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	18-03

Document Change History

Date	Release	Changed by	Description
2017-03-29	18-03	AUTOSAR Release Management	• Extended and updated service interface • Introduction of Software Package • Introduction to securing update process
2017-10-27	17-10	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Introduction and functional overview

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

The `Update and Configuration Management` functional cluster is responsible for:

- Version reporting of the software present in the Adaptive Platform
- Receiving and buffering software updates
- Checking enough memory is 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 Adaptive Platform, the Update and Configuration Management is also responsible for updates and changes to the 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 Update and Configuration Management that are not included in the [1, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
UCM	Update and Configuration Management

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_TR_Glossary

[2] Specification of Execution Management
AUTOSAR_SWS_ExecutionManagement

[3] Requirements on Update and Configuration Management
AUTOSAR_RS_UpdateAndConfigManagement

[4] Requirements on Security Management for Adaptive Platform
AUTOSAR_RS_SecurityManagement

[5] Explanation of Adaptive Platform Design
AUTOSAR_EXP_PlatformDesign

[6] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[7] Specification of Persistency
AUTOSAR_SWS_Persistency

[8] Specification of Crypto Interface for Adaptive Platform
AUTOSAR_SWS_AdaptiveCryptoInterface

[9] Specification of Identity and Access Management
AUTOSAR_SWS_IdentityAndAccessManagement

3.2 Related specification

See chapter 3.1.

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 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.

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 Adaptive Platform. UCM can update Adaptive Applications, the 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.

4.2 Applicability to car domains

No restrictions to applicability.

5 Dependencies between functional clusters

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

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 Group`s and its states. It is the responsibility of the platform integrator to define this state or `Function Group`s. And the application accessing the UCM, should make sure that the platform is switched to this state (using interfaces from Execution Management [2]), before starting the update.

The UCM shall use ara::com events to declare that an update was successfully completed.

6 Requirements Tracing

The following tables reference the requirements specified in [3] 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_UCM_00001]	UCM shall support installing new software on Adaptive Platform	[SWS_UCM_00001] [SWS_UCM_00017] [SWS_UCM_00037] [SWS_UCM_00059] [SWS_UCM_00072] [SWS_UCM_00073]
[RS_UCM_00002]	UCM shall support reporting version information for an Adaptive Platform	[SWS_UCM_00004] [SWS_UCM_00038] [SWS_UCM_00039] [SWS_UCM_00040] [SWS_UCM_00052] [SWS_UCM_00071] [SWS_UCM_00077] [SWS_UCM_00078] [SWS_UCM_00079]
[RS_UCM_00003]	UCM shall support updating installed software on Adaptive Platform	[SWS_UCM_00017] [SWS_UCM_00037] [SWS_UCM_00059] [SWS_UCM_00072]
[RS_UCM_00004]	UCM shall support uninstalling software on Adaptive Platform	[SWS_UCM_00001] [SWS_UCM_00037] [SWS_UCM_00059] [SWS_UCM_00072]
[RS_UCM_00005]	UCM shall make sure that persistent data owned by uninstalled software is deleted	[SWS_UCM_00001]
[RS_UCM_00006]	UCM shall check Software Package authentication during processing using signature verification	[SWS_UCM_00028] [SWS_UCM_00036] [SWS_UCM_00038] [SWS_UCM_00039] [SWS_UCM_00040] [SWS_UCM_00057] [SWS_UCM_00074] [SWS_UCM_00077] [SWS_UCM_00078] [SWS_UCM_00079]
[RS_UCM_00007]	UCM shall check that software dependencies are fulfilled	[SWS_UCM_00026]
[RS_UCM_00008]	UCM shall support a recovery mechanism in case of failed update process	[SWS_UCM_00005] [SWS_UCM_00024] [SWS_UCM_00046] [SWS_UCM_00049] [SWS_UCM_00063] [SWS_UCM_00064]

Requirement	Description	Satisfied by
[RS_UCM_00010]	UCM shall support reporting of Software Packages downloaded for Adaptive Platform	[SWS_UCM_00038] [SWS_UCM_00039] [SWS_UCM_00040] [SWS_UCM_00054] [SWS_UCM_00069] [SWS_UCM_00077] [SWS_UCM_00078] [SWS_UCM_00079]
[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_00053] [SWS_UCM_00077] [SWS_UCM_00078] [SWS_UCM_00079]
[RS_UCM_00012]	UCM shall check the consistency of Software Package during processing	[SWS_UCM_00036] [SWS_UCM_00038] [SWS_UCM_00039] [SWS_UCM_00040] [SWS_UCM_00057] [SWS_UCM_00077] [SWS_UCM_00078] [SWS_UCM_00079]
[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_00033] [SWS_UCM_00034] [SWS_UCM_00035] [SWS_UCM_00036] [SWS_UCM_00055] [SWS_UCM_00056] [SWS_UCM_00057]
[RS_UCM_00014]	UCM shall check that correct amount of data has been transferred for the Software Package	[SWS_UCM_00035] [SWS_UCM_00036] [SWS_UCM_00056] [SWS_UCM_00057]
[RS_UCM_00015]	UCM shall remove all unneeded data after Software Package processing has finished	[SWS_UCM_00020] [SWS_UCM_00050] [SWS_UCM_00051] [SWS_UCM_00058] [SWS_UCM_00065]
[RS_UCM_00016]	UCM shall check installed software is consistent with provided Software Cluster Manifest	[SWS_UCM_00029] [SWS_UCM_00036] [SWS_UCM_00037] [SWS_UCM_00057] [SWS_UCM_00059] [SWS_UCM_00072]
[RS_UCM_00017]	UCM shall support installing and updating the persistent data storage for an Adaptive Application	[SWS_UCM_00011]

Requirement	Description	Satisfied by
[RS_UCM_00018]	UCM shall announce when an application has been installed, updated or uninstalled	[SWS_UCM_00021] [SWS_UCM_00037] [SWS_UCM_00048] [SWS_UCM_00059] [SWS_UCM_00060] [SWS_UCM_00068] [SWS_UCM_00072]
[RS_UCM_00019]	UCM shall support simultaneous transfer from multiple clients	[SWS_UCM_00007] [SWS_UCM_00008] [SWS_UCM_00010] [SWS_UCM_00031] [SWS_UCM_00033] [SWS_UCM_00034] [SWS_UCM_00035] [SWS_UCM_00036] [SWS_UCM_00055] [SWS_UCM_00056] [SWS_UCM_00057] [SWS_UCM_00075]
[RS_UCM_00020]	UCM shall support cancel of an update or install operation	[SWS_UCM_00003] [SWS_UCM_00047] [SWS_UCM_00062]
[RS_UCM_00021]	UCM shall support atomic activation of installed or updated packages	[SWS_UCM_00022] [SWS_UCM_00023] [SWS_UCM_00025] [SWS_UCM_00046] [SWS_UCM_00064] [SWS_UCM_00076]
[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_00066] [SWS_UCM_00067]
[RS_UCM_00023]	UCM shall provide an interface to read progress of the update	[SWS_UCM_00018] [SWS_UCM_00061]
[RS_UCM_00024]	UCM shall provide an interface to read internal status of UCM	[SWS_UCM_00019] [SWS_UCM_00037] [SWS_UCM_00044] [SWS_UCM_00072]
[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_00033] [SWS_UCM_00034] [SWS_UCM_00035] [SWS_UCM_00036] [SWS_UCM_00055] [SWS_UCM_00056] [SWS_UCM_00057]

Requirement	Description	Satisfied by
[RS_UCM_00026]	UCM shall process installation of new Software Packages, updates and removal of Software Packages sequentially	[SWS_UCM_00017] [SWS_UCM_00044] [SWS_UCM_00059]

7 Functional specification

7.1 Technical Overview

One of the declared goals of Adaptive AUTOSAR is the ability to flexibly update the software and its configuration through over-the-air updates. During the lifecycle of an Adaptive Platform, UCM is responsible to perform software modifications on the machine and to retain consistency of the whole system. Software updates processed by UCM can address any application on the platform but also the Adaptive Platform itself, including all functional clusters and the underlying OS.

The UCM functional cluster provides a service interface that exposes its functionality to retrieve 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 Adaptive Platform, but also remote clients are possible. It is up to the Identity and Access Management [4] to secure the access to the UCM service interface.

The service interface has been primarily designed with the goal to make it possible to use standard diagnostic services for uploading and installing software updates for the 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 how data is transferred to the 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 and DM. The Diagnostic Client uploads 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.

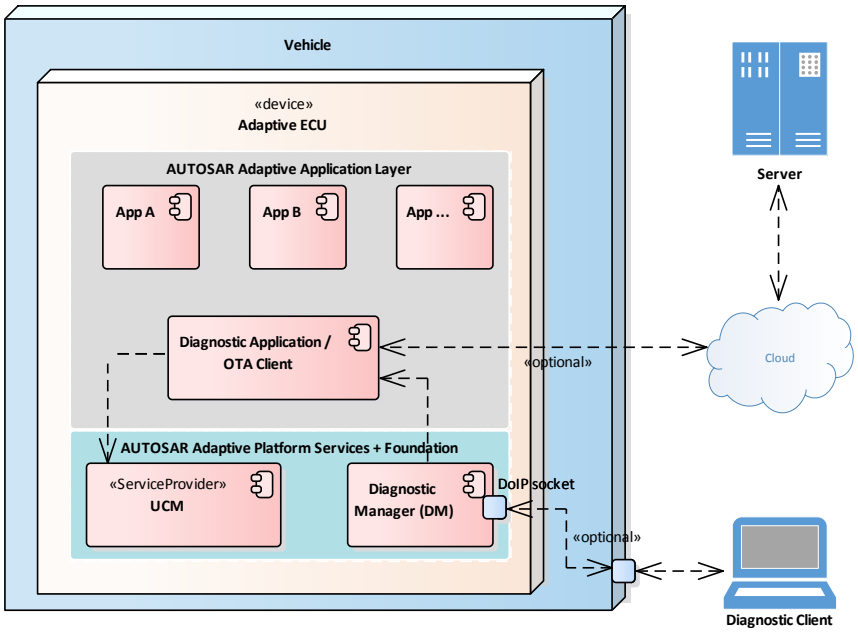

Figure 7.1: Architecture overview for diagnostic use case

7.2 Software Package Management

7.2.1 Software Package

The unit for deployment which is the input for the functional cluster Update and Configuration Management is a Software Package, see [1]. The content of a Software Package includes, for example, one or several executables of (Adaptive) Applications, kernel or firmware updates, or updated configuration and calibration data that shall be deployed on the Adaptive Platform. An exemplary implementation of the adaptive workflow with Software Packages can be seen in chapter Methodology and Manifest in [5].

Formally, a software package is modeled as a so-called `SoftwareCluster` which describes the content of a software package that shall be uploaded to the Adaptive Platform, see [6]. The term Software Package is used for the physical, uploadable software package that is processed by `UCM` whereas the term `SoftwareCluster` is used for the modeling element. In the model, the content of a `SoftwareCluster` is determined by references to all required model elements. The `SoftwareCluster` and the related model elements are put into a manifest that is part of the Software Package. The software package format and the update scope, e.g. complete or a delta update of a `SoftwareCluster`, is implementation specific and not covered by this specification.

A `SoftwareCluster` can act in two roles. Either it is a 'Root'- `SoftwareCluster` which has a diagnostic target address or it is a 'Sub'-`SoftwareCluster` without diagnostic target address. The two roles are expressed by reserved values of the attribute

SoftwareCluster.category. The 'Sub'-SoftwareCluster is a SoftwareCluster with processes, executables and further elements. A 'Root'-SoftwareCluster may reference several other 'Sub'-SoftwareClusters, which thus form a logical group.

7.2.1.1 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). Within the manifest of the SoftwareCluster a version is provided.

7.2.1.2 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. The details of how to relate keys with its values or files shall be defined in the application's manifest but are not defined by this specification and left free for specific implementation. However, the type UploadablePackageElement shall be used as specified in Manifest Specification [6] in order to be compatible with classes specified by Persistency Specification [7]. ⌋*(RS_UCM_00017)*

7.2.1.3 Runtime dependencies

Both 'Sub'- and 'Root'-SoftwareCluster can have runtime dependencies to other SoftwareClusters. Runtime dependencies are checked by UCM in the end of the update process before switching to the new software version. If UCM has to process several Software Packages, then runtime dependencies may not be fulfilled at all times during the Software Packages process.

7.2.2 Transferring Software Packages

The PackageManagement interface offers methods for receiving Software Packages and triggering their processing. The UCM update sequence is clustered into three different phases:

- Software Package transfer (see this section),
- Software Package processing (see 7.2.3) and
- Activation (see 7.2.6).

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. 7.2 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 TransferId that UCM provides to the client.

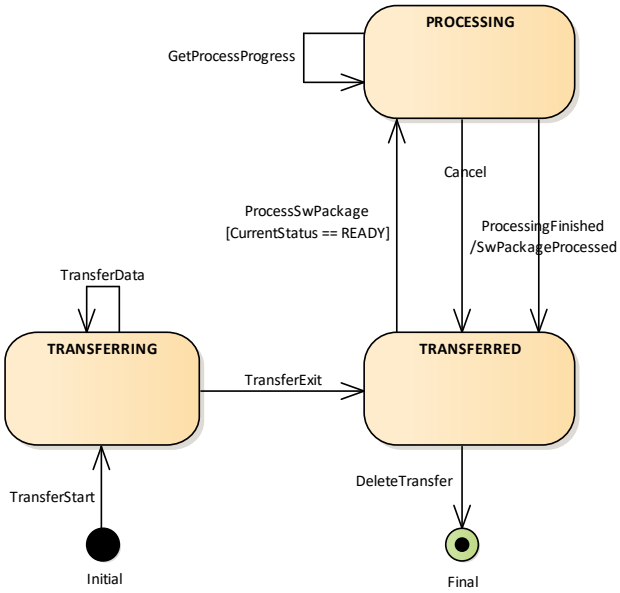

Figure 7.2: State Machine for transferring packages using service interface PackageManagement

[SWS_UCM_00007] Preparation of data transfer ⌈ UCM shall provide support to prepare a data transfer of a Software Package at any time. Preparation of several data transfers shall be supported by UCM. To prepare the data transfer, the method TransferStart shall be used. The method TransferStart shall support a parameter for the size (number of bytes) of the Software Package. Following return values shall be provided by UCM:

- UCM shall set result to Success if UCM is ready to receive the data and the buffer for the Software Package was Successfully allocated in memory.
- UCM shall set result to InsufficientMemory if the buffer could not be allocated in memory.
- In case of result is set to Success UCM shall set the TransferId to a valid value. This TransferId shall be used for all subsequent calls regarding transferring. The state of the Software Package shall be set to Transferring.

⌋(*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`. The method `TransferData` shall support the following parameters

- `id` Returned by the `TransferStart`
- `blockCounter` starting with 0x01 and incremented by one for each subsequent block.
- `data` The payload of the transferred block

Following return values shall be provided by `UCM`:

- `UCM` shall set `result` to `Success` if data has been successfully received and stored in the local buffer.
- `UCM` shall set `result` to `IncorrectBlock` if the block counter does not match with the expected value.
- `UCM` shall set `result` to `IncorrectSize` if the overall size of the stored data for the Software Package exceeds the size previously provided in `TransferStart`.

⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

[SWS_UCM_00010] End of data transfer ⌈ After transmission of all blocks of a software package, the transmission can be finished with method `TransferExit`. `UCM` shall perform following steps 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 set `result` to `InsufficientData`.
- `UCM` shall set `result` to `Success` if the transfer of data could be successfully finished.
- `UCM` shall set `result` to `PackageInconsistent`, if the data integrity check fails.

⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

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

[SWS_UCM_00021] Deleting transferred Software Packages ⌈ `UCM` shall provide a method `DeleteTransfer` that shall delete the targeted Software Package. ⌋ *(RS_UCM_00018)*

For each Software Package the sequence `TransferStart`, `TransferData`, `TransferExit` shall be used. 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 Adaptive Platform) of Software Packages described by the state `Processing` is further detailed in Sect. 7.2.3.

The state of a Software Package can be retrieved by `GetSwPackages`. The method `GetSwPackages` returns all Software Packages that have been successfully transferred and are ready to be processed.

[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)*

7.2.3 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 application manifest.

[SWS_UCM_00001] Starting the package processing ⌈ `UCM` shall provide a method `ProcessSwPackage` to process transferred Software Package. `TransferId` corresponding to Software Package shall be provided for this method. ⌋*(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_00003] Canceling 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 update 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.

7.2.4 Status Reporting

Once Software Packages are transferred to UCM, they are ready to be processed to finally apply changes to the 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 7.2.3 and 7.2.6.

The global state of UCM can be queried using the field CurrentStatus. The state machine for CurrentStatus is shown in Fig. 7.3. In addition to method calls in data transfer phase, at least the sequence of method calls specified in figure 7.3 has to be supported by UCM, further sequences can be supported by vendor-specific implementations.

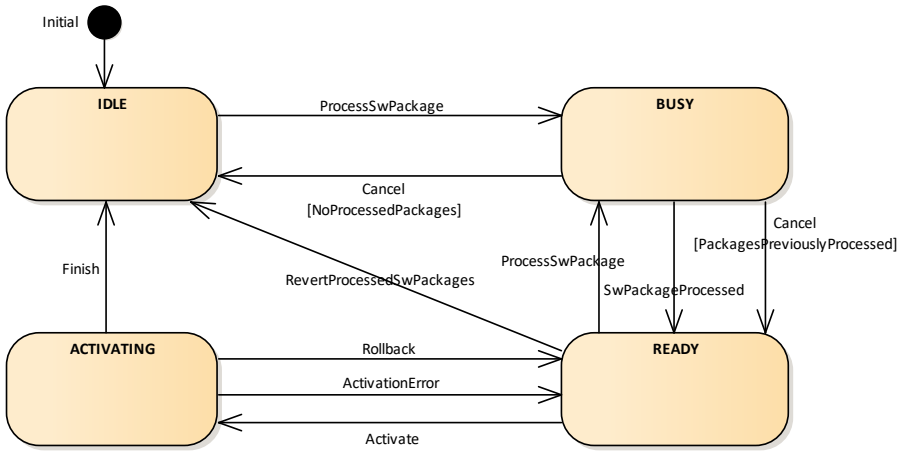

Figure 7.3: State Machine for the package processing using service interface `PackageManagement`

[SWS_UCM_00019] Status Field of Package Management ⌈

UCM shall provide a field CurrentStatus with following possible states.

- Idle This shall be the default state. Once ProcessSwPackage is performed successfully, this state shall only be entered under following conditions:
  - Finish has been called to finish the Software Package processing or
  - Cancel has been called before any other package has been processed or the package processing fails before activating it with Activate.

  - RevertProcessedSwPackages has been called in order to cancel package processing done via ProcessSwPackage before.

- Busy state shall be set if ProcessSwPackage has been called. This shall only be possible, if CurrentStatus reported as Idle or Ready.
- Ready state shall be reported
  - after the package processing is finished successfully or
  - at least one Software Package is processed and Cancel has been called,
  - when Activate fails or an error during Activating happens. In this case, internal UCM triggers ActivationError.
  - when Rollback is performed. This indicates that it is possible to process other Software Packages or revert all processed packages.
- Activating shall be set when Activate is called. This activates temporarily the current processed software for the next restart only. This means in case of a A/B partition for instance, the B partition is temporarily set as active for the next restart only.

  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, the particular Software Cluster 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 runs as expected. This check makes sure, that the system never stays in a non-functional state.

  If the system check fails, for instance because the system is not booting up, the watchdog has to restart the system with A partition again (A/B partition case) or previously active SWCL (no A/B partition case). UCM then detects that activating the processed software has failed and automatically changes the CurrentStatus to Ready by triggering ActivationError.

  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 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, the current CurrentStatus changes to Idle.

⌋(*RS_UCM_00024*)

The value ServiceNotSupported of parameter errorContext of Application Error UCMServiceFailed supported by every method shown in 7.3 can be used to decline a service when the current state is not supporting it. This also applies to all methods for package transmission, e.g. TransferExit before TransferStart is not supported.

[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 `ProcessingResultType` value `Reject` as long as `CurrentStatus` is reported as `Busy`. ⌋*(RS_UCM_00001, RS_UCM_00003, RS_UCM_00026)*

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

[SWS_UCM_00023] Activating State ⌈ Once method `Activate` has been called by a client, further calls to the `ProcessSwPackage` method shall be rejected with `ProcessingResultType` value `Reject`. Calling `ProcessSwPackage` is possible again once `Finish` is called to finalize the current update process. ⌋*(RS_UCM_00021)*

7.2.5 SoftwareCluster lifecycle

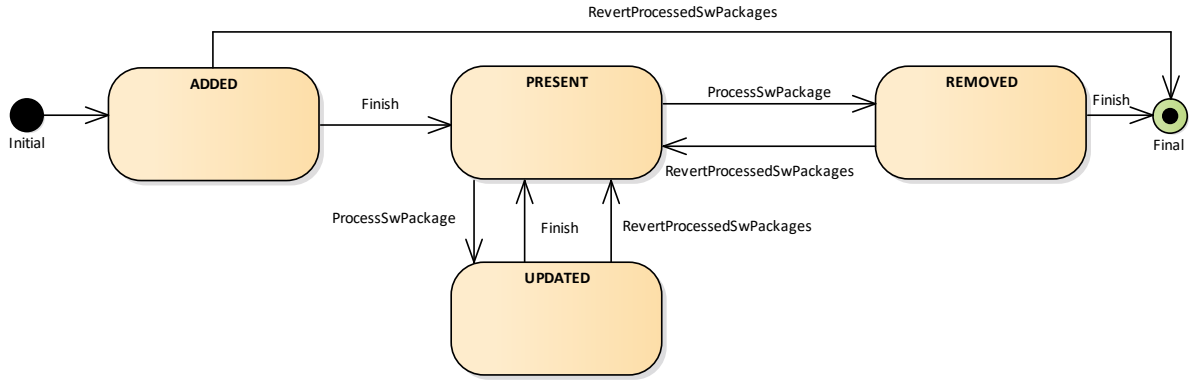

Figure 7.4: State Machine for a `SoftwareCluster`

The state machine in Fig. 7.4 describes the states of a `SoftwareCluster`. After processing a Software Package with a new `SoftwareCluster` that was not yet existing on the adaptive platform, the new `SoftwareCluster` starts its lifecycle with state `Added`. After finishing update process with method `Finish`, it is in state `Present`. In another update process, by processing a Software Package with new data for the `SoftwareCluster`, it changes to `Updated` and returns to `Present` once update process has finished. If a Software Package is processed and it involves the deletion of an existing `SoftwareCluster` the state changes to `Removed`. `Finish` commits the change and the removed `SoftwareCluster` is not reported by `UCM` any more.

7.2.6 Activation and Rollback

The state `Present` does not express whether a `SoftwareCluster` is currently executed or not. The state management on the level of execution is handled by the vendor and platform specific implementation.

UCM has to be able to update several SoftwareCluster for an update campaign. However, these SoftwareCluster could have dependencies not satisfied if one activate or launch them one after the other. Therefore, UCM splits the activation action from the general package processing. UCM shall make sure all dependencies are satisfied before activating a partition or launching a SoftwareCluster. In case of A/B partition, UCM shall not allow to modify any active partition.

[SWS_UCM_00029] Consistency Check of Manifests ⌈ The meta data contained in the Software Package is crucial for a successful software modification and therefore consistency between manifests and processed software shall be checked before SoftwareClusters can be activated. ⌋*(RS_UCM_00016)*

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

[SWS_UCM_00076] Locking of activated software ⌈ UCM shall block any request to modify any SoftwareClusters after Activate has been called. Requests shall be blocked until Rollback or Finish is called. ⌋*(RS_UCM_00021)*

The UCM does not trigger the restart of processed software. This needs to be performed by the client application.

[SWS_UCM_00026] Dependency Check ⌈ At activation, UCM shall check that dependencies described in the SoftwareClusters are all fulfilled. Unfulfilled dependencies shall raise an Application Error with errorContext set to value MissingDependencies. ⌋*(RS_UCM_00007)*

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

[SWS_UCM_00027] Notification of Activation or Rollback ⌈ UCM shall notify other Functional Clusters of the 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.

7.2.7 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 processing. It could also remove Software Packages, logs or any older versions of changed software to save storage space. ⌋*(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.

7.2.8 Logging

[SWS_UCM_00012] Logging ⌈ UCM shall provide a method GetLog to retrieve all log messages that have been stored by UCM for specific TransferId. UCM shall provide a method SetLogLevel to provide a log level for all subsequent log messages that are stored by UCM. GetLog and SetLogLevel shall be provided only in the context of a TransferId, otherwise an Application Error shall be raised. ⌋*(RS_UCM_00022)*

7.2.9 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 Active. ⌋ *(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 Added, Updated or Removed. ⌋*(RS_UCM_00011)*

7.2.10 Securing Software Updates

UCM provides service interface using ara::com. There is no authentication step in UCM's update sequence. Instead it is up to the Identity and Access Management [4] to secure the access to the UCM service interface. This way, only authorized applications are able to access UCM's services.

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

Content of the manifest might contain signatures of the Software Package and ProcessSwPackage uses of checksum algorithms, cryptographic signatures or other vendor and/or OEM specific mechanisms to validate the package.

[SWS_UCM_00074] Failure of authentication ⌈ When authentication of Software Package fails, UCM shall raise an Application Error with errorContext set to value AuthenticationFailed. ⌋*(RS_UCM_00006)*

8 API specification

8.1 Type definitions

This chapter lists the types provided by the UCM.

8.1.1 TransferIdType

Name	TransferIdType
Kind	Type
Derived from	uint32
Description	Represents a handle identifier used to reference a particular transfer request.

Table 8.1: Definition of TransferIdType

[SWS_UCM_00031] TransferIdType table ⌈ Table 8.1 describes the type `TransferIdType`.⌋ *(RS_UCM_00019, RS_UCM_00025)*

8.1.2 SwInfoName

Name	SwInfoName
Kind	Type
Derived from	string
Description	Represents a name identifying a `SoftwareCluster`

Table 8.2: Definition of SwInfoName

[SWS_UCM_00071] SwInfoName table ⌈ Table 8.2 describes the type `SwInfoName`.⌋ *(RS_UCM_00002)*

8.1.3 ByteVectorType

Name	ByteVectorType
Kind	Type
Derived from	Vector of type uint8
Description	Byte vector representing raw data.

Table 8.3: Definition of ByteVectorType

[SWS_UCM_00032] ByteVectorType table ⌈ Table 8.3 describes the type `ByteVectorType`.⌋*(RS_UCM_00025)*

8.1.4 TransferStartSuccessType and TransferStartReturnType

Name	TransferStartSuccessType		
Kind	Type		
Derived from	uint8		
Description	Represents the return type of the TransferStart method.		
Range	kSuccess	0x00	Transfer of data has been successfully initiated.
	kInsufficientMemory	0x01	Not enough memory to store the Software Package.
	kReserved	0x02-0xFF	Reserved for future use.

Table 8.4: Definition of TransferStartSuccessType

[SWS_UCM_00033] TransferStartSuccessType table ⌈ Table 8.4 describes the type `TransferStartSuccessType`.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

Name	TransferStartReturnType		
Kind	Struct		
Description	Represents the return struct for TransferStart.		
Members	***Name***	***Type***	***Description***
	TransferId	TransferId Type	The TransferId generated by `UCM` for this transfer
	TransferStartSuccess	Transfer Start Success Type	The result of the transfer start operation.

Table 8.5: Definition of TransferStartReturnType

[SWS_UCM_00034] TransferStartReturnType table ⌈ Table 8.5 describes the type `TransferStartReturnType`.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

8.1.5 TransferDataReturnType

Name	TransferDataReturnType

Kind	Type		
Derived from	uint8		
Description	Represents the return type of the TransferData method.		
Range	kSuccess	0x00	Data has been successfully transmitted and stored.
	kIncorrectBlock	0x01	The block counter value is not as expected by UCM.
	kIncorrectSize	0x02	The size of the Software Package exceeds the provided size in TransferStart.
	kInsufficientMemory	0x03	Not enough memory to store the Software Package.
	kInvalidTransferId	0x04	The Transfer ID is invalid.
	kOperationNotPermitted	0x05	The operation is not permitted in the current state.
	kReserved	0x06-0xFF	Reserved for future use.

Table 8.6: Definition of TransferDataReturnType

[SWS_UCM_00035] TransferDataReturnType table ⌈ Table 8.6 describes the type `TransferDataReturnType`.⌋ *(RS_UCM_00013, RS_UCM_00014, RS_UCM_00019, RS_UCM_00025)*

8.1.6 TransferExitReturnType

Name	TransferExitReturnType
Kind	Type
Derived from	uint8
Description	Represents the return type of the TransferExit method.

Range	kSuccess	0x00	Data transmission has been successfully finished. The complete Software Package has been received.
	kInsufficientData	0x01	TransferExit has been called but total transferred data size does not match expected data size provided with TransferStart call.
	kPackageInconsistent	0x02	Transferred package integrity or authentication check failed.
	kInvalidTransferId	0x03	The Transfer ID is invalid.
	kOperationNotPermitted	0x04	The operation is not permitted in the current state.
	kReserved	0x05-0xFF	Reserved for future use.

Table 8.7: Definition of TransferExitReturnType

[SWS_UCM_00036] TransferExitReturnType table ⌈ Table 8.7 describes the type `TransferExitReturnType`.⌋*(RS_UCM_00006, RS_UCM_00012, RS_UCM_00013, RS_UCM_00014, RS_UCM_00016, RS_UCM_00019, RS_UCM_00025)*

8.1.7 ProcessResultType and ProcessSwPackageReturnType

Name	ProcessResultType
Kind	Type
Derived from	uint8
Description	Represents the result of processing one or several software packages.

Range	kSuccess	0x00	Software package has been successfully processed.
	kReject	0x01	Another processing is already ongoing and therefore the current processing request has to be rejected.
	kInvalidManifest	0x02	Package manifest could not be read.
	kInsufficientMemory	0x03	Insufficient memory to perform processing.
	kInvalidTransferId	0x04	The Transfer ID is invalid.
	kOperationNotPermitted	0x05	The operation is not permitted in the current state.
	kErrorAuthenticationFailed	0x06	Authentication of the software package failed.
	kReserved	0x07-0xFF	Reserved for future use.

Table 8.8: Definition of ProcessResultType

[SWS_UCM_00037] ProcessResultType table ⌈ Table 8.8 describes the type `ProcessResultType`.⌋*(RS_UCM_00001, RS_UCM_00003, RS_UCM_00004, RS_UCM_00016, RS_UCM_00018, RS_UCM_00024)*

Name	ProcessSwPackageReturnType		
Kind	Struct		
Description	Represents the return struct for ProcessSwPackageReturnType.		
Members	***Name***	***Type***	***Description***
	result	Process Result Type	The result of the process operation
	transferId	TransferId Type	The transfer ID of the processed Software Package

Table 8.9: Definition of ProcessSwPackageReturnType

[SWS_UCM_00072] ProcessSwPackageReturnType table ⌈ Table 8.9 describes the type `ProcessSwPackageReturnType`.⌋*(RS_UCM_00001, RS_UCM_00003, RS_UCM_00004, RS_UCM_00016, RS_UCM_00018, RS_UCM_00024)*

8.1.8 SwPackageInfoStateType

Name	SwPackageInfoStateType		
Kind	Type		
Derived from	uint8		
Description	Represents the state of a Software Package on the Platform.		
	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.

Table 8.10: Definition of SwPackageInfoStateType

[SWS_UCM_00038] SwPackageInfoStateType table ⌈ Table 8.10 describes the type `SwPackageInfoStateType`.⌋(*RS_UCM_00002*, *RS_UCM_00006*, *RS_UCM_00010*, *RS_UCM_00011*, *RS_UCM_00012*)

8.1.9 SwPackageInfoType and SwPackageInfoVectorType

Name	SwPackageInfoType		
Kind	Struct		
Description	Represents the information of a Software Package		
Members	**Name**	**Type**	**Description**
	Name	SwInfoName	Name of the Software Package
	Version	String	Version of the Software Package
	TransferID	TransferIdType	TransferID of the Software Package
	State	SwPackageInfo StateType	State of the Software Package

Table 8.11: Definition of SwPackageInfoType

[SWS_UCM_00039] SwPackageInfoType table ⌈ Table 8.11 describes the type `SwPackageInfoType`.⌋(*RS_UCM_00002*, *RS_UCM_00006*, *RS_UCM_00010*, *RS_UCM_00011*, *RS_UCM_00012*)

Name	SwPackageInfoVectorType
Kind	Type
Derived from	Vector of type SwPackageInfoType

Description	Represents a list of Software Packages

Table 8.12: Definition of SwPackageInfoVectorType

[SWS_UCM_00040] SwPackageInfoVectorType table ⌈ Table 8.12 describes the type `SwPackageInfoVectorType`.⌋*(RS_UCM_00002, RS_UCM_00006, RS_UCM_00010, RS_UCM_00011, RS_UCM_00012)*

8.1.10 SwClusterInfoStateType

Name	SwClusterInfoStateType		
Kind	Type		
Derived from	uint8		
Description	Represents the state of a `SoftwareCluster` on the adaptive platform.		
	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 but not yet activated.
	kRemoved	0x03	State of a `SoftwareCluster` that has been removed.
	kReserved	0x04-0xFF	Reserved for future use.

Table 8.13: Definition of SwClusterInfoStateType

[SWS_UCM_00077] SwClusterInfoStateType table ⌈ Table 8.13 describes the type `SwClusterInfoStateType`.⌋*(RS_UCM_00002, RS_UCM_00006, RS_UCM_00010, RS_UCM_00011, RS_UCM_00012)*

8.1.11 SwClusterInfoType and SwClusterInfoVectorType

Name	SwClusterInfoType
Kind	Struct

Description	Represents the information of a `SoftwareCluster`		
Members	***Name***	***Type***	***Description***
	Name	SwInfoName	Name of the `SoftwareCluster`
	Version	String	Version of the `SoftwareCluster`
	State	SwClusterInfo StateType	State of the `SoftwareCluster`

Table 8.14: Definition of SwClusterInfoType

[SWS_UCM_00078] SwClusterInfoType table ⌈ Table 8.14 describes the type `SwClusterInfoType`.⌋(*RS_UCM_00002*, *RS_UCM_00006*, *RS_UCM_00010*, *RS_UCM_00011*, *RS_UCM_00012*)

Name	SwClusterInfoVectorType
Kind	Type
Derived from	Vector of type SwClusterInfoType
Description	Represents a list of `SoftwareClusters`

Table 8.15: Definition of SwClusterInfoVectorType

[SWS_UCM_00079] SwClusterInfoVectorType table ⌈ Table 8.15 describes the type `SwClusterInfoVectorType`.⌋(*RS_UCM_00002*, *RS_UCM_00006*, *RS_UCM_00010*, *RS_UCM_00011*, *RS_UCM_00012*)

8.1.12 LogLevel, LogEntryType and LogType

Name	LogLevelType
Kind	Type
Derived from	uint8
Description	Represents the state of a Software Package on the Platform.

Range	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.
	kReserved	0x07-0xFF	Reserved for future use.

Table 8.16: Definition of LogLevelType

[SWS_UCM_00041] LogLevelType table ⌈ Table 8.16 describes the type `LogLevelType`.⌋*(RS_UCM_00022)*

Name	LogEntryType		
Kind	Struct		
Description	Represents a single log message with a log level.		
Members	***Name***	***Type***	***Description***
	LogLevel	uint8	The log level of the log message.
	Message	String	The log message.

Table 8.17: Definition of LogEntryType

[SWS_UCM_00042] LogEntryType table ⌈ Table 8.17 describes the type `LogEntryType`.⌋*(RS_UCM_00022)*

Name	LogType
Kind	Type
Derived from	Vector of LogEntryType
Description	Represents a list of log messages.

Table 8.18: Definition of LogType

[SWS_UCM_00043] LogType table ⌈ Table 8.18 describes the type `LogType`.⌋ *(RS_UCM_00022)*

8.1.13 PackageManagerStatusType

Name	PackageManagerStatusType		
Kind	Type		
Derived from	uint8		
Description	Represents the state of UCM.		
Range	kIdle	0x00	UCM is ready to receive software packages or 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.
	kReserved	0x04-0xFF	Reserved for future use.

Table 8.19: Definition of PackageManagerStatusType

[SWS_UCM_00044] PackageManagerStatusType table ⌈ Table 8.19 describes the type PackageManagerStatusType.⌋*(RS_UCM_00024, RS_UCM_00026)*

8.1.14 ActivateReturnType

Name	ActivateReturnType
Kind	Type
Derived from	uint8
Description	Represents the return type of the Activate method.

Range	kSuccess	0x00	Activation successful.
	kErrorNoProcessedPackages	0x01	Activate failed because no packages where processed.
	kErrorDuringActivation	0x02	Activate failed.
	kErrorNoValidProcessing	0x03	Activate cannot be performed because previous processing is invalid.
	kMissingDependencies	0x04	Activate cannot be performed because of missing dependencies.
	kReserved	0x05-0xFF	Reserved for future use.

Table 8.20: Definition of ActivateReturnType

[SWS_UCM_00046] ActivateReturnType table ⌈ Table 8.20 describes the type `ActivateReturnType`.⌋*(RS_UCM_00008, RS_UCM_00021)*

8.1.15 CancelReturnType

Name	CancelReturnType		
Kind	Type		
Derived from	uint8		
Description	Represents the return type of the Cancel method.		
Range	kSuccess	0x00	Cancel successful.
	kError	0x01	Cancel failed.
	kOperationNotPermitted	0x02	The operation is not permitted in the current state.
	kReserved	0x03-0xFF	Reserved for future use.

Table 8.21: Definition of CancelReturnType

[SWS_UCM_00047] CancelReturnType table ⌈ Table 8.21 describes the type `CancelReturnType`.⌋*(RS_UCM_00020)*

8.1.16 RevertProcessedSwPackagesReturnType

Name	RevertProcessedSwPackagesReturnType
Kind	Type

Derived from	uint8		
Description	Represents the return type of the RevertProcessedSwPackages method.		
Range	kSuccess	0x00	Packages have been reverted successfully.
	kNothingToRevert	0x01	RevertProcessedSwPackages has been called without prior processing of a Software Package.
	kNotAbleToRevertPackages	0x02	RevertProcessedSwPackages failed.
	kReserved	0x03-0xFF	Reserved for future use.

Table 8.22: Definition of RevertProcessedSwPackagesReturnType

[SWS_UCM_00048] RevertProcessedSwPackagesReturnType table ⌈ Table 8.22 describes the type `RevertProcessedSwPackagesReturnType`.⌋ *(RS_UCM_00018)*

8.1.17 RollbackReturnType

Name	RollbackReturnType		
Kind	Type		
Derived from	uint8		
Description	Represents the return type of the Rollback method.		
Range	kSuccess	0x00	Rollback successful.
	kNothingToRollback	0x01	Rollback cannot be performed due to no rollback data available.
	kNotAbleToRollback	0x02	Rollback failed.
	kReserved	0x03-0xFF	Reserved for future use.

Table 8.23: Definition of RollbackReturnType

[SWS_UCM_00049] RollbackReturnType table ⌈ Table 8.23 describes the type `RollbackReturnType`.⌋ *(RS_UCM_00008)*

8.1.18 GeneralResponseType

Name	GeneralResponseType
Kind	Type

Derived from	uint8		
Description	General response type used for SetLogLevel.		
Range	kSuccess	0x00	Request successful.
	kGeneralReject	0x01	General reject.
	kGeneralMemoryError	0x02	A general memory error occured.
	kTransferIdInvalid	0x03	The transfer ID parameter is invalid.
	kOperationNotPermitted	0x04	The operation is not supported in the current context.
	kReserved	0x05-0xFF	Reserved for future use.

Table 8.24: Definition of GeneralResponseType

[SWS_UCM_00050] GeneralResponseType table ⌈ Table 8.24 describes the type `GeneralResponseType`.⌋*(RS_UCM_00015)*

8.1.19 FinishReturnType

Name	FinishReturnType		
Kind	Type		
Derived from	uint8		
Description	General response type used for SetLogLevel.		
Range	kSuccess	0x00	General reject.
	kUndefinedError	0x01	A general memory error occured.
	kOperationNotPermitted	0x02	The operation is not permitted in the current state.
	kReserved	0x03-0xFF	Reserved for future use.

Table 8.25: Definition of FinishReturnType

[SWS_UCM_00051] FinishReturnType table ⌈ Table 8.25 describes the type `FinishReturnType`.⌋*(RS_UCM_00015)*

8.2 Ports and Service Interfaces

Name	PackageManagement		
Kind	ProvidedPort	***Interface***	PackageManagement

Description	Provides access to package management.

Table 8.26: Port - PackageManagement

[SWS_UCM_00073] ProvidedPort PackageManagement ⌈Table 8.26 describes the provided port `PortPackageManagement`.⌋*(RS_UCM_00001)*

8.2.1 PackageManagement

This service interface offers methods for basic package management tasks.

8.2.1.1 Methods

Name	GetSwClusterInfo		
Description	This method returns a list of `SoftwareCluster`s that are in state kPresent.		
Parameters	SwInfo	**Description**	List of installed `SoftwareCluster`s that are in state kPresent.
		Type	SwClusterInfoVectorType
		Direction	OUT

Table 8.27: Method signature for GetSwClusterInfo

[SWS_UCM_00052] PackageManager GetSwClusterInfo ⌈Table 8.27 describes the interface `GetSwClusterInfo`.⌋*(RS_UCM_00002)*

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

Table 8.28: Method signature for GetSwClusterChangeInfo

[SWS_UCM_00053] PackageManager GetSwClusterChangeInfo ⌈Table 8.28 describes the interface `GetSwClusterChangeInfo`.⌋*(RS_UCM_00011)*

Name	GetSwPackages		
Description	This method returns the Software Packages that available in `UCM`		
Parameters	Packages	***Description***	List of Software Packages.
		Type	SwPackageInfoVectorType
		Direction	OUT

Table 8.29: Method signature for GetSwPackages

[SWS_UCM_00054] PackageManager GetSwPackages ⌈Table 8.29 describes the interface `GetSwPackages`.⌋*(RS_UCM_00010)*

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 and SetLogLevel.		
Parameters	size	***Description***	Size (in bytes) of the Software Package to be transferred.
		Type	uint32
		Direction	IN
	result	***Description***	Return value of the method.
		Type	TransferStartReturnType
		Direction	OUT

Table 8.30: Method signature for TransferStart

[SWS_UCM_00055] PackageManager TransferStart ⌈Table 8.30 describes the interface `TransferStart`.⌋*(RS_UCM_00013, RS_UCM_00019, RS_UCM_00025)*

Name	TransferData
Description	Block-wise transfer of a Software Package to UCM.

Parameters	id	***Description***	Transfer ID.
		Type	TransferIdType
		Direction	IN
	data	***Description***	Data block of the Software Package.
		Type	ByteVectorType
		Direction	IN
	blockCounter	***Description***	Block counter value of the current block.
		Type	uint32
		Direction	IN
	result	***Description***	Return value of the method.
		Type	TransferDataReturnType
		Direction	OUT

Table 8.31: Method signature for TransferData

[SWS_UCM_00056] PackageManager TransferData ⌈Table 8.31 describes the interface `TransferData`.⌋*(RS_UCM_00013, RS_UCM_00014, RS_UCM_00019, RS_UCM_00025)*

Name	TransferExit		
Description	Finish the transfer of a Software Package to UCM.		
Parameters	id	***Description***	Transfer ID of the currently running request.
		Type	TransferIdType
		Direction	IN
	result	***Description***	Return value of the method.
		Type	TransferExitReturnType
		Direction	OUT

Table 8.32: Method signature for TransferExit

[SWS_UCM_00057] PackageManager TransferExit ⌈Table 8.32 describes the interface `TransferExit`.⌋*(RS_UCM_00006, RS_UCM_00012, RS_UCM_00013, RS_UCM_00014, RS_UCM_00016, RS_UCM_00019, RS_UCM_00025)*

Name	DeleteTransfer
Description	Delete a transferred Software Package.

Parameters	id	***Description***	Transfer ID of the currently running request.
		Type	TransferIdType
		Direction	IN
	result	***Description***	Return value of the method.
		Type	GeneralResponseType
		Direction	OUT

Table 8.33: Method signature for DeleteTransfer

[SWS_UCM_00058] PackageManager DeleteTransfer ⌈Table 8.33 describes the interface `DeleteTransfer`.⌋*(RS_UCM_00015)*

Name	ProcessSwPackage		
Description	Process a previously transferred Software Package.		
Parameters	id	***Description***	The Transfer ID of this Software Package.
		Type	TransferIdType
		Direction	IN
	result	***Description***	Result of the processing request.
		Type	ProcessSwPackage ReturnType
		Direction	OUT

Table 8.34: Method signature for ProcessSwPackage

[SWS_UCM_00059] PackageManager ProcessSwPackage ⌈Table 8.34 describes the interface `ProcessSwPackage`.⌋*(RS_UCM_00001, RS_UCM_00003, RS_UCM_00004, RS_UCM_00016, RS_UCM_00018, RS_UCM_00026)*

Name	RevertProcessedSwPackages		
Description	Revert the changes done by processing (ProcessSwPackage) of one or several software packages.		
Parameters	result	***Description***	The result of the RevertProcessedSwPackages call.
		Type	RevertProcessedSw PackagesReturnType
		Direction	OUT

Table 8.35: Method signature for RevertProcessedSwPackages

[SWS_UCM_00060] PackageManager RevertProcessedSwPackages ⌈Table 8.35 describes the interface `RevertProcessedSwPackages`.⌋*(RS_UCM_00018)*

Name	GetSwProcessProgress		
Description	Get the progress (0 - 100%) of the currently processed Software Package.		
Parameters	id	***Description***	The Transfer ID of the Software Package.
		Type	TransferIdType
		Direction	IN
	progress	***Description***	The progress of the current package processing (0% - 100%). 0x00 ... 0x64, 0xFF for "'No information available"'
		Type	uint8
		Direction	OUT

Table 8.36: Method signature for GetSwProcessProgress

[SWS_UCM_00061] PackageManager GetSwProcessProgress ⌈Table 8.36 describes the interface `GetSwProcessProgress`.⌋*(RS_UCM_00023)*

Name	Cancel		
Description	This method aborts an ongoing processing of a Software Package.		
Parameters	id	***Description***	The Transfer ID.
		Type	TransferIdType
		Direction	IN
	result	***Description***	Result of the cancel request.
		Type	CancelReturnType
		Direction	OUT

Table 8.37: Method signature for Cancel

[SWS_UCM_00062] PackageManager Cancel ⌈Table 8.37 describes the interface `Cancel`.⌋*(RS_UCM_00020)*

Name	Rollback		
Description	Rollback the system to the state before the packages were processed.		
Parameters	result	***Description***	The result of the rollback operation.
		Type	RollbackReturnType
		Direction	OUT

Table 8.38: Method signature for Rollback

[SWS_UCM_00063] PackageManager Rollback ⌈Table 8.38 describes the interface `Rollback`.⌋*(RS_UCM_00008)*

Name	Activate		
Description	This method activates the processed components.		
Parameters	result	***Description***	The result of the activate call.
		Type	ActivateReturnType
		Direction	OUT

Table 8.39: Method signature for Activate

[SWS_UCM_00064] PackageManager Activate ⌈Table 8.39 describes the interface `Activate`.⌋*(RS_UCM_00008, RS_UCM_00021)*

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.		
Parameters	result	***Description***	The result of the finish call.
		Type	FinishReturnType
		Direction	OUT

Table 8.40: Method signature for Finish

[SWS_UCM_00065] PackageManager Finish ⌈Table 8.40 describes the interface `Finish`.⌋*(RS_UCM_00015)*

Name	SetLogLevel		
Description	This method sets the log level for the internal logging of the update process.		
Parameters	id	***Description***	The Transfer ID.
		Type	TransferIdType
		Direction	IN
	logLevel	***Description***	The new log level to be used.
		Type	LogLevelType
		Direction	IN

Table 8.41: Method signature for SetLogLevel

[SWS_UCM_00066] PackageManager ⌈Table 8.41 describes the interface `SetLogLevel`.⌋*(RS_UCM_00022)*

Name	GetLog
Description	Getter method to poll for the log messages of the current Session.

Parameters	id	***Description***	The Transfer ID.
		Type	TransferIdType
		Direction	IN
	log	***Description***	The log messages.
		Type	LogType
		Direction	OUT

Table 8.42: Method signature for GetLog

[SWS_UCM_00067] PackageManager GetLog ⌈Table 8.42 describes the interface `GetLog`.⌋*(RS_UCM_00022)*

8.2.1.2 Fields

Name	CurrentStatus
Description	The current status of `UCM`.
Type	PackageManagerStatusType
HasGetter	yes
HasSetter	no
HasNotifier	yes

Table 8.43: Field properties for CurrentStatus

[SWS_UCM_00068] Field CurrentStatus ⌈Table 8.43 describes the field `CurrentStatus`.⌋*(RS_UCM_00018)*

9 Sequence diagrams

9.1 Update process

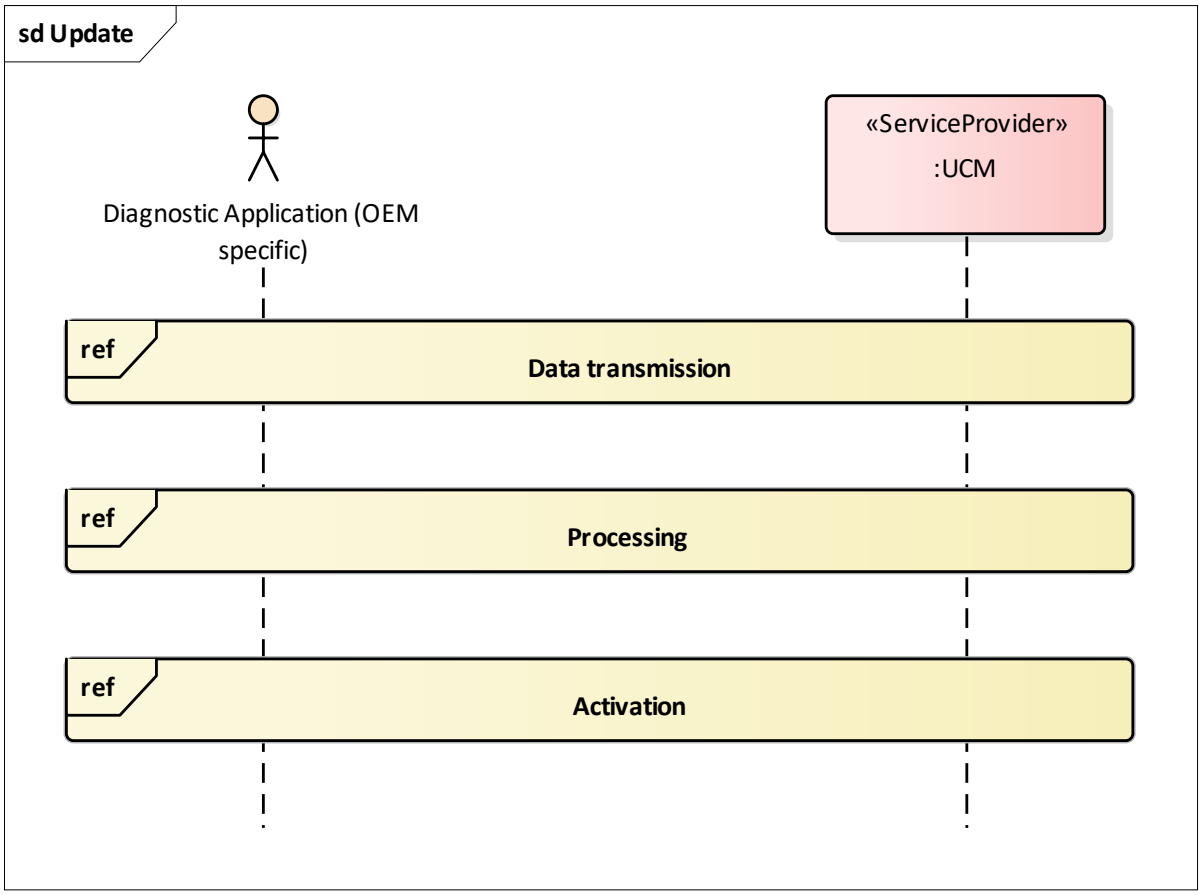

Figure 9.1: Sequence diagram showing the update process

9.2 Data transmission

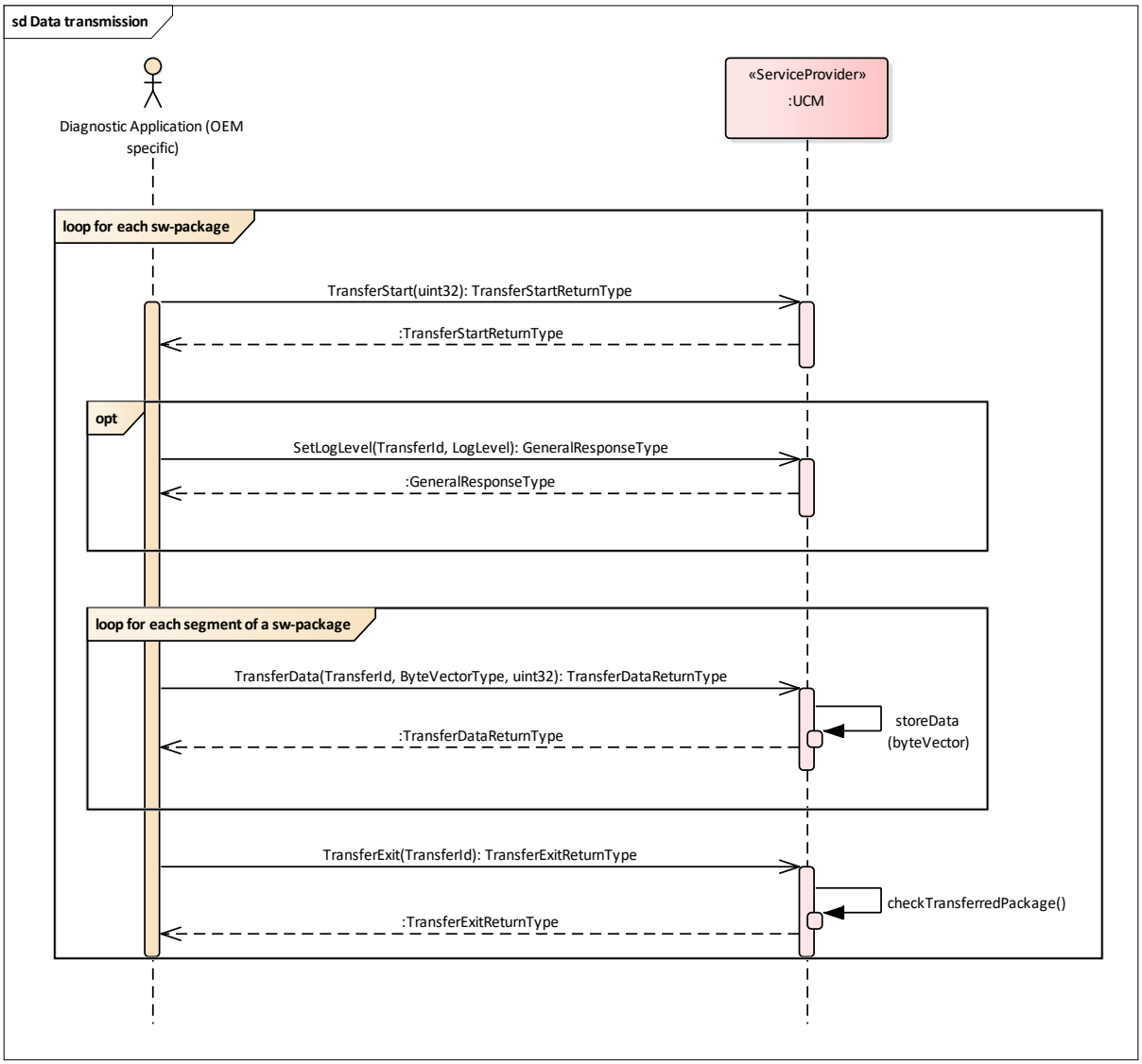

Figure 9.2: Sequence diagram showing the data transmission

9.3 Package processing

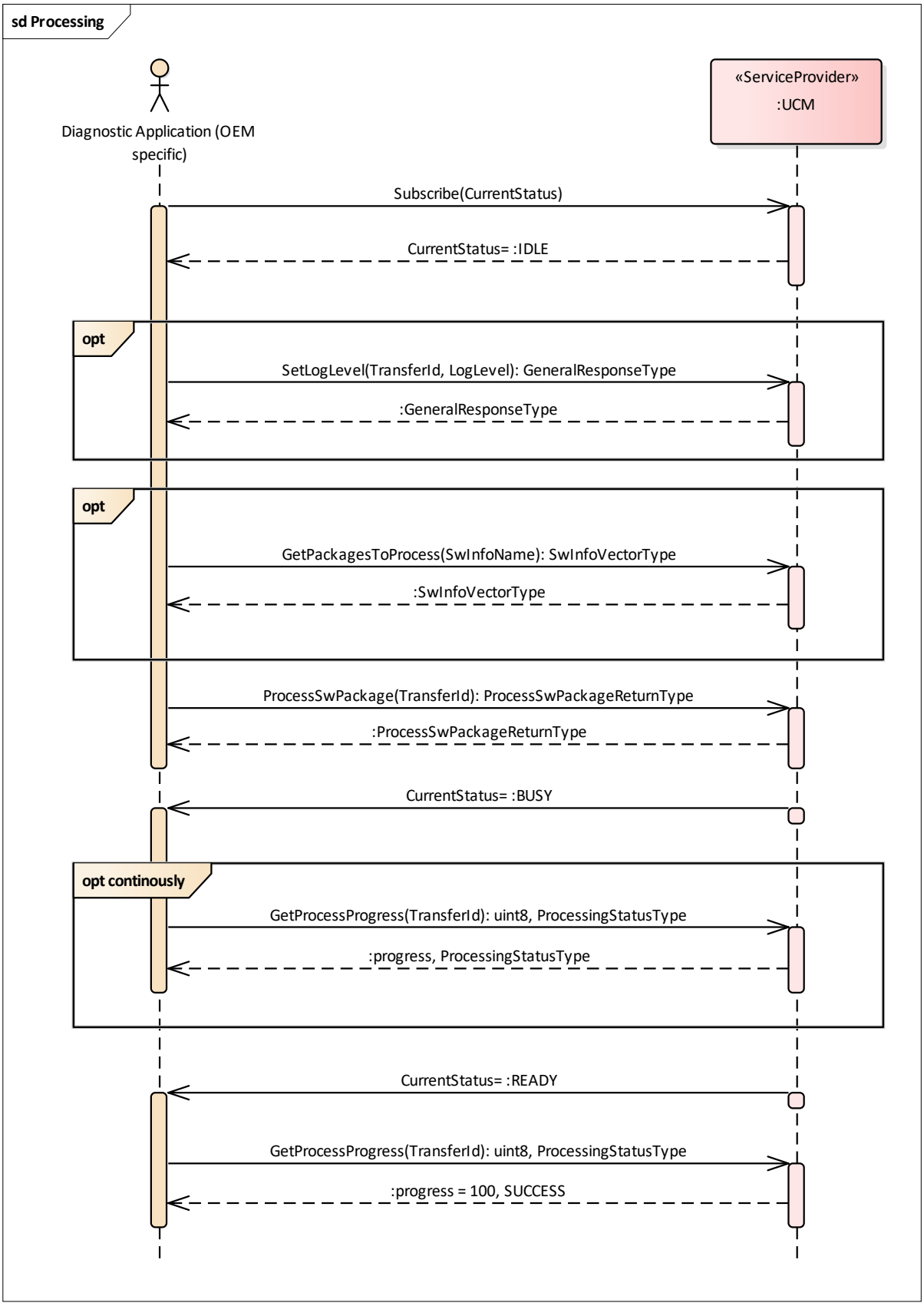

Figure 9.3: Sequence diagram showing the package processing

9.4 Activation

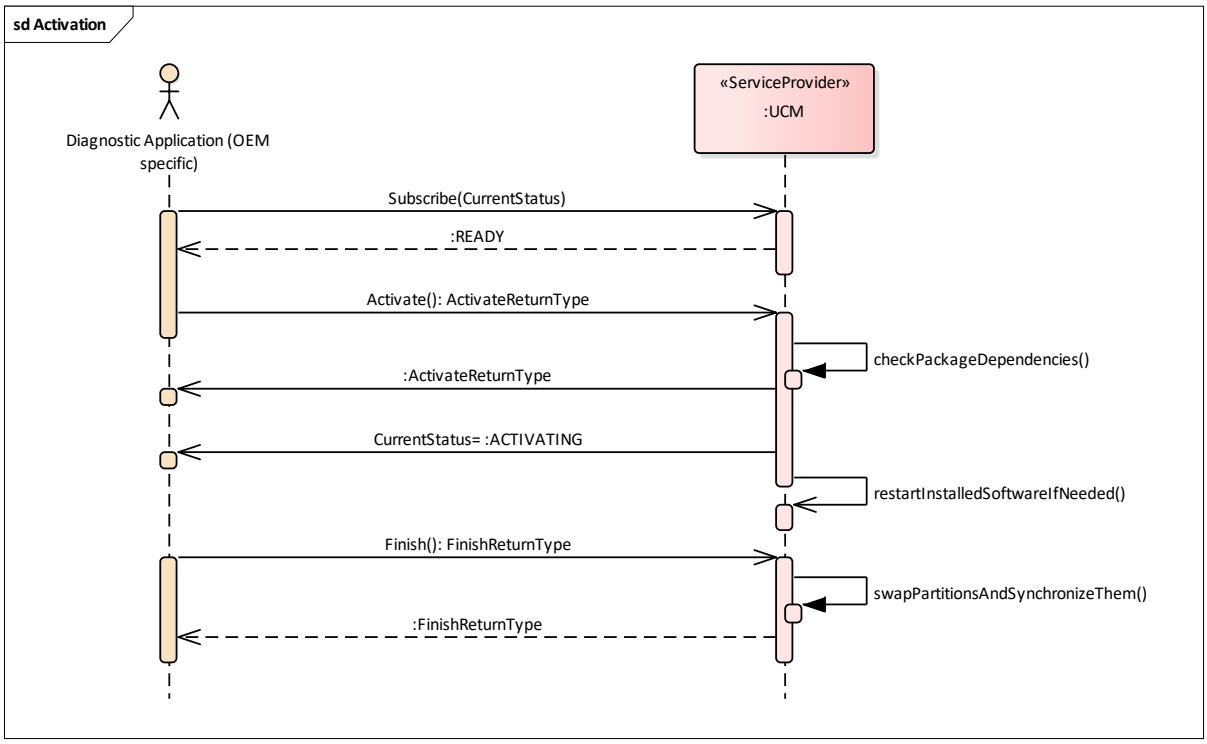

Figure 9.4: Sequence diagram showing the activation process

A Interfaces to other Functional Clusters (informative)

A.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. Additional interfaces, parameters and return values can be added.

A.2 Interfaces Tables

A.2.1 Interfaces to Execution Management

As described in chapter 5 UCM uses interfaces to Execution Management [2] to modify machine state. Please refer to this document for definition of the used interfaces.

In addition to the interfaces provided in [2], the following functions are required.

	Name	Description	Requirements
Target	Execution Management		
Name proposal	*StopProcess*		
Functionality	Request to stop a process	Stop a specific process on the request from the Update and Configuration Management.	
Parameters (in)	process identifier	Unique identifier of the process to be stopped.	
Parameters (inout)	None		
Parameters (out)	None		
Return value	Operation succeeded		
	general error	process could not be stopped	

Table A.1: Process Stop Request

	Name	Description	Requirements
Target	Execution Management		
Name proposal	*StartProcess*		
Functionality	Request to start a process	Start a specific process on the request from the Update and Configuration Management.	
Parameters (in)	process identifier	Unique identifier of the process to be started.	
Parameters (inout)	None		
Parameters (out)	None		
Return value	Operation succeeded		
	general error	process could not be started	

Table A.2: Process Start Request

A.2.2 Interfaces to Adaptive Crypto Interface

UCM uses Crypto Interface for Adaptive Platform [8] to verify package signatures and to decrypt encrypted update data.

A.2.3 Interfaces to Identity and Access Management

UCM uses Identity and Access Management [9] to validate the authorization of requests.