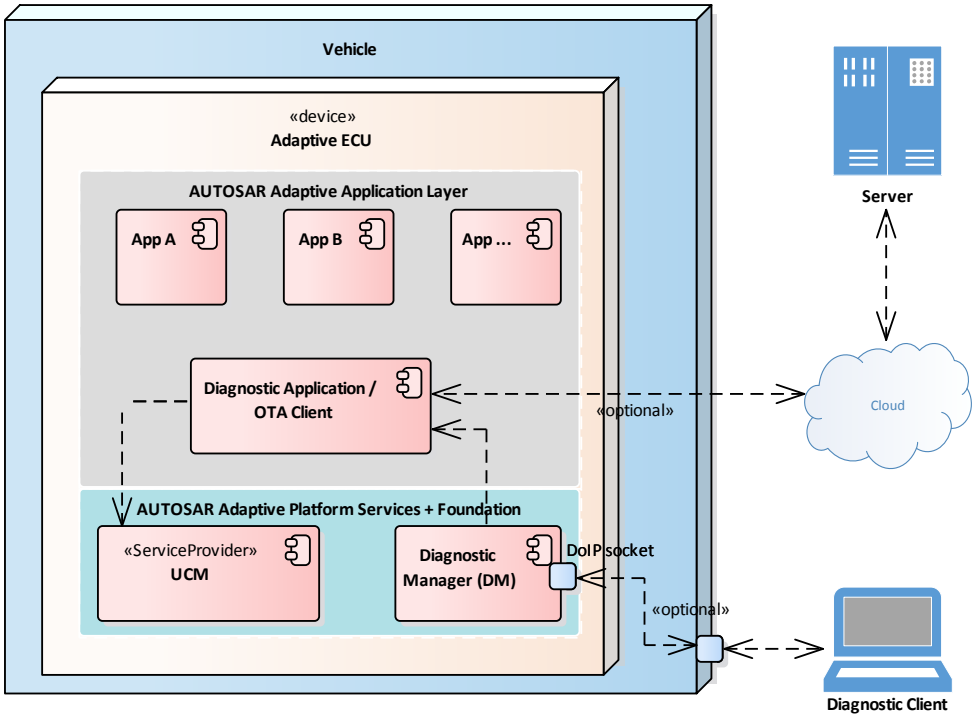

Figure 7.2: Architecture overview for diagnostic use case

7.1.2.1 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 7.1.3.
- Software Package processing: A phase in which the UCM performs the operation (kInstall, kUpdate, kRemove) on the relevant SoftwareCluster. For further information see chapter 7.1.5.
- 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) prior to finishing the update. For further information see chapter 7.1.6

7.1.2.1.1 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.

A SoftwareCluster can be of the following categories expressed by the attribute SoftwareCluster.category :

- APPLICATION_LAYER: the SoftwareCluster can be removed by UCM
- PLATFORM_CORE: the SoftwareCluster cannot be removed as it would break the system.
- PLATFORM: the SoftwareCluster is part of the platform software and can be removed

[SWS_UCM_00245]{DRAFT} **Software Cluster category** ⌈UCM shall not remove a SoftwareCluster that has category set to PLATFORM_CORE.⌋*(RS_UCM_00028, RS_UCM_00029)*

A Software Package has to be modelled as a so-called SoftwareCluster which describes the content of a Software Package that is downloaded or 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].

7.1.2.1.2 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 the example sketched in Figure 7.3.

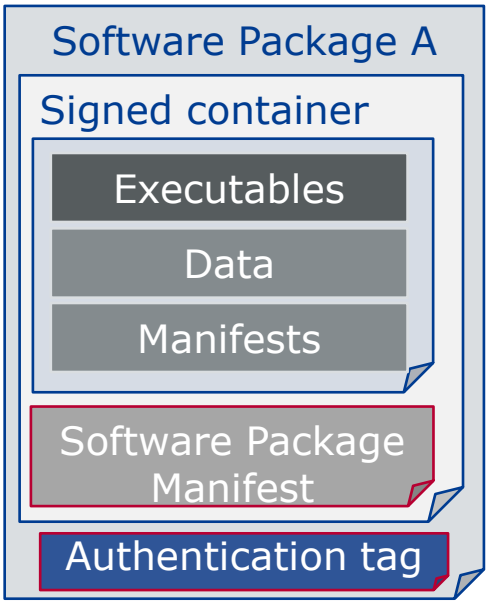

Figure 7.3: `Software Package` content description

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]. For more details on the `Software Package` class, you can refer to `SoftwarePackage`

[SWS_UCM_00112] `Software Cluster` and version ⌈`SoftwareCluster`'s manifest shall include a name and a version following description of `StrongRevisionLabelString`.⌋*(RS_UCM_00002)*

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

If the constraint is violated, an error will be raised according to [SWS_UCM_00103].

A higher version number is achieved by an increment of the MajorVersion, the MinorVersion, or the PatchVersion.

If there is a need to downgrade a failing `SoftwareCluster` (for instance, malfunction in the field that was not detected at activation), it will therefore be needed to repackage the same old `SoftwareCluster` that was properly working with an higher version number.

[SWS_UCM_00190]{DRAFT} **Reinstallation of older `Software Cluster` version than previously removed** ⌈New `Software Cluster`s getting installed shall be compared with the history of all installed `Software Cluster`s to prevent installation of

a Software Cluster with a lower or equal version than previously installed.⌋(*RS_UCM_00002*, *RS_UCM_00032*)

[SWS_UCM_00130] Software Cluster and version error ⌈If SoftwareCluster's manifest does not contain any SoftwareCluster.version as specified in [SWS_UCM_00112] [SWS_UCM_00190], UCM shall raise the ApplicationError InvalidPackageManifest.⌋(*RS_UCM_00002*)

7.1.2.1.3 Applications Persisted Data

Updating and rolling back of persisted data is handled completely by the application using persistency without involvement of UCM. A detailed explanation can be found in the Persistency Specification [11]. An exception here is the removal of persistent data after a SoftwareCluster is removed.

[SWS_UCM_00184]{DRAFT} **Persistent data clean-up after Software Cluster removal** ⌈UCM shall remove persistent data of a removed SoftwareCluster by aggregating the information given in the application manifest, namely PersistencyKeyValueStorage.uri and PersistencyFileStorage.uri, in order to leave the AUTOSAR Adaptive Platform and the file system clean.⌋(*RS_UCM_00026*, *RS_UCM_00017*, *RS_UCM_00004*)

For more details, please refer to [SWS_PER_00397] in Persistency Specification [11].

7.1.2.2 Runtime dependencies

Processes within a SoftwareCluster can have functional 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.

7.1.2.3 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 an updatable 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 a predefined 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_00257]{DRAFT} **Update session** ⌈To confirm the system is in an updatable state, `UCM` shall start an update session by calling `State Management` `UpdateRequest` Service Interface `StartUpdateSession` method after its dependency check triggered by `Activate` method call.⌋*(RS_UCM_00026, RS_UCM_00003)*

[SWS_UCM_00258]{DRAFT} **Update session rejected** ⌈If `State Management` `UpdateRequest` Service Interface `StartUpdateSession` method call raises error `kRejected`, `UCM` shall transition from `kActivating` to `kReady` states and `Activate` method call shall return `ApplicationError` `UpdateSessionRejected`.⌋*(RS_UCM_00026, RS_UCM_00024)*

If update session could be recurrently rejected, it is up to implementer to cache the dependency check result in order to avoid unnecessary computation and compute it only once.

During the update session, the minimum applications required for the Update process should be 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.

Update of some components require a Machine reset to be performed. These components should be configured to be part of `Function Group` `MachineFG`, as the update sequence of `Function Group` `MachineFG` includes a `Machine` reset. `Execution Management`, `State Management`, `Communication Management` 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 `Function Group` `MachineFG` can be found in `State Management`.

7.1.3 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 machines in Fig. 7.4 and Fig. 7.5 specify 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.

The UCM has the possibility to keep the Software Package in kTransferred states in case it failed and retry later: transferring Software Package can be costly, if it is authenticated, there could be no reason to delete it if the update has not been successfully finished.

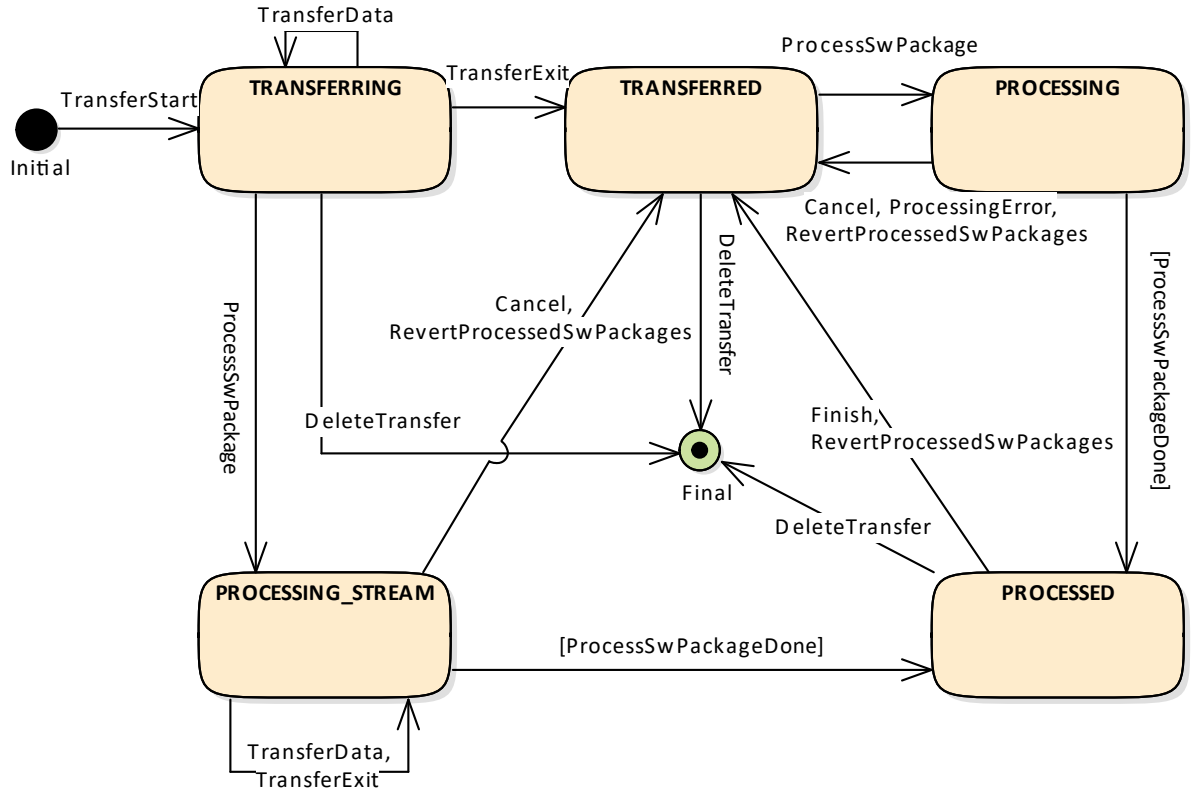

Figure 7.4: State Machine representing Software Packages lifecycle, with storing option

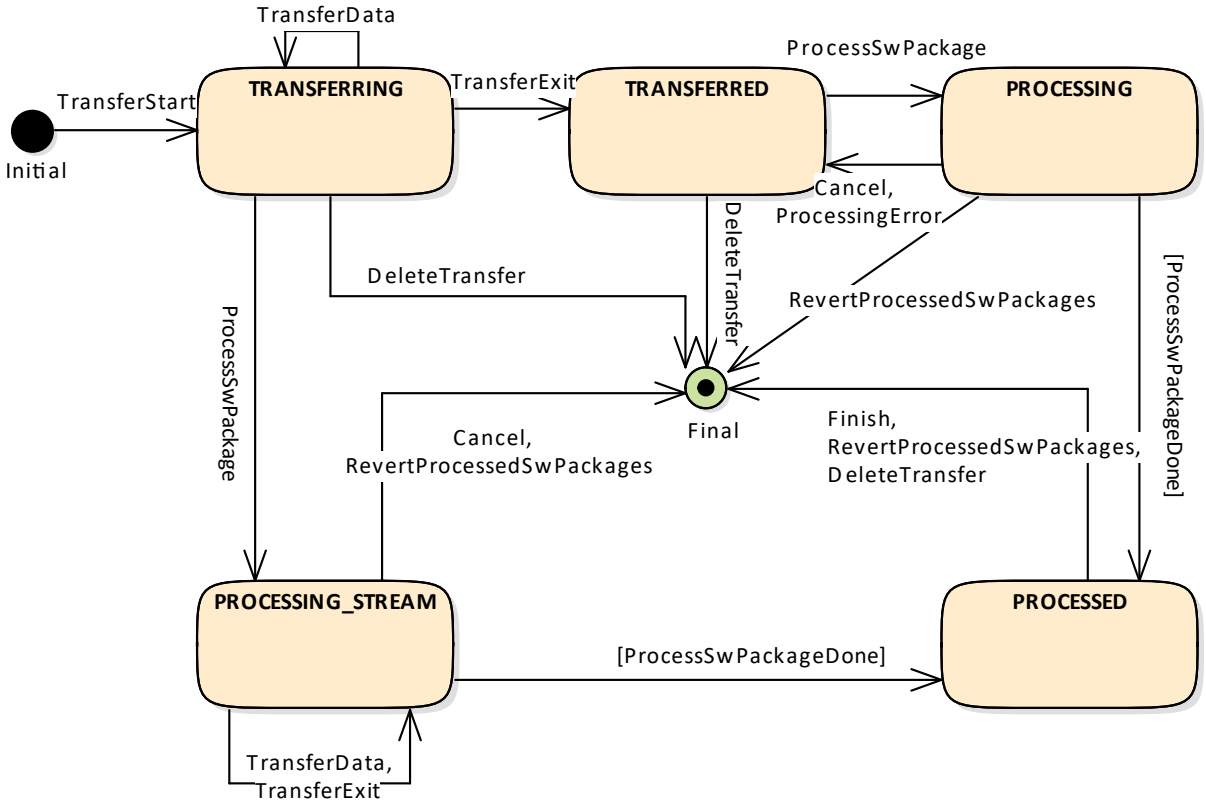

Figure 7.5: State Machine representing `Software Packages` lifecycle, without storing option

[SWS_UCM_00007] 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] 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 an `id` for the `Software Package` to be transferred.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

[SWS_UCM_00140] `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)*

While a `Software Package` is being transferred, if `UCM` receives a subsequent `TransferStart` call targeting another `Software Package`, `UCM` should make sure that the sum of the size of both `Software Package`s (the one being transferred and the one requested to be transferred) does not exceed the size of the `UCM` buffer. Otherwise, the `TransferStart` should raise the `ApplicationError InsufficientMemory` and the newly requested transmission should be rejected as described above.

[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_00243]{DRAFT} **Too big block size received by UCM** ⌈In the case the received block size with TransferData exceeds the block size returned by TransferStart for the same TransferId, UCM shall raise the ApplicationError IncorrectBlockSize.⌋*(RS_UCM_00013, RS_UCM_00014, RS_UCM_00025)*

[SWS_UCM_00203]{DRAFT} **TransferData InvalidTransferId** ⌈TransferData shall raise the error ApplicationError InvalidTransferId in case an invalid TransferId (An ID that was not initiated by TransferStart or marked invalid by DeleteTransfer or RevertProcessedSwPackages) is sent by the client.⌋*(RS_UCM_00019)*

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

[SWS_UCM_00204]{DRAFT} **TransferData IncorrectBlock** ⌈TransferData shall raise ApplicationError IncorrectBlock upon receipt of a block counter value that is successfully transmitted to UCM before or upon receipt of an unexpected block counter value.⌋*(RS_UCM_00014, RS_UCM_00019)*

[SWS_UCM_00205]{DRAFT} **TransferData IncorrectSize** ⌈In case the transferred Software package size exceeds the provided size in TransferStart, TransferData shall raise ApplicationError IncorrectSize⌋*(RS_UCM_00014, RS_UCM_00019)*

[SWS_UCM_00206]{DRAFT} **TransferData InsufficientMemory** ⌈TransferData shall raise the error ApplicationError InsufficientMemory if resources to store the Software Package ceased to exist during the transfer operation.⌋*(RS_UCM_00013, RS_UCM_00019)*

[SWS_UCM_00207]{DRAFT} **TransferData BlockInconsistent** ⌈TransferData shall raise the error ApplicationError BlockInconsistent in case Consistency check for transferred block fails.⌋*(RS_UCM_00012)*

[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_00208]{DRAFT} **TransferData OperationNotPermitted** ⌈Calling TransferData after calling TransferExit for a specific TransferId shall raise the error ApplicationError OperationNotPermitted⌋*(RS_UCM_00019)*

[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_00209]{DRAFT} **`TransferData` PackageInconsistent** ⌈TransferData shall raise the error `ApplicationError PackageInconsistent` in case the `Software Package` integrity check fails.⌋*(RS_UCM_00006, RS_UCM_00012)*

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

[SWS_UCM_00028]{DRAFT} **`Software Package` Authentication** ⌈`UCM` shall check authentication of the `Software Package` or the transferred block before processing it.⌋*(RS_UCM_00006)*

[SWS_UCM_00250]{DRAFT} **`TransferData` AuthenticationFailed** ⌈TransferData shall raise the error `ApplicationError AuthenticationFailed` in case the authentication of the `Software Package` fails.⌋*(RS_UCM_00006)*

`Software Package` contains authentication and integrity tags, which are used during the transfer sequence to authenticate the content of the `Software Package`.

[SWS_UCM_00098]{DRAFT} **`Software Package` Authentication failure** ⌈`UCM` shall raise the `ApplicationError AuthenticationFailed`, if the `Software Package` authentication check fails.⌋*(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)*

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. 7.1.5.

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

[SWS_UCM_00148] Transfer sequence order ⌈Calling `TransferExit` without calling `TransferData` at least once or after `TransferExit` is called for a specific TransferID, shall raise the `ApplicationError OperationNotPermitted`.⌋*(RS_UCM_00019)*

[SWS_UCM_00211]{DRAFT} **`TransferData` TransferInterrupted** ⌈TransferData shall raise the error `ApplicationError TransferInterrupted` if transfer has been interrupted with a higher priority protocol.⌋*(RS_UCM_00014)*

[SWS_UCM_00212]{DRAFT} **`TransferExit` InvalidTransferId** ⌈TransferExit shall raise the error `ApplicationError InvalidTransferId` in case an invalid TransferId is sent by the client.⌋*(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 `Software Packages`' identifiers, names, versions, states, consecutive bytes received and consecutive blocks received.⌋*(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 TransferID, ConsecutiveBytesReceived and ConsecutiveBlocksReceived at this particular state.

[SWS_UCM_00213]{DRAFT} **`TransferExit` InvalidPackageManifest** ⌈TransferExit shall raise the error `ApplicationError InvalidPackageManifest` upon receival of an invalid manifest.⌋*(RS_UCM_00012)*

[SWS_UCM_00214]{DRAFT} **`DeleteTransfer` InvalidTransferId** ⌈DeleteTransfer shall raise the error `ApplicationError InvalidTransferId` in case an invalid TransferId is sent by the client.⌋*(RS_UCM_00019)*

[SWS_UCM_00215]{DRAFT} **`DeleteTransfer` OperationNotPermitted** ⌈Calling DeleteTransfer during processing or during the processing stream shall raise the error `ApplicationError OperationNotPermitted`.⌋*(RS_UCM_00019)*

[SWS_UCM_00216]{DRAFT} **Validity of TransferId** ⌈The TransferId of a `Software Package` shall be invalidated for further use when it reaches final lifecycle state.⌋*(RS_UCM_00019)*

[SWS_UCM_CONSTR_00010]{DRAFT} **UCM Client update sequence** ⌈Any `UCM` Client should confirm that `UCM` is in `kIdle CurrentStatus` state before starting any update (transfer/process/activate).⌋*(RS_UCM_00035)*

7.1.4 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, RS_UCM_00025)*

[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, RS_UCM_00025)*

[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, RS_UCM_00025)*

[SWS_UCM_00168]{DRAFT} **Transferring while processing from stream** ⌈Software Package state shall remain in kProcessingStream when TransferData is called.⌋*(RS_UCM_00024, RS_UCM_00025)*

[SWS_UCM_00169]{DRAFT} **Finishing transfer while processing from stream** ⌈Software Package state shall be set to kProcessed when TransferExit is called and the Software Package is completely processed.⌋*(RS_UCM_00024, RS_UCM_00025)*

[SWS_UCM_00200]{DRAFT} **Failing authentication** ⌈UCM shall delete the Software Package if authentication is failing at TransferExit or ProcessSwPackage call.⌋*(RS_UCM_00039, RS_UCM_00006)*

7.1.5 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] 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] Processing several update Software Packages ⌈UCM shall support processing of several Software Packages, not in parallel, by calling method ProcessSwPackage several times in sequence.⌋*(RS_UCM_00001, RS_UCM_00004, RS_UCM_00005)*

[SWS_UCM_00217]{DRAFT} **ProcessSwPackage InsufficientMemory** ⌈ProcessSwPackage method shall raise the ApplicationError InsufficientMemory if the UCM buffer has not enough resources to process the corresponding Software Package.⌋*(RS_UCM_00013, RS_UCM_00025)*

[SWS_UCM_00218]{DRAFT} **ProcessSwPackage InvalidTransferId** ⌈ProcessSwPackage shall raise the error ApplicationError InvalidTransferId in case an invalid TransferId is sent by the client.⌋*(RS_UCM_00026)*

[SWS_UCM_00219]{DRAFT} **ProcessSwPackage OperationNotPermitted** ⌈ProcessSwPackage shall raise the error ApplicationError OperationNotPermitted in case the processing of the specified Software Package is already done, or in case the processed Software Package action is update or removal of a non-existing software cluster or in case streaming is not possible.⌋*(RS_UCM_00025, RS_UCM_00026)*

During package processing, the progress is provided.

[SWS_UCM_00018] 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_00220]{DRAFT} **GetSwProcessProgress InvalidTransferId** ⌈GetSwProcessProgress shall raise the error ApplicationError InvalidTransferId in case an invalid TransferId is sent by the client.⌋*(RS_UCM_00023)*

[SWS_UCM_00029] 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 InvalidPackageManifest if it finds discrepancies there.⌋*(RS_UCM_00012)*

[SWS_UCM_00104] 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_00230]{DRAFT} **ProcessSwPackage AuthenticationFailed** ⌈ProcessSwPackage shall raise the error ApplicationError AuthenticationFailed in case the authentication of the Software Package fails.⌋*(RS_UCM_00006)*

When AuthenticationFailed error is raised it is up to client to decide if a deleteTransfer will be called or not. The behavior may vary depending on the life cycle meaning R&D phase or on the field phase.

[SWS_UCM_00231]{DRAFT} **ProcessSwPackage IncompatibleDelta** ⌈ProcessSwPackage shall raise the error ApplicationError IncompatibleDelta if delta package dependency fails at processing.⌋*(RS_UCM_00007)*

[SWS_UCM_00232]{DRAFT} **ProcessSwPackage** ⌈If ApplicationError IncompatibleDelta is raised, UCM shall terminate the processing and shall delete the software package blocks that has been transferred.⌋*(RS_UCM_00007)*

[SWS_UCM_00003] 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_00233]{DRAFT} **Cancel Operation CancelFailed** ⌈Cancel shall raise the error ApplicationError CancelFailed in case cancelling of processing of a Software Package fails.⌋*(RS_UCM_00020)*

[SWS_UCM_00234]{DRAFT} **Cancel OperationNotPermitted** ⌈Cancel shall raise the error ApplicationError OperationNotPermitted in case the targeted Software Package processing has not yet started or has been already finished.⌋ *(RS_UCM_00020)*

[SWS_UCM_00235]{DRAFT} **Cancel InvalidTransferId** ⌈Cancel shall raise the error ApplicationError InvalidTransferId in case an invalid TransferId is sent by the client.⌋*(RS_UCM_00020)*

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

The main difference between a RevertProcessedSwPackages and a Rollback is that the former can only be performed before the successful activation of the targeted Software Package(s) while the latter can only be performed after such activation.

[SWS_UCM_00236]{DRAFT} **RevertProcessedSwPackages NotAbleToRevertPackages** ⌈RevertProcessedSwPackages shall raise the error ApplicationError NotAbleToRevertPackages in case reverting of processed Software Packages have failed.⌋*(RS_UCM_00020)*

[SWS_UCM_00237]{DRAFT} **RevertProcessedSwPackages OperationNotPermitted** ⌈RevertProcessedSwPackages method call shall raise the error ApplicationError OperationNotPermitted in case the processed Software Packages are successfully activated or it is called at other states than kReady (Software Package(s) are finished being processed) or kProcessing states.⌋*(RS_UCM_00020)*

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] 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.

7.1.6 Activation and Rollback

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] Activated state ⌈UCM state kActivated shall express that new version of updated SoftwareCluster is verified.⌋*(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] Dependency Check ⌈At activation (i.e. after Activate method is called), UCM shall perform a dependency check to ensure that all the Software Packages having dependencies toward each other have been processed successfully, otherwise return ApplicationError MissingDependencies.⌋*(RS_UCM_00007)*

[SWS_UCM_00027] Delta Package activation ⌈Applicable version of SoftwareCluster on which to apply delta shall be included into related SoftwarePackage's deltaPackageApplicableVersion attribute.⌋*(RS_UCM_00007)*

[SWS_UCM_00201]{DRAFT} **Delta Package dependency error** ⌈The Activate method of the service interface shall raise the error IncompatibleDelta if version present in SoftwarePackage's deltaPackageApplicableVersion attribute does not correspond to the version already present in the AUTOSAR Adaptive Platform.⌋*(RS_UCM_00007)*

[SWS_UCM_00025] Activation of SoftwareClusters ⌈UCM shall offer method Activate to enable execution of any pending changes from the previously processed Software Packages.⌋*(RS_UCM_00021)*

After Activate, the new set of SoftwareClusters can be started. Activation covers all the processed Software Packages for all the clients.

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

The activation method could lead to a full system reset. When Software Package updates underlying OS, AUTOSAR Adaptive Platform or any Adaptive Application which is configured to be part of Function Group MachineFG, the execution of updated software occurs through system reset by calling State Management UpdateRequest Service Interface ResetMachine method. Meta-data of Software Package defines the activation method.

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).

In principle, it is possible to activate multiple versions of the same SoftwareCluster in one activation step. This could be useful for example with delta package updates but does not apply to firmware updates. The specification does not prohibit to create this kind of chained updates. The decision to use chained updates should be based on safety aspects and the applicability of the underlying update technology, if the update is for a classic or an adaptive platform, if a file system is involved or if the used platform even support it.

[SWS_UCM_00241]{DRAFT} **Activate OperationNotPermitted** ⌈Activate shall raise the error ApplicationError OperationNotPermitted in case the UCM state is not kReady.⌋*(RS_UCM_00021)*

[SWS_UCM_00242]{DRAFT} **Activate PreActivationFailed** ⌈Activate shall raise the error ApplicationError PreActivationFailed in case of activation state transition failure from State Management side.⌋*(RS_SM_00001)*

7.1.6.2 Rollback

[SWS_UCM_00005] 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] Rolling-back the software update ⌈At kRolling-Back state, UCM shall disable the changes done by the software update by calling State Management UpdateRequest Service Interface PrepareRollback method for each Function Group of the processed Software Cluster in the update session. Then UCM shall call State Management UpdateRequest Service Interface ResetMachine method if any Software Cluster requires a machine reboot to be rolled back.⌋*(RS_UCM_00008)*

[SWS_UCM_00238]{DRAFT} **Rollback NotAbleToRollback** ⌈Rollback shall raise the error ApplicationError NotAbleToRollback in case failure has occurred during Rollback.⌋*(RS_UCM_00020)*

[SWS_UCM_00239]{DRAFT} **`Rollback` OperationNotPermitted** ⌈Rollback shall raise the error `ApplicationError OperationNotPermitted` in case `UCM` current state is not `kActivated` nor `kVerifying`.⌋*(RS_UCM_00020)*

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] 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 `kRollingBack` state, `UCM` modifies the `Boot options` so that in the next restart of the updated software the original versions will be executed.

7.1.6.4 Finishing activation

[SWS_UCM_00259]{DRAFT} **Ending the update session** ⌈`UCM` shall call `State Management UpdateRequest` Service Interface `StopUpdateSession` method when `UCM` is entering the `kCleaning-up` state.⌋*(RS_UCM_00021, RS_UCM_00018)*

[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.⌋*(RS_UCM_00015)*

`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`.

[SWS_UCM_00240]{DRAFT} **`Finish` OperationNotPermitted** ⌈Finish shall raise the error `ApplicationError OperationNotPermitted` in case there are no activated nor rolled-back `Software Packages` pending finalization (i.e `UCM` state is not `kActivated` nor `kRolledBack`.⌋*(RS_UCM_00001, RS_UCM_00026)*

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 `SoftwareCluster`s aren't executed anymore.

7.1.7 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 7.1.5 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. 7.6.

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

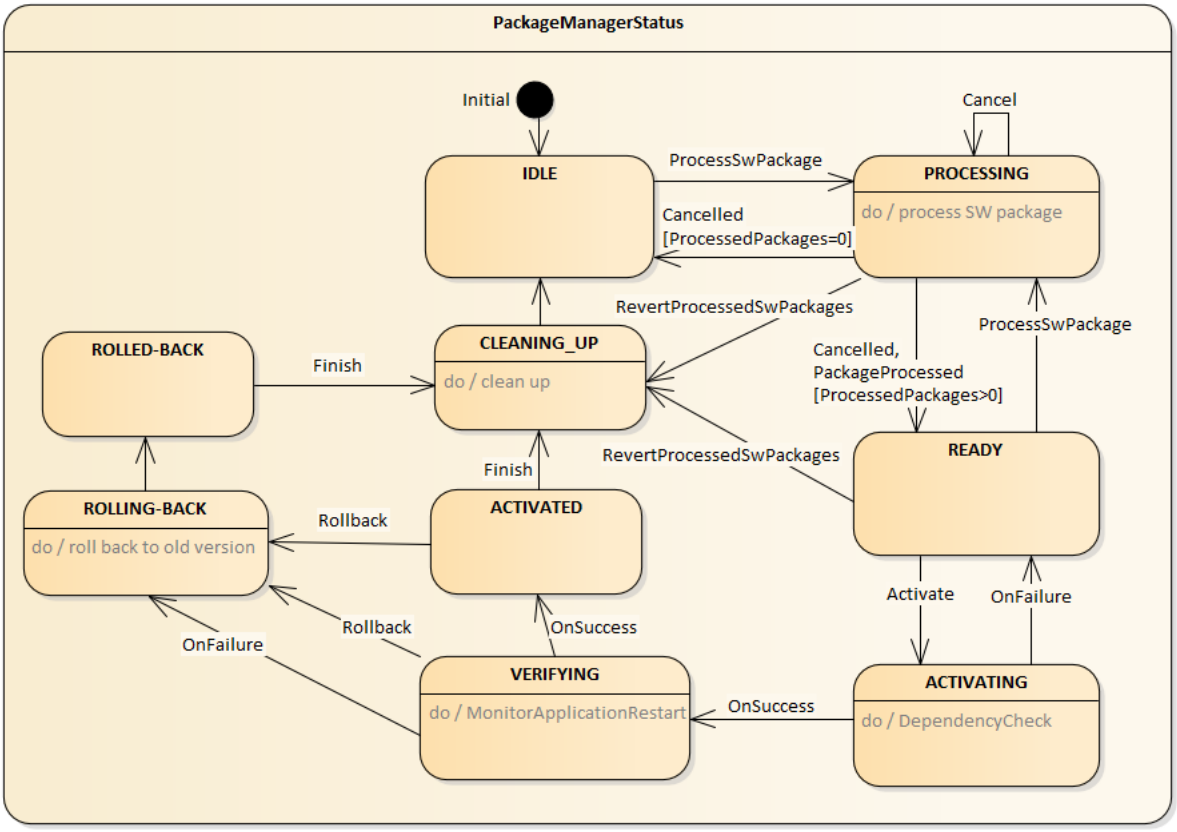

Figure 7.6: State Machine for the package processing using service interface: PackageManagement

UCM supported method calls for each value of field CurrentStatus are shown in Fig. 7.6.

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

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

[SWS_UCM_00150] 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] Return to the Idle state from Processing state ⌈kIdle state shall be set when ProcessSwPackage returns with error code ProcessSwPackageCancelled and if no other Software Packages were previously processed during this processing operation.⌋*(RS_UCM_00024)*

[SWS_UCM_00151] Entering the Ready state of Package Management after a Cancel call ⌈If ProcessSwPackage has been cancelled, it shall return error code ProcessSwPackageCancelled 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] 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] 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] 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] 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 kActivating state of Package Management** ⌈kActivating shall be set when Activate is called. This triggers the dependency check and returns ApplicationError MissingDependencies if this check fails.⌋*(RS_UCM_00024)*

[SWS_UCM_00153]{DRAFT} **Action in kActivating state of Package Management** ⌈When kActivating is set and after the State Management UpdateRequest Service Interface StartUpdateSession method call by UCM, the UCM shall call the State Management UpdateRequest Service Interface PrepareUpdate method for each Function Groups to eventually stop them.⌋*(RS_UCM_00024)*

[SWS_UCM_00260]{DRAFT} **PrepareUpdate, VerifyUpdate and PrepareRollback orders** ⌈UCM shall compute the order of the State Management UpdateRequest Service Interface PrepareUpdate, VerifyUpdate and PrepareRollback method

calls from the dependency model included in the Software Cluster manifests.⌋ *(RS_UCM_00007, RS_UCM_00021, RS_UCM_00030)*

[SWS_UCM_00261]{DRAFT} **PrepareUpdate, VerifyUpdate and PrepareRollback synchronous calls** ⌈Calls to State Management UpdateRequest Service Interface PrepareUpdate, VerifyUpdate and PrepareRollback methods shall not be concurrent.⌋ *(RS_UCM_00026)*

[SWS_UCM_00262]{DRAFT} **Update preparation rejected** ⌈If any one of the State Management UpdateRequest Service Interface PrepareUpdate method call returns error kRejected too many times or for too long (implementation specific thresholds), UCM shall transition from kActivating to kReady states.⌋ *(RS_UCM_00026)*

[SWS_UCM_00263]{DRAFT} **Update preparation failure** ⌈If any one of the State Management UpdateRequest Service Interface PrepareUpdate method returns error kPrepareFailed, UCM shall transition from kActivating to kReady states.⌋ *(RS_UCM_00026)*

[SWS_UCM_00154]{DRAFT} **Entering the Verifying state of Package Management** ⌈kVerifying shall be set when the dependency check have been performed successfully (all dependencies are satisfied) and that the preparation of the Software Clusters by the State Management has been successfully performed.⌋ *(RS_UCM_00024)*

The machine could most likely 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 [12] is running and user can be protected from any issues caused by the update after the update has finished.

An update could most likely require to reparse the manifests if a machine reset is not needed. It is up to implementer to define if the most suitable timing is after performing the atomic activation of the Software Clusters (switching A/B partition, changing symlinks, etc.) or being triggered by the State Management after the first call of State Management UpdateRequest Service Interface VerifyUpdate method.

[SWS_UCM_00085]{DRAFT} **Entering the kActivated 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 and verified indicated by successful return of State Management UpdateRequest Service Interface VerifyUpdate method calls.⌋ *(RS_UCM_00024)*

kVerifying state gives the client controlling the update process a chance to perform verification test by calling State Management UpdateRequest Service Interface [SWS_SM_91017] VerifyUpdate method, 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. This is started by calling State Management UpdateRequest Service Interface [SWS_SM_91017] PrepareRollback method for each Software Cluster. On success, UCM goes to kRolled-Back 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 kRolling-Back state after a Rollback call** ⌈The state kRolling-Back shall be set when Rollback is called.⌋*(RS_UCM_00008, RS_UCM_00030)*

[SWS_UCM_00155]{DRAFT} **Entering the kRolling-Back state after a failure in the kVerifying state** ⌈The state kRolling-Back shall be set if any of the State Management UpdateRequest Service Interface VerifyUpdate method calls returns the result kVerifyFailed, indicating an internal error in UCM.⌋*(RS_UCM_00008, RS_UCM_00030)*

[SWS_UCM_00264]{DRAFT} **Update verification rejected** ⌈If any one of the State Management UpdateRequest Service Interface VerifyUpdate returns error kRejected too many times or for too long (implementation specific thresholds), UCM shall transition to kRolling-Back state.⌋*(RS_UCM_00030, RS_UCM_00008)*

[SWS_UCM_00111]{DRAFT} **Entering the kRolled-Back state** ⌈The state kRolled-Back shall be set after all calls to State Management UpdateRequest Service Interface PrepareRollback have returned successfully.⌋*(RS_UCM_00008, RS_UCM_00030)*

[SWS_UCM_00146] 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] Entering the Cleaning-up state after a RevertProcessedSwPackages call ⌈The state kCleaning-up shall be set when RevertProcessedSwPackages is called in kProcessing or kReady states and the UCM starts to perform cleanup actions.⌋*(RS_UCM_00008, RS_UCM_00030)*

[SWS_UCM_00163] 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] 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] 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)*

[SWS_UCM_00147] Return to the Idle state from Cleaning-up state ⌈`kIdle` state shall be set when the Clean-up operation has been completed successfully.⌋*(RS_UCM_00024)*

7.1.8 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] Detection of reset ⌈At start up UCM shall identify if uncontrolled reset occurred.⌋*(RS_UCM_00027)*

[SWS_UCM_00158] Cleanup of interrupted actions ⌈After an uncontrolled reset, UCM shall check non volatile memory integrity, recover processed artifacts in case it is corrupted and resume interrupted actions in order to return the system into a state from where UCM can continue serving its Clients.⌋*(RS_UCM_00027)*

7.1.8.1 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.

7.1.9 History

[SWS_UCM_00115]{DRAFT} **History** ⌈UCM shall provide a method `GetHistory` to retrieve all actions that have been performed by UCM when exiting `kVerifying` state from a specific time window input parameter.⌋*(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)*

7.1.10 Version Reporting

[SWS_UCM_00004] 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] 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)*

[SWS_UCM_00185]{DRAFT} **Provide Software Cluster general information** ⌈UCM shall provide a method GetSwClusterDescription to return the version, type approval, license and release notes of the SoftwareCluster that are in state kPresent.⌋*(RS_UCM_00002, RS_UCM_00011)*

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 7.1.3

[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_CONSTR_00002]{DRAFT} **UCM confidential information handling** ⌈The methods GetSwClusterInfo, GetSwClusterChangeInfo, GetHistory, GetSwClusterDescription and GetSwPackages shall only be mapped via ara::com to a secure endpoint using secure communication channel providing confidentiality protection.⌋*(RS_UCM_00002, RS_UCM_00010, RS_UCM_00011)*

[SWS_UCM_00202]{DRAFT} **Trusted Platform compliance** ⌈UCM shall ensure that after provisioning updates, all the necessary changes to maintain the Trusted Platform are carried out.⌋*(RS_EM_00014)*

The authentication tag of the Trusted Platform corresponding to the updated/removed/added executable files should also be updated/removed/added. See also Chapter 7.10 of the Execution Management [2] for details on the Trusted Platform.

7.1.12 Functional cluster lifecycle

7.1.12.1 Shutdown behaviour

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

7.2 UCM Master

7.2.1 UCM Master Functional Cluster lifecycle

[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)*

7.2.2 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 targets (UCM subordinate or Diagnostic Application) and orchestrates the processing, activations and eventual rollbacks. All these actions are what is called a campaign which UCM Master is coordinating. The UCM of the machines in the same network of a UCM Master, candidates target of a campaign, are referred to as UCM subordinates.

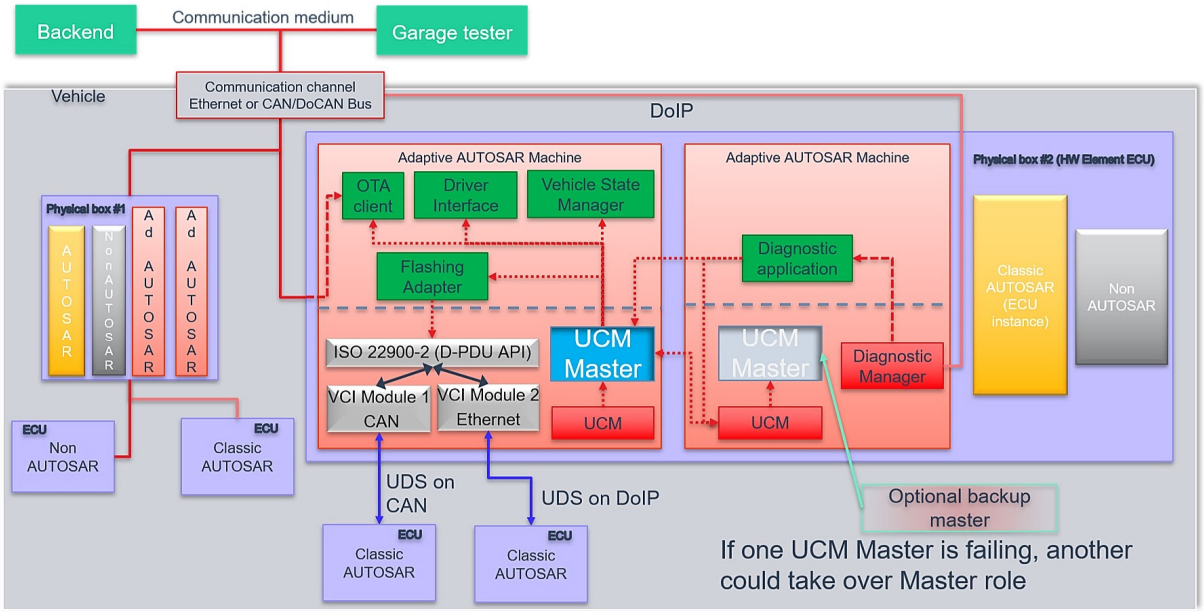

Figure 7.7: Example of UCM Master architecture overview within a vehicle

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.

The OTA Client establishes a communication between Backend and UCM Master so that they can exchange information of the installed Software Clusters in the

vehicle and the Software Clusters available in the Backend. This communication could be triggered by OTA Client with a scheduler and UCM Master to request the updates in case of newly available Software Clusters (pull case) or by Backend to push, for instance, an important security update to a fleet of vehicles (push case). 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.

7.2.3 UCM Master general behaviour

The UCM Master acts as a client of the service interface offered by the UCM subordinates, already specified in UCM. However, the UCM Master also offers three different service interfaces to OTA Client, Vehicle Driver interface and Vehicle State Manager respectively. UCM Master aggregates UCM subordinates states and can report its status field to a Backend through 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_01003]{DRAFT} **UCM Master checks states of UCM subordinates** ⌈An UCM Master shall check the 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)*

7.2.4 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.

7.2.5 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} **Start transfer of a Vehicle Package 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_01011]{DRAFT} **TransferVehiclePackage InsufficientMemory** ⌈TransferVehiclePackage method shall raise the ApplicationError InsufficientMemory if the UCM buffer has not enough resources to process the corresponding Vehicle Package.⌋*(RS_UCM_00013)*

[SWS_UCM_01012]{DRAFT} **`TransferVehiclePackage` InsufficientComputationPower** ⌈`TransferVehiclePackage` method shall raise the error `ApplicationError InsufficientComputationPower` if there is no enough computational resources to initiate the transfer.⌋*(RS_UCM_00013)*

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

[SWS_UCM_01014]{DRAFT} **Packages transferring sequence** ⌈`TransferStart` method shall raise the `ApplicationError UnexpectedPackage` if the `Software Package` name parameter was not a value of the `RequestedPackage` field.⌋*(RS_UCM_00043)*

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

[SWS_UCM_01013]{DRAFT} **Too big block size received by `UCM Master`** ⌈In the case the received block size with `TransferData` exceeds the block size returned by `TransferStart` or `TransferVehiclePackage` for the same TransferId, `UCM Master` shall raise the `ApplicationError IncorrectBlockSize`.⌋*(RS_UCM_00035)*

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

[SWS_UCM_01015]{DRAFT} **Invalid Vehicle Package manifest** ⌈`TransferExit` shall raise the error `InvalidPackageManifest` when a Vehicle Package manifest is not compliant with the Autosar schema.⌋*(RS_UCM_00036, RS_UCM_00043)*

[SWS_UCM_01016]{DRAFT} **Invalid Package Manifest** ⌈`UCM Master` shall raise the error `InvalidPackageManifest` in case a manifest file is not compliant with the AUTOSAR schema.⌋*(RS_UCM_00036, RS_UCM_00043)*

[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_00036)*

[SWS_UCM_01017]{DRAFT} **RequestedPackage field** ⌈`UCM Master` shall provide the field `RequestedPackage` containing the requested `Software Package` name and version as defined in update campaign. Changing this field is a notification for the `OTA Client` to start transfer of the requested `Software Package`.⌋*(RS_UCM_00042)*

`OTA Client` does not know what `Software Packages` should be transferred in a given campaign contained in a `Vehicle Package`. `OTA Client` can know what `Software Package` is expected to be transferred by subscribing to `UCM Master`'s `RequestedPackage` field. Version is added to support campaigns which need an update path for a `Software Package` requiring an intermediate update to a transitional

version. In this case the version parameter makes it unambiguous which package version shall be transferred as both have the same name assigned.

7.2.6 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.

7.2.6.1 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 about Software Clusters present in Backend for the vehicle.⌋*(RS_UCM_00033)*

[SWS_UCM_01103]{DRAFT} **Inform Backend of needed Software Clusters for an update** ⌈On SwPackageInventory call, UCM Master shall compare the supplied list of available Software Clusters in the Backend for the vehicle to its own internal information of present Software Clusters in the vehicle and return the list of Software Clusters selected for update.⌋*(RS_UCM_00033)*

The OTA Client uses this returned Software Clusters list to request the selected packages to the Backend.

[SWS_UCM_01119]{DRAFT} **Report information of Software Packages** ⌈UCM Master shall provide a method GetSwPackages to return the identifiers, names, versions, Consecutive Bytes Received, Consecutive Blocks Received and states of Software Packages.⌋*(RS_UCM_00035)*

7.2.6.2 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** ⌈`UCM Master` shall provide a method `DriverApproval` in order to receive the confirmation of the vehicle driver's approval.⌋*(RS_UCM_00038)*

The Vehicle Driver Interface `Adaptive Application` could adapt its notification content related to safety by subscribing to the UCM Master's `SafetyPolicy` field.

[SWS_UCM_01117]{DRAFT} **UCM Master `SafetyState` field** ⌈`UCM Master` shall provide to vehicle driver interface the `SafetyState` field.⌋*(RS_UCM_00038, RS_UCM_00037)*

`UCM Master` can notify vehicle driver with `SafetyState` field if the vehicle safety is breached during the update, by for instance popping-up a message.

[SWS_UCM_01118]{DRAFT} **UCM Master waiting for vehicle driver approval** ⌈In the case approval from driver is requested as configured in `VehiclePackage`, `UCM Master` shall wait for `DriverApproval` method with parameter `Approval`=True before transitioning state from `kVehiclePackageTransferring` to `kSoftwarePackage_Transferring`, `kSoftwarePackage_Transferring` to `kProcessing` or `kProcessing` to `kActivating`.⌋*(RS_UCM_00038)*

[SWS_UCM_CONSTR_00003]{DRAFT} **Exclusive use of Vehicle Driver Interface** ⌈Software Integrator shall ensure that only one `Adaptive Application` is using the `UCM Master`'s Vehicle Driver Interface.⌋*(RS_UCM_00035, RS_UCM_00037)*

For example, the integrator may restrict the access of Vehicle Driver Interface from `UCM Master` by 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_CONSTR_00004]{DRAFT} **Unsupported safety policy by Vehicle driver interface** ⌈In the case `SafetyPolicy` field is not a supported safety policy Vehicle driver interface, it shall call the method `DriverApproval` with parameter `SafeToUpdate`=False and `SafetyPolicy`='notSupportedSafetyPolicy'.⌋*(RS_UCM_00037)*

[SWS_UCM_01120]{DRAFT} **Provide `Software Packages` general information** ⌈`UCM Master` shall provide a method `GetSwPackageDescription` to return the description of each `Software Packages` that are part of current campaign and that are contained in `Vehicle Package`.⌋*(RS_UCM_00033, RS_UCM_00038)*

7.2.6.3 Vehicle State Manager

Vehicle State Manager is collecting states from the several vehicle `ECU`s and informs `UCM Master` when the safety state computed based on the safety policy referred in the `Vehicle Package` is changing. 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} **`UCM Master` provides a safety policy interface** ⌈`UCM Master` shall provide a field `SafetyPolicy` for which values are available in `VehiclePackage`.⌋*(RS_UCM_00037)*

[SWS_UCM_01110]{DRAFT} **`UCM Master` SafetyState method** ⌈`UCM Master` shall provide a method `SafetyState` to get informed of vehicle state changes.⌋*(RS_UCM_00037)*

[SWS_UCM_CONSTR_00005]{DRAFT} **Safety state change** ⌈Vehicle State Manager Adaptive Application shall call `SafetyState` method provided by `UCM Master` when the safety state is changing.⌋*(RS_UCM_00035, RS_UCM_00037)*

[SWS_UCM_CONSTR_00009]{DRAFT} **Safety policy change** ⌈Vehicle State Manager Adaptive Application shall call `SafetyState` method provided by `UCM Master` when the field `SafetyPolicy` is changing.⌋*(RS_UCM_00035, RS_UCM_00037)*

[SWS_UCM_CONSTR_00006]{DRAFT} **Exclusive use of Vehicle State Manager** ⌈System Integrator shall ensure that Vehicle State Manager is the exclusive user of the `SafetyState` method.⌋*(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_CONSTR_00007]{DRAFT} **Unsupported safety policy by Vehicle State Manager** ⌈In the case the requested `SafetyPolicy` field is not supported by Vehicle State Manager, it shall call `SafetyState` method with parameter `SafeToUpdate`=False and `SafetyPolicy`='notSupportedSafetyPolicy'.⌋*(RS_UCM_00037)*

[SWS_UCM_CONSTR_00008]{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.

7.2.6.4 Flashing Adapter

Flashing Adapter is an application that is used in the case UCM Master is updating a AUTOSAR Classic Platform or any platform that can be flashed using diagnostic. It contains OEM specific diagnostic sequences and communicates via ara::com with the UCM Master and the AUTOSAR Adaptive Platform, and uses an implementation of diagnostic protocol data unit application programming interface (D-PDU API) to communicate with Classic ECUs over the Vehicle Bus.

[SWS_UCM_CONSTR_00011]{DRAFT} **Flashing Adapter provided interface** ⌈Flashing Adapter shall provide the same ara::com service interface as UCM ([SWS_UCM_00131]).⌋ *(RS_UCM_00035)*

7.2.7 Non Adaptive Platform update

[SWS_UCM_01121]{DRAFT} **Adaptive Platform interface provided for Flashing Adapter** ⌈The interface provided by the AUTOSAR Adaptive Platform in order to update non AUTOSAR Platform should comply with ISO 22900-2:2017 (D-PDU API) but as this standard's coverage is wide, it is allowed to implement a reduced API that is needed to update for instance a AUTOSAR Classic Platform.⌋*(RS_UCM_00035)*

The implementation of the D-PDU API is processing binary data from the Flashing Adapter and do all of the required session, transport and network layer handling to send and receive the data on the physical vehicle bus with respect to the underlying protocols. The reason of using ISO 22900-2:2017 is to ensure that the specific Flashing Adapter from any vehicle or tool manufacturer can operate on a common software interface and can easily exchange MVCI (Modular Vehicle Communication Interface) protocol module implementations.

In the case the targeted ECU by an update does not have the capability to switch between current and new Software Cluster, the vehicle package campaign should foresee to download not only the new version but also the currently installed version of the Software Cluster to be updated in order to make possible a rollback from the new version to the old version of the Software Cluster. The location to store the current Software Package could be the Flashing Adapter but ultimately it has to be available to Flashing Adapter in order to flash it in case of a rollback.

7.2.7.1 D-PDU API implementation support

[SWS_UCM_01122]{DRAFT} **Supported physical layers by D-PDU API implementation** ⌈ISO_11898_2_DWCAN (Dual Wire CAN), ISO_11898_3_DWFTCAN (Dual Wire CAN Fault tolerant), SAE_J2411_SWCAN (Single Wire CAN) and IEEE_802_3(Ethernet) physical layers shall be supported if their respective physical vehicle bus is available inside the ECU, all other physical layers present in D-PDU API are optional.⌋*(RS_UCM_00035)*

[SWS_UCM_01123]{DRAFT} **Supported application layers by D-PDU API implementation** ⌈ISO_15765_3 (Unified diagnostic services, UDS on CAN, ISO withdrawn UDS), ISO_14229_3 (Unified diagnostic services on CAN implementation, UDSonCAN) and ISO_14229_5 (Unified diagnostic services on Internet Protocol implementation, UDSonIP) application layers shall be supported if their respective application layer is available inside the ECU, all other application layers present in D-PDU API are optional.⌋*(RS_UCM_00035)*

[SWS_UCM_01124]{DRAFT} **Supported protocols by D-PDU API implementation** ⌈ISO UDS on CAN with Application layer ISO_15765_3, ISO UDS on CAN with Application layer ISO_14229_3 (UDSonCAN) and ISO UDS on DoIP with Application layer ISO_14229_5 (UDSonIP) protocols shall be supported, all other protocols are optional.⌋*(RS_UCM_00035)*

These protocols are present in 'Table B.2 - Standard protocol combination list' of ISO 22900-2:2017(E).

7.2.7.2 Not required D-PDU API concepts

Dynamic Link Libraries for Windows operating system are not required. The Windows installation process out of ISO 22900-2:2017(E) chapter 8.7.2 is not applicable to the AUTOSAR Adaptive Platform which is using POSIX Operating System.

[SWS_UCM_01125]{DRAFT} **Separation of D-PDU API-Software with the MVCI protocol module firmware** ⌈A D-PDU API implementation may be split at OSI-Layer 4 into a D-PDU API implementation on OSI-Layer 5 (usually in the PC itself) and the VCI-Module on OSI-Layers 3 and 4 (usually the VCI itself).⌋*(RS_UCM_00035)*

[SWS_UCM_01126]{DRAFT} **Root description file (RDF)** ⌈Within an AUTOSAR Adaptive Platform, only one D-PDU API implementation is required for UCM, therefore the D-PDU API implementation may not use the D-PDU API root description file (RDF).⌋*(RS_UCM_00035)*

The only instance of the D-PDU API within a Software Cluster can be statically linked with the Flashing Adapter.

[SWS_UCM_01127]{DRAFT} **Module Description File (MDF)** ⌈The D-PDU API implementation should not implement a protocol description file.⌋*(RS_UCM_00035)*

The supported protocol module types are fixed in the UCM use case.

[SWS_UCM_01128]{DRAFT} **Symbolic names and IDs** ⌈The Flashing Adapter may operate the D-PDU API without using symbolic names and IDs during runtime. If the use case excludes frequent changes to the MDFs, simple Flashing Adapter may even hardcode (e.g. in a header file) all necessary IDs and operate the D-PDU API without symbolic names.⌋*(RS_UCM_00035)*

[SWS_UCM_01129]{DRAFT} **SAE J2534-1 and RP 1210a compatibility** ⌈D-PDU API implementation may not be compatible to SAE J2534-1 and RP 1210a.⌋*(RS_UCM_00035)*

The Adaptive Platform does not need any migration path.

[SWS_UCM_01130]{DRAFT} **ComPrimitives in RawMode** ⌈D-PDU API implementation may not implement the IOCTL filter data structure.⌋*(RS_UCM_00035)*

7.2.7.3 Not required D-PDU API functions

PDULockResource() and PDUUnlockResource() are used to lock and unlock exclusive access to a ComLogicalLink in case of parallel usage of the D-PDU API implementation by multiple applications on the same physical communication link. Flashing of

a Classic ECU always requires some exclusive access and should be handled in the AUTOSAR Adaptive Platform itself.

[SWS_UCM_01131]{DRAFT} **PDUIoCtl(PDU_IOCTL_RESET)** ⌈The parameter PDU_IOCTL_RESET may not be implemented in D-PDU API implementation so the call of PDUIoCtl(PDU_IOCTL_RESET) shall return the error code PDU_ERR_ID_NOT_SUPPORTED.⌋*(RS_UCM_00035)*

[SWS_UCM_01132]{DRAFT} **PDUIoCtl(PDU_IOCTL_START_MSG_FILTER), PDUIoCtl(PDU_IOCTL_CLEAR_MSG_FILTER), PDUIoCtl(PDU_IOCTL_STOP_MSG_FILTER)** ⌈The call of PDUIoCtl() with the parameters PDU_IOCTL_START_MSG, PDU_IOCTL_CLEAR_MSG_FILTER and PDU_IOCTL_CLEAR_MSG_FILTER shall return the error code PDU_ERR_ID_NOT_SUPPORTED.⌋*(RS_UCM_00035)*

The parameters PDU_IOCTL_START_MSG, PDU_IOCTL_CLEAR_MSG_FILTER and PDU_IOCTL_CLEAR_MSG_FILTER are intended for the PassThru-Mode for com-primitives and therefore an implementation is not required for the Flashing Adapter.

[SWS_UCM_01133]{DRAFT} **PDUIoCtl(PDU_IOCTL_SEND_BREAK)** ⌈The IOCTL command PDU_IOCTL_SEND_BREAK shall return PDU_ERR_ID_NOT_SUPPORTED.⌋*(RS_UCM_00035)*

The IOCTL command PDU_IOCTL_SEND_BREAK is used to send a break signal on the ComLogicalLink. A break signal can only be sent on certain physical layers (e.g. SAE J1850 VPW physical links and UART physical links) which are not supported by UCM.

[SWS_UCM_01134]{DRAFT} **Not used D-PDU API function return codes** ⌈The return codes PDU_ERR_CABLE_UNKNOWN, PDU_ERR_RSC_LOCKED, PDU_ERR_RSC_NOT_LOCKED, PDU_ERR_API_SW_OUT_OF_DATE and PDU_ERR_MODULE_FW_OUT_OF_DATE may not be implemented into the D-PDU API of the AUTOSAR Adaptive Platform.⌋*(RS_UCM_00035)*

There is no cable attached to the ECU and therefore no cable detection return code PDU_ERR_CABLE_UNKNOWN could occur.

Locking is not required for the Flashing Adapter, therefore PDU_ERR_RSC_LOCKED and PDU_ERR_RSC_NOT_LOCKED return code could not occur.

There is no separation of D-PDU API-Software with the MVCI protocol module firmware required in the AUTOSAR Adaptive Platform, so PDU_ERR_API_SW_OUT_OF_DATE and PDU_ERR_MODULE_FW_OUT_OF_DATE return codes could not occur.

7.2.8 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.

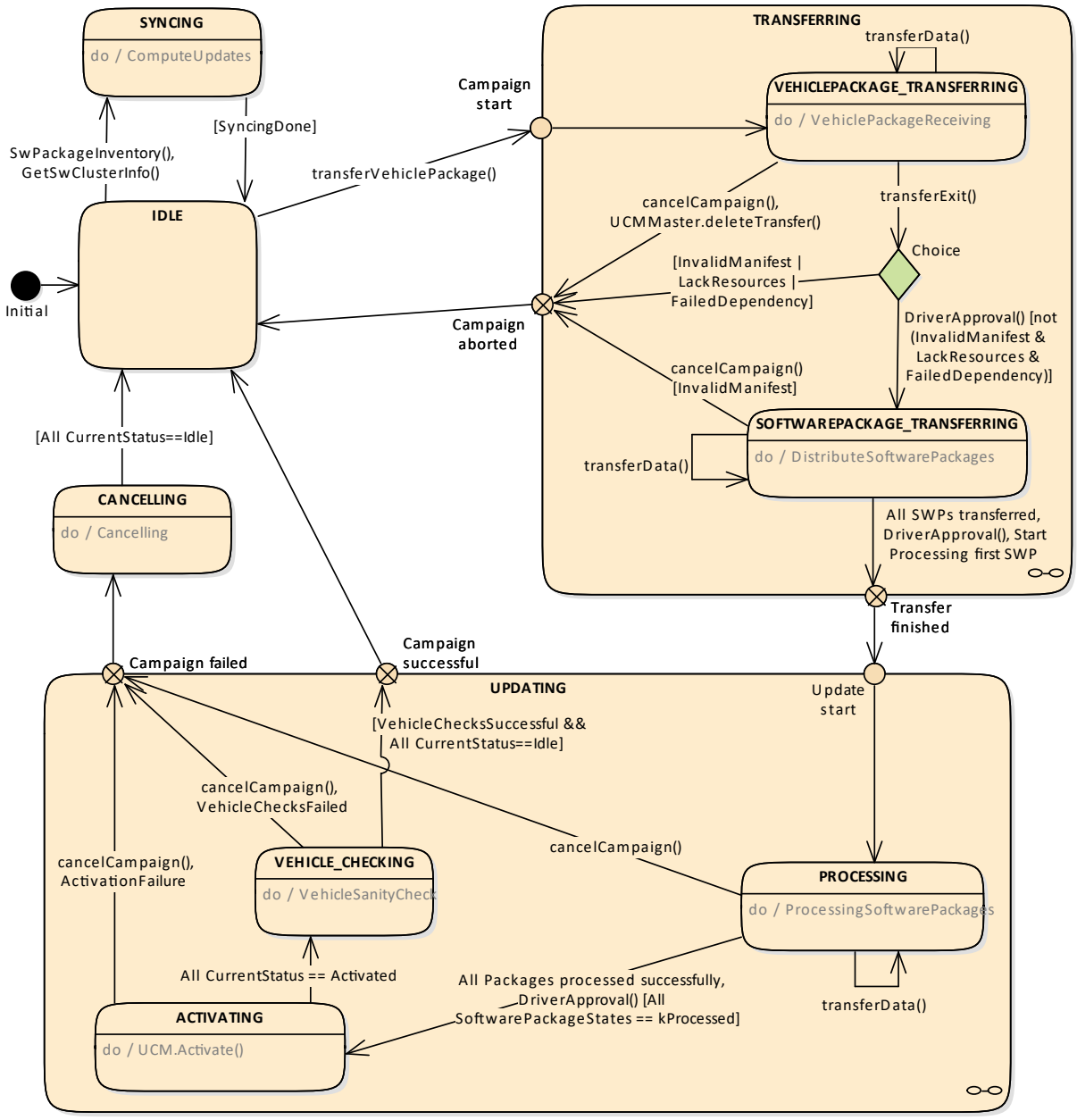

Figure 7.8: Campaign State Machine (`CampaignState` field)

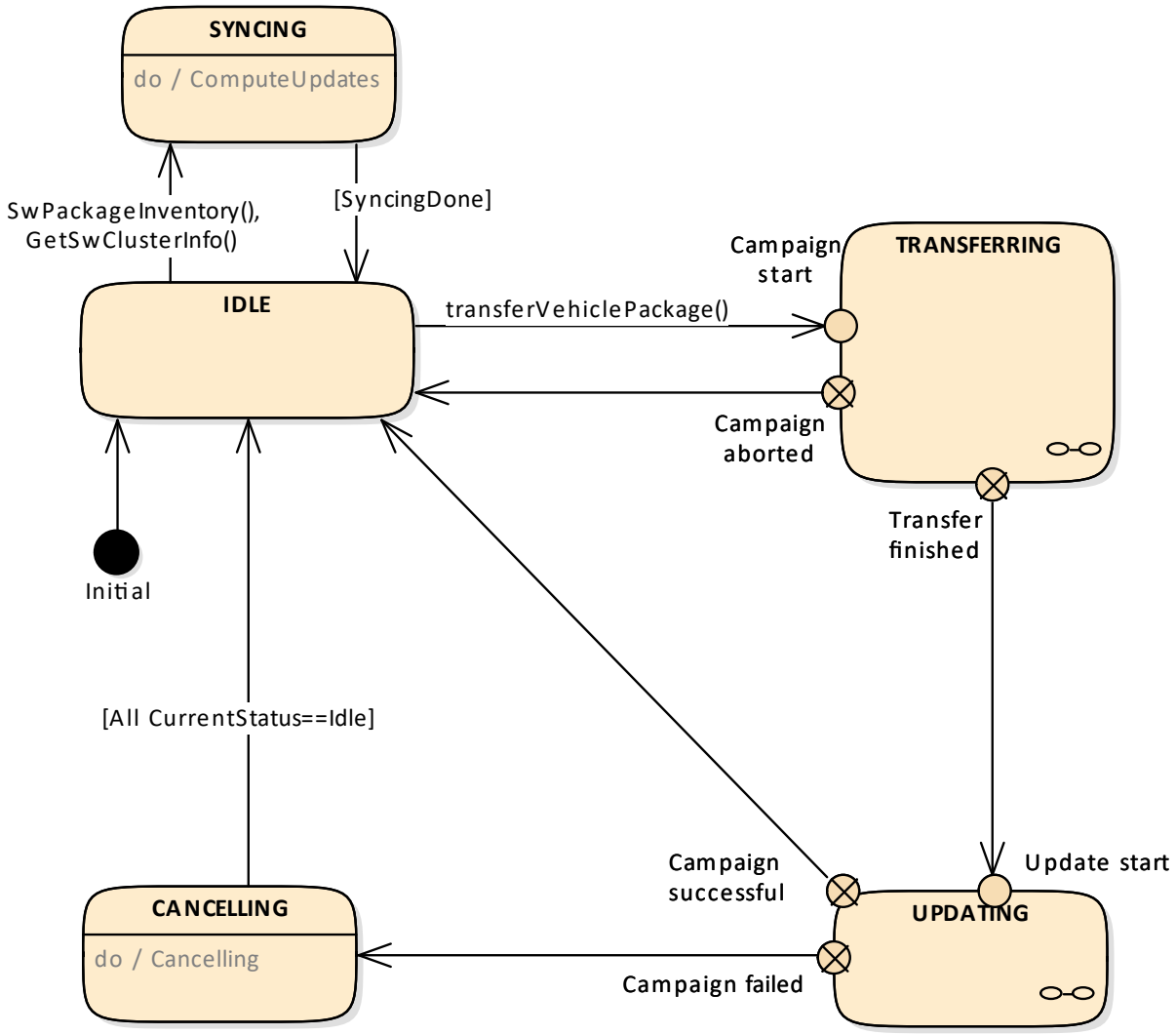

Figure 7.9: Campaign State Machine for OTA Client (`TransferState` field)

[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_01265]{DRAFT} **TransferState field** ⌈UCM Master shall provide the state of a campaign over the TransferState field of the UCM Master's VehiclePackageManagement service interface.⌋*(RS_UCM_00042)*

[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. 7.8 detailing UCM Master campaign states and transitions.

7.2.8.1 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_01207]{DRAFT} **Trigger on `kSoftwarePackage_Transferring` state** ⌈On transition to `kSoftwarePackage_Transferring` 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_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 `kProcessing` state** ⌈If `UCM Master` is in `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_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)*

7.2.8.2 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 `kSoftwarePackage_Transferring` state** ⌈If UCM Master is in kVehiclePackageTransferring state, UCM Master shall enter the kSoftwarePackage_Transferring state on successful completion of TransferExit (Vehicle Package).⌋*(RS_UCM_00035, RS_UCM_00037, RS_UCM_00038)*

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

[SWS_UCM_01228]{DRAFT} **Transition from `kSoftwarePackage_Transferring` state to `kProcessing` state** ⌈If UCM Master is in kSoftwarePackage_Transferring state, all Software Packages are ready for processing (transfer is complete without errors) or at least one Software Package started being processed by ProcessSwPackage call according to the campaign orchestration, UCM Master shall enter the kProcessing state.⌋*(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 kSoftwarePackage_Transferring 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_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.⌋(*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 (CancelCampaign) and completion.⌋(*RS_UCM_00035*)

[SWS_UCM_01239]{DRAFT} **Transition from kActivating state to kCancelling state** ⌈If UCM Master is in kActivating state, UCM Master shall enter the kCancelling 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 kVehicleChecking 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 CancelCampaign to any of its client to cancel from kTransferring or kProcessing.⌋(*RS_UCM_00035, RS_UCM_00037*)

CancelCampaign 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 CancelCampaign to any of its client to cancel from kActivating.⌋(*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 or kUpdating), UCM Master shall cancel and go back to kIdle.⌋*(RS_UCM_00035, RS_UCM_00037)*

[SWS_UCM_01270]{DRAFT} **New campaign disabling** ⌈UCM Master shall remain in kIdle when a CancelCampaign method has been called with DisableCampaign parameter set.⌋*(RS_UCM_00035)*

[SWS_UCM_01271]{DRAFT} **New campaign enabling** ⌈UCM Master shall provide a method AllowCampaign to any of its client to reallow new campaign after a CancelCampaign method was called with DisableCampaign parameter set.⌋*(RS_UCM_00035)*

7.2.9 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] Method to read History Report ⌈UCM Master shall provide a method GetCampaignHistory to retrieve all actions that have been performed by UCM Master when exiting state kUpdating from a specific time window.⌋*(RS_UCM_00034)*

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

[SWS_UCM_01266]{DRAFT} **Subordinate Not Available On The Network** ⌈UCM Master shall record persistently the error SubordinateNotAvailableOnTheNetwork in case one of the UCM subordinate involved in the current campaign stops offering its service interface and later report it with GetCampaignHistory.⌋*(RS_UCM_00034)*

[SWS_UCM_01267]{DRAFT} **Vehicle State Manager Communication Error** ⌈UCM Master shall record persistently the error VehicleStateManagerCommunicationError in case the communication with Vehicle State Manager is not possible and later report it with GetCampaignHistory.⌋*(RS_UCM_00034)*

[SWS_UCM_01268]{DRAFT} **Vehicle Driver Interface Communication Error** ⌈UCM Master shall record persistently the error VehicleDriverInterfaceCommunicationError in case the communication with Vehicle Driver Interface is no longer possible and later report it with GetCampaignHistory.⌋*(RS_UCM_00034)*

[SWS_UCM_01269]{DRAFT} **Campaign cancellation history** ⌈If CancelCampaign method is called, UCM Master shall record persistently this event to later report it with GetCampaignHistory.⌋*(RS_UCM_00034)*

7.2.10 Content of Vehicle Package

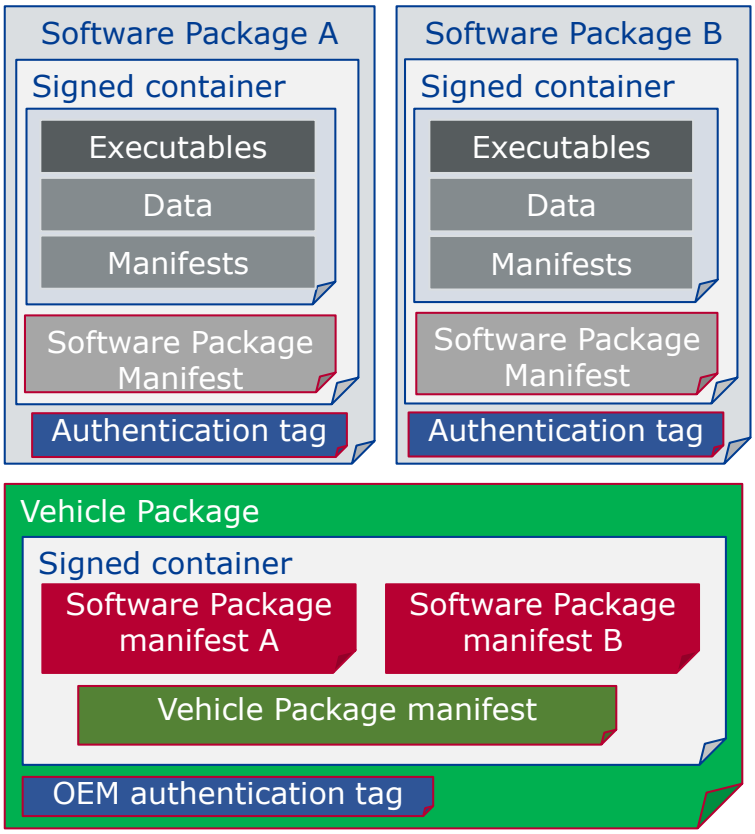

Figure 7.10: Vehicle package overview

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:

- 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)*

[SWS_UCM_01305]{DRAFT} **Vehicle Package format** ⌈Vehicle Package shall contain Vehicle Package manifest and Software Packages manifests of ARXML format.⌋*(RS_UCM_00035, RS_UCM_00043)*

[SWS_UCM_01306]{DRAFT} **TransferExit Invalid package manifest** ⌈TransferExit shall raise the error ApplicationErrorInvalidPackageManifest upon receive of an invalid manifest.⌋*(RS_UCM_00012)*

7.2.11 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 `GetCampaignHistory` 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} ⌈

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

⌋(*RS_UCM_00036*)

9.1.2 TransferIdType

[SWS_UCM_00031]{DRAFT} ⌈

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

⌋(*RS_UCM_00019*, *RS_UCM_00025*)

9.1.3 SwNameType

[SWS_UCM_00071]{DRAFT} ⌈

Name	SwNameType
Kind	STRING
Derived from	-
Description	SoftwareCluster or SoftwarePackage shortName attribute inherited from referrable meta Class.

⌋(*RS_UCM_00002*)

9.1.4 SwNameVectorType

[SWS_UCM_00174]{DRAFT} ⌈

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

⌋*(RS_UCM_00002)*

9.1.5 StrongRevisionLabelString

[SWS_UCM_00175]{DRAFT} ⌈

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

⌋*(RS_UCM_00002)*

9.1.6 SwNameVersionType

[SWS_UCM_00176]{DRAFT} ⌈

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

⌋*(RS_UCM_00002)*

9.1.7 SwNameVersionVectorType

[SWS_UCM_00177]{DRAFT} ⌈

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

⌋(*RS_UCM_00002*)

9.1.8 ByteVectorType

[SWS_UCM_00032]{DRAFT} ⌈

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

⌋(*RS_UCM_00025*)

9.1.9 SwPackageStateType

[SWS_UCM_00038]{DRAFT} ⌈

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.

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

9.1.10 SwPackageInfoType

[SWS_UCM_00039]{DRAFT} ⌈

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.

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

9.1.11 SwPackageInfoVectorType

[SWS_UCM_00040]{DRAFT} ⌈

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

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

9.1.12 SwDescType

[SWS_UCM_00186]{DRAFT} ⌈

Name	SwDescType
Kind	STRUCTURE
Subelements	Name `SwNameType` Version `StrongRevisionLabelString` TypeApproval `string` License `string` ReleaseNotes `string` Size `uint64_t`
Derived from	-

▽

△

Description	Contains general information related to SoftwareCluster that can be used by Vehicle Driver Application or Human Interface.

⌋(*RS_UCM_00002*, *RS_UCM_00011*)

9.1.13 SwDescVectorType

[SWS_UCM_00187]{DRAFT} ⌈

Name	SwDescVectorType
Kind	VECTOR
Subelements	`SwDescType`
Derived from	-
Description	Represents a dynamic size array of SoftwareCluster description

⌋(*RS_UCM_00002*, *RS_UCM_00011*)

9.1.14 SwClusterStateType

[SWS_UCM_00077]{DRAFT} ⌈

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.

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

9.1.15 SwClusterInfoType

[SWS_UCM_00078]{DRAFT} ⌈

Name	SwClusterInfoType
Kind	STRUCTURE
Subelements	Name `SwNameType` Version `StrongRevisionLabelString` State `SwClusterStateType`
Derived from	-
Description	Represents the information of a SoftwareCluster.

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

9.1.16 SwClusterInfoVectorType

[SWS_UCM_00079]{DRAFT} ⌈

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

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

9.1.17 PackageManagerStatusType

[SWS_UCM_00044]{DRAFT} ⌈

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.

⌋(*RS_UCM_00024*, *RS_UCM_00026*)

9.1.18 ActionType

[SWS_UCM_00132]{DRAFT} ⌈

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.

⌋(*RS_UCM_00032*)

9.1.19 ResultType

[SWS_UCM_00133]{DRAFT} ⌈

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.

⌋(*RS_UCM_00032*)

9.1.20 GetHistoryType

[SWS_UCM_00134]{DRAFT} ⌈

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).

⌋(*RS_UCM_00032*)

9.1.21 GetHistoryVectorType

[SWS_UCM_00135]{DRAFT} ⌈

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

⌋(*RS_UCM_00032*)

9.1.22 CampaignHistoryType

[SWS_UCM_00251]{DRAFT} ⌈

Name	CampaignHistoryType
Kind	STRUCTURE
Subelements	CampaignError `CampaignErrorType` HistoryVector `HistoryVectorType`
Derived from	-
Description	Campaign history

⌋(*RS_UCM_00034*)

9.1.23 CampaignErrorType

[SWS_UCM_00252]{DRAFT} ⌈

Name	CampaignErrorType
Kind	STRUCTURE
Subelements	CampaignFailure `CampaignFailureType` UCMStepError `UCMStepErrorType`
Derived from	-
Description	Campaign Error

⌋(*RS_UCM_00034*)

9.1.24 CampaignFailureType

[SWS_UCM_00256]{DRAFT} ⌈

Name	CampaignFailureType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Campaign failure	
Range / Symbol	***Limit***	***Description***
kUCMError	0x01	UCM error
kInvalidVehiclePackage Manifest	0x02	Vehicle Package manifest is invalid
kSubordinateNotAvailableOn TheNetwork	0x03	UCM subordinate not reachable
kVehicleStateManager CommunicationError	0x04	Communication error with Vehicle State Manager
kVehicleDriverInterface CommunicationError	0x05	Communication error with Vehicle Driver Interface
kCampaignCancelled	0x06	Campaign was cancelled

⌋(*RS_UCM_00034*)

9.1.25 UCMStepErrorType

[SWS_UCM_00253]{DRAFT} ⌈

Name	UCMStepErrorType
Kind	STRUCTURE
Subelements	id `UCMIdentifierType` SoftwarePackageStep `SoftwarePackageStepType` ReturnedError `uint8_t`
Derived from	-
Description	UCM Error

⌋(*RS_UCM_00034*)

9.1.26 SoftwarePackageStepType

[SWS_UCM_00255]{DRAFT} ⌈

Name	SoftwarePackageStepType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	UCM Software Package step at which error occured	
Range / Symbol	***Limit***	***Description***
kTransfer	0x00	Software Package transfer
kProcess	0x01	Software Package processing
kPreActivate	0x02	Software Cluster pre activation
kVerify	0x03	Software Cluster verification

⌋(*RS_UCM_00034*)

9.1.27 HistoryVectorType

[SWS_UCM_00254]{DRAFT} ⌈

Name	HistoryVectorType
Kind	STRUCTURE
Subelements	id `UCMIdentifierType` HistoryVector `GetHistoryVectorType`
Derived from	-
Description	History of an UCM

⌋(*RS_UCM_00034*)

9.1.28 CampaignStateType

[SWS_UCM_01177]{DRAFT} ⌈

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.
kSoftwarePackage_ Transferring	0x03	UCM Master is transferring software packages to the UCM subordinates.
kProcessing	0x04	The processing of software packages on UCM subordinates is ongoing. The transferring of software packages may still occur.
kActivating	0x05	The activation of SWCLs on UCM subordinates is ongoing.
kVehicleChecking	0x06	UCM Master is performing post-activation checks (OEM specific).
kCancelling	0x07	UCM Master is rolling-back the activated SWCLs on the UCM subordinates.

⌋(*RS_UCM_00032*)

9.1.29 TransferStateType

[SWS_UCM_01178]{DRAFT} ⌈

Name	TransferStateType	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Represents the state of an update from OTA Client perspective.	
Range / Symbol	***Limit***	***Description***
kIdle	0x00	UCM Master is ready to start a software update campaign.
kTransferring	0x01	Vehicle or Software Packages are being transferred.
kUpdating	0x02	Software Clusters are being updated in the vehicle.
kCancelling	0x03	An error occurred, campaign is being cancelled, reverting changes.

⌋(*RS_UCM_00032*)

9.1.30 SafetyPolicyType

[SWS_UCM_01114]{DRAFT} ⌈

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

⌋(*RS_UCM_00002*)

9.2 Provided Service Interfaces

9.2.1 Package Management

This chapter lists all provided service interfaces of the UCM.

Port

[SWS_UCM_00073]{DRAFT} ⌈

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

⌋(*RS_UCM_00001*)

Service Interface

[SWS_UCM_00131]{DRAFT} ⌈

Name	PackageManagement
NameSpace	ara::ucm::pkgmgr

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

Method	Activate	
Description	This method activates the processed components.	
FireAndForget	false	
Application Errors	MissingDependencies	Activate cannot be performed because of missing dependencies.
Application Errors	OperationNotPermitted	The operation is not supported in the current context.
Application Errors	UpdateSessionRejected	Start of an update session was rejected by State Management
Application Errors	PreActivationFailed	Error during preActivation step.
Application Errors	VerificationFailed	Error during verification step.

Method	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	Invalid-TransferId	The Transfer ID is invalid.
Application Errors	Opera-tionNotPer-mitted	The operation is not supported in the current context.

Method	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	Invalid-TransferId	The Transfer ID is invalid.
Application Errors	Opera-tionNotPer-mitted	The operation is not supported in the current context.

Method	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	Opera-tionNotPer-mitted	The operation is not supported in the current context.

Method	GetHistory	
Description	Getter method to retrieve all actions that have been performed by UCM.	
FireAndForget	false	
	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

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

Method	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

Method	GetSwClusterDescription	
Description	This method returns the general information of the Software Clusters present in the platform	
FireAndForget	false	
Parameter	SwCluster	
	Description	List of SoftwareClusters present in the platform.
	Type	SwDescVectorType
	Variation	
	Direction	OUT

Method	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

Method	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

Method	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
	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	Invalid-TransferId	The Transfer ID is invalid.

Method	ProcessSwPackage	
Description	Process a previously transferred Software Package.	
FireAndForget	false	
Parameter	id	
	Description	The Transfer ID of the Software Package which should be processed.
	Type	TransferIdType
	Variation	
	Direction	IN
Application Errors	AuthenticationFailed	Package authentication failed.
Application Errors	IncompatibleDelta	Delta package dependency check failed.
Application Errors	IncompatiblePackageVersion	The version of the Software or Vehicle Package to be processed is not compatible with the current version of UCM or UCM Master.
Application Errors	InsufficientComputationPower	Insufficient computation power to perform the requested operation.
Application Errors	InsufficientMemory	Insufficient memory to perform operation.
Application Errors	InvalidPackageManifest	Package manifest could not be read.
Application Errors	Invalid-TransferId	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	ProcessedSoftwarePackageInconsistent	The processed Software Package integrity check has failed.
Application Errors	ServiceBusy	Another processing is already ongoing and therefore the current processing request has to be rejected.

Method	RevertProcessedSwPackages	
Description	Revert the changes done by processing (ProcessSwPackage) of one or several software packages.	
FireAndForget	false	
Application Errors	NotAbleToRevertPackages	RevertProcessedSwPackages failed.
Application Errors	OperationNotPermitted	The operation is not supported in the current context.

Method	Rollback	
Description	Rollback the system to the state before the packages were processed.	
FireAndForget	false	
Application Errors	NotAbleToRollback	Rollback failed.
Application Errors	OperationNotPermitted	The operation is not supported in the current context.

Method	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	IncorrectBlock	The same block number is received twice.
Application Errors	IncorrectBlockSize	The size of the block exceeds the provided block size from TransferStart or TransferVehiclePackage.
Application Errors	AuthenticationFailed	Package authentication failed.
Application Errors	BlockInconsistent	Consistency check for transferred block failed.

▽

△

Application Errors	IncompatiblePackageVersion	The version of the Software or Vehicle Package to be processed is not compatible with the current version of UCM or UCM Master.
Application Errors	IncorrectSize	The size of the Software or Vehicle Package exceeds the provided size in Transfer Start.
Application Errors	InsufficientComputationPower	Insufficient computation power to perform the requested operation.
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	PackageInconsistent	Package integrity check failed.
Application Errors	TransferInterrupted	Transfer has been interrupted.

Method	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	AuthenticationFailed	Package authentication failed.
Application Errors	IncompatiblePackageVersion	The version of the Software or Vehicle Package to be processed is not compatible with the current version of UCM or UCM Master.
Application Errors	InsufficientData	TransferExit has been called but total transferred data size does not match expected data size provided with TransferStart call.
Application Errors	InvalidPackageManifest	Package manifest could not be read.
Application Errors	InvalidTransferId	The Transfer ID is invalid.
Application Errors	MissingDependencies	Activate cannot be performed because of missing dependencies.
Application Errors	OldVersion	Software Package version is too old.
Application Errors	OperationNotPermitted	The operation is not supported in the current context.
Application Errors	PackageInconsistent	Package integrity check failed.

Method	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
Parameter	BlockSize	
	Description	Size of the blocks to be received with TransferData method.
	Type	`uint32_t`
	Variation	
	Direction	OUT
Application Errors	`InsufficientComputationPower`	Insufficient computation power to perform the requested operation.
Application Errors	`InsufficientMemory`	Insufficient memory to perform operation.

⌋(*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 Vehicle Package Management

This chapter lists all provided service interfaces of the `UCM Master` to OTA Client Adaptive Application.

Port

[SWS_UCM_00178]{DRAFT} ⌈

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

⌋(*RS_UCM_00035*)

Service Interface

[SWS_UCM_00181]{DRAFT} ⌈

Name	VehiclePackageManagement
NameSpace	ara::ucm::pkgmgr

Field	TransferState
Description	The current status of Campaign from an OTA Client perspective.
Type	TransferStateType
HasGetter	true
HasNotifier	true
HasSetter	false

Field	RequestedPackage
Description	Software Package to be transfered to UCM Master.
Type	SwNameVersionType
HasGetter	true
HasNotifier	true
HasSetter	false

Field	SafetyState
Description	Vehicle state computed by the Vehicle State Manager Adaptive Application.
Type	bool
HasGetter	true
HasNotifier	true
HasSetter	false

Method	CancelCampaign	
Description	This method aborts an ongoing campaign processing of a Vehicle Package.	
FireAndForget	false	
	DisableCampaign	
	Description	To forbid new campaign
	Type	bool

▽

△

	Variation	
	Direction	IN
Application Errors	`CancelFailed`	Cancel failed.
Application Errors	`InvalidTransferId`	The Transfer ID is invalid.
Application Errors	`OperationNotPermitted`	The operation is not supported in the current context.

Method	AllowCampaign
Description	To allow a new campaign to start
FireAndForget	false

Method	DeleteTransfer	
Description	Delete a transferred Software or Vehicle Package.	
FireAndForget	false	
Parameter	id	
	Description	Transfer ID of the currently running request.
	Type	`TransferIdType`
	Variation	
	Direction	IN
Application Errors	`InvalidTransferId`	The Transfer ID is invalid.
Application Errors	`OperationNotPermitted`	The operation is not supported in the current context.

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

▽

△

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

Method	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

Method	GetSwPackages	
Description	This method returns the Software Packages that are part of current campaign handled by UCM Master.	
FireAndForget	false	
Parameter	Packages	
	Description	List of Software Packages.
	Type	`SwPackageInfoVectorType`
	Variation	
	Direction	OUT

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

▽

△

	Variation	
	Direction	OUT

Method	TransferData	
Description	Block-wise transfer of a Software or Vehicle Package to UCM Master.	
FireAndForget	false	
Parameter	id	
	Description	Transfer ID.
	Type	`TransferIdType`
	Variation	
	Direction	IN
Parameter	data	
	Description	Data block of the Software or Vehicle Package.
	Type	`ByteVectorType`
	Variation	
	Direction	IN
Parameter	blockCounter	
	Description	Block counter value of the current block.
	Type	`uint64_t`
	Variation	
	Direction	IN
Application Errors	`IncorrectBlock`	The same block number is received twice.
Application Errors	`IncorrectBlockSize`	The size of the block exceeds the provided block size from TransferStart or TransferVehiclePackage.
Application Errors	`AuthenticationFailed`	Package authentication failed.
Application Errors	`BlockInconsistent`	Consistency check for transferred block failed.
Application Errors	`IncompatiblePackageVersion`	The version of the Software or Vehicle Package to be processed is not compatible with the current version of UCM or UCM Master.
Application Errors	`IncorrectSize`	The size of the Software or Vehicle Package exceeds the provided size in TransferStart.
Application Errors	`InsufficientComputationPower`	Insufficient computation power to perform the requested operation.
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	PackageInconsistent	Package integrity check failed.
Application Errors	TransferInterrupted	Transfer has been interrupted.

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

Method	TransferStart	
Description	Start the transfer of a Software Package. The name of the Software Package to be transferred to UCM Master must be provided. UCM Master will generate a Transfer ID for subsequent calls to TransferData, TransferExit, DeleteTransfer. Size of Software Package to be used to transfer to UCM subordinate is available in the Vehicle Package and its contained Software Package Manifests.	
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
Parameter	BlockSize	
	Description	Size of the blocks to be received with TransferData method.
	Type	`uint32_t`
	Variation	
	Direction	OUT
Application Errors	`UnexpectedPackage`	The Software Package name does not correspond to the RequestedPackage field value.
Application Errors	`InsufficientComputationPower`	Insufficient computation power to perform the requested operation.
Application Errors	`InsufficientMemory`	Insufficient memory to perform operation.

Method	TransferVehiclePackage	
Description	Start the transfer of a Vehicle Package. The size of the Vehicle Package to be transferred to UCM Master must be provided. UCM Master will generate a Transfer ID for subsequent calls to TransferData, Transfer Exit, ProcessSwPackage, DeleteTransfer.	
FireAndForget	false	
Parameter	size	
	Description	Size (in bytes) of the Vehicle Package to be transferred.
	Type	`uint64_t`
	Variation	
	Direction	IN
Parameter	id	
	Description	Return TransferId.
	Type	`TransferIdType`
	Variation	
	Direction	OUT
Parameter	BlockSize	
	Description	Size of the blocks to be received with TransferData method.
	Type	`uint32_t`
	Variation	
	Direction	OUT
Application Errors	`NewCampaignDisabled`	New campaigns are disabled, calling AllowCampaign will enable new campaigns.

▽

△

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

⌋(*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*)

9.2.3 Vehicle Driver Application Interface

This chapter lists all provided service interfaces of the `UCM Master` to the Vehicle Driver Adaptive Application.

Port

[SWS_UCM_00180]{DRAFT} ⌈

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

⌋(*RS_UCM_00038*, *RS_UCM_00043*)

Service Interface

[SWS_UCM_00182]{DRAFT} ⌈

Name	VehicleDriverApplication

Field	ApprovalRequired
Description	Flag to inform Adaptive Application if approval from Vehicle Driver is required at current state based on Vehicle Package Manifest.
Type	`bool`
HasGetter	true
HasNotifier	true
HasSetter	false

Field	CampaignState
Description	The current status of Campaign.
Type	`CampaignStateType`

▽

△

HasGetter	true
HasNotifier	true
HasSetter	false

Field	SafetyPolicy
Description	Safety policy from the Vehicle Package to be computed by the Vehicle State Manager Adaptive Application.
Type	`SafetyPolicyType`
HasGetter	true
HasNotifier	true
HasSetter	false

Field	SafetyState
Description	Vehicle state computed by the Vehicle State Manager Adaptive Application.
Type	`bool`
HasGetter	true
HasNotifier	true
HasSetter	false

Method	CancelCampaign	
Description	This method aborts an ongoing campaign processing of a Vehicle Package.	
FireAndForget	false	
Parameter	DisableCampaign	
	Description	To forbid new campaign
	Type	`bool`
	Variation	
	Direction	IN
Application Errors	`CancelFailed`	Cancel failed.
Application Errors	`Invalid-TransferId`	The Transfer ID is invalid.
Application Errors	`Opera-tionNotPer-mitted`	The operation is not supported in the current context.

Method	AllowCampaign
Description	To allow a new campaign to start
FireAndForget	false

Method	DriverApproval	
Description	Called by Adaptive Application to inform UCM Master of the driver's notification resolution (approve or reject)	
FireAndForget	false	
Parameter	Approval	
	Description	Driver's notification resolution
	Type	`bool`
	Variation	
	Direction	IN
Parameter	SafetyPolicy	
	Description	Safety policy computed by the Vehicle State Manager Adaptive Application
	Type	`SafetyPolicyType`
	Variation	
	Direction	IN

Method	GetCampaignHistory	
Description	Getter method to retrieve all actions that have been performed by UCM Master.	
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 Master.
	Type	`CampaignHistoryType`
	Variation	
	Direction	OUT

Method	GetSwClusterDescription	
Description	This method returns the general information of the Software Clusters present in the Adaptive Platform	
FireAndForget	false	
Parameter	SoftwareClusterDescriptions	
	Description	List of SoftwareClusters general information
	Type	SwDescVectorType
	Variation	
	Direction	OUT

Method	GetSwPackageDescription	
Description	This method returns the general information of the Software Packages that are part of current campaign handled by UCM Master.	
FireAndForget	false	
Parameter	Packages	
	Description	List of Software Packages.
	Type	SwDescVectorType
	Variation	
	Direction	OUT

Method	GetSwProcessProgress	
Description	Get the progress (0 - 100%) of the currently package 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

Method	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

⌋(*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*)

9.2.4 Vehicle State Manager

This chapter lists all provided service interfaces of the `UCM Master` to the Vehicle State Manager Adaptive Application.

Port

[SWS_UCM_00179]{DRAFT} ⌈

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

⌋(*RS_UCM_00037*, *RS_UCM_00043*)

Service Interface

[SWS_UCM_00183]{DRAFT} ⌈

Name	VehicleStateManager

Field	SafetyPolicy
Description	Safety policy from the Vehicle Package to be computed by the Vehicle State Manager Adaptive Application.
Type	`SafetyPolicyType`
HasGetter	true
HasNotifier	true
HasSetter	false

Method	SafetyState	
Description	Method called by Vehicle State Manager Adaptive Application when safety state is changed	
FireAndForget	false	
	SafetyPolicy	
	Description	Safety policy computed by the Vehicle State Manager Adaptive Application
	Type	`SafetyPolicyType`
	Variation	

▽

△

Parameter	***Direction***	IN
	SafeToUpdate	
	Description	Vehicle safety state
	Type	`bool`
	Variation	
	Direction	OUT

⌋(*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 Required Interface

9.3.1 State Management Update Request

UCM requires the `UpdateRequest` Service Interface [SWS_SM_91017] provided by State Management

9.4 Application Errors

9.4.1 Application Error Domain

9.4.1.1 UCMErrorDomain

This section lists all application errors of the UCM.

[SWS_UCM_00136]{DRAFT} ⌈

Name	***Code***	***Description***
NewCampaignDisabled	31	New campaigns are disabled, calling AllowCampaign will enable new campaigns.
UnexpectedPackage	32	The Software Package name does not correspond to the RequestedPackage field value.
IncorrectBlock	2	The same block number is received twice.
IncorrectBlockSize	30	The size of the block exceeds the provided block size from Transfer Start or TransferVehiclePackage.
AuthenticationFailed	8	Package authentication failed.
BlockInconsistent	25	Consistency check for transferred block failed.
CancelFailed	16	Cancel failed.

▽

△

IncompatibleDelta	29	Delta package dependency check failed.
IncompatiblePackageVersion	24	The version of the Software or Vehicle Package to be processed is not compatible with the current version of UCM or UCM Master.
IncorrectSize	3	The size of the Software or Vehicle Package exceeds the provided size in TransferStart.
InsufficientComputationPower	28	Insufficient computation power to perform the requested operation.
InsufficientData	6	TransferExit has been called but total transferred data size does not match expected data size provided with TransferStart call.
InsufficientMemory	1	Insufficient memory to perform operation.
InvalidPackageManifest	13	Package manifest could not be read.
InvalidTransferId	4	The Transfer ID is invalid.
MissingDependencies	21	Activate cannot be performed because of missing dependencies.
NotAbleToRevertPackages	15	RevertProcessedSwPackages failed.
NotAbleToRollback	18	Rollback failed.
OldVersion	9	Software Package version is too old.
OperationNotPermitted	5	The operation is not supported in the current context.
PackageInconsistent	7	Package integrity check failed.
ProcessSwPackageCancelled	22	The processing operation has been interrupted by a Cancel() call.
ProcessedSoftwarePackageInconsistent	23	The processed Software Package integrity check has failed.
ServiceBusy	12	Another processing is already ongoing and therefore the current processing request has to be rejected.
TransferInterrupted	26	Transfer has been interrupted.
UpdateSessionRejected	33	Start of an update session was rejected by State Management
PreActivationFailed	19	Error during preActivation step.
VerificationFailed	27	Error during verification step.

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

10 Sequence diagrams

10.1 Update process

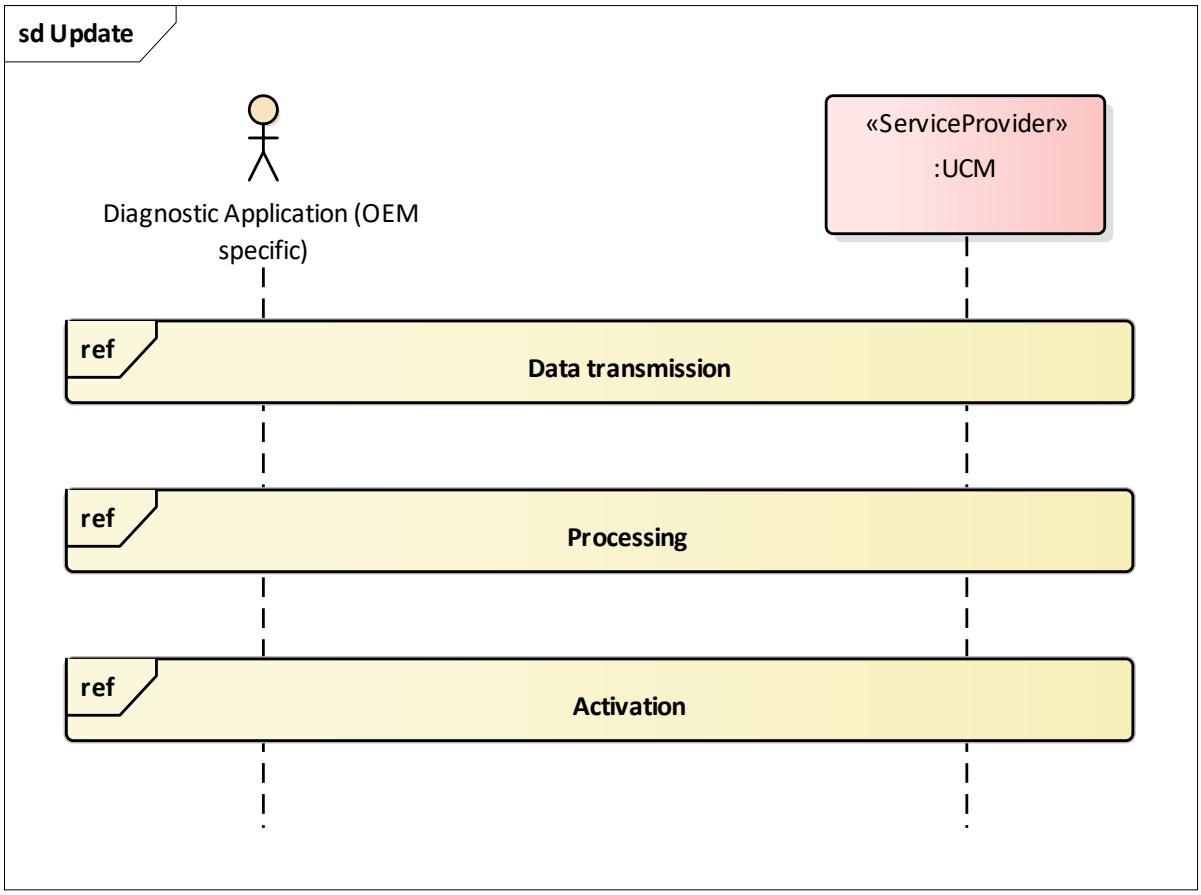

Figure 10.1: Sequence diagram showing the update process

10.2 Data transmission

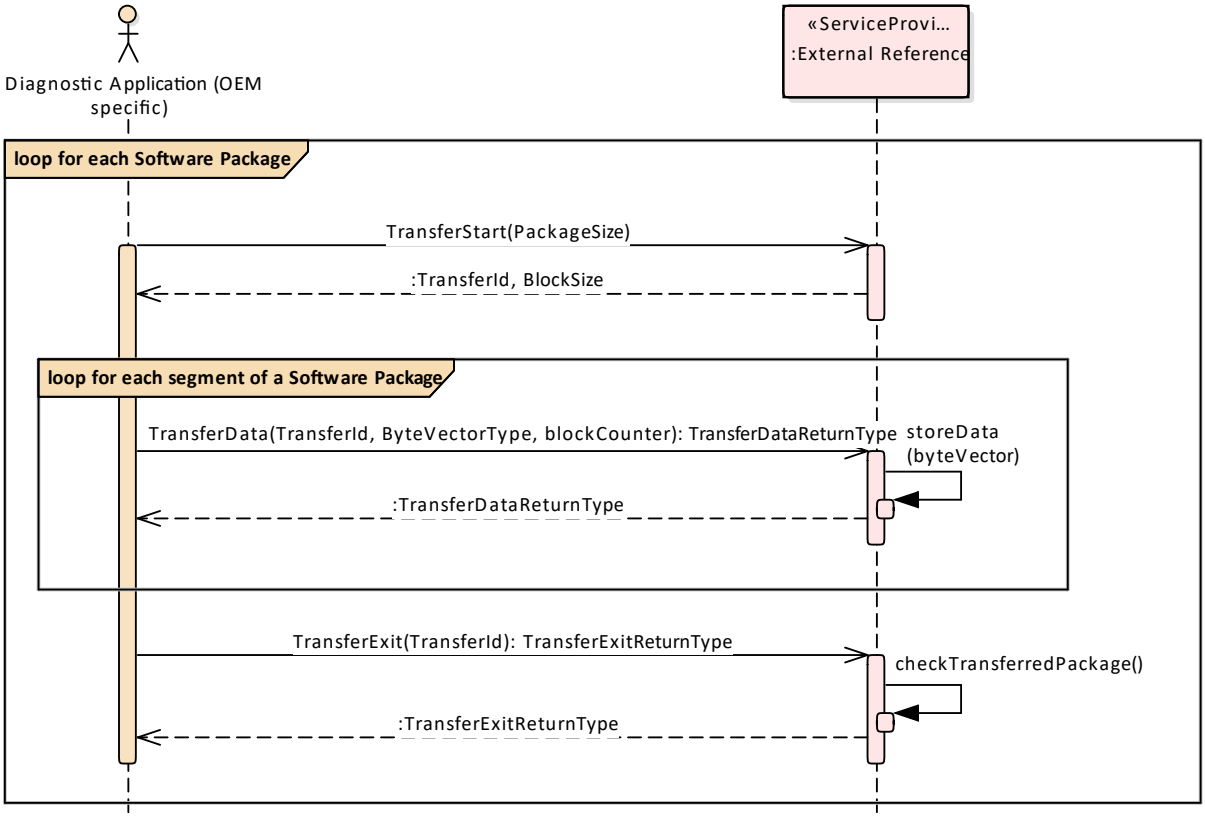

Figure 10.2: Sequence diagram showing the data transmission

10.3 Package processing

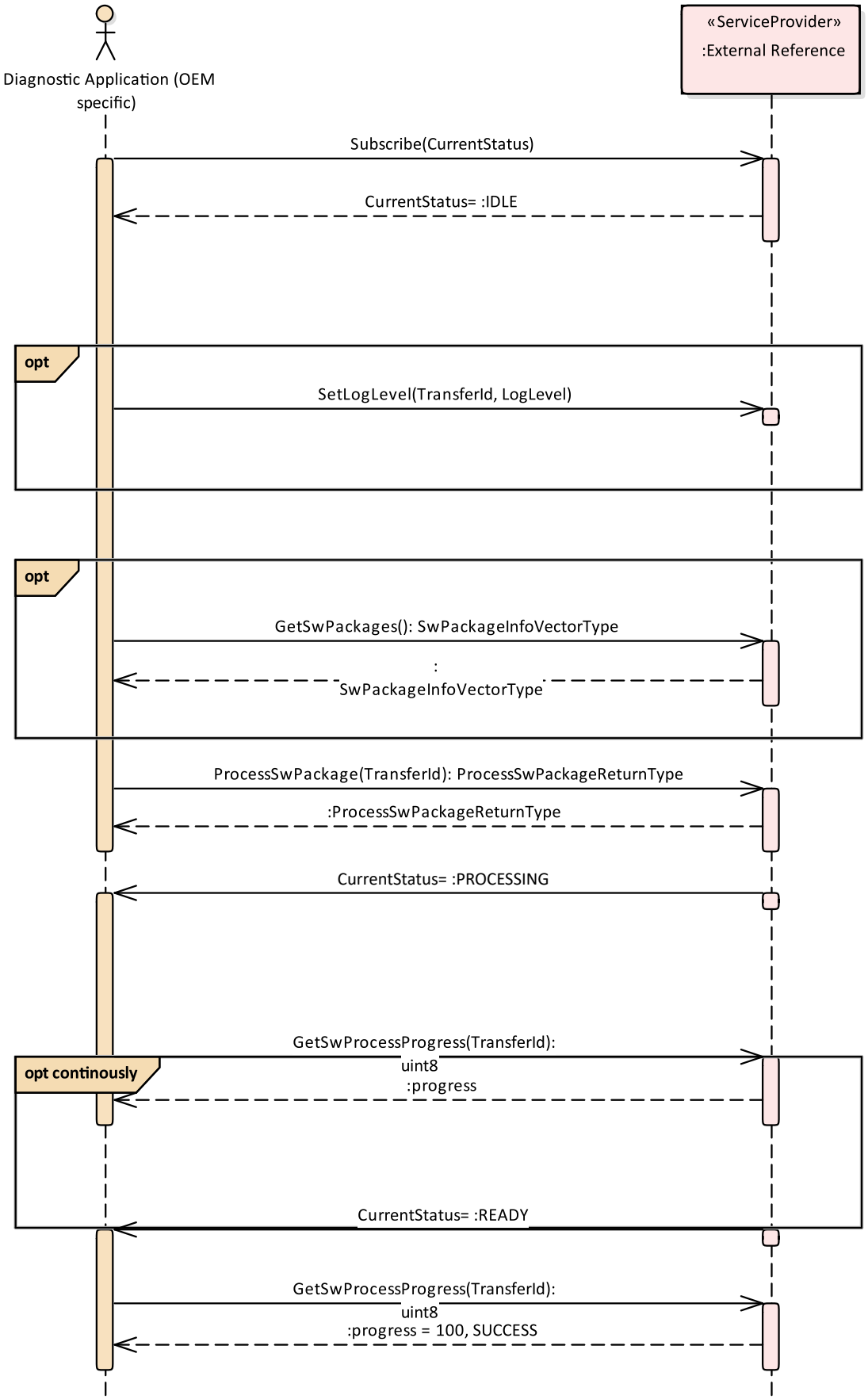

Figure 10.3: Sequence diagram showing the package processing

10.4 Activation

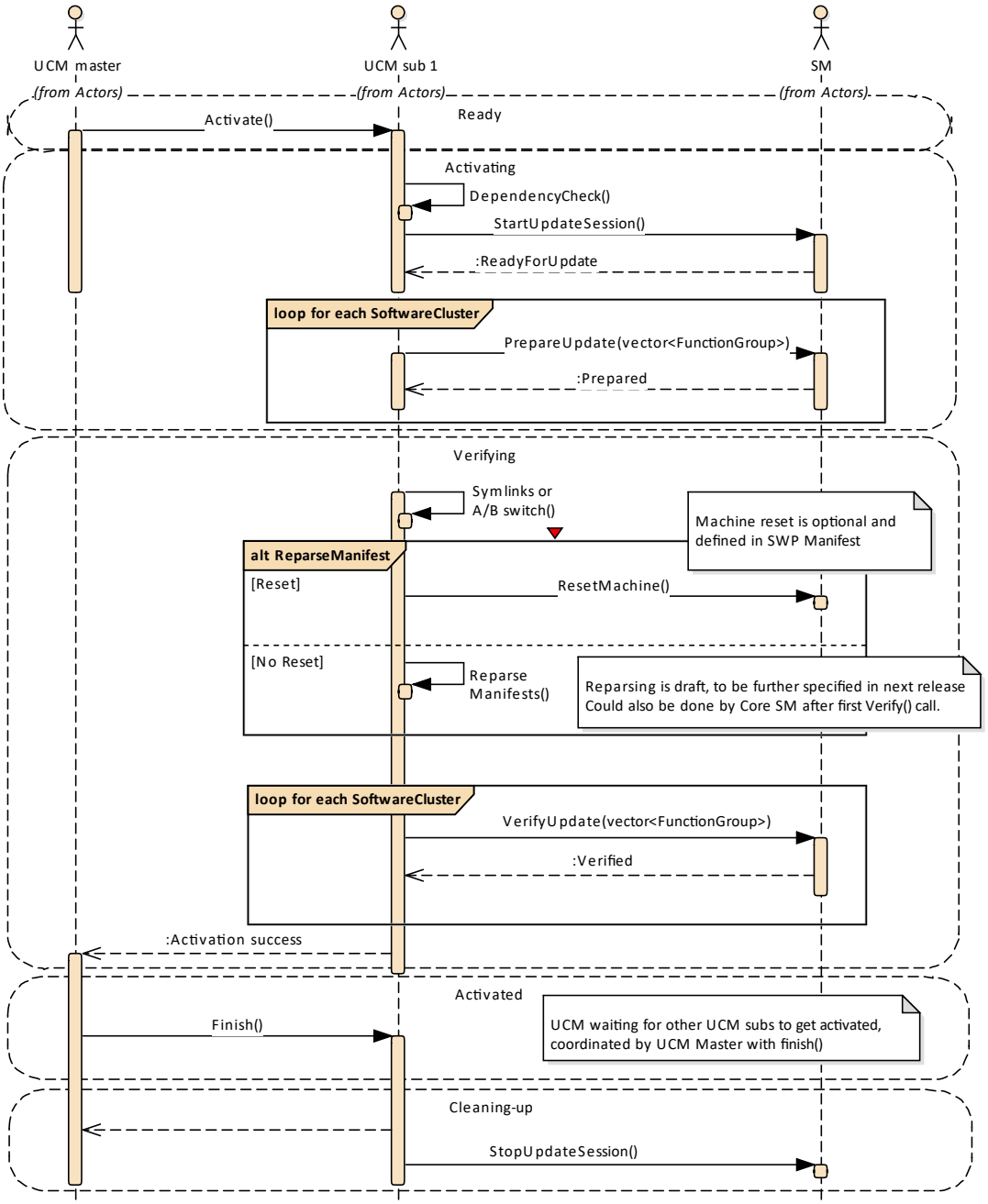

Figure 10.4: Sequence diagram showing the activation process

10.5 Failing activation

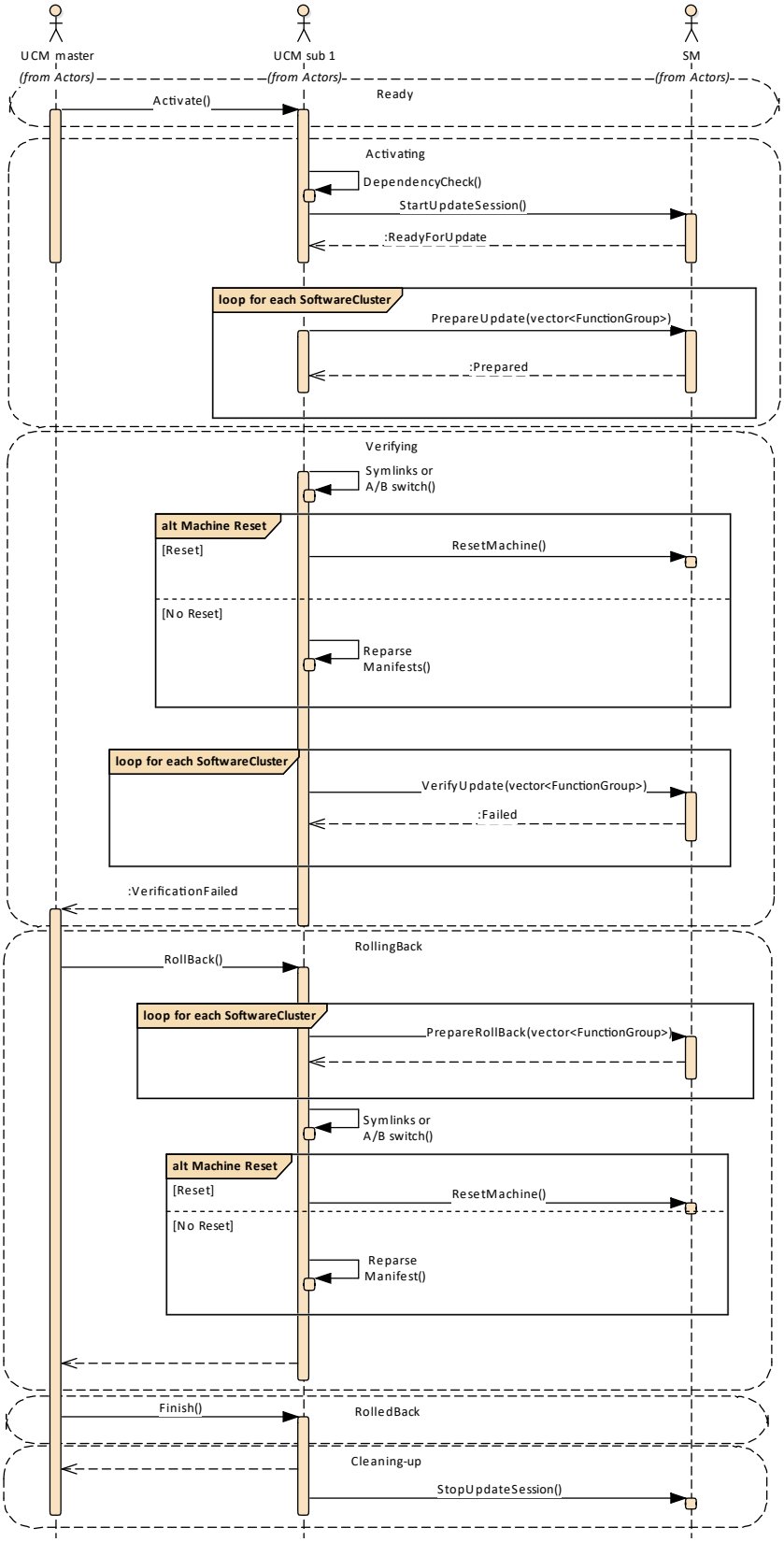

Figure 10.5: Sequence diagram showing an activation failing

10.6 UCM Master simplified vehicle update

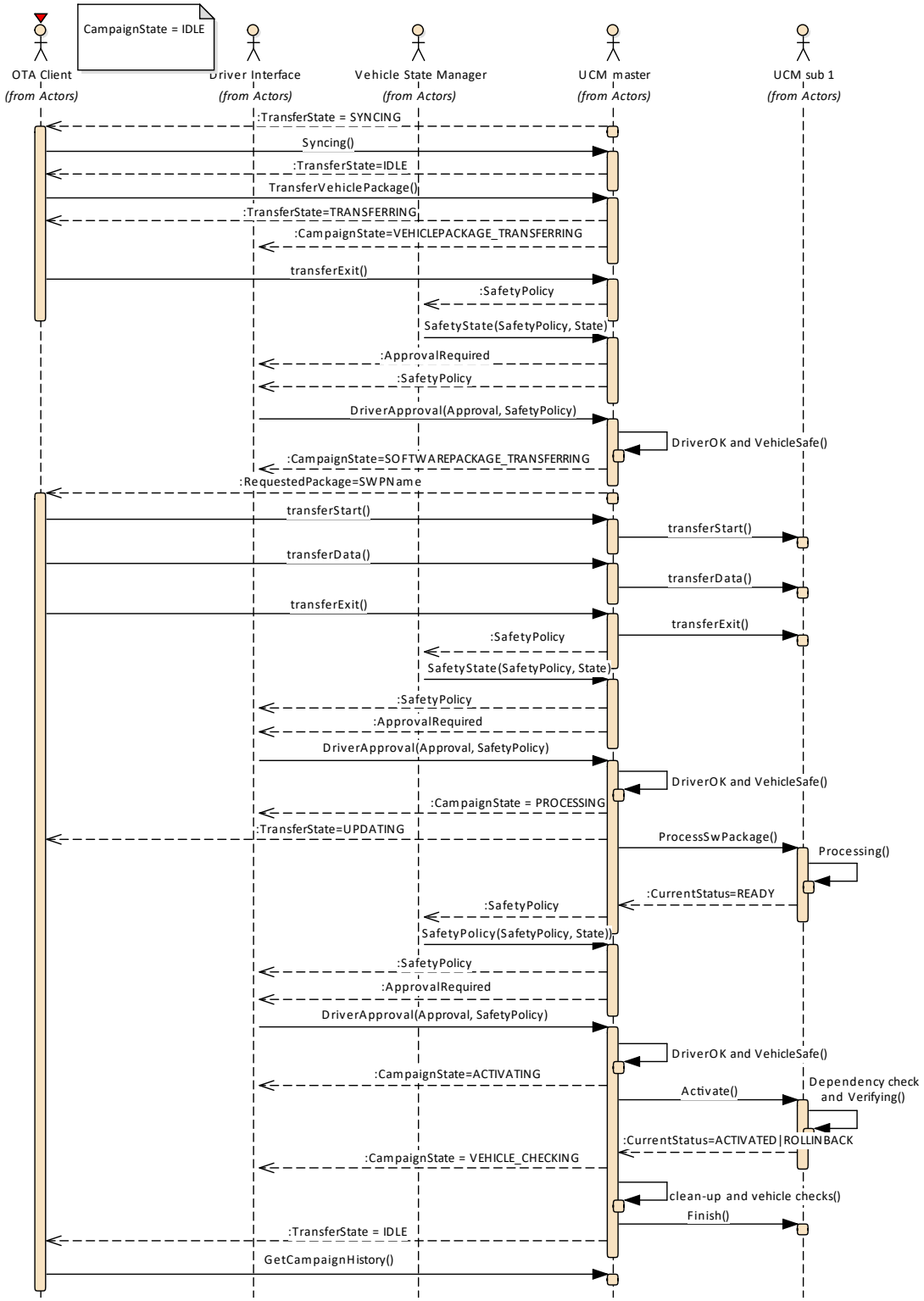

Figure 10.6: Sequence diagram showing vehicle update

A Mentioned Manifest Elements

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.

Chapter is generated.

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, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstanceFilter*, *AbstractServiceInstance*, *Abstract SignalBasedToISignalTriggeringMapping*, *AdaptiveModuleInstantiation*, AdaptiveSwcInternalBehavior, ApplicationEndpoint, ApplicationError, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, Autosar OperationArgumentInstance, AutosarVariableInstance, *BuildActionEntity*, BuildActionEnvironment, Chapter, CheckpointTransition, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *Communication Connector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, Coupling Port, *CouplingPortStructuralElement*, CryptoCertificate, CryptoKeySlot, CryptoProvider, *CryptoService Mapping*, DataPrototypeGroup, DataTransformation, DependencyOnArtifact, DeterministicClient ResourceNeeds, *DiagEventDebounceAlgorithm*, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticFunctionInhibitSource, *DiagnosticRoutineSubfunction*, DltArgument, DltLogChannel, Dlt Message, DoIpInterface, DoIpLogicAddress, DoIpRoutingActivation, E2EProfileConfiguration, End2End EventProtectionProps, EndToEndProtection, EthernetWakeupSleepOnDatalineConfig, EventMapping, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeature MapCondition, FMFeatureMapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FieldMapping, FireAndForgetMapping, *FrameTriggering*, GeneralParameter, GlobalSupervision, Global TimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HealthChannel*, *HeapUsage*, HwAttributeDef, Hw AttributeLiteralDef, HwPin, HwPinGroup, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, I SignalTriggering, *IdentCaption*, InterfaceMapping, InternalTriggeringPoint, Keyword, LifeCycleState, Linker, MacMulticastGroup, McDataInstance, MemorySection, MethodMapping, ModeDeclaration, Mode DeclarationMapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, *NmNode*, *PackageableElement*, ParameterAccess, PduToFrameMapping, PduTriggering, *PersistencyDeploymentElement*, *Persistency InterfaceElement*, *PhmSupervision*, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, PossibleError Reaction, ProcessDesignToMachineDesignMapping, ProcessToMachineMapping, Processor, Processor Core, PskIdentityToKeySlotMapping, RecoveryNotification, ResourceConsumption, ResourceGroup, *RestAbstractEndpoint*, RestElementDef, RestResourceDef, RootSwClusterDesignComponentPrototype, RootSwComponentPrototype, RootSwCompositionPrototype, RptComponent, RptContainer, Rpt ExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, *Sdg Attribute*, SdgClass, SecOcJobMapping, SecOcJobRequirement, *SecureComProps*, Secure CommunicationAuthenticationProps, *SecureCommunicationDeployment*, SecureCommunication FreshnessProps, SecurityEventContextProps, *ServiceEventDeployment*, *ServiceFieldDeployment*, ServiceInstanceToSignalMapping, *ServiceInterfaceElementMapping*, ServiceInterfaceElementSecure ComConfig, ServiceInterfaceMapping, *ServiceMethodDeployment*, *ServiceNeeds*, SignalService TranslationEventProps, SignalServiceTranslationProps, SocketAddress, SoftwarePackageStep, Someip EventGroup, SomeipProvidedEventGroup, SomeipTpChannel, *SpecElementReference*, *StackUsage*, StartupConfig, StaticSocketConnection, StructuredReq, SupervisionCheckpoint, SwGenericAxisParam Type, SwServiceArg, SwcServiceDependency, SystemMapping, SystemMemoryUsage, *TimeBase Resource*, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, Timing ModeInstance, TlsCryptoCipherSuite, TlsJobMapping, Topic1, TpAddress, TraceableTable, Traceable Text, *TracedFailure*, *TransformationProps*, TransformationPropsToServiceInterfaceElementMapping, TransformationTechnology, Trigger, UcmDescription, UcmStep, VariableAccess, VariationPointProxy, VehicleRolloutStep, ViewMap, VlanConfig			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

Class	***Identifiable*** (abstract)			
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 A.1: Identifiable

Class	**SoftwareCluster**
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution

▽

△

Class	**SoftwareCluster**			
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.Status=draft atp.recommendedPackage=SoftwareClusters			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
claimed FunctionGroup	ModeDeclarationGroup Prototype	*	ref	Each SoftwareCluster can reserve the usage of a given functionGroup such that no other SoftwareCluster is allowed to use it **Tags:**atp.Status=draft
conflictsTo	SoftwareCluster DependencyFormula	0..1	aggr	This aggregation handles conflicts. If it yields true then the SoftwareCluster shall not be installed. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=conflictsTo atp.Status=draft
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
contained Process	Process	*	ref	This reference represent the processes contained in the enclosing SoftwareCluster. **Tags:**atp.Status=draft
dependsOn	SoftwareCluster DependencyFormula	0..1	aggr	This aggregation can be taken to identify a dependency for the enclosing SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=dependsOn 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

▽

△

Class	**SoftwareCluster**			
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
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 A.2: SoftwareCluster

Class	**SoftwarePackage**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
Note	This meta-class represents the ability to formalize the content of a software package. **Tags:** 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.

▽

△

Class	**SoftwarePackage**			
compressed Software PackageSize	PositiveInteger	1	attr	This size represents the size of the compressed Software Package.
deltaPackage Applicable Version	StrongRevisionLabel String	0..1	attr	This attribute identifies the version of the included SoftwareCluster for which the enclosing SoftwarePackage can be used as a delta update
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 (ordered)	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 **InstanceRef implemented by:**FunctionGroupStateIn FunctionGroupSetInstanceRef
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 (ordered)	ModeDeclaration	*	iref	The referenced function group states shall be established for the verification of the activated software. **Tags:**atp.Status=draft **InstanceRef implemented by:**FunctionGroupStateIn FunctionGroupSetInstanceRef

Table A.3: SoftwarePackage

Primitive	**StrongRevisionLabelString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive represents a revision label which identifies an object under version control. It represents a pattern which requires three integer numbers separated by a dot, representing from left to right Major Version, MinorVersion, PatchVersion and additional labels for pre-release version and build metadata. Legal patterns are for example: 1.0.0-alpha+001 1.0.0+20130313144700 1.0.0-beta+exp.sha.5114f85

▽

▽

△

Primitive	**StrongRevisionLabelString**
	△ **Tags:** atp.Status=draft xml.xsd.customType=STRONG-REVISION-LABEL-STRING xml.xsd.pattern=(0|[1-9]\d*)\.(0|[1-9]\d*)\.(0|[1-9]\d*)(-((0|[1-9]\d*|\d*[a-zA-Z-][0-9a-z A-Z-]*)(\.(0|[1-9]\d*|\d*[a-zA-Z-][0-9a-zA-Z-]*))*))?(\+([0-9a-zA-Z-]+(\.[0-9a-zA-Z-]+)*))? xml.xsd.type=string

Table A.4: StrongRevisionLabelString

Class	**VehiclePackage**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
Note	This meta-class represents the ability to define a vehicle package for executing an update campaign. **Tags:** atp.Status=draft atp.recommendedPackage=VehiclePackages			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*			
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 A.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***

▽

△

Class	**UcmModuleInstantiation**			
identifier	String	1	attr	This represents the identification of a UCM.
maxNumberOf Parallel Transfers	PositiveInteger	0..1	attr	This attribute supports the configuration of the maximum number of parallel transfers that the Ucm on the enclosing Machine is allowed to create.

Table A.6: UcmModuleInstantiation

B Interfaces to other Functional Clusters (informative)

B.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.

B.2 Interfaces Tables

B.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.

C Packages distribution within vehicle detailed sequence examples

C.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.

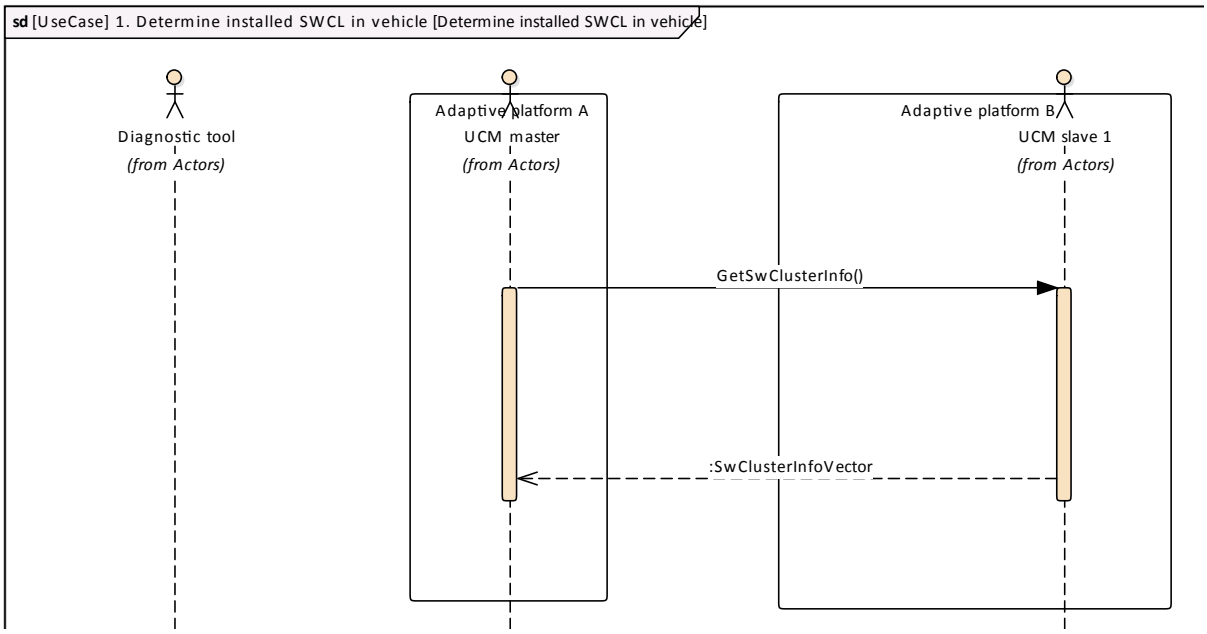

Figure C.1: Collect information of Software Clusters present in vehicle from several AUTOSAR Adaptive Platforms

C.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.

C.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.

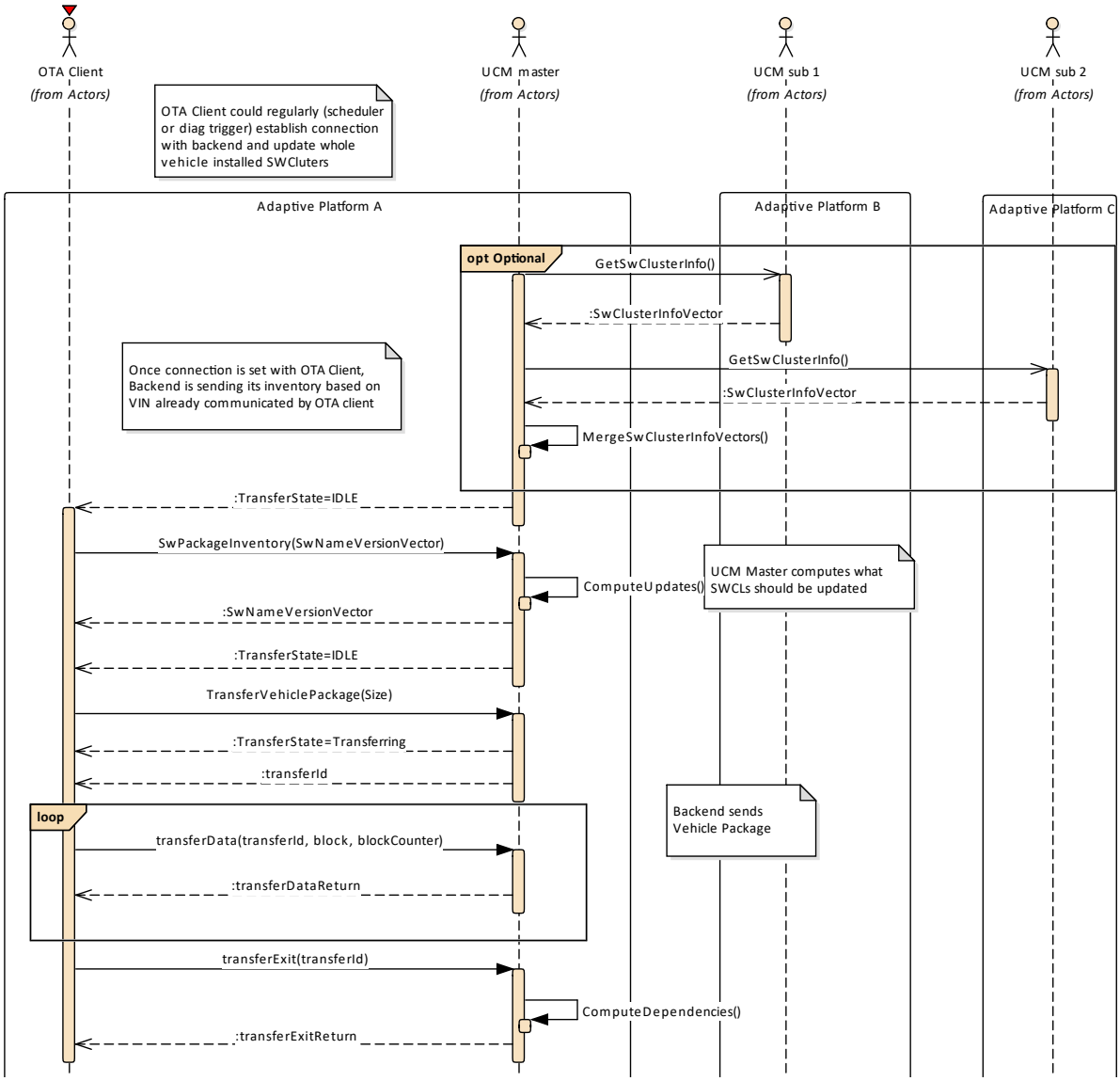

Figure C.2: Pull package from backend

C.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.

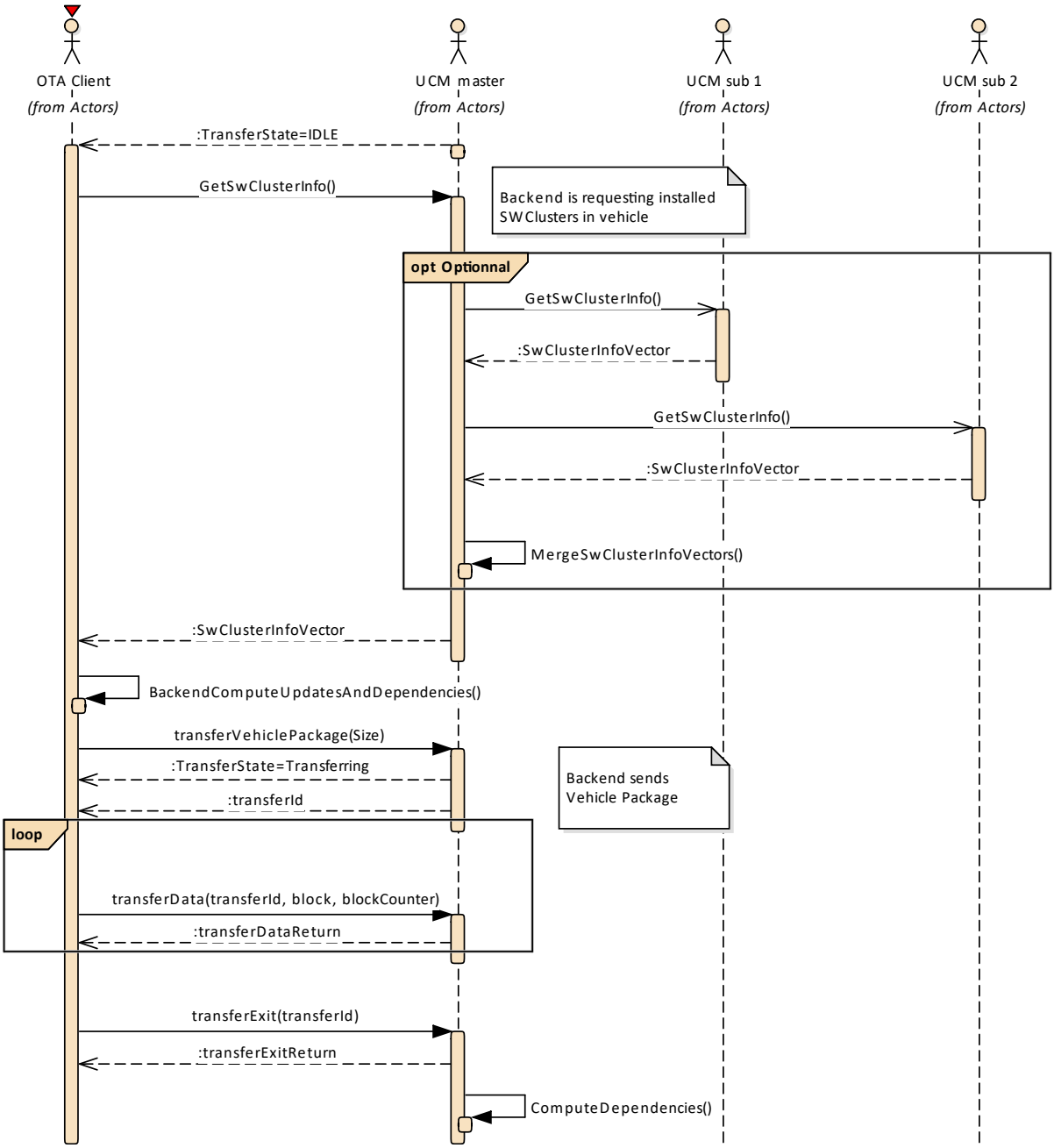

Figure C.3: Push package from backend

C.3 Packages transfer from backend into targeted UCM

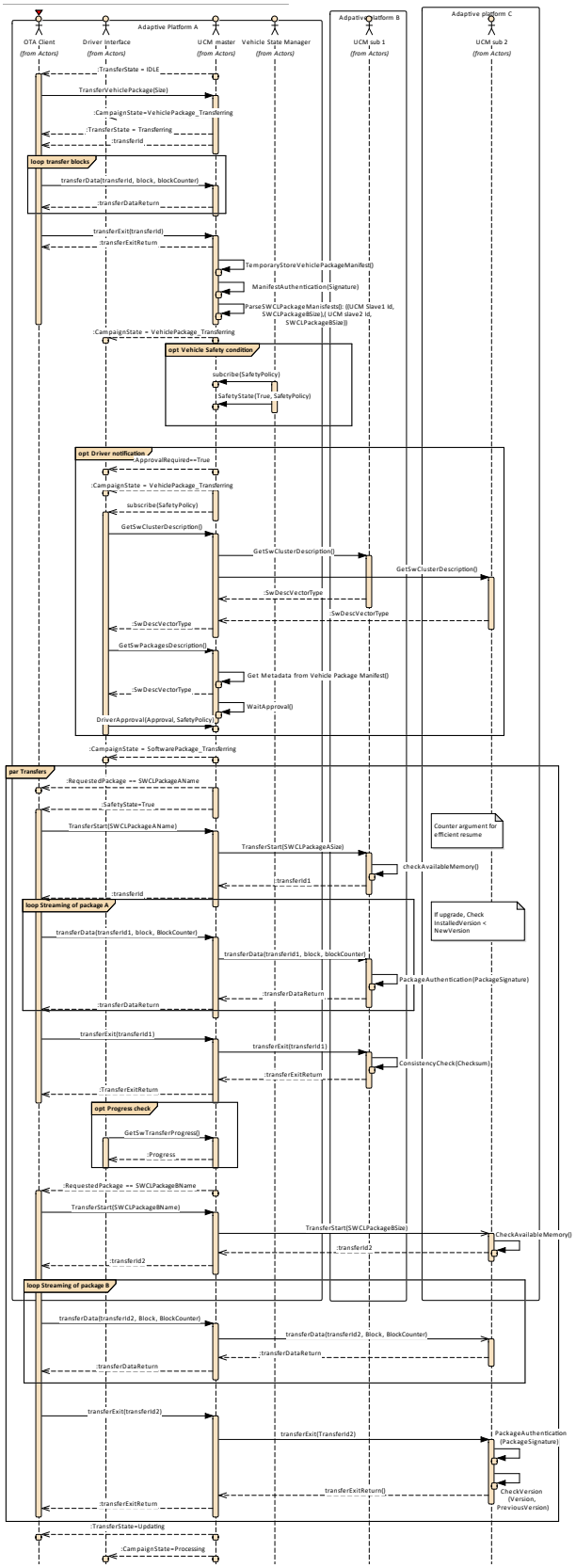

Figure C.4: Stream packages blocks from backend into targeted UCM

C.4 Package processing

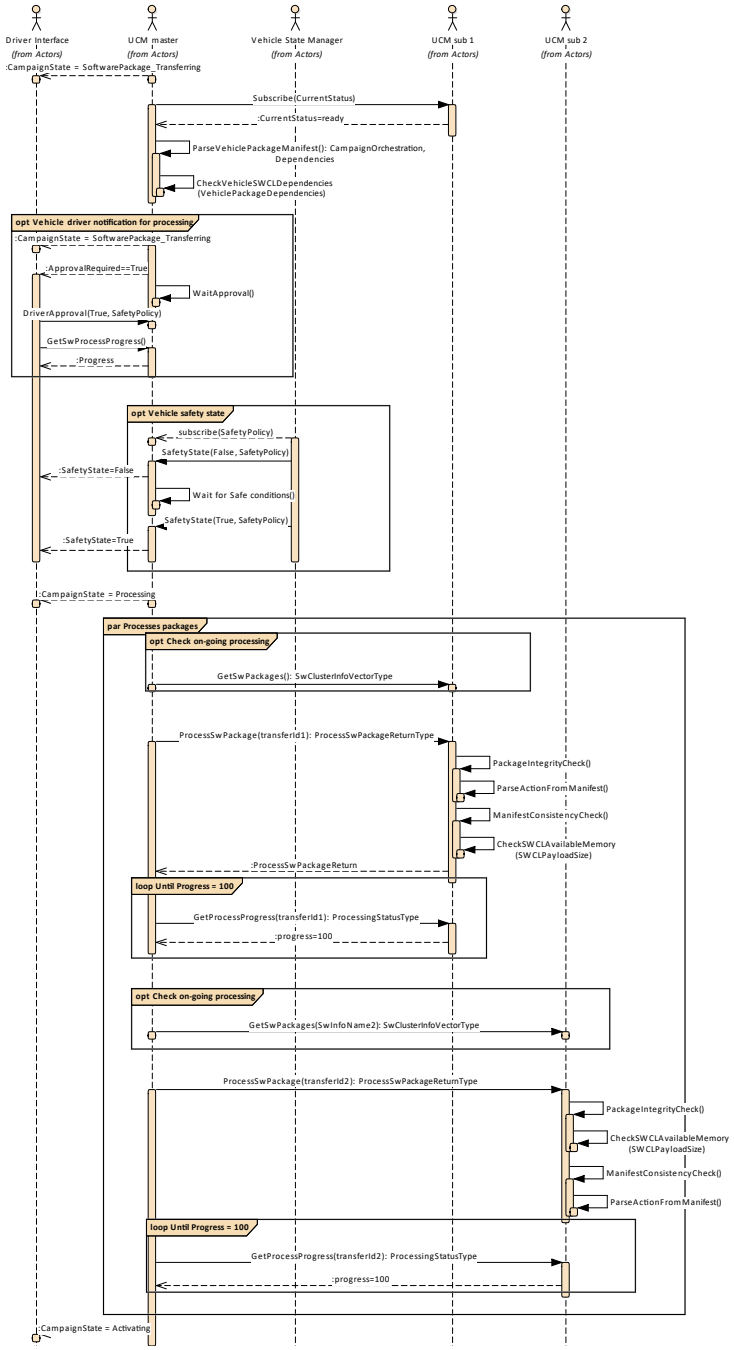

Figure C.5: Packages processing by UCMs

C.5 Package activation

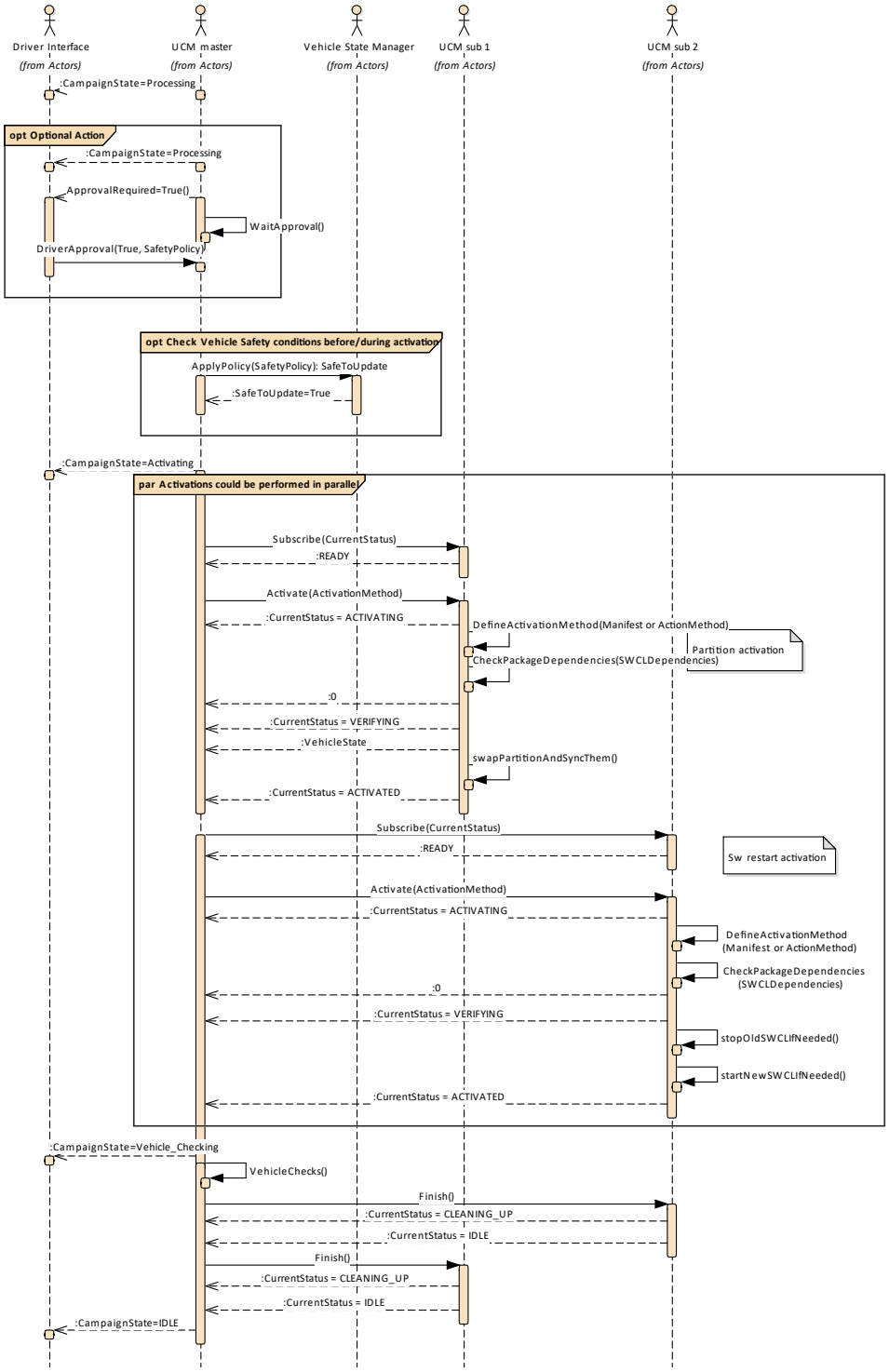

Figure C.6: Packages activation by UCMs

C.6 Package rollback

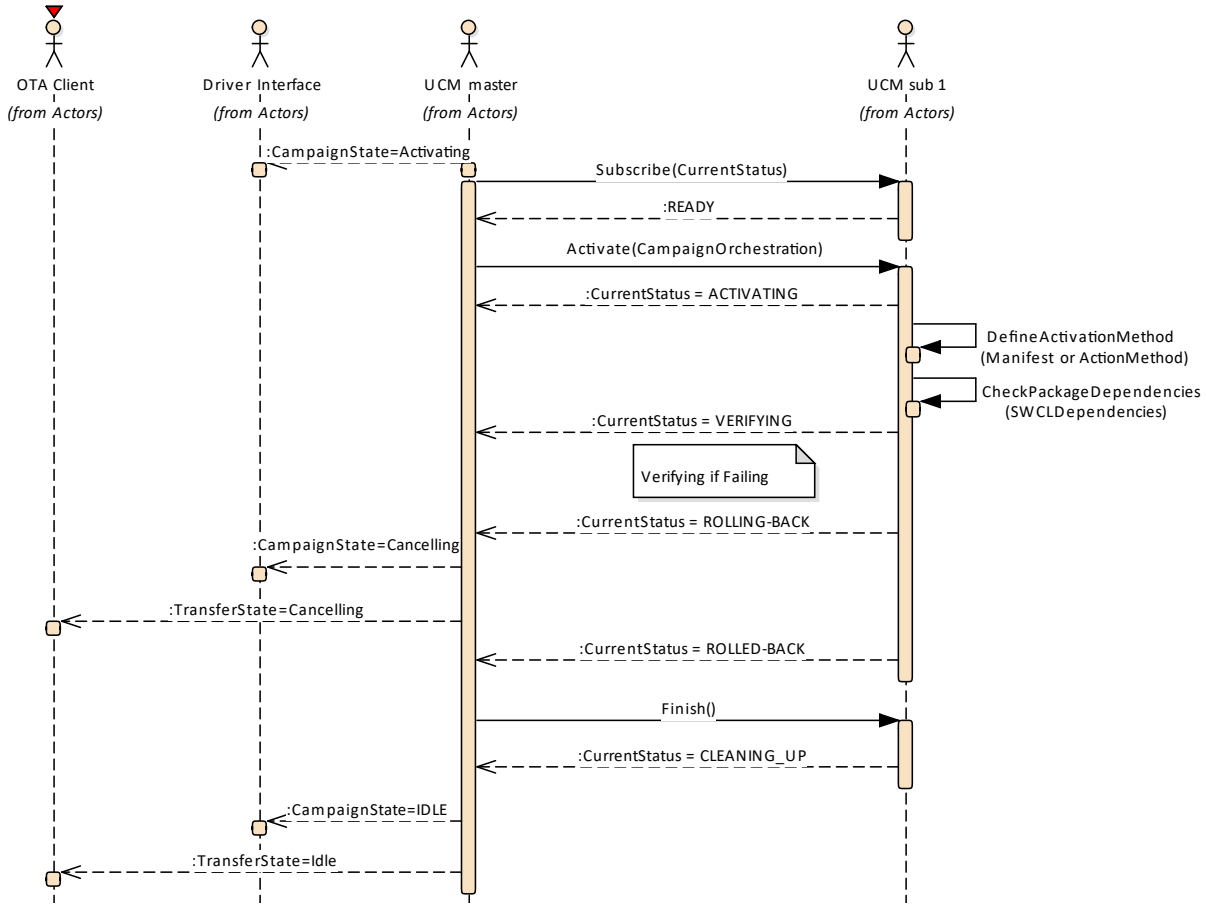

Figure C.7: Packages rollback by UCMs

C.7 Campaign reporting

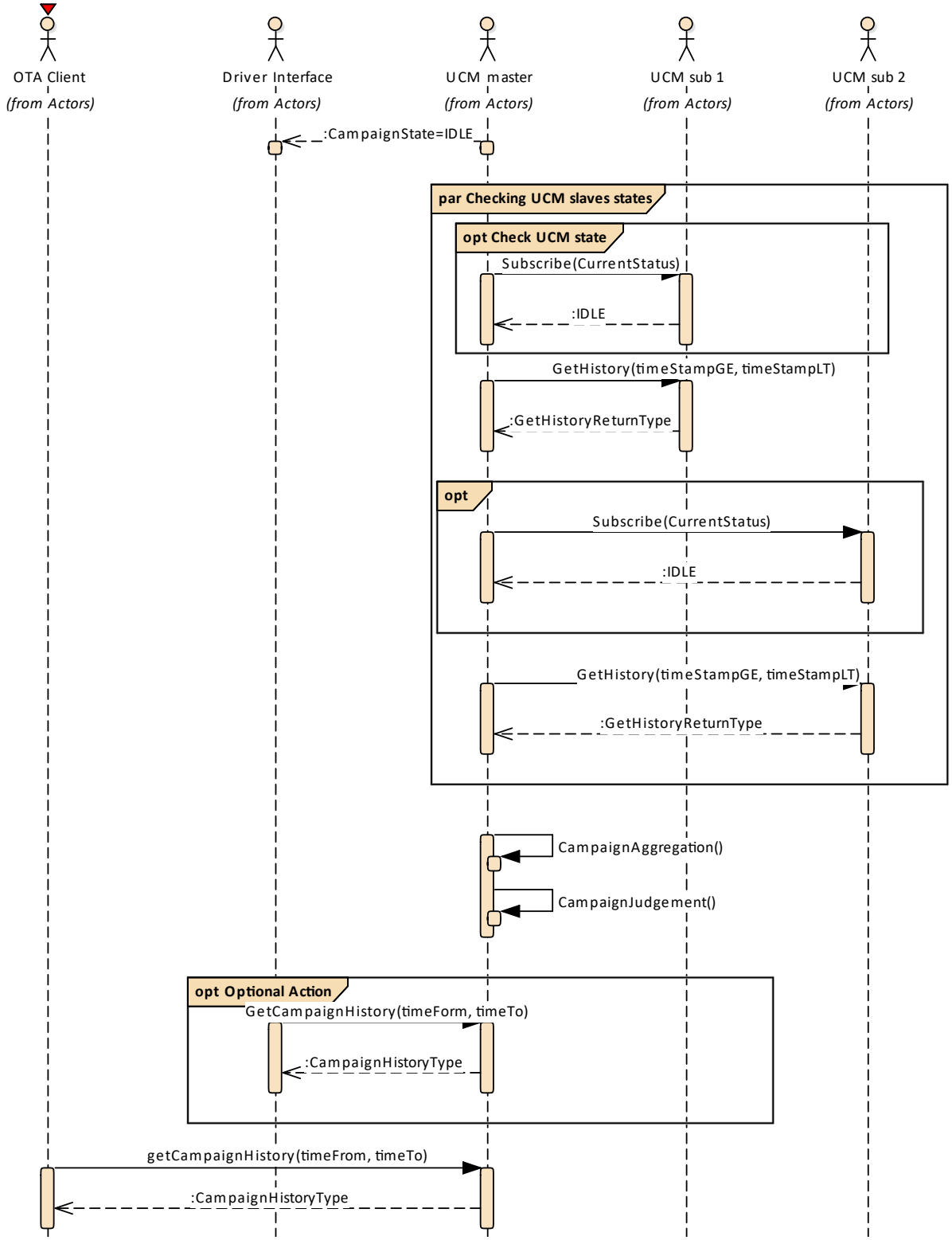

Figure C.8: Campaign reporting to backend

D 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:

D.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.

D.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, 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.

D.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.

D.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.

D.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.

E History of Constraints and Specification Items

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

E.1 Constraint and Specification Item History of this document according to AUTOSAR Release R19-11.

E.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
[SWS_UCM_00210]	Transferring of software packages on `kProcessApproving` or `kProcessing` state

△

Number	Heading
[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 UCMs in vehicle
[SWS_UCM_01006]	Vehicle Package transfer to UCM Master
[SWS_UCM_01007]	Start transfer of a Vehicle Package or Software Packageto 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
[SWS_UCM_01211]	Trigger on kActivateApproving state
[SWS_UCM_01212]	Trigger on kActivating state

▽

△

Number	Heading
[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
[SWS_UCM_01247]	Method to read History Report
[SWS_UCM_01248]	Content of History Report

▽

△

Number	Heading
[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 E.1: Added Traceables in R19-11

E.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 E.2: Changed Traceables in R19-11

E.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 E.3: Deleted Traceables in R19-11

E.1.4 Added Constraints in R19-11

none

E.1.5 Changed Constraints in R19-11

none

E.1.6 Deleted Constraints in R19-11

none