Document Title	Specification of Vehicle Update and Configuration Management
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	1090

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

Document Change History			
Date	**Release**	**Changed by**	**Description**
2024-11-27	R24-11	AUTOSAR Release Management	• Refactored progress monitoring • Adapted V-UCM to UCM new suspend and resume requirement
2023-11-23	R23-11	AUTOSAR Release Management	• Initial release from split of SWS UCM document • Vehicle dependency handling • VSM interface refactoring

Disclaimer

Contents

1 Introduction and functional overview

This specification describes the functionality, API and the configuration for the AUTOSAR functional cluster Vehicle Update and Configuration Management which belongs to the AUTOSAR Adaptive Platform Services. One of the declared goals of the AUTOSAR Adaptive Platform is the ability to flexibly update the software and its configuration through local ("tester-based") or remote ("over-the-air") updates. V-UCM provides services for updating the software and its configuration in a vehicle. V-UCM is coordinating an update campaign within the vehicle. Therefore this document includes requirements on the following functionalities:

- Interact with Backend to:
  - Identify Software Clusters that could be updated, installed or removed
  - Authenticate Vehicle Package
  - Confirm dependencies between Software Clusters within vehicle before starting campaign
  - Inform Backend of needed Software Packages, receives them and dispatch them to targeted ECUs
- Interact with Vehicle State Manager:
  - Inform which safety conditions that have to be applied according to Vehicle Package
  - Share the computed vehicle state to other Applications or Functional Clusters involved in the update campaign
- Interact with Human Driver about update campaign:
  - Provides campaign state to trigger interaction with Human during update campaign
  - Get vehicle modification approval or consent from Human when configured in Vehicle Package
- Provide information of installed software in vehicle
- Provide information of update campaigns history
- Recovery in case of failure

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the V-UCM module that are not included in the AUTOSAR TR Glossary.

Abbreviation / Acronym:	Description:
DM	AUTOSAR Adaptive Diagnostic Management
UCM	Update and Configuration Management
V-UCM	V-UCM is distributing packages and coordinating an update campaign in a vehicle (former name was UCM Master)
Backend	Backend is a server hosting Software Packages
OTA Client	OTA Client is an Adaptive Application in communication with Backend Over The Air
Vehicle Driver Application	Vehicle Driver Application is an Adaptive Application in communication with Vehicle Driver Human to Machine Interface
Application Error	Errors returned by UCM
Boot options	Boot Manager Configuration
VCI	Vehicle Communication Interface
MVCI	Modular Vehicle Communication Interface
D-PDU API	Diagnostic Protocol Data Unit Application Programming Interface
RDF	Root Description File
MDF	Module Description File
integrity check	verification method proving there has not been any alteration of the artefact content
dependency check	verification method proving that all configured dependencies in vehicle will be fulfilled before transfer of Software Packages.

Table 2.1: Acronyms and Abbreviations

Below acronyms and abbreviations relevant for this document are included in the AUTOSAR TR Glossary. This is to avoid duplicate definition of the technical term. And to refer to the correct document.

Term	Description:
Adaptive Application	see AUTOSAR TR Glossary
AUTOSAR Adaptive Platform	see AUTOSAR TR Glossary
AUTOSAR Classic Platform	see AUTOSAR TR Glossary
Functional Cluster	see AUTOSAR TR Glossary
Service	see AUTOSAR TR Glossary
Electronic Control Unit	see AUTOSAR TR Glossary
Machine	see AUTOSAR TR Glossary
Manifest	see AUTOSAR TR Glossary
Software Package	see AUTOSAR TR Glossary
Software Cluster	see AUTOSAR TR Glossary
Vehicle Package	see AUTOSAR TR Glossary
Vehicle State Manager	see [1] AUTOSAR Glossary

Table 2.2: Reference to Technical Terms

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] General Requirements specific to Adaptive Platform
AUTOSAR_AP_RS_General

[3] Explanation of Adaptive Platform Software Architecture
AUTOSAR_AP_EXP_SWArchitecture

[4] Requirements on Vehicle Update and Configuration Management
AUTOSAR_AP_RS_VehicleUpdateAndConfigurationManagement

[5] Specification of Update and Configuration Management
AUTOSAR_AP_SWS_UpdateAndConfigurationManagement

[6] Explanation of Adaptive Platform Design
AUTOSAR_AP_EXP_PlatformDesign

[7] Specification of Manifest
AUTOSAR_AP_TPS_ManifestSpecification

3.2 Further applicable specification

AUTOSAR provides a general specification [2] which is also applicable for V-UCM. The specification RS General shall be considered as additional and required specification for implementation of V-UCM.

4 Constraints and assumptions

4.1 Known limitations

No limitations

4.2 Applicability to car domains

No restrictions to applicability.

5 Dependencies to other Functional Clusters

This chapter provides an overview of the dependencies to other Functional Clusters in the AUTOSAR Adaptive Platform. Section 5.1 “Provided Interfaces” lists the interfaces provided by `Vehicle Update and Configuration Management` to other Functional Clusters. Section 5.2 “Required Interfaces” lists the interfaces required by `Vehicle Update and Configuration Management`.

A detailed technical architecture documentation of the AUTOSAR Adaptive Platform is provided in [3].

5.1 Provided Interfaces

Interface	*Functional Cluster*	*Purpose*
No provided interfaces		

Table 5.1: Interfaces provided to other Functional Clusters

5.2 Required Interfaces

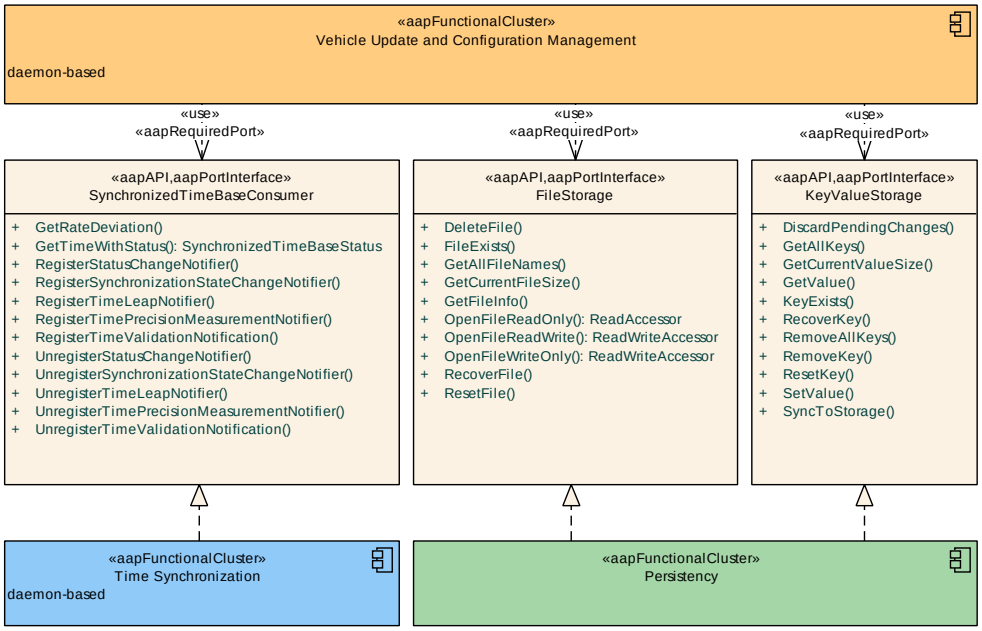

Figure 5.1: Interfaces required by Vehicle Update and Configuration Management from other Functional Clusters

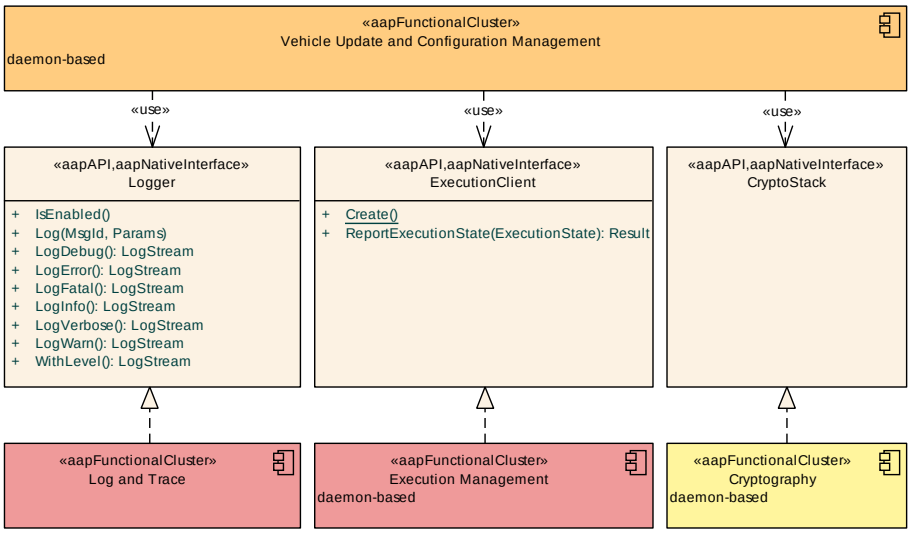

Figure 5.2: Interfaces required by Vehicle Update and Configuration Management from other Functional Clusters

Figures 5.1 and 5.2 show the interfaces required by `Vehicle Update and Configuration Management` from other Functional Clusters within the AUTOSAR Adaptive Platform.

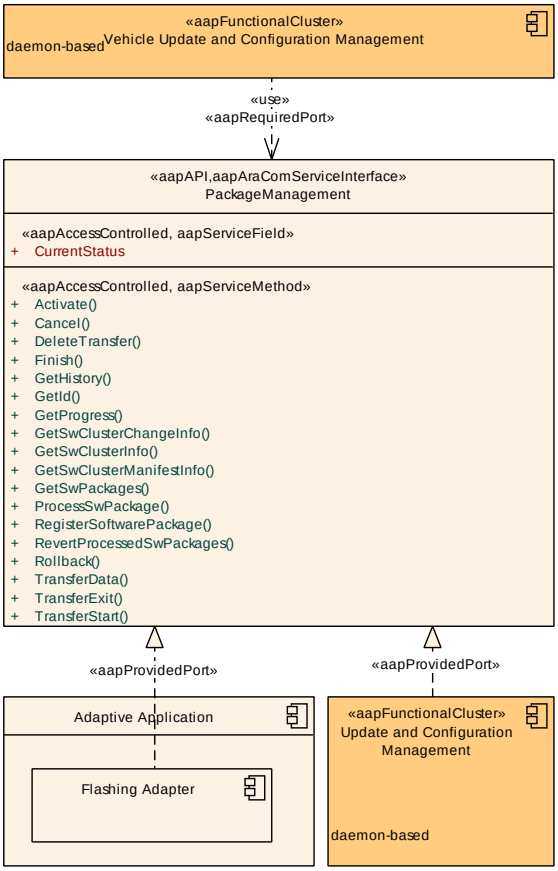

Figure 5.3: Interfaces required by Vehicle Update and Configuration Management from Update and Configuration Management

Figure 5.3 shows the interfaces required by Vehicle Update and Configuration Management from Update and Configuration Management.

Functional Cluster	*Interface*	*Purpose*
Cryptography	CryptoStack	This interface may be used e.g., to verify the integrity and authenticity of Vehicle Packages.
Execution Management	ExecutionClient	This interface shall be used by the daemon process(es) inside Vehicle Update and Configuration Management to report their execution state to Execution Management.
Log and Trace	Logger	Vehicle Update and Configuration Management shall use this interface to log standardized messages.
Persistency	FileStorage	Used to store files of received vehicle packages.
Persistency	KeyValueStorage	Used to store the internal state of Vehicle Update and Configuration Management.
Time Synchronization	SynchronizedTimeBaseConsumer	Vehicle Update and Configuration Management shall use this interface to get latest timestamp.
Update and Configuration Management	PackageManagement	This interface is used to control different Update and Configuration Management instances and e.g., applications implementing the same interface located within the vehicle that act as an adapter to install software packages on third-party systems. Vehicle Update and Configuration Management is able to differentiate between the service instances by matching the result of GetId with an ID provided in a Software Package.

Table 5.2: Interfaces required from other Functional Clusters

6 Requirements Tracing

The following tables reference the requirements specified in [4] 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_00013]	UCM shall check that it has enough resources to receive, process and store the Software Package and associated data	[SWS_VUCM_01011]
[RS_VUCM_00033]	V-UCM shall support reporting version information of a complete vehicle	[SWS_VUCM_00181] [SWS_VUCM_00182] [SWS_VUCM_00268] [SWS_VUCM_00269] [SWS_VUCM_01101] [SWS_VUCM_01103] [SWS_VUCM_01120] [SWS_VUCM_01135] [SWS_VUCM_01140] [SWS_VUCM_01150] [SWS_VUCM_01151] [SWS_VUCM_01152] [SWS_VUCM_01168] [SWS_VUCM_01169] [SWS_VUCM_01170] [SWS_VUCM_01171] [SWS_VUCM_01218] [SWS_VUCM_01311] [SWS_VUCM_CONSTR_00013]
[RS_VUCM_00034]	V-UCM shall record all V-UCM's action history	[SWS_VUCM_00181] [SWS_VUCM_00182] [SWS_VUCM_00251] [SWS_VUCM_00252] [SWS_VUCM_00253] [SWS_VUCM_00254] [SWS_VUCM_00255] [SWS_VUCM_00256] [SWS_VUCM_00290] [SWS_VUCM_00291] [SWS_VUCM_00296] [SWS_VUCM_00304] [SWS_VUCM_01034] [SWS_VUCM_01149] [SWS_VUCM_01167] [SWS_VUCM_01247] [SWS_VUCM_01248] [SWS_VUCM_01266] [SWS_VUCM_01267] [SWS_VUCM_01268] [SWS_VUCM_01269] [SWS_VUCM_01279] [SWS_VUCM_01283]
[RS_VUCM_00035]	V-UCM shall coordinate software update in a vehicle across multiple Electronic Control Units	[SWS_VUCM_00178] [SWS_VUCM_00181] [SWS_VUCM_00183] [SWS_VUCM_00210] [SWS_VUCM_00297] [SWS_VUCM_00298] [SWS_VUCM_00307] [SWS_VUCM_00308] [SWS_VUCM_01013] [SWS_VUCM_01018] [SWS_VUCM_01020] [SWS_VUCM_01021] [SWS_VUCM_01022] [SWS_VUCM_01033] [SWS_VUCM_01119] [SWS_VUCM_01122] [SWS_VUCM_01123] [SWS_VUCM_01124] [SWS_VUCM_01131] [SWS_VUCM_01132] [SWS_VUCM_01143] [SWS_VUCM_01145] [SWS_VUCM_01146] [SWS_VUCM_01156] [SWS_VUCM_01157] [SWS_VUCM_01160] [SWS_VUCM_01161] [SWS_VUCM_01172] [SWS_VUCM_01175] [SWS_VUCM_01177] [SWS_VUCM_01178] [SWS_VUCM_01204] [SWS_VUCM_01205] [SWS_VUCM_01207] [SWS_VUCM_01209] [SWS_VUCM_01212] [SWS_VUCM_01214] [SWS_VUCM_01215] [SWS_VUCM_01216] [SWS_VUCM_01217] [SWS_VUCM_01218] [SWS_VUCM_01219] [SWS_VUCM_01220] [SWS_VUCM_01221] [SWS_VUCM_01222] [SWS_VUCM_01227] [SWS_VUCM_01228] [SWS_VUCM_01229] [SWS_VUCM_01234] [SWS_VUCM_01236] [SWS_VUCM_01239] [SWS_VUCM_01240] [SWS_VUCM_01241] [SWS_VUCM_01242] [SWS_VUCM_01243] [SWS_VUCM_01244]

△

Requirement	Description	Satisfied by
		△ [SWS_VUCM_01246] [SWS_VUCM_01270] [SWS_VUCM_01271] [SWS_VUCM_01272] [SWS_VUCM_01273] [SWS_VUCM_01274] [SWS_VUCM_01276] [SWS_VUCM_01277] [SWS_VUCM_01283] [SWS_VUCM_01303] [SWS_VUCM_01305] [SWS_VUCM_CONSTR_00003] [SWS_VUCM_CONSTR_00005] [SWS_VUCM_CONSTR_00006] [SWS_VUCM_CONSTR_00009] [SWS_VUCM_CONSTR_00011] [SWS_VUCM_CONSTR_00015] [SWS_VUCM_CONSTR_00018]
[RS_VUCM_00036]	V-UCM shall use platform communication services for interacting with UCMs	[SWS_VUCM_01005] [SWS_VUCM_01015] [SWS_VUCM_01016] [SWS_VUCM_01021] [SWS_VUCM_01033] [SWS_VUCM_01034] [SWS_VUCM_01143] [SWS_VUCM_01156] [SWS_VUCM_01157] [SWS_VUCM_01172]
[RS_VUCM_00037]	V-UCM shall ensure it is safe to perform any modification to the vehicle	[SWS_VUCM_00177] [SWS_VUCM_00179] [SWS_VUCM_00183] [SWS_VUCM_00297] [SWS_VUCM_00298] [SWS_VUCM_01109] [SWS_VUCM_01114] [SWS_VUCM_01117] [SWS_VUCM_01138] [SWS_VUCM_01139] [SWS_VUCM_01141] [SWS_VUCM_01145] [SWS_VUCM_01146] [SWS_VUCM_01160] [SWS_VUCM_01161] [SWS_VUCM_01174] [SWS_VUCM_01179] [SWS_VUCM_01222] [SWS_VUCM_01228] [SWS_VUCM_01229] [SWS_VUCM_01234] [SWS_VUCM_01240] [SWS_VUCM_01244] [SWS_VUCM_01246] [SWS_VUCM_01275] [SWS_VUCM_01278] [SWS_VUCM_01309] [SWS_VUCM_01310] [SWS_VUCM_CONSTR_00003] [SWS_VUCM_CONSTR_00004] [SWS_VUCM_CONSTR_00005] [SWS_VUCM_CONSTR_00006] [SWS_VUCM_CONSTR_00007] [SWS_VUCM_CONSTR_00009] [SWS_VUCM_CONSTR_00018] [SWS_VUCM_CONSTR_00019]
[RS_VUCM_00038]	V-UCM shall interact with driver	[SWS_VUCM_00180] [SWS_VUCM_00182] [SWS_VUCM_01105] [SWS_VUCM_01117] [SWS_VUCM_01118] [SWS_VUCM_01120] [SWS_VUCM_01135] [SWS_VUCM_01142] [SWS_VUCM_01144] [SWS_VUCM_01147] [SWS_VUCM_01148] [SWS_VUCM_01159] [SWS_VUCM_01173] [SWS_VUCM_01222] [SWS_VUCM_01228] [SWS_VUCM_01234] [SWS_VUCM_CONSTR_00017]
[RS_VUCM_00039]	V-UCM shall prevent processing of compromised Vehicle Packages	[SWS_VUCM_00136] [SWS_VUCM_00181] [SWS_VUCM_01176] [SWS_VUCM_01221] [SWS_VUCM_01301] [SWS_VUCM_01302] [SWS_VUCM_01306]
[RS_VUCM_00042]	V-UCM shall provide an interface to read the state of an update campaign	[SWS_VUCM_00181] [SWS_VUCM_01017] [SWS_VUCM_01022] [SWS_VUCM_01143] [SWS_VUCM_01156] [SWS_VUCM_01157] [SWS_VUCM_01169] [SWS_VUCM_01172] [SWS_VUCM_01177] [SWS_VUCM_01178] [SWS_VUCM_01203] [SWS_VUCM_01205] [SWS_VUCM_01265] [SWS_VUCM_CONSTR_00016]

▽

△

Requirement	Description	Satisfied by
[RS_VUCM_00043]	V-UCM shall orchestrate a software update campaign according to the Vehicle Package's Manifest	[SWS_VUCM_00136] [SWS_VUCM_00179] [SWS_VUCM_00180] [SWS_VUCM_00181] [SWS_VUCM_00182] [SWS_VUCM_00210] [SWS_VUCM_01003] [SWS_VUCM_01014] [SWS_VUCM_01015] [SWS_VUCM_01016] [SWS_VUCM_01021] [SWS_VUCM_01023] [SWS_VUCM_01142] [SWS_VUCM_01143] [SWS_VUCM_01144] [SWS_VUCM_01145] [SWS_VUCM_01146] [SWS_VUCM_01147] [SWS_VUCM_01148] [SWS_VUCM_01156] [SWS_VUCM_01157] [SWS_VUCM_01158] [SWS_VUCM_01159] [SWS_VUCM_01160] [SWS_VUCM_01161] [SWS_VUCM_01162] [SWS_VUCM_01163] [SWS_VUCM_01164] [SWS_VUCM_01165] [SWS_VUCM_01166] [SWS_VUCM_01172] [SWS_VUCM_01173] [SWS_VUCM_01174] [SWS_VUCM_01180] [SWS_VUCM_01201] [SWS_VUCM_01207] [SWS_VUCM_01209] [SWS_VUCM_01212] [SWS_VUCM_01228] [SWS_VUCM_01275] [SWS_VUCM_01280] [SWS_VUCM_01282] [SWS_VUCM_01301] [SWS_VUCM_01302] [SWS_VUCM_01303] [SWS_VUCM_01305] [SWS_VUCM_01306] [SWS_VUCM_01307] [SWS_VUCM_01308] [SWS_VUCM_CONSTR_00018]
[RS_VUCM_00046]	V-UCM initialization	[SWS_VUCM_01019]
[RS_VUCM_00047]	UCM shall support standardized trace points	[SWS_VUCM_01024] [SWS_VUCM_01025] [SWS_VUCM_01026] [SWS_VUCM_01027] [SWS_VUCM_01028] [SWS_VUCM_01029] [SWS_VUCM_01030] [SWS_VUCM_01031] [SWS_VUCM_01032] [SWS_VUCM_01312] [SWS_VUCM_01313] [SWS_VUCM_01314] [SWS_VUCM_01315] [SWS_VUCM_01316] [SWS_VUCM_01317] [SWS_VUCM_01318] [SWS_VUCM_01319] [SWS_VUCM_01320]

Table 6.1: Requirements Tracing

7 Functional specification

7.1 Functional cluster life-cycle

[SWS_VUCM_01205] V-UCM internal state persistency

Upstream requirements: RS_VUCM_00035, RS_VUCM_00042

⌈V-UCM shall persist its state to be able to resume on-going update campaign after an intended or unintended reboot.⌋

7.1.1 Startup

[SWS_VUCM_01019] V-UCM initialization

Upstream requirements: RS_VUCM_00046

⌈V-UCM shall offer its services only after its internal initialization has been completed, and then report `kRunning` state.⌋

This requirement prevents calling V-UCM API while internal initialization is on-going. The concrete initialization tasks are implementation specific.

7.1.2 Shutdown

No specific requirements for V-UCM's shutdown

7.2 Technical Overview

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

V-UCM receives packages from Backend or Diagnostic tool (through Diagnostic Application), parses and interprets the Vehicle Package, transfers or streams Software Packages to suitable targets (UCM or Flashing Adapter) and orchestrates the processing, activations and eventual rollbacks. All these actions are what is called a campaign which V-UCM is coordinating. The UCM of the machines in the same network of a V-UCM, candidates target of a campaign, are referred to as UCMs.

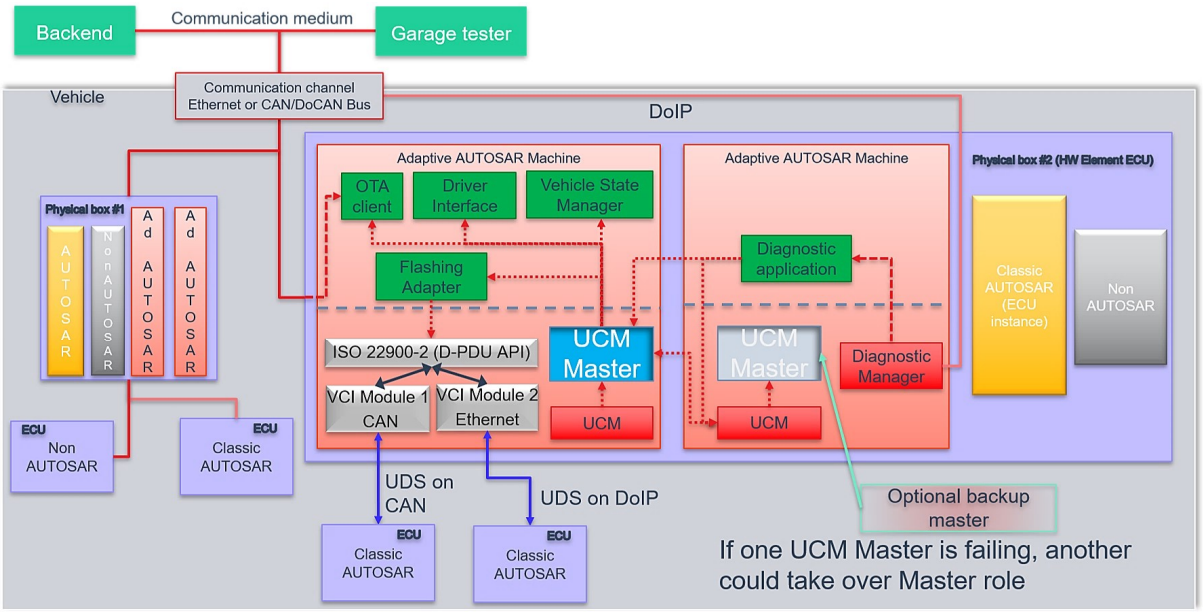

Figure 7.1: Example of V-UCM architecture overview within a vehicle

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

The OTA Client establishes a communication between Backend and V-UCM so that they can exchange information of the installed Software Clusters in the vehicle and the Software Clusters available in the Backend. This communication could be triggered by OTA Client with a scheduler and V-UCM to request the updates in case of newly available Software Clusters (pull case) or by Backend to push, for instance, an important security update to a fleet of vehicles (push case). The computation to find new Software Clusters versions and resolution of dependencies between Software Clusters can be either done at V-UCM or Backend.

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

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

7.3 V-UCM general behavior

The V-UCM acts as a client of the service interface offered by the UCM, already specified in UCM. However, the V-UCM also offers three different service interfaces to OTA Client, Vehicle Driver interface and Vehicle State Manager respectively. V-UCM aggregates UCMs states and can report its status field to a Backend through its OTA Client.

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

[SWS_VUCM_01003] V-UCM checks states of UCMs

Upstream requirements: RS_VUCM_00043

⌈A V-UCM shall check the status of its UCMs involved in the campaign as described in the Vehicle Package are all at kPreparing update state (UpdateStateType) of the UCMs CurrentStatus field before transitioning from kVehiclePackageTransferring to kSoftwarePackage_Transferring and before transitioning to kProcessing.⌋

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

[SWS_VUCM_CONSTR_00018] V-UCM uniqueness in vehicle

Upstream requirements: RS_VUCM_00043, RS_VUCM_00035, RS_VUCM_00037

⌈At any given point in time, at most one V-UCM shall provide VehiclePackageManagement, VehicleDriverApplicationInterface and VehicleStateManagerInterface services interfaces per network domain.⌋

There can be several V-UCMs running in the vehicle as backup but only one at a time can provide services in the vehicle.

[SWS_VUCM_01023] UcmStep order

Upstream requirements: RS_VUCM_00043

⌈All UcmStep from one VehicleRolloutStep can be done concurrently and in any order.⌋

7.4 UCM identification

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

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

[SWS_VUCM_01005] V-UCM is discovering UCMs in vehicle

Upstream requirements: RS_VUCM_00036

⌈V-UCM shall continuously look for UCM service instances (use of StartFindService() call).⌋

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

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

7.5 V-UCM Software Packages transfer or streaming

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

It is necessary to distinguish Vehicle Package (V-UCM specific) from Software Packages transfer.

[SWS_VUCM_01011] TransferVehiclePackage InsufficientMemory

Upstream requirements: RS_UCM_00013

⌈TransferVehiclePackage method shall raise the ApplicationError kMemoryInsufficient if the UCM buffer has not enough resources to process the corresponding Vehicle Package.⌋

[SWS_VUCM_01018] TransferVehiclePackage kBusyWithCampaign

Upstream requirements: RS_VUCM_00035

⌈TransferVehiclePackage method shall return the ApplicationError kBusyWithCampaign, if V-UCM is not at kIdle state⌋

A diagnostic tester can interfere with the V-UCM, which often results in a campaign constantly failing. Therefore, V-UCM allows new campaigns to be temporarily blocked to avoid this type of situation.

[SWS_VUCM_01014] Packages transferring sequence

Upstream requirements: RS_VUCM_00043

⌈TransferStart method shall raise the ApplicationError kPackageUnexpected if the Software Package name parameter was not a value of the RequestedPackage field.⌋

[SWS_VUCM_01013] Too big block size received by V-UCM

Upstream requirements: RS_VUCM_00035

⌈In the case the received block size with TransferData exceeds the block size returned by TransferStart or TransferVehiclePackage for the same TransferId, V-UCM shall raise the ApplicationError kBlockSizeIncorrect.⌋

[SWS_VUCM_01015] Invalid Vehicle Package manifest

Upstream requirements: RS_VUCM_00036, RS_VUCM_00043

⌈TransferExit shall raise the ApplicationError kPackageManifestInvalid when a Vehicle Package manifest is not compliant with the AUTOSAR schema.⌋

[SWS_VUCM_01016] Invalid Package Manifest

Upstream requirements: RS_VUCM_00036, RS_VUCM_00043

⌈V-UCM shall raise the ApplicationError kPackageManifestInvalid in case a manifest file is not compliant with the AUTOSAR schema.⌋

[SWS_VUCM_01017] RequestedPackage field

Upstream requirements: RS_VUCM_00042

⌈The field RequestedPackage shall contain the requested Software Package name and version as configured in campaign which is modelled by VehiclePackage.⌋

[SWS_VUCM_01033] Unreachable UCM during Packages transferring sequence

Upstream requirements: RS_VUCM_00035, RS_VUCM_00036

⌈If during the Software Package Transfer from V-UCM Client through V-UCM to an UCM using:

- TransferStart,
- TransferData,
- TransferExit,
- DeleteTransfer,

methods of Vehicle Package Management service interface the target UCM becomes unreachable, i.e. method invocation failed due to network errors, V-UCM shall raise the ApplicationError kUCMNotAvailableOnTheNetwork and add this event into Campaign History providing the reason in "returnedError" attribute of UCMStepErrorType as kUCMNotAvailableOnTheNetwork.⌋

OTA Client does not know what Software Packages should be transferred in a given campaign contained in a Vehicle Package. OTA Client can know what Software Package is expected to be transferred by subscribing to V-UCM's RequestedPackage field. Version is added to support campaigns which need an update path for a Software Package requiring an intermediate update to a transitional version. In this case the version parameter makes it unambiguous which package version shall be transferred as both have the same name assigned.

For the VehiclePackageManagement Service Interface methods

- TransferStart,
- TransferData,
- TransferExit,
- DeleteTransfer,

V-UCM can transfer Software Packages only to UCMs that are available. V-UCM is not responsible of waking up any potentially sleeping UCMs.

7.6 V-UCM retry strategy

When V-UCM or OTA Client calls TransferData method and it raises ApplicationError kBlockInconsistent, V-UCM or OTA Client can retry the TransferData method again later for the same block. This behaviour is configured by UcmRetryStrategy.

[SWS_VUCM_01020] Retry Strategy for BlockInconsistent

Upstream requirements: RS_VUCM_00035

⌈When `TransferData` returns `ApplicationError kBlockInconsistent` more than `maximumNumberOfRetries` within `retryIntervalTime`, then `V-UCM` shall cancel the active campaign by transitioning to `kCancelling` state and delete the failing package.⌋

If no retry strategy is needed, the maximum number of attempts can be specified as 0 in `UcmRetryStrategy`.

When `V-UCM` calls `ProcessSwPackage` method and it raises `ApplicationError kServiceBusy`, `V-UCM` can retry the `ProcessSwPackage` method again later for the same package. This behaviour is configured by `UcmRetryStrategy`.

[SWS_VUCM_00297] Retry Strategy for ServiceBusy

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037

⌈When `ProcessSwPackage` returns `ApplicationError kServiceBusy` more than `maximumNumberOfRetries` within `retryIntervalTime`, then `V-UCM` shall cancel the active campaign by transitioning to `kCancelling` state and delete the failing package.⌋

If no retry strategy is needed, the maximum number of attempts can be specified as 0 in `UcmRetryStrategy`.

When `V-UCM` calls `Activate` method and it raises `ApplicationError kUpdateSessionRejected`, `V-UCM` can retry the `Activate` method again later to enter Update Session again. This behaviour is configured by `UcmRetryStrategy`.

[SWS_VUCM_00298] Retry Strategy for UpdateSessionRejected

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037

⌈When `Activate` returns `ApplicationError kUpdateSessionRejected` more than `maximumNumberOfRetries` within `retryIntervalTime`, then `V-UCM` shall cancel the active campaign by transitioning to `kCancelling` state.⌋

If no retry strategy is needed, the maximum number of attempts can be specified as 0 in `UcmRetryStrategy`.

[SWS_VUCM_01034] Retry Strategy for Transfer methods

Upstream requirements: RS_VUCM_00034, RS_VUCM_00036

⌈If `V-UCM` cannot reach one or more UCMs when calling the PackageManagement Service Interface methods

- `TransferStart`,

- TransferData,
- TransferExit,
- DeleteTransfer,

more than maximumNumberOfRetries within retryIntervalTime, then V-UCM shall cancel the active campaign by transitioning to kCancelling state, add this event into Campaign History providing the reason in "returnedError" attribute of UCMStepErrorType as kUCMNotAvailableOnTheNetwork. If no retry strategy is needed, the maximum number of attempts can be specified as 0 in UcmRetryStrategy.⌋

The same attribute UcmRetryStrategy is used for retry strategies specified in [SWS_VUCM_01020].

7.7 Adaptive Applications interacting with V-UCM

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

7.7.1 OTA Client

OTA Client is an Adaptive Application that sets communication channel between Backend and V-UCM. The communication between Backend and OTA Client is abstracted and details like protocol are out of scope for this specification document. OTA Client should make sure Backend is providing the right information and packages to the vehicle by identifying the vehicle, by for instance sending VIN to Backend.

OTA Client uses the V-UCM as a service provider via ara::com. Since transferring Vehicle Packages and Software Packages from Backend to V-UCM is OTA Client's responsibility, OTA Client should be able to accommodate any proprietary communication protocol used between OTA Client and Backend and convert it into ara::com transport protocol. OTA Client should support UCM Software Packages transfer or streaming as specified in [5], it should then provide at least the following functionality:

- Comply to the requirements of [5] in the context of package transfer between OTA Client and V-UCM.
- OTA Client should subscribe to V-UCM's RequestedPackage field to know what Software Package is expected to be transferred
- OTA Client should subscribe to V-UCM's TransferState field to know what is campaign state

- OTA Client should subscribe to V-UCM's VehicleConditionCollection field to eventually make sure vehicle is in a safe state before transferring Packages
- OTA Client could support multiple data transfers in parallel, as specified in [SWS_UCM_00075] of [5]

In addition, OTA Client could support the ability to pause or resume the package transfer for the current campaign to prioritize the transfer of the packages from a different campaign. The ability of OTA Client to pause or resume the package transfer might be helpful in the case there is a need to cancel an ongoing campaign at kTransferring state to allow higher priority campaign to be performed. If a campaign is cancelled during transfer of packages, OTA Client has the possibility to recover the transfers using GetSwPackages method (providing transferId, consecutiveBytesReceived and consecutiveBlocksReceived) but could also simply delete the partially transferred packages with DeleteTransfer method.

Only one V-UCM has to be used by OTA Clients per network domain. As V-UCM is distributing Software Packages and coordinating UCM subordinates, OTA Clients in the same network domain have to make sure there are no already on-going campaigns when starting a new campaign with TransferVehiclePackage method call by checking V-UCM's state with TransferState field, in order to avoid any interference and guarantee success of an update campaign.

[SWS_VUCM_01101] Provide information of installed Software Clusters in vehicle

Upstream requirements: RS_VUCM_00033

⌈V-UCM shall provide a method GetSwClusterInfo to return information of all Software Cluster that are in state kPresent.⌋

V-UCM can aggregate Software Cluster information from several UCMs within a vehicle and returns the result to a Backend which can compute if there is any new Software Cluster available and decide to send to V-UCM through OTA Client a Vehicle Package. It is up to OTA Client to make sure the synchronisation of the versions of Software Packages present in Backend and Software Clusters in the vehicles using GetSwClusterInfo or SwPackageInventory is recent enough before starting a campaign with TransferVehiclePackage call.

[SWS_VUCM_01103] Inform Backend of needed Software Packages for an update

Upstream requirements: RS_VUCM_00033

⌈On SwPackageInventory call, V-UCM shall compare the supplied list of available Software Packages in the Backend for the vehicle to its own internal information of present Software Clusters in the vehicle and return the list of Software Packages selected for update.⌋

The OTA Client uses this returned Software Packages list to request the selected packages to the Backend. As required by constraint [SWS_UCM_CONSTR_00014] of [5], each Software Cluster corresponds to one Software Package and share the same shortName.

[SWS_VUCM_CONSTR_00016] OTA Client use of RequestedPackage field

Upstream requirements: RS_VUCM_00042

⌈When V-UCM updates the RequestedPackage field, the OTA Client shall start the transfer of the requested Software Package.⌋

[SWS_VUCM_01119] Report information of Software Packages

Status: OBSOLETE
Upstream requirements: RS_VUCM_00035

⌈V-UCM shall provide a method GetSwPackages to return the identifiers, names, versions, Consecutive Bytes Received, Consecutive Blocks Received and states of Software Packages.⌋

At the invocation of method GetSwPackages, V-UCM returns the identifiers, names, versions, Consecutive Bytes Received, Consecutive Blocks Received and states of Software Packages that are handled by V-UCM which could be stored in V-UCM or already transfered into UCM, depending on Vehicle Package content (SoftwarePackageStoring.storing).

OTA Client can use the GetSwPackages method to get the complete picture of the packages that are part of current campaign. Then, based on this information, the OTA Client can either start or continue the transfer of necessary software packages.

[SWS_VUCM_01175] Software Package deletion

Upstream requirements: RS_VUCM_00035

⌈When deleting a Software Package with DeleteTransfer method, V-UCM shall delete the Software Package eventually stored in V-UCM and its equivalent in UCM if already transferred.⌋

[SWS_VUCM_01176] Software Package authentication failure

Upstream requirements: RS_VUCM_00039

⌈In case UCM fails to authenticate a Software Package and raises ApplicationError kAuthenticationFailed, V-UCM shall delete the Software Packages eventually stored in V-UCM and its equivalent in UCM by having a DeleteTransfer method call to UCM.⌋

[SWS_VUCM_01180] DeleteTranfer OperationNotPermited error

Upstream requirements: RS_VUCM_00043

⌈If V-UCM receives DeleteTransfer call at state not being kIdle, kSyncing or kCancelling, V-UCM shall raise ApplicationError kOperationNotPermitted.⌋

For the methods GetSwPackages, GetSwClusterInfo, GetCampaignHistory, SwPackageInventory, V-UCM can collect information from several UCMs that are available. V-UCM is not responsible of waking up any potentially sleeping UCMs.

7.7.2 Vehicle Driver Interface

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

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

[SWS_VUCM_01105] Interaction of V-UCM with Vehicle Driver

Status: OBSOLETE
Upstream requirements: RS_VUCM_00038

⌈V-UCM shall provide a method Approve in order to receive the confirmation of the vehicle driver's approval.⌋

[SWS_VUCM_CONSTR_00017] Interaction of V-UCM with Vehicle Driver

Upstream requirements: RS_VUCM_00038

⌈When vehicle driver accepts the campaign, Vehicle Driver Application shall call Approve method of VehicleDriverApplicationInterface to inform V-UCM of the vehicle driver's decision.⌋

The Vehicle Driver Interface Adaptive Application could adapt its notification content related to safety by subscribing to the V-UCM's VehicleConditionCollection field.

[SWS_VUCM_01117] V-UCM VehicleConditionCollection field

Upstream requirements: RS_VUCM_00038, RS_VUCM_00037

⌈V-UCM shall provide to vehicle driver interface the VehicleConditionCollection field containing the required safety condition for the campaign as configured in safetyCondition.⌋

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

[SWS_VUCM_01118] V-UCM waiting for vehicle driver approval

Upstream requirements: RS_VUCM_00038

⌈In the case approval from driver is requested as configured in VehiclePackage, V-UCM shall wait for Approve method call before transitioning from kVehiclePackageTransferring to kSoftwarePackage_Transferring, kSoftwarePackage_Transferring to kProcessing or kProcessing to kActivating.⌋

[SWS_VUCM_CONSTR_00003] Exclusive use of Vehicle Driver Interface

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037

⌈Software Integrator shall ensure that only one Adaptive Application is using the V-UCM's Vehicle Driver Interface.⌋

For example, the integrator may restrict the access of Vehicle Driver Interface from V-UCM by configuring the Identity and Access Management functional cluster accordingly.

[SWS_VUCM_CONSTR_00004] Unsupported safety by Vehicle driver interface

Upstream requirements: RS_VUCM_00037

⌈In the case the Vehicle Driver Interface Adaptive Application does not support at least one safety condition contained in the VehicleConditionCollection field, the Vehicle Driver Interface Adaptive Application shall call the method ReportUnsupportedSafetyConditions with parameter UnsupportedSafetyConditions containing all the safety conditions which are not supported by Vehicle Driver Interface Adaptive Application.⌋

[SWS_VUCM_01141] V-UCM behavior in case a safety condition is not supported by Vehicle Driver Interface

Upstream requirements: RS_VUCM_00037

⌈If the method ReportUnsupportedSafetyConditions is called with parameter UnsupportedSafetyConditions containing at least one safety condition which is not supported by Vehicle Driver Interface Adaptive Application, V-UCM shall set VUCMResolutionType to kUnsupportedSafetyCondition.⌋

In case there is at least one safety condition unsupported by Vehicle Driver Application Interface and as long as the computed safety conditions are reported to be Safe by Vehicle State Manager PublishSafetyState method, it is not necessary to cancel the on-going campaign. However, in case an OEM or integrator wants to cancel the campaign because of an unsupported safety condition by Vehicle Driver Application

Interface, Vehicle Driver Application could call ReportUnsupportedSafetyConditions and perform cancellation of campaign.

[SWS_VUCM_01120] Provide Software Packages general information

Upstream requirements: RS_VUCM_00033, RS_VUCM_00038

⌈V-UCM shall provide a method GetSwPackageDescription to return the description of each Software Packages that are part of current campaign and that are contained in Vehicle Package.⌋

[SWS_VUCM_01135] Get Software Clusters descriptions from a vehicle

Status: OBSOLETE
Upstream requirements: RS_VUCM_00033, RS_VUCM_00038

⌈At GetSwClusterInfo method call via VehicleDriverApplicationInterface interface, UCM Master shall return the descriptions of all Software Clusters which are aggregated from all the UCM Subordinates and all Flashing Adapters.⌋

7.7.3 Vehicle State Manager

Vehicle State Manager is collecting states from the several vehicle ECUs and informs V-UCM when the safety state computed based on the safety policy referred in the Vehicle Package is changing. If the safety policy is not met, the V-UCM can for instance:

- Inform vehicle driver that the safety conditions are not met to continue the update
- Based on vehicle package configuration, pause the update until policy is met or cancel campaign

[SWS_VUCM_01139] V-UCM configured behavior when vehicle safety is not met

Upstream requirements: RS_VUCM_00037

⌈When a safety condition is not met, V-UCM shall wait in case violatedSafetyConditionBehavior is set to waitForVehicleSafeState or cancel the campaign when violatedSafetyConditionBehavior is set to cancelCampaign.⌋

[SWS_VUCM_01109] V-UCM provides a safety interface

Upstream requirements: RS_VUCM_00037

⌈V-UCM shall set the VehicleConditionCollection field taking the value from safetyCondition attribute for each VehicleRolloutStep of the VehiclePackage.⌋

`Vehicle State Manager` Adaptive Application can inform any vehicle state changes by calling `PublishSafetyState` method.

[SWS_VUCM_CONSTR_00005] Safety state change

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037

⌈`Vehicle State Manager` Adaptive Application shall call `PublishSafetyState` method provided by `V-UCM` when the safety state is changing.⌋

[SWS_VUCM_CONSTR_00009] Safety condition change

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037

⌈`Vehicle State Manager` Adaptive Application shall call `PublishSafetyState` method provided by `V-UCM` when at least one state subelement of the field `VehicleConditionCollection` has been changed.⌋

[SWS_VUCM_CONSTR_00019] Safety handling after reboot

Upstream requirements: RS_VUCM_00037

⌈During an on-going campaign and after a `V-UCM` reboot, `V-UCM` shall set the field SafetyConditions and wait to receive a `PublishSafetyState` method call from `Vehicle State Manager` Adaptive Application for `V-UCM` to resume the on-going campaign.⌋

Integrator ensures that setting of any value of SafetyConditions field (including same values) will actually be delivered to `Vehicle State Manager` Adaptive application in order to trigger its `PublishSafetyState` method call.

[SWS_VUCM_CONSTR_00015] Trigger on `kVehicleChecking` state

Upstream requirements: RS_VUCM_00035

⌈On transition to `kVehicleChecking` state provided by `CampaignState` from `VehicleStateManagerInterface`, `Vehicle State Manager` shall first perform checks to assess the post-activation state of the vehicle.⌋

`Vehicle State Manager` could be responsible for performing post-activation checks, interfacing with an application performing such checks, confirming backend is still reachable and further updates are still possible.

[SWS_VUCM_01272] `VehicleCheck` call not permitted

Upstream requirements: RS_VUCM_00035

⌈`V-UCM` shall return `ApplicationError kOperationNotPermitted` if `VehicleCheck` method is called in another `V-UCM` state than `kVehicleChecking`.⌋

[SWS_VUCM_CONSTR_00006] Exclusive use of Vehicle State Manager

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037

⌈System Integrator shall ensure that Vehicle State Manager is the exclusive user of the PublishSafetyState method.⌋

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

[SWS_VUCM_01275] Safety conditions during activation

Upstream requirements: RS_VUCM_00043, RS_VUCM_00037

⌈V-UCM shall apply the safetyCondition of the last VehicleRolloutStep to VehicleConditionCollection field during the ECUs activations as configured in the VehiclePackage⌋

[SWS_VUCM_CONSTR_00007] Unsupported safety conditions by Vehicle State Manager

Upstream requirements: RS_VUCM_00037

⌈In the case the requested VehicleConditionCollection field is not referring to an existing safety condition implemented by Vehicle State Manager, the Vehicle State Manager shall call VehicleStateManagerInterface service interface PublishSafetyState method with parameter safetyStates containing at least one value equal to 'NotSupported'.⌋

[SWS_VUCM_01278] V-UCM behaviour in case a safety condition is not supported by Vehicle State Manager

Upstream requirements: RS_VUCM_00037

⌈If there is at least one safety state returned by PublishSafetyState method which equals to 'NotSupported', V-UCM shall cancel the campaign and set VUCMResolutionType to kUnsupportedSafetyCondition.⌋

For the methods GetCampaignHistory, GetSwClusterInfo, V-UCM can collect information from several UCMs that are available. V-UCM is not responsible of waking up any potentially sleeping UCMs.

7.7.4 Flashing Adapter

Flashing Adapter is an application that is used in the case V-UCM is updating a AUTOSAR Classic Platform or any platform that can be flashed using diagnostic. It contains OEM specific diagnostic sequences and communicates via ara::com with the V-UCM and the AUTOSAR Adaptive Platform, and uses an implementation of

diagnostic protocol data unit application programming interface (D-PDU API) to communicate with Classic ECUs over the Vehicle Bus.

Flashing Adapter contains the needed logic to put the AUTOSAR Classic Platform ECUs into the right update mode in order to avoid any timeout issues during update.

The data transfer from Flashing Adapter to the target ECU via diagnostic communication can be subject to interruptions if communication on a higher priority protocol occurs, e.g. OBD services. In that case the Flashing Adapter can use a project specific strategy to detect the interruption, retry the transfer from the beginning, and decide whether to notify or not the client about the transfer interruption.

[SWS_VUCM_CONSTR_00011] Flashing Adapter provided interface

Upstream requirements: RS_VUCM_00035

⌈Flashing Adapter shall provide the same `ara::com` service interface as UCM ([SWS_UCM_00131] of [5]).⌋

7.7.5 UCM Diagnostic Application

	Diagnostic Application as UCM Client	**Diagnostic Application as V-UCM Client**
Purpose	Update standalone ECU/Machine without involvement of V-UCM	Update ECU/Machine as part of vehicle update through V-UCM
Flow	Diagnostic Tool -> Diagnostic Manager -> Diagnostic App -> UCM	Diagnostic Tool -> Diagnostic Manager ->Diagnostic App -> V-UCM
Instance	One instance per standalone Adaptive ECU/Machine	One instance per vehicle
Artifacts handled	Receives Software Packages	Receives Vehicle Packages and Software Packages
UCM API (Service)	Package Management	Vehicle Package Management
ECUs/Machines being updated	Adaptive only (incl. Classic Machines on Adaptive ECU, if needed)	Any ECU (Adaptive, Classic, Proprietary)
Implemented by	ECU Vendor and/or OEM	OEM

	Diagnostic Application as UCM Client	Diagnostic Application as V-UCM Client
References	• Figures "Architecture overview for diagnostic use case", "Sequence diagram showing the update process", "Sequence diagram showing the data transmission", "Sequence diagram showing the package processing" in the document [5] • Figure "Vehicle Update Architecture" in AUTOSAR_EXP_PlatformDesign [6] • Figure "Interfaces of UCM" in AUTOSAR_EXP_SWArchitecture [3]	• Figure 7.1, Figure 7.2 in this document, Figure "Sequence diagram showing the data transmission" in [5]

Table 7.1: The usage of UCM Diagnostic Application

7.7.6 Non Adaptive Platform update

The interface provided by the AUTOSAR Adaptive Platform in order to update non AUTOSAR Platform complies with the subset of ISO 22900-2:2017 (D-PDU API) requirements. As this standard's coverage is wide, it is allowed to implement a reduced API that is needed to update for instance a AUTOSAR Classic Platform.

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

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

As an implementation example, the pdu api source code and the communication parameters can be at appendix F.

7.7.6.1 D-PDU API implementation support

[SWS_VUCM_01122] Supported physical layers by D-PDU API implementation

Upstream requirements: RS_VUCM_00035

⌈ISO_11898_2_DWCAN (Dual Wire CAN), ISO_11898_3_DWFTCAN (Dual Wire CAN Fault tolerant), SAE_J2411_SWCAN (Single Wire CAN) and IEEE_802_3(Ethernet) physical layers shall be supported if their respective physical vehicle bus is available inside the ECU⌋

All other physical layers present in D-PDU API are optional.

[SWS_VUCM_01123] Supported application layers by D-PDU API implementation

Upstream requirements: RS_VUCM_00035

⌈ISO_15765_3 (Unified diagnostic services, UDS on CAN, ISO withdrawn UDS), ISO_14229_3 (Unified diagnostic services on CAN implementation, UDSonCAN) and

ISO_14229_5 (Unified diagnostic services on Internet Protocol implementation, UDSonIP) application layers shall be supported if their respective application layer is available inside the ECU.⌋

All other application layers present in D-PDU API are optional.

[SWS_VUCM_01124] Supported protocols by D-PDU API implementation

Upstream requirements: RS_VUCM_00035

⌈ISO UDS on CAN with Application layer ISO_15765_3, ISO UDS on CAN with Application layer ISO_14229_3 (UDSonCAN) and ISO UDS on DoIP with Application layer ISO_14229_5 (UDSonIP) protocols shall be supported.⌋

All other protocols are optional.

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

7.7.6.2 Not required D-PDU API concepts

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

A D-PDU API implementation can be split at OSI-Layer 4 into a D-PDU API implementation on OSI-Layer 5 and the VCI-Module on OSI-Layers 3 and 4.

The D-PDU API implementation does not use the D-PDU API root description file (RDF) as only one D-PDU API implementation is required for UCM within an AUTOSAR Adaptive Platform.

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

The D-PDU API implementation does not have to implement a protocol description file.

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

The Flashing Adapter can operate the D-PDU API without using symbolic names and IDs during runtime. If the use case excludes frequent changes to the MDFs, simple Flashing Adapter can even hardcode (e.g. in a header file) all necessary IDs and operate the D-PDU API without symbolic names.

D-PDU API implementation does not need to be compatible to SAE J2534-1 and RP 1210a.

The Adaptive Platform does not need any migration path.

D-PDU API implementation does not need to implement the IOCTL filter data structure.

7.7.6.3 Not required D-PDU API functions

PDULockResource() and PDUUnlockResource() are used to lock and unlock exclusive access to a ComLogicalLink in case of parallel usage of the D-PDU API implementation by multiple applications on the same physical communication link. Flashing of a Classic ECU always requires some exclusive access and should be handled in the AUTOSAR Adaptive Platform itself.

The D-PDU API implementation does not have to implement the parameter PDU_IOCTL_RESET

[SWS_VUCM_01131] PDUIoCtl(PDU_IOCTL_RESET)

Upstream requirements: RS_VUCM_00035

⌈The call of PDUIoCtl(PDU_IOCTL_RESET) shall return the error code PDU_ERR_ID_NOT_SUPPORTED, if PDU_IOCTL_RESET is not implemented.⌋

[SWS_VUCM_01132] PDUIoCtl(PDU_IOCTL_START_MSG_FILTER), PDUIoCtl(PDU_IOCTL_CLEAR_MSG_FILTER), PDUIoCtl(PDU_IOCTL_STOP_MSG_FILTER)

Upstream requirements: RS_VUCM_00035

⌈The call of PDUIoCtl() with any of the parameters PDU_IOCTL_START_MSG, PDU_IOCTL_CLEAR_MSG_FILTER, PDU_IOCTL_SEND_BREAK shall return the error code PDU_ERR_ID_NOT_SUPPORTED.⌋

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

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

The D-PDU API implementation of the AUTOSAR Adaptive Platform does not have to implement the returned codes PDU_ERR_CABLE_UNKNOWN, PDU_ERR_RSC_LOCKED, PDU_ERR_RSC_NOT_LOCKED, PDU_ERR_API_SW_OUT_OF_DATE and PDU_ERR_MODULE_FW_OUT_OF_DATE.

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

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

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

7.7.6.4 Classic platform update with V-UCM and diagnostic tool

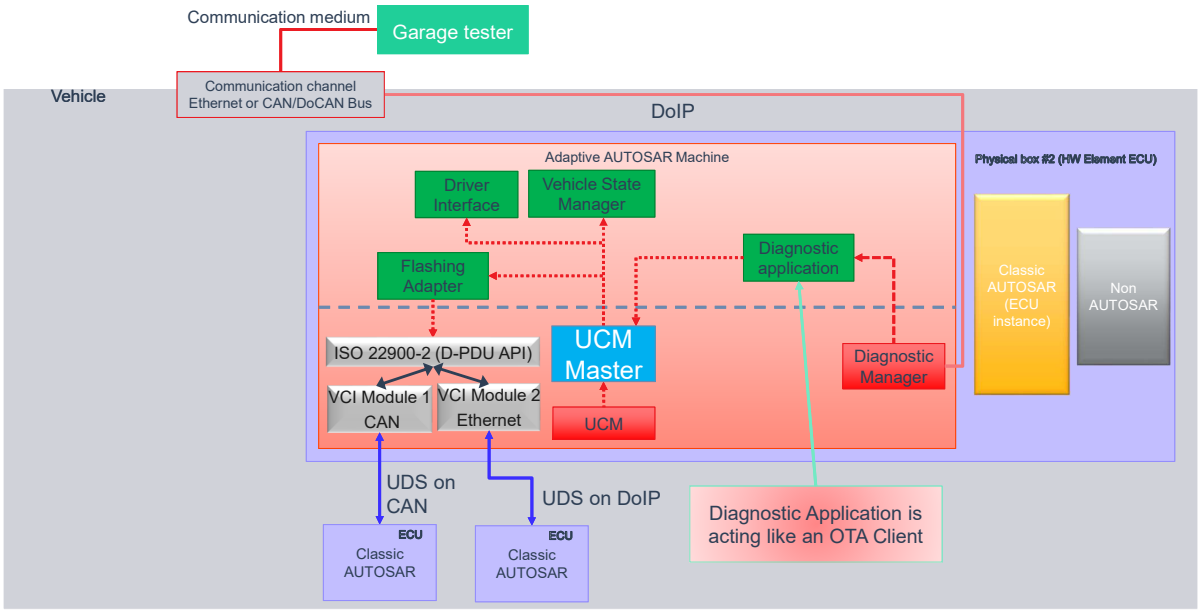

Figure 7.2: Classic platform update with V-UCM and diagnostic tool

The Diagnostic Manager connects the Diagnostic tool to the Adaptive Platform. The diagnostic application is acting like an OTA Client and uses the V-UCM services to push Vehicle Packages and Software Packages.

Note that this approach allows to update through Diagnostic Tool not only Classic ECU, but also Adaptive or Proprietary/Legacy Machines.

7.8 Status reporting

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

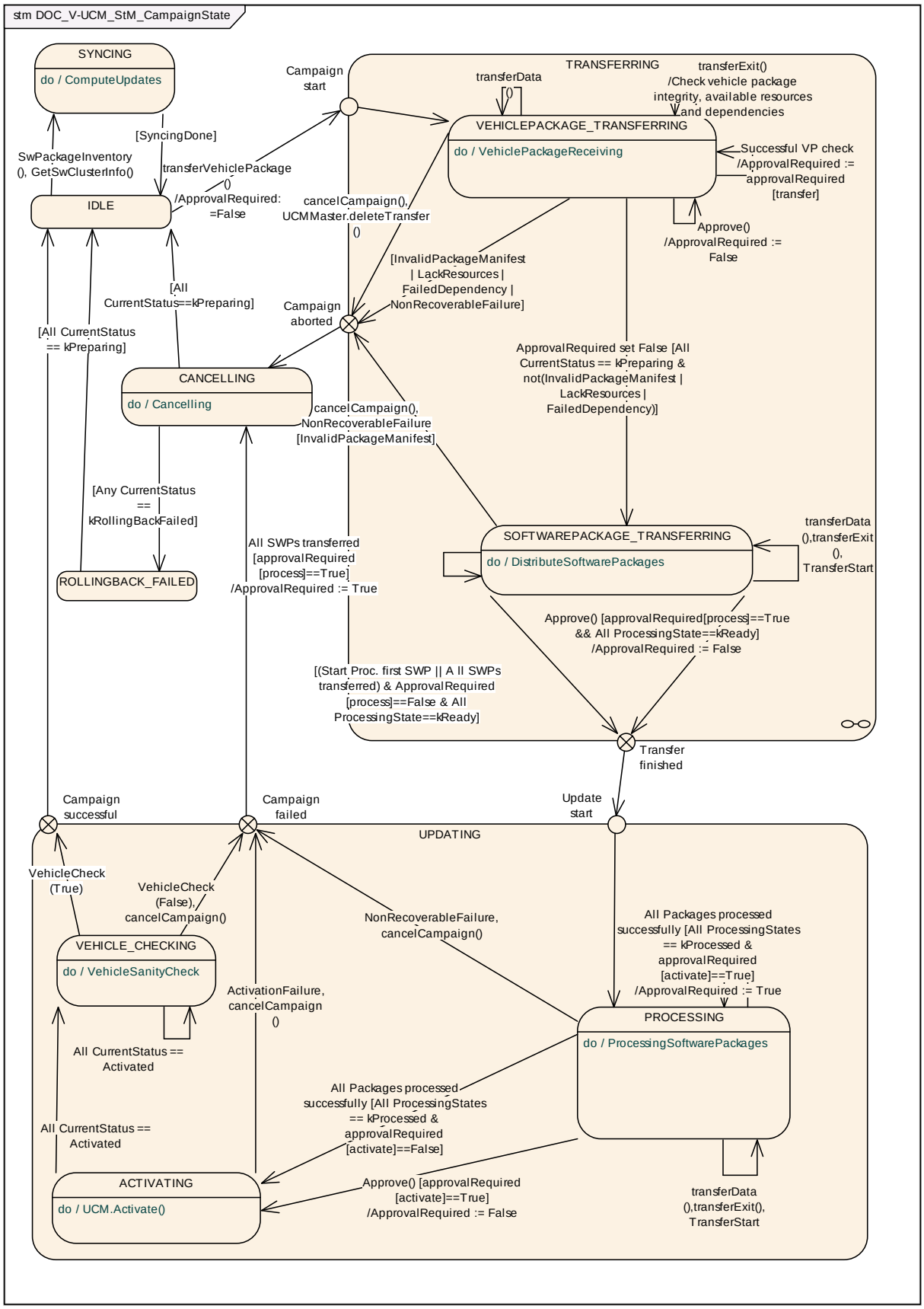

Figure 7.3: Campaign State Machine (`CampaignState` field)

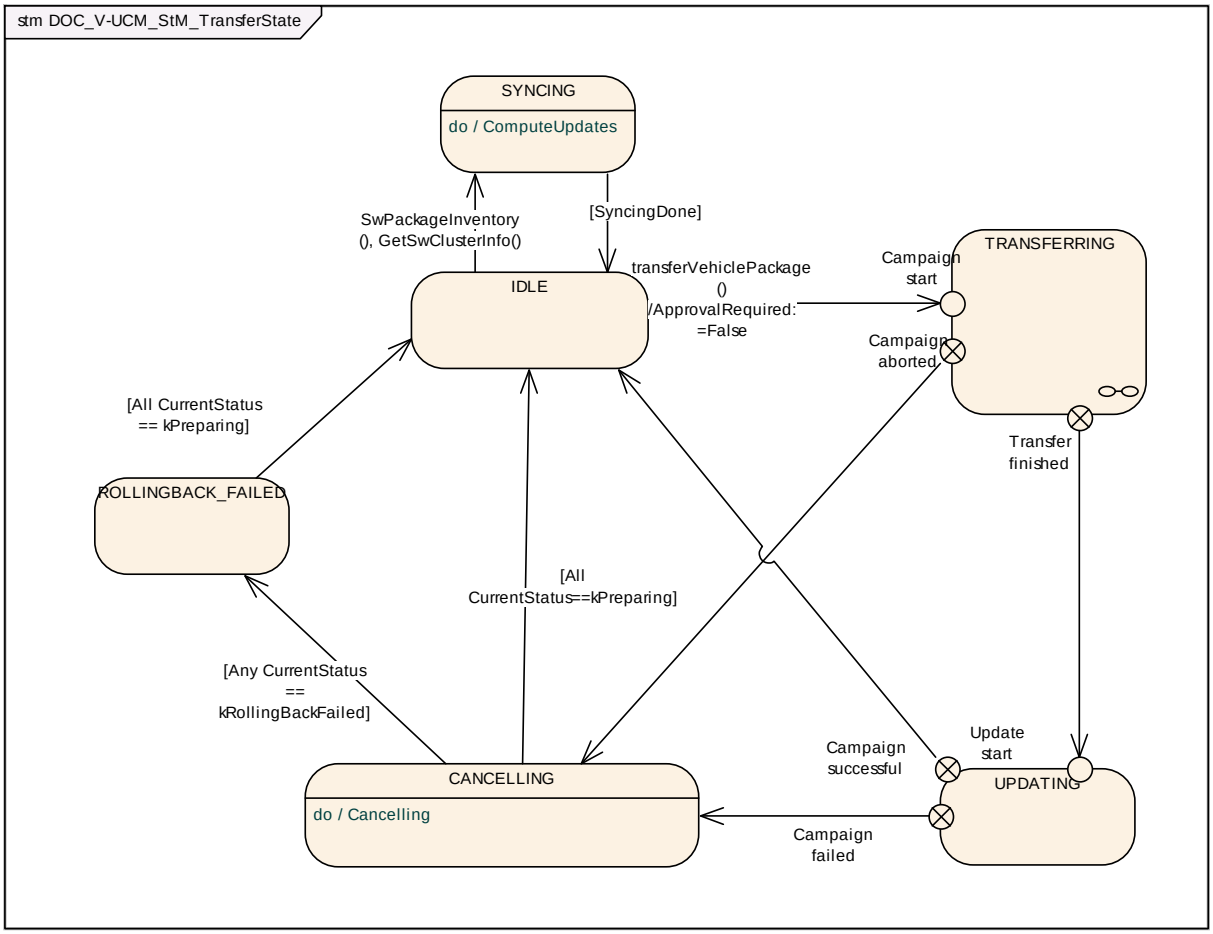

Figure 7.4: Campaign State Machine for OTA Client (`TransferState` field)

Diagrams 7.3 and 7.4 do not include behaviour after reset ([SWS_VUCM_01205] for more details)

[SWS_VUCM_01201] Sequential orchestration of campaigns

Upstream requirements: RS_VUCM_00043

⌈V-UCM shall orchestrate at most a single campaign at any one time.⌋

[SWS_VUCM_01265] TransferState field

Upstream requirements: RS_VUCM_00042

⌈

VPM::CampaignStateType	VPM::TransferState
kIdle	kIdle
kSyncing	kSyncing
kVehiclePackageTransferring	kTransferring
kSoftwarePackage_Transferring	kTransferring
kProcessing	kUpdating
kActivating	kUpdating
kVehicleChecking	kUpdating
kRollingBackFailed	kRollingBackFailed
kCancelling	kCancelling

V-UCM shall provide the state of a campaign over the TransferState field of the V-UCM's VehiclePackageManagement service interface following this above table

⌋

[SWS_VUCM_01203] CampaignState field

Upstream requirements: RS_VUCM_00042

⌈V-UCM shall provide the state of a campaign over the CampaignState field of the V-UCM VehicleDriverApplicationInterface and VehicleStateManagerInterface.⌋

There is an overview of the campaign state machine in Fig. 7.3 detailing V-UCM campaign states and transitions.

7.8.1 States

[SWS_VUCM_01204] Initial state

Upstream requirements: RS_VUCM_00035

⌈V-UCM shall have kIdle default state.⌋

[SWS_VUCM_01207] Trigger on kSoftwarePackage_Transferring state

Upstream requirements: RS_VUCM_00035, RS_VUCM_00043

⌈On transition to kSoftwarePackage_Transferring state and if all UCMs part of the campaign are in kPreparing state, V-UCM shall start or resume transferring (

TransferStart and TransferData as well as TransferExit if no streaming required) the software packages to the UCMs according to the campaign orchestration.⌋

[SWS_VUCM_01209] Trigger on kProcessing state

Upstream requirements: RS_VUCM_00035, RS_VUCM_00043

⌈On transition to kProcessing state, V-UCM shall call ProcessSwPackage method to UCMs to start or resume processing the software packages ready for processing according to the campaign orchestration.⌋

[SWS_VUCM_00210] Transferring of software packages on kProcessing state

Upstream requirements: RS_VUCM_00035, RS_VUCM_00043

⌈If V-UCM is in kProcessing state, V-UCM shall transfer Software Packages to the UCMs according to the campaign orchestration.⌋

[SWS_VUCM_01212] Trigger on kActivating state

Upstream requirements: RS_VUCM_00035, RS_VUCM_00043

⌈On transition to kActivating state, V-UCM shall ask UCMs to activate the software with Activate method call according to the campaign orchestration as configured in the VehiclePackage.⌋

[SWS_VUCM_01214] Final action on kVehicleChecking state

Upstream requirements: RS_VUCM_00035

⌈If V-UCM is in kVehicleChecking state and receives the method VehicleCheck call with parameter vehicleCheckResolution=True, V-UCM shall secondly commit (Finish) the software on all UCMs part of the campaign.⌋

[SWS_VUCM_01215] Trigger on kCancelling state

Upstream requirements: RS_VUCM_00035

⌈On transition to kCancelling state, V-UCM shall rollback (Rollback) the activated or being verified Software Clusters, and revert the processed packages (RevertProcessedSwPackages) of the UCMs part of the campaign.⌋

[SWS_VUCM_01216] Final action on kCancelling state

Upstream requirements: RS_VUCM_00035

⌈If V-UCM is in kCancelling state and the rollback of software on all UCMs is successful (successful Rollback and transition from kRollingBack to kRolledBack), V-UCM shall secondly commit (Finish) the software on all UCMs part of the campaign.⌋

[SWS_VUCM_01217] Monitoring of UCMs

Upstream requirements: RS_VUCM_00035

⌈V-UCM shall subscribe to the CurrentStatus field, in order to follow the current campaign from the state of the UCMs.⌋

7.8.2 States Transitions

[SWS_VUCM_01218] Transition from kIdle state to kSyncing state

Upstream requirements: RS_VUCM_00035, RS_VUCM_00033

⌈If V-UCM is in kIdle state for CampaignState field, V-UCM shall enter the kSyncing state for CampaignState on a request to GetSwClusterInfo or SwPackageInventory.⌋

[SWS_VUCM_01219] Transition from kSyncing state to kIdle state

Upstream requirements: RS_VUCM_00035

⌈If V-UCM is in kSyncing state for CampaignState field, V-UCM shall enter the kIdle state on completion of GetSwClusterInfo or SwPackageInventory.⌋

[SWS_VUCM_01220] Transition from kIdle state to kVehiclePackageTransferring

Upstream requirements: RS_VUCM_00035

⌈If V-UCM is in kIdle state for CampaignState field, V-UCM shall enter the kVehiclePackageTransferring state on successful completion of TransferVehiclePackage.⌋

[SWS_VUCM_01221] Transition from kVehiclePackageTransferring state

Upstream requirements: RS_VUCM_00035, RS_VUCM_00039

⌈If V-UCM is in kVehiclePackageTransferring state for CampaignState field, V-UCM shall enter the kCancelling state for CampaignState on unsuccessful completion of TransferExit (Vehicle Package) or successful completion of DeleteTransfer (Vehicle Package) or non recoverable error of TransferData.⌋

[SWS_VUCM_01222] Transition from kVehiclePackageTransferring state to kSoftwarePackage_Transferring state

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037, RS_VUCM_00038

⌈If V-UCM is in kVehiclePackageTransferring state, V-UCM shall enter the kSoftwarePackage_Transferring state on successful completion of TransferExit (Vehicle Package).⌋

[SWS_VUCM_01227] Transition from `kSoftwarePackage_Transferring` state to `kCancelling` state

Upstream requirements: RS_VUCM_00035

⌈If V-UCM is in kSoftwarePackage_Transferring state for CampaignState, V-UCM shall enter the kCancelling state for CampaignState on successful cancellation request (CancelCampaign) or if there is a non recoverable transfer failure from one of the UCMs.⌋

[SWS_VUCM_01228] Transition from `kSoftwarePackage_Transferring` state to `kProcessing` state

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037, RS_VUCM_00038, RS_VUCM_00043

⌈When V-UCM is in kSoftwarePackage_Transferring state for CampaignState, if all Software Packages are ready for processing, all Software Packages from all UCMs are at state kTransferred or at least one Software Package started being processed by ProcessSwPackage call to one UCM according to the campaign orchestration, V-UCM shall enter the kProcessing state for CampaignState.⌋

[SWS_VUCM_01229] SafetyConditions while processing stream

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037

⌈In the case there is transition from kSoftwarePackage_Transferring state to kProcessing state, the SafetyConditions for kProcessing state shall apply even though there are Software Packages transferring.⌋

It is integrator's responsibility to make sure in this use case that safety conditions for Processing will also cover safety approach of transferring.

[SWS_VUCM_01234] Transition from `kProcessing` state to `kActivating` state

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037, RS_VUCM_00038

⌈If V-UCM is in kProcessing state and all software packages of the campaign have been successfully (successful ProcessSwPackage) processed and all UCMs part to the campaign are in the kPreparing state, V-UCM shall enter the kActivating state.⌋

[SWS_VUCM_01236] Transition from `kProcessing` state to `kCancelling` state

Upstream requirements: RS_VUCM_00035

⌈If V-UCM is in kProcessing state for CampaignState, V-UCM shall enter the kCancelling state for CampaignState on successful cancellation request (CancelCampaign) or in case of non recoverable processing failure of one of the UCMs.⌋

[SWS_VUCM_01239] Transition from `kActivating` state to `kCancelling` state

Upstream requirements: RS_VUCM_00035

⌈If V-UCM is in kActivating state for CampaignState, V-UCM shall enter the kCancelling state for CampaignState if any UCMs part of the campaign unsuccessfully (unsuccessful Activate and transition from kVerifying to kRollingBack) completed activation.⌋

[SWS_VUCM_01240] Transition from `kActivating` state to `kVehicleChecking` state

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037

⌈If V-UCM is in kActivating state, V-UCM shall enter the kVehicleChecking state if all UCMs part of the campaign successfully (successful Activate and transition from kVerifying to kActivated) completed activation.⌋

[SWS_VUCM_01241] Transition from `kVehicleChecking` state to `kCancelling` state

Upstream requirements: RS_VUCM_00035

⌈If V-UCM is in kVehicleChecking state for CampaignState, and receives the method VehicleCheck call with parameter vehicleCheckResolution=False, V-UCM shall enter the kCancelling state for CampaignState.⌋

[SWS_VUCM_01242] Transition from `kVehicleChecking` stateto `kIdle` state

Upstream requirements: RS_VUCM_00035

⌈If V-UCM is in kVehicleChecking state for CampaignState and all UCMs part of the campaign transitioned from kCleaningUp to kPreparing, V-UCM shall enter the kIdle state for CampaignState.⌋

[SWS_VUCM_01282] Transition from `kVehicleChecking` state to `kIdle` state

Upstream requirements: RS_VUCM_00043

⌈V-UCM shall remove locally stored Software Packages used during the current campaign.⌋

Transitioning from kVehicleChecking state to kIdle state means the campaign was successful, therefore stored Software Packages are not needed anymore and would anyway infringe downgrade protection as required by [SWS_UCM_00103].

[SWS_VUCM_01276] Transition from `kRollingBackFailed` state to `kIdle` state

Upstream requirements: RS_VUCM_00035

⌈If V-UCM is in kRollingBackFailed state for CampaignState field, V-UCM shall enter the kIdle state when all UCM CurrentStatus fields have the update state (UpdateStateType) set to kPreparing.⌋

[SWS_VUCM_01277] Transition from `kCancelling` state to `kRollingBackFailed` state

Upstream requirements: RS_VUCM_00035

⌈If V-UCM is in kCancelling state for CampaignState field, V-UCM shall enter the kRollingBackFailed state when at least one UCM CurrentStatus field transitions to kRollingBackFailed.⌋

[SWS_VUCM_01243] Transition from `kCancelling` state to `kIdle` state

Upstream requirements: RS_VUCM_00035

⌈If V-UCM is in kCancelling state for CampaignState and all UCMs part of the campaign transitioned from kCleaningUp to kPreparing, V-UCM shall enter the kIdle state for CampaignState.⌋

[SWS_VUCM_01246] Unreachable UCM during update campaign

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037

⌈In case a UCM is not reachable by V-UCM during an update campaign (from kVehiclePackageTransferring, kSoftwarePackage_Transferring, kProcessing, kActivating, kVehicleChecking), V-UCM shall transit to kCancelling state for CampaignState.⌋

7.9 Campaign cancelling

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

In case an update campaign was cancelled, a new update campaign could use again the already transferred Software Packages. V-UCM could list transferred Software Packages by calling the UCMs with GetSwPackages.

[SWS_VUCM_01244] Cancellation of an update campaign shall be possible

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037

⌈Method CancelCampaign from V-UCM shall trigger a campaign cancel from kVehiclePackageTransferring, kSoftwarePackage_Transferring, kProcessing, kActivating, kVehicleChecking states by transitioning to state kCancelling.⌋

[SWS_VUCM_01270] New campaign disabling

Upstream requirements: RS_VUCM_00035

⌈V-UCM shall remain in kIdle or kSyncing when a CancelCampaign method has been called with disableCampaign parameter set to True. The V-UCM remains in this state (even across multiple power-down/reboot cycles) until CancelCampaign method is invoked again with the disableCampaign parameter set with false⌋

[SWS_VUCM_01271] New campaign enabling

Upstream requirements: RS_VUCM_00035

⌈Method AllowCampaign from V-UCM shall reallow new campaign after a CancelCampaign method was called with disableCampaign parameter set.⌋

[SWS_VUCM_01280] Maximum campaign duration

Upstream requirements: RS_VUCM_00043

⌈V-UCM shall cancel the campaign when the campaign duration (time from when V-UCM transitions to kTransfer, considering only the active or powered Machine time and excluding time waiting for driver approvals) exceeds maximumDurationOfCampaign⌋

Time counter starts after TransferVehiclePackage method call and have to be persisted in order to resume after shutdown and restart of Machine.

[SWS_VUCM_01273] CancelCampaign kCancelFailed error

Upstream requirements: RS_VUCM_00035

⌈CancelCampaign shall raise the error ApplicationError kCancelFailed in case cancelling of a campaign fails.⌋

[SWS_VUCM_01274] CancelCampaign kOperationNotPermitted error

Upstream requirements: RS_VUCM_00035

⌈CancelCampaign shall raise the error ApplicationError kOperationNotPermitted in case the V-UCM states are at kIdle, kSyncing or kCancelling.⌋

7.10 Campaign Reporting

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

[SWS_VUCM_01247] Method to read History Report

Upstream requirements: RS_VUCM_00034

⌈V-UCM shall provide a method GetCampaignHistory to retrieve all actions that have been performed by V-UCM when exiting state kUpdating from a specific time window.⌋

[SWS_VUCM_01248] Content of History Report

Upstream requirements: RS_VUCM_00034

⌈V-UCM shall save the point in time when TransferVehiclePackage method is called and the point in time when kIdle state is entered from any state except kSyncing based on timeBaseResource and the campaign result in CampaignHistoryType⌋

The elements of UCMHistoryVectorType [SWS_VUCM_00254] are not sorted.

[SWS_VUCM_01279] Keep history of Driver notification during campaign

Upstream requirements: RS_VUCM_00034

⌈If all approvalRequired values are false in the campaign when V-UCM is recording campaign history, V-UCM shall unset driverNotified attribute otherwise set it.⌋

[SWS_VUCM_01266] Subordinate Not Available On The Network

Upstream requirements: RS_VUCM_00034

⌈V-UCM shall record persistently into history the error kUCMNotAvailableOnTheNetwork in case one of the UCM subordinate involved in the current campaign stops offering its Service Interface.⌋

[SWS_VUCM_01267] Vehicle State Manager Communication Error

Upstream requirements: RS_VUCM_00034

⌈V-UCM shall record persistently the error kVehicleStateManagerCommunicationError into history in case the communication with Vehicle State Manager is not possible.⌋

[SWS_VUCM_01268] Vehicle Driver Interface Communication Error

Upstream requirements: RS_VUCM_00034

⌈V-UCM shall record persistently the error kVehicleDriverInterfaceCommunicationError into history in case the communication with Vehicle Driver Interface is no longer possible.⌋

[SWS_VUCM_01269] Campaign cancellation history

Upstream requirements: RS_VUCM_00034

⌈If `CancelCampaign` method is called, `V-UCM` shall record persistently the error `kCampaignCancelled` into history.⌋

[SWS_VUCM_01283] Errors returned by UCM during campaign

Upstream requirements: RS_VUCM_00034, RS_VUCM_00035

⌈While `V-UCM` being in a campaign, if UCM returns one of the following non-recoverable errors: `kMemoryInsufficient`, `kTransferFailed`, `kDataInsufficient`, `kPackageFormatUnsupported`, `kAuthenticationFailed`, `kPackageManifestInvalid`, `kPackageVersionIncompatible`, `kPackageInconsistent`, `kOldVersion`, `kDependencyMissing`, `kPersistencyAllocationFailed`, `kUpdateSessionRejected`, `kPrepareUpdateFailed`, `kVerificationFailed`, `kSoftwareClusterMissing`, `kDeltaIncompatible`, `kChecksumDescriptionInvalid`, `kSwclRemovalDenied`, `V-UCM` shall cancel the campaign and persistently report the error in history UCMStepErrorType.returnedError.⌋

`V-UCM` behavior is not specified for any other error coming from UCM not listed in above requirement: the `V-UCM` could recover, cancel the campaign or do something else.

7.11 Content of Vehicle Package

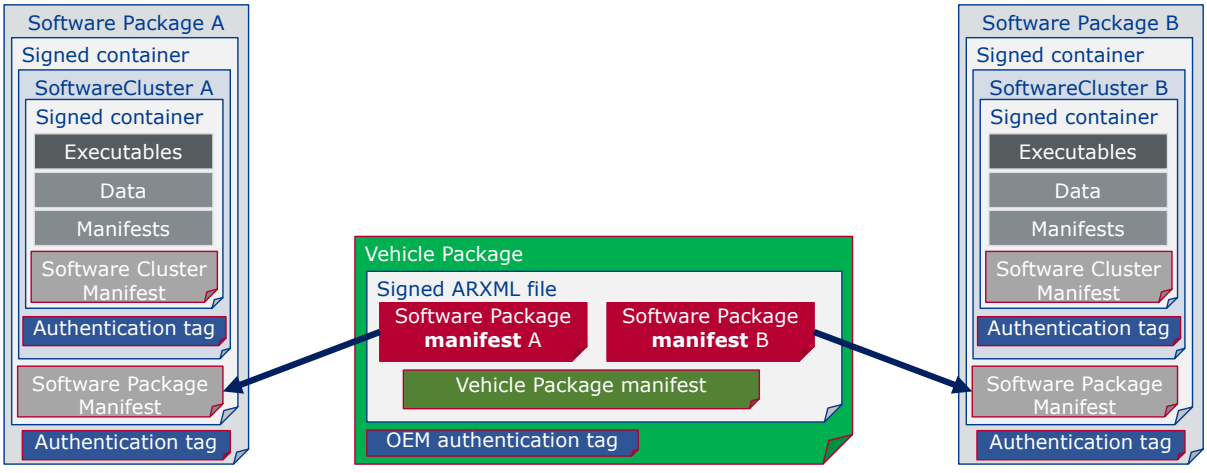

Figure 7.5: Vehicle package overview

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

These Software Packages have to be modelled as a so-called SoftwarePackage which describes the content of the Software Package. A Vehicle Package contains only one Vehicle Package Manifest. Several archive file formats could be used for the Vehicle Package or Software Package like .zip, gz, bz2, etc. UCM and V-UCM could implement several extraction algorithm compatible with several archiving formats and identify the right format to apply by analysing the file header.

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

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

- Repository: uri, repository or diagnostic address, for history, tracking and security purposes
- Vehicle description: vehicle description
- Vehicle Driver notifications: it might be needed to ask vehicle driver if V-UCM can start transferring Software Packages, processing it and activating it but also inform him of the necessary safety requirements if applicable.
- Safety policy: safety policy index to be used as argument to subscribe a field to vehicle safety manager. With this field, V-UCM will be informed at any time of campaign if vehicle safety is met or not.
- V-UCM identifiers list: defines backup V-UCMs
- Campaign orchestration: You can refer to [7] for more details. This campaign model allows to group activation of several UCMs and group Software Packages processing and transferring.

[SWS_VUCM_01301] Vehicle Package authentication

Upstream requirements: RS_VUCM_00039, RS_VUCM_00043

⌈The Vehicle Package shall be successfully authenticated by the V-UCM using CryptoServiceCertificate at TransferData or TransferExit call before any transfer of the Software Packages, otherwise raise ApplicationError kOperationNotPermitted.⌋

If Vehicle Package is not yet authenticated, V-UCM does not accept any transfer of Software Packages that are not confirmed to be part of Campaign for security purpose.

[SWS_VUCM_01302] Vehicle Package authentication failure

Upstream requirements: RS_VUCM_00039, RS_VUCM_00043

⌈In case Vehicle Package authentication fails at TransferExit call, V-UCM shall raise the ApplicationError kAuthenticationFailed.⌋

[SWS_VUCM_01303] Dependencies between Software Clusters

Upstream requirements: RS_VUCM_00035, RS_VUCM_00043

⌈V-UCM shall check dependencies based on Vehicle Package Manifest, Software Packages Manifests and information on already installed Software Clusters and their dependencies retrieved from UCMs using Package Manager service interface method GetSwClusterManifestInfo before a transfer of Software Packages.⌋

The vehicle dependency tree could be resolved by the backend after calling GetSwClusterInfo.

[SWS_VUCM_01311] Semantic versionning

Upstream requirements: RS_VUCM_00033

⌈V-UCM shall compute SoftwareCluster dependency check comparing only MajorVersion and MinorVersion.⌋

For SoftwareCluster dependency check [SWS_UCM_00319] or Vehicle Package compatibility against V-UCM [SWS_VUCM_01308], PatchVersion and additional labels of StrongRevisionLabelString are not considered.

The Vehicle Package contains a Vehicle Package manifest and Software Packages manifests of ARXML format in order to have interoperability between vendors.

[SWS_VUCM_01305] Vehicle Package format

Upstream requirements: RS_VUCM_00035, RS_VUCM_00043

⌈When the V-UCM receives a Vehicle Package whose Vehicle Package manifest and Software Package manifests formats are not ARXML, then the V-UCM shall return ApplicationError kPackageManifestInvalid from either TransferData or TransferExit method call and transition from kVehiclePackageTransferring or kTransferring to kIdle⌋

[SWS_VUCM_01307] `Vehicle Package` format not supported

Upstream requirements: RS_VUCM_00043

⌈`V-UCM` shall return `ApplicationError kPackageFormatUnsupported` from `TransferData` or `TransferExit` methods call in the case the `Vehicle Package` format is not supported.⌋

[SWS_VUCM_01306] `TransferExit` Invalid package manifest

Upstream requirements: RS_VUCM_00039, RS_VUCM_00043

⌈`TransferExit` shall raise the error `ApplicationError kPackageManifestInvalid` upon reception of an invalid manifest.⌋

[SWS_VUCM_01308] Check Vehicle Package version compatibility against V-UCM version

Upstream requirements: RS_VUCM_00043

⌈At `TransferExit` call, `V-UCM` shall raise `ApplicationError kPackageVersionIncompatible` if the version for the `Vehicle Package` transferred expressed by `minimumSupportedUcmMasterVersion` attribute is higher than the current version of V-UCM expressed by `version` attribute.⌋

7.12 Vehicle update security and confidentiality

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

[SWS_VUCM_CONSTR_00013] Confidential information protection

Upstream requirements: RS_VUCM_00033

⌈The `VehiclePackageManagement` and `VehicleDriverApplicationInterface` interfaces shall only be called over secure communication channel providing confidentiality protection.⌋

The `GetSwClusterInfo`, `SwPackageInventory`, `GetCampaignHistory` and `GetSwPackages` methods are using data that could identify vehicle user and therefore should be protected for confidentiality.

7.13 Reporting

7.13.1 Security Events

This functional cluster does not define any security events.

7.13.2 Log Messages

7.13.2.1 Standardized Logging

During an update campaign V-UCM interacts with multiple applications and UCM's. There are multiple events based on the response during the interaction as part of the update campaign. Therefore, it is important to provide a way to trace update campaign events within the V-UCM. The following trace points are introduced to be able to do analysis of important events during an update campaign.

[SWS_VUCM_01312] Campaign start - log start of update campaign

Upstream requirements: RS_VUCM_00047

⌈Whenever V-UCM starts receiving a Software Package (see [SWS_VUCM_01177]), V-UCM shall log a DltMessage of type CampaignStarted.⌋

[SWS_VUCM_01313] Campaign abort - log abort of update campaign

Upstream requirements: RS_VUCM_00047

⌈Whenever V-UCM aborts the campaign due to failure or explicit cancel campaign request (see [SWS_VUCM_00181], [SWS_VUCM_01244]), V-UCM shall log a DltMessage of type CampaignAborted.⌋

[SWS_VUCM_01314] Software Package transfer - log start of Software Package transfer

Upstream requirements: RS_VUCM_00047

⌈Whenever V-UCM finishes processing of Vehicle Package and starts transfer of Software Packages to UCM (see [SWS_VUCM_01177]), V-UCM shall log a DltMessage of type SoftwarePackageTransferStarted.⌋

[SWS_VUCM_01315] Software Package transfer - log end of Software Package transfer

Upstream requirements: RS_VUCM_00047

⌈Whenever V-UCM finishes transfer of Software Packages to UCM (see [SWS_VUCM_00181]), V-UCM shall log a DltMessage of type SoftwarePackageTransferFinished.⌋

[SWS_VUCM_01316] Vehicle Package transfer failure - log failure during transfer of Vehicle Package

Upstream requirements: RS_VUCM_00047

⌈Whenever V-UCM detects failure in transfer of Vehicle Package (see [SWS_VUCM_00181]), V-UCM shall log a DltMessage of type VehiclePackageTransferFailed.⌋

[SWS_VUCM_01317] `Software Package` transfer failure - log failure during transfer of `Software Package`

Upstream requirements: RS_VUCM_00047

⌈Whenever V-UCM detects failure in transfer of Software Package (see [SWS_VUCM_00181]), V-UCM shall log a DltMessage of type SoftwarePackageTransferFailed.⌋

[SWS_VUCM_01318] Start of update - log start of update process

Upstream requirements: RS_VUCM_00047

⌈Whenever V-UCM starts the update process by requesting package processing to UCM (see [SWS_VUCM_00181]), V-UCM shall log a DltMessage of type UpdateStarted.⌋

[SWS_VUCM_01319] Campaign result - log failure of update campaign

Upstream requirements: RS_VUCM_00047

⌈Whenever update campaign fails (see [SWS_VUCM_00252]), V-UCM shall log a DltMessage of type CampaignFailed.⌋

[SWS_VUCM_01320] Campaign result - log success of update campaign

Upstream requirements: RS_VUCM_00047

⌈Whenever update campaign is successful (see [SWS_VUCM_00252]), V-UCM shall log a DltMessage of type CampaignSuccessful.⌋

[SWS_VUCM_01025] LogMessage CampaignAborted

Status: DRAFT
Upstream requirements: RS_VUCM_00047

⌈

Dlt-Message	CampaignAborted		
Description	Message that is sent by V-UCM if the ongoing campaign is aborted.		
MessageId	0x8000d001		
MessageType Info	DLT_LOG_INFO		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
Campaign AssociatedWith VehiclePackage	Campaign associated with vehicle package	predefined text	
TransferId	TransferId	uint8 [16]	NoUnit
IsAborted	is aborted	predefined text	

⌋

[SWS_VUCM_01031] LogMessage CampaignFailed

Status: DRAFT
Upstream requirements: RS_VUCM_00047

⌈

Dlt-Message	CampaignFailed		
Description	Message that is sent by V-UCM after the failure of campaign associated with vehicle package.		
MessageId	0x8000d007		
MessageType Info	DLT_LOG_ERROR		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
Campaign AssociatedWith VehiclePackage	Campaign associated with vehicle package	predefined text	
VehiclePackage ShortName	Vehicle package shortname	uint8 [encoding UTF-8]	NoUnit
FailedWith CampaignResult	failed with campaign result	predefined text	
UCMMaster ResolutionType	UCM Master resolution type	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_VUCM_01024] LogMessage CampaignStarted

Status: DRAFT
Upstream requirements: RS_VUCM_00047

⌈

Dlt-Message	CampaignStarted		
Description	Message that is sent by V-UCM after starting the campaign by receiving vehicle package.		
MessageId	0x8000d000		
MessageType Info	DLT_LOG_INFO		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
Campaign AssociatedWith VehiclePackage	Campaign associated with vehicle package	predefined text	
TransferId	TransferId	uint8 [16]	NoUnit
HasStarted	has started	predefined text	

⌋

[SWS_VUCM_01032] LogMessage CampaignSuccessful

Status: DRAFT
Upstream requirements: RS_VUCM_00047

⌈

Dlt-Message	CampaignSuccessful		
Description	Message that is sent by V-UCM after success of campaign associated with vehicle package.		
MessageId	0x8000d008		
MessageType Info	DLT_LOG_INFO		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
Campaign AssociatedWith VehiclePackage	Campaign associated with vehicle package	predefined text	
VehiclePackage ShortName	Vehicle package shortname	uint8 [encoding UTF-8]	NoUnit
SuccessfulWith CampaignResult	successful with campaign result	predefined text	
UCMMaster ResolutionType	UCM Master resolution type	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_VUCM_01029] LogMessage SoftwarePackageTransferFailed

Status: DRAFT
Upstream requirements: RS_VUCM_00047

⌈

Dlt-Message	SoftwarePackageTransferFailed		
Description	Message that is sent by V-UCM after failure in transferring of software package (failure in TransferData/ TransferExit).		
MessageId	0x8000d005		
MessageType Info	DLT_LOG_ERROR		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TransferOf SoftwarePackage	Transfer of software package	predefined text	
SoftwarePackage ShortName	Software package shortname	uint8 [encoding UTF-8]	NoUnit
FailedWith ApplicationError Code	failed with application error code	predefined text	
ErrorCode	Error Code	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_VUCM_01027] LogMessage SoftwarePackageTransferFinished

Status: DRAFT
Upstream requirements: RS_VUCM_00047

⌈

Dlt-Message	SoftwarePackageTransferFinished		
Description	Message that is sent by V-UCM after transferring software package.		
MessageId	0x8000d003		
MessageType Info	DLT_LOG_INFO		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
SoftwarePackage	Software package	predefined text	
SoftwarePackage ShortName	Software package shortname	uint8 [encoding UTF-8]	NoUnit
TransferredTo UCM	transferred to UCM	predefined text	
UcmModule Instantiation Identifier	UCM module instantiation identifier	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_VUCM_01026] LogMessage SoftwarePackageTransferStarted

Status: DRAFT
Upstream requirements: RS_VUCM_00047

⌈

Dlt-Message	SoftwarePackageTransferStarted		
Description	Message that is sent by V-UCM after it processes vehicle package and starts transferring software packages to UCM.		
MessageId	0x8000d002		
MessageType Info	DLT_LOG_INFO		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
VehiclePackage	Vehicle package	predefined text	
VehiclePackage ShortName	Vehicle package shortname	uint8 [encoding UTF-8]	NoUnit
IsProcessedAnd TransferOf Software PackagesStarted	is processed and transfer of software packages started	predefined text	

⌋

[SWS_VUCM_01030] LogMessage UpdateStarted

Status: DRAFT
Upstream requirements: RS_VUCM_00047

⌈

Dlt-Message	UpdateStarted		
Description	Message that is sent by V-UCM after requesting package processing to UCM.		
MessageId	0x8000d006		
MessageType Info	DLT_LOG_INFO		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
UpdateOf SoftwarePackage	Update of software package	predefined text	
SoftwarePackage ShortName	Software package shortname	uint8 [encoding UTF-8]	NoUnit
StartedForUCM	started for UCM	predefined text	
UcmModule Instantiation Identifier	UCM module instantiation identifier	uint8 [encoding UTF-8]	NoUnit

⌋

[SWS_VUCM_01028] LogMessage VehiclePackageTransferFailed

Status: DRAFT
Upstream requirements: RS_VUCM_00047

⌈

Dlt-Message	VehiclePackageTransferFailed		
Description	Message that is sent by V-UCM after failure in transferring of vehicle (failure in TransferData/Transfer Exit).		
MessageId	0x8000d004		
MessageType Info	DLT_LOG_ERROR		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
TransferOf VehiclePackage WithTransferId	Transfer of vehicle package with transferid	predefined text	
TransferId	TransferId	uint8 [16]	NoUnit
FailedWith ApplicationError Code	failed with application error code	predefined text	
ErrorCode	Error Code	uint8 [encoding UTF-8]	NoUnit

⌋

7.13.3 Violation Messages

No violation messages applicable to V−UCM

7.13.4 Production Errors

No production errors applicable to V-UCM

8 API specification

There are no APIs defined in this release.

9 Service Interfaces

This chapter lists all provided and required service interfaces of the V-UCM.

Tables are generated out of 'arxml' folder content.

9.1 Type definitions

This chapter lists all types provided by the V-UCM.

The following figures are informative and only meant to support reader having global view of V-UCM types and service interfaces.

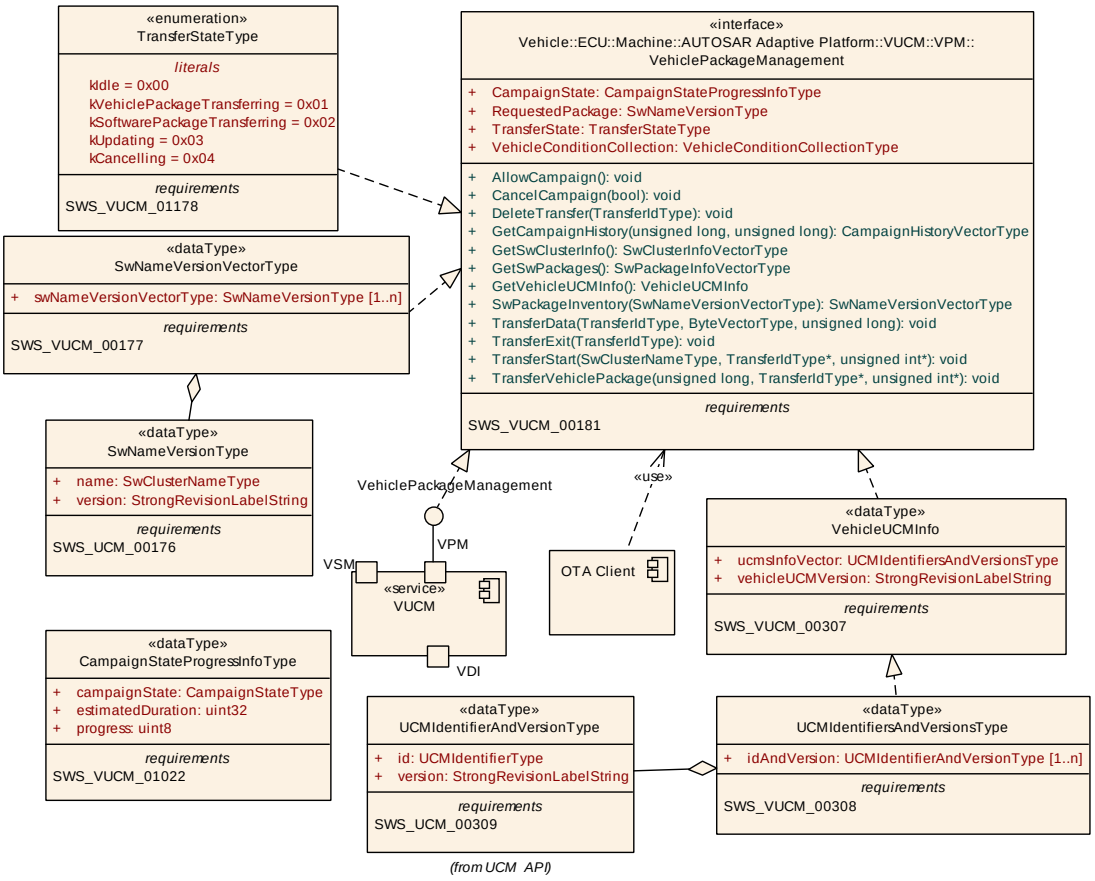

Figure 9.1: V-UCM Vehicle Package Management Service Interface

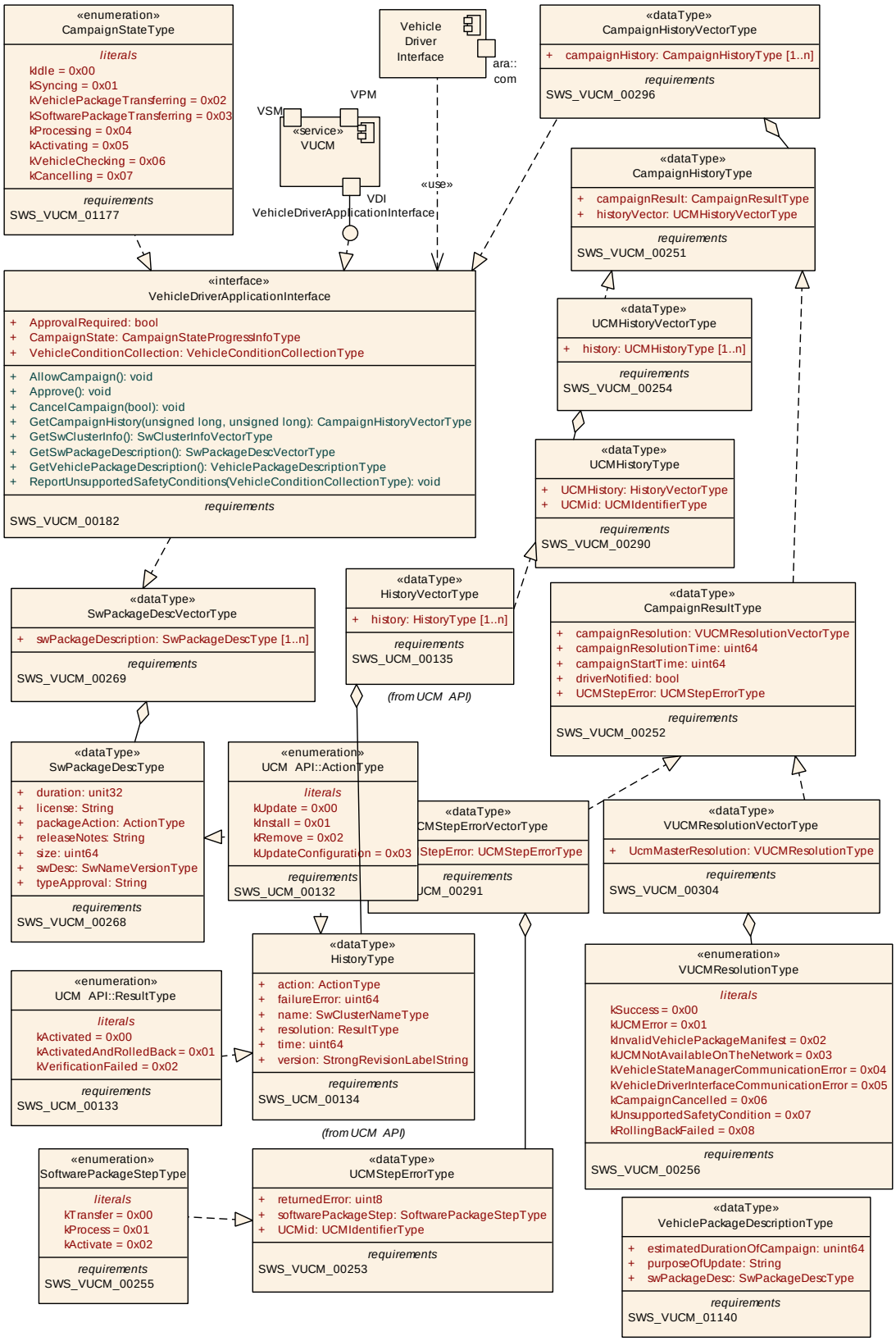

Figure 9.2: V-UCM Vehicle Driver Application Interface

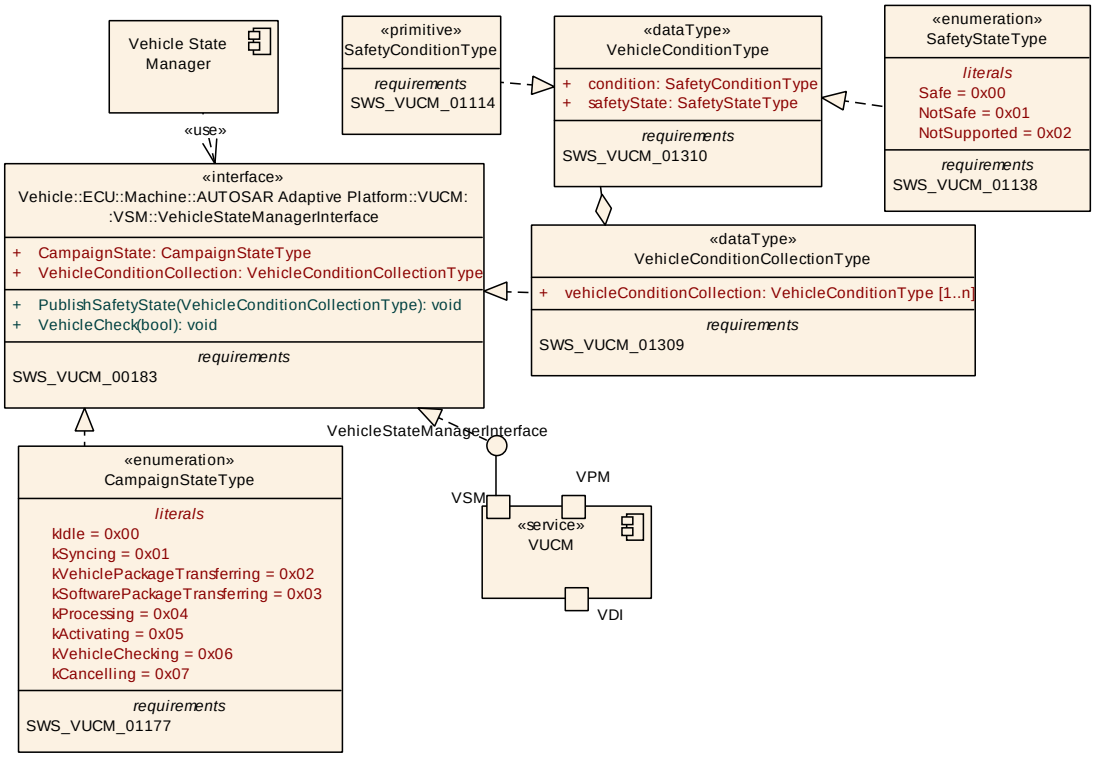

Figure 9.3: V-UCM Vehicle State Manager Interface

9.1.1 CampaignHistoryType

[SWS_VUCM_00251] Definition of ImplementationDataType CampaignHistoryType

Upstream requirements: RS_VUCM_00034

⌈

Name	CampaignHistoryType
Namespace	ara::vucm
Kind	STRUCTURE
Sub-elements	campaignResult `CampaignResultType` historyVector `UCMHistoryVectorType` repository `UriString`
Derived from	-
Description	Campaign history

⌋

9.1.2 CampaignHistoryVectorType

[SWS_VUCM_00296] Definition of ImplementationDataType CampaignHistoryVectorType

Upstream requirements: RS_VUCM_00034

⌈

Name	CampaignHistoryVectorType
Namespace	ara::vucm
Kind	VECTOR <CampaignHistoryType>
Derived from	-
Description	Represents a list of Campaign history

⌋

9.1.3 CampaignResultType

[SWS_VUCM_00252] Definition of ImplementationDataType CampaignResultType

Upstream requirements: RS_VUCM_00034

⌈

Name	CampaignResultType
Namespace	ara::vucm
Kind	STRUCTURE
Sub-elements	campaignResolution VUCMResolutionVectorType UCMStepError UCMStepErrorVectorType campaignStartTime uint64_t campaignResolutionTime uint64_t driverNotified bool
Derived from	-
Description	Campaign Resolution. UCM gets time from Time Sync Functional Cluster via UcmToTimeBaseResourceMapping.timeBaseResource

⌋

9.1.4 VUCMResolutionVectorType

[SWS_VUCM_00304] Definition of ImplementationDataType VUCMResolutionVectorType

Upstream requirements: RS_VUCM_00034

⌈

Name	VUCMResolutionVectorType
Namespace	ara::vucm
Kind	VECTOR <VUCMResolutionType>
Derived from	-
Description	Vector of V-UCM errors

⌋

9.1.5 VUCMResolutionType

[SWS_VUCM_00256] Definition of ImplementationDataType VUCMResolutionType

Upstream requirements: RS_VUCM_00034

⌈

Name	VUCMResolutionType	
Namespace	ara::vucm	
Kind	TYPE_REFERENCE	
Derived from	uint8_t	
Description	Campaign resolution	
Range / Symbol	***Limit***	***Description***
kSuccess	0x00	Successful campaign
kUCMError	0x01	UCM error
kInvalidVehiclePackage Manifest	0x02	Vehicle Package manifest is invalid
kUCMNotAvailableOnThe Network	0x03	UCM subordinate not reachable
kVehicleStateManager CommunicationError	0x04	Communication error with Vehicle State Manager
kVehicleDriverInterface CommunicationError	0x05	Communication error with Vehicle Driver Interface
kCampaignCancelled	0x06	Campaign was cancelled
kUnsupportedSafetyCondition	0x07	Safety condition not supported by Vehicle Driver Interface Adaptive Application
kRollingBackFailed	0x08	One UCM failed to revert changes introduced with processed packages.

⌋

9.1.6 UCMStepErrorVectorType

[SWS_VUCM_00291] Definition of ImplementationDataType UCMStepErrorVectorType

Upstream requirements: RS_VUCM_00034

⌈

Name	UCMStepErrorVectorType
Namespace	ara::vucm
Kind	VECTOR <UCMStepErrorType>
Derived from	-
Description	Vector of UCM's errors

⌋

9.1.7 UCMStepErrorType

[SWS_VUCM_00253] Definition of ImplementationDataType UCMStepErrorType

Upstream requirements: RS_VUCM_00034

⌈

Name	UCMStepErrorType
Namespace	ara::vucm
Kind	STRUCTURE
Sub-elements	ucmId UCMIdentifierType softwarePackageStep SoftwarePackageStepType returnedError uint8_t
Derived from	-
Description	UCM Error

⌋

9.1.8 SoftwarePackageStepType

[SWS_VUCM_00255] Definition of ImplementationDataType SoftwarePackageStepType

Upstream requirements: RS_VUCM_00034

⌈

Name	SoftwarePackageStepType	
Namespace	ara::vucm	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	UCM Software Package step at which error occurred	
Range / Symbol	***Limit***	***Description***
kTransfer	0x00	Software Package transfer
kProcess	0x01	Software Package processing
kActivate	0x02	Software Cluster activation

⌋

9.1.9 UCMHistoryType

[SWS_VUCM_00290] Definition of ImplementationDataType UCMHistoryType

Upstream requirements: RS_VUCM_00034

⌈

Name	UCMHistoryType
Namespace	ara::vucm
Kind	STRUCTURE
Sub-elements	ucmId `UCMIdentifierType` historyVector `HistoryVectorType`
Derived from	-
Description	History of an UCM

⌋

9.1.10 UCMHistoryVectorType

[SWS_VUCM_00254] Definition of ImplementationDataType UCMHistoryVectorType

Upstream requirements: RS_VUCM_00034

⌈

Name	UCMHistoryVectorType
Namespace	ara::vucm
Kind	VECTOR <UCMHistoryType>
Derived from	-
Description	Vector of UCM sub histories

⌋

9.1.11 CampaignStateType

[SWS_VUCM_01177] Definition of ImplementationDataType CampaignStateType

Upstream requirements: RS_VUCM_00035, RS_VUCM_00042

⌈

Name	CampaignStateType	
Namespace	ara::vucm	
Kind	TYPE_REFERENCE	
Derived from	uint8_t	
Description	Represents the status of Campaign.	
Range / Symbol	***Limit***	***Description***
kIdle	0x00	V-UCM is ready to start a software update campaign.
kSyncing	0x01	V-UCM is providing the list of installed SWCLs (GetSwClusterInfo) or computing the list of SWCLs to install (SwPackageInventory).
kVehiclePackageTransferring	0x02	A vehicle package is being transferred to V-UCM.
kSoftwarePackage_ Transferring	0x03	V-UCM is transferring software packages to the UCM subordinates.
kProcessing	0x04	The processing of software packages on UCM subordinates is ongoing. The transferring of software packages may still occur.
kActivating	0x05	The activation of SWCLs on UCM subordinates is ongoing.
kVehicleChecking	0x06	V-UCM is performing post-activation checks (OEM specific).
kCancelling	0x07	V-UCM is rolling-back the activated SWCLs on the UCM subordinates.
kRollingBackFailed	0x08	One UCM failed to revert changes introduced with processed packages.

⌋

9.1.12 CampaignStateProgressInfoType

[SWS_VUCM_01022] Definition of ImplementationDataType CampaignStateProgressInfoType

Upstream requirements: RS_VUCM_00035, RS_VUCM_00042

⌈

Name	CampaignStateProgressInfoType
Namespace	ara::vucm
Kind	STRUCTURE
Sub-elements	campaignState `CampaignStateType` progress `uint8_t` estimatedDuration `uint32_t`
Derived from	-
Description	Provides state and progress information of the work done in current campaign. The progress is set to a value representing the progress between 0% and 100% (0x00 ... 0x64). The estimatedDuration is set in seconds, where 0 determines that no estimation is available if the progress in not equal to 100%. The campaignState is set to the current state of the V-UCM state machine.

⌋

9.1.13 TransferStateType

[SWS_VUCM_01178] Definition of ImplementationDataType TransferStateType

Upstream requirements: RS_VUCM_00035, RS_VUCM_00042

⌈

Name	TransferStateType	
Namespace	ara::vucm	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Represents the state of an update from OTA Client perspective.	
Range / Symbol	***Limit***	***Description***
kIdle	0x00	V-UCM is ready to start a software update campaign.
kSyncing	0x01	V-UCM is providing the list of installed SWCLs (GetSwClusterInfo) or computing the list of SWCLs to install (SwPackageInventory).
kTransferring	0x02	Vehicle or Software Packages are being transferred.
kUpdating	0x03	Software Clusters are being updated in the vehicle.
kCancelling	0x04	An error occurred, campaign is being cancelled, reverting changes.
kRollingBackFailed	0x05	One UCM failed to revert changes introduced with processed packages.

⌋

9.1.14 SafetyConditionType

[SWS_VUCM_01114] Definition of ImplementationDataType SafetyConditionType

Upstream requirements: RS_VUCM_00037

⌈

Name	SafetyConditionType
Namespace	ara::vucm
Kind	STRING
Derived from	-
Description	The type of the Safety Conditions.

⌋

9.1.15 VehicleConditionCollectionType

[SWS_VUCM_01309] Definition of ImplementationDataType VehicleCondition CollectionType

Upstream requirements: RS_VUCM_00037

⌈

Name	VehicleConditionCollectionType
Namespace	ara::vucm
Kind	VECTOR <VehicleConditionType>
Derived from	-
Description	Represents a dynamic size array of vehicle safety conditions.

⌋

9.1.16 VehicleConditionType

[SWS_VUCM_01310] Definition of ImplementationDataType VehicleCondition Type

Upstream requirements: RS_VUCM_00037

⌈

Name	VehicleConditionType
Namespace	ara::vucm
Kind	STRUCTURE
Sub-elements	condition `SafetyConditionType` state `SafetyStateType`
Derived from	-
Description	Safety state of one safety condition.

⌋

9.1.17 SafetyStateType

[SWS_VUCM_01138] Definition of ImplementationDataType SafetyStateType

Upstream requirements: RS_VUCM_00037

⌈

Name	SafetyStateType	
Namespace	ara::vucm	
Kind	TYPE_REFERENCE	
Derived from	`uint8_t`	
Description	Represents the vehicle safety state.	
Range / Symbol	***Limit***	***Description***
Safe	0x00	Safe Safety State.
NotSafe	0x01	Not safe Safety State.
NotSupported	0x02	Unsupported Safety State.

⌋

9.1.18 SwNameVersionVectorType

[SWS_VUCM_00177] Definition of ImplementationDataType SwNameVersionVectorType

Upstream requirements: RS_VUCM_00037

⌈

Name	SwNameVersionVectorType
Namespace	ara::vucm
Kind	VECTOR <SwNameVersionType>
Derived from	-
Description	Represents a dynamic size array of Software Name and Version

⌋

9.1.19 VehicleUCMInfo

[SWS_VUCM_00307] Definition of ImplementationDataType VehicleUCMInfo

Upstream requirements: RS_VUCM_00035

⌈

Name	VehicleUCMInfo
Namespace	ara::vucm
Kind	STRUCTURE
Sub-elements	vehicleUCMVersion StrongRevisionLabelString ucmsInfoVector UCMIdentifiersAndVersionsType
Derived from	-
Description	Represents version information of Vehicle UCM and a dynamic size array of UCM subordinates Identifer and Version.

⌋

9.1.20 UCMIdentifiersAndVersionsType

[SWS_VUCM_00308] Definition of ImplementationDataType UCMIdentifiersAndVersionsType

Upstream requirements: RS_VUCM_00035

⌈

Name	UCMIdentifiersAndVersionsType
Namespace	ara::vucm
Kind	VECTOR <`UCMIdentifierAndVersionType`>
Derived from	-
Description	Represents a vector of UCM Module Instantiation numbers and versions of UCMs.

⌋

9.1.21 SwPackageDescType

[SWS_VUCM_00268] Definition of ImplementationDataType SwPackageDescType

Upstream requirements: RS_VUCM_00033

⌈

Name	SwPackageDescType
Namespace	ara::vucm
Kind	STRUCTURE
Sub-elements	swDesc `SwNameVersionType` packageAction `ActionType` duration `uint32_t` typeApproval `StringType` license `StringType` releaseNotes `StringType` size `uint64_t`
Derived from	-
Description	Contains general information related to SoftwarePackage that can be used by Human Interface.

⌋

9.1.22 SwPackageDescVectorType

[SWS_VUCM_00269] Definition of ImplementationDataType SwPackageDescVectorType

Upstream requirements: RS_VUCM_00033

⌈

Name	SwPackageDescVectorType
Namespace	ara::vucm
Kind	VECTOR <SwPackageDescType>
Derived from	-
Description	Represents a dynamic size array of SwPackageDescType.

⌋

9.1.23 VehiclePackageDescriptionType

[SWS_VUCM_01140] Definition of ImplementationDataType VehiclePackageDescriptionType

Upstream requirements: RS_VUCM_00033

⌈

Name	VehiclePackageDescriptionType
Namespace	ara::vucm
Kind	STRUCTURE
Sub-elements	estimatedDurationOfCampaign `uint64_t` purposeOfUpdate `StringType` swPackageDesc `SwPackageDescVectorType`
Derived from	-
Description	Contains the general information from the Vehicle Package that can be used by Human Interface.

⌋

9.2 Provided Service Interfaces

9.2.1 Vehicle Package Management

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

Port

[SWS_VUCM_00178] Definition of Port VehiclePackageManagement provided by functional cluster VUCM

Upstream requirements: RS_VUCM_00035

⌈

Name	VehiclePackageManagement		
Kind	ProvidedPort	***Interface***	`VehiclePackageManagement`
Description	Provide services like receiving and dispatching packages, provide history and status of update campaigns, generally used by OTA Client.		
Variation			

⌋

Service Interface

[SWS_VUCM_00181] Definition of ServiceInterface VehiclePackageManagement

Upstream requirements: RS_VUCM_00033, RS_VUCM_00034, RS_VUCM_00035, RS_VUCM_-00039, RS_VUCM_00042, RS_VUCM_00043

⌈

Name	VehiclePackageManagement
Namespace	ara::vucm
Version	1.0
Fields	• `TransferState` • `CampaignState` • `RequestedPackage` • `VehicleConditionCollection`
Methods	• `CancelCampaign` • `AllowCampaign` • `DeleteTransfer` • `GetCampaignHistory` • `GetSwClusterInfo` • `GetSwPackages` • `GetVehicleUCMInfo` • `SwPackageInventory` • `TransferData` • `TransferExit` • `TransferStart` • `TransferVehiclePackage`

⌋

[SWS_VUCM_01156] Definition of Field VehiclePackageManagement.TransferState

Upstream requirements: RS_VUCM_00035, RS_VUCM_00036, RS_VUCM_00042, RS_VUCM_00043

⌈

Field	TransferState
Description	The current status of Campaign from an OTA Client perspective.
Version	1.0
Type	TransferStateType
HasGetter	true
HasNotifier	true
HasSetter	false
Enclosing Service Interface	VehiclePackageManagement

⌋

[SWS_VUCM_01157] Definition of Field VehiclePackageManagement.CampaignState

Upstream requirements: RS_VUCM_00035, RS_VUCM_00036, RS_VUCM_00042, RS_VUCM_00043

⌈

Field	CampaignState
Description	The current status of Campaign.
Version	1.0
Type	CampaignStateProgressInfoType
HasGetter	true
HasNotifier	true
HasSetter	false
Enclosing Service Interface	VehiclePackageManagement

⌋

[SWS_VUCM_01158] Definition of Field VehiclePackageManagement.RequestedPackage

Upstream requirements: RS_VUCM_00043

⌈

Field	RequestedPackage
Description	Software Package to be transferred to V-UCM, containing Software Package name and version as defined in Vehicle Package.
Version	1.0

▽

△

Type	SwNameVersionType
HasGetter	true
HasNotifier	true
HasSetter	false
Enclosing Service Interface	VehiclePackageManagement

⌋

[SWS_VUCM_01159] Definition of Field VehiclePackageManagement.VehicleConditionCollection

Upstream requirements: RS_VUCM_00038, RS_VUCM_00043

⌈

Field	VehicleConditionCollection
Description	A set of safety conditions along with their corresponding states.
Version	1.0
Type	VehicleConditionCollectionType
HasGetter	true
HasNotifier	true
HasSetter	false
Enclosing Service Interface	VehiclePackageManagement

⌋

[SWS_VUCM_01160] Definition of Method VehiclePackageManagement.CancelCampaign

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037, RS_VUCM_00043

⌈

Method	CancelCampaign	
Description	This method aborts an ongoing campaign processing of a Vehicle Package.	
Version	1.0	
FireAndForget	false	
Parameter	disableCampaign	
	Description	To forbid new campaign
	Type	bool
	Variation	
	Direction	IN
Application Errors	kOperationNotPermitted	The operation is not supported in the current context.
Application Errors	kCancelFailed	Cancel failed.

▽

△

Enclosing Service Interface	VehiclePackageManagement

⌋

[SWS_VUCM_01161] Definition of Method VehiclePackageManagement.AllowCampaign

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037, RS_VUCM_00043

⌈

Method	AllowCampaign
Description	To allow a new campaign to start
Version	1.0
FireAndForget	false
Enclosing Service Interface	VehiclePackageManagement

⌋

[SWS_VUCM_01162] Definition of Method VehiclePackageManagement.DeleteTransfer

Upstream requirements: RS_VUCM_00043

⌈

Method	DeleteTransfer	
Description	Delete a transferred Software or Vehicle Package.	
Version	1.0	
FireAndForget	false	
Parameter	id	
	Description	Transfer ID of the currently running request.
	Type	TransferIdType
	Variation	
	Direction	IN
Application Errors	kTransferId-Invalid	The Transfer ID is invalid.
Application Errors	kOpera-tionNotPer-mitted	The operation is not supported in the current context.
Enclosing Service Interface	VehiclePackageManagement	

⌋

[SWS_VUCM_01167] Definition of Method VehiclePackageManagement.GetCampaignHistory

Upstream requirements: RS_VUCM_00034

⌈

Method	GetCampaignHistory	
Description	Getter method to retrieve all actions that have been performed by V-UCM.	
Version	1.0	
FireAndForget	false	
Parameter	timestampGE	
	Description	Earliest timestamp (inclusive)
	Type	`uint64_t`
	Variation	
	Direction	IN
Parameter	timestampLT	
	Description	Latest timestamp (exclusive)
	Type	`uint64_t`
	Variation	
	Direction	IN
Parameter	campaignHistory	
	Description	The history of all actions that have been performed by V-UCM.
	Type	`CampaignHistoryVectorType`
	Variation	
	Direction	OUT
Application Errors	`kUCMNotAvailableOnTheNetwork`	UCM subordinate not available on the network
Enclosing Service Interface	`VehiclePackageManagement`	

⌋

[SWS_VUCM_01168] Definition of Method VehiclePackageManagement.GetSwClusterInfo

Upstream requirements: RS_VUCM_00033

⌈

Method	GetSwClusterInfo	
Description	This method returns the information of the Software Clusters present in the Adaptive Platform, aggregated from UCM Subordinates or Flashing Adapters.	
Version	1.0	
FireAndForget	false	
Parameter	swInfo	
	Description	List of installed SoftwareClusters that are in state kPresent.
	Type	`SwClusterInfoVectorType`

▽

△

	Variation	
	Direction	OUT
Application Errors	`kUCMNotAvailableOnTheNetwork`	UCM subordinate not available on the network
Enclosing Service Interface	`VehiclePackageManagement`	

⌋

[SWS_VUCM_01169] Definition of Method VehiclePackageManagement.GetSwPackages

Upstream requirements: RS_VUCM_00033, RS_VUCM_00042

⌈

Method	GetSwPackages	
Description	This method returns the Software Packages that are part of current campaign handled by V-UCM.	
Version	1.0	
FireAndForget	false	
Parameter	packages	
	Description	List of Software Packages.
	Type	`SwPackageInfoVectorType`
	Variation	
	Direction	OUT
Application Errors	`kUCMNotAvailableOnTheNetwork`	UCM subordinate not available on the network
Enclosing Service Interface	`VehiclePackageManagement`	

⌋

[SWS_VUCM_01170] Definition of Method VehiclePackageManagement.GetVehicleUCMInfo

Upstream requirements: RS_VUCM_00033

⌈

Method	GetVehicleUCMInfo	
Description	This method returns Vehicle UCM version and list of UCM Indentifiers and Versions.	
Version	1.0	
FireAndForget	false	
Parameter	vehicleUCMInfo	
	Description	version of available Vehicle UCM and UCM subordinates.

▽

△

	Type	VehicleUCMInfo
	Variation	
	Direction	OUT
Enclosing Service Interface	VehiclePackageManagement	

⌋

[SWS_VUCM_01171] Definition of Method VehiclePackageManagement.SwPackageInventory

Upstream requirements: RS_VUCM_00033

⌈

Method	SwPackageInventory	
Description		
Version	1.0	
FireAndForget	false	
Parameter	availableSoftwarePackages	
	Description	List of available Software Packages in Backend corresponding to VIN.
	Type	SwNameVersionVectorType
	Variation	
	Direction	IN
Parameter	requiredSoftwarePackages	
	Description	List of Software Packages to be sent to V-UCM.
	Type	SwNameVersionVectorType
	Variation	
	Direction	OUT
Application Errors	kUCMNotAvailableOnTheNetwork	UCM subordinate not available on the network
Enclosing Service Interface	VehiclePackageManagement	

⌋

[SWS_VUCM_01163] Definition of Method VehiclePackageManagement.TransferData

Upstream requirements: RS_VUCM_00043

⌈

Method	TransferData
Description	Block-wise transfer of a Software or Vehicle Package to V-UCM.
Version	1.0

▽

△

FireAndForget	false	
Parameter	id	
	Description	Transfer ID.
	Type	`TransferIdType`
	Variation	
	Direction	IN
Parameter	data	
	Description	Data block of the Software or Vehicle Package.
	Type	`ByteVectorType`
	Variation	
	Direction	IN
Parameter	blockCounter	
	Description	Block counter value of the current block.
	Type	`uint64_t`
	Variation	
	Direction	IN
Application Errors	`kOperationNotPermitted`	The operation is not supported in the current context.
Application Errors	`kTransferIdInvalid`	The Transfer ID is invalid.
Application Errors	`kBlockIncorrect`	The same block number is received twice.
Application Errors	`kBlockSizeIncorrect`	The size of the block exceeds the provided block size from TransferStart or TransferVehiclePackage.
Application Errors	`kSizeIncorrect`	The size of the Software or Vehicle Package exceeds the provided size in TransferStart.
Application Errors	`kMemoryInsufficient`	Insufficient memory to perform operation.
Application Errors	`kTransferFailed`	UCM cannot persist transferred block.
Application Errors	`kBlockInconsistent`	Consistency check for transferred block failed.
Application Errors	`kPackageFormatUnsupported`	The Vehicle Package or Software Package archiving format is not supported.
Application Errors	`kAuthenticationFailed`	Package authentication failed.
Application Errors	`kPackageManifestInvalid`	Package manifest could not be read.
Application Errors	`kPackageVersionIncompatible`	The version of the Software or Vehicle Package to be processed is not compatible with the current version of UCM or V-UCM.
Application Errors	`kPackageInconsistent`	Package integrity check failed.
Application Errors	`kOldVersion`	Software Package version is too old.
Enclosing Service Interface	`VehiclePackageManagement`	

⌋

[SWS_VUCM_01164] Definition of Method VehiclePackageManagement.TransferExit

Upstream requirements: RS_VUCM_00043

⌈

Method	TransferExit	
Description	Finish the transfer of a Software or Vehicle Package to V-UCM.	
Version	1.0	
FireAndForget	false	
Parameter	id	
	Description	Transfer ID of the currently running request.
	Type	`TransferIdType`
	Variation	
	Direction	IN
Application Errors	`kOperationNotPermitted`	The operation is not supported in the current context.
Application Errors	`kTransferIdInvalid`	The Transfer ID is invalid.
Application Errors	`kDataInsufficient`	TransferExit has been called but total transferred data size does not match expected data size provided with TransferStart call.
Application Errors	`kAuthenticationFailed`	Package authentication failed.
Application Errors	`kPackageFormatUnsupported`	The Vehicle Package or Software Package archiving format is not supported.
Application Errors	`kPackageInconsistent`	Package integrity check failed.
Application Errors	`kPackageVersionIncompatible`	The version of the Software or Vehicle Package to be processed is not compatible with the current version of UCM or V-UCM.
Application Errors	`kPackageManifestInvalid`	Package manifest could not be read.
Application Errors	`kDependencyMissing`	Activation is not allowed because dependencies are missing.
Application Errors	`kOldVersion`	Software Package version is too old.
Enclosing Service Interface	`VehiclePackageManagement`	

⌋

[SWS_VUCM_01165] Definition of Method VehiclePackageManagement.TransferStart

Upstream requirements: RS_VUCM_00043

⌈

Method	TransferStart	
Description	Start the transfer of a Software Package. The name of the Software Package to be transferred to V-UCM must be provided. V-UCM will generate a Transfer ID for subsequent calls to TransferData, TransferExit, DeleteTransfer. Size of Software Package to be used to transfer to UCM subordinate is available in the Vehicle Package and its contained Software Package Manifests.	
Version	1.0	
FireAndForget	false	
Parameter	softwarePackageName	
	Description	Software Package Short Name of the Software Package to be transferred.
	Type	`SwPackageNameType`
	Variation	
	Direction	IN
Parameter	id	
	Description	Return TransferId.
	Type	`TransferIdType`
	Variation	
	Direction	OUT
Parameter	blockSize	
	Description	Size of the blocks to be received with TransferData method.
	Type	`uint32_t`
	Variation	
	Direction	OUT
Application Errors	`kPackageUnexpected`	The Software Package name does not correspond to the RequestedPackage field value.
Application Errors	`kMemoryInsufficient`	Insufficient memory to perform operation.
Enclosing Service Interface	`VehiclePackageManagement`	

⌋

[SWS_VUCM_01166] Definition of Method VehiclePackageManagement.TransferVehiclePackage

Upstream requirements: RS_VUCM_00043

⌈

Method	TransferVehiclePackage
Description	Start the transfer of a Vehicle Package. The size of the Vehicle Package to be transferred to V-UCM must be provided. V-UCM will generate a Transfer ID for subsequent calls to TransferData, TransferExit, ProcessSwPackage, DeleteTransfer. This call starts a new campaign.
Version	1.0
FireAndForget	false

▽

△

Parameter	size	
	Description	Size (in bytes) of the Vehicle Package to be transferred.
	Type	`uint64_t`
	Variation	
	Direction	IN
Parameter	id	
	Description	Return TransferId.
	Type	`TransferIdType`
	Variation	
	Direction	OUT
Parameter	blockSize	
	Description	Size of the blocks to be received with TransferData method.
	Type	`uint32_t`
	Variation	
	Direction	OUT
Application Errors	`kBusyWithCampaign`	Campaign has already started.
Application Errors	`kNewCampaignDisabled`	New campaigns are disabled, calling AllowCampaign will enable new campaigns.
Application Errors	`kMemoryInsufficient`	Insufficient memory to perform operation.
Enclosing Service Interface	`VehiclePackageManagement`	

⌋

9.2.2 Vehicle Driver Application Interface

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

Port

[SWS_VUCM_00180] Definition of Port VehicleDriverApplicationInterface provided by functional cluster VUCM

Upstream requirements: RS_VUCM_00038, RS_VUCM_00043

⌈

Name	VehicleDriverApplicationInterface		
Kind	ProvidedPort	***Interface***	`VehicleDriverApplicationInterface`
Description	To be used by Vehicle Driver Interface Adaptive Application in order to interact with human user.		
Variation			

⌋

Service Interface

[SWS_VUCM_00182] Definition of ServiceInterface VehicleDriverApplicationInterface

Upstream requirements: RS_VUCM_00033, RS_VUCM_00034, RS_VUCM_00038, RS_VUCM_00043

⌈

Name	VehicleDriverApplicationInterface
Namespace	ara::vucm
Version	1.0
Fields	• ApprovalRequired • CampaignState • VehicleConditionCollection
Methods	• CancelCampaign • AllowCampaign • Approve • ReportUnsupportedSafetyConditions • GetCampaignHistory • GetSwClusterInfo • GetSwPackageDescription • GetVehiclePackageDescription

⌋

[SWS_VUCM_01142] Definition of Field VehicleDriverApplicationInterface.ApprovalRequired

Upstream requirements: RS_VUCM_00038, RS_VUCM_00043

⌈

Field	ApprovalRequired
Description	Flag to inform Adaptive Application if approval from Vehicle Driver is required at current state based on Vehicle Package Manifest.
Version	1.0
Type	bool
HasGetter	true
HasNotifier	true
HasSetter	false
Enclosing Service Interface	VehicleDriverApplicationInterface

⌋

[SWS_VUCM_01143] Definition of Field VehicleDriverApplicationInterface.CampaignState

Upstream requirements: RS_VUCM_00035, RS_VUCM_00036, RS_VUCM_00042, RS_VUCM_00043

⌈

Field	CampaignState
Description	The current status of Campaign.
Version	1.0
Type	`CampaignStateProgressInfoType`
HasGetter	true
HasNotifier	true
HasSetter	false
Enclosing Service Interface	`VehicleDriverApplicationInterface`

⌋

[SWS_VUCM_01144] Definition of Field VehicleDriverApplicationInterface.VehicleConditionCollection

Upstream requirements: RS_VUCM_00038, RS_VUCM_00043

⌈

Field	VehicleConditionCollection
Description	A set of safety conditions along with its corresponding states.
Version	1.0
Type	`VehicleConditionCollectionType`
HasGetter	true
HasNotifier	true
HasSetter	false
Enclosing Service Interface	`VehicleDriverApplicationInterface`

⌋

[SWS_VUCM_01145] Definition of Method VehicleDriverApplicationInterface.CancelCampaign

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037, RS_VUCM_00043

⌈

Method	CancelCampaign
Description	This method aborts an ongoing campaign processing of a Vehicle Package.
Version	1.0
FireAndForget	false

▽

△

Parameter	disableCampaign	
	Description	To forbid new campaign
	Type	`bool`
	Variation	
	Direction	IN
Application Errors	`kOperationNotPermitted`	The operation is not supported in the current context.
Application Errors	`kCancelFailed`	Cancel failed.
Enclosing Service Interface	`VehicleDriverApplicationInterface`	

⌋

[SWS_VUCM_01146] Definition of Method VehicleDriverApplicationInterface.AllowCampaign

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037, RS_VUCM_00043

⌈

Method	AllowCampaign
Description	To allow a new campaign to start
Version	1.0
FireAndForget	false
Enclosing Service Interface	`VehicleDriverApplicationInterface`

⌋

[SWS_VUCM_01147] Definition of Method VehicleDriverApplicationInterface.Approve

Upstream requirements: RS_VUCM_00038, RS_VUCM_00043

⌈

Method	Approve
Description	Called by Adaptive Application to inform V-UCM of the driver's notification approval
Version	1.0
FireAndForget	false
Enclosing Service Interface	`VehicleDriverApplicationInterface`

⌋

[SWS_VUCM_01148] Definition of Method VehicleDriverApplicationInterface.ReportUnsupportedSafetyConditions

Upstream requirements: RS_VUCM_00038, RS_VUCM_00043

⌈

Method	ReportUnsupportedSafetyConditions	
Description	Called by Adaptive Application to inform V-UCM on all unsupported safety conditions	
Version	1.0	
FireAndForget	false	
Parameter	UnsupportedSafetyConditions	
	Description	The list of all unsupported safety conditions
	Type	`VehicleConditionCollectionType`
	Variation	
	Direction	IN
Enclosing Service Interface	`VehicleDriverApplicationInterface`	

⌋

[SWS_VUCM_01149] Definition of Method VehicleDriverApplicationInterface.GetCampaignHistory

Upstream requirements: RS_VUCM_00034

⌈

Method	GetCampaignHistory	
Description	Getter method to retrieve all actions that have been performed by V-UCM.	
Version	1.0	
FireAndForget	false	
Parameter	timestampGE	
	Description	Earliest timestamp (inclusive)
	Type	`uint64_t`
	Variation	
	Direction	IN
Parameter	timestampLT	
	Description	Latest timestamp (exclusive)
	Type	`uint64_t`
	Variation	
	Direction	IN
Parameter	history	
	Description	The history of all actions that have been performed by V-UCM.
	Type	`CampaignHistoryVectorType`
	Variation	
	Direction	OUT

▽

△

Application Errors	kUCMNo-tAvail-ableOn-TheNetwork	UCM subordinate not available on the network
Enclosing Service Interface	VehicleDriverApplicationInterface	

⌋

[SWS_VUCM_01150] Definition of Method VehicleDriverApplicationInterface.GetSwClusterInfo

Upstream requirements: RS_VUCM_00033

⌈

Method	GetSwClusterInfo	
Description	This method returns the information of the Software Clusters present in the Adaptive Platform, aggregated from UCM Subordinates or Flashing Adapters.	
Version	1.0	
FireAndForget	false	
Parameter	swInfo	
	Description	List of installed SoftwareClusters that are in state kPresent.
	Type	SwClusterInfoVectorType
	Variation	
	Direction	OUT
Application Errors	kUCMNo-tAvail-ableOn-TheNetwork	UCM subordinate not available on the network
Enclosing Service Interface	VehicleDriverApplicationInterface	

⌋

[SWS_VUCM_01151] Definition of Method VehicleDriverApplicationInterface.GetSwPackageDescription

Upstream requirements: RS_VUCM_00033

⌈

Method	GetSwPackageDescription	
Description	This method returns the general information of the Software Packages that are part of current campaign handled by V-UCM.	
Version	1.0	
FireAndForget	false	
Parameter	packages	
	Description	List of Software Packages.
	Type	SwPackageDescVectorType

▽

△

	Variation	
	Direction	OUT
Enclosing Service Interface	VehicleDriverApplicationInterface	

⌋

[SWS_VUCM_01152] Definition of Method VehicleDriverApplicationInterface.GetVehiclePackageDescription

Upstream requirements: RS_VUCM_00033

⌈

Method	GetVehiclePackageDescription	
Description	This method returns the Vehicle Package relevant for communication with user according to UN ECE 156.	
Version	1.0	
FireAndForget	false	
Parameter	vehiclePackageDescription	
	Description	Vehicle Package metadata.
	Type	VehiclePackageDescriptionType
	Variation	
	Direction	OUT
Enclosing Service Interface	VehicleDriverApplicationInterface	

⌋

9.2.3 Vehicle State Manager

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

Port

[SWS_VUCM_00179] Definition of Port VehicleStateManagerInterface provided by functional cluster VUCM

Upstream requirements: RS_VUCM_00037, RS_VUCM_00043

⌈

Name	VehicleStateManagerInterface		
Kind	ProvidedPort	***Interface***	VehicleStateManagerInterface
Description	To receive the vehicle safety states from Vehicle State Manager Adaptive Application.		
Variation			

⌋

Service Interface

[SWS_VUCM_00183] Definition of ServiceInterface VehicleStateManagerInterface

Upstream requirements: RS_VUCM_00035, RS_VUCM_00037

⌈

Name	VehicleStateManagerInterface
Namespace	ara::vucm
Version	1.0
Fields	• CampaignState • VehicleConditionCollection
Methods	• PublishSafetyState • VehicleCheck

⌋

[SWS_VUCM_01172] Definition of Field VehicleStateManagerInterface.CampaignState

Upstream requirements: RS_VUCM_00035, RS_VUCM_00036, RS_VUCM_00042, RS_VUCM_00043

⌈

Field	CampaignState
Description	The current status of Campaign.
Version	1.0
Type	CampaignStateType
HasGetter	true
HasNotifier	true
HasSetter	false
Enclosing Service Interface	VehicleStateManagerInterface

⌋

[SWS_VUCM_01173] Definition of Field VehicleStateManagerInterface.VehicleConditionCollection

Upstream requirements: RS_VUCM_00038, RS_VUCM_00043

⌈

Field	VehicleConditionCollection
Description	A set of safety conditions, for which values are available in the Vehicle Package, to be computed by the Vehicle State Manager Adaptive Application, along with its corresponding computed state.
Version	1.0
Type	VehicleConditionCollectionType
HasGetter	true
HasNotifier	true
HasSetter	false
Enclosing Service Interface	VehicleStateManagerInterface

⌋

[SWS_VUCM_01174] Definition of Method VehicleStateManagerInterface.PublishSafetyState

Upstream requirements: RS_VUCM_00037, RS_VUCM_00043

⌈

Method	PublishSafetyState	
Description	Method called by Vehicle State Manager Adaptive Application when safety state is changed	
Version	1.0	
FireAndForget	false	
Parameter	safetyStates	
	Description	Safety conditions computed by the Vehicle State Manager Adaptive Application.
	Type	VehicleConditionCollectionType
	Variation	
	Direction	IN
Enclosing Service Interface	VehicleStateManagerInterface	

⌋

[SWS_VUCM_01179] Definition of Method VehicleStateManagerInterface.VehicleCheck

Upstream requirements: RS_VUCM_00037

⌈

Method	VehicleCheck	
Description	Method for Vehicle State Manager to inform V-UCM of vehicle check resolution	
Version	1.0	
FireAndForget	false	
Parameter	vehicleCheckResolution	
	Description	Vehicle check resolution. True if check succeeded.
	Type	`bool`
	Variation	
	Direction	IN
Enclosing Service Interface	`VehicleStateManagerInterface`	

⌋

9.3 Required Interface

[SWS_VUCM_01021] Definition of Port PackageManagement required by functional cluster VUCM

Upstream requirements: RS_VUCM_00035, RS_VUCM_00036, RS_VUCM_00043

⌈

Name	PackageManagement		
Kind	RequiredPort	***Interface***	`PackageManagement`
Description	Provides for the Adaptive Platform Machine services like receiving, processing and activating Software Packages. Also providing update history, status and Software Package and Software Cluster information.		
Variation			

⌋

9.4 Application Errors

9.4.1 Application Error Domain

9.4.1.1 UCMErrorDomain

This section lists all application errors of the `V-UCM`.

[SWS_VUCM_00136] Definition of Application Error Domain of functional cluster VUCM

Upstream requirements: RS_VUCM_00039, RS_VUCM_00043

⌈

Name	*Code*	*Description*
kAuthenticationFailed	8	Package authentication failed.
kBlockInconsistent	25	Consistency check for transferred block failed.
kBlockIncorrect	2	The same block number is received twice.
kBlockSizeIncorrect	30	The size of the block exceeds the provided block size from Transfer Start or TransferVehiclePackage.
kBusyWithCampaign	34	Campaign has already started.
kCancelFailed	16	Cancel failed.
kChecksumDescriptionInvalid	35	Checksum attribute not recognised.
kDataInsufficient	6	TransferExit has been called but total transferred data size does not match expected data size provided with TransferStart call.
kDeltaIncompatible	29	Delta package dependency check failed.
kDependencyMissing	21	Activation is not allowed because dependencies are missing.
kMemoryInsufficient	1	Insufficient memory to perform operation.
kNewCampaignDisabled	31	New campaigns are disabled, calling AllowCampaign will enable new campaigns.
kNotAbleToRevertPackages	15	RevertProcessedSwPackages failed.
kOldVersion	9	Software Package version is too old.
kOperationNotPermitted	5	The operation is not supported in the current context.
kPackageFormatUnsupported	40	The Vehicle Package or Software Package archiving format is not supported.
kPackageInconsistent	7	Package integrity check failed.
kPackageManifestInvalid	13	Package manifest could not be read.
kPackageUnexpected	32	The Software Package name does not correspond to the RequestedPackage field value.
kPackageVersionIncompatible	24	The version of the Software or Vehicle Package to be processed is not compatible with the current version of UCM or V-UCM.
kPrepareUpdateFailed	19	Error during update preparation step.
kProcessSwPackageCanceled	22	The processing operation has been interrupted by a Cancel() call.
kProcessedSoftwarePackageInconsistent	23	The processed Software Package integrity check has failed.
kServiceBusy	12	Another processing is already ongoing and therefore the current processing request has to be rejected.
kSizeIncorrect	3	The size of the Software or Vehicle Package exceeds the provided size in TransferStart.
kSoftwareClusterMissing	37	The Software Cluster is not present in the Machine.
kSwclRemovalDenied	39	Attempt to remove PLATFORM_CORE Software Cluster.
kTransferFailed	38	UCM cannot persist transferred block.
kTransferIdInvalid	4	The Transfer ID is invalid.
kUCMNotAvailableOnTheNetwork	42	UCM subordinate not available on the network
kUpdateSessionRejected	33	Start of an update session was rejected by State Management
kVerificationFailed	36	State Management returned verification failure

⌋

10 Configuration

The configuration structure of Vehicle Update And Configuration Management is described in [7] by Platform Module Development / Update and Configuration Management. This chapter defines default values and semantic constraints for this configuration model.

10.1 Default Values

This section defines the default values for attributes defined in [7].

10.2 Semantic Constraints

This section defines semantic constraints for the configuration elements of Vehicle Update and Configuration Management defined in [7].

A Mentioned Manifest Elements

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

Chapter is generated.

Class	**CryptoServiceCertificate**			
Package	M2::AUTOSARTemplates::SystemTemplate::SecureCommunication			
Note	This meta-class represents the ability to model a cryptographic certificate. **Tags:** atp.recommendedPackage=CryptoServiceCertificates			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
algorithmFamily	CryptoCertificate AlgorithmFamilyEnum	0..1	attr	This attribute represents a description of the family of crypto algorithm used to generate public key and signature of the cryptographic certificate.
format	CryptoCertificateFormat Enum	0..1	attr	This attribute can be used to provide information about the format used to create the certificate
maximum Length	PositiveInteger	0..1	attr	This attribute represents the ability to define the maximum length of the certificate in bytes.
nextHigher Certificate	CryptoService Certificate	0..1	ref	The reference identifies the next higher certificate in the certificate chain.
serverName Identification	String	0..1	attr	Server Name Indication (SNI) is needed if the IP address hosts multiple servers (on the same port), each of them using a different certificate. If the client sends the SNI to the Server in the client hello, the server looks the SNI up in its certificate list and uses the certificate identified by the SNI.

Table A.1: CryptoServiceCertificate

Class	***Referrable*** (abstract)			
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable			
Note	Instances of this class can be referred to by their identifier (while adhering to namespace borders).			
Base	*ARObject*			
Subclasses	*AtpDefinition*, BswDistinguishedPartition, *BswModuleCallPoint*, BswModuleClientServerEntry, Bsw VariableAccess, CouplingPortTrafficClassAssignment, *CppImplementationDataTypeContextTarget*, *DiagnosticEnvModeElement*, EthernetPriorityRegeneration, ExclusiveAreaNestingOrder, *HwDescription Entity*, *ImplementationProps*, ModeTransition, *MultilanguageReferrable*, NmNetworkHandle, Pnc MappingIdent, *SingleLanguageReferrable*, SoConIPduIdentifier, SocketConnectionBundle, Someip RequiredEventGroup, TimeSyncServerConfiguration, TpConnectionIdent			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
shortName	Identifier	1	attr	This specifies an identifying shortName for the object. It needs to be unique within its context and is intended for humans but even more for technical reference. **Stereotypes:** atpIdentityContributor **Tags:** xml.enforceMinMultiplicity=true xml.sequenceOffset=-100

▽

△

Class	***Referrable*** (abstract)			
shortName Fragment	ShortNameFragment	*	aggr	This specifies how the Referrable.shortName is composed of several shortNameFragments. **Tags:** xml.sequenceOffset=-90

Table A.2: Referrable

Class	**SoftwareCluster**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
Note	This meta-class represents the ability to define an uploadable software-package, i.e. the SoftwareCluster shall contain all software and configuration for a given purpose. **Tags:** atp.recommendedPackage=SoftwareClusters			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
artifact Checksum	ArtifactChecksum	*	aggr	This aggregation carries the checksums for artifacts contained in the enclosing SoftwareCluster. Please note that the value of these checksums is only applicable at the time of configuration. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=artifactChecksum.shortName, artifact Checksum.uri
artifactLocator	ArtifactLocator	*	aggr	This aggregation represents the artifact locations that are relevant in the context of the enclosing SoftwareCluster
claimed FunctionGroup	ModeDeclarationGroup Prototype	*	ref	Each SoftwareCluster can reserve the usage of a given functionGroup such that no other SoftwareCluster is allowed to use it
conflictsTo	SoftwareCluster DependencyFormula	0..1	aggr	This aggregation handles conflicts. If it yields true then the SoftwareCluster shall not be installed. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=conflictsTo
contained ARElement	ARElement	*	ref	This reference represents the collection of model elements that cannot derive from UploadablePackage Element and that contribute to the completeness of the definition of the SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=containedARElement
containedFibex Element	FibexElement	*	ref	This allows for referencing FibexElements that need to be considered in the context of a SoftwareCluster.
contained Package Element	UploadablePackage Element	*	ref	This reference identifies model elements that are required to complete the manifest content. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=containedPackageElement
contained Process	Process	*	ref	This reference represent the processes contained in the enclosing SoftwareCluster.
dependsOn	SoftwareCluster DependencyFormula	0..1	aggr	This aggregation can be taken to identify a dependency for the enclosing SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=dependsOn

▽

△

Class	**SoftwareCluster**			
design	SoftwareClusterDesign	*	ref	This reference represents the identification of all Software ClusterDesigns applicable for the enclosing Software Cluster. **Stereotypes:** atpUriDef
diagnostic Deployment Props	SoftwareCluster DiagnosticDeployment Props	0..1	ref	This reference identifies the applicable SoftwareCluster DiagnosticDeploymentProps that are applicable for the referencing SoftwareCluster.
installation Behavior	SoftwareCluster InstallationBehavior Enum	0..1	attr	This attribute controls the behavior of the SoftwareCluster in terms of installation.
license	Documentation	*	ref	This attribute allows for the inclusion of the full text of a license of the enclosing SoftwareCluster. In many cases open source licenses require the inclusion of the full license text to any software that is released under the respective license.
module Instantiation	AdaptiveModule Instantiation	*	ref	This reference identifies AdaptiveModuleInstantiations that need to be included with the SoftwareCluster in order to establish infrastructure required for the installation of the SoftwareCluster. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=moduleInstantiation
releaseNotes	Documentation	0..1	ref	This attribute allows for the explanations of changes since the previous version. The list of changes might require the creation of multiple paragraphs of test.
typeApproval	String	0..1	attr	This attribute carries the homologation information that may be specific for a given country.
vendorId	PositiveInteger	0..1	attr	Vendor ID of this Implementation according to the AUTOSAR vendor list.
vendor Signature	CryptoService Certificate	0..1	ref	This reference identifies the certificate that represents the vendor's signature.
version	StrongRevisionLabel String	0..1	attr	This attribute can be used to describe a version information for the enclosing SoftwareCluster.

Table A.3: SoftwareCluster

Class	**SoftwarePackage**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
Note	This meta-class represents the ability to formalize the content of a software package. **Tags:** atp.recommendedPackage=SoftwarePackages			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
actionType	SoftwarePackageAction TypeEnum	0..1	attr	This attribute defines the action to be taken in the step of processing the enclosing SoftwarePackage.
activationAction	SoftwarePackage ActivationActionEnum	0..1	attr	This attribute governs the action to be taken after the installation of the SoftwareCluster completed.
artifactLocator	ArtifactLocator	0..1	aggr	This attribute identifies the software package at configuration time, out of the context of an AUTOSAR model.

▽

△

Class	**SoftwarePackage**			
compressed Software PackageSize	PositiveInteger	0..1	attr	This size represents the size of the compressed Software Package.
deltaPackage Applicable Version	StrongRevisionLabel String	0..1	attr	This attribute identifies the version of the included SoftwareCluster for which the enclosing SoftwarePackage can be used as a delta update
estimated DurationOf Operation	TimeValue	0..1	attr	This attribute provides an estimation about how long the operation of the SoftwarePackage is going to take for its transfer, processing and activation when updated standalone (not within an update campaign)
minimum SupportedUcm Version	RevisionLabelString	0..1	attr	This attribute identifies the minimum supported version of the UCM for this SoftwarePackage.
packagerId	PositiveInteger	0..1	attr	This attribute identifies Id of the organization that provides the packager generating the SoftwarePackage.
packager Signature	CryptoService Certificate	0..1	ref	This reference identifies the certificate that represents the packager's signature.
purposeOf Update	Documentation	0..1	ref	The referenced Documentation is supposed to provide a description of the purpose of the update.
softwareCluster	SoftwareCluster	0..1	ref	This reference identifies the SoftwareCluster that belongs to the SoftwarePackage. The nature of this relation is actually more like an aggregation than a reference. But the relation is still modelled as a reference because two ARElements cannot aggregate each other.
uncompressed SoftwareCluster Size	PositiveInteger	0..1	attr	This attribute gives an indication about the storage that has to be available on the target.

Table A.4: SoftwarePackage

Primitive	**StrongRevisionLabelString**
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::PrimitiveTypes
Note	This primitive represents a revision label which identifies an object under version control. It represents a pattern which requires three integer numbers separated by a dot, representing from left to right Major Version, MinorVersion, PatchVersion and additional labels for pre-release version and build metadata. Legal patterns are for example: 1.0.0-alpha+001 1.0.0+20130313144700 1.0.0-beta+exp.sha.5114f85 **Tags:** xml.xsd.customType=STRONG-REVISION-LABEL-STRING xml.xsd.pattern=(0|[1-9]\d*)\.(0|[1-9]\d*)\.(0|[1-9]\d*)(-((0|[1-9]\d*|\d*[a-zA-Z-][0-9a-z A-Z-]*)(\.(0|[1-9]\d*|\d*[a-zA-Z-][0-9a-zA-Z-]*))*))?(\+([0-9a-zA-Z-]+(\.[0-9a-zA-Z-]+)*))? xml.xsd.type=string

Table A.5: StrongRevisionLabelString

Class	***UcmModuleInstantiation*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Ucm			
Note	This meta-class represents the ability to define the target-configuration of a UCM instantiation.			
Base	*ARObject*, *AdaptiveModuleInstantiation*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *NonOsModuleInstantiation*, *Referrable*			
Subclasses	UcmMasterModuleInstantiation, UcmSubordinateModuleInstantiation			
Aggregated by	*AtpClassifier*.atpFeature, Machine.moduleInstantiation			
Attribute	*Type*	*Mult.*	*Kind*	*Note*

▽

△

Class	***UcmModuleInstantiation*** (abstract)			
identifier	String	0..1	attr	This represents the identification of a UCM.
maxBlockSize	PositiveInteger	0..1	attr	This attribute denotes the maximum block size (unit: bytes) used in the UCM implementation.
version	StrongRevisionLabel String	0..1	attr	This attribute defines the software version of the UCM on this platform. Note that the definition of the version is required if the ability of the SoftwarePackage to require a minimum version of the UCM is utilized.

Table A.6: UcmModuleInstantiation

Class	**UcmRetryStrategy**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Ucm			
Note	This meta-class describes the configuration of the retry strategy for a sub-class of UcmModule Implementation.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	UcmMasterModuleInstantiation.blockInconsistent, UcmMasterModuleInstantiation.serviceBusy, UcmMasterModuleInstantiation.ucmNotAvailableOnTheNetwork, UcmMasterModuleInstantiation.updateSessionRejected, UcmSubordinateModuleInstantiation.prepareRollback, UcmSubordinateModuleInstantiation.prepareUpdate, UcmSubordinateModuleInstantiation.verifyUpdate			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
maximumNumberOfRetries	PositiveInteger	0..1	attr	This attribute defines the maximum number of time the UCM module instantiation shall attempt a retry.
retryIntervalTime	TimeValue	0..1	attr	This attribute defines the time (in seconds) between two retry attempts.

Table A.7: UcmRetryStrategy

Class	**UcmStep**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
Note	This meta-class represents one rollout step in which software packages are processed on a specific Ucm.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	VehicleRolloutStep.ucmProcessing			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
softwarePackageStep (ordered)	SoftwarePackageStep	*	aggr	This aggregation represents the sequence of activities to be carried out in the context of the respective UCM.
ucm	UcmDescription	0..1	ref	This reference identifies the UCM for which the rollout step applies.

Table A.8: UcmStep

Class	**UcmToTimeBaseResourceMapping**
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::Ucm
Note	This meta-class maps the UCM Module Instantiation to the TimeSync Module Instantiation. **Tags:** atp.recommendedPackage=FCInteractions

▽

△

Class	UcmToTimeBaseResourceMapping			
Base	*ARElement*, *ARObject*, *CollectableElement*, *FunctionalClusterInteractsWithFunctionalClusterMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeployment Element*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
timeBase Resource	TimeBaseResource	0..1	ref	This reference identifies the relevant TimeBaseResource.
ucm	UcmModuleInstantiation	0..1	ref	This reference identifies the relevant UcmModule Instantiation.

Table A.9: UcmToTimeBaseResourceMapping

Class	VehicleDriverNotification			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
Note	This meta-class provides the ability to configure a notification of the vehicle driver with respect to the update of vehicle software.			
Base	*ARObject*			
Aggregated by	VehiclePackage.driverNotification			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
approval Required	Boolean	0..1	attr	This attribute controls whether approval is required for the driver notification.
notificationState	VehicleDriver NotificationEnum	0..1	attr	This attribute is used to configure the notification state.

Table A.10: VehicleDriverNotification

Class	VehiclePackage			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
Note	This meta-class represents the ability to define a vehicle package for executing an update campaign. **Tags:** atp.recommendedPackage=VehiclePackages			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
driver Notification	VehicleDriver Notification	*	aggr	This aggregation provides the ability to configure the necessary driver notifications.
estimated DurationOf Campaign	TimeValue	0..1	attr	This attribute provides an estimation about how long the campaign based on the VehiclePackage is going to take.
maximum DurationOf Campaign	TimeValue	0..1	attr	Maximum time allowed for the campaign to be active until UCM Master automatically cancels the campaign.
minimum SupportedUcm MasterVersion	RevisionLabelString	0..1	attr	This attribute identifies the minimum supported version of the UCM Master for this VehiclePackage.
packager Signature	CryptoService Certificate	0..1	ref	This reference identifies the certificate that represents the packager's signature.
repository	UriString	0..1	attr	This attribute identifies the repository where the Vehicle Package is stored.

▽

△

Class	**VehiclePackage**			
rollout Qualification (ordered)	VehicleRolloutStep	*	aggr	This represents the rollout qualification.
ucm	UcmDescription	*	aggr	This aggregation represents the UcmDescriptions to be considered in the context of the VehiclePackage.
ucmMaster Fallback (ordered)	UcmDescription	*	ref	This reference lists the fallback order of Ucms that can take over the master role if the master goes down.
vehicle Description	Documentation	0..1	ref	This reference identifies the vehicle description.

Table A.11: VehiclePackage

Class	**VehicleRolloutStep**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution			
Note	This meta-class represents the ability to define a rollout-condition for a vehicle update campaign.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	VehiclePackage.rolloutQualification			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
safetyCondition	String	*	attr	This attribute represents a list of textual safety conditions (e.g.: close the driver window) that need to be fulfilled before the rollout step can proceed and need to be maintained while the campaign's rolloutQualification is executed.
ucmProcessing	UcmStep	*	aggr	This aggregation collects the UcmProcessingSteps that make up the rollout step.
violatedSafety Condition Behavior	ViolatedSafetyCondition BehaviorEnum	0..1	attr	This attribute provides options for the configuration of the reaction to a violated safety condition.

Table A.12: VehicleRolloutStep

Enumeration	**ViolatedSafetyConditionBehaviorEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::SoftwareDistribution
Note	This enumeration provides formal options for the configuration of the reaction to a violated safety condition.
Aggregated by	VehicleRolloutStep.violatedSafetyConditionBehavior
Literal	***Description***
cancelCampaign	This enumerator supports the ability to cancel the update campaign in response to a violated safety condition. **Tags:** atp.EnumerationLiteralIndex=1
waitForVehicleSafe State	This enumerator supports the ability to wait for the vehicle to acquire a safe state in response to a violated safety condition. **Tags:** atp.EnumerationLiteralIndex=0

Table A.13: ViolatedSafetyConditionBehaviorEnum

B Platform Extension API (normative)

The focus of the APIs in this section are for OEM-specific platform extensions. The abstraction of the interfaces is lower which could lead to a higher machine dependency.

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 to describe Inter-Functional Cluster (IFC) interfaces. In addition, the standardized public interfaces which are accessible by user space applications (see chapters 8 and 9) 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.

C.2 Interface Tables

D Packages distribution within vehicle detailed sequence examples

D.1 Collect information of present Software Clusters in vehicle

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

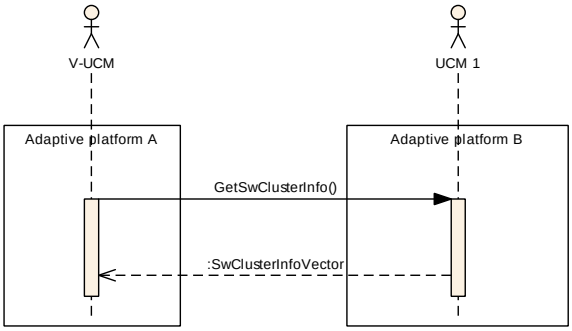

Figure D.1: Collect information of Software Clusters present in vehicle from several AUTOSAR Adaptive Platforms

D.2 Action computation

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

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

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

D.2.1 Pull package from Backend into vehicle

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

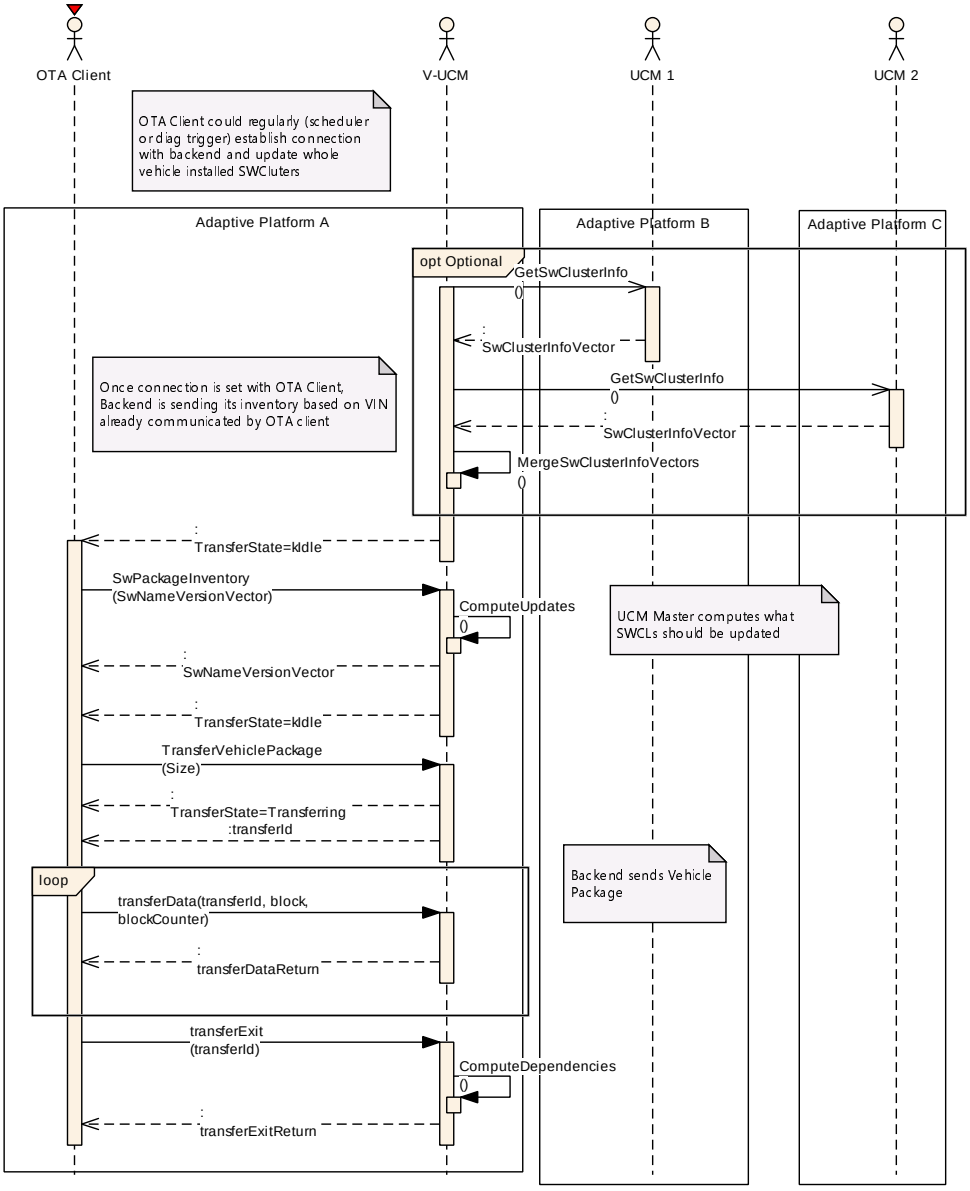

Figure D.2: Pull package from backend

D.2.2 Push package from backend into vehicle

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

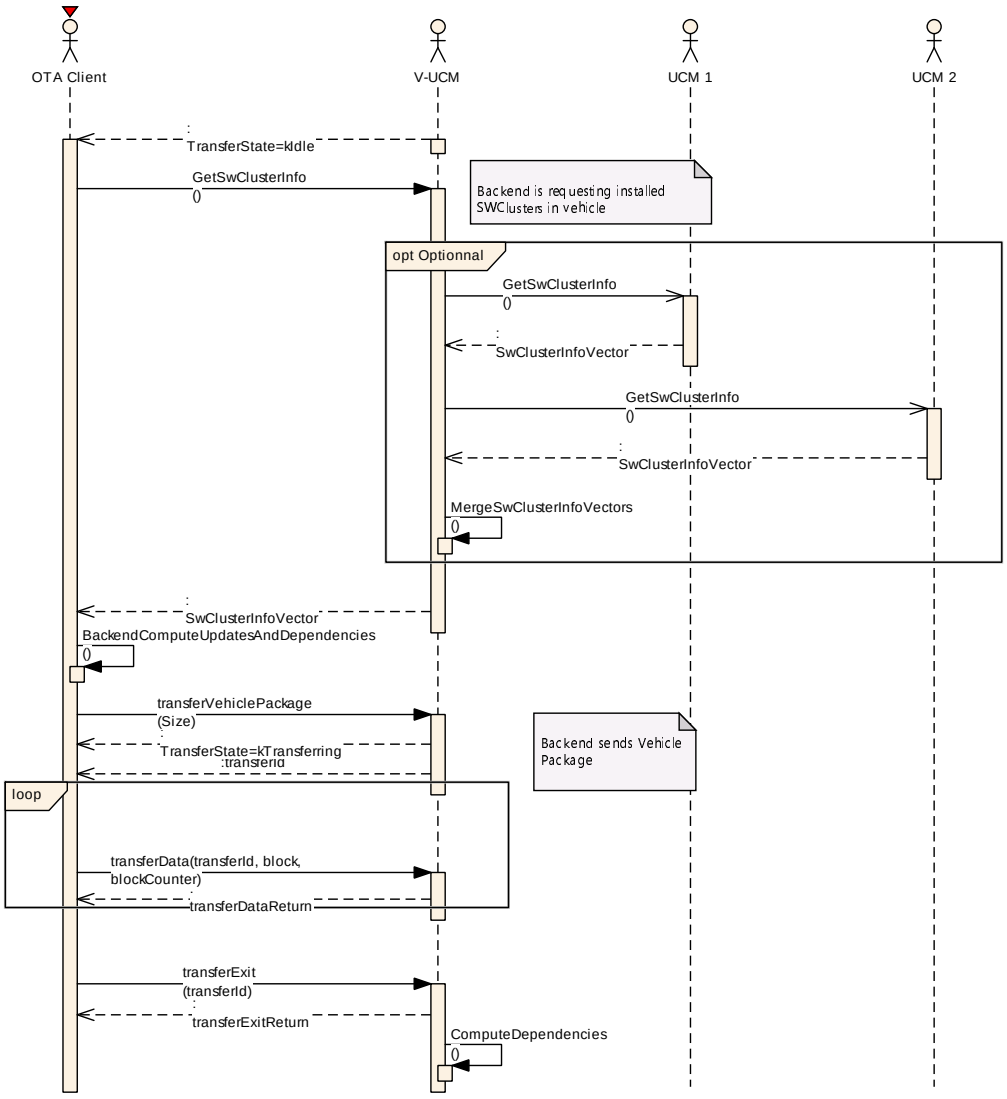

Figure D.3: Push package from backend

D.3 Packages transfer from backend into targeted UCM

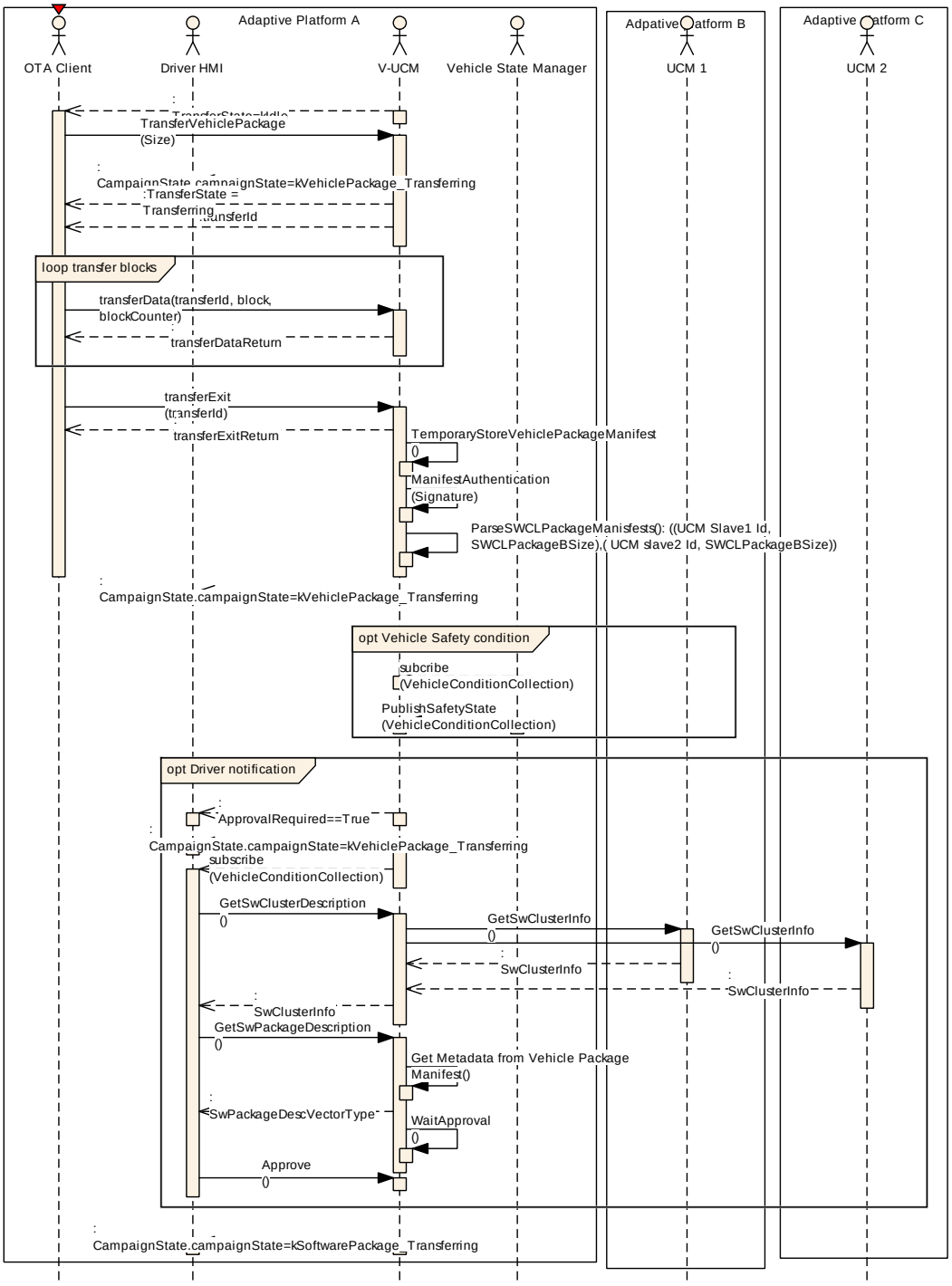

Figure D.4: Stream packages blocks from backend into targeted UCM

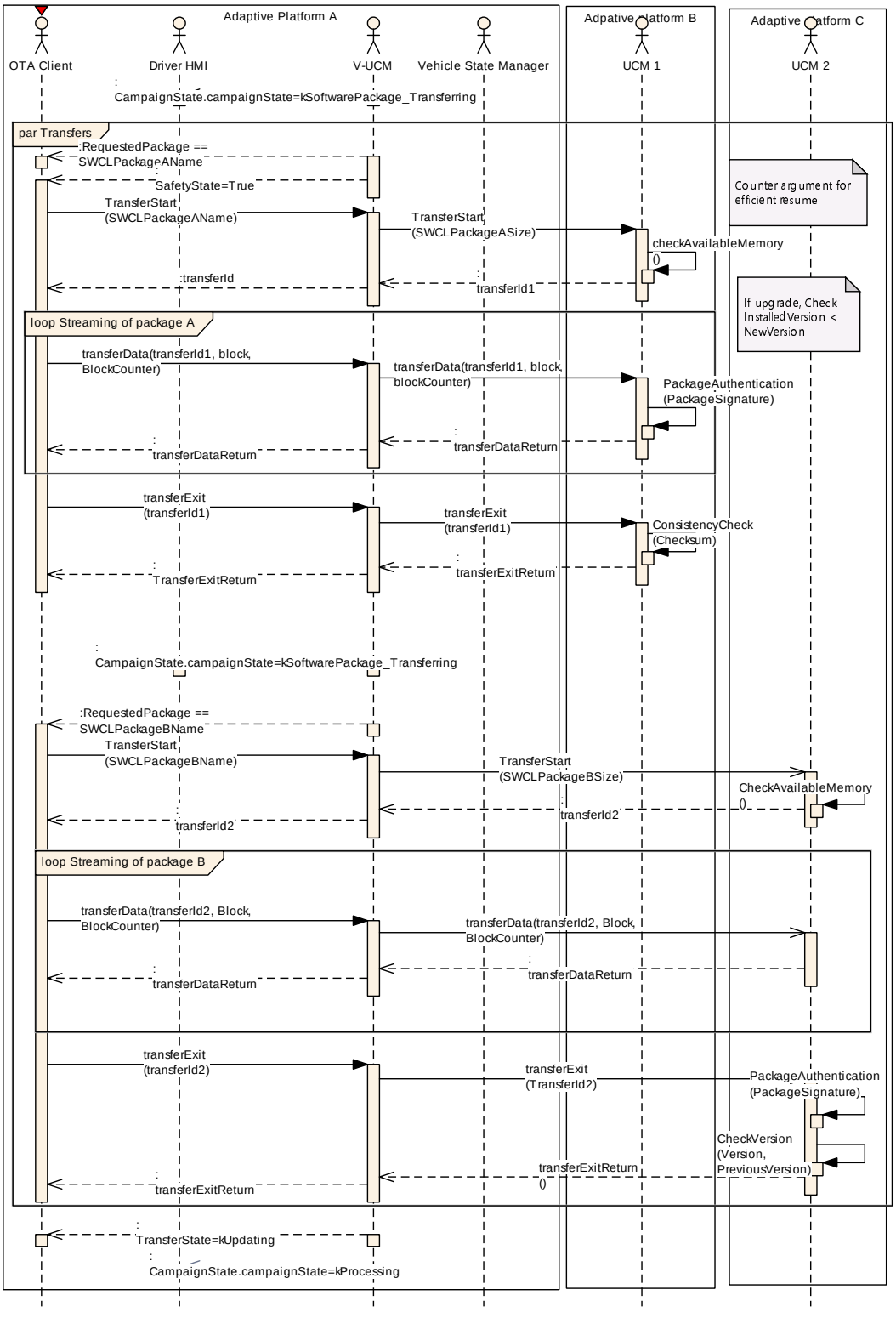

Figure D.5: Stream packages blocks from backend into targeted UCM

D.4 Package processing

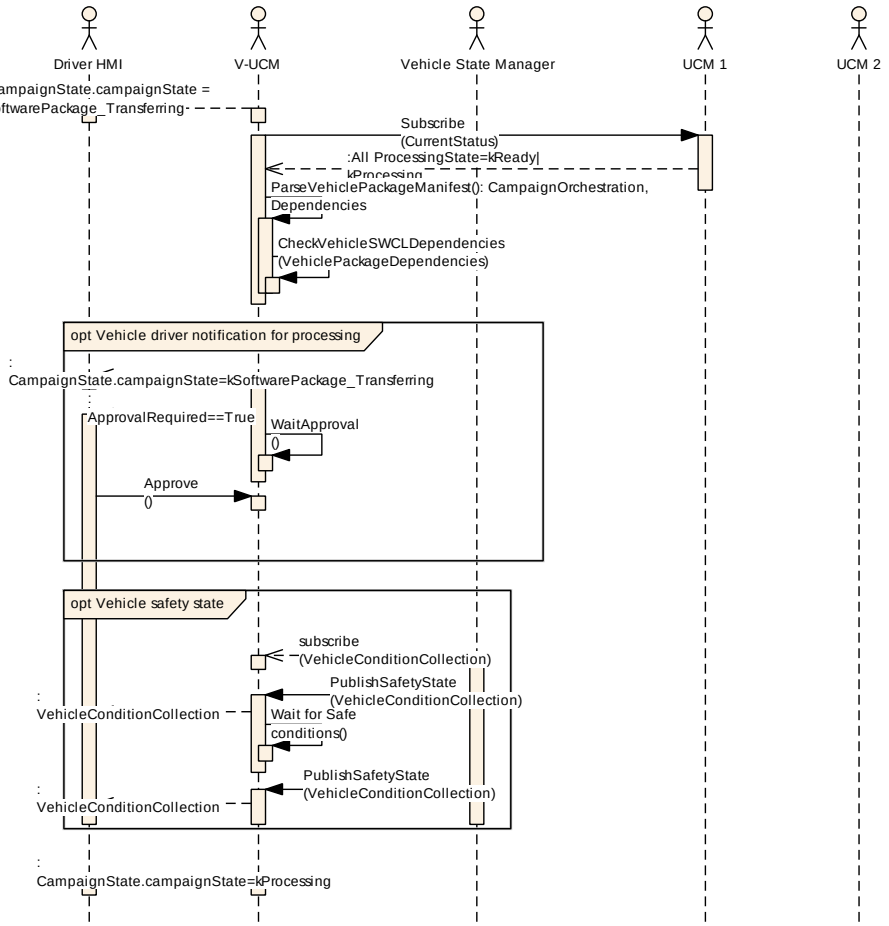

Figure D.6: Packages processing by UCMs

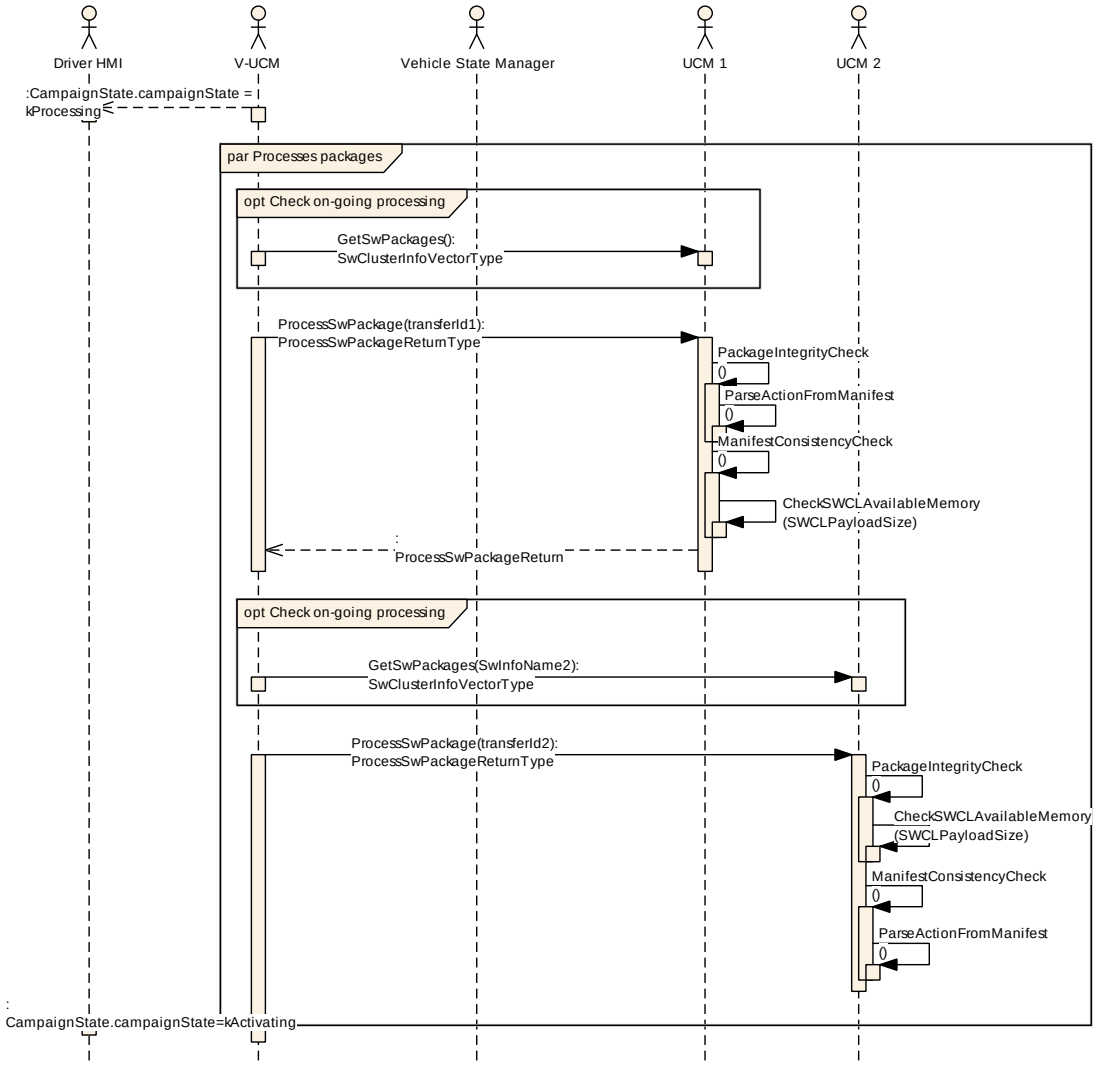

Figure D.7: Packages processing by UCMs

D.5 Package activation

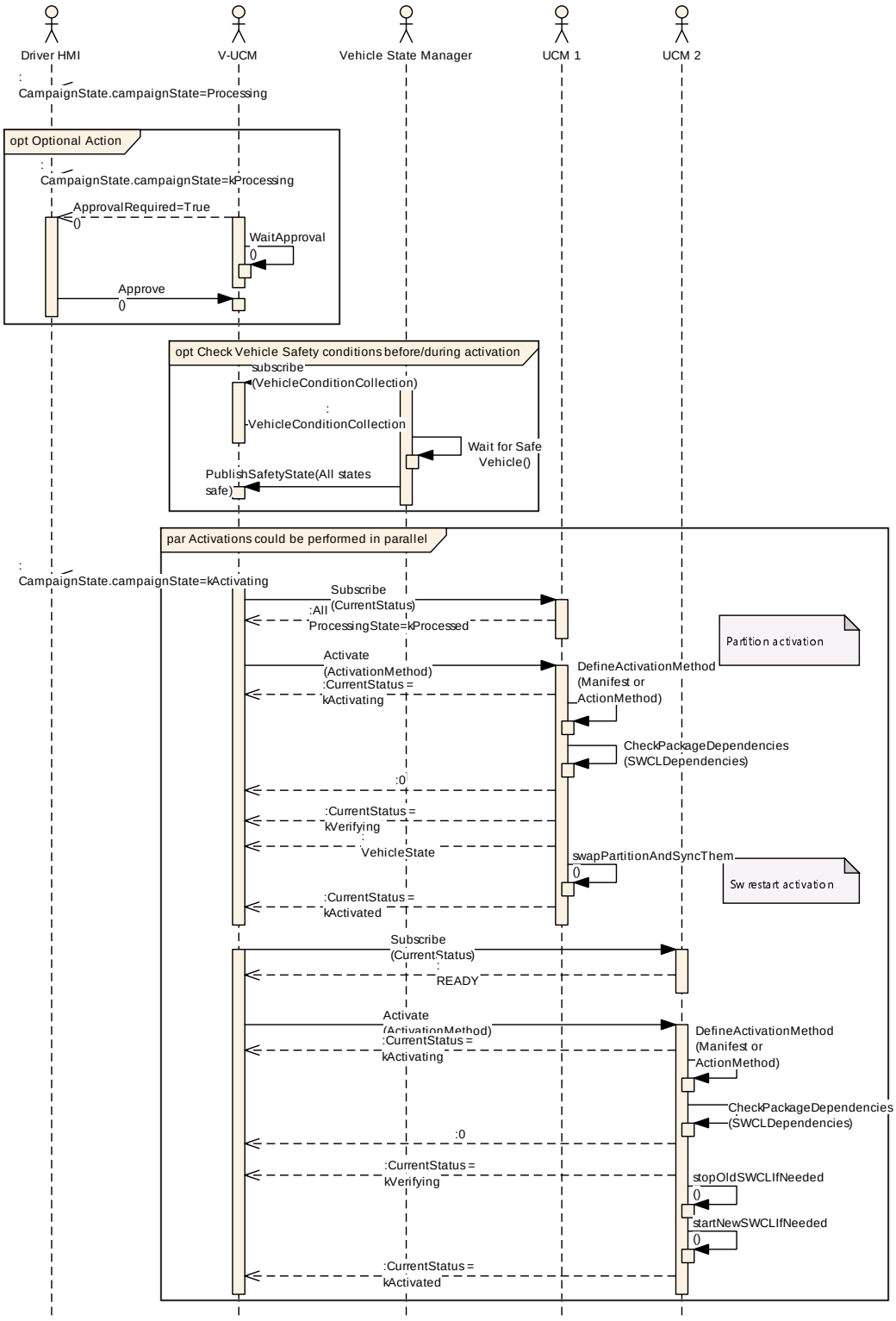

Figure D.8: Packages activation by UCMs

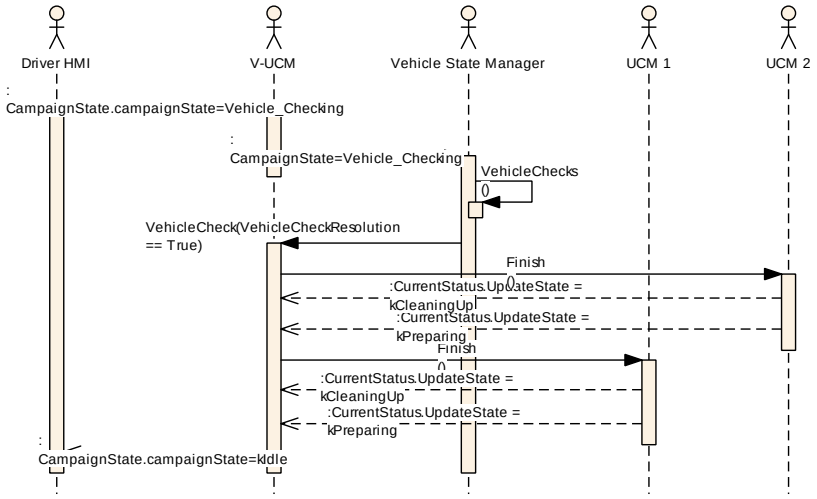

Figure D.9: Packages activation by UCMs

D.6 Package rollback

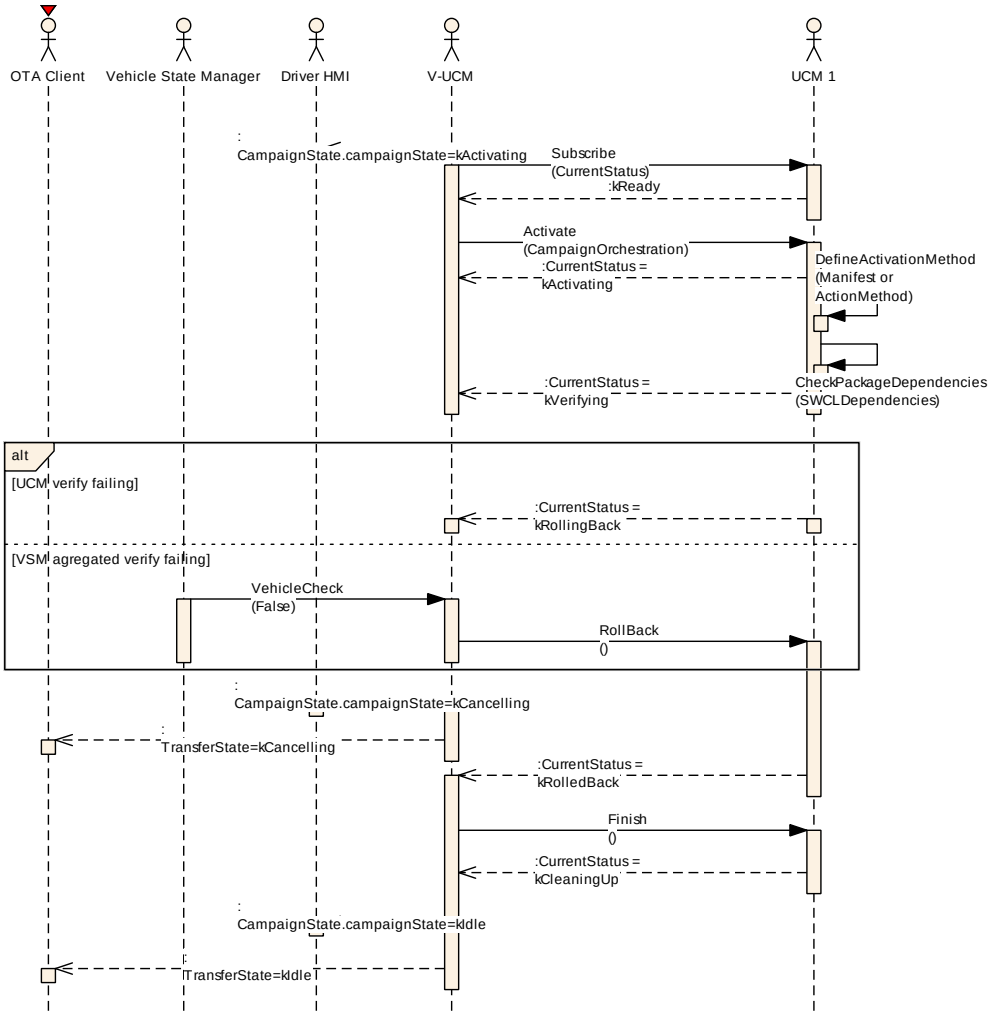

Figure D.10: Packages rollback by UCMs

D.7 Campaign reporting

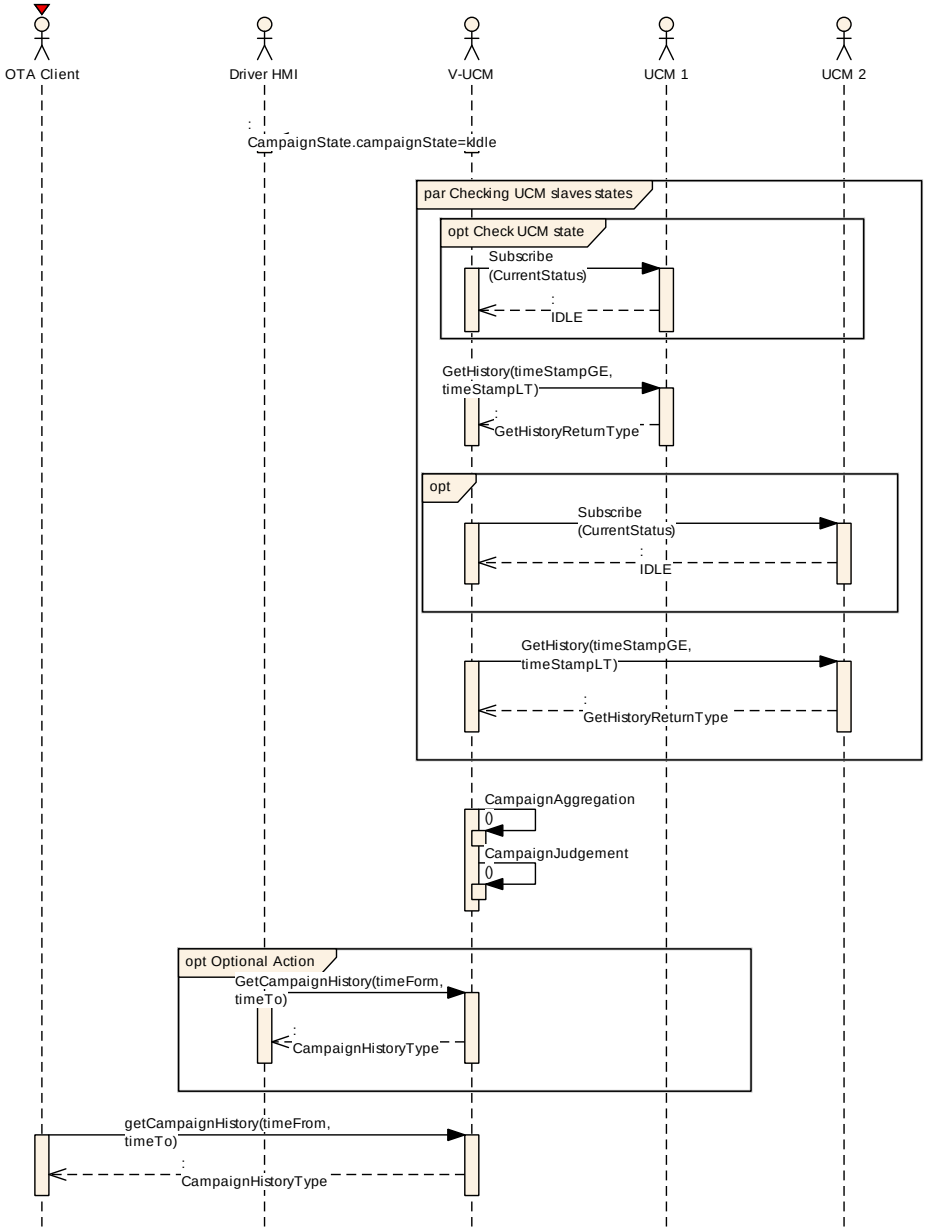

Figure D.11: Campaign reporting to backend

E Security Analysis of Installation and Update

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

E.1 Securing Vehicle Package

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

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

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

E.2 Securing Calls to V-UCM

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

Moreover, some API methods of the V-UCM interfaces returns sensitive information about the platform. This subset (for instance GetSwClusterInfo, GetCampaingHistory,

GetSwPackages, SwPackageInventory) shall be protected against information disclosure and should only be reachable over a channel that provides confidentiality.

E.3 Suppressing Call to V-UCM

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

E.4 Resource Starvation

According to the current specification, the available resources for transferring a Vehicle Package is only checked when TransferStart is called but not reserved. This means, while the transfer is ongoing, the system storage can be exhausted by other processes using the same storage media. In this regard, a solution could be to reserve the necessary resources for the Vehicle Package transfer or processing from the beginning to prevent attacks aiming at such scenarios.

F Demonstrator Examples

Following is pdu api example header file:

```
#ifdef _PDUAPI_EXPORT
#    define EXTERNC extern "C"
#else  // _PDUAPI_EXPORT
#    define EXTERNC /* EXTERNC */
#endif  // _PDUAPI_EXPORT

typedef uint8_t UNUM8; /* Unsigned numeric 8 bits.*/
typedef int8_t SNUM8; /* Signed numeric 8 bits.*/
typedef uint16_t UNUM16; /* Unsigned numeric 16 bits.*/
typedef int16_t SNUM16; /* Signed numeric 16 bits.*/
typedef uint32_t UNUM32; /* Unsigned numeric 32 bits.*/
typedef int32_t SNUM32; /* Signed numeric 32 bits.*/
typedef char CHAR8; /* ASCII-coded 8-bit character value (ISO 8859-1 (Latin
    1)).*/

constexpr UNUM32 PDU_ID_UNDEF = 0xFFFFFFFE; /* Undefined ID value. Used to
   indicate
                                              an ID value is undefined.*/
constexpr UNUM32 PDU_HANDLE_UNDEF = 0xFFFFFFFF; /* Undefined handle value.
   Used to indicate a Handle value is
                                                  undefined. */

typedef enum E_PDU_IT
{
    PDU_IT_IO_UNUM32 = 0x1000, /* IOCTL UNUM32 item. */
    PDU_IT_IO_PROG_VOLTAGE = 0x1001, /* IOCTL Program Voltage item. */
    PDU_IT_IO_BYTEARRAY = 0x1002, /* IOCTL Byte Array item. */
    PDU_IT_IO_FILTER = 0x1003, /* IOCTL Filter item. */
    PDU_IT_IO_EVENT_QUEUE_PROPERTY = 0x1004, /* IOCTL Event Queue Property
   item. */
    PDU_IT_RSC_STATUS = 0x1100, /* Resource Status item */
    PDU_IT_PARAM = 0x1200, /* ComParam item */
    PDU_IT_RESULT = 0x1300, /* Result item */
    PDU_IT_STATUS = 0x1301, /* Status notification item */
    PDU_IT_ERROR = 0x1302, /* Error notification item */
    PDU_IT_INFO = 0x1303, /* Information notification item */
    PDU_IT_RSC_ID = 0x1400, /* Resource ID item */
    PDU_IT_RSC_CONFLICT = 0x1500, /* Resource Conflict Item */
    PDU_IT_MODULE_ID = 0x1600, /* Module item */
    PDU_IT_UNIQUE_RESP_ID_TABLE = 0x1700, /* Unique Response Id Table Item
   */
} T_PDU_IT;

typedef enum E_PDU_COPT
{
    PDU_COPT_STARTCOMM = 0x8001,
    PDU_COPT_STOPCOMM = 0x8002,
    PDU_COPT_UPDATEPARAM = 0x8003,
    PDU_COPT_SENDRECV = 0x8004,
    PDU_COPT_DELAY = 0x8005,
    PDU_COPT_RESTORE_PARAM = 0x8006
} T_PDU_COPT;
```

```
48
49 typedef enum E_PDU_OBJT
50 {
51     PDU_OBJT_PROTOCOL = 0x8021, /* Object type for object PROTOCOL of MDF.
       */
52     PDU_OBJT_BUSTYPE = 0x8022, /* Object type for object BUSTYPE of MDF.*/
53     PDU_OBJT_IO_CTRL = 0x8023, /* Object type for object IO_CTRL of MDF.*/
54     PDU_OBJT_COMPARAM = 0x8024, /* Object type for object COMPARAM of MDF.
       */
55     PDU_OBJT_PINTYPE = 0x8025, /* Object type for object PINTYPE of MDF.*/
56     PDU_OBJT_RESOURCE = 0x8026 /* Object type for object RESOURCE of MDF.*/
57 } T_PDU_OBJT;
58
59 typedef enum E_PDU_STATUS
60 {
61     /* ComPrimitive status */
62     PDU_COPST_IDLE = 0x8010,
63     PDU_COPST_EXECUTING = 0x8011,
64     PDU_COPST_FINISHED = 0x8012,
65     PDU_COPST_CANCELLED = 0x8013,
66     PDU_COPST_WAITING = 0x8014,
67     /* ComLogicalLink status */
68     PDU_CLLST_OFFLINE = 0x8050,
69     PDU_CLLST_ONLINE = 0x8051,
70     PDU_CLLST_COMM_STARTED = 0x8052,
71     /* Module status */
72     PDU_MODST_READY = 0x8060,
73     PDU_MODST_NOT_READY = 0x8061,
74     PDU_MODST_NOT_AVAIL = 0x8062,
75     PDU_MODST_AVAIL = 0x8063,
76 } T_PDU_STATUS;
77
78 typedef enum E_PDU_INFO
79 {
80     PDU_INFO_MODULE_LIST_CHG = 0x8070,
81     PDU_INFO_RSC_LOCK_CHG = 0x8071,
82     PDU_INFO_PHYS_COMPARAM_CHG = 0x8072
83 } T_PDU_INFO;
84
85 typedef enum E_PDU_EVT_DATA
86 {
87     PDU_EVT_DATA_AVAILABLE = 0x0801,
88     PDU_EVT_DATA_LOST = 0x0802
89 } T_PDU_EVT_DATA;
90
91 typedef enum E_PDU_FILTER
92 {
93     PDU_FLT_PASS = 0x00000001,
94     PDU_FLT_BLOCK = 0x00000002,
95     PDU_FLT_PASS_UUDT = 0x00000011,
96     PDU_FLT_BLOCK_UUDT = 0x00000012
97 } T_PDU_FILTER;
98
99 typedef enum E_PDU_QUEUE_MODE
100 {
```

```
101     PDU_QUE_UNLIMITED = 0x00000000, /* In Unlimited Mode, the QueueSize is
    ignored.*/
102     PDU_QUE_LIMITED = 0x00000001, /* When the ComLogicalLink's event queue
    is full, no new items
103                                     are placed on the event queue.*/
104     PDU_QUE_CIRCULAR = 0x00000002 /* When the ComLogicalLink's event queue
    is full (i.e. maximum
105                                     size has been reached), then the
    oldest event item in the
106                                     queue is deleted so that the new event
     item can then be
107                                     placed in the event queue.*/
108 } T_PDU_QUEUE_MODE;
109
110 typedef enum E_PDU_ERROR
111 {
112     PDU_STATUS_NOERROR = 0x00000000, /* No error for the function call */
113     PDU_ERR_FCT_FAILED = 0x00000001, /* Function call failed (generic
    failure) */
114     PDU_ERR_RESERVED_1 = 0x00000010, /* Reserved by ISO 22900-2 */
115     PDU_ERR_COMM_PC_TO_VCI_FAILED = 0x00000011, /* Communication between
    host and
116                                                    MVCI protocol module
    failed */
117     PDU_ERR_PDUAPI_NOT_CONSTRUCTED = 0x00000020, /* The D-PDU API has not
    yet been constructed */
118     PDU_ERR_SHARING_VIOLATION = 0x00000021, /* A PDUDestruct was not called
     before another PDUConstruct */
119     PDU_ERR_RESOURCE_BUSY = 0x00000030, /* the requested resource is
    already in use.*/
120     PDU_ERR_RESOURCE_TABLE_CHANGED = 0x00000031, /* Not used by the D-PDU
    API */
121     PDU_ERR_RESOURCE_ERROR = 0x00000032, /* Not used by the D-PDU API */
122     PDU_ERR_CLL_NOT_CONNECTED = 0x00000040, /* The ComLogicalLink cannot be
     in the PDU_CLLST_OFFLINE state
123                                                to perform the requested
    operation.*/
124     PDU_ERR_CLL_NOT_STARTED = 0x00000041, /* The ComLogicalLink must be in
    the PDU_CLLST_COMM_STARTED
125                                              state to perform the requested
     operation. */
126     PDU_ERR_INVALID_PARAMETERS = 0x00000050, /* One or more of the
    parameters supplied in the function are
127                                                 invalid. */
128     PDU_ERR_INVALID_HANDLE = 0x00000060, /* One or more of the handles
    supplied in
129                                             the function are invalid. */
130     PDU_ERR_VALUE_NOT_SUPPORTED = 0x00000061, /* One of the option values
    in PDUConstruct is invalid. */
131     PDU_ERR_ID_NOT_SUPPORTED = 0x00000062, /* IOCTL command id not
    supported by the implementation of the
132                                               D-PDU API */
133     PDU_ERR_COMPARAM_NOT_SUPPORTED = 0x00000063, /* ComParam id not
    supported by the implementation of the
134                                                    D-PDU API */
```

```
135     PDU_ERR_COMPARAM_LOCKED = 0x00000064, /* Physical ComParam cannot be
    changed because it is locked by
136                                           another ComLogicalLink. */
137     PDU_ERR_TX_QUEUE_FULL = 0x00000070, /* The ComLogicalLink's transmit
    queue is full; the
138                                         ComPrimitive could not be queued
    . */
139     PDU_ERR_EVENT_QUEUE_EMPTY = 0x00000071, /* No more event items are
    available to be read from the
140                                           requested queue. */
141     PDU_ERR_VOLTAGE_NOT_SUPPORTED = 0x00000080, /* The voltage value
    supplied in the IOCTL call is not
142                                             supported by the MVCI
    protocol module. */
143     PDU_ERR_MUX_RSC_NOT_SUPPORTED = 0x00000081, /* The specified pin /
    resource are not supported by the MVCI
144                                             protocol module for the
    IOCTL call. */
145     PDU_ERR_CABLE_UNKNOWN = 0x00000082, /* The cable attached to the MVCI
    protocol
146                                         module is of an unknown type. */
147     PDU_ERR_NO_CABLE_DETECTED = 0x00000083, /* No cable is detected by the
    MVCI protocol module */
148     PDU_ERR_CLL_CONNECTED = 0x00000084, /* The ComLogicalLink is already in
     the
149                                         PDU_CLLST_ONLINE state. */
150     PDU_ERR_TEMPPARAM_NOT_ALLOWED = 0x00000090, /* Physical ComParams
    cannot be changed as a temporary
151                                             ComParam. */
152     PDU_ERR_RSC_LOCKED = 0x000000A0, /* The resource is already locked. */
153     PDU_ERR_RSC_LOCKED_BY_OTHER_CLL = 0x000000A1, /* The ComLogicalLink's
    resource is currently locked by
154                                               another ComLogicalLink
    . */
155     PDU_ERR_RSC_NOT_LOCKED = 0x000000A2, /* The resource is already in the
    unlocked state. */
156     PDU_ERR_MODULE_NOT_CONNECTED = 0x000000A3, /* The module is not in the
    PDU_MODST_READY state. */
157     PDU_ERR_API_SW_OUT_OF_DATE = 0x000000A4, /* The API software is older
    than the
158                                           MVCI protocol module
    Software*/
159     PDU_ERR_MODULE_FW_OUT_OF_DATE = 0x000000A5, /* The MVCI protocol module
     software is older than the API
160                                             software. */
161     PDU_ERR_PIN_NOT_CONNECTED = 0x000000A6 /* The requested Pin is not
    routed by supported cable */
162 } T_PDU_ERROR;
163
164 typedef enum E_PDU_ERR_EVT
165 {
166     PDU_ERR_EVT_NOERROR = 0x00000000, /* No Error. Event type only returned
     on a PDUGetLastError if
167                                       there were no previous errors for
    the requested handle */
```

```
168     PDU_ERR_EVT_FRAME_STRUCT = 0x00000100, /* CLL/CoP Error: The structure
    of the received protocol frame
169                                                is incorrect (e.g. wrong
    frame number, missing FC). */
170     PDU_ERR_EVT_TX_ERROR = 0x00000101, /* CLL/CoP Error: Error encountered
    during
171                                            transmit of a ComPrimitive PDU.
    */
172     PDU_ERR_EVT_TESTER_PRESENT_ERROR = 0x00000102, /* CLL/CoP Error: Error
    encountered in transmitting a Tester
173                                                    Present message or in
     receiving an expected response to a
174                                                    Tester Present
    message. */
175     PDU_ERR_EVT_RX_TIMEOUT = 0x00000103, /* CLL/CoP Error: Receive timer (e
    .g. P2Max) expired with no
176                                              expected responses received
    from the vehicle.*/
177     PDU_ERR_EVT_RX_ERROR = 0x00000104, /* CLL/CoP Error: Error encountered
    in receiving a message
178                                            from the vehicle bus (e.g.
    checksum error). */
179     PDU_ERR_EVT_PROT_ERR = 0x00000105, /* CLL/CoP Error: Protocol error
    encountered during handling
180                                            of a ComPrimitive (e.g. if the
    protocol cannot handle the
181                                            length of a ComPrimitive).*/
182     PDU_ERR_EVT_LOST_COMM_TO_VCI = 0x00000106, /* Module Error:
    Communication to a MVCI protocol module has
183                                                    been lost.*/
184     PDU_ERR_EVT_VCI_HARDWARE_FAULT = 0x00000107, /* Module Error: The MVCI
    protocol module has detected a
185                                                      hardware error.*/
186     PDU_ERR_EVT_INIT_ERROR = 0x00000108, /* CLL/CoP Error: A failure
    occurred during a protocol
187                                              initialization sequence. */
188     PDU_ERR_EVT_RSC_LOCKED = 0x00000109 /* CLL Error: A physical ComParam
    was not set because of a
189                                            physical ComParam lock. */
190 } T_PDU_ERR_EVT;
191
192 typedef enum E_PDU_PC
193 {
194     PDU_PC_UNDEFINED = 0,
195     PDU_PC_TIMING = 1,
196     PDU_PC_INIT = 2,
197     PDU_PC_COM = 3,
198     PDU_PC_ERRHDL = 4,
199     PDU_PC_BUSTYPE = 5,
200     PDU_PC_UNIQUE_ID = 6,
201     PDU_PC_TESTER_PRESENT = 7
202 } T_PDU_PC;
203
204 typedef enum E_PDU_PT
205 {
206     PDU_PT_UNDEFINED = 0x00000000,
```

```
207     PDU_PT_UNUM8 = 0x00000101, /* Unsigned byte */
208     PDU_PT_SNUM8 = 0x00000102, /* Signed byte */
209     PDU_PT_UNUM16 = 0x00000103, /* Unsigned two bytes */
210     PDU_PT_SNUM16 = 0x00000104, /* Signed two bytes */
211     PDU_PT_UNUM32 = 0x00000105, /* Unsigned four bytes */
212     PDU_PT_SNUM32 = 0x00000106, /* Signed four bytes */
213     PDU_PT_BYTEFIELD = 0x00000107, /* Structure contains an array of UNUM8
    bytes with a maximum
214                                       length and actual length fields. See
    ComParam BYTEFIELD
215                                       data type for the definition. */
216     PDU_PT_STRUCTFIELD = 0x00000108, /* Structure contains a void * pointer
     to an array of
217                                         structures. The ComParamStructType
    item determines the type
218                                         of structure to be typecasted onto
    the void * pointer. This
219                                         structure contains a field for
    maximum number of struct
220                                         entries and the actual number of
    struct entries. See
221                                         ComParam STRUCTFIELD data type for
    the definition. */
222     PDU_PT_LONGFIELD = 0x00000109 /* Structure contains an array of UNUM32
    entries with a maximum
223                                      length and actual length fields. See
    ComParam LONGFIELD Data
224                                      Type for the definition. */
225 } T_PDU_PT;
226
227 typedef enum E_PDU_CPST
228 {
229     PDU_CPST_SESSION_TIMING = 0x00000001, /* \see
    PDU_PARAM_STRUCT_SESS_TIMING*/
230     PDU_CPST_ACCESS_TIMING = 0x00000002 /* \see
    PDU_PARAM_STRUCT_ACCESS_TIMING*/
231 } T_PDU_CPST;
232
233 typedef struct
234 {
235     T_PDU_IT ItemType; /* See T_PDU_IT.*/
236 } PDU_ITEM;
237
238 typedef struct
239 {
240     T_PDU_IT ItemType; /* Value= one of the IOCTL constants from T_PDU_IT
    (\ref
241                           E_PDU_IT).*/
242     void* pData; /* Pointer to the specific IOCTL data structure.*/
243 } PDU_DATA_ITEM;
244
245 typedef struct
246 {
247     UNUM32 ProgVoltage_mv; /* Programming voltage [mV].*/
248     UNUM32 PinOnDLC; /* Pin number on Data Link Connector.*/
249 } PDU_IO_PROG_VOLTAGE_DATA;
```

```
250
251 typedef struct
252 {
253     UNUM32 DataSize; /* Number of bytes in the data array.*/
254     UNUM8* pData; /* Pointer to the data array.*/
255 } PDU_IO_BYTEARRAY_DATA;
256
257 typedef struct
258 {
259     T_PDU_FILTER
260     FilterType; /* type of filter being configured \see T_PDU_FILTER.*/
261     UNUM32 FilterNumber; /* Filter Number. Used to replace filters and stop
262                             filters.*/
263     UNUM32 FilterCompareSize; /* Number of bytes used out of each of the
    filter
264                                  messages arrays Range 1-12.*/
265     UNUM8 FilterMaskMessage[12]; /* Mask message to be ANDed to each
    incoming
266                                     message.*/
267     UNUM8 FilterPatternMessage[12]; /* Pattern message to be compared to
    the
268                                        incoming message after the
269                                        FilterMaskMessage has been applied.
    */
270 } PDU_IO_FILTER_DATA;
271
272 typedef struct
273 {
274     UNUM32
275     NumFilterEntries; /* Number of Filter entries in the filter list array.
    */
276     PDU_IO_FILTER_DATA* pFilterData; /* Pointer to an array of filter data.
    */
277 } PDU_IO_FILTER_LIST;
278
279 typedef struct
280 {
281     UNUM32 QueueSize; /* Maximum size of event queue. */
282     T_PDU_QUEUE_MODE QueueMode; /* Queue mode. see T_PDU_QUEUE_MODE event
    queue
283                                    mode type values.*/
284 } PDU_IO_EVENT_QUEUE_PROPERTY_DATA;
285
286 typedef struct
287 {
288     UNUM32 hMod; /* Handle of a MVCI protocol module (IN parameter).*/
289     UNUM32 ResourceId; /*  ID (IN parameter).*/
290     UNUM32 ResourceStatus; /* Resource Information Status (OUT Parameter).
    */
291 } PDU_RSC_STATUS_DATA;
292
293 typedef struct
294 {
295     T_PDU_IT ItemType; /*! value=PDU_IT_RSC_STATUS (IN parameter).*/
296     UNUM32
```

```
297     NumEntries; /* number of entries in pResourceStatusData  (IN Parameter)
    .*/
298     PDU_RSC_STATUS_DATA* pResourceStatusData; /* array to contain resource
    status (IN Parameter).*/
299 } PDU_RSC_STATUS_ITEM;
300
301 typedef struct
302 {
303     T_PDU_IT ItemType; /* value= PDU_IT_PARAM.*/
304     UNUM32 ComParamId; /* ComParam Id.*/
305     T_PDU_PT ComParamDataType; /* Defines the data type of the ComParam \
    ref
306                                   T_PDU_PT; ComParam data type.*/
307     T_PDU_PC ComParamClass; /* ComParam Class type.*/
308     void* pComParamData; /* pointer to ComParam data of type
    ComParamDataType.*/
309 } PDU_PARAM_ITEM;
310
311 typedef struct
312 {
313     UNUM32 ModuleTypeId; /* MVCI protocol moduleTypeId.*/
314     UNUM32 hMod; /* handle of MVCI protocol module assigned by D-PDU.*/
315     CHAR8* pVendorModuleName; /* Vendor specific information string for the
     unique
316                                  module identification.*/
317     CHAR8* pVendorAdditionalInfo; /* Vendor specific additional information
     string.*/
318     T_PDU_STATUS ModuleStatus; /* Status of MVCI protocol module detected
    by D-PDU
319                                   API session.*/
320 } PDU_MODULE_DATA;
321
322 typedef struct
323 {
324     T_PDU_IT ItemType; /* value= PDU_IT_MODULE_ID */
325     UNUM32 NumEntries; /* number of entries written to the pModuleData
    array */
326     PDU_MODULE_DATA* pModuleData; /* pointer to array containing module
    types and
327                                      module handles */
328 } PDU_MODULE_ITEM;
329
330 typedef struct
331 {
332     UNUM32 hMod; /* Module handle*/
333     UNUM32 NumIds; /* number of resources that match PDU_RSC_DATA */
334     UNUM32* pResourceIdArray; /* pointer to a list of resource ids*/
335 } PDU_RSC_ID_ITEM_DATA;
336
337 typedef struct
338 {
339     T_PDU_IT ItemType; /* value = PDU_IT_RSC_ID (IN parameter)*/
340     UNUM32 NumModules; /* number of entries in pResourceIdDataArray. */
341     PDU_RSC_ID_ITEM_DATA
342     *pResourceIdDataArray; /* pointer to an array of resource Id Item Data
    */
```

```
} PDU_RSC_ID_ITEM;

typedef struct
{
    UNUM32 DLCPinNumber; /* Pin number on DLC */
    UNUM32 DLCPinTypeId; /* Pin ID */
} PDU_PIN_DATA;

typedef struct
{
    UNUM32 BusTypeId; /* Bus Type Id (IN parameter) */
    UNUM32 ProtocolId; /* Protocol Id (IN parameter) */
    UNUM32 NumPinData; /* Number of items in the following array */
    PDU_PIN_DATA* pDLCPinData; /* Pointer to array of PDU_PIN_DATA
    structures*/
} PDU_RSC_DATA;

typedef struct
{
    UNUM32 hMod; /* Handle of the MVCI protocol module with conflict.*/
    UNUM32 ResourceId; /* Conflicting Resource ID.*/
} PDU_RSC_CONFLICT_DATA;

typedef struct
{
    T_PDU_IT ItemType; /* value= PDU_IT_RSC_CONFLICT.*/
    UNUM32 NumEntries; /* Number of entries written to pRscConflictData.*/
    PDU_RSC_CONFLICT_DATA
    *pRscConflictData; /* Pointer to array of PDU_RSC_CONFLICT_DATA.*/
} PDU_RSC_CONFLICT_ITEM;

typedef struct
{
    UNUM32 UniqueRespIdentifier; /* filled out by application */
    UNUM32 NumParamItems; /* number of ComParams for the Unique Identifier
    */
    PDU_PARAM_ITEM
    *pParams; /* pointer to array of ComParam items to uniquely define a
    ECU
                response. The list is protocol specific */
} PDU_ECU_UNIQUE_RESP_DATA;

typedef struct
{
    T_PDU_IT ItemType; /* Value= PDU_IT_UNIQUE_RESP_ID_TABLE */
    UNUM32 NumEntries; /* Number of entries in the table.*/
    PDU_ECU_UNIQUE_RESP_DATA* pUniqueData; /* Pointer to array of table
    entries for each ECU response.*/
} PDU_UNIQUE_RESP_ID_TABLE_ITEM;

typedef struct
{
    T_PDU_IT ItemType; /* Value= PDU_IT_RESULT or PDU_IT_STATUS or
    PDU_IT_ERROR or
                        PDU_IT_INFO.*/
    UNUM32
```

```
394     hCoP; /* If item is from a ComPrimitive then the hCop contains the
        valid
395              ComPrimitive handle, else it contains PDU_HANDLE_UNDEF.*/
396     void* pCoPTag; /* ComPrimitive Tag. Should be ignored if
397                       hCop=PDU_HANDLE_UNDEF.*/
398     UNUM32 Timestamp; /* Timestamp in microseconds.*/
399     void* pData; /* Points to the data for the specified Item Type.*/
400 } PDU_EVENT_ITEM;
401
402 typedef T_PDU_STATUS PDU_STATUS_DATA;
403
404 typedef struct
405 {
406     T_PDU_INFO InfoCode; /* Information code.*/
407     UNUM32 ExtraInfoData; /* Optional additional information.*/
408 } PDU_INFO_DATA;
409
410 typedef struct
411 {
412     T_PDU_ERR_EVT ErrorCodeId; /* Error code, binary information.*/
413     UNUM32 ExtraErrorInfoId; /* Optional additional error information, text
414                                 translation via MDF file. Binary
        Information, 0
415                                 indicates no additional error information.
        */
416 } PDU_ERROR_DATA;
417
418 typedef struct
419 {
420     UNUM32 NumFlagBytes; /* Number of bytes in pFlagData array.*/
421     UNUM8* pFlagData; /* Pointer to flag bytes used for TxFlag, RxFlag, and
422                          CllCreateFlag.*/
423 } PDU_FLAG_DATA;
424
425 typedef struct
426 {
427     UNUM32 NumHeaderBytes; /* Number of header bytes contained in
        pHeaderBytes
428                               array. */
429     UNUM32 NumFooterBytes; /* Number of footer bytes contained in
        pFooterBytes
430                               array. */
431     UNUM8* pHeaderBytes; /* Reference pointer to Response PDU Header bytes,
         NULL
432                             if NumHeaderBytes = 0.*/
433     UNUM8* pFooterBytes; /* Reference pointer to Response PDU Footer bytes,
         NULL
434                             if NumFooterBytes = 0.*/
435 } PDU_EXTRA_INFO;
436
437 typedef struct
438 {
439     PDU_FLAG_DATA RxFlag;
440     UNUM32 UniqueRespIdentifier;
441     UNUM32 AcceptanceId;
442     PDU_FLAG_DATA TimestampFlags;
```

```
443     UNUM32 TxMsgDoneTimestamp;
444     UNUM32 StartMsgTimestamp;
445     PDU_EXTRA_INFO* pExtraInfo;
446     UNUM32 NumDataBytes;
447     UNUM8* pDataBytes;
448 } PDU_RESULT_DATA;
449
450 typedef struct
451 {
452     UNUM32 MVCI_Part1StandardVersion;
453     UNUM32 MVCI_Part2StandardVersion;
454     UNUM32 HwSerialNumber;
455     CHAR8 HwName[64];
456     UNUM32 HwVersion;
457     UNUM32 HwDate;
458     UNUM32 HwInterface;
459     CHAR8 FwName[64];
460     UNUM32 FwVersion;
461     UNUM32 FwDate;
462     CHAR8 VendorName[64];
463     CHAR8 PDUApiSwName[64];
464     UNUM32 PDUApiSwVersion;
465     UNUM32 PDUApiSwDate;
466 } PDU_VERSION_DATA;
467
468 typedef struct
469 {
470     UNUM32 ResponseType; /* 0 = positive response; 1 = negative response.*/
471     UNUM32 AcceptanceId; /* ID assigned by application to be returned in
472                             PDU_RESULT_DATA, which indicates which expected
473                             response matched.*/
474     UNUM32 NumMaskPatternBytes; /* number of bytes in the Mask Data and
    Pattern
475                                    Data.*/
476     UNUM8* pMaskData; /* Pointer to Mask Data. Bits set to a '1' are care
    bits,
477                          '0' are don't care bits.*/
478     UNUM8* pPatternData; /* Pointer to Pattern Data. Bytes to compare after
     the
479                             mask is applied.*/
480     UNUM32 NumUniqueRespIds; /* Number of items in the following array of
    unique
481                                 response identifiers.*/
482     UNUM32* pUniqueRespIds; /* Array containing unique response identifiers
    . Only
483                                responses with a unique response identifier
    found
484                                in this array are considered, when trying to
     match
485                                them to this expected response. */
486 } PDU_EXP_RESP_DATA;
487
488 typedef struct
489 {
490     UNUM32 Time; /* Cycle time in ms for cyclic send operation or delay
    time for
```

```
491                     PDU_COPT_DELAY ComPrimitive.*/
492     SNUM32 NumSendCycles; /* Number of send cycles to be performed.*/
493     SNUM32 NumReceiveCycles; /* Number of receive cycles to be performed.*/
494     UNUM32 TempParamUpdate; /* Temporary ComParam settings for the
    ComPrimitive.*/
495     PDU_FLAG_DATA TxFlag; /* Transmit Flag used to indicate protocol
    specific
496                              elements for the ComPrimitive's execution..*/
497     UNUM32 NumPossibleExpectedResponses; /* number of entries in \ref
498                                             pExpectedResponseArray.*/
499     PDU_EXP_RESP_DATA
500     *pExpectedResponseArray; /* pointer to an array of expected responses.
    */
501 } PDU_COP_CTRL_DATA;
502
503 typedef struct
504 {
505     UNUM32 ParamMaxLen; /* Contains the maximum number of UNUM8 bytes the
    ComParam
506                            can contain in \ref pDataArray.*/
507     UNUM32 ParamActLen; /* Contains the actual number of UNUM8 bytes in
508                            pDataArray..*/
509     UNUM8* pDataArray; /* Pointer to an array of \ref UNUM8 values.*/
510 } PDU_PARAM_BYTEFIELD_DATA;
511
512 typedef struct
513 {
514     T_PDU_CPST ComParamStructType; /* Type of ComParam Structure being used
    .*/
515     UNUM32 ParamMaxEntries; /* Contains the maximum number of struct
    entries in
516                                \ref pStructArray.*/
517     UNUM32 ParamActEntries; /* Contains the maximum number of struct
    entries the
518                                ComParam can contain in \ref pStructArray.*/
519     void* pStructArray; /* Pointer to an array of structs (typecasted to
    the \ref
520                            ComParamStructType).*/
521 } PDU_PARAM_STRUCTFIELD_DATA;
522
523 typedef struct
524 {
525     UNUM16
526     session; /* Session Number, for the diagnostic session of ISO 15765-3.
    */
527     UNUM8 P2Max_high; /* Default P2Can_Server_max timing (1 ms resolution)
528                          supported by the server for the activated
    diagnostic
529                          session. Used for ComParam CP_P2Max.*/
530     UNUM8 P2Max_low; /* Timing used for ComParam CP_P2Min (1 ms resolution)
    .*/
531     UNUM8 P2Star_high; /* Enhanced (NRC 78 hex) P2Can_Server_max (10 ms
532                           resolution) supported by the server for the
    activated
533                           diagnostic session. Used for ComParam CP_P2Star.
    */
```

```
534     UNUM8
535     P2Star_low; /* Timing (10 ms resolution) used for internal ECU use only
        .*/
536 } PDU_PARAM_STRUCT_SESS_TIMING;
537
538 typedef struct
539 {
540     UNUM8 P2Min; /* Minimum time (0,5 ms resolution) between tester request
         and
541                     ECU response(s). Used for ComParam CP_P2Min.*/
542     UNUM8 P2Max; /* Maximum time between tester request and ECU response(s)
        .*/
543     UNUM8 P3Min; /* Minimum time between end of ECU responses and start of
        new
544                     tester request.*/
545     UNUM8 P3Max; /* Maximum time between ECU responses and start of new
        tester
546                     request.*/
547     UNUM8 P4Min; /* Minimum inter byte time for tester request.*/
548     UNUM8 TimingSet; /* Set number allowing multiple sets of timing
        parameters.*/
549 } PDU_PARAM_STRUCT_ACCESS_TIMING;
550
551 typedef struct
552 {
553     UNUM32 ParamMaxLen; /* Contains the maximum number of UNUM32 entries
        the
554                            ComParam can contain in \ref pDataArray.*/
555     UNUM32 ParamActLen; /* Contains the current number of UNUM32 entries
        the
556                            ComParam can contain in \ref pDataArray.*/
557     UNUM32* pDataArray; /* Pointer to an array of UNUM32 values.*/
558 } PDU_PARAM_LONGFIELD_DATA;
559
560 typedef void (*CALLBACKFNC)(T_PDU_EVT_DATA eventType, UNUM32 hMod, UNUM32
        hCll, void* pCllTag, void* pAPITag);
561
562 EXTERNC T_PDU_ERROR PDUConstruct(CHAR8* pszOption, void* pAPITag);
563 EXTERNC T_PDU_ERROR PDUDestruct();
564 EXTERNC T_PDU_ERROR PDUModuleConnect(UNUM32 hMod);
565 EXTERNC T_PDU_ERROR PDUModuleDisconnect(UNUM32 hMod);
566 EXTERNC T_PDU_ERROR PDUGetModuleIds(PDU_MODULE_ITEM** pModuleIdList);
567 EXTERNC T_PDU_ERROR PDUGetResourceIds(UNUM32 hMod, PDU_RSC_DATA*
        pResourceIdData, PDU_RSC_ID_ITEM** pResourceIdList);
568 EXTERNC T_PDU_ERROR PDUGetResourceStatus(PDU_RSC_STATUS_ITEM*
        pResourceStatus);
569 EXTERNC T_PDU_ERROR PDUCreateComLogicalLink(UNUM32 hMod,
570     PDU_RSC_DATA* pRscData,
571     UNUM32 uiResourceId,
572     void* pCllTag,
573     UNUM32* phCll,
574     PDU_FLAG_DATA* pCllCreateFlag);
575 EXTERNC T_PDU_ERROR PDUDestroyComLogicalLink(UNUM32 hMod, UNUM32 hCll);
576 EXTERNC T_PDU_ERROR PDURegisterEventCallback(UNUM32 hMod, UNUM32 hCll,
        CALLBACKFNC fnCB);
```

```
577 EXTERNC T_PDU_ERROR PDUGetComParam(UNUM32 hMod, UNUM32 hCll, UNUM32
      uiParamId, PDU_PARAM_ITEM** pParamItem);
578 EXTERNC T_PDU_ERROR PDUSetComParam(UNUM32 hMod, UNUM32 hCll, PDU_PARAM_ITEM
      * pParamItem);
579 EXTERNC T_PDU_ERROR PDUConnect(UNUM32 hMod, UNUM32 hCll);
580 EXTERNC T_PDU_ERROR PDUDisconnect(UNUM32 hMod, UNUM32 hCll);
581 EXTERNC T_PDU_ERROR PDUStartComPrimitive(UNUM32 hMod,
582     UNUM32 hCll,
583     T_PDU_COPT uiCoPType,
584     UNUM32 uiCoPDataSize,
585     UNUM8* pCoPData,
586     PDU_COP_CTRL_DATA* pCopCtrlData,
587     void* pCoPTag,
588     UNUM32* phCoP);
589 EXTERNC T_PDU_ERROR PDUCancelComPrimitive(UNUM32 hMod, UNUM32 hCll, UNUM32
      hCoP);
590 EXTERNC T_PDU_ERROR PDUGetEventItem(UNUM32 hMod, UNUM32 hCll,
      PDU_EVENT_ITEM** pEventItem);
591 EXTERNC T_PDU_ERROR PDUDestroyItem(PDU_ITEM* pItem);
592 EXTERNC T_PDU_ERROR PDUGetVersion(UNUM32 hMod, PDU_VERSION_DATA*
      pVersionData);
593 EXTERNC T_PDU_ERROR PDUGetLastError(UNUM32 hMod,
594     UNUM32 hCll,
595     T_PDU_ERR_EVT* pErrorCode,
596     UNUM32* phCoP,
597     UNUM32* pTimestamp,
598     UNUM32* pExtraErrorInfo);
599 EXTERNC T_PDU_ERROR PDUGetUniqueRespIdTable(UNUM32 hMod,
600     UNUM32 hCll,
601     PDU_UNIQUE_RESP_ID_TABLE_ITEM** pUniqueRespIdTable);
602 EXTERNC T_PDU_ERROR PDUSetUniqueRespIdTable(UNUM32 hMod,
603     UNUM32 hCll,
604     PDU_UNIQUE_RESP_ID_TABLE_ITEM* pUniqueRespIdTable);
605 EXTERNC T_PDU_ERROR
606 PDUIoCtl(UNUM32 hMod, UNUM32 hCll, UNUM32 uiIoCtlCommandId, PDU_DATA_ITEM*
      pInputData, PDU_DATA_ITEM** pOutputData);
607 EXTERNC T_PDU_ERROR PDUGetObjectId(UNUM32 uiPDUObjectType, CHAR8*
      pszShortName, UNUM32* pPDUObjectId);
608 EXTERNC T_PDU_ERROR PDUGetTimestamp(UNUM32 hMod, UNUM32* puiTimestamp);
609 EXTERNC T_PDU_ERROR PDUGetStatus(UNUM32 hMod,
610     UNUM32 hCll,
611     UNUM32 hCoP,
612     T_PDU_STATUS* puiStatusCode,
613     UNUM32* puiTimestamp,
614     UNUM32* puiExtraInfo);
615 EXTERNC T_PDU_ERROR PDUGetConflictingResources(UNUM32 uiResourceId,
616     PDU_MODULE_ITEM* pModuleList,
617     PDU_RSC_CONFLICT_ITEM** pConflictList);
618 EXTERNC T_PDU_ERROR PDULockResource(UNUM32 hMod, UNUM32 hCll, UNUM32
      uiLockMask);
619 EXTERNC T_PDU_ERROR PDUUnlockResource(UNUM32 hMod, UNUM32 hCll, UNUM32
      uiLockMask);
620
621
622 using BYTEFIELD_PTR = std::shared_ptr<const PDU_PARAM_BYTEFIELD_DATA>;
623 using STRUCTFIELD_PTR = std::shared_ptr<const PDU_PARAM_STRUCTFIELD_DATA>;
```

```
624 using LONGFIELD_PTR = std::shared_ptr<const PDU_PARAM_LONGFIELD_DATA>;
625
626 using ComParamData
627     = std::variant<UNUM8, SNUM8, UNUM16, SNUM16, UNUM32, SNUM32,
    BYTEFIELD_PTR, STRUCTFIELD_PTR, LONGFIELD_PTR>;
628
629 /// @brief ComParamDictionary
630 class ComParamDictionary
631 {
632 public:
633     virtual ~ComParamDictionary() = default;
634
635     ///
636     /// @brief Get the ComParam based on its name.
637     ///
638     /// @param name ComParam name (e.g, CP_P2Max)
639     /// @return ComParam value
640     ///
641     virtual ComParamData getComParamByName(ara::core::StringView name)
    const = 0;
642 };
643
644 /// @brief Communication Parameter Buffer
645 class ComParamBuffer : public ComParamDictionary
646 {
647 public:
648     ///
649     /// @brief Set ComParam based on the given PDU_PARAM_ITEM structure.
650     ///
651     /// @param param reference to PDU_PARAM_ITEM structure to be set
652     /// @return true if the ComParam is successfully set.
653     /// @return false if the ComParam Id is invalid or the paramType is
    different
654     ///         than UNUM32
655     ///
656     virtual bool setComParam(const PDU_PARAM_ITEM& param) = 0;
657
658     ///
659     /// @brief Get a ComParam based on its Id.
660     ///
661     /// @param paramId ComParam Id
662     /// @return pointer to the filled PDU_PARAM_ITEM structure or Null
    pointer if
663     ///         the ParamId is invalid.
664     ///
665     virtual std::shared_ptr<PDU_PARAM_ITEM> getComParam(UNUM32 paramId)
    const = 0;
666 };
667
668 /// @brief Communication Parameter Buffer implementation
669 class ComParamBufferImpl final : public ComParamBuffer
670 {
671 public:
672     /// @copybrief ComParamBuffer::setComParam()
673     /// @copydetails ComParamBuffer::setComParam()
674     bool setComParam(const PDU_PARAM_ITEM& param) override;
```

```
675
676     /// @copybrief ComParamBuffer::getComParam()
677     /// @copydetails ComParamBuffer::setComParam()
678     std::shared_ptr<PDU_PARAM_ITEM> getComParam(UNUM32 paramId) const
    override;
679
680     /// @copybrief ComParamDictionary::getComParamByName
681     /// @copydetails ComParamDictionary::getComParamByName
682     ComParamData getComParamByName(ara::core::StringView name) const
    override;
683
684     ///
685     /// @brief Add a ComParam to the ComParams list (@params_)
686     ///
687     /// @param id ComParam Id
688     /// @param name ComParam name
689     /// @param paramType ComParam Type
690     /// @param paramClass ComParam Class
691     /// @param data ComParam value
692     /// @return true if the ComParam is successfully added.
693     ///         false if the ComParam Id/name is invalid or the paramType
    is
694     ///         different than UNUM32
695     ///
696     bool addComParam(UNUM32 id,
697         const ara::core::String& name,
698         T_PDU_PT paramType,
699         T_PDU_PC paramClass,
700         const void* data);
701
702     ///
703     /// @brief Get the number of the ComParams.
704     ///
705     /// @return number of the ComParams
706     std::size_t size() const;
707
708     ///
709     /// @brief Add a ComParam to the URID table (@paramBuffer_)
710     ///
711     /// @param id ComParam Id
712     /// @param name ComParam name
713     /// @param paramType ComParam Type
714     /// @param paramClass ComParam Class
715     /// @param data ComParam value
716     /// @return true if the ComParam is successfully added.
717     ///         false if the parameter class is different than
    PDU_PC_UNIQUE_ID
718     ///
719     ara::core::String getComParamNameById(UNUM32 paramId) const;
720
721     /// @brief ComParam Key (Comparam Id and name)
722     struct ParamKey
723     {
724         UNUM32 id;
725         ara::core::String name;
726     };
```

```
727
728     /// @brief ComParam Type and Class
729     struct ComParameter
730     {
731         const T_PDU_PT paramType;
732         const T_PDU_PC paramClass;
733         ComParamData data;
734     };
735
736     /// @brief ComParamKey compare
737     struct cmpParamKey
738     {
739         bool operator()(const ParamKey& a, const ParamKey& b) const
740         {
741             if (a.id == b.id || a.name == b.name)
742                 return false;
743             return (a.id < b.id) || (a.name < b.name);
744         }
745     };
746
747     using const_iterator = ara::core::Map<ParamKey, ComParameter,
    cmpParamKey>::const_iterator;
748     const_iterator cbegin() const
749     {
750         return params_.cbegin();
751     }
752
753     const_iterator cend() const
754     {
755         return params_.cend();
756     }
757
758 private:
759     /// @brief List of ComParams
760     ara::core::Map<ParamKey, ComParameter, cmpParamKey> params_;
761 };
762
763 /// @brief URID Table Entry
764 class URIDTableEntry : public ComParamDictionary
765 {
766 public:
767     /// @brief Unique Response Identifier
768     virtual UNUM32 uniqueRespIdentifier() const = 0;
769 };
770
771 /// @brief Unique Response Identifier Table
772 class URIDTable
773 {
774 public:
775     virtual ~URIDTable() = default;
776
777     ///
778     /// @brief Check if there is an entry in the URID table.
779     ///
780     /// @return true if there is no entry in the URID table.
781     /// @return false if there is one or more entries in the URID table.
```

```
782     ///
783     virtual bool empty() const = 0;
784     virtual const URIDTableEntry& operator[](std::size_t pos) const = 0;
785
786     ///
787     /// @brief Create the URID Table (PDU_UNIQUE_RESP_ID_TABLE_ITEM
        structure).
788     ///
789     /// @return pointer to the PDU_UNIQUE_RESP_ID_TABLE_ITEM structure.
790     ///
791     virtual std::shared_ptr<PDU_UNIQUE_RESP_ID_TABLE_ITEM> getTableItem()
        const = 0;
792
793     ///
794     /// @brief Set an URID Table entry
795     ///
796     /// @return true if the URID Table entry is successfully set.
797     /// @return false if Wrong number of com parameters is provided or a
        com
798     ///         parameter not supported by the protocol
799     ///
800     virtual bool set(const PDU_UNIQUE_RESP_ID_TABLE_ITEM&) = 0;
801 };
802 /// @brief Unique Response ID Table Entry implementation
803 class URIDTableEntryImpl final : public URIDTableEntry
804 {
805 public:
806     explicit URIDTableEntryImpl(UNUM32 uniqueRespIdentifier = PDU_ID_UNDEF)
        ;
807
808     /// @brief Unique Response Identifier
809     UNUM32 uniqueRespIdentifier() const override;
810
811     /// @copybrief ComParamDictionary::getComParamByName
812     /// @copydetails ComParamDictionary::getComParamByName
813     ComParamData getComParamByName(ara::core::StringView name) const
        override;
814
815     ///
816     /// @brief Get a ComParam ,based on its Id, from the URID table.
817     ///
818     /// @param paramId ComParam Id
819     /// @return pointer to the filled PDU_PARAM_ITEM structure or Null
        pointer if
820     ///         the ParamId is invalid.
821     ///
822     std::shared_ptr<PDU_PARAM_ITEM> getComParam(UNUM32 paramId) const;
823
824     ///
825     /// @brief Add a ComParam to the URID table (@paramBuffer_)
826     ///
827     /// @param id ComParam Id
828     /// @param name ComParam name
829     /// @param paramType ComParam Type
830     /// @param paramClass ComParam Class
831     /// @param data ComParam value
```

```
832     /// @return true if the ComParam is successfully added.
833     ///         false if the parameter class is different than
        PDU_PC_UNIQUE_ID
834     ///
835     bool addComParam(UNUM32 id,
836         const ara::core::String& name,
837         T_PDU_PT paramType,
838         T_PDU_PC paramClass,
839         const void* data);
840
841     std::size_t size() const;
842     ara::core::String getComParamNameById(UNUM32 paramId) const;
843
844     using const_iterator = ComParamBufferImpl::const_iterator;
845     const_iterator cbegin() const
846     {
847         return paramBuffer_->cbegin();
848     }
849
850     const_iterator cend() const
851     {
852         return paramBuffer_->cend();
853     }
854
855 private:
856     /// @brief Unique Response Identifier
857     UNUM32 uniqueRespIdentifier_;
858
859     /// @brief URID table
860     std::unique_ptr<ComParamBufferImpl> paramBuffer_;
861 };
862
863 /// @brief URID Table implementation
864 class URIDTableImpl final : public URIDTable
865 {
866 public:
867     /// @copybrief URIDTable::empty()
868     /// @copydetails URIDTable::empty()
869     bool empty() const override;
870     const URIDTableEntry& operator[](std::size_t pos) const override;
871
872     /// @copybrief URIDTable::getTableItem()
873     /// @copydetails URIDTable::getTableItem()
874     std::shared_ptr<PDU_UNIQUE_RESP_ID_TABLE_ITEM> getTableItem() const
        override;
875     bool set(const PDU_UNIQUE_RESP_ID_TABLE_ITEM& tableItem) override;
876
877     /// @copybrief URIDTable::URIDTableImpl()
878     /// @copydetails URIDTable::URIDTableImpl()
879     explicit URIDTableImpl(URIDTableEntryImpl&& entry);
880
881 private:
882     /// @brief URID table entries
883     ara::core::Vector<URIDTableEntryImpl> entries_;
884 };
885
```

```
886 ///
887 /// @brief Diagnostic protocol data unit (D-PDU API)
888 ///
889 /// This class implement the main D-PDU API functions.
890 class PduApi
891 {
892 public:
893     ///
894     /// @brief Get the instance of the PduApi Singleton class
895     ///
896     /// @return instance of the PduApi Singleton class
897     ///
898     static PduApi* getInstance()
899     {
900         static PduApi instance;
901         return &instance;
902     }
903
904     ///
905     /// @brief Detecting all the available MVCI protocol modules
906     ///
907     /// @param OptionStr Not used
908     /// @param pAPITag Not used
909     /// @return PDU_ERR_SHARING_VIOLATION if the D-PDU API is already
    constructed,
910     ///         PDU_STATUS_NOERROR f the D-PDU API is not constructed and
911     ///         available modules are detected.
912     ///
913     T_PDU_ERROR PDUConstruct(CHAR8* OptionStr, void* pAPITag);
914
915     ///
916     /// @brief Destroy/erase the available modules and their data
917     ///
918     ///        Set the D-PDU API status to "Unconstructed"
919     /// @return PDU_STATUS_NOERROR
920     ///
921     T_PDU_ERROR PDUDestruct();
922
923     ///
924     /// @brief Establish connection to the specified MVCI protocol module
925     ///
926     ///        Set the module status to "Ready"
927     /// @param hMod Handle of the MVCI protocl module to be connected.
928     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the D-PDU API is not
929     ///         constructed,
930     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module Handle
    is
931     ///         invalid. PDU_STATUS_NOERROR if the module is successfully
932     ///         connected.
933     ///
934     T_PDU_ERROR PDUModuleConnect(UNUM32 hMod);
935
936     ///
937     /// @brief Disconnect the specified MVCI protocol module
938     ///
939     ///        Set the module status to "Available"
```

```
940     /// @param hMod Handle of the MVCI protocl module to be diconnected.
941     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the D-PDU API is not
942     ///         constructed,
943     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module Handle
    is
944     ///         invalid.
945     ///         PDU_STATUS_NOERROR if the module is successfully
    disconnected.
946     ///
947     T_PDU_ERROR PDUModuleDisconnect(UNUM32 hMod);
948
949     ///
950     /// @brief Get module type Id, module handle information, vendor-
    specific
951     ///        string information and module status. Allocate
    PDU_MODULE_ITEM
952     ///        structure and fill the call-by-reference variable
    pModuleIdList
953     /// @param pModuleIdList Pointer for storing the pointer to Module Type
     Ids
954     ///        and the Module handles for all the modules that are
    connected.
955     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the D-PDU API is not
956     ///         constructed,
957     ///         PDU_ERR_INVALID_PARAMETERS if the pModuleIdList parameter
    is
958     ///         NULL.
959     ///         PDU_STATUS_NOERROR if the function is successfully called.
960     ///
961     T_PDU_ERROR PDUGetModuleIds(PDU_MODULE_ITEM** pModuleIdList);
962
963     ///
964     /// @brief Destroy the given item.
965     ///
966     ///        Free memory reserved by the D-PDU API for the given item.
967     /// @param pItem Pointer to the item to be destroyed.
968     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the D-PDU API is not
969     ///         constructed,
970     ///         PDU_ERR_INVALID_PARAMETERS if the pItem is invalid or NULL,
971     ///         PDU_STATUS_NOERROR if the item type is sucessfully
    destroyed.
972     ///
973     T_PDU_ERROR PDUDestroyItem(PDU_ITEM* pItem);
974
975     ///
976     /// @brief Create a ComLogicalLink for the given ressource Id.
977     ///
978     ///        Stores the ComLogicalLink handle in phCll parameter.
979     /// @param hMod Hanlde of the MVCI protocl module.
980     /// @param pRscData Not used.
981     /// @param resourceId Ressource Id
982     /// @param pCllTag Not used.
983     /// @param phCll Call-by-reference for storage of the ComLogicalLink
    handle.
984     /// @param pCllCreateFlag Not used.
985     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the D-PDU API is not
```

```
986     ///         constructed,
987     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module Handle
        is
988     ///         invalid.
989     ///         PDU_ERR_INVALID_PARAMETERS in case of Invalid NULL pointer
        for
990     ///         phCLL or invalid ressource id.
991     ///         PDU_STATUS_NOERROR if the ComLogicalLink is sucessfully
        created.
992     ///
993     T_PDU_ERROR PDUCreateComLogicalLink(UNUM32 hMod,
994         PDU_RSC_DATA* pRscData,
995         UNUM32 resourceId,
996         void* pCllTag,
997         UNUM32* phCll,
998         PDU_FLAG_DATA* pCllCreateFlag);
999
1000    ///
1001    /// @brief Destroy the given ComLogicalLink
1002    ///
1003    ///         Free memory reserved by the D-PDU API for the given
1004    ///         ComLogicalLink.
1005    /// @param hMod Handle of the MVCI protocl module.
1006    /// @param hCll Handle of the ComLogicalLink to be destroyed.
1007    /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the D-PDU API is not
1008    ///         constructed,
1009    ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1010    ///         ComLogicalLink handles are invalid,
1011    ///         PDU_ERR_MODULE_NOT_CONNECTED if the MVCI protocol module
        has not
1012    ///         been connected (i.e the module is not in "Ready" state).
1013    ///         PDU_STATUS_NOERROR if the ComLogicalLink is successfully
1014    ///         destroyed.
1015    ///
1016    T_PDU_ERROR PDUDestroyComLogicalLink(UNUM32 hMod, UNUM32 hCll);
1017
1018    ///
1019    /// @brief Connect the ComLogicalLink to the vehicle bus.
1020    ///
1021    ///        Associate the ComLogicalLink to the corresponding ComParams
1022    ///        working buffer and the URID table. Set the ComLogicalLink to
        the
1023    ///        PDU_CLLST_ONLINE state.
1024    /// @param hMod Handle of MVCI protocol module.
1025    /// @param hCll Handle of the ComLogicalLink to be connected.
1026    /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
        constructed,
1027    ///         PDU_ERR_MODULE_NOT_CONNECTED if the module is not connected
        ,
1028    ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1029    ///         ComLogicalLink handles are invalid,
1030    ///         PDU_ERR_CLL_CONNECTED if the ComLogicalLink is already in
        the
1031    ///         "online" state.
1032    ///         PDU_STATUS_NOERROR if the ComLogicalLink is successfully
1033    ///         connected.
```

```
1034     ///
1035     T_PDU_ERROR PDUConnect(UNUM32 hMod, UNUM32 hCll);
1036
1037     ///
1038     /// @brief Disonnect the ComLogicalLink from the vehicle bus.
1039     ///
1040     /// @param hMod Handle of MVCI protocol module.
1041     /// @param hCll Hanlde of the ComLogicalLink to be disconnected.
1042     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
         constructed,
1043     ///         PDU_ERR_MODULE_NOT_CONNECTED if the module is not connected
         ,
1044     ///         PDU_ERR_CLL_NOT_CONNECTED if the ComLogicalLink is not
         connected,
1045     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1046     ///         ComLogicalLink handles are invalid,
1047     ///         PDU_STATUS_NOERROR if theComLogicalLink is successfully
1048     ///         disconnected.
1049     ///
1050     T_PDU_ERROR PDUDisconnect(UNUM32 hMod, UNUM32 hCll);
1051
1052     ///
1053     /// @brief Register a callback function for event notification.
1054     ///
1055     ///        Add the callback function pointer to the propoer
         ComLogicalLink
1056     ///        object.
1057     /// @param hMod Handle of MVCI protocol module.
1058     /// @param hCll Hanlde of the ComLogicalLink.
1059     /// @param fnCB Reference of callback function to be used for event
1060     ///         notification.
1061     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
         constructed,
1062     ///         PDU_ERR_MODULE_NOT_CONNECTED if the module is not connected
         ,
1063     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1064     ///         ComLogicalLink handles are invalid,
1065     ///         PDU_ERR_FCT_FAILED if the fnCB pointer is NULL.
1066     ///         PDU_STATUS_NOERROR if the callback function is successfully
1067     ///         registered.
1068     ///
1069     T_PDU_ERROR PDURegisterEventCallback(UNUM32 hMod, UNUM32 hCll,
         CALLBACKFNC fnCB);
1070
1071     ///
1072     /// @brief Get the event item data (PDU_EVENT_ITEM) for the given event
1073     ///        source.
1074     ///
1075     ///         Allocate memory for PDU_EVENT_ITEM and fill out the event
         item
1076     ///         information.
1077     /// @param hMod Handle of MVCI protocol module.
1078     /// @param hCll Hanlde of the ComLogicalLink.
1079     /// @param pEventItem Call-by-reference place for storing the pointer
         to the
1080     /// event item corresponding to the given event, hMod and hCLL.
```

```
1081     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
    constructed,
1082     ///         PDU_ERR_INVALID_PARAMETERS if pEventItem pointer is NULL.
1083     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1084     ///         ComLogicalLink handles are invalid,
1085     ///         PDU_ERR_EVENT_QUEUE_EMPTY ifno more event items are
    available,
1086     ///         PDU_STATUS_NOERROR if the event item is successfully
    retrieved.
1087     ///
1088     T_PDU_ERROR PDUGetEventItem(UNUM32 hMod, UNUM32 hCll, PDU_EVENT_ITEM**
    pEventItem);
1089
1090     ///
1091     /// @brief Creates a ComPrimitive for sending/receiving data and place
    it in
1092     ///        the CoP Queue.
1093     ///
1094     /// @param hMod Handle of MVCI protocol module.
1095     /// @param hCll Hanlde of the ComLogicalLink.
1096     /// @param uiCoPType Type of the ComPrimitive to be started.
1097     /// @param uiCoPDataSize Size of the ComPrimitive data.
1098     /// @param pCoPData Reference of the buffer holding the data.
1099     /// @param pCopCtrlData Pointer to the control data structure for the
1100     ///        ComPrimitive.
1101     /// @param pCoPTag Not used.
1102     /// @param phCoP Call-by-reference place for storage of ComPrimitive
    handle
1103     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
    constructed,
1104     ///         PDU_ERR_INVALID_PARAMETERS if pCoPData or pCopCtrlData
    pointers
1105     ///         are NULL.
1106     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1107     ///         ComLogicalLink handles are invalid,
1108     ///         PDU_ERR_FCT_FAILED if uiCoPType is different than
1109     ///         PDU_COPT_SENDRECV (the current D-PDU API implementation
    support
1110     ///         only the Send-Receive type)
1111     ///         PDU_STATUS_NOERROR if the ComPrimitive is successfully
    started.
1112     ///
1113     T_PDU_ERROR PDUStartComPrimitive(UNUM32 hMod,
1114         UNUM32 hCll,
1115         T_PDU_COPT uiCoPType,
1116         UNUM32 uiCoPDataSize,
1117         UNUM8* pCoPData,
1118         PDU_COP_CTRL_DATA* pCopCtrlData,
1119         void* pCoPTag,
1120         UNUM32* phCoP);
1121
1122     ///
1123     /// @brief Get a ComParam from the ComParams buffer for the given
    module and
1124     /// ComLogicalLink handles.
1125     ///
```

```
1126     ///         Allocate memory for the PDU_PARAM_ITEM results and fill out
         the
1127     ///         ComParam information in it.
1128     /// @param hMod Handle of MVCI protocol module.
1129     /// @param hCll Hanlde of the ComLogicalLink.
1130     /// @param uiParamId ID value of the ComParam to be requested (see MDF
         to get
1131     ///        the list of ComParam IDs).
1132     /// @param pParamItem Call-by-reference place for storing the ComParam
         item.
1133     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
         constructed,
1134     ///         PDU_ERR_MODULE_NOT_CONNECTED if the module is not connected
         ,
1135     ///         PDU_ERR_INVALID_PARAMETERS if uiParamId is invalid or
         pParamItem
1136     ///         pointer is NULL,
1137     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1138     ///         ComLogicalLink handles are invalid,
1139     ///         PDU_STATUS_NOERROR if the requested ComParam is
         successfully
1140     ///         retrieved.
1141     ///
1142     T_PDU_ERROR PDUGetComParam(UNUM32 hMod, UNUM32 hCll, UNUM32 uiParamId,
         PDU_PARAM_ITEM** pParamItem);
1143
1144     ///
1145     /// @brief Set a ComParam in the ComParams buffer for the given module
         and
1146     ///        ComLogicalLink handles.
1147     ///
1148     /// @param hMod Handle of MVCI protocol module.
1149     /// @param hCll Hanlde of the ComLogicalLink.
1150     /// @param pParamItem ComParam item structure with the ComParam element
         to be
1151     ///        set.
1152     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
         constructed,
1153     ///         PDU_ERR_MODULE_NOT_CONNECTED if the module is not connected
         ,
1154     ///         PDU_ERR_INVALID_PARAMETERS if pParamItem pointer is NULL or
         the
1155     ///         ComParam ID is invalid,
1156     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1157     ///         ComLogicalLink handles are invalid,
1158     ///         PDU_ERR_FCT_FAILED if the ComLogicalLink is in "online"
         state,
1159     ///         PDU_STATUS_NOERROR if the given ComParam is successfully
         set.
1160     ///
1161     T_PDU_ERROR PDUSetComParam(UNUM32 hMod, UNUM32 hCll, PDU_PARAM_ITEM*
         pParamItem);
1162
1163     ///
1164     /// @brief Get information of all unique response identifiers
         configured for
```

```
1165     ///        the ComLogicalLink.
1166     ///
1167     /// @param hMod Handle of MVCI protocol module.
1168     /// @param hCll Hanlde of the ComLogicalLink.
1169     /// @param pUniqueRespIdTable Call-by-reference place for storing the
         Unique
1170     ///        Response ID Table for the given CLL.
1171     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
         constructed,
1172     ///         PDU_ERR_MODULE_NOT_CONNECTED if the module is not connected
         ,
1173     ///         PDU_ERR_INVALID_PARAMETERS if pUniqueRespIdTable pointer is
          NULL
1174     ///         or the ComParam ID is invalid,
1175     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1176     ///         ComLogicalLink handles are invalid,
1177     ///         PDU_STATUS_NOERROR if the URID table is successfully
         retrieved.
1178     ///
1179     T_PDU_ERROR
1180     PDUGetUniqueRespIdTable(UNUM32 hMod, UNUM32 hCll,
         PDU_UNIQUE_RESP_ID_TABLE_ITEM** pUniqueRespIdTable);
1181
1182     ///
1183     /// @brief Set Unique Response Id Table for a ComLogicalLink.
1184     ///
1185     /// @param hMod Handle of MVCI protocol module.
1186     /// @param hCll  Hanlde of the ComLogicalLink.
1187     /// @param pUniqueRespIdTable Call-by-reference witch contains the URID
          table
1188     ///        for the given ComLogicalLink.
1189     /// @return PDU_ERR_PDUAPI_NOT_CONSTRUCTED if the module is not
         constructed,
1190     ///         PDU_ERR_MODULE_NOT_CONNECTED if the module is not connected
         ,
1191     ///         PDU_ERR_INVALID_PARAMETERS if pUniqueRespIdTable pointer is
          NULL
1192     ///         or the ComParam ID is invalid,
1193     ///         PDU_ERR_INVALID_HANDLE if the MVCI protocol module or
1194     ///         ComLogicalLink handles are invalid,
1195     ///         PDU_ERR_FCT_FAILED if the ComLogicalLink is in "online"
         state.
1196     ///         PDU_STATUS_NOERROR if the URID table is successfully
         retrieved.
1197     ///
1198     T_PDU_ERROR
1199     PDUSetUniqueRespIdTable(UNUM32 hMod, UNUM32 hCll,
         PDU_UNIQUE_RESP_ID_TABLE_ITEM* pUniqueRespIdTable);
1200
1201     ~PduApi() = default;
1202
1203 private:
1204     enum class PduApiState
1205     {
1206         kUnconstructed = 0,
1207         kConstructed
```

```
1208     };
1209
1210     using PduDataType = std::variant<std::shared_ptr<PDU_EVENT_ITEM>,
1211         std::shared_ptr<PDU_PARAM_ITEM>,
1212         std::shared_ptr<PDU_RSC_DATA>,
1213         std::shared_ptr<PDU_MODULE_ITEM>,
1214         std::shared_ptr<PDU_RSC_ID_ITEM>,
1215         std::shared_ptr<PDU_UNIQUE_RESP_ID_TABLE_ITEM>>;
1216
1217     PduApi();
1218
1219     /// @brief Allocate PDU_MODULE_ITEM structure and fill the call-by-
    reference
1220     /// variable pModuleIdList
1221     ///
1222     /// @return the filled PDU_MODULE_ITEM structure
1223     std::shared_ptr<PDU_MODULE_ITEM> createModuleItem();
1224
1225     /// @brief D-PDU API states (constructed, Not constructed)
1226     PduApiState pduapiStatus_{PduApiState::kUnconstructed};
1227
1228     /// @brief list of the available MVCI protocol modules.
1229     ara::core::Map<UNUM32, Module> availableModules_;
1230
1231     /// @brief D-PDU API data, events and param items
1232     std::list<PduDataType> pduData_;
1233 };
```

G Change history of AUTOSAR traceable items

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

G.1 Traceable item history of this document according to AUTOSAR Release R23-11

G.1.1 Added Specification Items in R23-11

Number	Heading
[SWS_VUCM_00136]	Definition of Application Error Domain of functional cluster VUCM
[SWS_VUCM_00177]	Definition of ImplementationDataType SwNameVersionVectorType
[SWS_VUCM_00178]	Definition of Port VehiclePackageManagement provided by functional cluster VUCM
[SWS_VUCM_00179]	Definition of Port VehicleStateManagerInterface provided by functional cluster VUCM
[SWS_VUCM_00180]	Definition of Port VehicleDriverApplicationInterface provided by functional cluster VUCM
[SWS_VUCM_00181]	Definition of ServiceInterface VehiclePackageManagement
[SWS_VUCM_00182]	Definition of ServiceInterface VehicleDriverApplicationInterface
[SWS_VUCM_00183]	Definition of ServiceInterface VehicleStateManagerInterface
[SWS_VUCM_00210]	Transferring of software packages on `kProcessing` state
[SWS_VUCM_00251]	Definition of ImplementationDataType CampaignHistoryType
[SWS_VUCM_00252]	Definition of ImplementationDataType CampaignResultType
[SWS_VUCM_00253]	Definition of ImplementationDataType UCMStepErrorType
[SWS_VUCM_00254]	Definition of ImplementationDataType UCMHistoryVectorType
[SWS_VUCM_00255]	Definition of ImplementationDataType SoftwarePackageStepType
[SWS_VUCM_00256]	Definition of ImplementationDataType UCMMasterResolutionType
[SWS_VUCM_00268]	Definition of ImplementationDataType SwPackageDescType
[SWS_VUCM_00269]	Definition of ImplementationDataType SwPackageDescVectorType
[SWS_VUCM_00290]	Definition of ImplementationDataType UCMHistoryType
[SWS_VUCM_00291]	Definition of ImplementationDataType UCMStepErrorVectorType
[SWS_VUCM_00296]	Definition of ImplementationDataType CampaignHistoryVectorType
[SWS_VUCM_00297]	Retry Strategy for ServiceBusy
[SWS_VUCM_00298]	Retry Strategy for UpdateSessionRejected
[SWS_VUCM_00304]	Definition of ImplementationDataType UCMMasterResolutionVectorType
[SWS_VUCM_00307]	Definition of ImplementationDataType VehicleUCMInfo
[SWS_VUCM_00308]	Definition of ImplementationDataType UCMIdentifiersAndVersionsType

▽

△

Number	Heading
[SWS_VUCM_01003]	V-UCM checks states of UCMs
[SWS_VUCM_01005]	V-UCM is discovering UCMs in vehicle
[SWS_VUCM_01011]	TransferVehiclePackage InsufficientMemory
[SWS_VUCM_01013]	Too big block size received by V-UCM
[SWS_VUCM_01014]	Packages transferring sequence
[SWS_VUCM_01015]	Invalid Vehicle Package manifest
[SWS_VUCM_01016]	Invalid Package Manifest
[SWS_VUCM_01017]	RequestedPackage field
[SWS_VUCM_01018]	TransferVehiclePackage kBusyWithCampaign
[SWS_VUCM_01019]	V-UCM initialization
[SWS_VUCM_01020]	Retry Strategy for BlockInconsistent
[SWS_VUCM_01021]	Definition of Port PackageManagement required by functional cluster VUCM
[SWS_VUCM_01101]	Provide information of installed Software Clusters in vehicle
[SWS_VUCM_01103]	Inform Backend of needed Software Packages for an update
[SWS_VUCM_01105]	Interaction of V-UCM with Vehicle Driver
[SWS_VUCM_01109]	V-UCM provides a safety interface
[SWS_VUCM_01114]	Definition of ImplementationDataType SafetyConditionType
[SWS_VUCM_01117]	V-UCM VehicleConditionCollection field
[SWS_VUCM_01118]	V-UCM waiting for vehicle driver approval
[SWS_VUCM_01119]	Report information of Software Packages
[SWS_VUCM_01120]	Provide Software Packages general information
[SWS_VUCM_01122]	Supported physical layers by D-PDU API implementation
[SWS_VUCM_01123]	Supported application layers by D-PDU API implementation
[SWS_VUCM_01124]	Supported protocols by D-PDU API implementation
[SWS_VUCM_01131]	PDUIoCtl(PDU_IOCTL_RESET)
[SWS_VUCM_01132]	PDUIoCtl(PDU_IOCTL_START_MSG_FILTER), PDUIoCtl(PDU_IOCTL_CLEAR_MSG_FILTER), PDUIoCtl(PDU_IOCTL_STOP_MSG_FILTER)
[SWS_VUCM_01135]	Get Software Clusters descriptions from a vehicle
[SWS_VUCM_01138]	Definition of ImplementationDataType SafetyStateType
[SWS_VUCM_01139]	V-UCM configured behavior when vehicle safety is not met
[SWS_VUCM_01177]	Definition of ImplementationDataType CampaignStateType
[SWS_VUCM_01178]	Definition of ImplementationDataType TransferStateType
[SWS_VUCM_01201]	Sequential orchestration of campaigns
[SWS_VUCM_01203]	CampaignState field
[SWS_VUCM_01204]	Initial state
[SWS_VUCM_01205]	V-UCM internal state persistency
[SWS_VUCM_01207]	Trigger on kSoftwarePackage_Transferring state
[SWS_VUCM_01209]	Trigger on kProcessing state

▽

△

Number	Heading
[SWS_VUCM_01212]	Trigger on kActivating state
[SWS_VUCM_01214]	Final action on kVehicleChecking state
[SWS_VUCM_01215]	Trigger on kCancelling state
[SWS_VUCM_01216]	Final action on kCancelling state
[SWS_VUCM_01217]	Monitoring of UCMs
[SWS_VUCM_01218]	Transition from kIdle state to kSyncing state
[SWS_VUCM_01219]	Transition from kSyncing state to kIdle state
[SWS_VUCM_01220]	Transition from kIdle state to kVehiclePackageTransferring and kTransferring states
[SWS_VUCM_01221]	Transition from kVehiclePackageTransferring state and kTransferring state to kCancelling state
[SWS_VUCM_01222]	Transition from kVehiclePackageTransferring state to kSoftwarePackage_Transferring state
[SWS_VUCM_01227]	Transition from kSoftwarePackage_Transferring state and kTransferring state to kCancelling state
[SWS_VUCM_01228]	Transition from kSoftwarePackage_Transferring state and kTransferring state to kProcessing state and kUpdating state
[SWS_VUCM_01229]	SafetyConditions while processing stream
[SWS_VUCM_01234]	Transition from kProcessing state to kActivating state
[SWS_VUCM_01236]	Transition from kProcessing state and kUpdating state to kCancelling state
[SWS_VUCM_01239]	Transition from kActivating state and kUpdating state to kCancelling state
[SWS_VUCM_01240]	Transition from kActivating state to kVehicleChecking state
[SWS_VUCM_01241]	Transition from kVehicleChecking state and kUpdating state to kCancelling state
[SWS_VUCM_01242]	Transition from kVehicleChecking state and kUpdating state to kIdle state
[SWS_VUCM_01243]	Transition from kCancelling state to kIdle state
[SWS_VUCM_01244]	Cancellation of an update campaign shall be possible
[SWS_VUCM_01246]	Unreachable UCM during update campaign
[SWS_VUCM_01247]	Method to read History Report
[SWS_VUCM_01248]	Content of History Report
[SWS_VUCM_01265]	TransferState field
[SWS_VUCM_01266]	Subordinate Not Available On The Network
[SWS_VUCM_01267]	Vehicle State Manager Communication Error
[SWS_VUCM_01268]	Vehicle Driver Interface Communication Error
[SWS_VUCM_01269]	Campaign cancellation history
[SWS_VUCM_01270]	New campaign disabling
[SWS_VUCM_01271]	New campaign enabling
[SWS_VUCM_01272]	VehicleCheck call not permitted

▽

△

Number	Heading
[SWS_VUCM_01273]	`CancelCampaign` `kCancelFailed` error
[SWS_VUCM_01274]	`CancelCampaign` `kOperationNotPermitted` error
[SWS_VUCM_01275]	Safety conditions during activation
[SWS_VUCM_01276]	Transition from `kRollingBackFailed` state to `kIdle` state
[SWS_VUCM_01277]	Transition from `kCancelling` state to `kRollingBackFailed` state
[SWS_VUCM_01278]	`V-UCM` behaviour in case a safety condition is not supported by `Vehicle State Manager`
[SWS_VUCM_01279]	Keep history of Driver notification during campaign
[SWS_VUCM_01280]	Maximum campaign duration
[SWS_VUCM_01301]	`Vehicle Package` authentication
[SWS_VUCM_01302]	`Vehicle Package` authentication failure
[SWS_VUCM_01303]	Dependencies between `Software Clusters`
[SWS_VUCM_01305]	`Vehicle Package` format
[SWS_VUCM_01306]	`TransferExit` Invalid package manifest
[SWS_VUCM_01307]	`Vehicle Package` format not supported
[SWS_VUCM_01308]	Check Vehicle Package version compatibility against V-UCM version
[SWS_VUCM_01309]	Definition of ImplementationDataType VehicleConditionCollectionType
[SWS_VUCM_01310]	Definition of ImplementationDataType VehicleConditionType
[SWS_VUCM_01311]	Semantic versionning

Table G.1: Added Specification Items in R23-11

G.1.2 Changed Specification Items in R23-11

none

G.1.3 Deleted Specification Items in R23-11

none

G.1.4 Added Constraints in R23-11

Number	Heading
[SWS_VUCM_CONSTR_00003]	Exclusive use of Vehicle Driver Interface
[SWS_VUCM_CONSTR_00004]	Unsupported safety by Vehicle driver interface
[SWS_VUCM_CONSTR_00005]	Safety state change
[SWS_VUCM_CONSTR_00006]	Exclusive use of Vehicle State Manager
[SWS_VUCM_CONSTR_00007]	Unsupported safety conditions by Vehicle State Manager
[SWS_VUCM_CONSTR_00009]	Safety condition change
[SWS_VUCM_CONSTR_00011]	Flashing Adapter provided interface
[SWS_VUCM_CONSTR_00013]	Confidential information protection
[SWS_VUCM_CONSTR_00015]	Trigger on `kVehicleChecking` state
[SWS_VUCM_CONSTR_00016]	OTA Client use of RequestedPackage field
[SWS_VUCM_CONSTR_00017]	Interaction of `V-UCM` with Vehicle Driver
[SWS_VUCM_CONSTR_00018]	V-UCM uniqueness in vehicle

Table G.2: Added Constraints in R23-11

G.1.5 Changed Constraints in R23-11

none

G.1.6 Deleted Constraints in R23-11

none

G.2 Traceable item history of this document according to AUTOSAR Release R24-11

G.2.1 Added Specification Items in R24-11

Number	Heading
[SWS_VUCM_01022]	Definition of ImplementationDataType CampaignStateProgressInfoType
[SWS_VUCM_01023]	UcmStep order
[SWS_VUCM_01024]	LogMessage CampaignStarted
[SWS_VUCM_01025]	LogMessage CampaignAborted
[SWS_VUCM_01026]	LogMessage SoftwarePackageTransferStarted

▽

△

Number	Heading
[SWS_VUCM_01027]	LogMessage SoftwarePackageTransferFinished
[SWS_VUCM_01028]	LogMessage VehiclePackageTransferFailed
[SWS_VUCM_01029]	LogMessage SoftwarePackageTransferFailed
[SWS_VUCM_01030]	LogMessage UpdateStarted
[SWS_VUCM_01031]	LogMessage CampaignFailed
[SWS_VUCM_01032]	LogMessage CampaignSuccessful
[SWS_VUCM_01033]	Unreachable UCM during Packages transferring sequence
[SWS_VUCM_01034]	Retry Strategy for Transfer methods
[SWS_VUCM_01140]	Definition of ImplementationDataType VehiclePackageDescriptionType
[SWS_VUCM_01141]	V-UCM behavior in case a safety condition is not supported by Vehicle Driver Interface
[SWS_VUCM_01142]	Definition of Field VehicleDriverApplicationInterface.ApprovalRequired
[SWS_VUCM_01143]	Definition of Field VehicleDriverApplicationInterface.CampaignState
[SWS_VUCM_01144]	Definition of Field VehicleDriverApplicationInterface.VehicleCondition Collection
[SWS_VUCM_01145]	Definition of Method VehicleDriverApplicationInterface.CancelCampaign
[SWS_VUCM_01146]	Definition of Method VehicleDriverApplicationInterface.AllowCampaign
[SWS_VUCM_01147]	Definition of Method VehicleDriverApplicationInterface.Approve
[SWS_VUCM_01148]	Definition of Method VehicleDriverApplicationInterface.ReportUnsupported SafetyConditions
[SWS_VUCM_01149]	Definition of Method VehicleDriverApplicationInterface.GetCampaignHistory
[SWS_VUCM_01150]	Definition of Method VehicleDriverApplicationInterface.GetSwClusterInfo
[SWS_VUCM_01151]	Definition of Method VehicleDriverApplicationInterface.GetSwPackage Description
[SWS_VUCM_01152]	Definition of Method VehicleDriverApplicationInterface.GetVehiclePackage Description
[SWS_VUCM_01156]	Definition of Field VehiclePackageManagement.TransferState
[SWS_VUCM_01157]	Definition of Field VehiclePackageManagement.CampaignState
[SWS_VUCM_01158]	Definition of Field VehiclePackageManagement.RequestedPackage
[SWS_VUCM_01159]	Definition of Field VehiclePackageManagement.VehicleConditionCollection
[SWS_VUCM_01160]	Definition of Method VehiclePackageManagement.CancelCampaign
[SWS_VUCM_01161]	Definition of Method VehiclePackageManagement.AllowCampaign
[SWS_VUCM_01162]	Definition of Method VehiclePackageManagement.DeleteTransfer
[SWS_VUCM_01163]	Definition of Method VehiclePackageManagement.TransferData
[SWS_VUCM_01164]	Definition of Method VehiclePackageManagement.TransferExit
[SWS_VUCM_01165]	Definition of Method VehiclePackageManagement.TransferStart
[SWS_VUCM_01166]	Definition of Method VehiclePackageManagement.TransferVehiclePackage
[SWS_VUCM_01167]	Definition of Method VehiclePackageManagement.GetCampaignHistory
[SWS_VUCM_01168]	Definition of Method VehiclePackageManagement.GetSwClusterInfo
[SWS_VUCM_01169]	Definition of Method VehiclePackageManagement.GetSwPackages

▽

△

Number	Heading
[SWS_VUCM_01170]	Definition of Method VehiclePackageManagement.GetVehicleUCMInfo
[SWS_VUCM_01171]	Definition of Method VehiclePackageManagement.SwPackageInventory
[SWS_VUCM_01172]	Definition of Field VehicleStateManagerInterface.CampaignState
[SWS_VUCM_01173]	Definition of Field VehicleStateManagerInterface.VehicleConditionCollection
[SWS_VUCM_01174]	Definition of Method VehicleStateManagerInterface.PublishSafetyState
[SWS_VUCM_01175]	Software Package deletion
[SWS_VUCM_01176]	Software Package authentication failure
[SWS_VUCM_01179]	Definition of Method VehicleStateManagerInterface.VehicleCheck
[SWS_VUCM_01180]	DeleteTranfer OperationNotPermited error
[SWS_VUCM_01282]	Transition from `kVehicleChecking` state to `kIdle` state
[SWS_VUCM_01283]	Errors returned by UCM during campaign
[SWS_VUCM_01312]	Campaign start - log start of update campaign
[SWS_VUCM_01313]	Campaign abort - log abort of update campaign
[SWS_VUCM_01314]	`Software Package` transfer - log start of `Software Package` transfer
[SWS_VUCM_01315]	`Software Package` transfer - log end of `Software Package` transfer
[SWS_VUCM_01316]	`Vehicle Package` transfer failure - log failure during transfer of `Vehicle Package`
[SWS_VUCM_01317]	`Software Package` transfer failure - log failure during transfer of `Software Package`
[SWS_VUCM_01318]	Start of update - log start of update process
[SWS_VUCM_01319]	Campaign result - log failure of update campaign
[SWS_VUCM_01320]	Campaign result - log success of update campaign

Table G.3: Added Specification Items in R24-11

G.2.2 Changed Specification Items in R24-11

Number	Heading
[SWS_VUCM_00136]	Definition of Application Error Domain of functional cluster VUCM
[SWS_VUCM_00181]	Definition of ServiceInterface VehiclePackageManagement
[SWS_VUCM_00182]	Definition of ServiceInterface VehicleDriverApplicationInterface
[SWS_VUCM_00183]	Definition of ServiceInterface VehicleStateManagerInterface
[SWS_VUCM_00252]	Definition of ImplementationDataType CampaignResultType
[SWS_VUCM_00256]	Definition of ImplementationDataType VUCMResolutionType
[SWS_VUCM_00268]	Definition of ImplementationDataType SwPackageDescType
[SWS_VUCM_00304]	Definition of ImplementationDataType VUCMResolutionVectorType
[SWS_VUCM_01003]	`V-UCM` checks states of `UCM`s

▽

△

Number	Heading
[SWS_VUCM_01018]	`TransferVehiclePackage kBusyWithCampaign`
[SWS_VUCM_01118]	V-UCM waiting for vehicle driver approval
[SWS_VUCM_01207]	Trigger on `kSoftwarePackage_Transferring` state
[SWS_VUCM_01218]	Transition from `kIdle` state to `kSyncing` state
[SWS_VUCM_01219]	Transition from `kSyncing` state to `kIdle` state
[SWS_VUCM_01220]	Transition from `kIdle` state to `kVehiclePackageTransferring`
[SWS_VUCM_01221]	Transition from `kVehiclePackageTransferring` state
[SWS_VUCM_01227]	Transition from `kSoftwarePackage_Transferring` state to `kCancelling` state
[SWS_VUCM_01228]	Transition from `kSoftwarePackage_Transferring` state to `kProcessing` state
[SWS_VUCM_01234]	Transition from `kProcessing` state to `kActivating` state
[SWS_VUCM_01236]	Transition from `kProcessing` state to `kCancelling` state
[SWS_VUCM_01239]	Transition from `kActivating` state to `kCancelling` state
[SWS_VUCM_01241]	Transition from `kVehicleChecking` state to `kCancelling` state
[SWS_VUCM_01242]	Transition from `kVehicleChecking` stateto `kIdle` state
[SWS_VUCM_01243]	Transition from `kCancelling` state to `kIdle` state
[SWS_VUCM_01244]	Cancellation of an update campaign shall be possible
[SWS_VUCM_01246]	Unreachable UCM during update campaign
[SWS_VUCM_01265]	TransferState field
[SWS_VUCM_01266]	Subordinate Not Available On The Network
[SWS_VUCM_01270]	New campaign disabling
[SWS_VUCM_01276]	Transition from `kRollingBackFailed` state to `kIdle` state
[SWS_VUCM_01277]	Transition from `kCancelling` state to `kRollingBackFailed` state
[SWS_VUCM_01278]	`V-UCM` behaviour in case a safety condition is not supported by `Vehicle State Manager`
[SWS_VUCM_01305]	`Vehicle Package` format
[SWS_VUCM_01307]	`Vehicle Package` format not supported
[SWS_VUCM_01308]	Check Vehicle Package version compatibility against V-UCM version

Table G.4: Changed Specification Items in R24-11

G.2.3 Deleted Specification Items in R24-11

none

G.2.4 Added Constraints in R24-11

Number	Heading
[SWS_VUCM_CONSTR_00019]	Safety handling after reboot

Table G.5: Added Constraints in R24-11

G.2.5 Changed Constraints in R24-11

Number	Heading
[SWS_VUCM_CONSTR_00017]	Interaction of V-UCM with Vehicle Driver

Table G.6: Changed Constraints in R24-11

G.2.6 Deleted Constraints in R24-11

none