Document Title	Specification of State Management
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	908

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	• Remove support for DiagnosticReset • Remove obsolete mentionings of CommunicationGroups • Remove non-implementable requirements for customer-specific StateManagement implementation • Adopt document to new SWS template
2023-11-23	R23-11	AUTOSAR Release Management	• Add Update and Configuration Management support to StateMachine approach • Add Network Management support to StateMachine approach • Add Controller/Agent StateMachine approach • Add UpdateAllowed service interface • Extend StartStartMachine feature of StateMachine approach • Replace Network Management service Interface by C++ API

△

2022-11-24	R22-11	AUTOSAR Release Management	• Introduction of StateMachine design • Harmonized error codes for UpdateRequest interface • Fixed wrong description in UpdateRequest interface • Removed LastResetCause Interface
2021-11-25	R21-11	AUTOSAR Release Management	• Updated method name in Interface towards Update And Configuration Management • Added new error codes in Interface towards Update And Configuration Management • Fixed error handling in Interface towards Update And Configuration Management • Removed timeout supervision for update session • Removed items regarding LastResetCause in Interface towards Diagnostic Management • Added references from chapter 7 to chapter 9
2020-11-30	R20-11	AUTOSAR Release Management	• Interface towards Update And Configuration Management updated • Interface towards Diagnostic Management updated • Introduced Diagnostic Reset based on Communication Groups • Interface towards Platform Health Management updated • Error reactions for supervised entity failures moved to State Management • Introduced PowerModes based on Communication Groups • RequestState and ReleaseRequest interface removed

▽

△

2019-11-28	R19-11	AUTOSAR Release Management	• Interface with ExecutionManagement changed to StateClient • RequestState and ReleaseRequest kept deprecated • Changed Document Status from Final to published
2019-03-29	19-03	AUTOSAR Release Management	• Removed components • RequestState and ReleaseRequest are now deprecated • State Managements internal states can now be influenced by "Trigger" and are distributed by "Notifier" fields
2018-10-31	18-10	AUTOSAR Release Management	• Initial release

Disclaimer

Contents

1 Introduction and functional overview

This document is the software specification of the State Management functional cluster within the Adaptive Platform Services.

State Management is responsible for determination the state of any of its internal statemachines, based on information received from other AUTOSAR Adaptive Platform Application or Adaptive Application.

Interaction with other applications includes requesting Function Group State transitions (as well as those for recovery actions), influencing the state from NetworkHandles, and supporting coordinated update sessions by transitioning Function Groups to different states according to the current update session phase. These interactions are facilitated through ara::com service interfaces and C++ APIs. State Management remains highly OEM and project-specific. Depending on the chosen implementation, chapter 7 provides two approaches for working with State Management. One approach focuses on the interface level only, allowing OEM flexibility in implementing internal logic and handling configurations. The other approach provides a standardized method for managing configurations and defining how these configurations interact with internal logic, following a StateMachine-based approach.

Section 7 describes how State Management concepts are realized within the AUTOSAR Adaptive Platform.

2 Acronyms and Abbreviations

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

Terms:	Description:
Network Handle	Network Handles are provided by Network Management. A handle represents a set of (partial) networks.
StateMachine	Collection of modelled StateMachine States, each associated with an ActionList, where TransitionRequestTable and ErrorRecoveryTable are defining possible StateMachine State transitions. A StateMachine can be used to control a set of software in a configurable way. StateMachine is represented by meta-class ModeDeclarationGroupPrototype. The StateManagementStateNotification.stateMachine.category for each StateMachine has to be configured.
StateMachine State	A modelled state of a StateMachine which can be entered based on StateMachine inputs. Each StateMachine State has an associated ActionList. StateMachine State is represented by meta-class ModeDeclaration.
Initial State	A StateMachine State that is configured as ModeDeclarationGroup.InitialMode. This state is used for the very first StateMachine State transition when StateMachine is started (e.g. ActionListItem [SWS_SM_00612] is processed) and no specific StateMachine State is provided as a parameter.
ActionList	Collection of modelled ActionListItems defining actions to be performed on entering a StateMachine State. ActionList is represented by meta-class StateManagementActionList.
ActionListItem	Defines a specific action to be performed, e.g. Function Group State transition, StateMachine State transition or a NetworkHandle switch. ActionListItem is represented by meta-class StateManagementActionItem.
TransitionRequestTable	A modelled set of rules which defines valid StateMachine State transitions for a StateMachine. TransitionRequestTable is represented by a set of meta-class StateManagementTriggerCompareRule.
StateMachine error notification	Notification towards a StateMachine triggered by Platform Health Management or Execution Management to inform StateMachine about a problem in a Function Group. Notification will lead to a change in StateMachine State.
ErrorRecoveryTable	A modelled set of rules which defines StateMachine State transitions for a StateMachine, based on received error events. ErrorRecoveryTable is represented by a set of meta-class StateManagementErrorCompareRule.
SMControlApplication	Project-specific Adaptive Application(s) which evaluates information from the system to request StateMachine State changes from a StateMachine via StateMachineService interface. The SMControlApplication, which is communicating with the Controller has to provide information if update is possible or not (This is done via UpdateAllowed field).

Controller	A StateMachine with StateManagementStateNotification.stateMachine.category set to **STATE_MANAGEMENT_CONTROLLER**. Exactly one instance has to be configured. Controller can make use of ActionListItems for starting and stopping StateMachines of type Agent and provides a configurable way to define lifecycle states of the Machine (e.g. startup and shutdown).
Agent	A StateMachine with StateManagementStateNotification.stateMachine.category set to **STATE_MANAGEMENT_AGENT**. An arbitrary number of instances can be configured. An Agent (in contrast to the Controller) cannot use ActionListItems for starting and stopping other StateMachines.

Table 2.1: Technical Terms

The following technical terms used in this document are defined in the corresponding document mentioned in the table below.

Term	Description
Communication Management	see [2] Specification of Communication Management
State Management	see [3] Requirements of State Management
Execution Management	see [4] Requirements on Execution Management
Modelled Process	see [4] Requirements on Execution Management
Function Group	see [4] Requirements on Execution Management
Function Group State	see [4] Requirements on Execution Management
Machine State	see [4] Requirements on Execution Management
Execution Manifest	see [5] Methodology for Adaptive Platform
Machine Manifest	see [5] Methodology for Adaptive Platform
Platform Health Management	see [6] Specification of Platform Health Management
Network Management	see [7] Specification of Network Management
Diagnostic Management	see [8] Specification of Diagnostics
Identity and Access Management	see [9] Requirements on Identity and Access Management
Update and Configuration Management	see [10] Specification of Update and Configuration Management
Adaptive Application	see [1] AUTOSAR Glossary
AUTOSAR Adaptive Platform	see [1] AUTOSAR Glossary
Adaptive Platform Foundation	see [1] AUTOSAR Glossary
Adaptive Platform Services	see [1] AUTOSAR Glossary
Process	see [1] AUTOSAR Glossary
Manifest	see [1] AUTOSAR Glossary
Executable	see [1] AUTOSAR Glossary
Functional Cluster	see [1] AUTOSAR Glossary
Software Cluster	see [1] AUTOSAR Glossary
Diagnostic Address	see [1] AUTOSAR Glossary
Machine	see [1] AUTOSAR Glossary
Service	see [1] AUTOSAR Glossary
Service Interface	see [1] AUTOSAR Glossary
Service Discovery	see [1] AUTOSAR Glossary

Table 2.2: Reference to Technical Terms

3 Related documentation

3.1 Input documents & related standards and norms

The main documents that serve as input for the specification of the State Management are:

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] Specification of Communication Management
AUTOSAR_AP_SWS_CommunicationManagement

[3] Requirements of State Management
AUTOSAR_AP_RS_StateManagement

[4] Requirements on Execution Management
AUTOSAR_AP_RS_ExecutionManagement

[5] Methodology for Adaptive Platform
AUTOSAR_AP_TR_Methodology

[6] Specification of Platform Health Management
AUTOSAR_AP_SWS_PlatformHealthManagement

[7] Specification of Network Management
AUTOSAR_AP_SWS_NetworkManagement

[8] Specification of Diagnostics
AUTOSAR_AP_SWS_Diagnostics

[9] Requirements on Identity and Access Management
AUTOSAR_AP_RS_IdentityAndAccessManagement

[10] Specification of Update and Configuration Management
AUTOSAR_AP_SWS_UpdateAndConfigurationManagement

[11] Specification of Adaptive Platform Core
AUTOSAR_AP_SWS_Core

[12] Explanation of Adaptive Platform Software Architecture
AUTOSAR_AP_EXP_SWArchitecture

[13] Specification of Manifest
AUTOSAR_AP_TPS_ManifestSpecification

3.2 Further applicable specification

AUTOSAR provides a core specification [11] which is also applicable for State Management. The chapter "General requirements for all FunctionalClusters" of this specification shall be considered an additional and required specification for implementing State Management.

4 Constraints and assumptions

4.1 Known limitations

This section lists known limitations of State Management and their relation to this release of the AUTOSAR Adaptive Platform with the intent to provide an indication how State Management within the context of the AUTOSAR Adaptive Platform will evolve in future releases.

The following functionality is mentioned within this document but is not (fully) specified in this release:

- interaction with Diagnostic Management has to be clarified in one of the upcoming releases.

5 Dependencies to other Functional Clusters

This chapter provides an informative guideline of the interaction of State Management with other Functional Clusters in the AUTOSAR Adaptive Platform. Section 5.1 “Provided Interfaces” lists the public interfaces provided by State Management to other Functional Clusters. Section 5.2 “Required Interfaces” lists the public interfaces required by State Management.

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 internal interfaces are up to the platform provider. Additional internal interfaces, parameters and return values can be added.

A detailed technical architecture documentation of the overall AUTOSAR Adaptive Platform is provided in [12].

5.1 Provided Interfaces

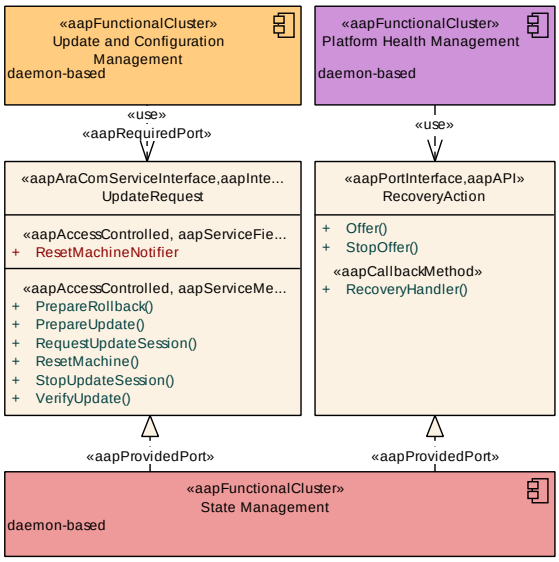

Figure 5.1: Interfaces provided by State Management to other Functional Clusters

Figure 5.1 shows interfaces provided by State Management to other Functional Clusters within the AUTOSAR Adaptive Platform. Table 5.1 provides a complete list of interfaces provided to other Functional Clusters within the AUTOSAR Adaptive Platform.

Interface	*Functional Cluster*	*Purpose*
`RecoveryAction`	Platform Health Management	`Platform Health Management` uses this interface to trigger failure recovery.
`UpdateRequest`	Update and Configuration Management	This interface is used to interact with `State Management` of the Adaptive Platform during an update.

Table 5.1: Interfaces provided to other Functional Clusters

5.2 Required Interfaces

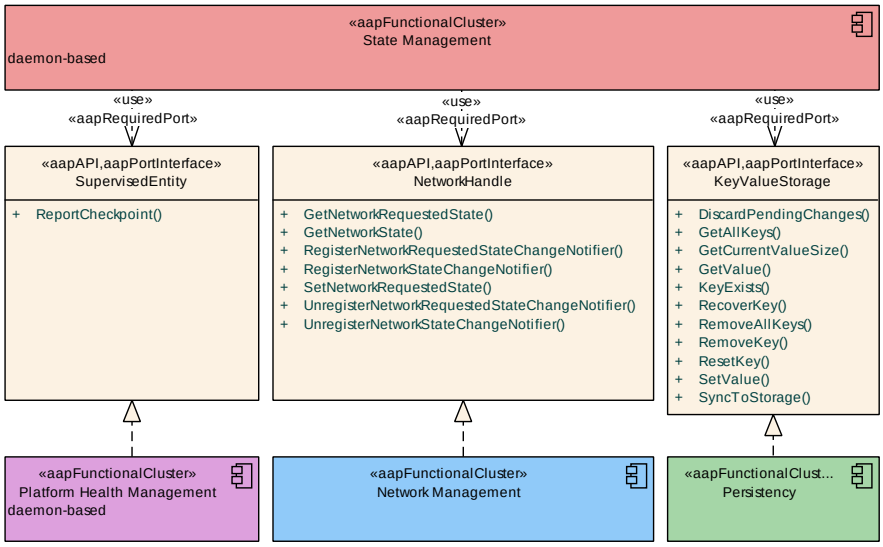

Figure 5.2: Interfaces required by State Management from other Functional Clusters

Figure 5.2 shows the interfaces required by `State Management` from other Functional Clusters within the `AUTOSAR Adaptive Platform`.

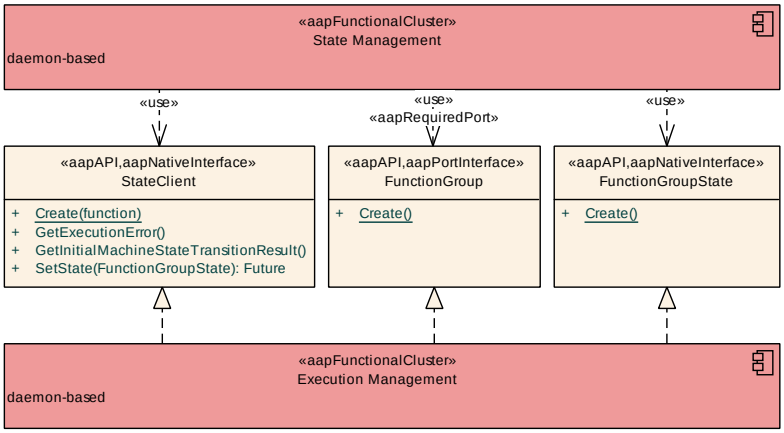

Figure 5.3: Interfaces required by State Management from Execution Management

Figure 5.3 shows interfaces required by `State Management` from `Execution Management` within the `AUTOSAR Adaptive Platform`. Table 5.2 provides a complete

list of required interfaces from other Functional Clusters within the AUTOSAR Adaptive Platform.

Functional Cluster	*Interface*	*Purpose*
Execution Management	ExecutionClient	This interface shall be used to report the state of the State Management process(es).
Execution Management	FunctionGroup	This interface shall be used to construct FunctionGroupStates.
Execution Management	FunctionGroupState	This interface shall be used to request FunctionGroupState transitions.
Execution Management	StateClient	This interface shall be used to request FunctionGroupState transitions.
Log and Trace	Logger	State Management shall use this interface to log standardized messages.
Network Management	NetworkHandle	This interface shall be used to retrieve information about the network status of a NetworkHandle.
Persistency	KeyValueStorage	Used to store the internal state of State Management.
Platform Health Management	SupervisedEntity	State Management shall use this interface to enable supervision of its process(es) by Platform Health Management.

Table 5.2: Interfaces required from other Functional Clusters

6 Requirements Tracing

The following tables reference the requirements specified in [3] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[RS_AP_00115]	Public namespaces	[SWS_SM_91017] [SWS_SM_91028]
[RS_AP_00119]	Return values / application errors	[SWS_SM_91010] [SWS_SM_91017] [SWS_SM_91028]
[RS_AP_00120]	Method and Function names	[SWS_SM_91017] [SWS_SM_91028]
[RS_AP_00121]	Parameter names	[SWS_SM_91017] [SWS_SM_91028]
[RS_AP_00122]	Type names	[SWS_SM_91018] [SWS_SM_91019] [SWS_SM_91020]
[RS_AP_00125]	Enumerator and constant names	[SWS_SM_91010]
[RS_AP_00142]	Handling of unsuccessful operations	[SWS_SM_91010] [SWS_SM_91017] [SWS_SM_91028]
[RS_AP_00149]	Error handling for non-initialized Functional Cluster	[SWS_SM_91010]
[RS_AP_00150]	Provide only interfaces that are intended to be used by AUTOSAR Applications and Functional Clusters	[SWS_SM_91010] [SWS_SM_91016] [SWS_SM_91017] [SWS_SM_91018] [SWS_SM_91019] [SWS_SM_91020] [SWS_SM_91021] [SWS_SM_91023] [SWS_SM_91024] [SWS_SM_91028]
[RS_SM_00001]	State Management shall coordinate and control multiple sets of Applications.	[SWS_SM_00203] [SWS_SM_00210] [SWS_SM_00400] [SWS_SM_00401] [SWS_SM_00600] [SWS_SM_00601] [SWS_SM_00602] [SWS_SM_00603] [SWS_SM_00604] [SWS_SM_00605] [SWS_SM_00606] [SWS_SM_00607] [SWS_SM_00608] [SWS_SM_00609] [SWS_SM_00610] [SWS_SM_00611] [SWS_SM_00612] [SWS_SM_00613] [SWS_SM_00614] [SWS_SM_00615] [SWS_SM_00616] [SWS_SM_00617] [SWS_SM_00618] [SWS_SM_00619] [SWS_SM_00620] [SWS_SM_00621] [SWS_SM_00622] [SWS_SM_00623] [SWS_SM_00624] [SWS_SM_00625] [SWS_SM_00626] [SWS_SM_00627] [SWS_SM_00628] [SWS_SM_00629] [SWS_SM_00630] [SWS_SM_00631] [SWS_SM_00633] [SWS_SM_00634] [SWS_SM_00635] [SWS_SM_00636] [SWS_SM_00638] [SWS_SM_00639] [SWS_SM_00640] [SWS_SM_00642] [SWS_SM_00643] [SWS_SM_00644] [SWS_SM_00645] [SWS_SM_00646] [SWS_SM_00647] [SWS_SM_00648] [SWS_SM_00649] [SWS_SM_00650] [SWS_SM_00651] [SWS_SM_00654] [SWS_SM_00655] [SWS_SM_00656] [SWS_SM_00657] [SWS_SM_00658] [SWS_SM_00659] [SWS_SM_00660] [SWS_SM_00661] [SWS_SM_00662] [SWS_SM_00663] [SWS_SM_00664] [SWS_SM_00665] [SWS_SM_91016] [SWS_SM_91017] [SWS_SM_91021] [SWS_SM_91022] [SWS_SM_91023] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_SM_91024] [SWS_SM_91025] [SWS_SM_91026] [SWS_SM_91027] [SWS_SM_91028] [SWS_SM_91100] [SWS_SM_91101] [SWS_SM_91102] [SWS_SM_91103] [SWS_SM_91104] [SWS_SM_91105] [SWS_SM_91106] [SWS_SM_91107] [SWS_SM_91108] [SWS_SM_91109]
[RS_SM_00004]	State Management shall provide standardized interfaces.	[SWS_SM_00202] [SWS_SM_00204] [SWS_SM_00205] [SWS_SM_00206] [SWS_SM_00207] [SWS_SM_00208] [SWS_SM_00209] [SWS_SM_00211] [SWS_SM_00212] [SWS_SM_00213] [SWS_SM_91010] [SWS_SM_91016] [SWS_SM_91017] [SWS_SM_91018] [SWS_SM_91019] [SWS_SM_91020] [SWS_SM_91021] [SWS_SM_91022] [SWS_SM_91023] [SWS_SM_91024] [SWS_SM_91025] [SWS_SM_91026] [SWS_SM_91027] [SWS_SM_91028] [SWS_SM_91100] [SWS_SM_91101] [SWS_SM_91102] [SWS_SM_91103] [SWS_SM_91104] [SWS_SM_91105] [SWS_SM_91106] [SWS_SM_91107] [SWS_SM_91108] [SWS_SM_91109]
[RS_SM_00005]	State Management internal states.	[SWS_SM_00203] [SWS_SM_00210] [SWS_SM_00600] [SWS_SM_00601] [SWS_SM_00602] [SWS_SM_00603] [SWS_SM_00604] [SWS_SM_00605] [SWS_SM_00606] [SWS_SM_00607] [SWS_SM_00608] [SWS_SM_00609] [SWS_SM_00610] [SWS_SM_00611] [SWS_SM_00612] [SWS_SM_00613] [SWS_SM_00614] [SWS_SM_00615] [SWS_SM_00616] [SWS_SM_00617] [SWS_SM_00618] [SWS_SM_00619] [SWS_SM_00620] [SWS_SM_00621] [SWS_SM_00622] [SWS_SM_00623] [SWS_SM_00624] [SWS_SM_00625] [SWS_SM_00626] [SWS_SM_00627] [SWS_SM_00628] [SWS_SM_00629] [SWS_SM_00630] [SWS_SM_00631] [SWS_SM_00633] [SWS_SM_00634] [SWS_SM_00635] [SWS_SM_00636] [SWS_SM_00638] [SWS_SM_00639] [SWS_SM_00640] [SWS_SM_00642] [SWS_SM_00643] [SWS_SM_00644] [SWS_SM_00645] [SWS_SM_00646] [SWS_SM_00647] [SWS_SM_00648] [SWS_SM_00649] [SWS_SM_00650] [SWS_SM_00651] [SWS_SM_00654] [SWS_SM_00655] [SWS_SM_00656] [SWS_SM_00657] [SWS_SM_00658] [SWS_SM_00659] [SWS_SM_00660] [SWS_SM_00661] [SWS_SM_00662] [SWS_SM_00663] [SWS_SM_00664] [SWS_SM_00665]
[RS_SM_00401]	State Management shall control Applications depending on dynamic communication paths .	[SWS_SM_00620] [SWS_SM_00621] [SWS_SM_00625] [SWS_SM_00626]

▽

△

Requirement	Description	Satisfied by
[RS_SM_00601]	State Management shall coordinate recovery actions.	[SWS_SM_00030] [SWS_SM_00031] [SWS_SM_00666]

Table 6.1: Requirements Tracing

7 Functional specification

State Management is a functional cluster contained in the Adaptive Platform Services. State Management is responsible for handling of incoming events, prioritization of these events/requests setting the corresponding internal States. State Management may consist of one or more state machines, which might be more or less loosely coupled depending on project needs.

Additionally the State Management takes care of not shutting down the system as long as an update session is active as part of State Managements internal State.

In dependency of the current internal States, State Management might decide to request Function Groups or Machine State to enter specific state by using interfaces of Execution Management.

State Management is responsible for en- and disabling (partial) networks by means of Network Management. State Management can influence Network Management's NetworkHandle in dependency of Function Groups states and - vice versa - can set Function Groups to a defined state depending on the value of Network Managements NetworkHandle.

This chapter describes the functional behavior of State Management and the relation to other AUTOSAR Adaptive Platform Applications State Management interacts with.

- Section 7.1 covers the core State Management run-time responsibilities including the start of Adaptive Applications.
- Section 7.3 describes how Update and Configuration Management interacts with State Management
- Section 7.4 documents support provided by Network Management to de-/activate (partial) networks in dependency of Function Group States and vice versa.
- Section 7.5 describes how Execution Management is used to change Function Group State or Machine State.

7.1 State Management Responsibilities

State Management is the functional cluster which is responsible for determining the current internal States, and for initiating Function Group and Machine State transitions by requesting them from Execution Management.

If an State Managements internal State change is triggered then Execution Management may be requested to set Function Groups or Machine State into new Function Group State.

The state change request for Function Groups can be issued by several entities, such as:

- Platform Health Management reports supervision errors that trigger error recovery, e.g. to activate fallback Functionality.
- Update and Configuration Management to switch the system into states where software or configuration can be updated and updates can be verified.
- Network Management to coordinate required functionality and network state. This is no active request by Network Management. Network Management provides several sets of NetworkHandles, where State Management registers to and reacts on changes of them.

The final decision if any effect is performed is taken by State Managements internal logic based on project-specific requirements or based on configuration in case of using StateMachine approach.

Adaptive Applications may provide their own property or event via an ara com interface, where the State Management is subscribing to, to trigger State Management internal events. Since State Management functionality is critical, access from other Adaptive Applications must be secured, e.g. by Identity and Access Management.

- State Management shall be monitored and supervised by Platform Health Management.
- State Management provides ara::com fields as interface to provide information about its current internal States

State Management is responsible for handling the following states:

- Machine State see Section 7.1.1
- Function Group State see Section 7.1.2
- NetworkHandle state see Section 7.4

7.1.1 Machine State

A Machine State is a specific type of Function Group State (see Section 7.1.2). Machine States and all other Function Group States are determined and requested by the State Management functional cluster, see Section 7.1.3. The set of active States is significantly influenced by vehiclewide events and modes which are evaluated into State Managements internal States. A Machine State is a specific type of Function Group State (see Section 7.1.2). Machine States and all other Function Group States are determined and requested by the State Management functional cluster, see Section 7.1.3. The set of active States is significantly influenced by vehiclewide events and modes which are evaluated into State Managements internal States.

The Function Group States, including the Machine State, define the current set of running Modelled Processes. Each Adaptive Application can declare in its Execution Manifests in which Function Group States its Modelled Processes have to be running.

The start-up sequence from initial state Startup to the point where State Management, requests the initial running machine state Driving is illustrated in Figure 7.1 as an example Driving Function Group State is no mandatory Function Group State.

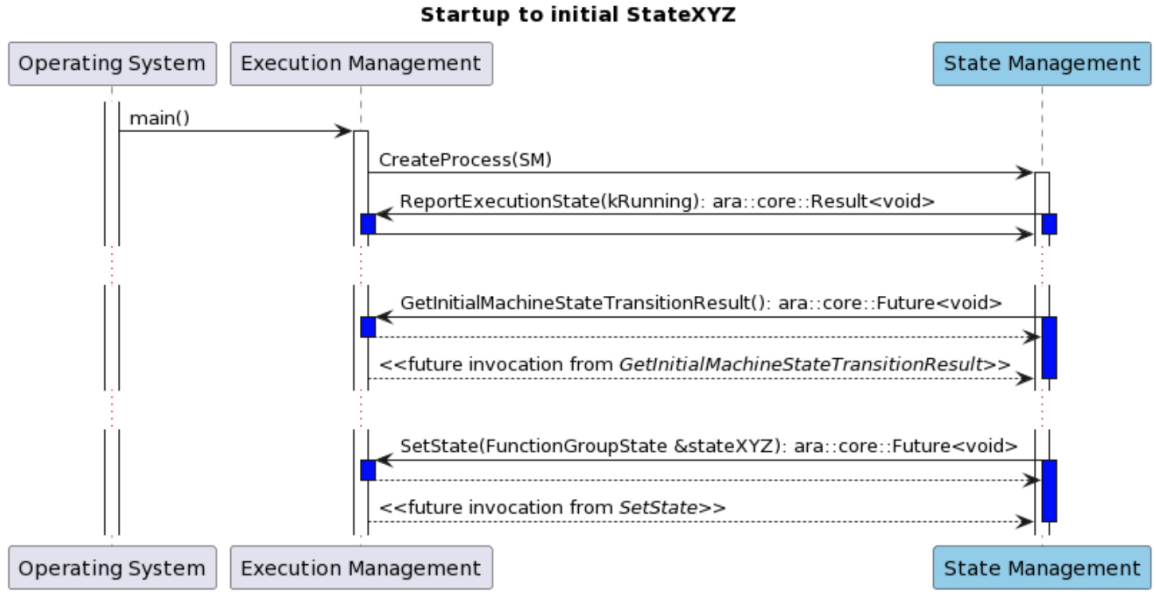

Figure 7.1: Start-up Sequence – from Startup to initial running state Driving

An arbitrary state change sequence to machine state StateXYZ is illustrated in Figure 7.2. Here, on receipt of the state change request, Execution Management terminates running Modelled Processes and then starts Modelled Processes active in the new state before confirming the state change to State Management.

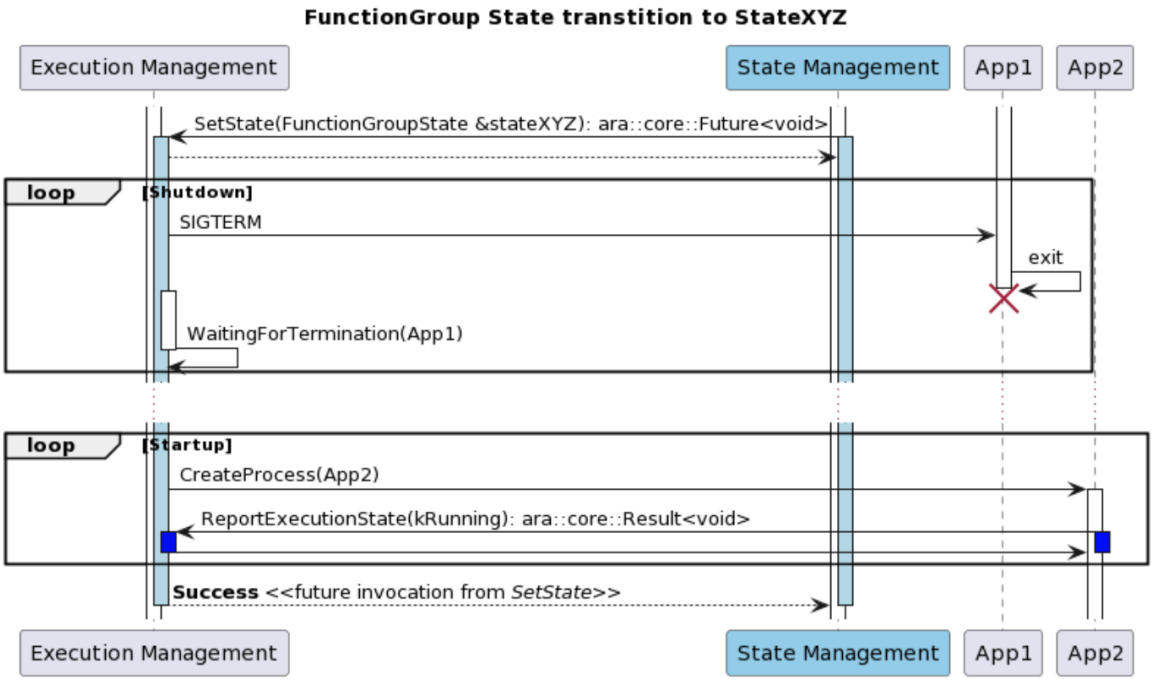

Figure 7.2: State Change Sequence – Transition to machine state `StateXYZ`

7.1.2 Function Group State

If more than one group of functionally coherent `Adaptive Applications` is installed on the same machine, the `Machine State` mechanism is not flexible enough to control these functional clusters individually, in particular if they have to be started and terminated with interleaving lifecycles. Many different `Machine States` would be required in this case to cover all possible combinations of active functional clusters.

To support this use case, additional `Function Groups` and `Function Group States` can be configured. Other use cases where starting and terminating individual groups of `Modelled Processes` might be necessary including error recovery.

In general, `Machine States` are used to control `Machine` lifecycle (startup/shutdown/restart) and `Modelled Processes` of platform level Applications while other `Function Group States` individually control `Modelled Processes` which belong to groups of functionally coherent user level `Adaptive Applications`.

`Modelled Processes` reference in their `Execution Manifest` the states in which they want to be executed. A state can be any `Function Group State`, including a `Machine State`. For details see [13], especially "Mode-dependent Startup Configuration" chapter and "Function Groups" chapter.

The arbitrary state change sequence as shown in Figure 7.2 applies to state changes of any `Function Group` - just replace `"MachineState"` by the name of the `Function Group`. On receipt of the state change request, `Execution Management` terminates not longer needed `Modelled Processes` and then starts `Modelled Processes`

active in the new Function Group State before confirming the state change to State Management.

From the point of view of Execution Management, Function Groups are independent entities that doesn't influence each other. However from the point of view of State Management this may not always be the true. Let's consider a simple use case of Machine shutdown. From the point of view of Execution Management State Management (at some point in time) will request a Machine State transition to Shutdown state. One of the Modelled Processes configured to run in that particular state, will initiate OS / HW shutdown and the Machine will power off. However from the point of view of State Management you will need to asses, if it's valid to request a Machine State transition to Shutdown state. Even if the assessment was positive and the Machine can be powered off, project specific requirements may mandate to switch all available Function Groups to Off state before we start power off sequence. For this reason we are considering existence of dependencies between Function Groups. Please note that currently those dependencies are implementation specific and configurable by integrator (i.e. all Function Groups are independent unless integrator change this).

7.1.3 State Management Architecture

State Management is the functional cluster which is responsible for determining the current set of active Function Group States, including the Machine State, and for initiating State transitions by requesting them from Execution Management. Execution Management performs the State transitions and controls the actual set of running Modelled Processes, depending on the current States.

State Management is the central point where new Function Group States can be requested and where the requests are arbitrated, including coordination of contradicting requests from different sources. Additional data and events might need to be considered for arbitration.

State Management functionality is highly project specific, and AUTOSAR decided against specifying functionality like the Classic Platform BswM for the Adaptive Platform. It is planned to only specify a set of basic service interfaces, and to encapsulate the actual arbitration logic into project specific code (e.g. a library), which can be plugged into the State Management framework and has standardized interfaces between framework and arbitration logic, so the code can be reused on different platforms.

The arbitration logic code might be individually developed or (partly) generated, based on standardized configuration parameters.

There are currently two architectural approaches within State Management:

- Only the interfaces are defined. As State Management functionality is highly project specific, the actual project specific arbitration logic code could be encapsulated within e.g. a library, which can be plugged into the State Management

framework, thus it has standardized interfaces between framework and arbitration logic, so the code can be reused on different platforms.

The arbitration logic code might be individually developed or (partly) generated, based on standardized configuration parameters.

- Additionally a `StateMachine` approach is defined, thus project specific arbitration logic code can be implemented in `SMControlApplication`, which will request configured transitions. `SMControlApplication` does not have to care about concrete `Function Group States`, related `NmNetworkHandle` settings and recovery actions.

An overview of the interaction of `State Management` with `AUTOSAR Adaptive Platforms` is shown in Figure 5.

7.2 Interaction with Platform Health Management

`Platform Health Management` is responsible for monitoring supervised entities via local supervision(s). Failures in local supervision(s) will be accumulated in a global supervision. The scope of a global supervision is a single `Function Group` (or a part of it). For details see SWS-PlatformHealthManagement [6]. As soon as a global supervision enters the stopped state, `Platform Health Management` will notify `State Management` via C++ API provided by Platform Health Manager. C++ interface is provided as a class with virtual functions, which have to be implemented by `State Management`.

When `State Management` receives notification from `Platform Health Management` it can evaluate the information from the notification and initiate the project-specific actions to recover from the failure (e.g. request `Execution Management` to switch a `Function Group` to another `Function Group State`, request `Execution Management` for a restart of the Machine, ...). Via the response value to RecoveryHandler() `State Management` can indicate to `Platform Health Management` whether the recovery can be handled in a controlled manner or it can request `Platform Health Management` to fire a watchdog reaction as a last resort.

[SWS_SM_00030] RecoveryHandler can not be handled

Upstream requirements: RS_SM_00601

⌈`State Management` shall return kSMCanNotHandleRecovery when the parameters provided in RecoveryHandler() are invalid (e.g. unknown FunctionGroup).⌋

When `State Management` performs project-specific recovery actions it might happen, that due to the performed recovery actions a new issue is reported by `Platform Health Management`. This is called "nested recovery".

[SWS_SM_00031] Nested recovery handling

Upstream requirements: RS_SM_00601

⌈In case of a nested recovery State Management shall return from all RecoveryHandler() calls only when the latest recovery action issued by the last RecoveryHandler() call is finished without any issue.⌋

Note: If it is a "nested recovery" is very project specific and therefore an implementation detail. For StateMachine approach this detail is explained in chapter 7.6.5.

Note: Platform Health Management monitors the return of the RecoverHandler() with a configurable timeout. If State Management gives no response to RecoveryHandler() Platform Health Management will do its own countermeasures by wrongly triggering or stop triggering the serviced watchdog.

If State Management is used in Safety Critical Platform, then it is suggested to use Alive/Logical/Deadline supervision(s) and report their checkpoints appropriately to Platform Health Management.

How issue notification from Platform Health Management towards State Management is handled when using StateMachine approach is shown in section 7.6.5.

7.3 Interaction with Update and Configuration Management

Update and Configuration Management is responsible for installing, removing or updating Software Clusters as smallest updatable entity. To enable Update and Configuration Management to fulfill its functionality State Management offers a service interface (see Section 9.2.1) to be used by Update and Configuration Management.

Please note that system integrator has to limit usage of this interface to Update and Configuration Management by configuring Identity and Access Management.

In a first step Update and Configuration Management will ask State Management if it is allowed to perform an update. The decision will depend on current state of the machine (or whole vehicle) and has to be done in a project specific way.

[SWS_SM_00203] Start update session

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈State Management shall provide the service interface UpdateRequest to Update and Configuration Management with the method call RequestUpdateSession to check if an update can be performed.⌋

[SWS_SM_00210] Active update session

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈The period between accepting an update session [SWS_SM_00631] and ending an update session [SWS_SM_00646] is considered by State Management as "active update session".⌋

As soon as State Management allows updating, it is necessary that State Management denies any further request for a new update session. To assure a higher consistency in the AUTOSAR Adaptive Platform, multiple update sessions at a time shall be not allowed.

For the StateMachine approach a separate interface UpdateAllowed is provided to check if an update is allowed or not. This interface was introduced, because the limited logic of the StateMachine is not able to decide if an update is allowed. Therefore this decision is delegated to a customer specific application e.g. SMControlApplication.

[SWS_SM_00209] Preventing multiple update sessions

Upstream requirements: RS_SM_00004

⌈RequestUpdateSession shall return kNotAllowedMultipleUpdateSessions in case the method RequestUpdateSession is called during an already active Update Session⌋

As soon as State Management allows updating, it is necessary that State Management prevents system from shutting down.

However AUTOSAR fully recognizes that there could be valid reasons to restart/shutdown machine even during an active update session (e.g. low voltage, high temperature,...). For that reasons AUTOSAR does not prevent State Management from restarting/shutting down machine, but advises that such a decision should be carefully evaluated before being executed. Please note that AUTOSAR also recognizes that projects could have an arbitrary timeout restriction on the duration of the update session. This could be done for practical reasons and is allowed from the perspective of the AUTOSAR.

Additionally State Management has to persist the information about an ongoing update session, thus, after a machine restart (independently if restart was expected or not), Update and Configuration Management can continue to update. To continue the update in a consistent way it will be needed that only a few Function Groups will be set to a meaningful Function Group State (project specific). At least Update and Configuration Management has to be in a running state.

[SWS_SM_00204] Persist session status

Upstream requirements: RS_SM_00004

⌈State Management shall persist information about ongoing update session, thus it can be read out after any kind of Machine reset.⌋

[SWS_SM_00213] UpdateRequest method call rejection

Upstream requirements: RS_SM_00004

⌈A call to PrepareUpdate, VerifyUpdate, PrepareRollback or StopUpdateSession shall return kOperationRejected if invoked outside an active update session.⌋

Please note that RequestUpdateSession is not in the list of the rejected method calls ([SWS_SM_00213]), because it is the call which starts the active update session.

In some cases it is needed that Update and Configuration Management issues a reset of the Machine (expected reset), e.g. when Functional Clusters like State Management, Platform Health Management or Execution Management are affected by the update. This has to be supported by State Management. At least this might be simply implemented by requesting Machine State restart from Execution Management.

[SWS_SM_00202] Reset Execution

Upstream requirements: RS_SM_00004

⌈State Management shall implement the service interface UpdateRequest to Update and Configuration Management with the method call ResetMachine to request a Machine reset.⌋

Update and Configuration Management has to inform State Management when no more operations for the update have to be done, thus State Management can clear now the information about an ongoing update and can continue its regular job. Please note, that all State Management activities after the StopUpdateSession is requested are fully project specific, like setting the impacted Function Groups into a meaningful Function Group State.

[SWS_SM_00205] Stop update session

Upstream requirements: RS_SM_00004

⌈State Management shall provide the service interface UpdateRequest to Update and Configuration Management with the method call StopUpdateSession thus it can inform State Management that the update session is finished.⌋

During the update there will be up to three different steps, depending if a Software Cluster is installed, removed or updated. If and when the steps are done depends additionally on the success or fail of the previous steps. To support Update and Configuration Management to request these steps State Management provides three different methods as part of the service interface UpdateRequest.

[SWS_SM_00206] prepare update

Upstream requirements: RS_SM_00004

⌈State Management shall provide the service interface UpdateRequest to Update and Configuration Management with the method call PrepareUpdate thus it can request State Management to perform a preparation of the given Function Groups to be updated.⌋

[SWS_SM_00207] prepare verify

Upstream requirements: RS_SM_00004

⌈State Management shall provide the service interface UpdateRequest to Update and Configuration Management with the method call VerifyUpdate thus it can request State Management to perform a verification of the given Function Groups.⌋

[SWS_SM_00208] prepare rollback

Upstream requirements: RS_SM_00004

⌈State Management shall provide the service interface UpdateRequest to Update and Configuration Management with the method call PrepareRollback thus it can request State Management to perform a preparation of the given Function Groups to be rolled back.⌋

For updating a Software Cluster Update and Configuration Management will call the method PrepareUpdate (as part of the service interface UpdateRequest) in a first step. State Management will at least set all the Function Groups, given as parameter, to Off state. In next step Update and Configuration Management will perform the real update (e.g. exchange executable, change manifests,...). As following step Update and Configuration Management uses the VerifyUpdate to request State Management to perform a verification of the update. Therefore State Management will at least set all the Function Groups, given as parameter, to Verify state. These request will be reported to Update and Configuration Management as failed when any of the Function Groups could not be set to the requested Function Group State. A failure will also be reported when one of these functions is called, before State Management granted the right to update.

Once the ResetMachine call is processed, the Machine will be restarted. This means Machine will go through a startup sequence and it will need to restore its

own state. For State Management this means a transition to the ContinueUpdate state for the StateMachine of type Controller. However, the Update and Configuration Management and the State Management need to synchronize again. For this reason the result of the ResetMachine request is connected to a notification mechanism, which can be traced by the Update and Configuration Management.

[SWS_SM_00211] ResetMachine notification

Upstream requirements: RS_SM_00004

⌈State Management shall provide the service interface UpdateRequest to Update and Configuration Management with the field ResetMachineNotifier thus it can trace the status from the ResetMachine call during and after it is performed.⌋

[SWS_SM_00212] Default value for ResetMachineNotifier

Upstream requirements: RS_SM_00004

⌈The default value for the field ResetMachineNotifier shall be kIdle.⌋

When any of these steps fails, Update and Configuration Management can decide to revert previous changes. Therefore Update and Configuration Management uses PrepareRollback function, where State Management will at least set all the Function Groups, given as parameter, to Off state.

For more detail about the update process see sequence diagrams and descriptions in [10].

How interaction between Update and Configuration Management and State Management is handled when using StateMachine approach is shown in section 7.6.12.

7.4 Interaction with Network Management

To be portable between different ECUs the Adaptive Applications should not have the need to know which networks are needed to fulfill its functionality, because on different ECUs the networks could be configured differently. To control the availability of networks for several Adaptive Applications State Management interacts with Network Management via a C++ API.

Network Management provides multiple instances of NetworkHandles, where each represents a set of (partial) networks.

To fulfill the project-specific needs StateMachine might set NmNetworkHandle states depending on Function Group States and vice versa.

How interaction between Network Management and State Management is handled when using StateMachine approach is shown in section 7.6.10 and in section 7.6.5.

7.5 Interaction with Execution Management

Execution Management is used to execute the Function Group State changes. The decision to change the state of Machine State or the Function Group State of Function Groups might come from inside of State Management based on State Management States (or other project specific requirements) or might be requested at State Management from an external Adaptive Application.

An overview of the interaction of State Management, Execution Management and Adaptive Applications is shown in Figure 7.3.

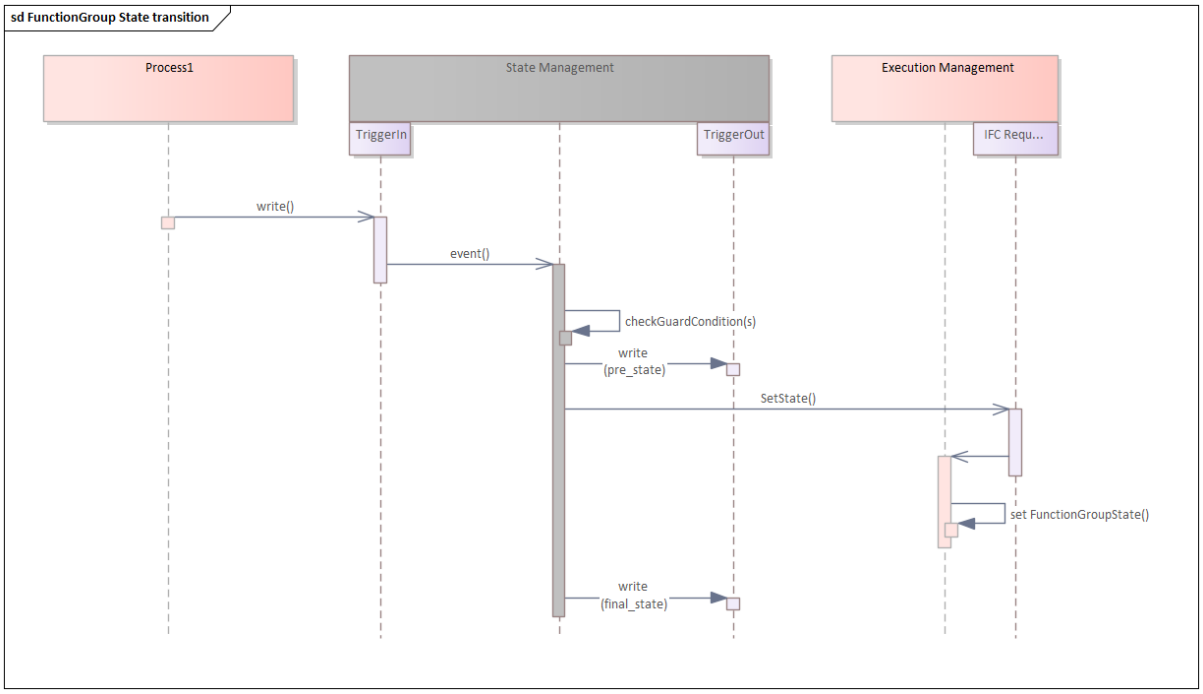

Figure 7.3: Switching FunctionGroup State by "Trigger"

[SWS_SM_00400] Execution Management

Upstream requirements: RS_SM_00001

⌈State Management shall use StateClient API of Execution Management to request a change in the Function Group State of any Function Group (including Machine State).⌋

Execution Management might not be able to carry out the requested Function Group State change due to several reasons (e.g. corrupted binary). Execution Management returns the result of the request.

When State Management gets kIntegrityOrAuthenticityCheckFailed as error to a Function Group SetState request it is expected that every subsequent request for the same Function Group State will fail with the same value. So any further action to solve this issue (e.g. update/fix application) is out of scope of State Management. Please note that this error indicates that the trusted platform has been compromised.

[SWS_SM_00401] Execution Management Results

Upstream requirements: RS_SM_00001

⌈State Management shall evaluate the results of request to Execution Management. Based on the results State Management may do project-specific actions⌋

Depending on ExecErrc returned by Execution Management during Function Group State transition, State Management can perform variety of countermeasures which include but are not limited to following actions

- request another Function Group State for the same Function Group e.g. set current Function Group to "Off" state
- request a Function Group State for another Function Group
- persist the error information (at least for current power cycle) to not request the Function Group State again, when it is an unrecoverable error e.g. kMetaModelError, kIntegrityOrAuthenticityCheckFailed
- trigger a system restart (e.g. report wrong supervision checkpoint to PHM, project specific) in case it is a generic unrecoverable error e.g. kGeneralError, kCommunicationError

Please note that these error reactions are only valid when State Management is individually implemented. When StateMachine approach is used, a change in the StateMachine state should be configured as error reaction.

Implementation hint: State Management needs to take into account that supervision failures may be reported by Platform Health Management before Execution Management has reported that a requested Function Group State has been reached.

7.6 StateManagement StateMachine

7.6.1 StateMachine introduction

Introducing StateMachines in the scope of State Management will give the integrator the possibility to define which set of Function Groups become active (Function Group State != "Off") under a certain condition. The integrator can define error reactions (violated supervisions, abnormal or unexpected termination) via configuration in the scope of a set of Function Group States, reflected by a StateMachine State of State Management.

StateMachines are comprised by set of StateMachine States. Each StateMachine has to have at least five StateMachine States: The Initial State, Off, PrepareUpdate, VerifyUpdate and PrepareRollback. There probably will be a number of additional project-specific StateMachine States (e.g. degraded States). Each State references an ActionList, which is comprised of a set of ActionListItems. All ActionListItems in an ActionList are executed as soon as a StateMachine State of a StateMachine is entered. Currently available Types for an ActionListItem are:

- Request Function Group State, (represented by meta-class StateManagementSetFunctionGroupStateActionItem)
- SYNC, (represented by meta-class StateManagementSyncActionItem)
- Start/Stop StateMachine, (represented by meta-class StateManagementStateMachineActionItem)
- Sleep (represented by meta-class StateManagementSleepActionItem) to delay processing the next ActionListItems
- SetNetworkHandle switches the provided NetworkHandle to the configured state(NoCom or FullCom, see NmStateRequestEnum) (represented by meta-class StateManagementNmActionItem)

A StateMachine State change can be triggered by several different types of actors:

- An Adaptive Application (called SMControlApplication) can request StateMachine State change through publicly available interface. Please note that IAM configuration may be applied here.
- Platform Health Management and Execution Management can trigger state change as a result of an error.
- Network Management can trigger state change as a result of change in a NmNetworkHandle.
- Update and Configuration Management can trigger state change temporary caused by processing an update.

if configured, the current StateMachine State will be published on the dedicated StateMachineNotification interface.

The following figure shows how Platform Health Management, Execution Management, Network Management, Update and Configuration Management, SMControlApplication and a StateMachine as part of State Management interact:

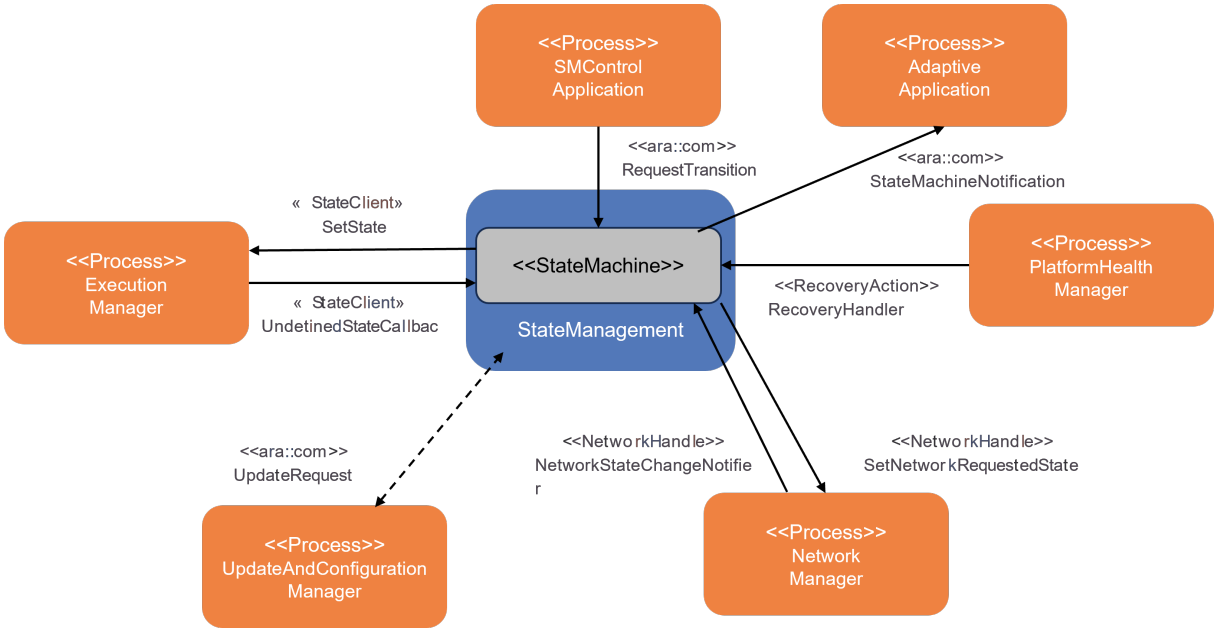

Figure 7.4: Interactions with StateMachine

StateMachines are an optional element of State Management. However, the integrator can decide to implement State Management fully by its own. This is achieved by keeping interfaces towards State Management public.

7.6.2 Controlling application for StateMachine States

As State Management shall not contain any project-specific logic (under which condition a StateMachine State is requested) it is assumed that a project-specific Process (SMControlApplication) exists. As SMControlApplication and StateMachine within State Management instance belong together it would make sense to instantiate them somehow together like follows:

- The Process is configured to run in the same Function Group State like the Process which contains the StateMachine.

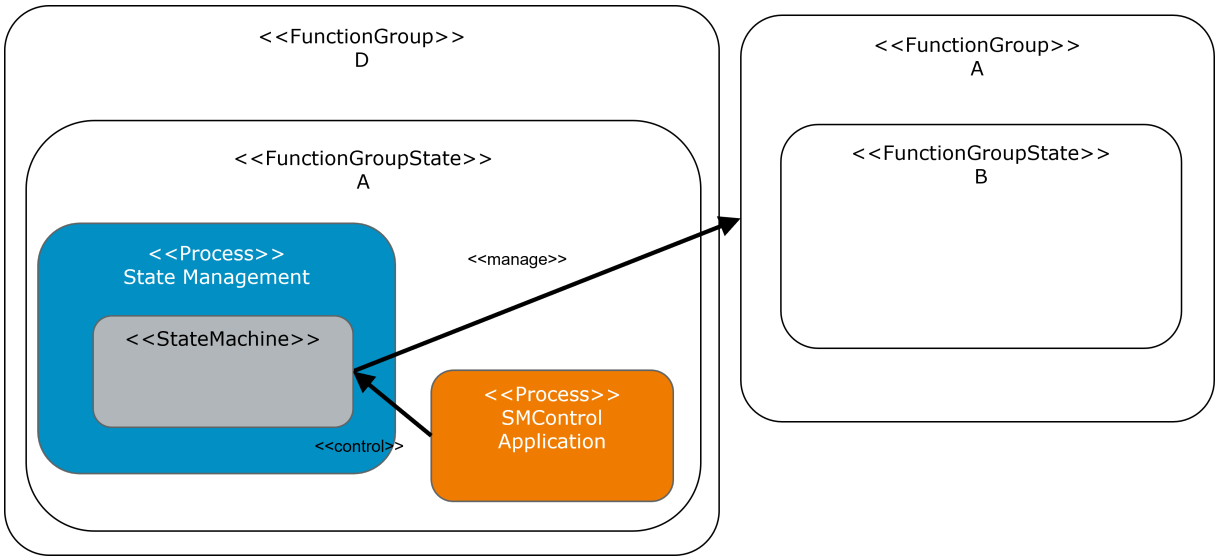

Figure 7.5: SMControlApplication and StateManagement Process started together

- The `Process` is configured to run in a `Function Group State`, as `ActionListItem` in the `ActionList` referenced by the `Initial State` of the `StateMachine`.

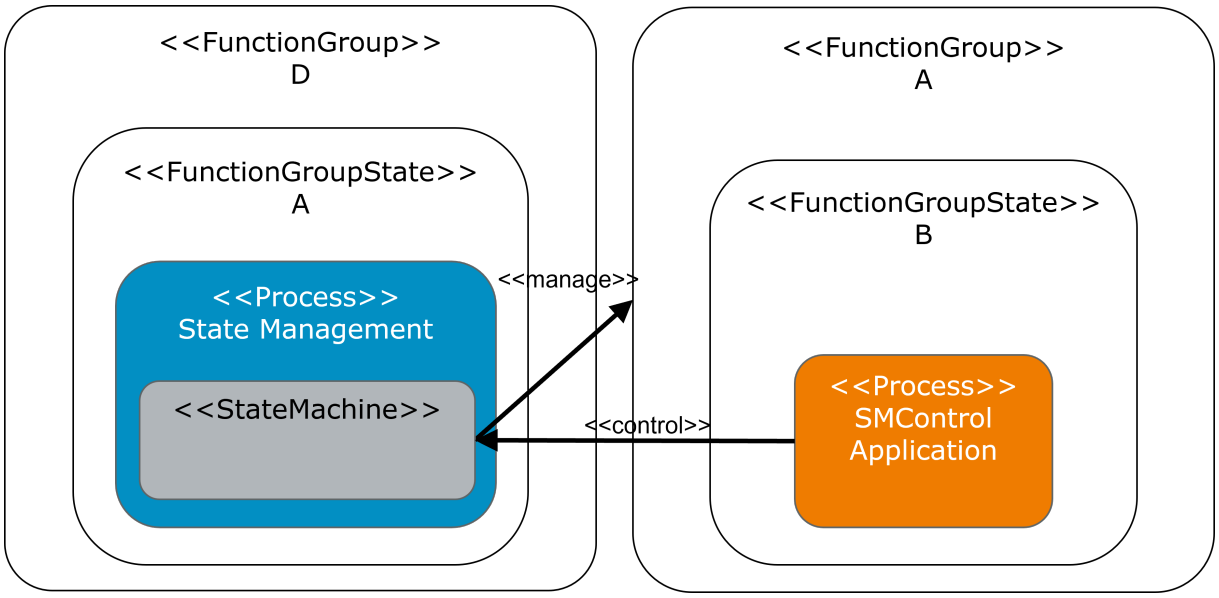

Figure 7.6: SMControlApplication started in initial State of StateManagements StateMachine

Even if it would make sense to start these `Processs` as shown above, they could be part of different, decoupled `Function Group States`, depending on project needs.

`SMControlApplication` is needed when arbitrary state changes could be requested as per `StateMachine` configuration. If the only functionality provided by `StateMachine` is the reaction to errors reported by `Platform Health Management` and/or `Execution Management`, or reaction to changes in `NetworkHandles`,

then there is no need to have a SMControlApplication. In that case, StateMachine should start intended functionality when it enters the Initial State.

[SWS_SM_CONSTR_00010] ActionItems in initial StateMachine State ⌈When there is no SMControlApplication at least one ActionListItem in the ActionList, referencing the Initial State of the StateMachine, shall reference a Function Group State different than "Off" or a Start StateMachine ActionListItem.⌋

The SMControlApplication, uses the RequestTransition method of StateMachineService(modelled as meta-class ServiceInterface) to request another StateMachine State. As not all transitions might be possible(project-specific) a mapping table (TransitionRequestTable) is introduced which maps the input value provided by SMControlApplication to StateMachines next state, depending on current StateMachine State.

Transition Request	Current State	Next State
1001	Off	On
1000	On	Off
1002	Recovery	Off
1001	Startup	Off
1000	Suspend	On
1000	Recovery	Off

Figure 7.7: TransitionRequestTable

Please note that appendix A.10 of TPS Manifest Specification [13] shows in detail how the TransitionRequestTable and the ErrorRecoveryTable can be build with the available meta-class elements.

[SWS_SM_00600] StateMachineService interface

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈State Management shall provide the ara::com based service StateMachineService for each instance of the StateMachine configured.⌋

[SWS_SM_00665] StateMachineNotification service interface

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈If configured StateMachineNotification State Management shall instantiate StateMachineNotification interface for that StateMachine⌋

Please note that the StateMachineNotification service interface mostly interacts with Adaptive Applications. Therefore it may be possible, in a project specific context, that some StateMachines are not relevant for the application layer, and therefore there is no need to force the creation and offering of their respective StateMachineNotification service interfaces.

[SWS_SM_00618] StateMachine service interfaces - Offer

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Each configured ara::com based service (StateMachineService, StateMachineNotification) for the StateMachine to be started shall be available (offered) when the ActionListItem "StartStateMachine" is processed successfully.⌋

Please note that see [SWS_SM_00618] allows the SMControlApplication the possibility to request a new StateMachine State transition immediately after the successful StateMachine creation, even if the StateMachine is processing the initial ActionList.

[SWS_SM_00619] StateMachine service interfaces - StopOffer

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Each configured ara::com based service (StateMachineService, StateMachineNotification) for a StateMachine shall be no longer available (offered) at the time when processing of ActionListItem "StopStateMachine" is finished.⌋

7.6.3 StateMachine design considerations

Even if it is possible to manage all Function Groups within a single StateMachine, it makes sense to control Function Group States of a sub-set of Function Groups in separate StateMachine instances. This design decision is heavily project-specific and depends e.g. on the number of installed Software Clusters, amount of Function Groups and their Function Group States. With an increasing number of these items and the needed combinations (project-specific), the number of states within a single StateMachine might become very hard to manage. For this reason State Management supports multiple StateMachine instances: As soon as any StateMachine is configured exactly one StateMachine has to have the role of a Controller. All other - optionally - configured StateMachines have to have the role of an Agent see StateManagementStateNotification.stateMachine.category.

[SWS_SM_CONSTR_00031] Existence of StateMachine of type Controller ⌈As soon as any StateMachine is configured in a Machine exactly one StateMachine has to have the role of a Controller, at the time when the creation of the manifest is finished.⌋

The Controller is the StateMachine, which is automatically started, when State Management starts. It is in the responsibility of Controller to manage the life-cyle of

- the whole Machine
- StateMachine Agents(if configured)

StateMachine of type Controller is responsible for starting StateMachine instances of type (Agent). Therefore the StateMachine of type Controller is the first StateMachine which has to be started in State Management Process.

[SWS_SM_00648] StateMachine of type Controller start

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When Modelled Process controlling StateMachine of type Controller starts it shall start StateMachine of type Controller.⌋

As Controller is managing the life-cycle of the Machine it has to reference Machine State ("MachineFG").

[SWS_SM_CONSTR_00017] ActionListItem "Function Group State" in ActionLists of StateMachine in the Controller ⌈All ActionLists, referencing states of the Controller StateMachine shall contain ActionListItem "Function Group State" for MachineFG.⌋

To be able to control life-cycle of the Machine in a consistent way no other StateMachine than the Controller is able to manage states of MachineFG. This is covered by [SWS_SM_CONSTR_00017] and [SWS_SM_CONSTR_00013].

Please note that the shutdown/ restart of the Machine is achieved by MachineFG Shutdown, respectively Restart state. Therefore it is recommended to configure states for the Controller, where the referencing ActionList references MachineFG Shutdown or Restart state.

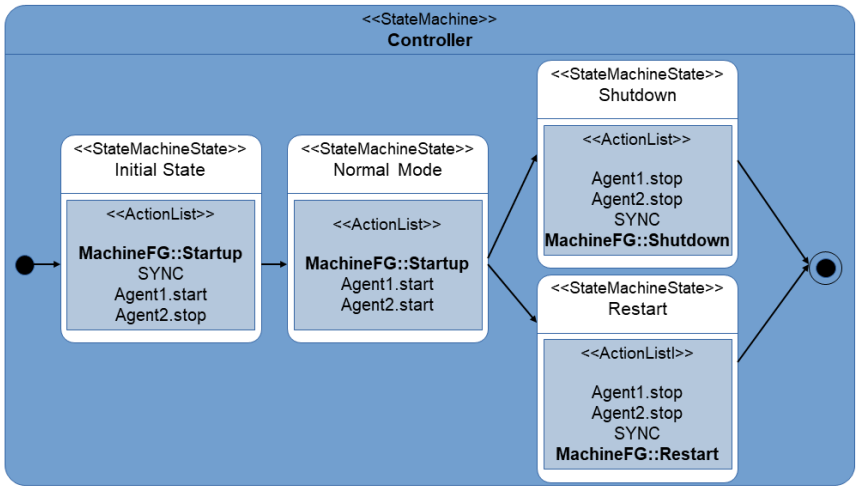

Figure 7.8: Example for Controller StateMachineStates with MachineFG

To support update ability of StateMachines it is needed, that the Function Groups, which are provided in the update steps, do not interfere with Function Groups, which are not affected by the update. As a Software Cluster is the scope of an update, Update and Configuration Management will provide the list of claimedFunctionGroups of the Software Cluster to be updated. Therefore it is needed that Agent do not manage Function Groups which are claimed by different Software Clusters.

[SWS_SM_CONSTR_00018] Limitations of managed FunctionGroups ⌈ StateMachines in the role Agent shall only manage Function Groups from the same set of claimedFunctionGroups.⌋

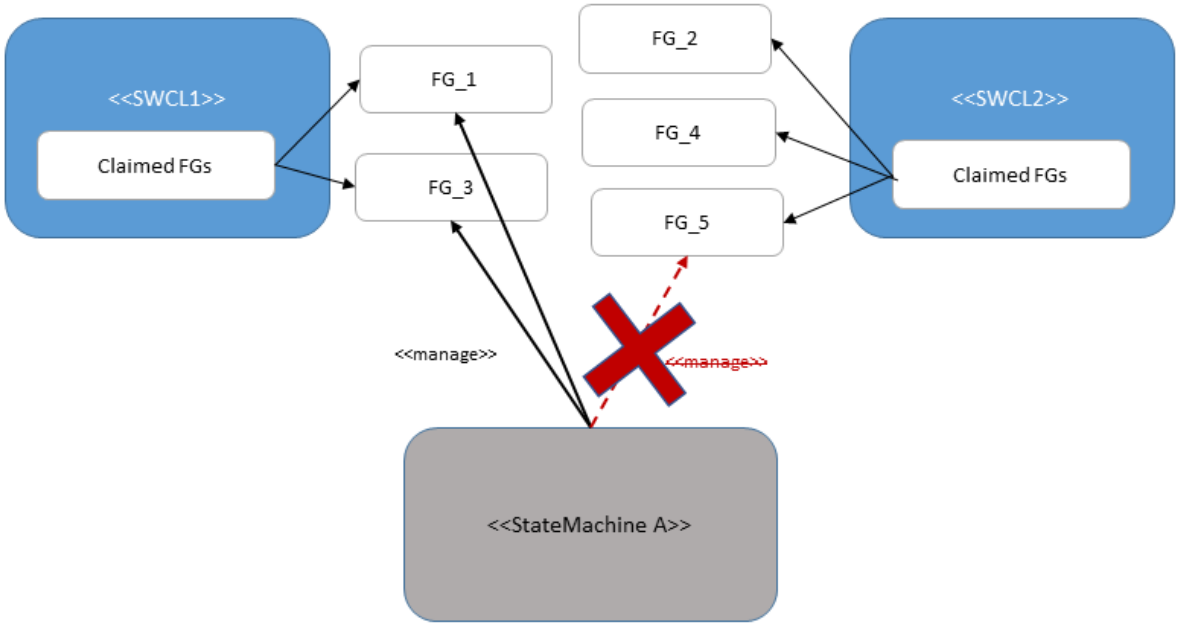

Figure 7.9: Agent - FunctionGroup relation

Please note that a `Controller` could manage `Function Groups` which are claimed by different `Software Clusters`, but that feature is only recommended to be used when no `Agents` are configured.

7.6.4 StateMachine general conditions

When a `StateMachine` exits it shall leave the system in a consistent state. This means that no `Function Group`, which are under control of the `StateMachine` should be in a state where no further influence on their state can be taken as error reaction. Therefore all controlled `Function Groups` shall be in "Off" state thus they do not cause any error.

[SWS_SM_CONSTR_00024] Existence of StateMachine Off state ⌈Each configured `StateMachine` of type `Agent` shall have corresponding "Off" `StateMachine State` configured, at the time when the creation of the manifest is finished.⌋

[SWS_SM_CONSTR_00011] ActionListItems allowed in the "Off" state of a StateMachine of type Agent ⌈In the `ActionList` referencing the "Off" State of a `StateMachine` of type `Agent`, only the following `ActionListItems` shall be allowed:

- `Function Group`::`Off`
- `NmNetworkHandle`::`NoCom`

- SYNC
- Sleep

⌋

It is recommended that any StateMachine State from the StateMachine of type Controller containing MachineFG::Shutdown or MachineFG::Restart should stop all StateMachines of type Agent. By not doing so, the still running processes would be abruptly terminated when host is shut down.

To keep a consistent Function Group State it is needed, that no Function Group is controlled by different StateMachines, as it would not be clear which StateMachine is finally responsible.

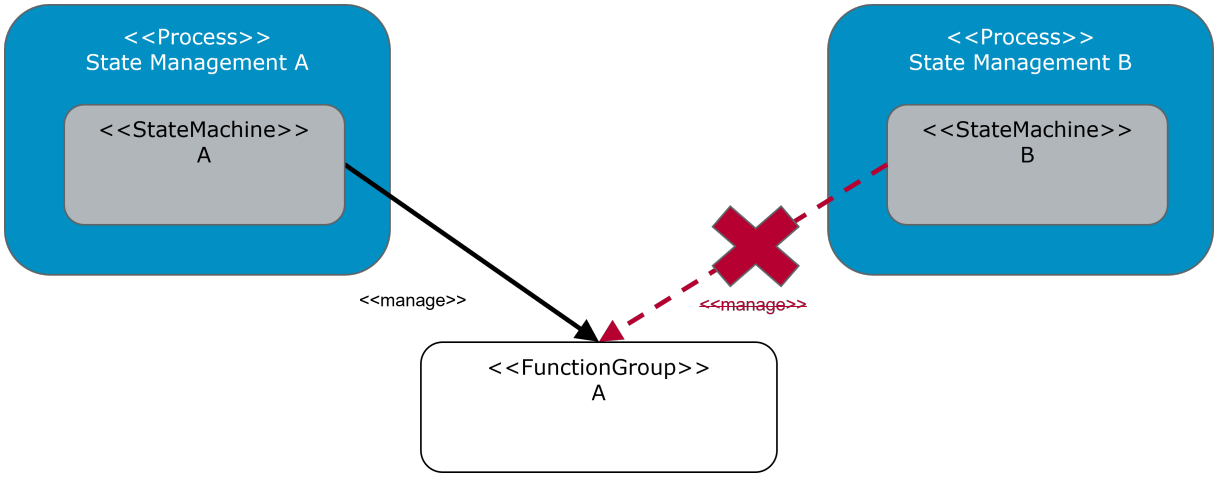

Figure 7.10: Function Group controlled by single StateMachine

[SWS_SM_CONSTR_00013] Function Group shall only be controlled by single StateMachine ⌈A Function Group shall only be referenced by ActionListItems of exactly one StateMachine.⌋

7.6.5 StateMachine state changes

One of the important configuration abilities is to define which StateMachine State shall be entered on which error. The reaction is the same, independent if the issue is reported by Platform Health Management or Execution Management, as the issue causing Process is the same. To achieve this, a mapping table, the ErrorRecoveryTable is introduced, which maps the Execution-Error::EventexecutionError (modelled as ProcessExecutionError.execution-Error) to the required StateMachine State.

Execution Error	Next State
11	Recovery
12	Startup
111	Recovery
23	Suspend
24	Off
ANY	Shutdown

Figure 7.11: ErrorRecoveryTable

To ensure that all errors are covered the following constraint is needed:

[SWS_SM_CONSTR_00014] Handling of non-mapped ExecutionError ⌈Each `ErrorRecoveryTable` shall have exactly one entry configured with value ANY as the ExecutionError⌋

The ANY entry will be used to change to the configured `StateMachine State` when a not configured ExecutionError is reported by `Platform Health Management` or `Execution Management`.

During an active update session, handling of the recovery actions (see [SWS_SM_00601] and [SWS_SM_00664]) should be treated differently, depending on whether the `StateMachine` itself is "ImpactedByUpdate" [SWS_SM_00654] or not. Otherwise errors occurred during or after methods called by the `Update and Configuration Management` could result in `StateMachines` transiting to recovery `StateMachine State`, which might not be the intended action.

[SWS_SM_00601] StateMachine error notification reaction of StateMachines not "ImpactedByUpdate"

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When EventexecutionErrorEvent::ExecutionError is reported from `Platform Health Management` or from `Execution Management` and the `StateMachine` is not "ImpactedByUpdate", `StateMachine` shall

- set internal flag that error recovery is ongoing
- evaluate the next `StateMachine State` configured for executionError from `ErrorRecoveryTable`
- stop processing `ActionListItems` from the `ActionList` referencing the current `StateMachine State`
- switch to the next `StateMachine State` immediately and start processing `ActionListItems` from the `ActionList` referencing this `StateMachine State`

⌋

[SWS_SM_00666] Nested recovery

Upstream requirements: RS_SM_00601

⌈In case that a new issue is reported by `Platform Health Management` via RecoveryHandler() call and the issue has to be handled by a `StateMachine` where internal flag ErrorRecoveryOngoing is set (see [SWS_SM_00601]), this shall be considered as "nested recovery" (see [SWS_SM_00031]).⌋

[SWS_SM_00602] StateMachine ErrorRecoveryOngoing flag reset

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈The internal flag that error recovery is ongoing, shall be reset, when all `ActionListItems` of an `ActionList` referencing a `StateMachine State`, which is requested due to error reaction, are successfully processed.⌋

When a request to change a `StateMachine State` is issued by a `SMControlApplication` there are more steps to consider:

[SWS_SM_00603] StateMachine service interface RequestTransition - not allowed transition

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈The `RequestTransition` method shall return `kTransitionNotAllowed` if the current state of the `StateMachine` is not configured for the `TransitionRequest` value in `TransitionRequestTable` and shall cease any further processing of the request.⌋

[SWS_SM_00604] StateMachine service interface RequestTransition - invalid transition

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈The `RequestTransition` method shall return `kInvalidValue` if `TransitionRequest` value is not configured in `TransitionRequestTable` and shall cease any further processing of the request.⌋

[SWS_SM_00605] StateMachine service interface RequestTransition - recovery ongoing

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈The `RequestTransition` method shall return `kRecoveryTransitionOngoing` if internal flag is set that error recovery is ongoing and shall cease any further processing of the request.⌋

[SWS_SM_00606] Canceling ongoing state transition of StateMachine

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈If transition request was accepted, `RequestTransition` method shall return `kOperationCanceled` to previous `RequestTransition` requests if any is still pending for the `StateMachine`.⌋

[SWS_SM_00607] StateMachine transition execution

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When `StateMachine` receives a valid state change request it shall

- evaluate the next `StateMachine State` configured for TransitionRequest value and current state from `TransitionRequestTable`
- stop processing `ActionListItems` from the `ActionList` referencing the current `StateMachine State`
- switch to the next `StateMachine State` immediately and start processing `ActionListItems` from the `ActionList` referencing this `StateMachine State`.

⌋

[SWS_SM_00650] StateMachine service interface RequestTransition - transition failed

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈The `RequestTransition` method shall return `kTransitionFailed`, if an error occurred during processing of `ActionListItems` (see [SWS_SM_00607]).⌋

There is another source for `StateMachine State` change requests: `NetworkManagement NetworkHandle` changes. As `NetworkHandles` are modelled as PortPrototypes, they can be used as input towards `TransitionRequestTable`. This means that a change in a `NetworkHandle` from `NoCom` to `FullCom` (or vice versa) will trigger `StateMachine States` when configured (and conditions are met). To make this work a mapping `NmInteractsWithSmMapping` between `NmNetworkHandle` and `StateManagementStateRequest` (as "input" to the transition table) has to be configured.

Transition Request	Current State	Next State
Nh1_FullCom	Off	Camera Active
Nh1_NoCom	Camera Active	Off

Figure 7.12: Extended transition request table

[SWS_SM_00620] StateMachine transition - NetworkHandle goes to FullCom

Upstream requirements: RS_SM_00001, RS_SM_00005, RS_SM_00401

⌈When `StateMachine` receives a change of a `NetworkHandles` to `FullCom` it shall

- evaluate the next `StateMachine State` configured for TransitionRequest value and current state from `TransitionRequestTable`
- stop processing `ActionListItems` from the `ActionList` referencing the current `StateMachine State`
- switch to the next `StateMachine State` immediately and start processing `ActionListItems` from the `ActionList` referencing this `StateMachine State`.

⌋

[SWS_SM_00621] StateMachine transition - NetworkHandle goes to NoCom

Upstream requirements: RS_SM_00001, RS_SM_00005, RS_SM_00401

⌈When `StateMachine` receives a change of a `NetworkHandles` to `NoCom` it shall

- evaluate the next `StateMachine State` configured for TransitionRequest value and current state from `TransitionRequestTable`
- stop processing `ActionListItems` from the `ActionList` referencing the current `StateMachine State`
- switch to the next `StateMachine State` immediately and start processing `ActionListItems` from the `ActionList` referencing this `StateMachine State`.

⌋

Please note that a change in a `NmNetworkHandle` can cause state transitions to more than one `StateMachine`. `NmNetworkHandle` could be seen as a kind of "remote

control", and for this reason a change in a NmNetworkHandle could activate functionality in more than one StateMachine. E.g. switching on parking assistance could activate the rear camera and proximity sensor, which could be controlled by different StateMachines.

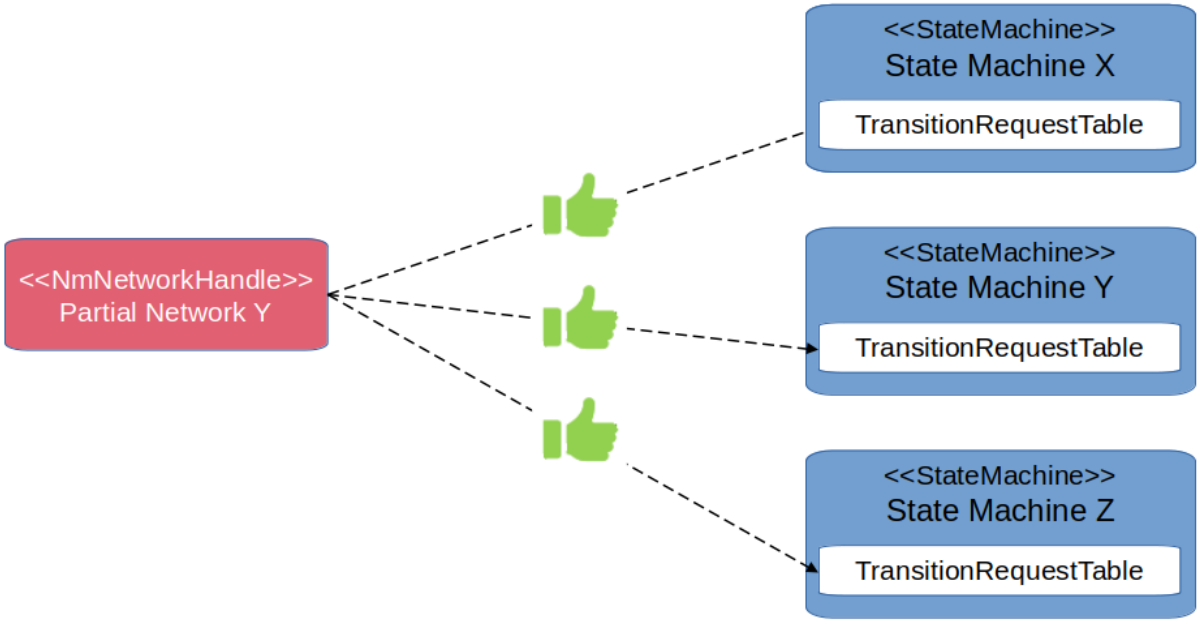

Figure 7.13: Example of one NmNetworkHandle influencing multiple StateMachines

7.6.6 StateMachine ActionLists

ActionLists are a collection of ActionListItems and are referencing a StateMachine State. An ActionList, respectively its ActionListItems are executed as soon as a StateMachine State is entered. ActionLists are represented by meta-class StateManagementActionList.

7.6.7 StateMachine ActionListItems

There are multiple kinds of ActionListItems:

- Requesting a Function Group State
- Start a StateMachine with optional parameter state
- Stop a StateMachine
- SYNC to sync between different ActionListItems
- Sleep to delay processing the next ActionListItems

- SetNetworkHandle switches the provided NetworkHandle to the configured state (NoCom or FullCom)

[SWS_SM_00608] ActionListItem - Function Group State

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When a Function Group State ActionListItem is found in the ActionLists, StateMachine shall request the configured Function Group State from Execution Management.⌋

To enable State Management to build a Function Group dependency the ActionListItems shall be executed in the order they are configured.

[SWS_SM_00609] ActionList processing order

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈The ActionListItems in the ActionLists shall be processed in the order they are configured.⌋

To fully support this kind of dependency a "SYNC" item is introduced, that waits till all ActionListItems since

- the beginning of the ActionList
- the last "SYNC" item

have been successfully executed.

[SWS_SM_00610] processing SYNC ActionListItem

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When processing "SYNC" ActionListItem on the list, StateMachine shall wait until all previously processed ActionListItems are finished before moving to the next item after "SYNC".⌋

[SWS_SM_00611] processing ActionListItem

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈ActionListItems shall be processed in parallel unless SYNC ActionListItem is processed.⌋

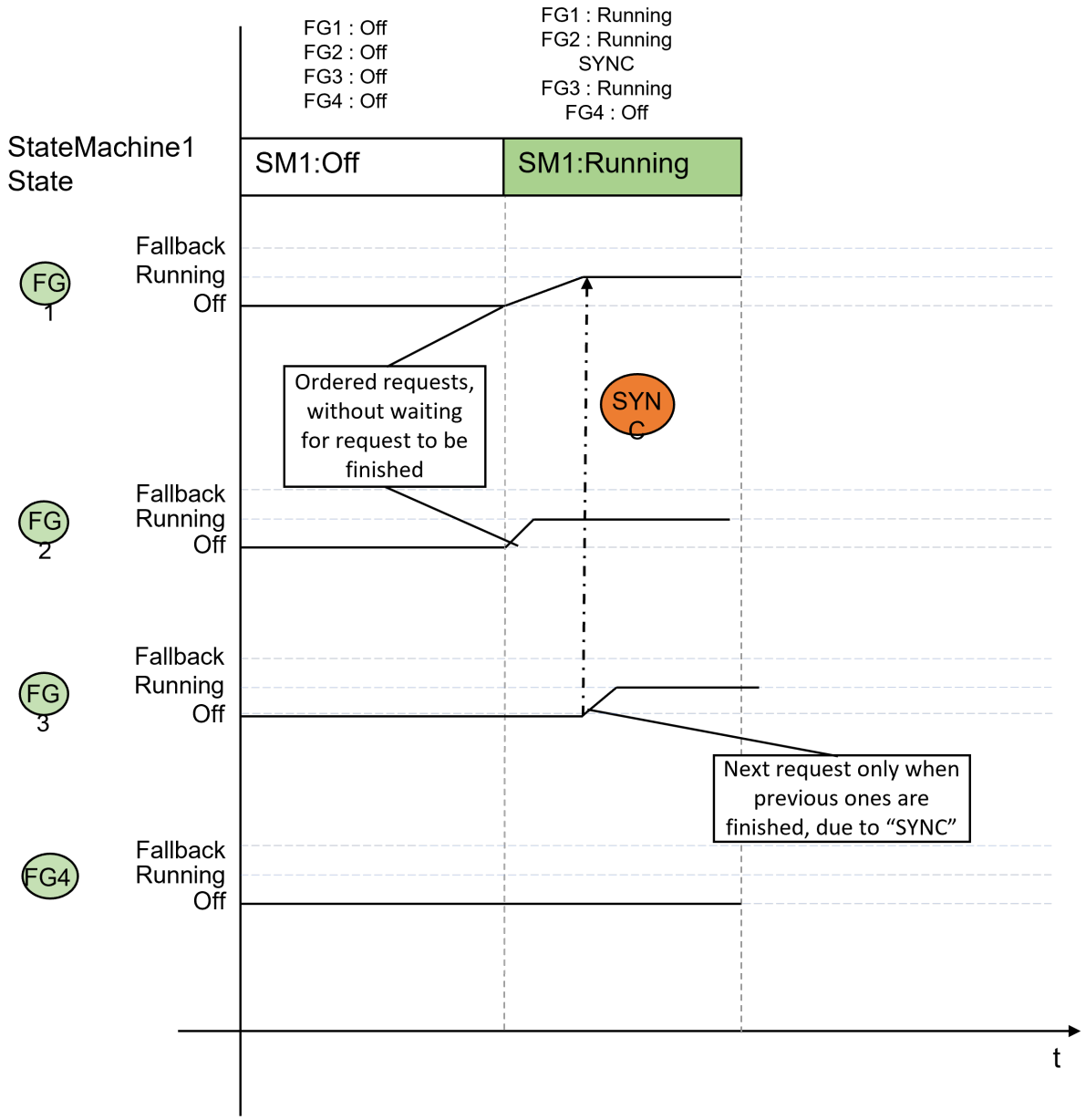

Figure 7.14: Parallel ActionListItem execution and SYNC

Please note that parallel execution of the `ActionListItems` is heavily dependent of the implementation and the underlaying hardware and operating system

As - together with the "SYNC" `ActionListItem` - `Function Group State` dependencies can be realized, the referenced `Function Groups` can be given in an arbitrary order to fulfill the project-specific needs.

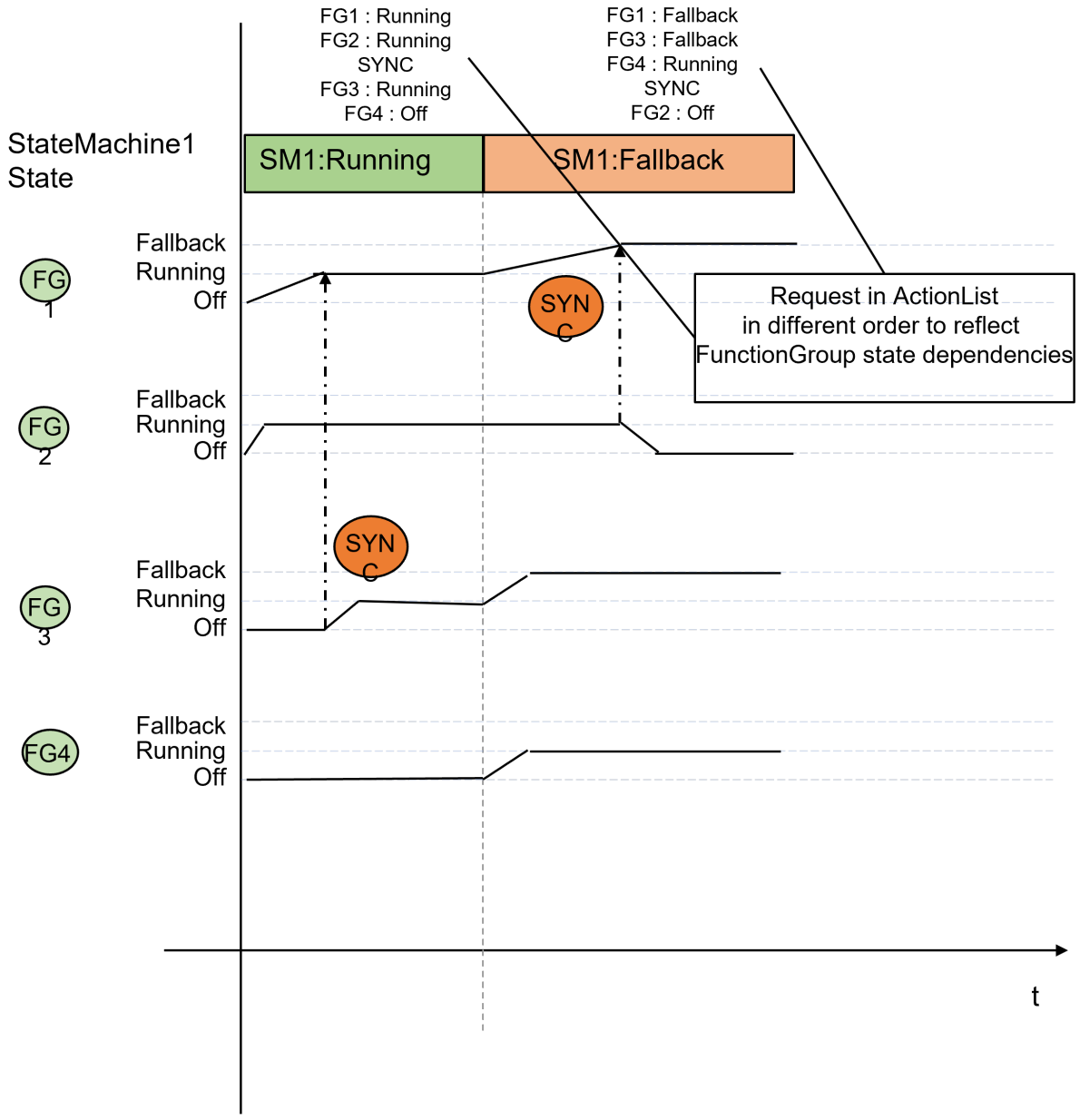

Figure 7.15: Arbitrary order for ActionListItems

To ensure that no `Function Group` nor any `NmNetworkHandle` is missed in any state, as it might lead to inconsistencies in the expected functionality, it is needed within a single `StateMachine`, that each `ActionList` contains the same `Function Groups` and `NmNetworkHandle`s, even if their state does not change from a `StateMachine State` to another.

[SWS_SM_CONSTR_00015] Completeness of controlled Function Groups ⌈Each `ActionList` referencing different `StateMachine States` of the same `StateMachine` shall reference the same set of `Function Groups`.⌋

[SWS_SM_CONSTR_00032] Completeness of controlled NmNetworkHandles ⌈Each `ActionList` referencing different `StateMachine States` of the same `StateMachine` shall reference the same set of `NmNetworkHandle`s.⌋

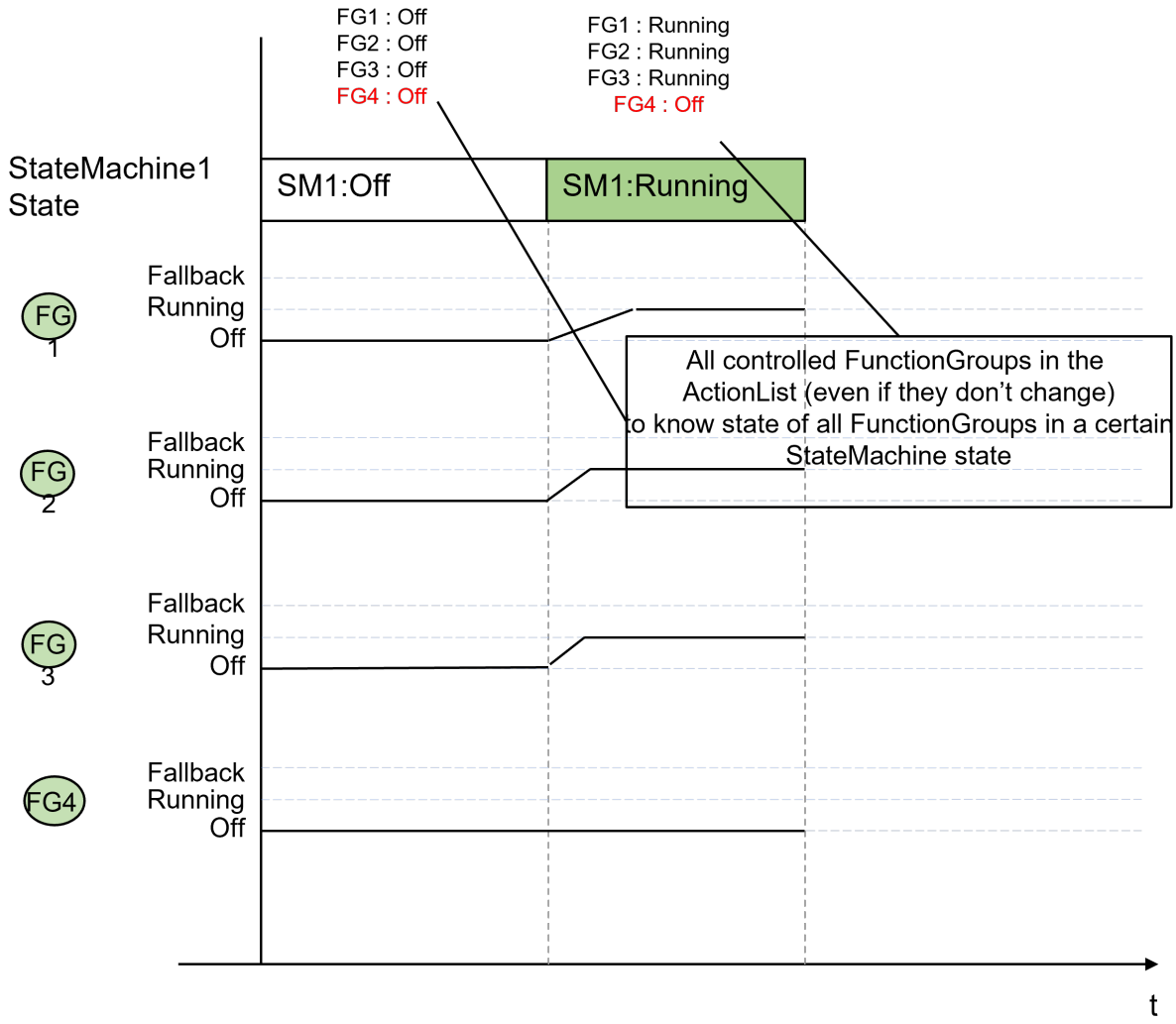

Figure 7.16: Completeness of controlled Function Groups

7.6.8 Controlling multiple StateMachine Instances

The `ActionListItem` approach offers the ability to start/stop `StateMachine` instances, as it might be needed in a project-specific environment.

To reduce complexity in configuration there should be only one level of `StateMachine` nesting. Therefore, only the `StateMachine` with the role `Controller` should be used to Start/Stop other `StateMachine` instances, called `Agents`.

[SWS_SM_CONSTR_00019] Usage of ActionListItem "StartStateMachine" and "StopStateMachine" ⌈Only the StateMachine with the role Controller shall use the ActionListItem "StartStateMachine" and "StopStateMachine".⌋

[SWS_SM_CONSTR_00016] Completeness of controlled StateMachines ⌈Each ActionList referencing a StateMachine State of the same StateMachine shall reference the same set of controlled StateMachines.⌋

[SWS_SM_00612] ActionListItem "Start StateMachine" without parameter, StateMachine is not running

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When the ActionListItem "Start StateMachine" is processed, the referenced StateMachine shall be started. The StateMachine shall transition to the configured initial state.⌋

[SWS_SM_00622] ActionListItem "Start StateMachine" with parameter, StateMachine is not running

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When the ActionListItem "Start StateMachine" is processed, the referenced StateMachine shall be started. The StateMachine shall transition to the state, which is provided as parameter.⌋

[SWS_SM_00613] ActionListItem "Start StateMachine" - without parameter, StateMachine is already running

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When the ActionListItem "Start StateMachine" is processed, and the referenced StateMachine is already started, this processing shall be skipped.⌋

[SWS_SM_00623] ActionListItem "Start StateMachine" - with parameter, StateMachine is already running

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When the ActionListItem "Start StateMachine" is processed, and the referenced StateMachine is already started, the StateMachine shall transition to the state, which is provided as parameter.⌋

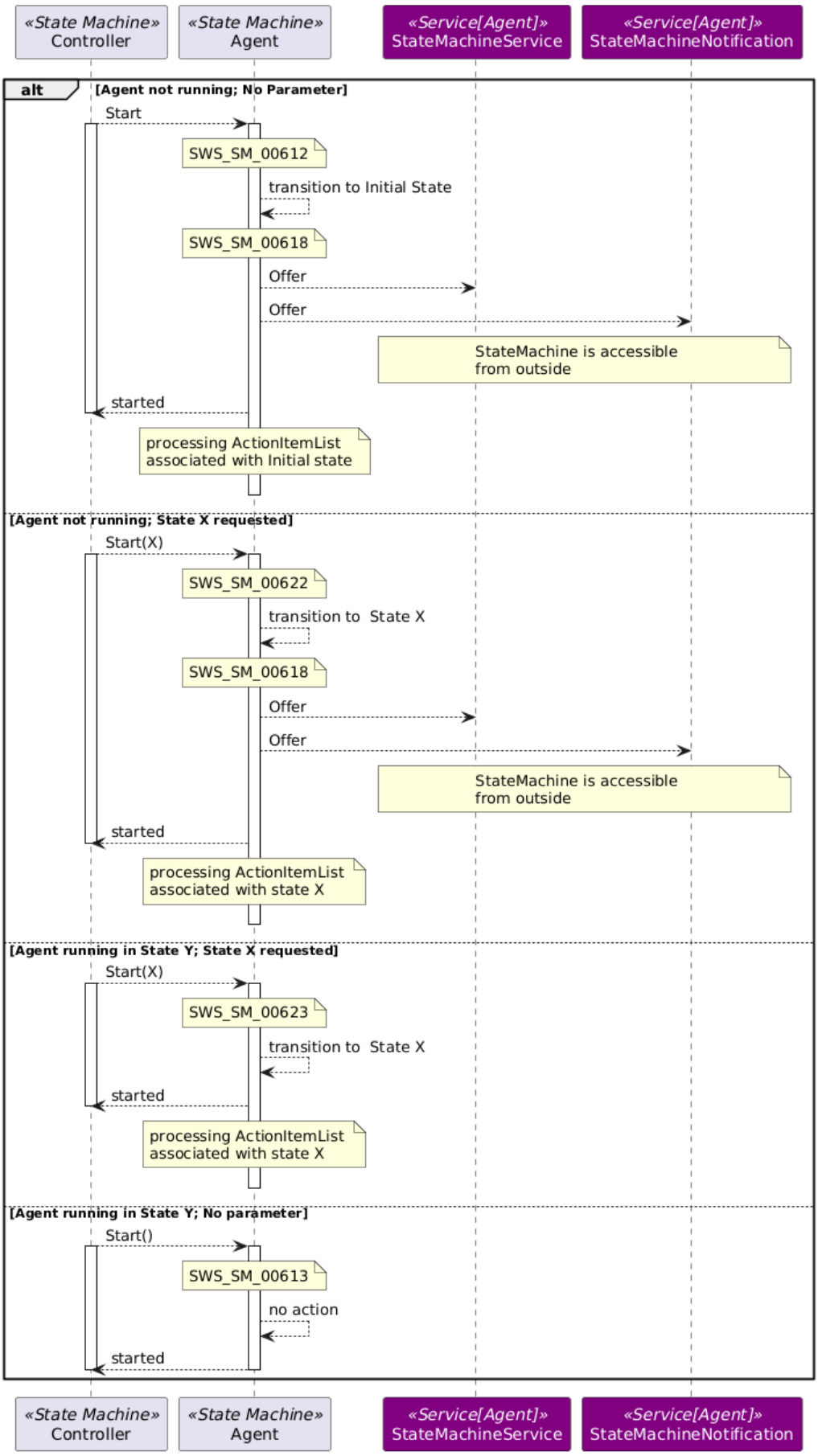

Figure 7.17: ActionListItem Start StateMachine

Please note that all ActionListItems of a requested StateMachine State will always be executed, independently if the StateMachine was already in the requested StateMachine State directly before the request. This is valid for [SWS_SM_00623], [SWS_SM_00620], [SWS_SM_00621], [SWS_SM_00601] and [SWS_SM_00607].

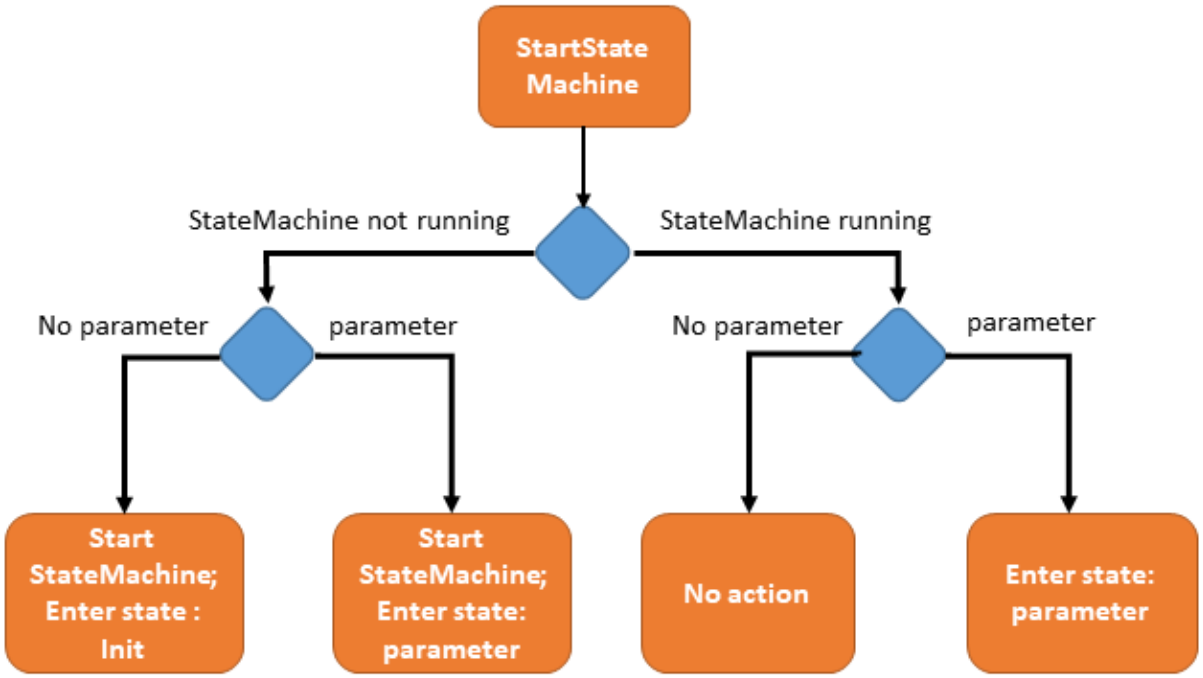

Figure 7.18: StartStateMachine decision tree

[SWS_SM_00614] ActionListItem "Stop StateMachine" processing

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When the ActionListItem "Stop StateMachine" is processed, the StateMachine with the provided ID shall be stopped.⌋

[SWS_SM_00615] ActionListItem "Stop StateMachine" processing - StateMachine is not running

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When the ActionListItem "Stop StateMachine" is processed, and the StateMachine with the provided ID is not running, this processing shall be skipped.⌋

[SWS_SM_00651] Processing StopStateMachine ActionListItem

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When an ActionListItem "Stop StateMachine" is processed (by StateMachine of type Controller) the given StateMachine shall transition to the "Off" StateMachine State.⌋

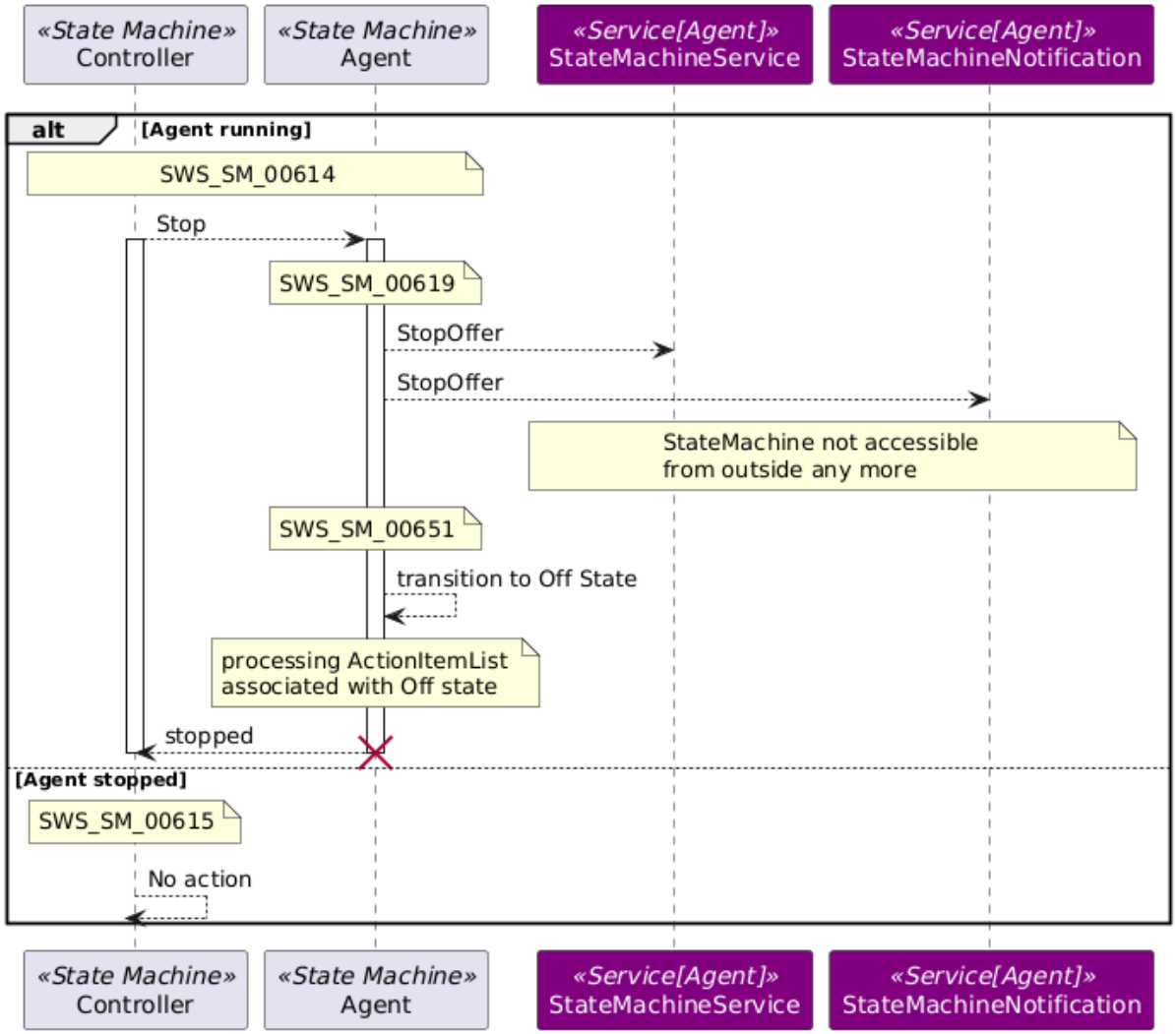

Figure 7.19: Processing ActionListItem StopStateMachine

Please note, that only `StateMachines` of type `Agent` need an "Off" `StateMachine State`. This is needed to ensure that, no processes or `NmNetworkHandle`s are left "uncontrolled" when the `StateMachine` is being stopped (see [SWS_SM_00614]). A `StateMachine` of type `Controller` is representing life-cycle of a `Machine`. For this reason stopping a `StateMachine` of type `Controller` should consider usage of `MachineFG Shutdown` state. The name of `StateMachine State` which is performing this task does not need to be standardized as the `State Management` does not intent to shutdown `Machine` on its own.

7.6.9 ActionListItem Sleep

To support timed actions of `StateMachine States` e.g. to realize "afterrun use-cases" the `Sleep ActionListItem` was introduced.

[SWS_SM_00624] ActionListItem - Sleep

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When a `Sleep` `ActionListItem` is found in the `ActionLists`, `StateMachine` shall delay processing next `ActionListItem` on the `ActionLists` for the configured time.⌋

Please note that `Sleep` `ActionListItem` will not "block" processing incoming triggers meanwhile. This means that a call to `RequestTransition`, an error Notification ([SWS_SM_00601]) or a change in a `NmNetworkHandle` ([SWS_SM_00620] / [SWS_SM_00621]) for the sleeping the `StateMachine State` might cause a `StateMachine State` change (depending on configuration).

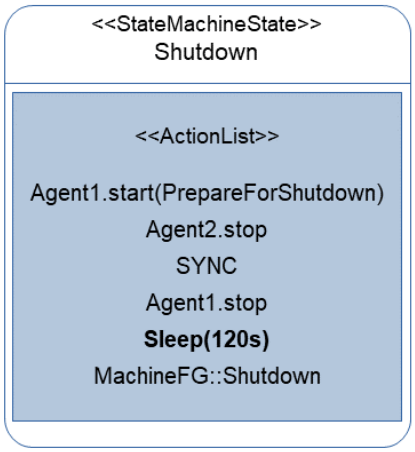

Figure 7.20: Example for an ActionList using ActionListItem Sleep

7.6.10 ActionListItem SetNetworkHandle

To support switching of NetworkHandles within `StateMachine States` the `SetNetworkHandle` `ActionListItem` was introduced. To make this work a mapping `SmInteractsWithNmMapping` between `NmNetworkHandle` and `StateManagementNmActionItem` has to be configured.

[SWS_SM_00625] ActionListItem - SetNetworkHandle FullCom

Upstream requirements: RS_SM_00001, RS_SM_00005, RS_SM_00401

⌈When a `SetNetworkHandle` `ActionListItem` with parameter `FullCom` is found in the `ActionLists`, `StateMachine` shall set the corresponding NetworkHandle to `FullCom`.⌋

[SWS_SM_00626] ActionListItem - SetNetworkHandle NoCom

Upstream requirements: RS_SM_00001, RS_SM_00005, RS_SM_00401

⌈When a `SetNetworkHandle` `ActionListItem` with parameter `NoCom` is found in the `ActionLists`, `StateMachine` shall set the corresponding NetworkHandle to `NoCom`.⌋

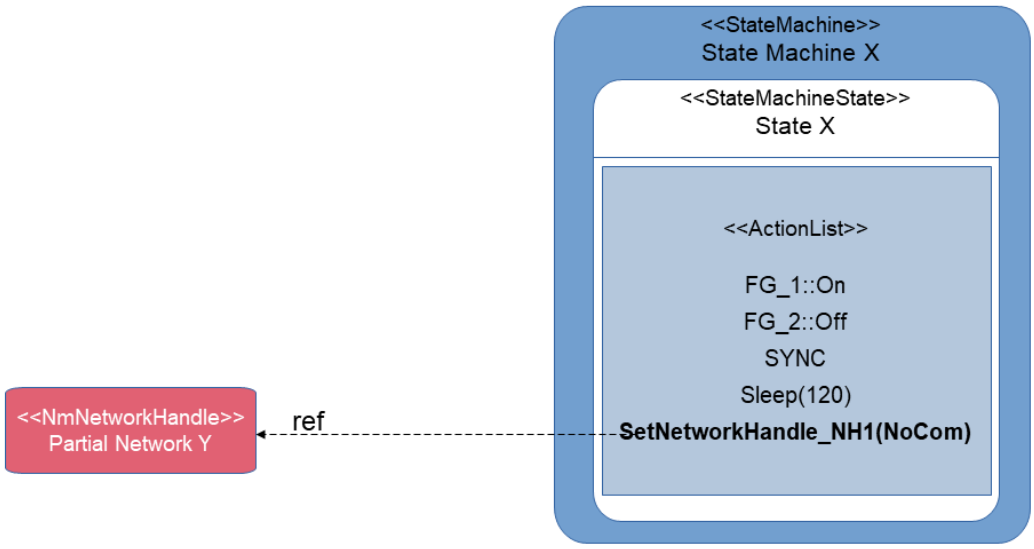

Figure 7.21: Afterrun example using the SetNetworkHandle in combination with Sleep

Please note that only one `StateMachine` should be able to request state changes to a specific `NmNetworkHandle`. Letting more than one `StateMachine` control the same `NmNetworkHandle` could bring non-predictable behavior to the state of the `NmNetworkHandle`.

[SWS_SM_CONSTR_00025] NmNetworkHandle shall only be controlled by single StateMachine ⌈A `NmNetworkHandle` shall only be referenced by `ActionListItems` of exactly one `StateMachine`.⌋

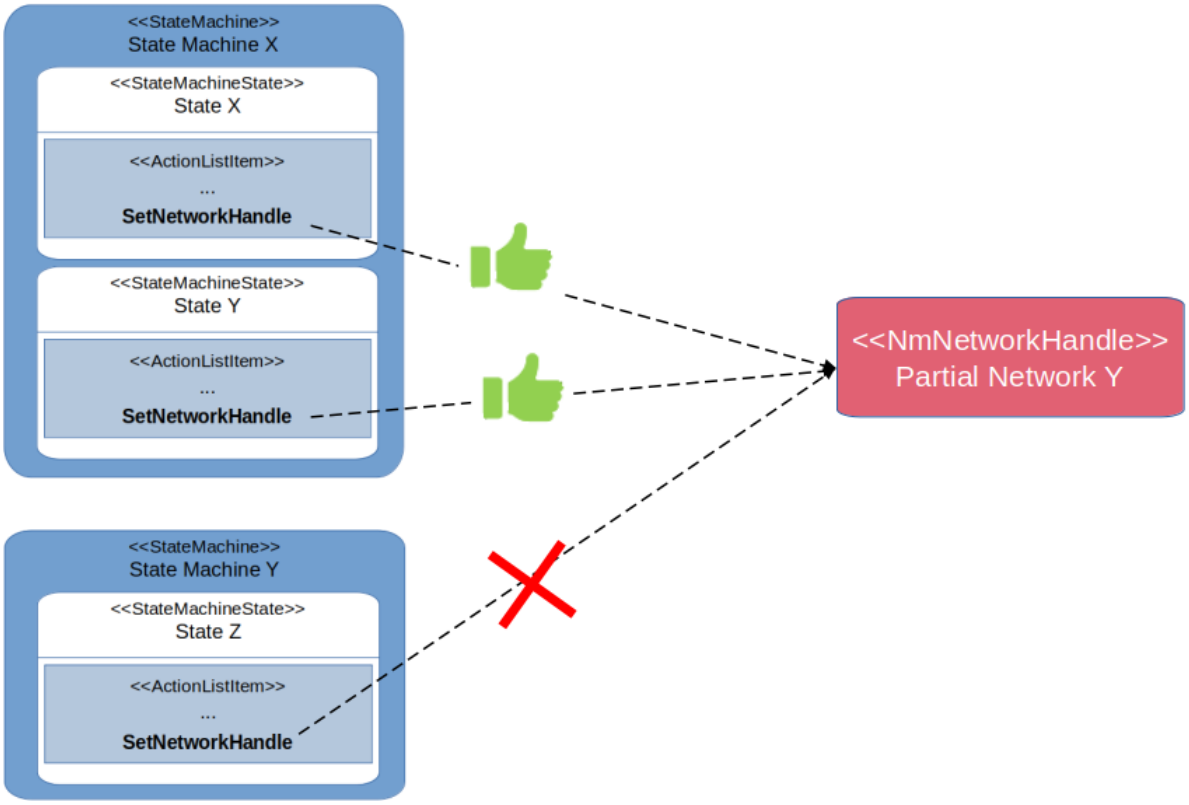

Figure 7.22: StateMachine to NmNetworkHandle restriction

7.6.11 StateMachine State notification

As State Management StateMachine States reflect the current functionality of a Machine, which might be in the interest of several entities in the Machine (e.g. Firewall, SystemHealthManagement, ...) it shall be possible to make the StateMachine States available to them. Therefore, it shall be possible to configure a StateMachineNotification service interface (modelled as meta-class ServiceInterface) for a StateMachine.

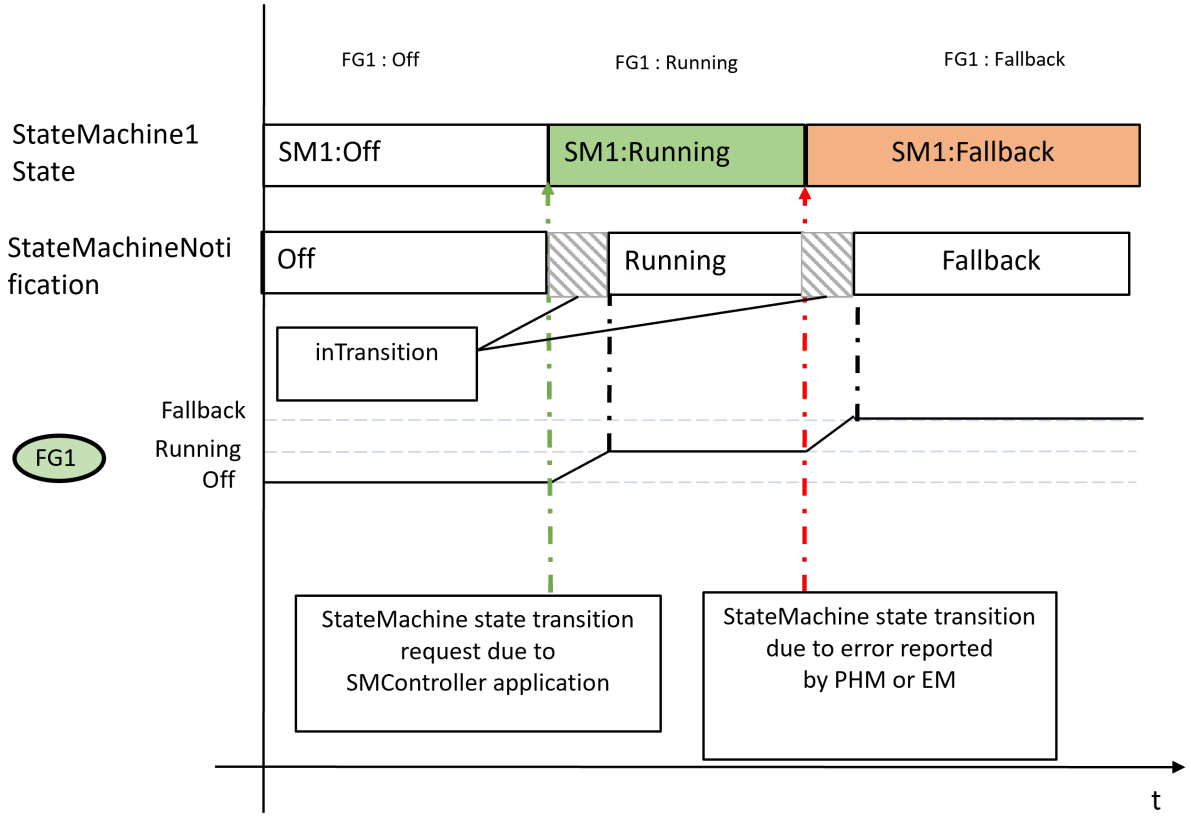

Figure 7.23: Value of configured StateMachineNotification::CurrentState field

[SWS_SM_00616] CurrentState value during StateMachine State transition

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When a `StateMachineNotification` interface is configured for the `StateMachine` and a `StateMachine State` transition has been started, the value of the `CurrentState` field shall be set to "inTransition".⌋

Please note that the value "inTransition" is set independently of the source (`Platform Health Management`, `Execution Management`, `SMControlApplication`, ...) and is kept, even if another `StateMachine State` transition, as reaction to an error notification, is performed.

[SWS_SM_00617] CurrentState value after StateMachine State transition

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When a `StateMachineNotification` interface is configured for the `StateMachine`the value of the `CurrentState` field shall be set to the current `StateMachine State` as soon as all `ActionListItems` (in the `ActionList` referencing the current `StateMachine State`) have been executed and all results have been collected.⌋

[SWS_SM_CONSTR_00026] Forbidden usage of "inTransition" as a StateMachine State ⌈At the time when the creation of the manifest is finished, each configured StateMachine shall not define a State named "inTransition".⌋

7.6.12 StateMachine support for Update and Configuration Management

To support Update and Configuration Management [10] during Machine update, State Management provides UpdateRequest interface. In general, update process can be roughly divided into five steps (when we look from State Management point of view):

- Starting update session.
- Preparing for update.
- Verification of the software after deployment on the Machine.
- Potential rollback of the software deployed to the Machine.
- Finishing update session.

This section provides a closer look at how Machine update is realized using StateMachines.

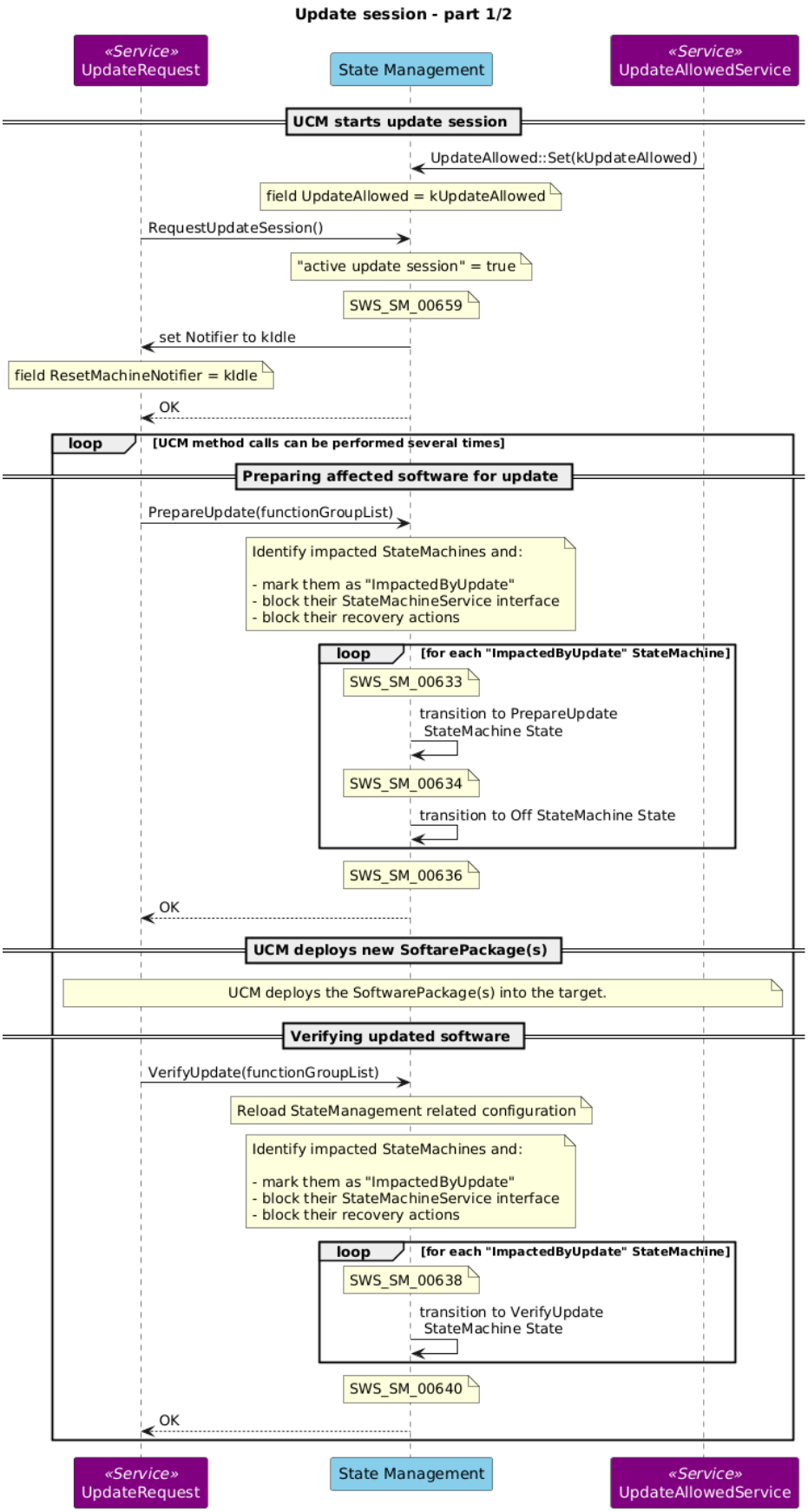

Figure 7.24: Overview of update session within StateMachine approach (part 1 of 2)

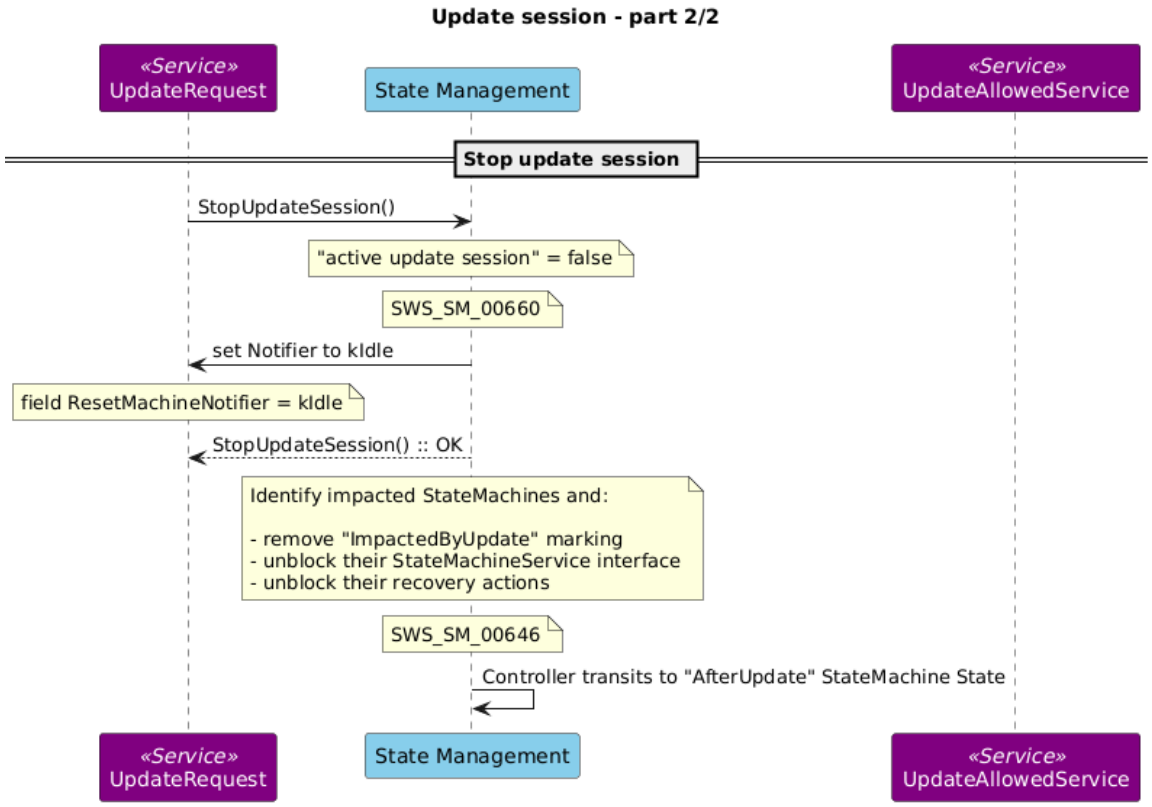

Figure 7.25: Overview of update session within StateMachine approach (part 2 of 2)

The Update and Configuration Management expects that a single logical entity will be responsible for StateMachine during update session. For this reason it is needed to restrict who can instantiate UpdateRequest interface and how many instances are permitted per Machine.

[SWS_SM_CONSTR_00020] Upper multiplicity of UpdateRequest interface ⌈In the context of Machine there shall be at most one instance of UpdateRequest interface at the time when the creation of the manifest is finished.⌋

[SWS_SM_00629] Only Process controlling StateMachine of type Controller can provide UpdateRequest interface

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Only Modelled Process controlling StateMachine of type Controller shall be able to instantiate UpdateRequest interface.⌋

Machine update starts with Update and Configuration Management calling RequestUpdateSession method. The Modelled Process controlling StateMachine of type Controller cannot decide on its own if the update can be started. This decision is delegated to SMControlApplication, where project specific logic can asses if update process can be started. SMControlApplication has to set

UpdateAllowed accordingly. Please note that it is expected the feasibility of an update campaign should be assessed at the vehicle level and Update and Configuration Management is not expected to call RequestUpdateSession without upfront synchronization. However, update campaign may involve multiple Machines and therefore take some time. During this time local circumstances may change and for this reason call to RequestUpdateSession is necessary.

When SMControlApplication does not allow update, Modelled Process controlling StateMachine of type Controller should refuse update request from Update and Configuration Management.

[SWS_SM_00630] Rejection of update session

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When UpdateAllowed is set to kUpdateNotAllowed, Modelled Process controlling StateMachine of type Controller shall return kOperationRejected error from the RequestUpdateSession method.⌋

If SMControlApplication allow update session to start, Modelled Process controlling StateMachine of type Controller should return a positive response back to Update and Configuration Management.

[SWS_SM_00631] Acceptance of update session

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When UpdateAllowed is set to kUpdateAllowed, Modelled Process controlling StateMachine of type Controller shall return success from the RequestUpdateSession method.⌋

[SWS_SM_00659] Set ResetMachineNotifier to its default value when update session starts

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Once an update session is accepted [SWS_SM_00631] Modelled Process controlling StateMachine of type Controller shall set the field ResetMachineNotifier to its default value (see [SWS_SM_00212])⌋

As per [SWS_SM_00630] and [SWS_SM_00631] the UpdateAllowed field is only evaluated during a call to RequestUpdateSession. For this reason once an update session is granted any subsequent change to the UpdateAllowed field will have no effect on the currently active session. Additionally it is possible that multiple SMControlApplications can have access to the UpdateAllowedService interface and could modify the UpdateAllowed field at the same time. Each project can configure access to this interface using IAM configuration.

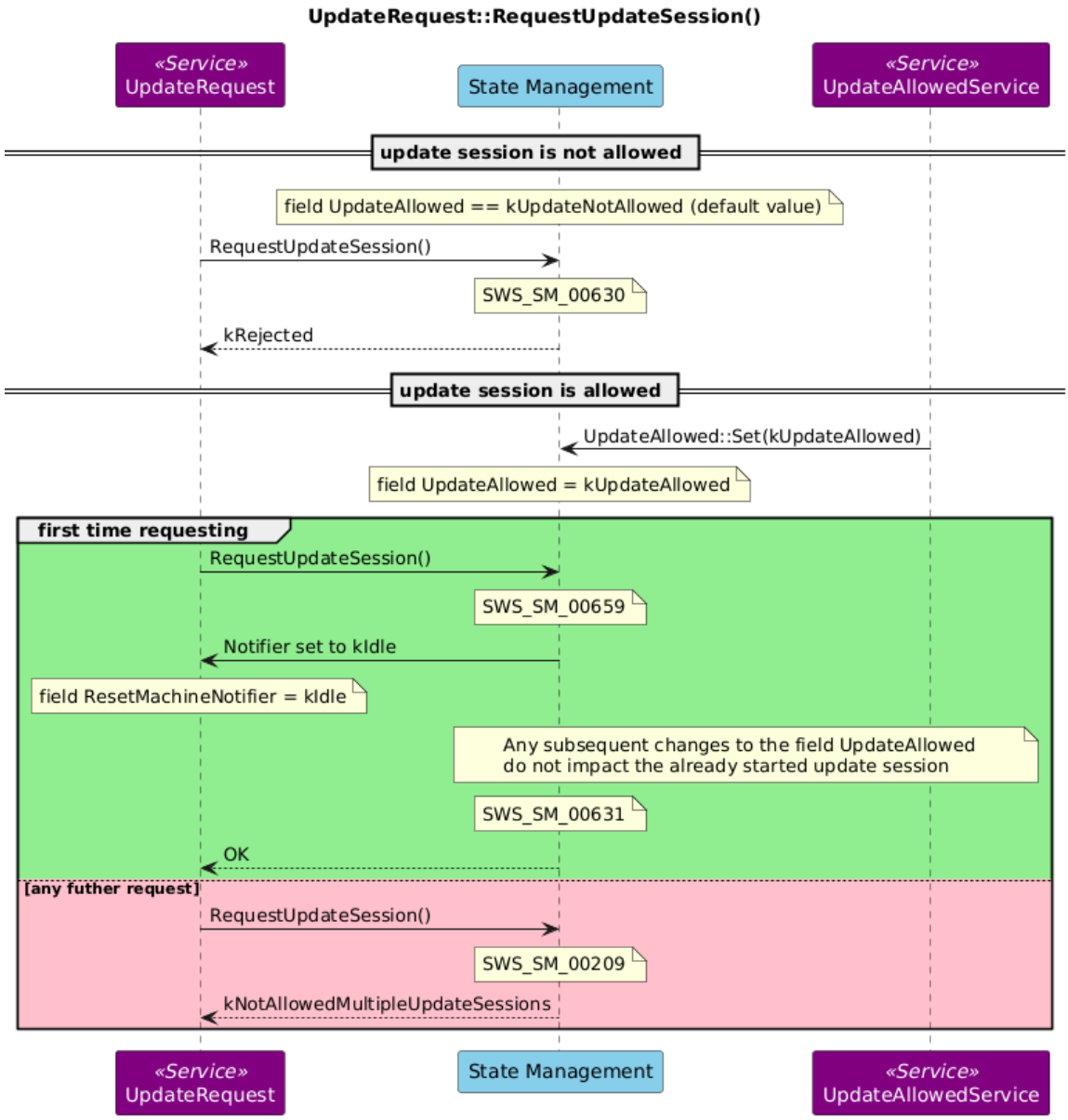

Figure 7.26: Requesting update session within StateMachine approach

Please note that it is deliberately left as an implementation detail when `RequestTransition` method should be blocked. `AUTOSAR Adaptive Platform` will only specify the latest point in time when this should happen. Implementations may choose to keep `StateMachine` of type `Controller` more responsive, by accepting state change requests, in case there is a delay between calling `RequestUpdateSession` and actual start of the update process.

[SWS_SM_00654] StateMachine marked as "ImpactedByUpdate"

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈During a call to `PrepareUpdate`, `VerifyUpdate` or `PrepareRollback`, the `Modelled Process` controlling the `StateMachine` of type `Controller` shall mark a

StateMachine as "ImpactedByUpdate", if any of the Function Groups managed by the StateMachine is listed in the parameter passed to the method call.⌋

Because the StateMachine of type Controller manages also the StateMachines of type Agent, the Controller is also affected by update session when a StateMachine of type Agent is marked "ImpactedByUpdate".

[SWS_SM_00655] Indirect marking of StateMachine of type Controller as "ImpactedByUpdate"

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Whenever a StateMachine of type Agent is marked as "ImpactedByUpdate" the StateMachine of type Controller shall also be marked as "ImpactedByUpdate".⌋

[SWS_SM_00664] StateMachine error reaction of StateMachines "ImpactedByUpdate"

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When ExecutionErrorEvent::executionError is reported from Platform Health Management or from Execution Management and the StateMachine is "ImpactedByUpdate" [SWS_SM_00654], the StateMachine shall

- ignore the recovery request
- log the event, if logging is activated

⌋

Please note that errors during an update session are notified to Update and Configuration Management via [SWS_SM_00635] for PrepareUpdate, [SWS_SM_00639] for VerifyUpdate, [SWS_SM_00644] for PrepareRollback and [SWS_SM_00663] for ResetMachine.

Preparation for update marks the next step in the update process. Before Update and Configuration Management can perform any software changes, all StateMachines affected by this update should be adequately prepared. For this reason every StateMachine should have a dedicated state configured and in that state all necessary actions should be performed. For simplicity reasons, if there is no need to perform any special operations before update can be started, all Function Groups managed by StateMachine can be transitioned to the Off state.

[SWS_SM_CONSTR_00021] Existence of StateMachine PrepareUpdate state
⌈Each configured StateMachine shall have corresponding PrepareUpdate StateMachine State configured, at the time when the creation of the manifest is finished.⌋

When Update and Configuration Management invoke PrepareUpdate method, actions that needs to be performed by Modelled Process controlling StateMachine of type Controller are relatively simple. As Update and Configuration Management needs exclusive access to the Machine and StateMachine of type Controller can not only command Function Groups, but also others StateMachines, it should prevent any further changes to its own StateMachine State to avoid a situation where, for example, a Function Group is at the same time updated and activated.

Please note that once a call to RequestTransition of StateMachine of type Controller has been answered with kUpdateInProgress, each consecutive call should be answered with kUpdateInProgress, until Update and Configuration Management calls StopUpdateSession (see [SWS_SM_00647]).

To enable a StateMachine of type Agent to fulfill all steps which are needed during an update session it is needed that the StateMachine State cannot be influenced from the outside if they are marked as "ImpactedByUpdate".

[SWS_SM_00649] Block RequestTransition method during an update session

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Any call to the RequestTransition for a StateMachine shall return kUpdateInProgress when the StateMachine is marked as "ImpactedByUpdate".⌋

[SWS_SM_00627] Evaluation of NetworkHandle changes during an update session

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈StateMachines shall keep their StateMachine State, if the StateMachine is marked as "ImpactedByUpdate" and changes in a NmNetworkHandle are recognized.⌋

After preventing changes to the internal state, Modelled Process controlling StateMachine of type Controller needs to identify which parts of the Machine are affected and should transition any affected StateMachines to the PrepareUpdate state. Identification can be based on the list that Update and Configuration Management supplies as a parameter to the PrepareUpdate method. Additionally any StateMachine of type Agent, that is affected by the update session, shall be stopped as a part of preparation process.

[SWS_SM_00633] Transition affected StateMachines to PrepareUpdate state

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Modelled Process controlling StateMachine of type Controller, during a call to PrepareUpdate method, shall transition every affected StateMachine to the PrepareUpdate state.⌋

[SWS_SM_00634] Shutdown of affected StateMachines during a call to PrepareUpdate method

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Modelled Process controlling StateMachine of type Controller, during a call to PrepareUpdate method, shall stop every affected StateMachine of type Agent.⌋

Please note that it is expected that [SWS_SM_00634] is only executed after a successful execution of [SWS_SM_00633] for a particular StateMachine.

Stopping an StateMachine effectively transition all Function Groups managed by that StateMachine, to the Off state. For this reason a transition to the Off state in [SWS_SM_CONSTR_00021] is not mandatory, but can be performed for clarity reasons.

If any of the steps required to prepare for update fails, Modelled Process controlling StateMachine of type Controller should return an error to Update and Configuration Management. For example, a transition of affected StateMachine to the PrepareUpdate state could fail. Continuing in such a scenario can be potentially fatal, as not all operations configured for that state were executed. In such scenario the Machine itself is not considered to be prepared for update.

[SWS_SM_00635] Failing to prepare for update

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈If Modelled Process controlling StateMachine of type Controller fails to prepare for the update process, it shall return kOperationFailed error from the PrepareUpdate method.⌋

When Modelled Process controlling StateMachine of type Controller is finally ready for update it should return a positive response back to Update and Configuration Management.

[SWS_SM_00636] Successful preparation for update

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When Modelled Process controlling StateMachine of type Controller successfully prepares for update, it shall return success from the PrepareUpdate method.⌋

After Modelled Process controlling StateMachine of type Controller successfully prepared for update, Update and Configuration Management will perform any necessary changes. When deployment is finished it is needed to verify if software was successfully updated. Software verification happens during a call to VerifyUpdate method. Here the steps that needs to be performed by Modelled Process

controlling StateMachine of type Controller are analogous to the steps for update preparation and thus will be discussed in less details. Each StateMachine should have VerifyUpdate state configured and in this state all necessary steps need to verify that software was successfully updated, should be configured. It is recommended that Verify state, which is mandatory for every Function Group, is used.

[SWS_SM_CONSTR_00022] Existence of StateMachine VerifyUpdate state ⌈Each configured StateMachine shall have corresponding VerifyUpdate StateMachine State configured, at the time when the creation of the manifest is finished.⌋

Before starting verification, it is needed to block RequestTransition method - when not already done.

As the next step, transition of all affected StateMachines to the VerifyUpdate state is needed. When identifying which StateMachines are affected, the list that Update and Configuration Management supplies as a parameter to the VerifyUpdate method can be used.

[SWS_SM_00638] Transition affected StateMachines to VerifyUpdate state

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Modelled Process controlling StateMachine of type Controller, during a call to VerifyUpdate method, shall transition every affected StateMachine to the VerifyUpdate state.⌋

As all affected StateMachines (except Controller) are stopped, this implies that they need to be started first.

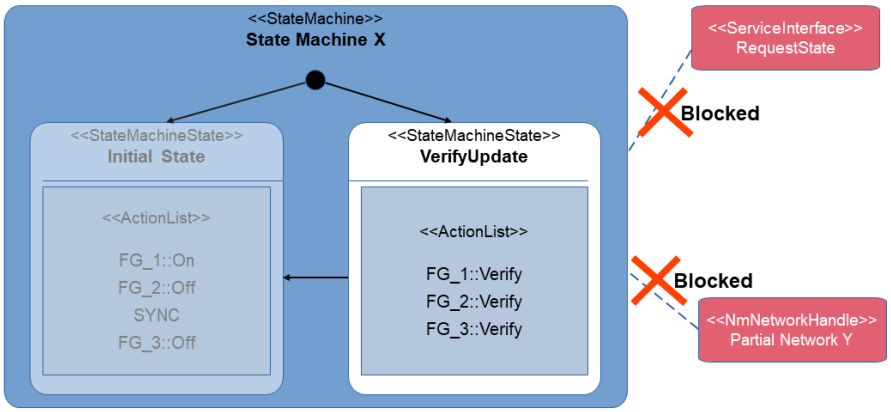

Figure 7.27: Example for StateMachineState VerifyUpdate for an Agent

For the same reason it is needed that changes in NetworkHandles are not evaluated during StateMachines of type Agent are in StateMachine State VerifyUpdate

This is only needed for StateMachine State VerifyUpdate and not for PrepareUpdate and PrepareRollback, as the corresponding StateMachine will be stopped after these StateMachine States (see [SWS_SM_00634])

As StateMachine State of StateMachine of type Controller should not change during an "active update session" it is additionally needed, that its StateMachine State does not change when a NmNetworkHandle changes.

[SWS_SM_00628] Evaluation of NetworkHandle changes for StateMachine of type Controller

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈StateMachine of type Controller shall keep its StateMachine State, when the RequestTransition for StateMachine of type Agent returns kUpdateInProgress and changes in a NmNetworkHandle are recognized.⌋

Modelled Process controlling StateMachine of type Controller needs to check the result of all operations needed for verification. For example, if the VerifyUpdate state for StateMachine of type Agent requires a Function Group state transition and that transition is unsuccessful, StateMachine of type Agent should pass this information to theModelled Process controlling StateMachine of type Controller. As mentioned earlier this cooperation is not restricted to the VerifyUpdate. The result of verification should be ultimately passed back to Update and Configuration Management.

[SWS_SM_00639] Unsuccessful verification of updated software

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈If Modelled Process controlling StateMachine of type Controller fails to verify any StateMachine marked as "ImpactedByUpdate", it shall return kOperationFailed error from the VerifyUpdate method.⌋

[SWS_SM_00640] Successful verification of updated software

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When Modelled Process controlling StateMachine of type Controller successfully verifies the StateMachines marked as "ImpactedByUpdate", it shall return success from the VerifyUpdate method.⌋

If verification of the updated software fails, Update and Configuration Management will have to roll back changes. Preparation for rollback is very similar to the

preparation for update, but it uses a separate configuration. Please note, that if processes remain running after a successful verification (see [SWS_SM_00640]) any error after that point in time is no longer relevant to the `Update and Configuration Management`.

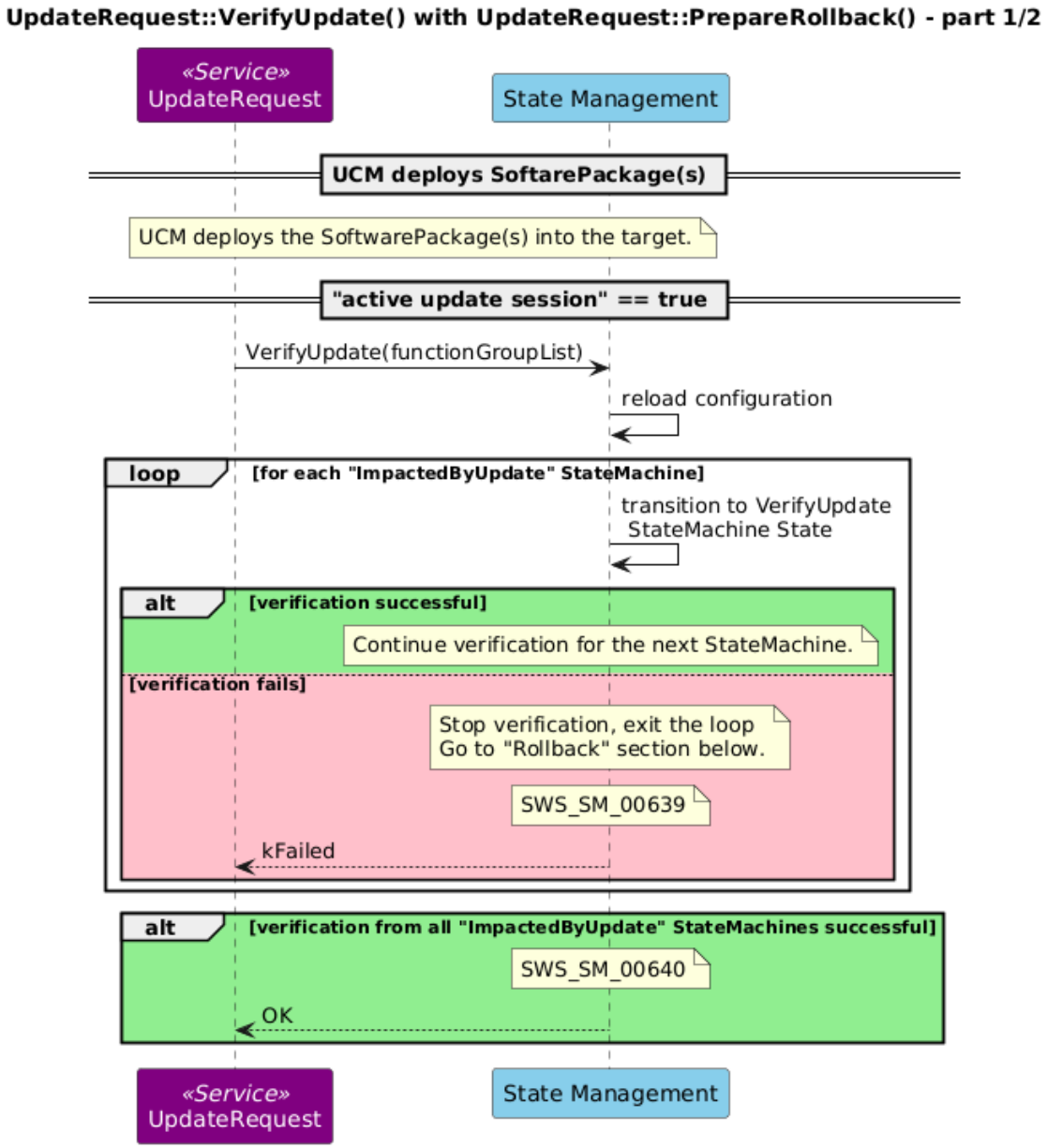

Figure 7.28: Verify and prepare rollback within StateMachine approach - part 1/2

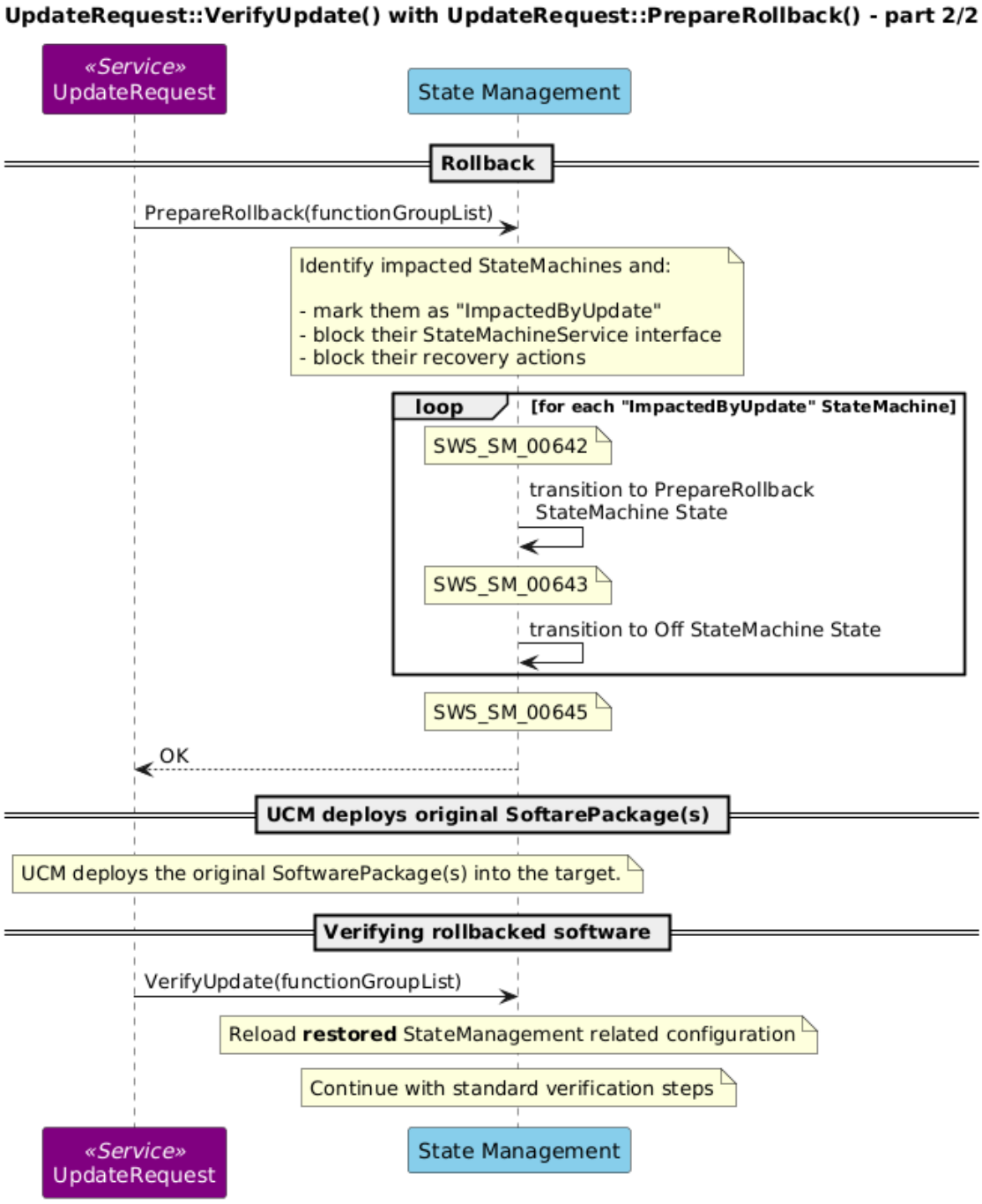

Figure 7.29: Verify and prepare rollback within StateMachine approach - part 2/2

[SWS_SM_CONSTR_00023] Existence of StateMachine PrepareRollback state ⌈Each configured `StateMachine` shall have `PrepareRollback` `StateMachine State` configured, at the time when the creation of the manifest is finished.⌋

After this preparation for rollback can be started.

[SWS_SM_00642] Transition affected StateMachines to PrepareRollback state

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Modelled Process controlling StateMachine of type Controller, during a call to PrepareRollback method, shall transition every affected StateMachine to the PrepareRollback state.⌋

[SWS_SM_00643] Shutdown of affected StateMachines during a call to PrepareRollback method

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Modelled Process controlling StateMachine of type Controller, during a call to PrepareRollback method, shall stop every affected StateMachine of type Agent.⌋

Result of the preparation for rollback should be communicated back to Update and Configuration Management.

[SWS_SM_00644] Failing to prepare for rollback

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈If Modelled Process controlling StateMachine of type Controller fails to prepare for the rollback process, it shall return kOperationFailed error from the PrepareRollback method.⌋

[SWS_SM_00645] Successful preparation for rollback

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When Modelled Process controlling StateMachine of type Controller successfully prepares for rollback, it shall return success from the PrepareRollback method.⌋

As already mentioned in chapter 7.3, a restart of the Machine should be supported during an active update session. Therefore a well defined Controller's StateMachine State shall support a coordinated shutdown of all running Agents, NetworkHandlers and Function Groups as well as ensure the request of the MachineFG Restart to the Execution Management.

[SWS_SM_CONSTR_00029] Existence of StateMachine State Restart for StateMachine of type Controller ⌈The configured StateMachine of type Controller shall have corresponding Restart StateMachine State configured, at the time when the creation of the manifest is finished.⌋

[SWS_SM_CONSTR_00030] Existence of MachineFG Restart in StateMachine State Restart ⌈The ActionList for the configured Restart StateMachine

State of the StateMachine of type Controller [SWS_SM_CONSTR_00029], shall contain an ActionListItem that references MachineFG Restart state.⌋

Please be aware that a project configuration may contain the ActionListItem that references MachineFG Restart state in more than one Controller's StateMachine State. Those StateMachine States may be entered following the Controller's TransitionRequestTable or ErrorRecoveryTable triggered by incoming Triggers or ExecutionErrors. But in the scope of an update session, only the StateMachine State Restart [SWS_SM_CONSTR_00029] is the state which will be processed once the Update and Configuration Management requests the restart of the Machine.

[SWS_SM_00658] Transition to Restart state for StateMachine of type Controller

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Modelled Process controlling StateMachine of type Controller, during a call to ResetMachine method, shall transition the StateMachine of type Controller to the Restart StateMachine State.⌋

[SWS_SM_00661] Set ResetMachineNotifier to kRejected

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈If ResetMachine is called outside an update session Modelled Process controlling StateMachine of type Controller shall set the Field ResetMachineNotifier to kRejected.⌋

Update session ends with a call to StopUpdateSession method. At that point the Machine is in an undefined state. StateMachines may have been installed, updated or removed. Depending on the changes done during the update, the StateMachine States and their ActionLists managed by the StateMachine of type Controller may have changed as well. To counter this situation the StateMachine of type Controller needs to retake full control of the Machine and transit it to a well defined StateMachine State.

[SWS_SM_CONSTR_00027] Existence of StateMachine State AfterUpdate for StateMachine of type Controller ⌈The configured StateMachine of type Controller shall have corresponding AfterUpdate StateMachine State configured, at the time when the creation of the manifest is finished.⌋

The ResetMachineNotifier field will be updated with its default value (see [SWS_SM_00212] at Machine startup. State Management performs initialization of the Field.

To enable the possibility to avoid an execution of processes which might have been changed during an update, it is needed that the StateMachine of type Controller behaves differently on startup after a restart (intended or unintended) of the Machine. Therefore a well defined state, that differs from the Initial State, for the StateMachine of type Controller is needed.

[SWS_SM_CONSTR_00028] Existence of StateMachine State ContinueUpdate
⌈The configured StateMachine of type Controller shall have ContinueUpdate StateMachine State configured, at the time when the creation of the manifest is finished.⌋

[SWS_SM_00657] Transition to StateMachine State ContinueUpdate

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When the StateMachine of type Controller is started [SWS_SM_00648] during an active update session it shall enter ContinueUpdate State instead of Initial State.⌋

Please note that a reset can happen either on request of Update and Configuration Management (see [SWS_SM_00202]) or in an unintended way (e.g. Watchdog reset, power loss, ...).

A different behavior is needed, because State Management is not aware how far Update and Configuration Management proceeded with the update. Therefore only processes should be started which are essential to continue the update.

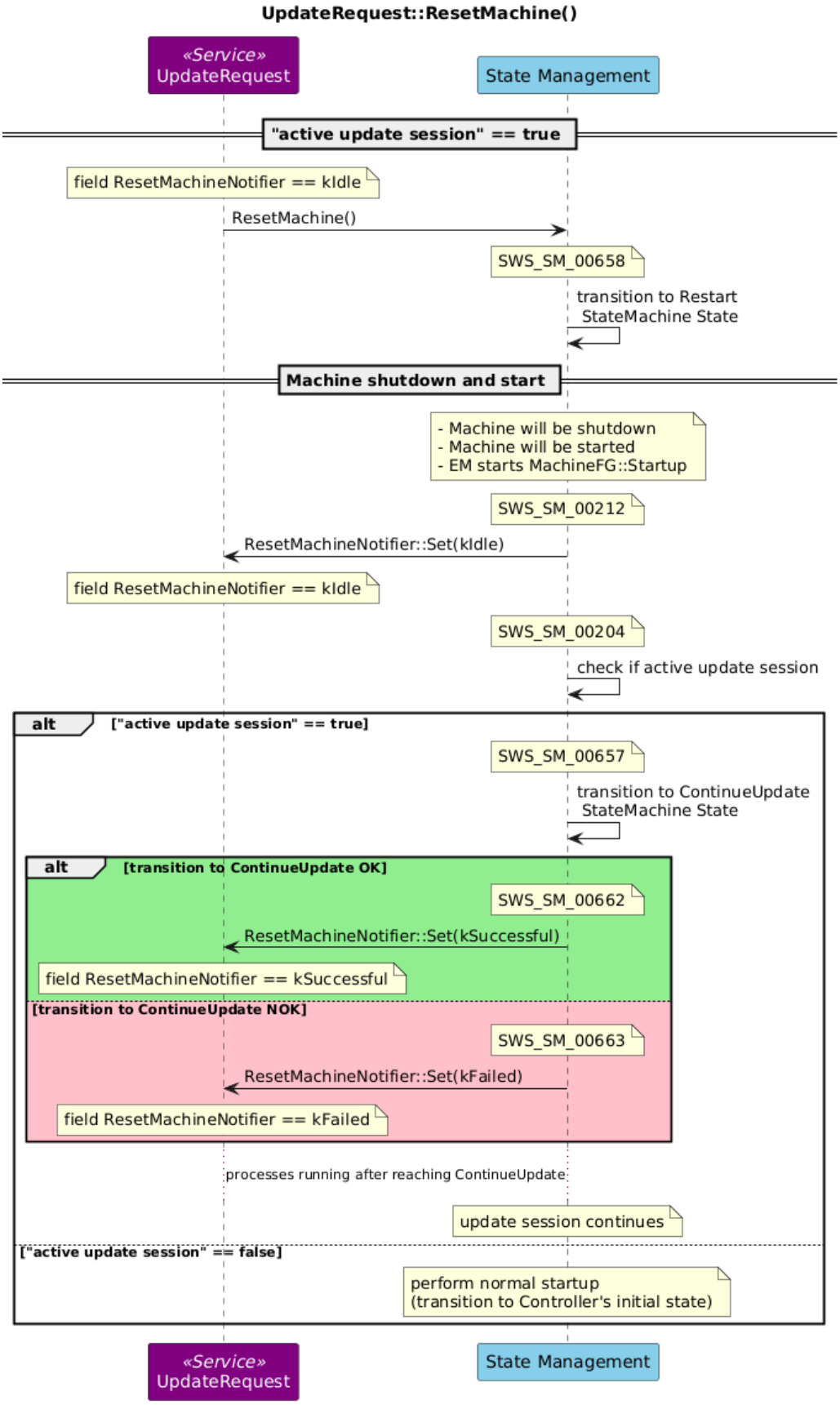

Figure 7.30: Reset machine handling within StateMachine approach

[SWS_SM_00662] Set ResetMachineNotifier to kSuccessful

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Modelled Process controlling StateMachine of type Controller, upon successfully finalizing the ContinueUpdate transition, shall set the Field ResetMachineNotifier to kSuccessful.⌋

[SWS_SM_00663] Set ResetMachineNotifier to kFailed

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Modelled Process controlling StateMachine of type Controller, upon failing to perform the transition to ContinueUpdate, shall set the Field ResetMachineNotifier to kFailed.⌋

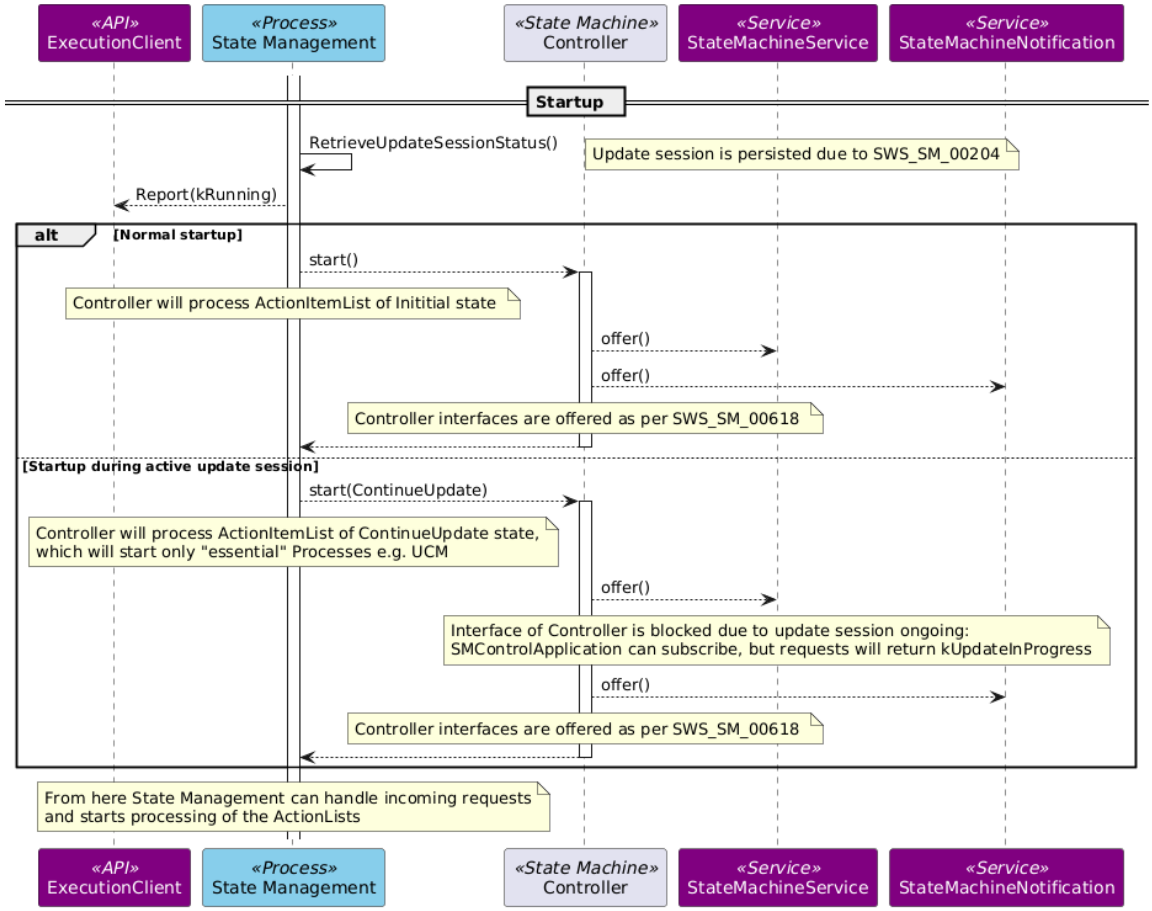

Figure 7.31: Start-up Sequence to Initial State or ContinueUpdate state

There are different - fully project-specific - solutions how to ensure that only the essential parts of the Machine are started. Here are some examples:

- Update and Configuration Management is part of MachineFG (Update and Configuration Management is intended to run in MachineFG::extended State; all other processes should not be changed (Platform Health Management, State Management, ...):

  - normal startup ⇒ Initial State is entered when StateMachine of type Controller starts:
    * MachineFG::Startup
    * SYNC
    * Agent1::start
    * ...
    * SYNC
    * MachineFG::extended ⇒ Update and Configuration Management available
  - startup during update ⇒ ContinueUpdate state is entered when StateMachine of type Controller starts:
    * MachineFG::extended ⇒ Update and Configuration Management available
    * Agent1::stop
    * ...

- Update and Configuration Management is not part of MachineFG (Update and Configuration Management is intended to run in Function Group State controlled by Agent1 (e.g. FG_UCM))
  - normal startup ⇒ Initial State is entered when StateMachine of type Controller starts:
    * MachineFG::Startup
    * SYNC
    * Agent2::start
    * ...
    * Agent1::start ⇒ Initial StateMachine State of Agent1 is entered ⇒ FG_UCM::On ⇒ Update and Configuration Management available
  - startup during update ⇒ ContinueUpdate state is entered when StateMachine of type Controller starts:
    * MachineFG::Startup
    * SYNC
    * Agent1::start ⇒ Initial StateMachine State of Agent1 is entered ⇒ FG_UCM::On ⇒ Update and Configuration Management available

* Agent2::stop
* ...

This kind of configuration is just an example for optimization to show how e.g. Update and Configuration Management could be started late on Machine startup.

[SWS_SM_00646] Transition Controller to AfterUpdate state

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Modelled Process controlling StateMachine of type Controller, upon receiving StopUpdateSession call, shall transition StateMachine of type Controller to the AfterUpdate StateMachine State.⌋

[SWS_SM_00660] Set ResetMachineNotifier to default value when stopping update session

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Modelled Process controlling StateMachine of type Controller, upon receiving StopUpdateSession call during an update session, shall set the Field ResetMachineNotifier to its default value (see [SWS_SM_00212]).⌋

After StopUpdateSession is called, requests to RequestTransition method as well as Recovery Actions will be enabled again as described in [SWS_SM_00656].

[SWS_SM_00647] Enabling RequestTransition method after StopUpdateSession call

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Once StopUpdateSession method has been invoked any call to RequestTransition for StateMachine of type Controller shall not return kUpdateInProgress any longer.⌋

[SWS_SM_00656] Unmark "ImpactedByUpdate" from StateMachine

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈Once the StopUpdateSession method has been invoked any StateMachine marked as "ImpactedByUpdate" shall be unmarked.⌋

When call to StopUpdateSession method ends, the update session is considered to be finished.

7.7 Functional cluster life-cycle

7.7.1 Startup

Execution Management will be controlled by State Management and therefore it should not execute any Function Group State changes on its own. This creates some expectations towards system configuration. The configuration shall be done in this way that State Management will run in every Machine State (this includes Startup, Shutdown and Restart). Above expectation is needed in order to ensure that there is always a software entity that can introduce changes in the current state of the Machine. If (for example) system integrator doesn't configure State Management to be started in Startup Machine State, then Machine will never be able transit to any other state and will be stuck forever in it. This also applies to any other Machine State state that doesn't have State Management configured.

As State Management might be supervised by Platform Health Management it might be needed to run Platform Health Management before State Management as part of Startup of Machine State. Additionally Some/IP and logging has to be available before State Management is started, as it is needed for execution. As soon as any Adaptive Application is interacting with State Management it has to call ara::core initialize before. During startup of State Management the state of ongoing update has to be recovered from ara::per, to ensure a correct sequence of update and to ensure that no Process is started, which might interfere with Update and Configuration Management

7.7.2 Shutdown

As mentioned in Section 7.7.1 AUTOSAR assumes that State Management will be configured to run in Shutdown. State transition is not a trivial system change and it can fail for a number of reasons. When ever this happens you may want State Management to be still alive, so you can report an error and wait for further instructions. Please note that the very purpose of this state is to shutdown Machine (this includes State Management) in a clean manner. Unfortunately this means that at some point State Management will no longer be available and it will not be able to report errors anymore. Those errors will be handled in a implementation specific way. At least it is assumed that State Management will run in every Machine State including shutdown. This means that there are only very rare cases, where State Management should react on SIGTERM from Execution Management. This depends at lest for StateMachine approach on configuration of Machine State. So on reception of SIGTERM State Management should terminate gracefully. It is expected that every SMControlApplication will terminate before State Management receives SIGTERM. Therefore each SMControlApplication should call ara::core::deinitialize before terminating. Platform Health Management, Some/IP and logging should be terminated after State Management has received SIGTERM, thus all dependencies are still fulfilled even in case of shutdown.

7.7.3 Restart

As mentioned in Section 7.7.1 AUTOSAR assumes that State Management will be configured to run in Machine State Restart. The reasons for doing so are the same as for Section 7.7.2. Only difference to shutdown is, that the Machine is being restarted instead of being just shutdown.

7.7.4 Daemon crash

The chapter shall define the behavior of the State Management in case the daemon crashes. As State Management is the central entity within a Machine the complete Machine becomes unusable. Therefore State Management should be supervised in terms of checkpoints by Platform Health Management. When Platform Health Management might trigger watchdog reaction, when Platform Health Management detects State Management to misbehave/being crashed.

7.8 Reporting

7.8.1 Security Events

Up to now no security events are defined for State Management.

7.8.2 Log Messages

Up to now no log messages are defined for State Management

7.8.3 Violation Messages

Up to now no violation messages are defined for State Management

7.8.4 Production Errors

Up to now no production are defined for State Management

8 API specification

State Management does not provide any API. All functional interfaces will be found in Section 9 Service Interfaces.

9 Service Interfaces

9.1 Implementation Data Types

9.1.1 Data types for Update And Configuration Management interaction

[SWS_SM_91018] Definition of ImplementationDataType FunctionGroupListType

Upstream requirements: RS_SM_00004, RS_AP_00150, RS_AP_00122

⌈

Name	FunctionGroupListType
Namespace	ara::sm
Kind	VECTOR <FunctionGroupNameType>
Derived from	-
Description	A list of FunctionGroups.

⌋

[SWS_SM_91019] Definition of ImplementationDataType FunctionGroupNameType

Upstream requirements: RS_SM_00004, RS_AP_00150, RS_AP_00122

⌈

Name	FunctionGroupNameType
Namespace	ara::sm
Kind	STRING
Derived from	-
Description	full qualified FunctionGroup shortName.

⌋

9.1.2 Data types for StateMachine interaction

[SWS_SM_91023] Definition of ImplementationDataType TransitionRequestType

Upstream requirements: RS_SM_00004, RS_SM_00001, RS_AP_00150

⌈

Name	TransitionRequestType
Namespace	ara::sm
Kind	TYPE_REFERENCE
Derived from	`uint32_t`
Description	A value which represents the TransitionRequest value to be used in the TransitionRequest Table.

⌋

9.1.3 Data types for StateMachine notification

[SWS_SM_91020] Definition of ImplementationDataType StateMachineStateNameType

Upstream requirements: RS_SM_00004, RS_AP_00150, RS_AP_00122

⌈

Name	StateMachineStateNameType
Namespace	ara::sm
Kind	STRING
Derived from	-
Description	A data type used to represent the name of the StateMachine State. For more details see [SWS_SM_91019].

⌋

9.1.4 Data types for UpdateAllowed service interface

[SWS_SM_91026] Definition of ImplementationDataType UpdateAllowedType

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Name	UpdateAllowedType	
Namespace	ara::sm	
Kind	TYPE_REFERENCE	
Derived from	`uint32_t`	
Description	UpdateAllowedType	
Range / Symbol	***Limit***	***Description***
kUpdateAllowed		kUpdateAllowed
kUpdateNotAllowed		kUpdateNotAllowed

⌋

9.1.5 Data types for ResetMachineNotifier

[SWS_SM_91027] Definition of ImplementationDataType UpdateStatusType

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Name	UpdateStatusType	
Namespace	ara::sm	
Kind	TYPE_REFERENCE	
Derived from	`uint32_t`	
Description	Defines the current state of the operation requested through the UpdateRequest service.	
Range / Symbol	***Limit***	***Description***
kIdle		no request was performed
kRejected		operation was requested outside of the update session
kSuccessful		the processing associated with the request successfully finished
kFailed		the processing associated with the request failed

⌋

9.2 Provided Service Interfaces

9.2.1 UpdateRequest

The `UpdateRequest` interface is intended to be used by `Update and Configuration Management` to interact with `State Management` to perform updates (including installation and removal) of `Software Clusters`.

Port

[SWS_SM_91016] Definition of Port UpdateRequest provided by functional cluster SM

Upstream requirements: RS_SM_00001, RS_SM_00004, RS_AP_00150

⌈

Name	UpdateRequest		
Kind	ProvidedPort	***Interface***	`UpdateRequest`
Description	To be used by Update And Configuration Management to request State Management to perform steps for updating SoftwareClusters.		
Variation			

⌋

Service Interface

[SWS_SM_91017] Definition of ServiceInterface UpdateRequest

Upstream requirements: RS_SM_00001, RS_SM_00004, RS_AP_00150, RS_AP_00115, RS_AP_00120, RS_AP_00142, RS_AP_00119, RS_AP_00121

⌈

Name	UpdateRequest
Namespace	ara::sm
Version	1.0
Fields	`ResetMachineNotifier`
Methods	• `ResetMachine` • `StopUpdateSession` • `RequestUpdateSession` • `PrepareUpdate` • `VerifyUpdate` • `PrepareRollback`

⌋

[SWS_SM_91106] Definition of Field UpdateRequest.ResetMachineNotifier

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Field	ResetMachineNotifier
Description	To be set by State Management to inform UCM about changes during and after processing the method ResetMachine().
Version	1.0
Type	`UpdateStatusType`
HasGetter	true
HasNotifier	true
HasSetter	false
Enclosing Service Interface	`UpdateRequest`

⌋

[SWS_SM_91100] Definition of Method UpdateRequest.ResetMachine

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Method	ResetMachine	
Description	Requests a reset of the machine. Before the reset is performed all information within the machine shall be persisted. Request will be rejected when RequestUpdateSession was not called successfully before.	
Version	1.0	
FireAndForget	true	
Application Errors	`kOperationRejected`	Requested operation was rejected due to State Managements/machines internal state.
Enclosing Service Interface	`UpdateRequest`	

⌋

[SWS_SM_91101] Definition of Method UpdateRequest.StopUpdateSession

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Method	StopUpdateSession	
Description	Has to be called by Update And Configuration Management once the update is finished to let State Management know that the update is done and the Machine is in a stable state. Request will be rejected when RequestUpdateSession was not called successfully before.	
Version	1.0	
FireAndForget	false	
Application Errors	`kOperationRejected`	Requested operation was rejected due to State Managements/machines internal state.

▽

△

Enclosing Service Interface	UpdateRequest

⌋

[SWS_SM_91102] Definition of Method UpdateRequest.RequestUpdateSession

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Method	RequestUpdateSession	
Description	Has to be called by Update And Configuration Management once it has to start interaction with State Management. State Management might decline this request when machine is not in a state to be updated.	
Version	1.0	
FireAndForget	false	
Application Errors	kOperationRejected	Requested operation was rejected due to State Managements/machines internal state.
Application Errors	kNotAllowedMultipleUpdateSessions	Request for new session was rejected as only single active (update) session is allowed.
Enclosing Service Interface	UpdateRequest	

⌋

[SWS_SM_91103] Definition of Method UpdateRequest.PrepareUpdate

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Method	PrepareUpdate	
Description	Has to be called by Update And Configuration Management after State Management allowed to update. State Management will decline this request when RequestUpdateSession was not called before successfully.	
Version	1.0	
FireAndForget	false	
Parameter	functionGroupList	
	Description	The list of FunctionGroups within the SoftwareCluster to be prepared to be updated.
	Type	FunctionGroupListType
	Variation	
	Direction	IN
Application Errors	kOperationRejected	Requested operation was rejected due to State Managements/machines internal state.
Application Errors	kOperationFailed	Requested operation failed.

▽

△

Enclosing Service Interface	UpdateRequest

⌋

[SWS_SM_91104] Definition of Method UpdateRequest.VerifyUpdate

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Method	VerifyUpdate	
Description	Has to be called by Update And Configuration Management after State Management allowed to update and the update preparation has been done. State Management will decline this request when Prepare Update was not called before successfully.	
Version	1.0	
FireAndForget	false	
Parameter	functionGroupList	
	Description	The list of FunctionGroups within the SoftwareCluster to be verified.
	Type	FunctionGroupListType
	Variation	
	Direction	IN
Application Errors	kOperationRejected	Requested operation was rejected due to State Managements/machines internal state.
Application Errors	kOperationFailed	Requested operation failed.
Enclosing Service Interface	UpdateRequest	

⌋

[SWS_SM_91105] Definition of Method UpdateRequest.PrepareRollback

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Method	PrepareRollback	
Description	Has to be called by Update And Configuration Management after State Management allowed to update.	
Version	1.0	
FireAndForget	false	
Parameter	functionGroupList	
	Description	The list of FunctionGroups within the SoftwareCluster to be prepared to roll back.
	Type	FunctionGroupListType
	Variation	
	Direction	IN
Application Errors	kOperationRejected	Requested operation was rejected due to State Managements/machines internal state.

▽

△

Application Errors	kOperationFailed	Requested operation failed.
Enclosing Service Interface	UpdateRequest	

⌋

9.2.2 StateMachine service

The StateMachineService interface is intended to be used by SMControlApplication to interact with State Management's StateMachine to request StateMachine State changes.

Port

[SWS_SM_91021] Definition of Port StateMachineService provided by functional cluster SM

Upstream requirements: RS_SM_00001, RS_SM_00004, RS_AP_00150

⌈

Name	StateMachineService		
Kind	ProvidedPort	***Interface***	StateMachineService
Description	To be used by SMControlApplications to request a change in the referenced StateMachine.		
Variation			

⌋

Service Interface

[SWS_SM_91022] Definition of ServiceInterface StateMachineService

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Name	StateMachineService
Namespace	ara::sm
Version	1.0
Methods	RequestTransition

⌋

[SWS_SM_91107] Definition of Method StateMachineService.RequestTransition

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Method	RequestTransition
Description	Has to be called by a SMControlApplication to request a change in the referenced StateMachine.
Version	1.0
FireAndForget	false
Parameter	TransitionRequest

▽

△

	Description	Represents the value to be used as TransitionRequest value in the Transition RequestTable.
	Type	`TransitionRequestType`
	Variation	
	Direction	IN
Application Errors	`kInvalid-Value`	The provided value is not mapped to any transition.
Application Errors	`kTransition-NotAllowed`	Requested transition is not possible from current StateMachine state.
Application Errors	`kRecovery-Transi-tionOngoing`	Request will not be carried out, because currently recovery is ongoing.
Application Errors	`kTransition-Failed`	During transition to the requested state an error occurred.
Application Errors	`kOpera-tionCanceled`	The request was replaced by a newer one and therefore it was cancelled
Application Errors	`kUpdateIn-Progress`	Requested operation is not allowed as update session is in progress.
Enclosing Service Interface	`StateMachineService`	

⌋

Service Interface

[SWS_SM_91028] Definition of ServiceInterface StateMachineNotification

Upstream requirements: RS_SM_00001, RS_SM_00004, RS_AP_00150, RS_AP_00115, RS_AP_00120, RS_AP_00142, RS_AP_00119, RS_AP_00121

⌈

Name	StateMachineNotification
Namespace	ara::sm
Version	1.0
Fields	`CurrentState`

⌋

[SWS_SM_91109] Definition of Field StateMachineNotification.CurrentState

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Field	CurrentState
Description	This field represents the current state of StateMachine. If StateMachine is currently in transition between two different states, then the value of this field is set to "InTransition". Adaptive Applications can use this field for notifications if they are interested in state changes of a particular StateMachine.
Version	1.0
Type	`StateMachineStateNameType`

▽

△

HasGetter	true
HasNotifier	true
HasSetter	false
Enclosing Service Interface	StateMachineNotification

⌋

9.2.3 StateMachine UpdateAllowed service

The UpdateAllowedService interface is intended to be used by SMControlApplication to interact with State Management's Controller. Content of the field will be used to grant update session or not.

Port

[SWS_SM_91024] Definition of Port UpdateAllowedService provided by functional cluster SM

Upstream requirements: RS_SM_00001, RS_SM_00004, RS_AP_00150

⌈

Name	UpdateAllowedService		
Kind	ProvidedPort	***Interface***	UpdateAllowedService
Description	To be used by SMControlApplications to allow or deny update session.		
Variation			

⌋

Service Interface

[SWS_SM_91025] Definition of ServiceInterface UpdateAllowedService

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Name	UpdateAllowedService
Namespace	ara::sm
Version	1.0
Fields	UpdateAllowed

⌋

[SWS_SM_91108] Definition of Field UpdateAllowedService.UpdateAllowed

Upstream requirements: RS_SM_00001, RS_SM_00004

⌈

Field	UpdateAllowed
Description	to be set by SMControlApplication to signal if update is allowed or not
Version	1.0
Type	UpdateAllowedType
HasGetter	true
HasNotifier	true
HasSetter	true

▽

△

Enclosing Service Interface	UpdateAllowedService

⌋

9.3 Required Service Interfaces

No required interfaces

9.4 Application Errors

This chapter lists all errors of State Management

9.4.1 StateManagement Error Domain

[SWS_SM_91010] Definition of Application Error Domain of functional cluster SM

Upstream requirements: RS_SM_00004, RS_AP_00150, RS_AP_00125, RS_AP_00142, RS_AP_00119, RS_AP_00149

⌈

Name	*Code*	*Description*
kInvalidValue	10	The provided value is not mapped to any transition.
kNotAllowedMultipleUpdateSessions	9	Request for new session was rejected as only single active (update) session is allowed.
kOperationCanceled	14	The request was replaced by a newer one and therefore it was cancelled
kOperationFailed	6	Requested operation failed.
kOperationRejected	5	Requested operation was rejected due to State Managements/ machines internal state.
kRecoveryTransitionOngoing	12	Request will not be carried out, because currently recovery is ongoing.
kTransitionFailed	13	During transition to the requested state an error occurred.
kTransitionNotAllowed	11	Requested transition is not possible from current StateMachine state.
kUpdateInProgress	15	Requested operation is not allowed as update session is in progress.

⌋

10 Configuration

The configuration structure of State Management (only valid for StateMachine approach) is described in TPS_Manifest.

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 TPS_Manifest.

There are no default values defined for State Management.

10.2 Semantic Constraints

This section defines semantic constraints for the configuration elements of State Management defined in TPS_Manifest.

State Management should be configured to run in every Machine State (this includes Startup, Shutdown and Restart) other than Off. This expectation is needed to ensure that there is always a software entity that can introduce changes in the current state of the Machine. If (for example) the system integrator does not configure State Management to be started in Startup Machine State, then Machine will never be able transit to any other state and will be stuck forever in it.

[SWS_SM_CONSTR_00001] Existence of State Management ⌈At least one Modelled Process with Process.functionClusterAffinity with the value STATE_MANAGEMENT shall be configured to run in each MachineFG state except Off, whenever one such Modelled Process is configured to run in MachineFG state Startup.⌋

[SWS_SM_CONSTR_00033] Configurable Namespace ⌈Configurable Namespace for StateManagement StateManagementPortInterface.namespace shall never exist.⌋

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.

Class	***Identifiable*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*
Subclasses	ARPackage, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractFunctionalClusterDesign*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstance Filter*, *AbstractServiceInstance*, *AbstractSignalBasedToISignalTriggeringMapping*, AdaptiveSwcInternal Behavior, ApApplicationEndpoint, *ApmcAbstractDefinition*, *ApmcConfigurationElementDef*, *Apmc ContainerElementValue*, ApmcContainerValue, ApmcEnumerationLiteralDef, ApplicationEndpoint, ApplicationError, AppliedStandard, ArtifactChecksum, ArtifactLocator, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BuildAction Entity*, BuildActionEnvironment, Chapter, CheckpointTransition, ClassContentConditional, ClientId Definition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *Comm ConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, *CouplingPortAbstractShaper*, *CouplingPortStructuralElement*, CryptoCertificate, CryptoKeySlot, CryptoKeySlotDesign, CryptoKeySlotUsageDesign, CryptoProvider, *CryptoServiceMapping*, DataPrototypeGroup, DataPrototypeTransformationPropsIdent, Data Transformation, DdsCpDomain, DdsCpPartition, DdsCpQosProfile, DdsCpTopic, DdsDomainRange, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticAuthTransmitCertificateEvaluation, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, Diagnostic FunctionInhibitSource, DiagnosticParameterElement, *DiagnosticRoutineSubfunction*, DiagnosticSovd MethodPrimitive, DltApplication, DltArgument, DltMessage, DoIpInterface, DoIpLogicAddress, DoIp LogicalAddress, DoIpNetworkConfigurationDesign, DoIpRoutingActivation, E2EProfileConfiguration, End2EndEventProtectionProps, End2EndMethodProtectionProps, EndToEndProtection, Ethernet WakeupSleepOnDatalineConfig, EventHandler, EventMapping, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMap Element, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FieldMapping, FireAndForget MethodMapping, FlexrayArTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, Global Supervision, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HealthChannel*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, *IEEE1722TpAcfBus*, *IEEE1722TpAcfBus Part*, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, ImpositionTime, InternalTriggeringPoint, Keyword, LifeCycleState, Linker, MacAddressVlanMembership, MacMulticastGroup, MacSecKayParticipant, McDataInstance, MemorySection, MemoryUsage, Method Mapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, *NmNode*, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PersistencyDeploymentElement*, *PersistencyInterfaceElement*, *Phm Supervision*, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKeySlotMapping, ResourceConsumption, ResourceGroup, RootSwClusterDesignComponentPrototype, RootSwComponentPrototype, RootSwComposition Prototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecution Context, RptProfile, RptServicePoint, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecOcJobMapping, SecOcJobRequirement, SecureCommunicationAuthenticationProps, *SecureCommunicationDeployment*, SecureCommunicationFreshnessProps, SecurityEventContextDataElement, SecurityEventContextProps, *ServiceEventDeployment*, *ServiceFieldDeployment*, ServiceInterfaceElementSecureComConfig, *Service MethodDeployment*, *ServiceNeeds*, SignalServiceTranslationEventProps, SignalServiceTranslation Props, SocketAddress, SoftwarePackageStep, SomeipEventGroup, SomeipProvidedEventGroup, SomeipTpChannel, *SpecElementReference*, *StackUsage*, *StateManagementActionItem*, State ManagementActionList, StateManagementStateNotification, *StateManagementStateRequest*, Static SocketConnection, StructuredReq, SupervisionCheckpoint, SupervisionMode, SupervisionMode Condition, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SwitchAsynchronous TrafficShaperGroupEntry, SystemMapping, *TimeBaseResource*, *TimingClock*, TimingClockSync Accuracy, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, Timing ▽

▽

△

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

Table A.1: Identifiable

Class	ModeDeclaration			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	Declaration of one Mode. The name and semantics of a specific mode is not defined in the meta-model.			
Base	*ARObject*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, ModeDeclarationGroup.modeDeclaration			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
value	PositiveInteger	0..1	attr	The RTE shall take the value of this attribute for generating the source code representation of this Mode Declaration.

Table A.2: ModeDeclaration

Class	ModeDeclarationGroup			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	A collection of Mode Declarations. Also, the initial mode is explicitly identified. **Tags:** atp.recommendedPackage=ModeDeclarationGroups			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
initialMode	ModeDeclaration	0..1	ref	The initial mode of the ModeDeclarationGroup. This mode is active before any mode switches occurred.
mode Declaration	ModeDeclaration	*	aggr	The ModeDeclarations collected in this ModeDeclaration Group. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=modeDeclaration.shortName, mode Declaration.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime

Table A.3: ModeDeclarationGroup

Class	ModeDeclarationGroupPrototype			
Package	M2::AUTOSARTemplates::CommonStructure::ModeDeclaration			
Note	The ModeDeclarationGroupPrototype specifies a set of Modes (ModeDeclarationGroup) which is provided or required in the given context.			
Base	*ARObject*, *AtpFeature*, *AtpPrototype*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, BswModuleDescription.providedModeGroup, BswModuleDescription.requiredModeGroup, FirewallStateSwitchInterface.firewallStateMachine, FunctionGroupSet.functionGroup, ModeSwitchInterface.modeGroup, Process.processStateMachine, StateManagementStateNotification.stateMachine			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
type	ModeDeclarationGroup	0..1	tref	The "collection of ModeDeclarations" (= ModeDeclarationGroup) supported by a component **Stereotypes:** isOfType

Table A.4: ModeDeclarationGroupPrototype

Class	**NmInteractsWithSmMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			
Note	This mapping represents an interaction from network management to state management. **Tags:** atp.Status=draft atp.recommendedPackage=FCInteractions			
Base	*ARElement*, *ARObject*, *CollectableElement*, *FunctionalClusterInteractsWithFunctionalClusterMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeployment Element*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
nmNetwork Handle	NmNetworkHandle	0..1	ref	This reference identifies the network management handle that wants to interact with state management. **Tags:** atp.Status=draft
stateRequest	StateManagementState Request	0..1	ref	This reference identifies the state management state request that is involved in the interaction with the network management. **Tags:** atp.Status=draft

Table A.5: NmInteractsWithSmMapping

Class	**NmNetworkHandle**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::AdaptiveModule Implementation			
Note	Group of partialNetworks and/or VLANs that can be controlled collectively.			
Base	*ARObject*, *Referrable*			
Aggregated by	NmInstantiation.networkHandle			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
partialNetwork	PncMappingIdent	*	ref	Reference to a Partial Network that is included in the Nm NetworkHandle. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=partialNetwork
vlan	EthernetCommunication Connector	*	ref	Reference to a VLAN that is included in the NmNetwork Handle.

Table A.6: NmNetworkHandle

Enumeration	**NmStateRequestEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement
Note	This enumeration defines the description of states that can be requested from the network management. **Tags:** atp.Status=draft
Aggregated by	StateManagementNmActionItem.nmStateRequest
Literal	***Description***
fullCom	This literal represents that case that full communication should be possible. **Tags:** atp.EnumerationLiteralIndex=1 atp.Status=draft

▽

△

Enumeration	**NmStateRequestEnum**
noCom	This literal represents that case that no communication should be possible. **Tags:** atp.EnumerationLiteralIndex=0 atp.Status=draft

Table A.7: NmStateRequestEnum

Class	***PortInterface*** (abstract)			
Package	M2::AUTOSARTemplates::SWComponentTemplate::PortInterface			
Note	Abstract base class for an interface that is either provided or required by a port of a software component.			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*			
Subclasses	*AbstractRawDataStreamInterface*, *AbstractSynchronizedTimeBaseInterface*, ClientServerInterface, *CryptoInterface*, *DataInterface*, *DiagnosticPortInterface*, FirewallStateSwitchInterface, *IdsmAbstractPortInterface*, LogAndTraceInterface, ModeSwitchInterface, NetworkManagementPortInterface, *PersistencyInterface*, *PlatformHealthManagementInterface*, ServiceInterface, *StateManagementPortInterface*, TriggerInterface			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
namespace (ordered)	SymbolProps	*	aggr	This represents the SymbolProps used for the definition of a hierarchical namespace applicable for the generation of code artifacts out of the definition of a ServiceInterface. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=namespace.shortName

Table A.8: PortInterface

Class	**ProcessExecutionError**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ExecutionManifest			
Note	This meta-class has the ability to describe the value of a execution error along with a documentation of its semantics. **Tags:** atp.recommendedPackage=ProcessExecutionErrors			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
executionError	PositiveInteger	0..1	attr	This attribute defines the numeric value which Execution Management and Platform Health Management reports to State Management if the Process terminates unexpectedly or violates its supervision. It shall give further error information for error recovery.

Table A.9: ProcessExecutionError

Class	**ServiceInterface**
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface
Note	This represents the ability to define a PortInterface that consists of a heterogeneous collection of methods, events and fields. **Tags:** atp.recommendedPackage=ServiceInterfaces

▽

△

Class	**ServiceInterface**			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
event	VariableDataPrototype	*	aggr	This represents the collection of events defined in the context of a ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=event.shortName, event.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=30
field	Field	*	aggr	This represents the collection of fields defined in the context of a ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=field.shortName, field.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=40
majorVersion	PositiveInteger	0..1	attr	Major version of the service contract. **Tags:** xml.sequenceOffset=10
method	ClientServerOperation	*	aggr	This represents the collection of methods defined in the context of a ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=method.shortName, method.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=50
minorVersion	PositiveInteger	0..1	attr	Minor version of the service contract. **Tags:** xml.sequenceOffset=20
trigger	Trigger	*	aggr	This represents the collection of triggers defined in the context of a ServiceInterface. **Stereotypes:** atpSplitable; atpVariation **Tags:** atp.Splitkey=trigger.shortName, trigger.variationPoint.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=60

Table A.10: ServiceInterface

Class	**SmInteractsWithNmMapping**
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement
Note	This mapping represents an interaction from state management to network management. **Tags:** atp.Status=draft atp.recommendedPackage=FCInteractions
Base	*ARElement*, *ARObject*, *CollectableElement*, *FunctionalClusterInteractsWithFunctionalClusterMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*
Aggregated by	ARPackage.element

▽

△

Class	SmInteractsWithNmMapping			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
actionItem	StateManagementNm ActionItem	0..1	ref	This reference identifies the action item with which the state management wants to interact with network management. **Tags:** atp.Status=draft
nmNetwork Handle	NmNetworkHandle	0..1	ref	This reference identifies the network management handle that is affected by the interaction with the state management. **Tags:** atp.Status=draft

Table A.11: SmInteractsWithNmMapping

Class	***StateManagementActionItem*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			
Note	This meta-class represents an action item that is executed in response to a state change. **Tags:** atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	StateManagementNmActionItem, StateManagementSetFunctionGroupStateActionItem, State ManagementSleepActionItem, StateManagementStateMachineActionItem, StateManagementSync ActionItem			
Aggregated by	StateManagementActionList.actionItem			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.12: StateManagementActionItem

Class	StateManagementActionList			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			
Note	This meta-class represents the ability to define an action list that is associated with a state of a state machine. **Tags:** atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	StateManagementModuleInstantiation.actionItemList			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
actionItem (ordered)	StateManagement ActionItem	*	aggr	This represents the collection of action items in the context of the action item list. **Tags:** atp.Status=draft
affectedState	ModeDeclaration	0..1	iref	This reference identifies the state for which the referencing action list applies. **Tags:** atp.Status=draft **InstanceRef implemented by:** ModeDeclarationInState ManagementStateNotificationInstanceRef

Table A.13: StateManagementActionList

Class	StateManagementErrorCompareRule			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			
Note	This meta-class represents the configuration of a compare rule for the processing of an error submission. **Tags:** atp.Status=draft			
Base	*ARObject*, *StateManagementCompareCondition*, *StateManagementCompareFormulaPart*			
Aggregated by	StateManagementCompareFormula.part			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table A.14: StateManagementErrorCompareRule

Class	StateManagementNmActionItem			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			
Note	This meta-class represents a state management action item to interact with the network management. **Tags:** atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StateManagementActionItem*			
Aggregated by	StateManagementActionList.actionItem			
Attribute	Type	Mult.	Kind	Note
nmState Request	NmStateRequestEnum	0..1	attr	This attribute defines the target network management state that is requested by state management. **Tags:** atp.Status=draft

Table A.15: StateManagementNmActionItem

Class	***StateManagementPortInterface*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This abstract class acts as a base class for PortInterfaces that are used in the context of state management on the AUTOSAR adaptive platform. **Tags:** atp.Status=draft			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Subclasses	*StateManagementNotificationInterface*, *StateManagementRequestInterface*			
Aggregated by	ARPackage.element			
Attribute	Type	Mult.	Kind	Note
–	–	–	–	–

Table A.16: StateManagementPortInterface

Class	StateManagementSetFunctionGroupStateActionItem			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			
Note	This meta-class represents a state management action item to set a specific state in a specific function group. **Tags:** atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StateManagementActionItem*			
Aggregated by	StateManagementActionList.actionItem			
Attribute	Type	Mult.	Kind	Note

▽

△

Class	**StateManagementSetFunctionGroupStateActionItem**			
rPortPrototype	RPortPrototype	0..1	iref	This reference identifies the PortPrototype over which the function group state switch shall be communicated. **Tags:** atp.Status=draft **InstanceRef implemented by:** RPortPrototypeIn ExecutableInstanceRef
setFunction GroupState	ModeDeclaration	0..1	iref	This reference identifies the funtion group step that shall become active after the action step terminates. **InstanceRef implemented by:** FunctionGroupStateIn FunctionGroupSetInstanceRef

Table A.17: StateManagementSetFunctionGroupStateActionItem

Class	**StateManagementSleepActionItem**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			
Note	This action item can be used to universally implement afterrun. One specific use case for afterrun comes up in the context of network management. **Tags:** atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StateManagementActionItem*			
Aggregated by	StateManagementActionList.actionItem			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
sleepTime	TimeValue	0..1	attr	This attribute represents the amount of time that the execution of the StateManagementActionItemList is supposed to go to sleep. **Tags:** atp.Status=draft

Table A.18: StateManagementSleepActionItem

Class	**StateManagementStateMachineActionItem**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			
Note	This meta-class represents a state management action item to start or stop a state machine. **Tags:** atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StateManagementActionItem*			
Aggregated by	StateManagementActionList.actionItem			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
overrideInitial State	ModeDeclaration	0..1	iref	The referenced ModeDeclaration shall be considered the initial state of the context ModeDeclarationGroup Prototype and the corresponding reference Mode DeclarationGroup.initialMode shall be ignored. **Tags:** atp.Status=draft **InstanceRef implemented by:** ModeDeclarationInState ManagementStateNotificationInstanceRef
startState Machine	ModeDeclarationGroup Prototype	0..1	ref	This reference identifies the state machine that shall be started when the enclosing action list item is executed. **Tags:** atp.Status=draft
stopState Machine	ModeDeclarationGroup Prototype	0..1	ref	This reference identifies the state machine that shall be stopped when the enclosing action list item is executed. **Tags:** atp.Status=draft

Table A.19: StateManagementStateMachineActionItem

Class	**StateManagementStateNotification**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			
Note	This meta-class represents the ability to formalize state notifications on the AUTOSAR adaptive platform. **Tags:** atp.Status=draft			
Base	*ARObject*, *AtpClassifier*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Aggregated by	StateManagementModuleInstantiation.notification			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
notificationPort	PPortPrototype	0..1	iref	This instanceRef identifies the PPortPrototype over which the notification is to be conveyed. **Tags:** atp.Status=draft **InstanceRef implemented by:** PPortPrototypeIn ExecutableInstanceRef
stateMachine	ModeDeclarationGroup Prototype	0..1	aggr	This aggregation represents the existence of an actual state machine. **Tags:** atp.Status=draft

Table A.20: StateManagementStateNotification

Class	***StateManagementStateRequest*** (abstract)			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			
Note	This abstract class serves as the base class for state requests on the AUTOSAR adaptive platform. **Tags:** atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	StateManagementRequestError, StateManagementRequestTrigger			
Aggregated by	StateManagementModuleInstantiation.request			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
stateRequest Port	RPortPrototype	0..1	iref	This represents the RPortPrototype in the application software that is issuing the request for state change. **Tags:** atp.Status=draft **InstanceRef implemented by:** RPortPrototypeIn ExecutableInstanceRef

Table A.21: StateManagementStateRequest

Class	**StateManagementSyncActionItem**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			
Note	This meta-class represents a state management action item to synchronize state machines. **Tags:** atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StateManagementActionItem*			
Aggregated by	StateManagementActionList.actionItem			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.22: StateManagementSyncActionItem

Class	**StateManagementTriggerCompareRule**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			
Note	This meta-class represents the configuration of a compare rule for the processing of a trigger request. **Tags:** atp.Status=draft			
Base	*ARObject*, *StateManagementCompareCondition*, *StateManagementCompareFormulaPart*			
Aggregated by	StateManagementCompareFormula.part			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
assumed CurrentState	ModeDeclaration	0..1	iref	This reference denotes the assumed current state for the given compare rule for trigger values. **Tags:** atp.Status=draft **InstanceRef implemented by:** ModeDeclarationInStateManagementStateNotificationInstanceRef

Table A.23: StateManagementTriggerCompareRule

B Demands and constraints on Base Software (normative)

There are no special demands of State Management for the Base Software on which the AUTOSAR Adaptive Platform is running on.

C Platform Extension Interfaces (normative)

There are currently no extensions for `State Management` foreseen.

D Not implemented requirements

[SWS_SM_NA] Not applicable requirements

Upstream requirements: RS_AP_00134, RS_AP_00153, RS_AP_00144, RS_AP_00145, RS_AP_00146, RS_AP_00147, RS_AP_00127, RS_AP_00143, RS_AP_00129, RS_AP_00135, RS_AP_00136, RS_AP_00137, RS_AP_00140, RS_AP_00148, RS_AP_00155, RS_AP_00128, RS_AP_00114, RS_AP_00151, RS_AP_00154, RS_AP_00116, RS_AP_00124, RS_AP_00141, RS_AP_00138, RS_AP_00139

⌈These requirements are not implemented as they are not within the scope of this release.⌋

E History of Constraints and Specification 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.

E.1 Constraint and Specification Item Changes between AUTOSAR Release R23-11 and R24-11

E.1.1 Added Specification Items in R24-11

Number	Heading
[SWS_SM_00030]	RecoveryHandler can not be handled
[SWS_SM_00031]	Nested recovery handling
[SWS_SM_00210]	Active update session
[SWS_SM_00211]	ResetMachine notification
[SWS_SM_00212]	Default value for ResetMachineNotifier
[SWS_SM_00213]	UpdateRequest method call rejection
[SWS_SM_00650]	StateMachine service interface RequestTransition - transition failed
[SWS_SM_00651]	Processing StopStateMachine ActionListItem
[SWS_SM_00654]	StateMachine marked as "ImpactedByUpdate"
[SWS_SM_00655]	Indirect marking of StateMachine of type Controller as "ImpactedByUpdate"
[SWS_SM_00656]	Unmark "ImpactedByUpdate" from StateMachine
[SWS_SM_00657]	Transition to StateMachine State ContinueUpdate
[SWS_SM_00658]	Transition to Restart state for StateMachine of type Controller
[SWS_SM_00659]	Set ResetMachineNotifier to its default value when update session starts
[SWS_SM_00660]	Set ResetMachineNotifier to default value when stopping update session
[SWS_SM_00661]	Set ResetMachineNotifier to kRejected
[SWS_SM_00662]	Set ResetMachineNotifier to kSuccessful
[SWS_SM_00663]	Set ResetMachineNotifier to kFailed
[SWS_SM_00664]	StateMachine error reaction of StateMachines "ImpactedByUpdate"
[SWS_SM_00665]	StateMachineNotification service interface
[SWS_SM_00666]	Nested recovery
[SWS_SM_91020]	Definition of ImplementationDataType StateMachineStateNameType
[SWS_SM_91027]	Definition of ImplementationDataType UpdateStatusType
[SWS_SM_91028]	Definition of ServiceInterface StateMachineNotification
[SWS_SM_91100]	Definition of Method UpdateRequest.ResetMachine
[SWS_SM_91101]	Definition of Method UpdateRequest.StopUpdateSession
[SWS_SM_91102]	Definition of Method UpdateRequest.RequestUpdateSession

△

Number	Heading
[SWS_SM_91103]	Definition of Method UpdateRequest.PrepareUpdate
[SWS_SM_91104]	Definition of Method UpdateRequest.VerifyUpdate
[SWS_SM_91105]	Definition of Method UpdateRequest.PrepareRollback
[SWS_SM_91106]	Definition of Field UpdateRequest.ResetMachineNotifier
[SWS_SM_91107]	Definition of Method StateMachineService.RequestTransition
[SWS_SM_91108]	Definition of Field UpdateAllowedService.UpdateAllowed
[SWS_SM_91109]	Definition of Field StateMachineNotification.CurrentState

Table E.1: Added Specification Items in R24-11

E.1.2 Changed Specification Items in R24-11

Number	Heading
[SWS_SM_00203]	Start update session
[SWS_SM_00204]	Persist session status
[SWS_SM_00209]	Preventing multiple update sessions
[SWS_SM_00400]	Execution Management
[SWS_SM_00600]	StateMachineService interface
[SWS_SM_00601]	StateMachine error notification reaction of StateMachines not "ImpactedBy Update"
[SWS_SM_00602]	StateMachine ErrorRecoveryOngoing flag reset
[SWS_SM_00603]	StateMachine service interface RequestTransition - not allowed transition
[SWS_SM_00604]	StateMachine service interface RequestTransition - invalid transition
[SWS_SM_00605]	StateMachine service interface RequestTransition - recovery ongoing
[SWS_SM_00606]	Canceling ongoing state transition of StateMachine
[SWS_SM_00607]	StateMachine transition execution
[SWS_SM_00608]	ActionListItem - Function Group State
[SWS_SM_00609]	ActionList processing order
[SWS_SM_00610]	processing SYNC ActionListItem
[SWS_SM_00611]	processing ActionListItem
[SWS_SM_00612]	ActionListItem "Start StateMachine" without parameter, StateMachine is not running
[SWS_SM_00613]	ActionListItem "Start StateMachine" - without parameter, StateMachine is already running
[SWS_SM_00614]	ActionListItem "Stop StateMachine" processing
[SWS_SM_00615]	ActionListItem "Stop StateMachine" processing - StateMachine is not running
[SWS_SM_00616]	CurrentState value during StateMachine State transition

▽

△

Number	Heading
[SWS_SM_00617]	CurrentState value after StateMachine State transition
[SWS_SM_00618]	StateMachine service interfaces - Offer
[SWS_SM_00619]	StateMachine service interfaces - StopOffer
[SWS_SM_00620]	StateMachine transition - NetworkHandle goes to FullCom
[SWS_SM_00621]	StateMachine transition - NetworkHandle goes to NoCom
[SWS_SM_00622]	ActionListItem "Start StateMachine" with parameter, StateMachine is not running
[SWS_SM_00623]	ActionListItem "Start StateMachine" - with parameter, StateMachine is already running
[SWS_SM_00624]	ActionListItem - Sleep
[SWS_SM_00625]	ActionListItem - SetNetworkHandle FullCom
[SWS_SM_00626]	ActionListItem - SetNetworkHandle NoCom
[SWS_SM_00627]	Evaluation of NetworkHandle changes during an update session
[SWS_SM_00628]	Evaluation of NetworkHandle changes for StateMachine of type Controller
[SWS_SM_00629]	Only Process controlling StateMachine of type Controller can provide UpdateRequest interface
[SWS_SM_00630]	Rejection of update session
[SWS_SM_00631]	Acceptance of update session
[SWS_SM_00633]	Transition affected StateMachines to PrepareUpdate state
[SWS_SM_00634]	Shutdown of affected StateMachines during a call to PrepareUpdate method
[SWS_SM_00635]	Failing to prepare for update
[SWS_SM_00636]	Successful preparation for update
[SWS_SM_00638]	Transition affected StateMachines to VerifyUpdate state
[SWS_SM_00639]	Unsuccessful verification of updated software
[SWS_SM_00640]	Successful verification of updated software
[SWS_SM_00642]	Transition affected StateMachines to PrepareRollback state
[SWS_SM_00643]	Shutdown of affected StateMachines during a call to PrepareRollback method
[SWS_SM_00644]	Failing to prepare for rollback
[SWS_SM_00645]	Successful preparation for rollback
[SWS_SM_00646]	Transition Controller to AfterUpdate state
[SWS_SM_00647]	Enabling RequestTransition method after StopUpdateSession call
[SWS_SM_00648]	StateMachine of type Controller start
[SWS_SM_00649]	Block RequestTransition method during an update session
[SWS_SM_91010]	Definition of Application Error Domain of functional cluster SM
[SWS_SM_91016]	Definition of Port UpdateRequest provided by functional cluster SM
[SWS_SM_91017]	Definition of ServiceInterface UpdateRequest
[SWS_SM_91018]	Definition of ImplementationDataType FunctionGroupListType
[SWS_SM_91019]	Definition of ImplementationDataType FunctionGroupNameType

▽

△

Number	Heading
[SWS_SM_91021]	Definition of Port StateMachineService provided by functional cluster SM
[SWS_SM_91022]	Definition of ServiceInterface StateMachineService
[SWS_SM_91023]	Definition of ImplementationDataType TransitionRequestType
[SWS_SM_91024]	Definition of Port UpdateAllowedService provided by functional cluster SM
[SWS_SM_91025]	Definition of ServiceInterface UpdateAllowedService
[SWS_SM_91026]	Definition of ImplementationDataType UpdateAllowedType

Table E.2: Changed Specification Items in R24-11

E.1.3 Deleted Specification Items in R24-11

Number	Heading
[SWS_SM_00001]	Available Function Group (states)
[SWS_SM_00005]	Function Group Calibration Support
[SWS_SM_00006]	Function Group Calibration Support
[SWS_SM_00020]	InternalState Propagation
[SWS_SM_00021]	InternalState Influence
[SWS_SM_00101]	Diagnostic Reset
[SWS_SM_00106]	Enabling of rapid shutdown
[SWS_SM_00107]	Disabling of rapid shutdown
[SWS_SM_00300]	NetworkHandle Configuration
[SWS_SM_00301]	NetworkHandle Registration
[SWS_SM_00302]	NetworkHandle to FunctionGroupState
[SWS_SM_00303]	FunctionGroupState to NetworkHandle
[SWS_SM_00304]	Network Afterrun
[SWS_SM_00500]	Virtualized/hierarchical State Management
[SWS_SM_00501]	Virtualized/hierarchical State Management internal State
[SWS_SM_00632]	Block RequestState method after PrepareUpdate call
[SWS_SM_00637]	Block RequestState method after VerifyUpdate call
[SWS_SM_00641]	Block RequestState method after PrepareRollback call
[SWS_SM_91001]	Definition of Port TriggerIn_{State} provided by functional cluster SM
[SWS_SM_91002]	Definition of Port TriggerOut_{State} provided by functional cluster SM
[SWS_SM_91003]	Definition of Port TriggerInOut_{State} provided by functional cluster SM
[SWS_SM_91004]	Definition of Port NetworkState_{NetworkHandle} required by functional cluster SM
[SWS_SM_91007]	Definition of ServiceInterface TriggerIn
[SWS_SM_91008]	Definition of ServiceInterface TriggerOut

▽

△

Number	Heading
[SWS_SM_91009]	Definition of ServiceInterface TriggerInOut

Table E.3: Deleted Specification Items in R24-11

E.1.4 Added Constraints in R24-11

Number	Heading
[SWS_SM_-CONSTR_-00024]	Existence of StateMachine Off state
[SWS_SM_-CONSTR_-00025]	NmNetworkHandle shall only be controlled by single StateMachine
[SWS_SM_-CONSTR_-00026]	Forbidden usage of "inTransition" as a StateMachine State
[SWS_SM_-CONSTR_-00027]	Existence of StateMachine State AfterUpdate for StateMachine of type Controller
[SWS_SM_-CONSTR_-00028]	Existence of StateMachine State ContinueUpdate
[SWS_SM_-CONSTR_-00029]	Existence of StateMachine State Restart for StateMachine of type Controller
[SWS_SM_-CONSTR_-00030]	Existence of MachineFG Restart in StateMachine State Restart
[SWS_SM_-CONSTR_-00031]	Existence of StateMachine of type Controller
[SWS_SM_-CONSTR_-00032]	Completeness of controlled NmNetworkHandles
[SWS_SM_-CONSTR_-00033]	Configurable Namespace

Table E.4: Added Constraints in R24-11

E.1.5 Changed Constraints in R24-11

Number	Heading
[SWS_SM_-CONSTR_-00001]	Existence of State Management
[SWS_SM_-CONSTR_-00010]	ActionItems in initial StateMachine State
[SWS_SM_-CONSTR_-00011]	ActionListItems allowed in the "Off" state of a StateMachine of type Agent
[SWS_SM_-CONSTR_-00013]	Function Group shall only be controlled by single StateMachine
[SWS_SM_-CONSTR_-00014]	Handling of non-mapped ExecutionError
[SWS_SM_-CONSTR_-00015]	Completeness of controlled Function Groups
[SWS_SM_-CONSTR_-00016]	Completeness of controlled StateMachines
[SWS_SM_-CONSTR_-00017]	ActionListItem "Function Group State" in ActionLists of StateMachine in the Controller
[SWS_SM_-CONSTR_-00018]	Limitations of managed FunctionGroups
[SWS_SM_-CONSTR_-00019]	Usage of ActionListItem "StartStateMachine" and "StopStateMachine"
[SWS_SM_-CONSTR_-00020]	Upper multiplicity of UpdateRequest interface
[SWS_SM_-CONSTR_-00021]	Existence of StateMachine PrepareUpdate state
[SWS_SM_-CONSTR_-00022]	Existence of StateMachine VerifyUpdate state
[SWS_SM_-CONSTR_-00023]	Existence of StateMachine PrepareRollback state

Table E.5: Changed Constraints in R24-11

E.1.6 Deleted Constraints in R24-11

Number	Heading
[SWS_SM_-CONSTR_-00012]	Stop running StateMachines in the final state of a StateMachine

Table E.6: Deleted Constraints in R24-11

E.2 Constraint and Specification Item Changes between AUTOSAR Release R22-11 and R23-11

E.2.1 Added Specification Items in R23-11

Number	Heading
[SWS_SM_00618]	StateMachine service interface - Offer
[SWS_SM_00619]	StateMachine service interface - StopOffer
[SWS_SM_00620]	StateMachine transition - NetworkHandle goes to FullCom
[SWS_SM_00621]	StateMachine transition - NetworkHandle goes to NoCom
[SWS_SM_00622]	ActionListItem "Start StateMachine" with parameter, StateMachine is not running
[SWS_SM_00623]	ActionListItem "Start StateMachine" - with parameter, StateMachine is already running
[SWS_SM_00624]	ActionListItem - Sleep
[SWS_SM_00625]	ActionListItem - SetNetworkHandle FullCom
[SWS_SM_00626]	ActionListItem - SetNetworkHandle NoCom
[SWS_SM_00627]	Evaluation of NetworkHandle changes during `VerifyUpdate` state
[SWS_SM_00628]	Evaluation of NetworkHandle changes for `StateMachine` of type `Controller`
[SWS_SM_00629]	Only Process controlling StateMachine of type Controller can provide UpdateRequest interface
[SWS_SM_00630]	Rejection of update session
[SWS_SM_00631]	Acceptance of update session
[SWS_SM_00632]	Block RequestState method after PrepareUpdate call
[SWS_SM_00633]	Transition affected StateMachines to PrepareUpdate state
[SWS_SM_00634]	Shutdown of affected StateMachines during a call to PrepareUpdate method
[SWS_SM_00635]	Failing to prepare for update
[SWS_SM_00636]	Successful preparation for update
[SWS_SM_00637]	Block RequestState method after VerifyUpdate call
[SWS_SM_00638]	Transition affected StateMachines to VerifyUpdate state

▽

△

Number	Heading
[SWS_SM_00639]	Unsuccessful verification of updated software
[SWS_SM_00640]	Successful verification of updated software
[SWS_SM_00641]	Block RequestState method after PrepareRollback call
[SWS_SM_00642]	Transition affected StateMachines to PrepareRollback state
[SWS_SM_00643]	Shutdown of affected StateMachines during a call to PrepareRollback method
[SWS_SM_00644]	Failing to prepare for rollback
[SWS_SM_00645]	Successful preparation for rollback
[SWS_SM_00646]	Restoring the last known state after update session
[SWS_SM_00647]	Enabling RequestState method after StopUpdateSession call
[SWS_SM_00648]	StateMachine of type Controller start
[SWS_SM_00649]	Block RequestState method in `VerifyUpdate` state
[SWS_SM_91024]	Definition of Port UpdateAllowedService provided by functional cluster SM
[SWS_SM_91025]	Definition of ServiceInterface UpdateAllowedService
[SWS_SM_91026]	Definition of ImplementationDataType UpdateAllowedType

Table E.7: Added Specification Items in R23-11

E.2.2 Changed Specification Items in R23-11

Number	Heading
[SWS_SM_00202]	Reset Execution
[SWS_SM_00203]	Start update session
[SWS_SM_00205]	Stop update session
[SWS_SM_00206]	prepare update
[SWS_SM_00207]	prepare verify
[SWS_SM_00208]	prepare rollback
[SWS_SM_00400]	Execution Management
[SWS_SM_00401]	Execution Management Results
[SWS_SM_00600]	StateMachine service interface
[SWS_SM_00612]	ActionListItem "Start StateMachine" without parameter, StateMachine is not running
[SWS_SM_00613]	ActionListItem "Start StateMachine" - without parameter, StateMachine is already running
[SWS_SM_91010]	Definition of Application Error Domain of functional cluster SM
[SWS_SM_91017]	Definition of ServiceInterface UpdateRequest

▽

△

Number	Heading
[SWS_SM_91022]	Definition of ServiceInterface StateMachineService

Table E.8: Changed Specification Items in R23-11

E.2.3 Deleted Specification Items in R23-11

Number	Heading
[SWS_SM_91011]	
[SWS_SM_91012]	
[SWS_SM_91013]	
[SWS_SM_91014]	
[SWS_SM_91015]	
[SWS_SM_91020]	

Table E.9: Deleted Specification Items in R23-11

E.2.4 Added Constraints in R23-11

Number	Heading
[SWS_SM_-CONSTR_-00017]	ActionListItem "Function Group State" in ActionLists of StateMachine in the Controller
[SWS_SM_-CONSTR_-00018]	Limitations of managed FunctionGroups
[SWS_SM_-CONSTR_-00019]	Usage of ActionListItem "StartStateMachine" and "StopStateMachine"
[SWS_SM_-CONSTR_-00020]	Upper multiplicity of UpdateRequest interface
[SWS_SM_-CONSTR_-00021]	Existence of StateMachine PrepareUpdate state
[SWS_SM_-CONSTR_-00022]	Existence of StateMachine VerifyUpdate state

▽

△

Number	Heading
[SWS_SM_-CONSTR_-00023]	Existence of StateMachine PrepareRollback state

Table E.10: Added Constraints in R23-11

E.2.5 Changed Constraints in R23-11

none

E.2.6 Deleted Constraints in R23-11

none

E.3 Constraint and Specification Item Changes between AUTOSAR Release R21-11 and R22-11

E.3.1 Added Specification Items in R22-11

Number	Heading
[SWS_SM_00600]	StateMachine service interface
[SWS_SM_00601]	StateMachine error notification reaction
[SWS_SM_00602]	StateMachine ErrorRecoveryOngoing flag reset
[SWS_SM_00603]	StateMachine service interface RequestState - not allowed transition
[SWS_SM_00604]	StateMachine service interface RequestState - invalid transition
[SWS_SM_00605]	StateMachine service interface RequestState - recovery ongoing
[SWS_SM_00606]	Canceling ongoing state transition of StateMachine
[SWS_SM_00607]	StateMachine transition execution
[SWS_SM_00608]	ActionListItem - Function Group State
[SWS_SM_00609]	ActionList processing order
[SWS_SM_00610]	processing SYNC ActionListItem
[SWS_SM_00611]	processing ActionListItem
[SWS_SM_00612]	ActionListItem "Start StateMachine" processing
[SWS_SM_00613]	ActionListItem "Start StateMachine" processing - StateMachine is already running
[SWS_SM_00614]	ActionListItem "Stop StateMachine" processing

▽

△

Number	Heading
[SWS_SM_00615]	ActionListItem "Stop StateMachine" processing - StateMachine is not running
[SWS_SM_00616]	Notifier value during StateMachine State transition
[SWS_SM_00617]	Notifier value after StateMachine State transition
[SWS_SM_91021]	
[SWS_SM_91022]	
[SWS_SM_91023]	

Table E.11: Added Specification Items in R22-11

E.3.2 Changed Specification Items in R22-11

Number	Heading
[SWS_SM_00400]	Execution Management
[SWS_SM_91001]	
[SWS_SM_91002]	
[SWS_SM_91003]	
[SWS_SM_91004]	
[SWS_SM_91007]	
[SWS_SM_91008]	
[SWS_SM_91009]	
[SWS_SM_91010]	
[SWS_SM_91011]	
[SWS_SM_91012]	
[SWS_SM_91013]	
[SWS_SM_91014]	
[SWS_SM_91015]	
[SWS_SM_91016]	
[SWS_SM_91017]	
[SWS_SM_91018]	
[SWS_SM_91019]	
[SWS_SM_91020]	

Table E.12: Changed Specification Items in R22-11

E.3.3 Deleted Specification Items in R22-11

Number	Heading
[SWS_SM_00103]	Diagnostic Reset Last Cause
[SWS_SM_00104]	Diagnostic Reset Last Cause Retrieval
[SWS_SM_00105]	Diagnostic Reset Last Cause Reset

Table E.13: Deleted Specification Items in R22-11

E.3.4 Added Constraints in R22-11

Number	Heading
[SWS_SM_CONSTR_00010]	ActionItems in initial StateMachine State
[SWS_SM_CONSTR_00011]	Function Group States referenced in the final state of a StateMachine
[SWS_SM_CONSTR_00012]	Stop running StateMachines in the final state of a StateMachine
[SWS_SM_CONSTR_00013]	Function Group shall only be controlled by single StateMachine
[SWS_SM_CONSTR_00014]	Handling of non-mapped ExecutionError
[SWS_SM_CONSTR_00015]	Completeness of controlled Function Groups
[SWS_SM_CONSTR_00016]	Completeness of controlled StateMachines

Table E.14: Added Constraints in R22-11

E.3.5 Changed Constraints in R22-11

none

E.3.6 Deleted Constraints in R22-11

none

E.4 Constraint and Specification Item Changes between AUTOSAR Release R20-11 and R21-11

E.4.1 Added Specification Items "in R21-11"

Number	Heading
[SWS_SM_00001]	Available Function Group (states)
[SWS_SM_00005]	Function Group Calibration Support
[SWS_SM_00006]	Function Group Calibration Support
[SWS_SM_00020]	InternalState Propagation
[SWS_SM_00021]	InternalState Influence
[SWS_SM_00101]	Diagnostic Reset
[SWS_SM_00103]	Diagnostic Reset Last Cause
[SWS_SM_00104]	Diagnostic Reset Last Cause Retrieval
[SWS_SM_00105]	Diagnostic Reset Last Cause Reset
[SWS_SM_00106]	Enabling of rapid shutdown
[SWS_SM_00107]	Disabling of rapid shutdown
[SWS_SM_00202]	Reset Execution
[SWS_SM_00203]	Start update session
[SWS_SM_00204]	Persist session status
[SWS_SM_00205]	Stop update session
[SWS_SM_00206]	prepare update
[SWS_SM_00207]	prepare verify
[SWS_SM_00208]	prepare rollback
[SWS_SM_00209]	Preventing multiple update sessions
[SWS_SM_00300]	NetworkHandle Configuration
[SWS_SM_00301]	NetworkHandle Registration
[SWS_SM_00302]	NetworkHandle to FunctionGroupState
[SWS_SM_00303]	FunctionGroupState to NetworkHandle
[SWS_SM_00304]	Network Afterrun
[SWS_SM_00400]	Execution Management
[SWS_SM_00401]	Execution Management Results
[SWS_SM_00500]	Virtualized/hierarchical State Management
[SWS_SM_00501]	Virtualized/hierarchical State Management internal State
[SWS_SM_91001]	
[SWS_SM_91002]	
[SWS_SM_91003]	
[SWS_SM_91004]	
[SWS_SM_91007]	

▽

△

Number	Heading
[SWS_SM_91008]	
[SWS_SM_91009]	
[SWS_SM_91010]	
[SWS_SM_91011]	
[SWS_SM_91012]	
[SWS_SM_91013]	
[SWS_SM_91014]	
[SWS_SM_91015]	
[SWS_SM_91016]	
[SWS_SM_91017]	
[SWS_SM_91018]	
[SWS_SM_91019]	
[SWS_SM_91020]	
[SWS_SM_-CONSTR_00001]	Existence of State Management
[SWS_SM_NA]	Not applicable requirements

Table E.15: Added Specification Items "in R21-11"

E.4.2 Changed Specification Items "in R21-11"

none

E.4.3 Deleted Specification Items "in R21-11"

none

E.4.4 Added Constraints "in R21-11"

none

E.4.5 Changed Constraints "in R21-11"

none

E.4.6 Deleted Constraints "in R21-11"

none

E.5 Constraint and Specification Item Changes between AUTOSAR Release R19-11 and R20-11

E.5.1 Added Specification Items in R20-11

Number	Heading
[SWS_SM_00001]	Available Function Group (states)
[SWS_SM_00005]	Function Group Calibration Support
[SWS_SM_00006]	Function Group Calibration Support
[SWS_SM_00020]	InternalState Propagation
[SWS_SM_00021]	InternalState Influence
[SWS_SM_00100]	Prevent Shutdown due to Diagnostic Session
[SWS_SM_00101]	Diagnostic Reset
[SWS_SM_00103]	Diagnostic Reset Last Cause
[SWS_SM_00104]	Diagnostic Reset Last Cause Retrieval
[SWS_SM_00105]	Diagnostic Reset Last Cause Reset
[SWS_SM_00200]	Prevent Shutdown during to Update Session
[SWS_SM_00201]	Supervision of Shutdown Prevention
[SWS_SM_00202]	Reset Execution
[SWS_SM_00203]	Start update session
[SWS_SM_00204]	Persist session status
[SWS_SM_00205]	Stop update session
[SWS_SM_00206]	prepare update
[SWS_SM_00207]	prepare verify
[SWS_SM_00208]	prepare rollback
[SWS_SM_00300]	NetworkHandle Configuration
[SWS_SM_00301]	NetworkHandle Registration
[SWS_SM_00302]	NetworkHandle to FunctionGroupState
[SWS_SM_00303]	FunctionGroupState to NetworkHandle
[SWS_SM_00304]	Network Afterrun
[SWS_SM_00400]	Execution Management
[SWS_SM_00401]	Execution Management Results
[SWS_SM_00402]	Function Group State Change Results
[SWS_SM_00500]	Virtualized/hierarchical State Management
[SWS_SM_00501]	Virtualized/hierarchical State Management internal State

△

Number	Heading
[SWS_SM_91001]	
[SWS_SM_91002]	
[SWS_SM_91003]	
[SWS_SM_91004]	
[SWS_SM_91007]	
[SWS_SM_91008]	
[SWS_SM_91009]	
[SWS_SM_91010]	
[SWS_SM_91011]	
[SWS_SM_91012]	
[SWS_SM_91013]	
[SWS_SM_91014]	
[SWS_SM_91015]	
[SWS_SM_91016]	
[SWS_SM_91017]	
[SWS_SM_91018]	
[SWS_SM_91019]	
[SWS_SM_91020]	

Table E.16: Added Specification Items in R20-11

E.5.2 Changed Specification Items in R20-11

none

E.5.3 Deleted Specification Items in R20-11

none

E.5.4 Added Constraints in R20-11

none

E.5.5 Changed Constraints in R20-11

none

E.5.6 Deleted Constraints in R20-11

none

E.6 Constraint and Specification Item Changes between AUTOSAR Release R19-03 and R19-11

E.6.1 Added Specification Items in 19-11

none

E.6.2 Changed Specification Items in 19-11

Number	Heading
[SWS_SM_00500]	Virtualized/hierarchical State Management
[SWS_SM_00501]	Virtualized/hierarchical State Management internal State

Table E.17: Changed Specification Items in 19-11

E.6.3 Deleted Specification Items in 19-11

none

E.6.4 Added Constraints in 19-11

none

E.6.5 Changed Constraints in 19-11

none

E.6.6 Deleted Constraints in 19-11

none

E.7 Constraint and Specification Item Changes in AUTOSAR Release R19-03

E.7.1 Added Specification Items in 19-03

Number	Heading
[SWS_SM_00020]	InternalState Propagation
[SWS_SM_00021]	InternalState Influence
[SWS_SM_00202]	Reset Execution

Table E.18: Added Specification Items in 19-03

E.7.2 Changed Specification Items in 19-03

Number	Heading
[SWS_SM_00002]	Function Group State Change Request
[SWS_SM_00003]	Function Group State Retrieval
[SWS_SM_00004]	Function Group State Change Request Result
[SWS_SM_00006]	Function Group Calibration Support
[SWS_SM_00200]	Prevent Shutdown during to Update Session
[SWS_SM_00201]	Supervision of Shutdown Prevention
[SWS_SM_00302]	NetworkHandle to FunctionGroupState
[SWS_SM_00401]	Execution Management Results
[SWS_SM_00402]	Function Group State Change Results
[SWS_SM_00500]	Virtualized/hierarchical State Management
[SWS_SM_00501]	Virtualized/hierarchical State Management internal State

Table E.19: Changed Specification Items in 19-03

E.7.3 Deleted Specification Items in 19-03

Number	Heading
[SWS_SM_00010]	Component (states)
[SWS_SM_00011]	Component (states) Handling
[SWS_SM_00012]	Component (states) Registration
[SWS_SM_00013]	Component (states) Configuration
[SWS_SM_00014]	Component (states) Enforcement

△

Number	Heading
[SWS_SM_00015]	Component (states) Transitions
[SWS_SM_00102]	Component States for Reset

Table E.20: Deleted Specification Items in 19-03

E.7.4 Added Constraints in 19-03

none

E.7.5 Changed Constraints in 19-03

none

E.7.6 Deleted Constraints in 19-03

none