

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

Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	18-10

Document Change History			
Date	Release	Changed by	Description
2018-10-31	18-10	AUTOSAR Release Management	<ul style="list-style-type: none"> • Updated interaction other functional clusters like PER and EMO/SM • Introduction of vehicle package distribution
2018-03-29	18-03	AUTOSAR Release Management	<ul style="list-style-type: none"> • Extended and updated service interface • Introduction of Software Package • Introduction to securing update process
2017-10-27	17-10	AUTOSAR Release Management	<ul style="list-style-type: none"> • Initial release

Disclaimer

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1	Introduction and functional overview	6
2	Acronyms and abbreviations	7
3	Related documentation	8
3.1	Input documents & related standards and norms	8
3.2	Related specification	8
3.3	Further applicable specification	8
4	Constraints and assumptions	9
4.1	Limitations	9
4.2	Applicability to car domains	9
5	Dependencies to other functional clusters	10
5.1	Interfaces to Adaptive State Management	10
5.2	UCM service over ara::com	10
5.3	Interfaces to Adaptive Crypto Interface	10
5.4	Interfaces to Identity and Access Management	10
6	Requirements Tracing	11
7	Functional specification	15
7.1	Technical Overview	15
7.1.1	Software Package Management	16
7.1.1.1	Software Package	16
7.1.1.2	Content of a Software Package	17
7.1.1.3	Applications Persisted Data	17
7.1.2	Runtime dependencies	18
7.1.3	Update scope and state management	18
7.2	Transferring Software Packages	19
7.3	Processing Software Packages	21
7.4	Status Reporting	23
7.5	Activation and Rollback	25
7.5.1	Activation	26
7.5.2	Rollback	27
7.5.3	Boot options	27
7.5.4	Failing Activation	28
7.5.5	Boot monitoring	28
7.5.6	Finishing activation	28
7.6	Logging and history	29
7.7	Version Reporting	29
7.8	SoftwareCluster lifecycle	29
7.9	Securing Software Updates	30
8	API specification	31

9	Service Interfaces	32
9.1	Type definitions	32
9.1.1	TransferIdType	32
9.1.2	SwInfoName	32
9.1.3	ByteVectorType	32
9.1.4	SwPackageStateType	33
9.1.5	SwPackageInfoType	33
9.1.6	SwPackageInfoVectorType	33
9.1.7	SwClusterStateType	34
9.1.8	SwClusterInfoType	34
9.1.9	SwClusterInfoVectorType	35
9.1.10	LogLevelType	35
9.1.11	LogEntryType	35
9.1.12	LogVectorType	36
9.1.13	PackageManagerStatusType	36
9.1.14	ActivateOptionType	37
9.1.15	ActionType	37
9.1.16	ResolutionType	37
9.1.17	GetHistoryType	38
9.1.18	GetHistoryVectorType	38
9.2	Service Interfaces	39
9.2.1	Provided Service Interfaces	39
9.2.1.1	Package Management	39
9.3	Application Errors	46
9.3.1	Application Error Domain	46
9.3.1.1	UCMErrorDomain	46
10	Sequence diagrams	48
10.1	Update process	48
10.2	Data transmission	49
10.3	Package processing	50
10.4	Activation	51
A	Not applicable requirements	52
B	Mentioned Class Tables	53
C	Interfaces to other Functional Clusters (informative)	57
C.1	Overview	57
C.2	Interfaces Tables	57
C.2.1	UCM update notification	57
D	Packages distribution within vehicle (informative)	58
D.1	Overview	58
D.2	Packages distribution sequence diagrams	59
D.2.1	UCM slave discovery	59
D.2.2	Collect information of present SWCLs in vehicle	60

D.2.3	Action computation	61
D.2.3.1	Pull package from backend into vehicle	61
D.2.3.2	Push package from backend into vehicle	63
D.2.4	Packages transfer from backend into targeted UCM	63
D.2.5	Package processing	65
D.2.6	Package activation	66
D.2.7	Package rollback	67
D.2.8	Campaign reporting	67

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

The definition of all technical terms used throughout this document, that are not included in the official [1] AUTOSAR Glossary or [2] TPS Manifest Specification.

Term	Description
Application Error	Errors returned by UCM
Boot options	Boot Manager Configuration

Table 2.1: Technical Terms

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
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 [3] AUTOSAR Execution Management
kRunning	see [3] AUTOSAR Execution Management

Table 2.2: Reference to Technical Terms

3 Related documentation

3.1 Input documents & related standards and norms

- [1] Glossary
AUTOSAR_TR_Glossary
- [2] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification
- [3] Specification of Execution Management
AUTOSAR_SWS_ExecutionManagement
- [4] General Specification of Adaptive Platform
AUTOSAR_SWS_General
- [5] Specification of State Management
AUTOSAR_SWS_StateManagement
- [6] Specification of Communication Management
AUTOSAR_SWS_CommunicationManagement
- [7] Specification of Identity and Access Management
AUTOSAR_SWS_IdentityAndAccessManagement
- [8] Requirements on Update and Configuration Management
AUTOSAR_RS_UpdateAndConfigManagement
- [9] Explanation of Adaptive Platform Design
AUTOSAR_EXP_PlatformDesign
- [10] Specification of Persistency
AUTOSAR_SWS_Persistency
- [11] Requirements on Security Management for Adaptive Platform
AUTOSAR_RS_SecurityManagement

3.2 Related specification

See chapter [3.1](#).

3.3 Further applicable specification

AUTOSAR provides a general specification [4] which is also applicable for [UCM](#). The specification SWS General shall be considered as additional and required specification for implementation of [UCM](#).

4 Constraints and assumptions

4.1 Limitations

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

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

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

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

4.2 Applicability to car domains

No restrictions to applicability.

5 Dependencies to other functional clusters

The [UCM](#) functional cluster exposes services to client applications via the `ara::com` middleware.

5.1 Interfaces to Adaptive State Management

Certain applications can conflict with the update process or the newly updated package, and they need to be stopped during the update process. This could be achieved by putting the machine to a safe `Machine State`, for example `Update State`, or by activating a combination of suitable `Function Groups` and its states. It is the responsibility of the platform integrator to define this state or `Function Groups`. The application accessing the [UCM](#), should make sure that the platform is switched to this state (using interfaces from State Management [5]), before starting the update.

[UCM](#) uses State Management interface field parameter `FunctionGroupState` to monitor the restart of the updated software.

5.2 UCM service over `ara::com`

The [UCM](#) shall provide a service interface over `ara::com` using methods and fields.

5.3 Interfaces to Adaptive Crypto Interface

[UCM](#) uses Crypto Interface for Adaptive Platform [?,] to verify package integrity and authenticity and to decrypt confidential update data.

5.4 Interfaces to Identity and Access Management

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

6 Requirements Tracing

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

Requirement	Description	Satisfied by
[RS_SM_00001]	State Management shall support Function Group state change requests.	[SWS_UCM_00102] [SWS_UCM_00124]
[RS_UCM_00001]	UCM shall support installing new software on AUTOSAR Adaptive Platform	[SWS_UCM_00001] [SWS_UCM_00017] [SWS_UCM_00073] [SWS_UCM_00099] [SWS_UCM_00131] [SWS_UCM_00137]
[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]
[RS_UCM_00003]	UCM shall support updating installed software on Adaptive Platform	[SWS_UCM_00017]
[RS_UCM_00004]	UCM shall support uninstalling software on AUTOSAR Adaptive Platform	[SWS_UCM_00001] [SWS_UCM_00137]
[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]
[RS_UCM_00007]	UCM shall check that software dependencies are fulfilled	[SWS_UCM_00026] [SWS_UCM_00120] [SWS_UCM_00128] [SWS_UCM_00136]

Requirement	Description	Satisfied by
[RS_UCM_00008]	UCM shall support a recovery mechanism in case of failed update process	[SWS_UCM_00005] [SWS_UCM_00024] [SWS_UCM_00095] [SWS_UCM_00096] [SWS_UCM_00097] [SWS_UCM_00107] [SWS_UCM_00109] [SWS_UCM_00110] [SWS_UCM_00111] [SWS_UCM_00113] [SWS_UCM_00126] [SWS_UCM_00127] [SWS_UCM_00131] [SWS_UCM_00142] [SWS_UCM_00146]
[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]
[RS_UCM_00011]	UCM shall support reporting software versions which have been installed and will be activated when new versions are activated	[SWS_UCM_00027] [SWS_UCM_00030] [SWS_UCM_00038] [SWS_UCM_00039] [SWS_UCM_00040] [SWS_UCM_00077] [SWS_UCM_00078] [SWS_UCM_00079] [SWS_UCM_00131]
[RS_UCM_00012]	UCM shall check the consistency of transferred Software Package	[SWS_UCM_00029] [SWS_UCM_00038] [SWS_UCM_00039] [SWS_UCM_00040] [SWS_UCM_00077] [SWS_UCM_00078] [SWS_UCM_00079] [SWS_UCM_00104] [SWS_UCM_00136]
[RS_UCM_00013]	UCM shall check that it has enough resources to receive, process and store the Software Package and associated data	[SWS_UCM_00007] [SWS_UCM_00008] [SWS_UCM_00010] [SWS_UCM_00087] [SWS_UCM_00088] [SWS_UCM_00091] [SWS_UCM_00092] [SWS_UCM_00098] [SWS_UCM_00136] [SWS_UCM_00140] [SWS_UCM_00145]

Requirement	Description	Satisfied by
[RS_UCM_00014]	UCM shall check that correct amount of data has been transferred for the Software Package	[SWS_UCM_00136]
[RS_UCM_00015]	UCM shall remove all unneeded data after Software Package processing has finished	[SWS_UCM_00020] [SWS_UCM_00131]
[RS_UCM_00017]	UCM shall support installing and updating the persistent data storage for an Adaptive Application	[SWS_UCM_00011] [SWS_UCM_00113]
[RS_UCM_00018]	UCM shall announce when an application has been installed, updated or uninstalled	[SWS_UCM_00021] [SWS_UCM_00131]
[RS_UCM_00019]	UCM shall support simultaneous transfers multiple update packages	[SWS_UCM_00007] [SWS_UCM_00008] [SWS_UCM_00010] [SWS_UCM_00031] [SWS_UCM_00075] [SWS_UCM_00087] [SWS_UCM_00088] [SWS_UCM_00091] [SWS_UCM_00092] [SWS_UCM_00093] [SWS_UCM_00098] [SWS_UCM_00140] [SWS_UCM_00141] [SWS_UCM_00145]
[RS_UCM_00020]	UCM shall support cancel of an update or install operation	[SWS_UCM_00003]
[RS_UCM_00021]	UCM shall support atomic activation of installed or updated packages	[SWS_UCM_00022] [SWS_UCM_00025] [SWS_UCM_00094] [SWS_UCM_00114] [SWS_UCM_00131]
[RS_UCM_00022]	UCM shall support logging of the update or installation process	[SWS_UCM_00012] [SWS_UCM_00041] [SWS_UCM_00042] [SWS_UCM_00043] [SWS_UCM_00131] [SWS_UCM_00143] [SWS_UCM_00144]
[RS_UCM_00023]	UCM shall provide an interface to read progress of the update	[SWS_UCM_00018] [SWS_UCM_00131]
[RS_UCM_00024]	UCM shall provide an interface to read the state of UCM	[SWS_UCM_00019] [SWS_UCM_00044] [SWS_UCM_00080] [SWS_UCM_00081] [SWS_UCM_00082] [SWS_UCM_00083] [SWS_UCM_00084] [SWS_UCM_00085] [SWS_UCM_00086] [SWS_UCM_00131]

Requirement	Description	Satisfied by
[RS_UCM_00025]	UCM shall support efficient streaming of Software Package data	[SWS_UCM_00007] [SWS_UCM_00008] [SWS_UCM_00010] [SWS_UCM_00031] [SWS_UCM_00032] [SWS_UCM_00087] [SWS_UCM_00088] [SWS_UCM_00091] [SWS_UCM_00092] [SWS_UCM_00098] [SWS_UCM_00131] [SWS_UCM_00140] [SWS_UCM_00145]
[RS_UCM_00026]	UCM shall process installation of new Software Packages, updates and removal of existing Software Packages sequentially	[SWS_UCM_00017] [SWS_UCM_00044] [SWS_UCM_00122]
[RS_UCM_00028]	UCM shall support updating Functional Clusters	[SWS_UCM_00100]
[RS_UCM_00029]	UCM shall support updating the underlying Operating System	[SWS_UCM_00101]
[RS_UCM_00030]	UCM shall be able to verify the updated software before activation	[SWS_UCM_00095] [SWS_UCM_00096] [SWS_UCM_00097] [SWS_UCM_00107] [SWS_UCM_00108] [SWS_UCM_00109] [SWS_UCM_00111] [SWS_UCM_00126] [SWS_UCM_00127] [SWS_UCM_00146]
[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 actions history	[SWS_UCM_00115] [SWS_UCM_00131] [SWS_UCM_00132] [SWS_UCM_00133] [SWS_UCM_00134] [SWS_UCM_00135]

7 Functional specification

7.1 Technical Overview

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

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

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

To illustrate the diagnostic use-case, Figure 7.1 shows a typical architecture with a Diagnostic Application establishing the link between **UCM DM** [?,]. The Diagnostic Client downloads a Software Package by standard UDS services. After decompression and verification steps inside the OEM-specific Diagnostic Application, the Software Package is passed to **UCM** and processed. The result of this package processing can be made available/readable for diagnostics.

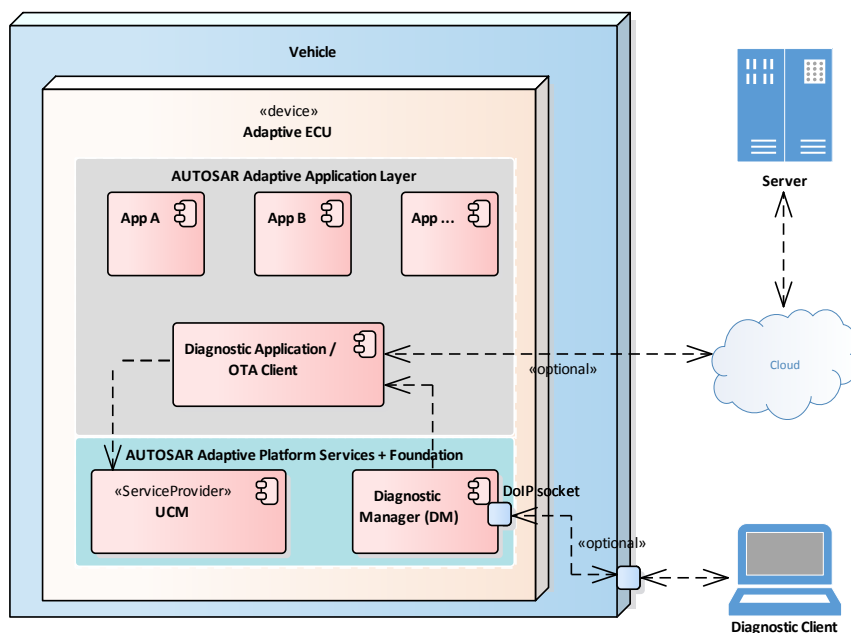


Figure 7.1: Architecture overview for diagnostic use case

7.1.1 Software Package Management

The UCM update sequence consists three different phases:

- Software Package transfer: A phase in which, one or several Software Packages are transferred from the Diagnostic Application Client to the internal buffer of the UCM. For further information see chapter 7.2.
- Software Package processing: A phase in which the UCM performs the operation (kInstall, kUpdate, kRemove) on the relevant SoftwareCluster (which shall be inactivated [not executed] during the operation). For further information see chapter 7.3.
- 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 [5]) prior to finishing the update. For further information see chapter 7.5

7.1.1.1 Software Package

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

A SoftwareCluster can act in two roles:

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

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

A software package has to be modelled as a so-called SoftwareCluster which describes the content of a software package that has to be uploaded to the AUTOSAR Adaptive Platform, see [2].

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 shall be determined by references to all required model elements. The SoftwareCluster and the related model elements shall be put into a 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 [9].

7.1.1.2 Content of a Software Package

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

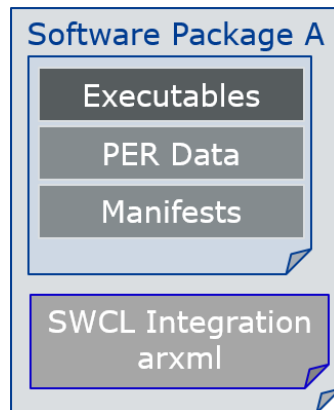


Figure 7.2: Software package content description

A single `SoftwarePackage` 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 [9].

[SWS_UCM_00112] Software Cluster and version [`SoftwareCluster`'s manifest shall include a name and a version following semantic versioning 2.0.0 (<https://semver.org/>). A time stamp shall be trailing the Major.Minor.Patch version.] ([RS_UCM_00002](#))

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

7.1.1.3 Applications Persisted Data

[SWS_UCM_00011] Updating persisted data [The UCM shall be able to create, update or remove any persistency data that is contained in the `SoftwareCluster`.] ([RS_UCM_00017](#))

Further details on the persistent data can be found in Persistency Specification [10].

[SWS_UCM_00113] Rollback of persisted data [The UCM shall be able to rollback changes done to persistent data during update process.] ([RS_UCM_00017](#), [RS_UCM_00008](#))

7.1.2 Runtime dependencies

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

Dependencies are described in the `SoftwareCluster` metamodel, see [2].

[SWS_UCM_00120] Execution dependencies check [UCM shall check execution dependencies in the end of the update process before activating/switching to the new software version.]([RS_UCM_00007](#))

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

[SWS_UCM_00128] [If dependency check fails, UCM shall raise the `Application-ErrorMissingDependencies` and change its state from `kActivating` to `kReady`.]([RS_UCM_00007](#))

7.1.3 Update scope and state management

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

[SWS_UCM_00099] Update of Adaptive Application [UCM shall be able to update Adaptive Applications]([RS_UCM_00001](#))

[SWS_UCM_00100] Update of Functional Clusters [UCM shall be able to update all Functional Clusters, including UCM itself.]([RS_UCM_00028](#))

[SWS_UCM_00101] Update of Host [UCM shall be able to update the underlying Operating System hosting the `AUTOSAR Adaptive Platform`.]([RS_UCM_00029](#))

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

[SWS_UCM_00102] Update state [For the updates of software components included into Machine State Function Group, UCM shall check that system is set to **update** state.]([RS_SM_00001](#))

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

It is the responsibility of the OEM Diagnostic Application to request the transition to **update** state, using suitable interfaces of Adaptive State Management [5].

[SWS_UCM_00124] Verify State [As minimal check UCM shall check that updated software is able to reach `kRunning` state. For checking if the updated system can reach the `kRunning` state, the system shall be set into **verify** state.](*RS_SM_00001*)

It is the responsibility of the OEM Diagnostic Application to request the transition to **verify** state, using suitable interfaces of Adaptive State Management [5]. Then, State Management [5] will return a successful state change only if all the relevant processes have reached the `kRunning` state. This gives a chance to perform a Rollback if some processes fails to reach the `kRunning` state.

Update of some components require a Machine reset to be performed. These components should be configured to be part of Machine State function group, as the update sequence of Machine State function group includes a Machine reset. Execution Manager, State Manager, Communication Manager and UCM itself are good examples which probably require a Machine reset to activate the update. Other such components could be applications involved in the update sequence or applications involved in safety monitoring. Further details on Machine State function group can be found in State Management [5].

7.2 Transferring Software Packages

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

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

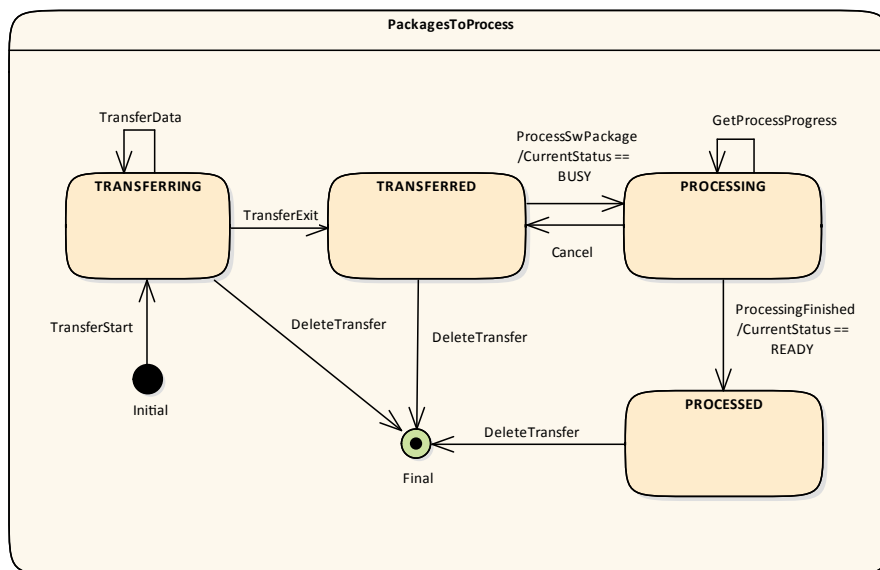


Figure 7.3: State Machine for transferring packages using service interface `PackageManagement`

[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 a `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 `ProcessSwPackage`.]([RS_UCM_00013](#), [RS_UCM_00019](#), [RS_UCM_00025](#))

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

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

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

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

[SWS_UCM_00092] Package manifest consistency [During `TransferExit` UCM shall raise the `ApplicationError InvalidManifest` if the manifest integrity check fails]([RS_UCM_00013](#), [RS_UCM_00019](#), [RS_UCM_00025](#))

[SWS_UCM_00028] Package Authentication [UCM shall authenticate the Software Package.]([RS_UCM_00006](#))

Content of the manifest shall contain signatures of the Software Package and `TransferExit` shall use hash algorithms and cryptographic signatures to validate the package authenticity as defined in [11].

[SWS_UCM_00098] Package Authentication failure [During `TransferExit` UCM shall raise the `ApplicationError AuthenticationFailed`, if the data authentication check fails.]([RS_UCM_00013](#), [RS_UCM_00019](#), [RS_UCM_00025](#))

[SWS_UCM_00091] Successful data transfer [During `TransferExit` UCM shall not raise any `ApplicationError` if the transfer of data could be successfully finished.]
([RS_UCM_00013](#), [RS_UCM_00019](#), [RS_UCM_00025](#))

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

[SWS_UCM_00141] UCM insufficient memory for parallel data transfer [While a software package is being transferred, if UCM receives a subsequent `TransferStart` call targeting another software package, UCM shall make sure that the sum of the size of both software packages (the one being transferred and the one requested to be transferred) does not exceed the size of the UCM buffer. Otherwise, the `TransferStart` shall raise the `ApplicationError InsufficientMemory` and the newly requested transmission shall be rejected.]([RS_UCM_00019](#))

If UCM provides 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. 7.3.

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

[SWS_UCM_00093] Transfer sequence [If `TransferData` or `TransferExit` is called without prior call to `TransferStart` UCM shall raise the `ApplicationError OperationNotPermitted`.]([RS_UCM_00019](#))

[SWS_UCM_00069] Report information on Software Packages [UCM shall provide a method `GetSwPackages` of the interface service `PackageManagement` to provide the identifiers, names and versions of Software Packages of any state.]
([RS_UCM_00010](#))

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

7.3 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 shall be deleted. Nevertheless, the communication sequence between the triggering application of the software modification and UCM shall be 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 by calling method `ProcessSwPackage` several times in sequence.]([RS_UCM_00001](#), [RS_UCM_00004](#), [RS_UCM_00005](#))

During package processing, the progress is provided.

[SWS_UCM_00018] Providing Progress Information [UCM shall provide a method `GetSwProcessProgress` to query the package processing progress. Parameter `progress` shall be set to a value representing the progress between 0% and 100% (0x00 ... 0x64).]([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 InvalidManifest`.]([RS_UCM_00012](#))

[SWS_UCM_00104] Consistency Check of Package [UCM shall raise the `ApplicationError PackageInconsistent` in case the package to be processed is not consistent.]([RS_UCM_00012](#))

[SWS_UCM_00003] Cancelling the package processing [UCM shall provide a method `Cancel` to cancel the running package processing. UCM shall then immediately abort the current package processing task, undo any changes and free any reserved resources.]([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](#))

Depending on the capabilities of UCM and of the updated target, `Cancel` and `RevertProcessedSwPackages` shall 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.

7.4 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 7.3 and 7.5.

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

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

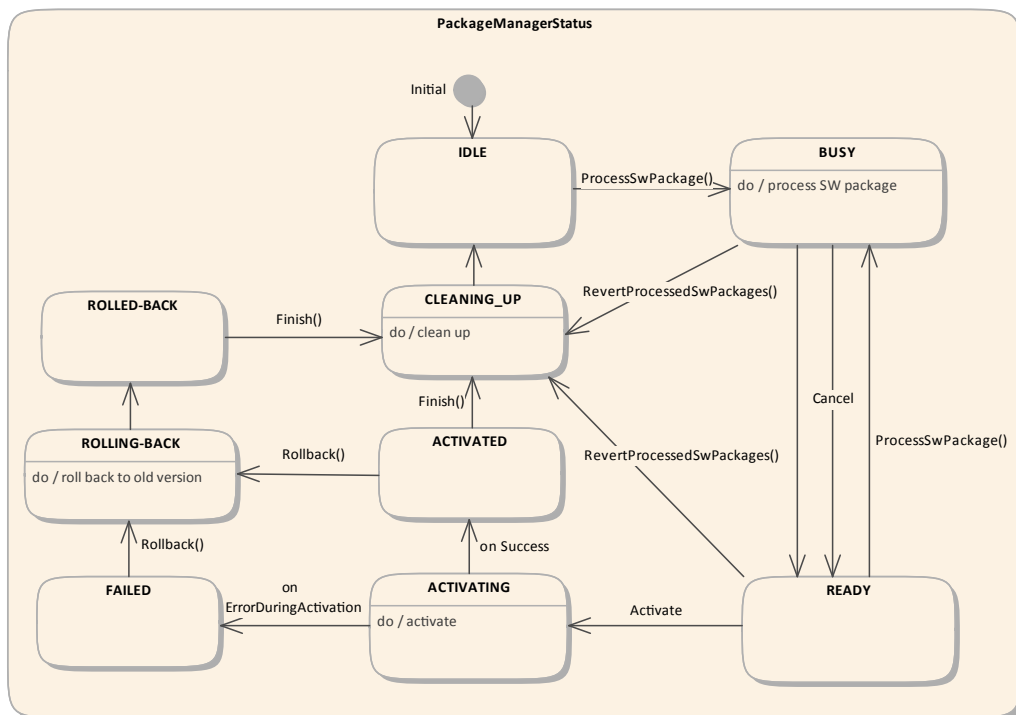


Figure 7.4: State Machine for the package processing using service interface: Package-Management

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

[SWS_UCM_00086] 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. Once `ProcessSwPackage` is performed successfully, this state shall only be entered if a Clean-up has been performed successfully.]([RS_UCM_00024](#))

[SWS_UCM_00081] Busy state of Package Management [`kBusy` 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 GeneralReject` as long as `CurrentStatus` is different than `kIdle` or `kReady`.]([RS_UCM_00001](#), [RS_UCM_00003](#), [RS_UCM_00026](#))

[SWS_UCM_00082] Exit from Busy state of Package Management [`kBusy` state shall be exited when processing of called method `ProcessSwPackage` or `RevertProcessedSwPackages` has finished or when the processing of the package has been interrupted by calling `Cancel`. Following state reported by `CurrentStatus` is `kCleaning-up` in case of a `RevertProcessedSwPackages` call or `kReady` in case of a `ProcessSwPackage` completion or in case of a `Cancel` call.]([RS_UCM_00024](#))

[SWS_UCM_00083] Ready state of Package Management [`kReady` state shall be reported

- after the package processing is finished successfully or
- `Cancel` has been performed.
- when `Activate` fails due to an `ApplicationError MissingDependencies`.

]([RS_UCM_00024](#))

[SWS_UCM_00084] Activating state of Package Management [`kActivating` shall be set when `Activate` is called. This prepares the updated software to be executed in the next restart of the system or Function Group.]([RS_UCM_00024](#))

After `Activate` was performed successfully (all dependencies are satisfied), the system has to be restarted in case a A/B partition is used. In case the A/B partition is not used, all affected Function Groups or the platform could be restarted. Immediately after the processed Software Cluster has been restarted, a system check has to be performed in order to make sure the system is able to start up as expected. With this check it is verified that other safety relevant software like Functional Cluster Platform Health Manager is running and user can be protected from any issues caused by the update after the update has finished.

[SWS_UCM_00085] Activated state of Package Management [`kActivated` state shall be set when system or all impacted Function Groups have been successfully restarted into `verify` Function Group state.]([RS_UCM_00024](#))

UCM monitors `FunctionGroupStates` from State Management [5] to conclude if activation was successful. `kActivated` state gives the client controlling the update process a chance to perform verification test, though functionality in verification 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 [5], then UCM goes to `kRolledBack` state. In this state all the changes introduced during update process have been deactivated and can be cleaned by calling [Finish](#).

[SWS_UCM_00126] Entering the RollingBack state [`kRollingBack` shall be set when [Rollback](#) is called. This prepares the original software to be executed in the next restart of the system or Function Group.]([RS_UCM_00008](#), [RS_UCM_00030](#))

[SWS_UCM_00111] Entering the Rolled-back state [UCM shall switch the state into `kRolledBack` when State Management `FunctionGroupState` field indicates that all the software updated have been restarted or shutdown.]([RS_UCM_00008](#), [RS_UCM_00030](#))

[SWS_UCM_00146] Entering the Cleaning-up state [UCM shall switch the state into `kCleaning-up` state when [Finish](#) or [RevertProcessedSwPackages](#) is called. At that point of the sequence, UCM shall clean up all temporary data of the processed packages (e.g. remove the "physical" software package [e.g. zip file] 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](#))

7.5 Activation and Rollback

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

[SWS_UCM_00027] Notification of Activation or Rollback [UCM shall notify the activation or rollback of software packages to other Functional Clusters of the AUTOSAR Adaptive Platform.]([RS_UCM_00011](#))

Vendor specific solution dictates to which modules this information shall be made available, in which form and if this is done directly when change is done or when change is executed.

7.5.1 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 executed.]([RS_UCM_00008](#), [RS_UCM_00030](#))

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

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

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

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

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

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

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

[SWS_UCM_00109] Entering the Activated state [UCM shall switch from `kActivating` state to `kActivated` when the State Management [5] `FunctionGroupState` field notifies that the updated software is executed successfully i.e. when **verify** state is reported by State Management [5].]([RS_UCM_00008](#), [RS_UCM_00030](#))

7.5.2 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 [During Rolling-Back UCM shall disable the changes done by the software update.]([RS_UCM_00008](#))

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

7.5.3 Boot options

During update process the executed software is switched from original software to updated software and in case of rollback back, 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 [3] might be started before UCM. In this case there can't be direct interface between UCM and Execution Management [3] 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 [3] to launch.]([RS_UCM_00021](#))

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

[SWS_UCM_00095] Entering the Activated state [UCM shall switch from `kActivating` state into the `kActivated` state when `FunctionGroupState` from State Management [5] indicates that updated software has successfully reached the Application state `kRunning` (i.e. when State Management report that the `FunctionGroup` that is mapped to the updated Software has reached the **verify** state).]([RS_UCM_00008](#), [RS_UCM_00030](#))

[SWS_UCM_00096] Entering the Rolled-back state [UCM shall switch from `kRollingBack` state into the `kRolledBack` state when `FunctionGroupState` from State Management [5] indicates that original software has successfully reached the Application state `kRunning`.]([RS_UCM_00008](#), [RS_UCM_00030](#))

7.5.4 Failing Activation

[SWS_UCM_00097] Entering the Failed state [UCM shall switch from `kActivating` state to `kFailed` state when `FunctionGroupState` from State Management [5] indicates that updated software has not successfully reached the Running state (i.e. when State Management reports that the `FunctionGroup` that is mapped to the updated Software has not been able to reach the Verify state because one [or more] process of the updated software has not successfully reach the Process state `kRunning`).]([RS_UCM_00008](#), [RS_UCM_00030](#))

7.5.5 Boot monitoring

Activation failure during OS and Platform-self updates can lead to state that system isn't able to reach point where UCM and the client are able to function as expected and are able to execute the rollback. For these cases 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.

7.5.6 Finishing activation

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

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

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

7.6 Logging and history

[SWS_UCM_00012] Log message retrieving [UCM shall provide a method `GetLog` to retrieve all log messages that have been stored by UCM for specific `id`.] (*RS_UCM_00022*)

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

[SWS_UCM_00144] Log error [`GetLog` and `SetLogLevel` shall be provided only in the context of a `id`, otherwise an `ApplicationError InvalidTransferId` shall be raised.] (*RS_UCM_00022*)

[SWS_UCM_00115] History [UCM shall provide a method `GetHistory` to retrieve all actions that have been performed by UCM.] (*RS_UCM_00032*)

7.7 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 `kActivated`.] (*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*)

7.8 SoftwareCluster lifecycle

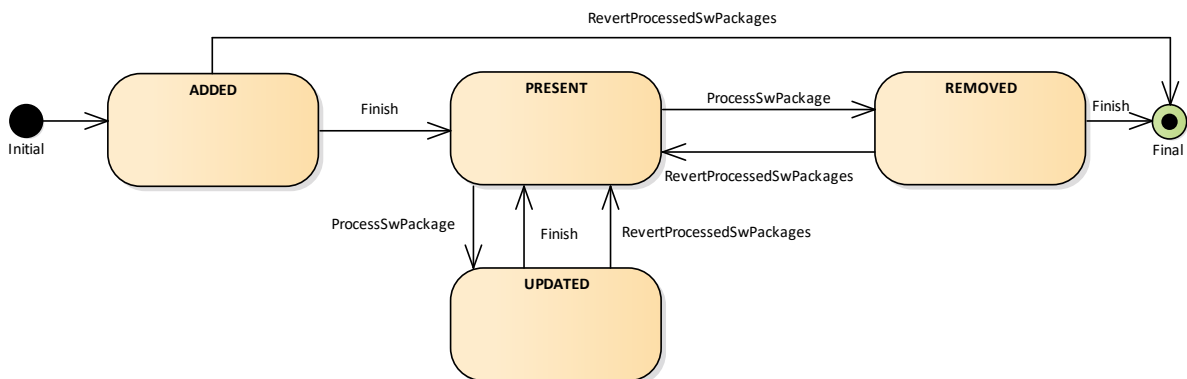


Figure 7.5: State Machine for a `SoftwareCluster`

The state machine in Fig. 7.5 describes the states of a `SoftwareCluster`. After processing a Software Package with a new `SoftwareCluster` that was not yet existing

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

7.9 Securing Software Updates

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

UCM shall authenticate the Software Package. You can refer to 7.2

[SWS_UCM_00103] 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](#))

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 TransferIdType

[SWS_UCM_00031] TransferIdType table [

Name	TransferIdType
Kind	TYPE_REFERENCE
Derived from	uint32_t
Description	Represents a handle identifier used to reference a particular transfer request.

Table 9.1: Implementation Data Type - TransferIdType

]([RS_UCM_00019](#), [RS_UCM_00025](#))

9.1.2 SwInfoName

[SWS_UCM_00071] SwInfoName table [

Name	SwInfoName
Kind	STRING
Derived from	-
Description	SoftwareCluster name.

Table 9.2: Implementation Data Type - SwInfoName

]([RS_UCM_00002](#))

9.1.3 ByteVectorType

[SWS_UCM_00032] ByteVectorType table [

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

Table 9.3: Implementation Data Type - ByteVectorType

]([RS_UCM_00025](#))

9.1.4 SwPackageStateType

[SWS_UCM_00038] SwPackageStateType table [

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

Table 9.4: Implementation Data Type - SwPackageStateType

]([RS_UCM_00002](#), [RS_UCM_00006](#), [RS_UCM_00010](#), [RS_UCM_00011](#),
[RS_UCM_00012](#))

9.1.5 SwPackageInfoType

[SWS_UCM_00039] SwPackageInfoType table [

Name	SwPackageInfoType
Kind	STRUCTURE
Subelements	Name SwInfoName Version StringType TransferID TransferIdType Size uint8_t State SwPackageStateType
Derived from	-
Description	Represents the information of a Software Package.

Table 9.5: Implementation Data Type - SwPackageInfoType

]([RS_UCM_00002](#), [RS_UCM_00006](#), [RS_UCM_00010](#), [RS_UCM_00011](#),
[RS_UCM_00012](#))

9.1.6 SwPackageInfoVectorType

[SWS_UCM_00040] SwPackageInfoVectorType table [

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

Table 9.6: Implementation Data Type - SwPackageInfoVectorType

|(RS_UCM_00002, RS_UCM_00006, RS_UCM_00010, RS_UCM_00011, RS_UCM_00012)

9.1.7 SwClusterStateType

[SWS_UCM_00077] SwClusterStateType table [

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

Table 9.7: Implementation Data Type - SwClusterStateType

|(RS_UCM_00002, RS_UCM_00006, RS_UCM_00010, RS_UCM_00011, RS_UCM_00012)

9.1.8 SwClusterInfoType

[SWS_UCM_00078] SwClusterInfoType table [

Name	SwClusterInfoType
Kind	STRUCTURE
Subelements	Name SwInfoName Version StringType State SwClusterStateType
Derived from	-
Description	Represents the information of a SoftwareCluster.

Table 9.8: Implementation Data Type - SwClusterInfoType

|(RS_UCM_00002, RS_UCM_00006, RS_UCM_00010, RS_UCM_00011, RS_UCM_00012)

9.1.9 SwClusterInfoVectorType

[SWS_UCM_00079] SwClusterInfoVectorType table [

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

Table 9.9: Implementation Data Type - SwClusterInfoVectorType

]([RS_UCM_00002](#), [RS_UCM_00006](#), [RS_UCM_00010](#), [RS_UCM_00011](#),
[RS_UCM_00012](#))

9.1.10 LogLevelType

[SWS_UCM_00041] LogLevelType table [

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

Table 9.10: Implementation Data Type - LogLevelType

]([RS_UCM_00022](#))

9.1.11 LogEntryType

[SWS_UCM_00042] LogEntryType table [

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

Table 9.11: Implementation Data Type - LogEntryType

]([RS_UCM_00022](#))

9.1.12 LogVectorType

[SWS_UCM_00043] LogVectorType table [

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

Table 9.12: Implementation Data Type - LogVectorType

]([RS_UCM_00022](#))

9.1.13 PackageManagerStatusType

[SWS_UCM_00044] PackageManagerStatusType table [

Name	PackageManagerStatusType	
Kind	TYPE_REFERENCE	
Derived from	uint8_t	
Description	Represents the state of UCM.	
Range / Symbol	Limit	Description
kIdle	0x00	UCM is ready to start processing if software packages are present.
kReady	0x01	UCM has processed one or several packages and waits for additional packages, activation or reversion of processed packages.
kBusy	0x02	UCM is currently in the middle of processing a Software Package, i.e. a client has called ProcessSwPackage.
kActivating	0x03	UCM is activating the processed 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.
kFailed	0x08	Activation failed.

Table 9.13: Implementation Data Type - PackageManagerStatusType

|(RS_UCM_00024, RS_UCM_00026)

9.1.14 ActivateOptionType

[SWS_UCM_00114] ActivateOptionType table [

Name	ActivateOptionType	
Kind	TYPE_REFERENCE	
Derived from	uint8_t	
Description	Represents the option parameter type of the Activate method.	
Range / Symbol	Limit	Description
kDefault	0x00	No override, default behaviour.
kFunctionGroupRestart	0x01	Request restart of the functional group.
kSystemReset	0x02	Request reset of the system.

Table 9.14: Implementation Data Type - ActivateOptionType

|(RS_UCM_00021)

9.1.15 ActionType

[SWS_UCM_00132] ActionType table [

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

Table 9.15: Implementation Data Type - ActionType

|(RS_UCM_00032)

9.1.16 ResolutionType

[SWS_UCM_00133] ResolutionType table [

Name	ResolutionType	
Kind	TYPE_REFERENCE	
Derived from	uint8_t	
Description	Represents the result of UCM action.	
Range / Symbol	Limit	Description
kSuccessfull	0x00	UCM's action was successful.
kFailed	0x01	UCM's action failed.

Table 9.16: Implementation Data Type - ResolutionType

](RS_UCM_00032)

9.1.17 GetHistoryType

[SWS_UCM_00134] GetHistoryType table [

Name	GetHistoryType
Kind	STRUCTURE
Subelements	Time uint64_t Name SwInfoName Version StringType Action ActionType Resolution ResolutionType
Derived from	-
Description	Time (UTC) in milliseconds of UCM's action resolution since 01.01.1970.

Table 9.17: Implementation Data Type - GetHistoryType

](RS_UCM_00032)

9.1.18 GetHistoryVectorType

[SWS_UCM_00135] GetHistoryType table [

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

Table 9.18: Implementation Data Type - GetHistoryVectorType

](RS_UCM_00032)

9.2 Service Interfaces

9.2.1 Provided Service Interfaces

This chapter lists all provided service interfaces of the [UCM](#).

9.2.1.1 Package Management

Port

[SWS_UCM_00073] ProvidedPort PackageManagement [

Name	PackageManagement		
Kind	ProvidedPort	Interface	PackageManagement
Description			
Variation			

Table 9.19: Port - PackageManagement

]([RS_UCM_00001](#))

Service Interface

[SWS_UCM_00131] ProvidedInterface PackageManagement [

Name	PackageManagement
NameSpace	ara::ucm::pkgmgr

Table 9.20: Service Interfaces - PackageManagement

Fields

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

Table 9.21: Service Interface PackageManagement - Field: CurrentStatus

Methods

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

Table 9.22: Service Interface PackageManagement - Method: GetSwClusterInfo

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

Table 9.23: Service Interface PackageManagement - Method: GetSwClusterChangeInfo

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

Table 9.24: Service Interface PackageManagement - Method: GetSwPackages

Name	TransferStart	
Description	Start the transfer of a Software Package. The size of the Software Package to be transferred to UCM must be provided. UCM will generate a Transfer ID for subsequent calls to TransferData, TransferExit, ProcessSwPackage, DeleteTransfer, GetLog and SetLogLevel.	
FireAndForget	false	
Parameter	size	
	Description	Size (in bytes) of the Software Package to be transferred.
	Type	uint32_t
	Variation	
	Direction	IN





Parameter	id	
	Description	Return TransferId.
	Type	TransferIdType
	Variation	
	Direction	OUT
Application Errors	InsufficientMemory	Insufficient memory to perform operation.

Table 9.25: Service Interface PackageManagement - Method: TransferStart

Name	TransferData	
Description	Block-wise transfer of a Software Package to UCM.	
FireAndForget	false	
Parameter	id	
	Description	Transfer ID.
	Type	TransferIdType
	Variation	
	Direction	IN
Parameter	data	
	Description	Data block of the Software Package.
	Type	ByteVectorType
	Variation	
	Direction	IN
Parameter	blockCounter	
	Description	Block counter value of the current block.
	Type	uint32_t
	Variation	
	Direction	IN
Application Errors	IncorrectBlock	The block counter value is not as expected.
Application Errors	IncorrectSize	The size of the Software Package exceeds the provided size in TransferStart.
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.

Table 9.26: Service Interface PackageManagement - Method: TransferData

Name	TransferExit	
Description	Finish the transfer of a Software Package to UCM.	
FireAndForget	false	
Parameter	id	





	Description	Transfer ID of the currently running request.
	Type	TransferIdType
	Variation	
	Direction	IN
Application Errors	InsufficientData	TransferExit has been called but total transferred data size does not match expected data size provided with TransferStart call.
Application Errors	PackageInconsistent	Package integrity check failed.
Application Errors	AuthenticationFailed	Software Package authentication failed.
Application Errors	OldVersion	Software Package version is too old.
Application Errors	InvalidTransferId	The Transfer ID is invalid.
Application Errors	OperationNotPermitted	The operation is not supported in the current context.

Table 9.27: Service Interface PackageManagement - Method: TransferExit

Name	DeleteTransfer	
Description	Delete a transferred Software Package.	
FireAndForget	false	
Parameter	id	
	Description	Transfer ID of the currently running request.
	Type	TransferIdType
	Variation	
	Direction	IN
Application Errors	GeneralReject	General reject.
Application Errors	GeneralMemoryError	A general memory error occurred.
Application Errors	InvalidTransferId	The Transfer ID is invalid.
Application Errors	OperationNotPermitted	The operation is not supported in the current context.

Table 9.28: Service Interface PackageManagement - Method: DeleteTransfer

Name	ProcessSwPackage	
Description	Process a previously transferred Software Package.	
FireAndForget	false	
Parameter	id	
	Description	The Transfer ID of this Software Package.
	Type	TransferIdType
	Variation	





	Direction	IN
Application Errors	ServiceBusy	Another processing is already ongoing and therefore the current processing request has to be rejected.
Application Errors	InvalidManifest	Package manifest could not be read.
Application Errors	PackageInconsistent	Package integrity check failed.
Application Errors	InsufficientMemory	Insufficient memory to perform operation.
Application Errors	InvalidTransferId	The Transfer ID is invalid.
Application Errors	OperationNotPermitted	The operation is not supported in the current context.

Table 9.29: Service Interface PackageManagement - Method: ProcessSwPackage

Name	RevertProcessedSwPackages	
Description	Revert the changes done by processing (ProcessSwPackage) of one or several software packages.	
FireAndForget	false	
Application Errors	NothingToRevert	RevertProcessedSwPackages has been called without prior processing of a Software Package.
Application Errors	NotAbleToRevertPackages	RevertProcessedSwPackages failed.
Application Errors	OperationNotPermitted	The operation is not supported in the current context.

Table 9.30: Service Interface PackageManagement - Method: RevertProcessedSwPackages

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

Table 9.31: Service Interface PackageManagement - Method: GetSwProcessProgress

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

Table 9.32: Service Interface PackageManagement - Method: Cancel

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

Table 9.33: Service Interface PackageManagement - Method: Rollback

Name	Activate	
Description	This method activates the processed components.	
FireAndForget	false	
Parameter	option	
	Description	The option of the activate.
	Type	ActivateOptionType
	Variation	
	Direction	IN
Application Errors	ErrorDuringActivation	Activate failed.
Application Errors	ErrorNoValidProcessing	Activate cannot be performed because previous processing is invalid.
Application Errors	MissingDependencies	Activate cannot be performed because of missing dependencies.
Application Errors	OperationNotPermitted	The operation is not supported in the current context.

Table 9.34: Service Interface PackageManagement - Method: Activate

Name	Finish	
Description	This method finishes the processing for the current set of processed Software Packages. It does a cleanup of all data of the processing including the sources of the Software Packages.	
FireAndForget	false	
Application Errors	GeneralRe- ject	General reject.
Application Errors	Opera- tionNotPer- mitted	The operation is not supported in the current context.

Table 9.35: Service Interface PackageManagement - Method: Finish

Name	SetLogLevel	
Description	This method sets the log level for a package.	
FireAndForget	false	
Parameter	id	
	Description	The Transfer ID.
	Type	TransferIdType
	Variation	
	Direction	IN
Parameter	logLevel	
	Description	The new log level to be used.
	Type	LogLevelType
	Variation	
	Direction	IN
Application Errors	Invalid- TransferId	The Transfer ID is invalid.

Table 9.36: Service Interface PackageManagement - Method: SetLogLevel

Name	GetLog	
Description	Getter method to poll for the log messages of the current Session.	
FireAndForget	false	
Parameter	id	
	Description	The Transfer ID.
	Type	TransferIdType
	Variation	
	Direction	IN
Parameter	log	
	Description	The log messages.
	Type	LogType
	Variation	
	Direction	OUT
Application Errors	Invalid- TransferId	The Transfer ID is invalid.

Table 9.37: Service Interface PackageManagement - Method: GetLog

Name	GetHistory	
Description	Getter method to retrieve all actions that have been performed by UCM.	
FireAndForget	false	
Parameter	history	
	Description	The history of all actions that have been performed by UCM.
	Type	GetHistoryVectorType
	Variation	
	Direction	OUT

Table 9.38: Service Interface PackageManagement - Method: GetHistory

[RS_UCM_00001](#), [RS_UCM_00002](#), [RS_UCM_00008](#), [RS_UCM_00010](#),
[RS_UCM_00011](#), [RS_UCM_00015](#), [RS_UCM_00018](#), [RS_UCM_00021](#),
[RS_UCM_00022](#), [RS_UCM_00023](#), [RS_UCM_00024](#), [RS_UCM_00025](#),
[RS_UCM_00032](#))

9.3 Application Errors

This chapter lists all application errors of the [UCM](#).

9.3.1 Application Error Domain

9.3.1.1 UCMErrDomain

[SWS_UCM_00136] UCMErrDomain [

Name	Code	Description
InsufficientMemory	1	Insufficient memory to perform operation.
IncorrectBlock	2	The block counter value is not as expected.
IncorrectSize	3	The size of the Software Package exceeds the provided size in TransferStart.
InvalidTransferId	4	The Transfer ID is invalid.
OperationNotPermitted	5	The operation is not supported in the current context.
InsufficientData	6	TransferExit has been called but total transferred data size does not match expected data size provided with TransferStart call.
PackageInconsistent	7	Package integrity check failed.
AuthenticationFailed	8	Software Package authentication failed.
OldVersion	9	Software Package version is too old.
GeneralReject	10	General reject.
GeneralMemoryError	11	A general memory error occurred.
ServiceBusy	12	Another processing is already ongoing and therefore the current processing request has to be rejected.





InvalidManifest	13	Package manifest could not be read.
NothingToRevert	14	RevertProcessedSwPackages has been called without prior processing of a Software Package.
NotAbleToRevertPackages	15	RevertProcessedSwPackages failed.
CancelFailed	16	Cancel failed.
NothingToRollback	17	Rollback cannot be performed due to no rollback data available.
NotAbleToRollback	18	Rollback failed.
ErrorDuringActivation	19	Activate failed.
ErrorNoValidProcessing	20	Activate cannot be performed because previous processing is invalid.
MissingDependencies	21	Activate cannot be performed because of missing dependencies.

Table 9.39: Application Errors of UCMErrDomain

|(RS_UCM_00006, RS_UCM_00007, RS_UCM_00012, RS_UCM_00013, RS_UCM_00014)

10 Sequence diagrams

10.1 Update process

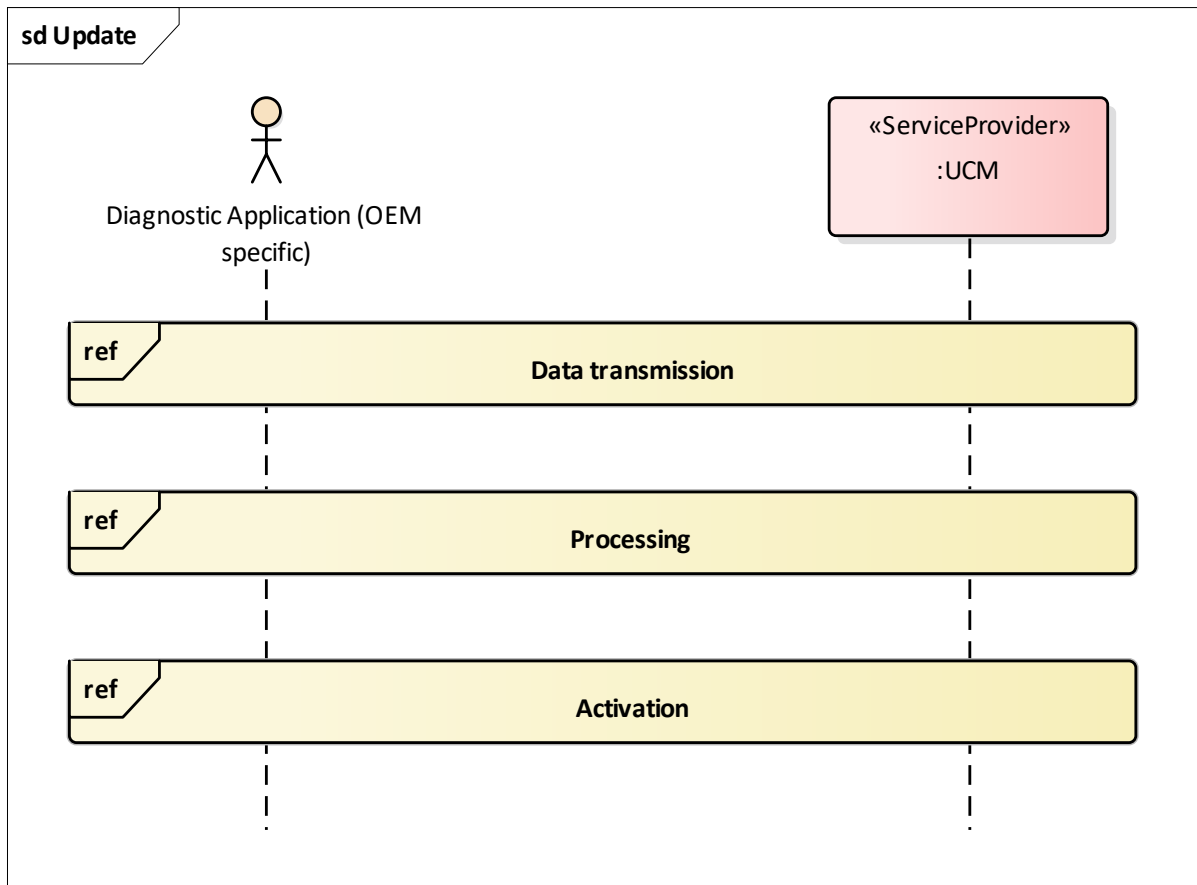


Figure 10.1: Sequence diagram showing the update process

10.2 Data transmission

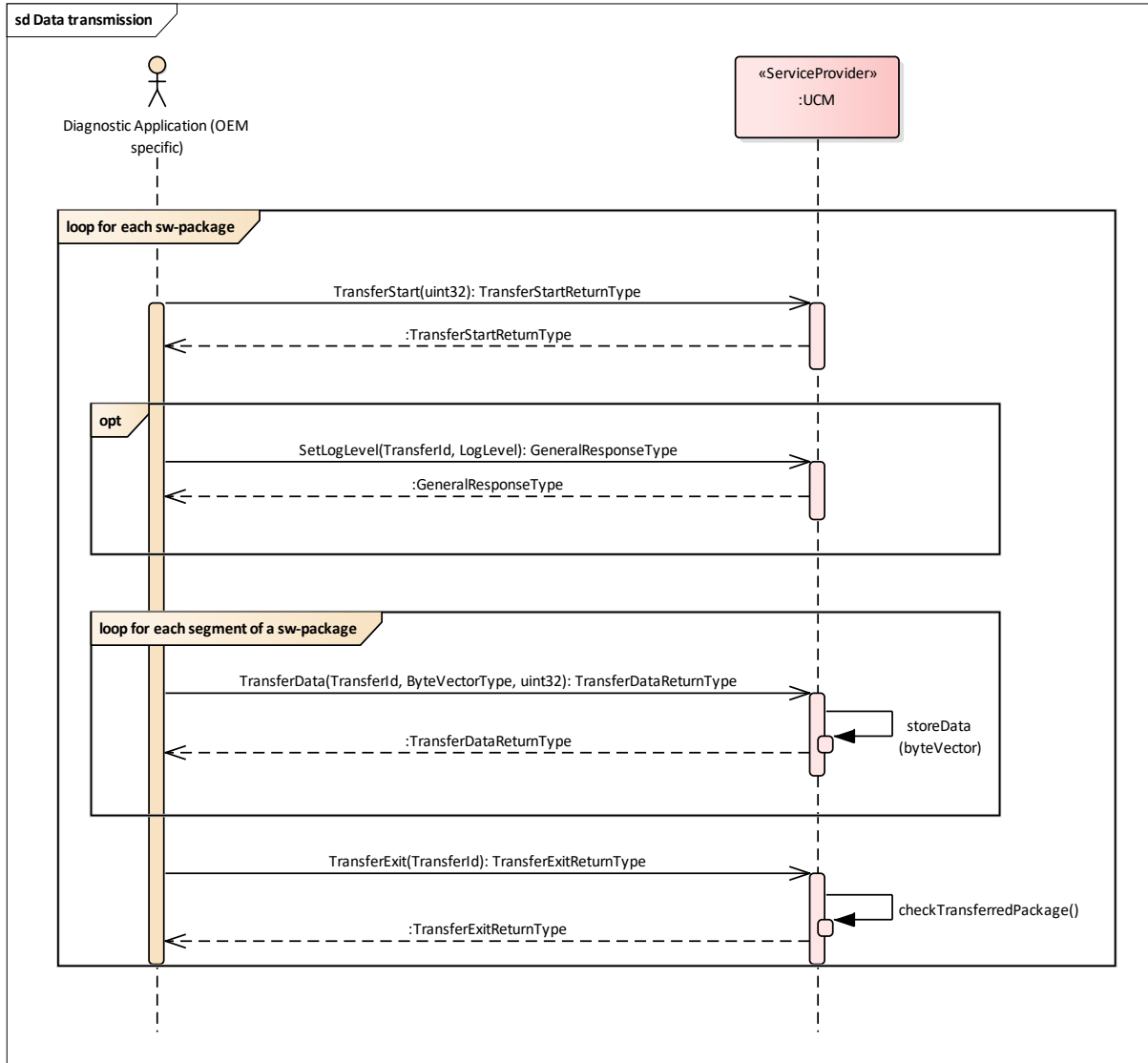


Figure 10.2: Sequence diagram showing the data transmission

10.3 Package processing

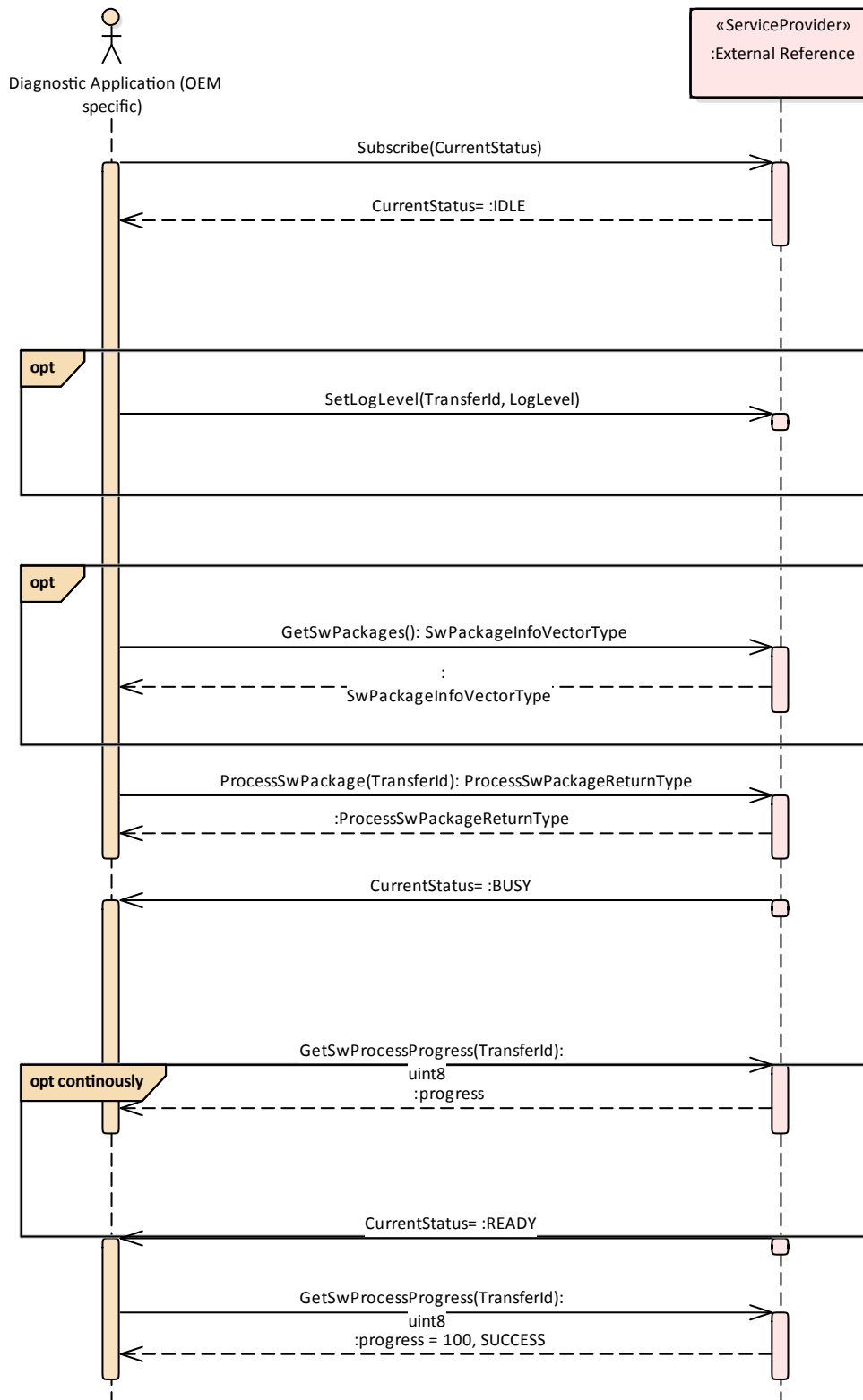


Figure 10.3: Sequence diagram showing the package processing

10.4 Activation

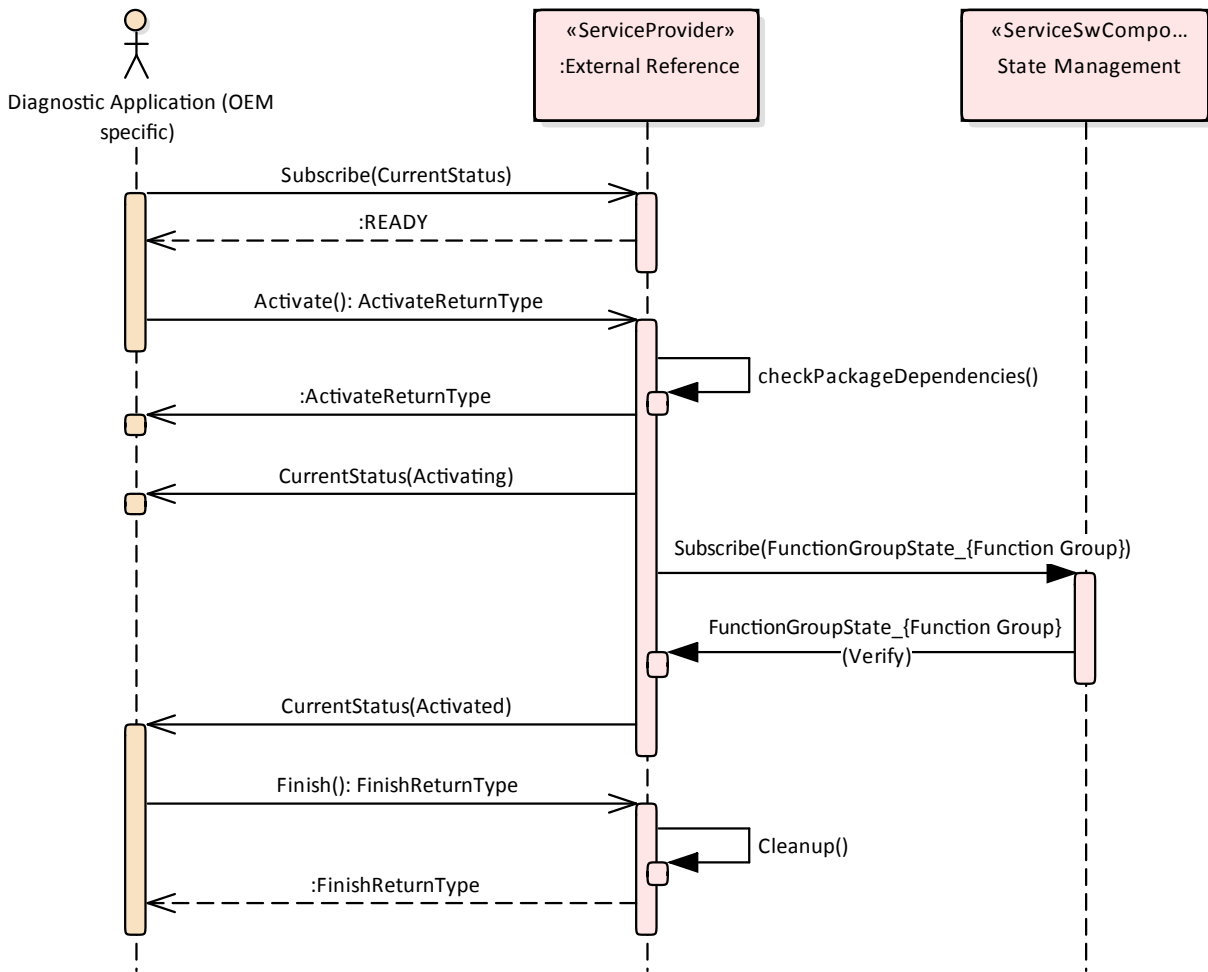


Figure 10.4: Sequence diagram showing the activation process

A Not applicable requirements

none

B Mentioned Class Tables

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

Class	Identifiable (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.			
Base	<i>ARObject, MultilanguageReferrable, Referrable</i>			
Subclasses	<p>ARPackage, <i>AbstractEvent, AbstractImplementationDataTypeElement, AbstractServiceInstance, AdaptiveModuleInstantiation, AdaptiveSwcInternalBehavior, ApplicationEndpoint, ApplicationError, ApplicationPartitionToEcuPartitionMapping, AsynchronousServerCallResultPoint, AtpBlueprint, AtpBlueprintable, AtpClassifier, AtpFeature, AutosarOperationArgumentInstance, AutosarVariableInstance, BswInternalTriggeringPoint, BswModuleDependency, BuildActionEntity, BuildActionEnvironment, CanTpAddress, CanTpChannel, CanTpNode, Chapter, CheckpointTransition, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, CollectableElement, ComManagementMapping, CommConnectorPort, CommunicationConnector, CommunicationController, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, CouplingPortStructuralElement, CryptoServiceMapping, DataPrototypeGroup, DataTransformation, DdsRpcServiceDeployment, DependencyOnArtifact, DeterministicClientResourceNeeds, DiagEventDebounceAlgorithm, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticFunctionInhibitSource, DiagnosticMasterToSlaveEventMapping, DiagnosticRoutineSubfunction, DolpLogicAddress, E2EProfileConfiguration, ECUMapping, EOCExecutableEntityRefAbstract, EcuPartition, EcuContainerValue, EcuDefinitionElement, EcuDestinationUriDef, EcuEnumerationLiteralDef, EcuQuery, EcuValidationCondition, End2EndEventProtectionProps, EndToEndProtection, EventMapping, ExclusiveArea, ExecutableEntity, ExecutionTime, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FieldMapping, FireAndForgetMapping, FlatInstanceDescriptor, FlexrayArTpNode, FlexrayTpConnectionControl, FlexrayTpNode, FlexrayTpPduPool, FrameTriggering, GeneralParameter, GlobalTimeGateway, GlobalTimeMaster, GlobalTimeSlave, HealthChannel, HeapUsage, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, IPSecRule, IPv6ExtHeaderFilterList, ISignalToPduMapping, ISignalTriggering, IdentCaption, InterfaceMapping, InternalTriggeringPoint, J1939SharedAddressCluster, J1939TpNode, Keyword, LifeCycleState, LinScheduleTable, LinTpNode, Linker, MacMulticastGroup, McDataInstance, MemorySection, MethodMapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint, NmCluster, NmNode, NvBlockDescriptor, PackageableElement, ParameterAccess, PduToFrameMapping, PduTriggering, PerlInstanceMemory, PersistencyFileProxy, PersistencyKeyValuePair, PhmAction, PhmActionItem, PhmActionList, PhmArbitration, PhmLogicalExpression, PhmRule, PhmSupervision, PhysicalChannel, PortGroup, PortInterfaceMapping, PossibleErrorReaction, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKeySlotMapping, ResourceConsumption, ResourceGroup, RestAbstractEndpoint, RestElementDef, RestResourceDef, RootSwComponentPrototype, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RunnableEntityGroup, SdgAttribute, SdgClass, SecOcJobMapping, SecOcJobRequirement, SecureComProps, SecureCommunicationAuthenticationProps, SecureCommunicationDeployment, SecureCommunicationFreshnessProps, ServerCallPoint, ServiceEventDeployment, ServiceFieldDeployment, ServiceInstanceToSignalMapping, ServiceInterfaceElementMapping, ServiceInterfaceElementSecureComConfig, ServiceInterfaceMapping, ServiceMethodDeployment, ServiceNeeds, SignalBasedFieldToSignalTriggeringMapping, SocketAddress, SomeipEventGroup, SomeipProvidedEventGroup, SomeipTpChannel, SpecElementReference, StackUsage, StartupConfig, StructuredReq, SupervisionCheckpoint, SwGenericAxisParamType, SwServiceArg, SwServiceDependency, SwToApplicationPartitionMapping, SwToEcuMapping, SwToImplMapping, SystemMapping, TcpOptionFilterList, TimeBaseResource, TimingCondition, TimingConstraint, TimingDescription, TimingExtensionResource, TimingModelInstance, TlsCryptoCipherSuite, TlsJobMapping, Topic1, TpAddress, TraceableText, TracedFailure, TransformationProps, TransformationPropsToServiceInterfaceElementMapping, TransformationTechnology, Trigger, VariableAccess, VariationPointProxy, ViewMap, VlanConfig, WaitPoint</i></p>			
Attribute	Type	Mul.	Kind	Note





Class	Identifiable (abstract)			
desc	MultiLanguageOverview Paragraph	0..1	aggr	<p>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.</p> <p>More elaborate documentation, (in particular how the object is built or used) should go to "introduction".</p> <p>Tags: xml.sequenceOffset=-60</p>
category	CategoryString	0..1	attr	<p>The category is a keyword that specializes the semantics of the Identifiable. It affects the expected existence of attributes and the applicability of constraints.</p> <p>Tags: xml.sequenceOffset=-50</p>
adminData	AdminData	0..1	aggr	<p>This represents the administrative data for the identifiable object.</p> <p>Tags: xml.sequenceOffset=-40</p>
annotation	Annotation	*	aggr	<p>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.</p> <p>Tags: xml.sequenceOffset=-25</p>
introduction	DocumentationBlock	0..1	aggr	<p>This represents more information about how the object in question is built or is used. Therefore it is a DocumentationBlock.</p> <p>Tags: xml.sequenceOffset=-30</p>
uuid	String	0..1	attr	<p>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.</p> <p>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.</p> <p>If the id namespace is omitted, DCE is assumed. An example is "DCE:2fac1234-31f8-11b4-a222-08002b34c003".</p> <p>The uuid attribute has no semantic meaning for an AUTOSAR model and there is no requirement for AUTOSAR tools to manage the timestamp.</p> <p>Tags: xml.attribute=true</p>

Table B.1: Identifiable

Class	SoftwareCluster			
Package	M2::AUTOSARTemplates::AdaptivePlatform::UploadableSoftwarePackage			
Note	<p>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.</p> <p>Tags: atp.Status=draft atp.recommendedPackage=SoftwareClusters</p>			
Base	ARElement, ARObject, CollectableElement, <i>Identifiable</i> , MultilanguageReferrable, PackageableElement, Referrable			
Attribute	Type	Mul.	Kind	Note
containedARElement	ARElement	*	ref	<p>This reference represents the collection of model elements that cannot derive from UploadablePackageElement and that contribute to the completeness of the definition of the SoftwareCluster.</p> <p>Stereotypes: atpSplitable Tags: atp.Splitkey=shortName atp.Status=draft</p>
containedFibexElement	FibexElement	*	ref	<p>This allows for referencing FibexElements that need to be considered in the context of a SoftwareCluster.</p> <p>Tags: atp.Status=draft</p>
containedPackageElement	UploadablePackageElement	*	ref	<p>This reference identifies model elements that are required to complete the manifest content.</p> <p>Stereotypes: atpSplitable Tags: atp.Splitkey=containedPackageElement atp.Status=draft</p>
containedProcess	Process	*	ref	<p>This reference represent the processes contained in the enclosing SoftwareCluster.</p> <p>Tags: atp.Status=draft</p>
dependsOn	SoftwareClusterDependency	*	aggr	<p>This aggregation can be taken to identify a dependency for the enclosing SoftwareCluster.</p> <p>Stereotypes: atpSplitable Tags: atp.Splitkey=dependsOn atp.Status=draft</p>
design	SoftwareClusterDesign	*	ref	<p>This reference represents the identification of all SoftwareClusterDesigns applicable for the enclosing SoftwareCluster.</p> <p>Stereotypes: atpUriDef Tags: atp.Status=draft</p>
diagnosticAddress	SoftwareClusterDiagnosticAddress	*	aggr	<p>This aggregation represents the collection of diagnostic addresses that apply for the SoftwareCluster.</p> <p>Stereotypes: atpSplitable Tags: atp.Splitkey=diagnosticAddress atp.Status=draft</p>
diagnosticExtract	DiagnosticContributionSet	0..1	ref	<p>This reference represents the definition of the diagnostic extract applicable to the referencing SoftwareCluster</p> <p>Tags: atp.Status=draft</p>
moduleInstantiation	AdaptiveModuleInstantiation	*	ref	<p>This reference identifies AdaptiveModuleInstantiations that need to be included with the SoftwareCluster in order to establish infrastructure required for the installation of the SoftwareCluster.</p> <p>Stereotypes: atpSplitable Tags: atp.Splitkey=moduleInstantiation atp.Status=draft</p>
subSoftwareCluster	SoftwareCluster	*	ref	<p>This reference is used to identify the sub-SoftwareClusters of an "umbrella" SoftwareCluster.</p> <p>Stereotypes: atpSplitable Tags: atp.Splitkey=subSoftwareCluster atp.Status=draft</p>





<i>Class</i>	SoftwareCluster			
version	String	1	attr	This attribute can be used to describe a version information for the enclosing SoftwareCluster. The format of the version as well as how to tell a lower from a higher version is not prescribed by the AUTOSAR standard.

Table B.2: SoftwareCluster

C Interfaces to other Functional Clusters (informative)

C.1 Overview

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

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

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

C.2 Interfaces Tables

C.2.1 UCM update notification

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

D Packages distribution within vehicle (informative)

D.1 Overview

To prepare next releases of this specification, Update and Configuration Management team appends to this specification its future image of packages distribution from a backend within a vehicle and UCMs coordination by sharing sequence diagrams. Intention of this appendix is to gather comments from Autosar community to ensure future API's quality. All described methods have to be later specified.

This vision involves so called UCM master (former Vehicle Update Manager, VUM) which is communicating with backend through network means that are out of scope for this work. This UCM master could be replaced by another UCM (UCM slaves) if UCM master is failing.

UCM master receives a vehicle package manifest container sent by backend which contains all software cluster descriptions along with campaign orchestration needed by UCM to distribute the Software Packages within vehicle. After manifest reception and authentication, UCM master streams the Software Packages to the related UCM slaves, perform processing and activation according to campaign orchestration defined sequence. More details could be found in document [9]

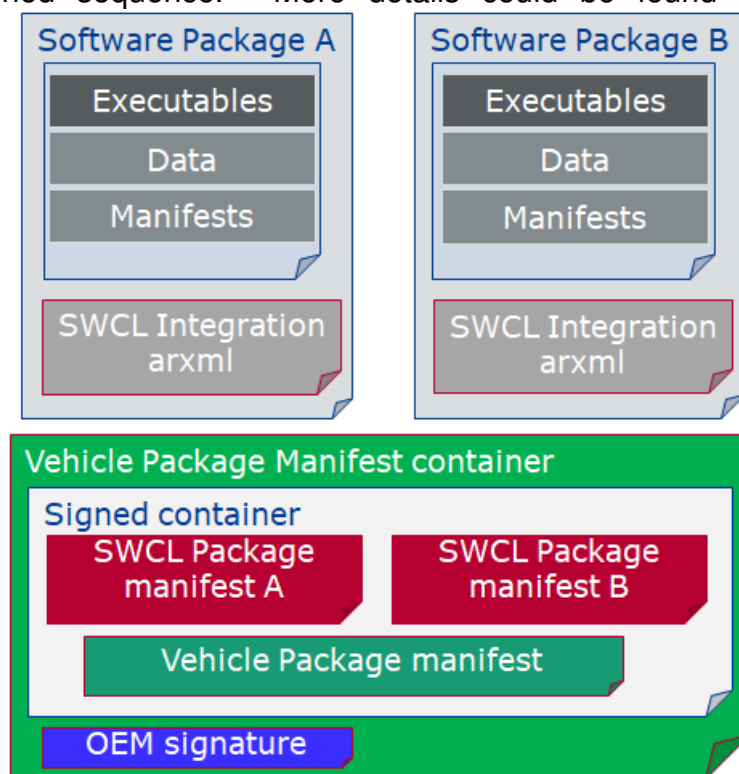


Figure D.1: Vehicle package overview

It is considered that distribution method of packages via internet network into vehicle could also be performed using diagnostic. For instance, if there would be a car communication issue, we could imagine a technician from a garage would download from

a backend a vehicle package manifest along software packages that would then be pushed to UCM master via diagnostic service manager and diagnostic application.

UCM master is generally acting like an application or UCM client, following the already specified general sequence (transfer, process, activate/rollback, finish).

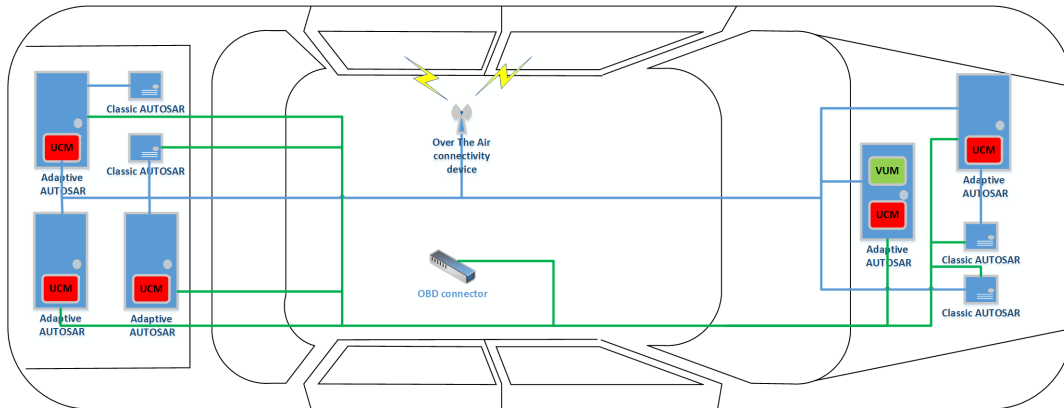


Figure D.2: Vehicle infrastructure example

D.2 Packages distribution sequence diagrams

D.2.1 UCM slave discovery

For UCM master to distribute software packages to other UCM slaves, UCM master has to discover other UCM slaves in vehicle. This discovery could be at boot or later but at least before any communication with backend are engaged. Each UCM has a unique identifier that is part of SwClusterInfo type to help UCM master streaming packages to target UCM slaves.

UCM master offers *UCM slave discovery service* and receives data which are necessary to transfer packages to certain destinations. UCM slaves send these data by responding the service from UCM master. The data could be a combination of the destination information (e.g. IP address, MAC address, port number) but also could be a key which could link to the destination information by using certain structure (e.g. KVS, mapping table). *UCM slave discovery* can be done in a similar process to SOME/IP Service Discovery [6].

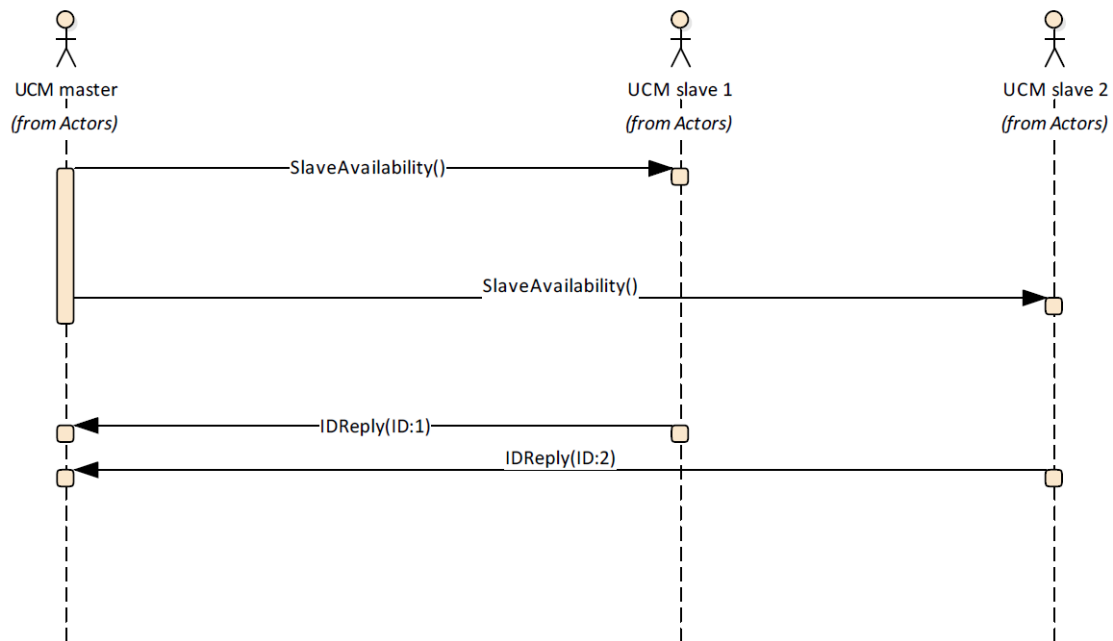


Figure D.3: Discover UCM slaves

D.2.2 Collect information of present SWCLs in vehicle

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

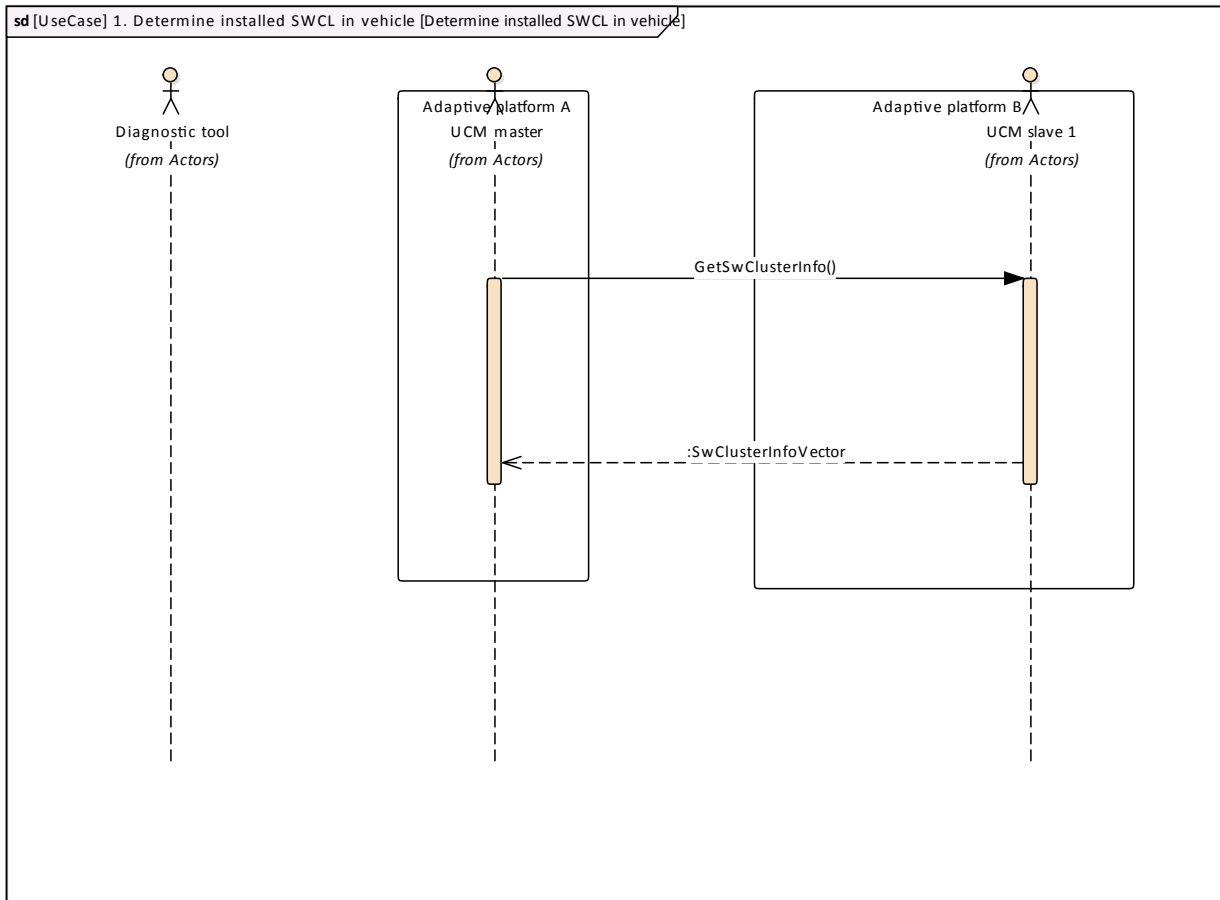


Figure D.4: Collect information of SWCLs present in vehicle from several Adaptive Platforms

D.2.3 Action computation

In order to find out if there is a new update available from backend or the need to install or remove a SWCL, vehicle and backend have to share their current status and either backend or vehicle have to compute what UCM action is needed.

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

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

D.2.3.1 Pull package from backend into vehicle

Case where vehicle is computing the difference between SWCLs versions that are present in vehicle and the ones available in backend.

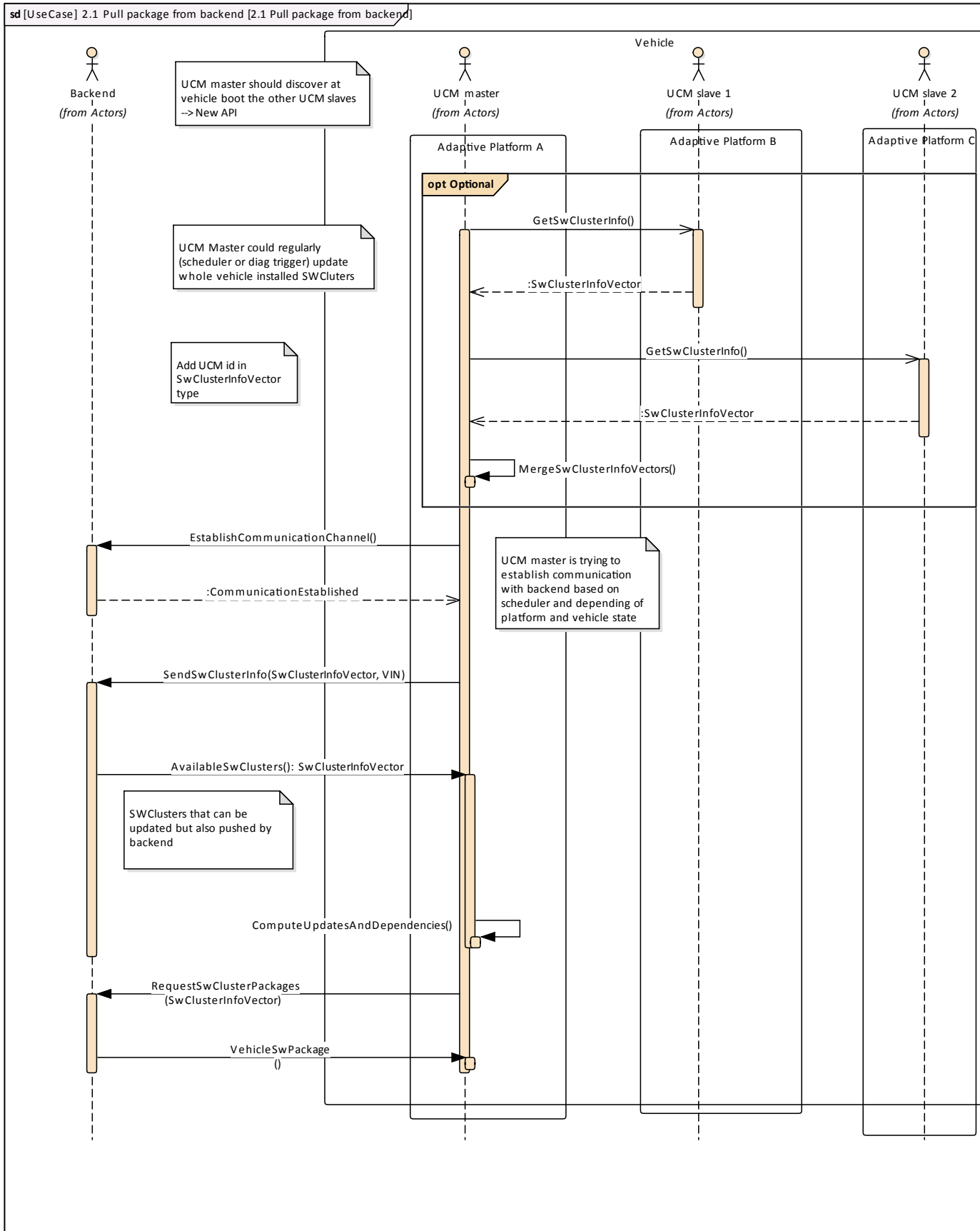


Figure D.5: Pull package from backend

D.2.3.2 Push package from backend into vehicle

Case where backend is computing the difference between SWCLs versions that are present in vehicle and the ones available in backend.

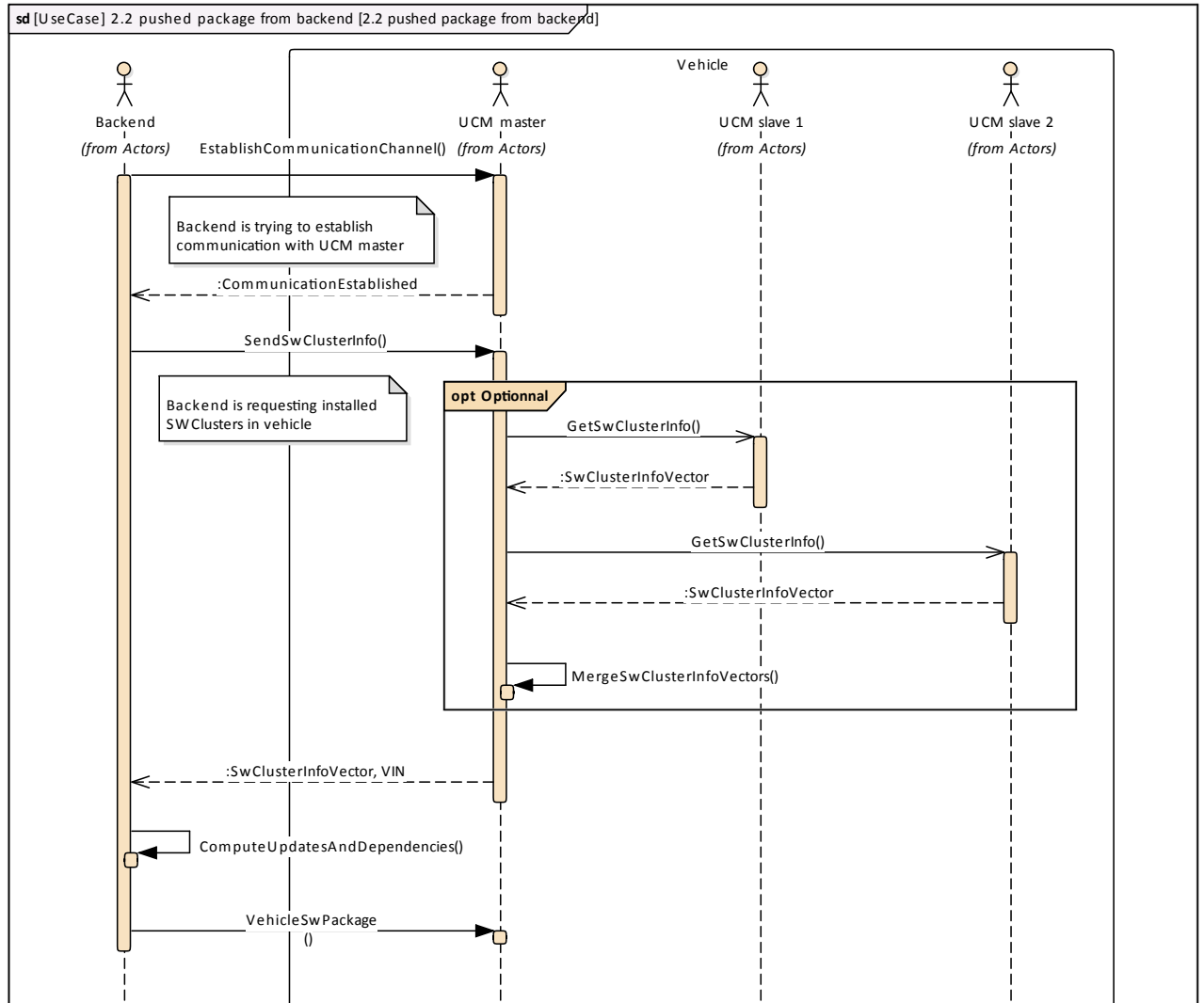


Figure D.6: Push package from backend

D.2.4 Packages transfer from backend into targeted UCM

To reduce as much as possible the amount of data temporarily stored between backend and targeted UCM, package is divided in blocks that are streamed and counted, allowing easy resume in case of communication loss. Driver should be asked before downloading package as communication could have safety and cost impact.

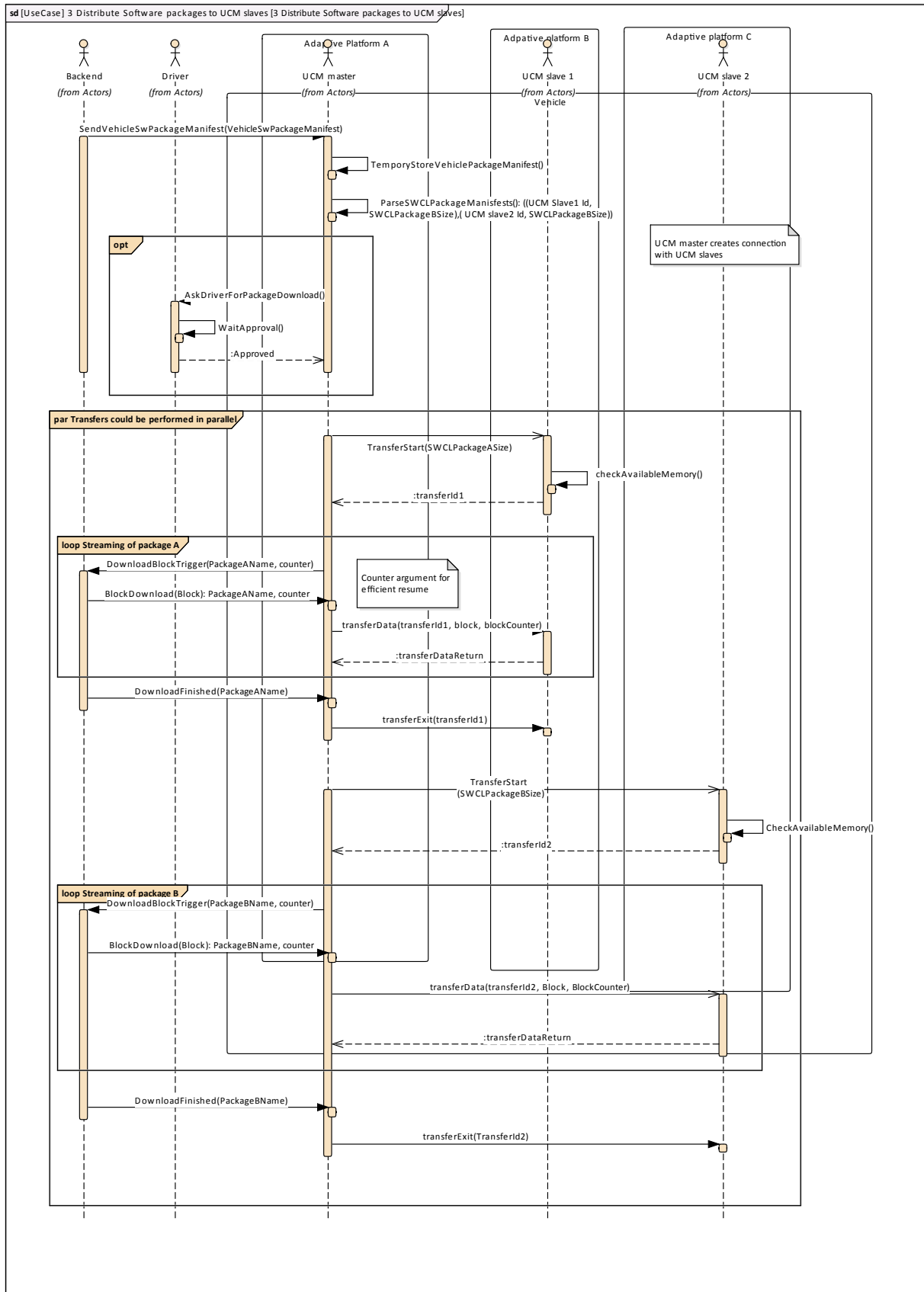


Figure D.7: Stream packages blocks from backend into targeted UCM

D.2.5 Package processing

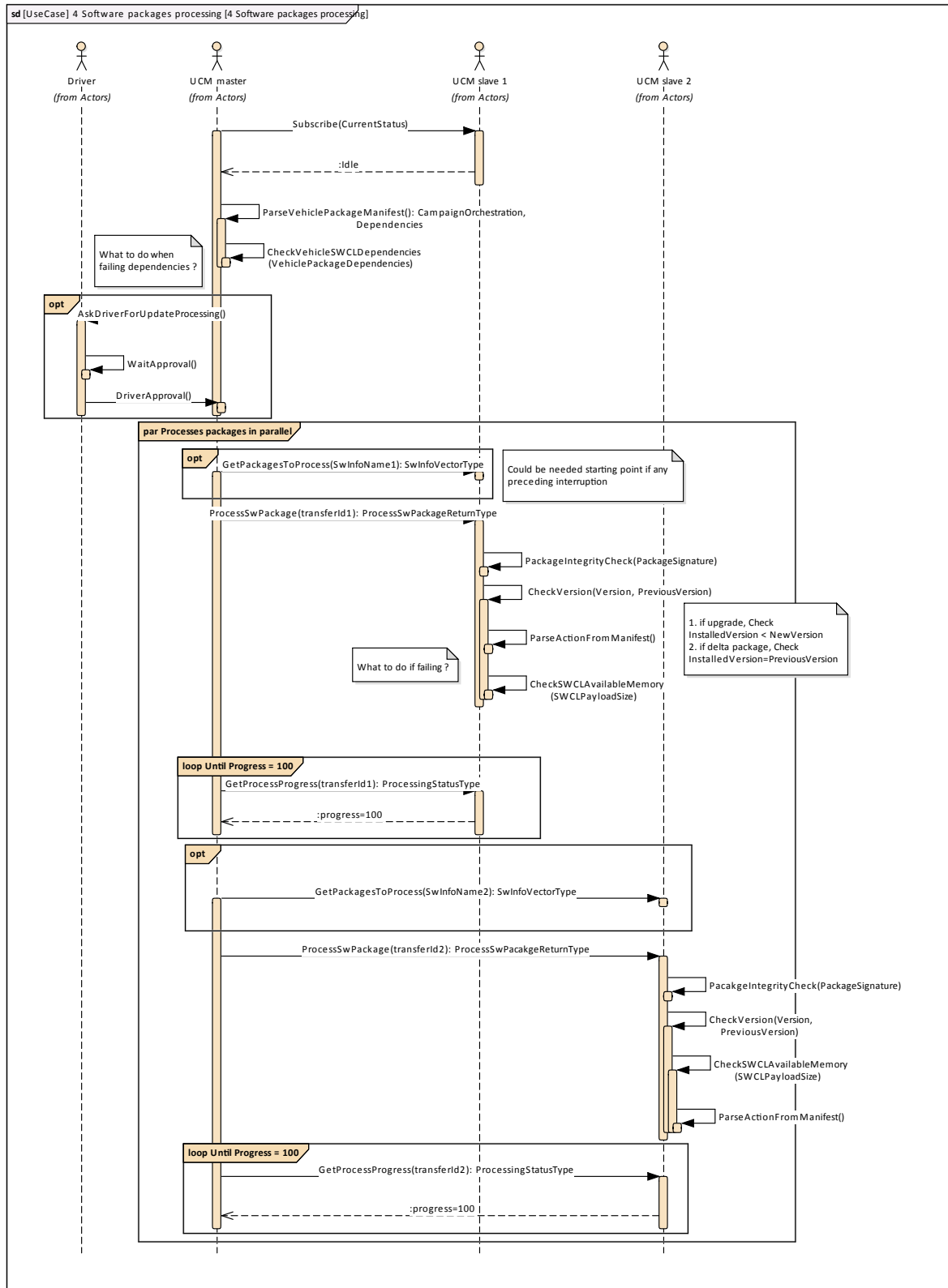


Figure D.8: Packages processing by UCMs

D.2.6 Package activation

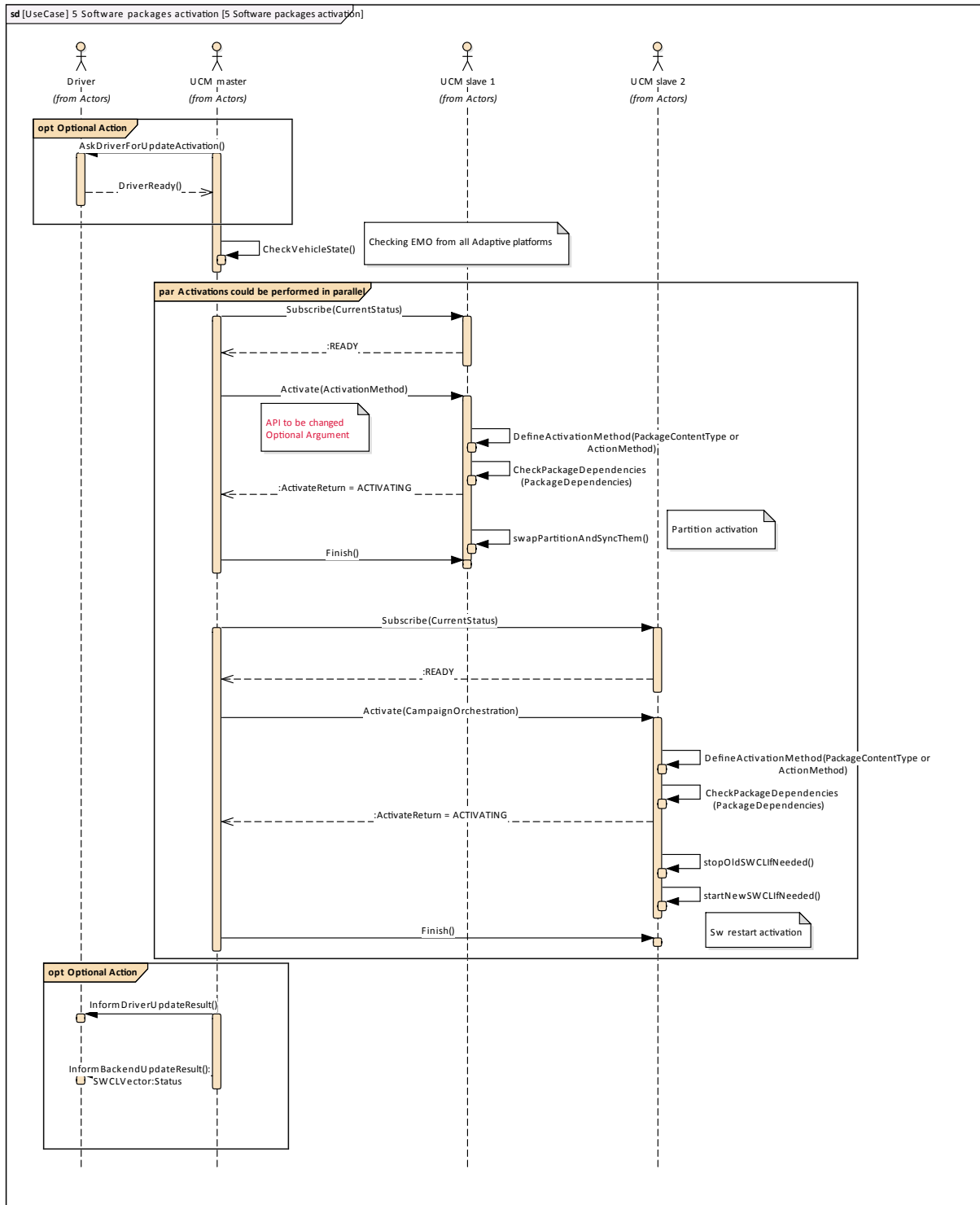


Figure D.9: Packages activation by UCMs

D.2.7 Package rollback

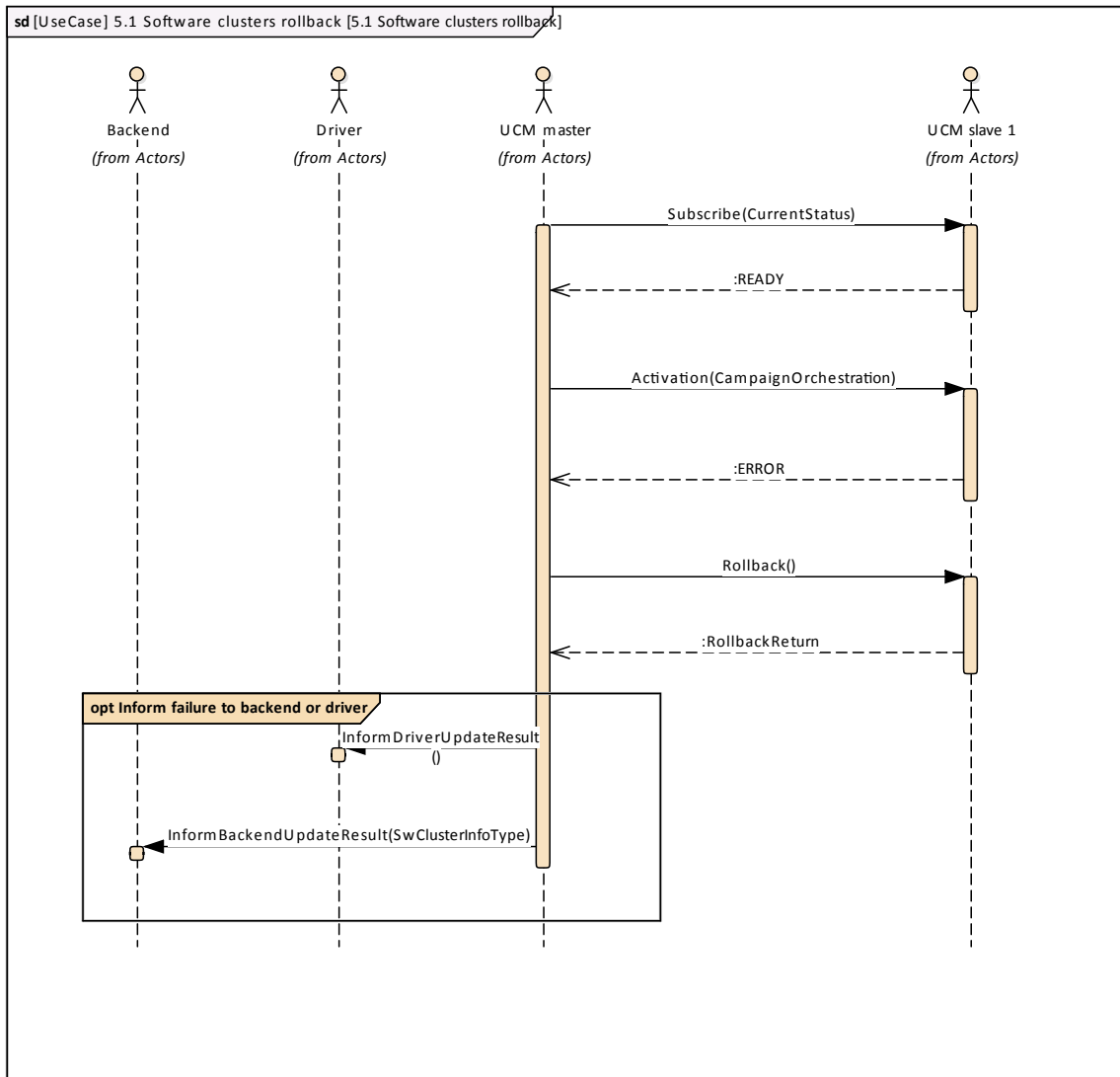


Figure D.10: Packages rollback by UCMs

D.2.8 Campaign reporting

After campaign is finished (finish method has been sent to all UCMs), UCM should report to backend server status of the vehicle, with for instance updated information of SWCLs present in vehicle.