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	R25-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2025-11-27	R25-11	AUTOSAR Release Management	• Adding support for Suspend to RAM • Introduction of ErrorCodes for StateMachines • Introduction of ActionListTimeout • Clarifying the order for ActionList processing
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

Table of Contents

1 Introduction and functional overview 9

2 Acronyms and Abbreviations 10

3 Related documentation 13
3.1 Input documents & related standards and norms . . . 13
3.2 Further applicable specification . . . 14

4 Constraints and assumptions 15
4.1 Known limitations . . . 15

5 Dependencies to other Functional Clusters 16
5.1 Provided Interfaces . . . 16
5.2 Required Interfaces . . . 17
5.3 Suspend-to-RAM - Security Considerations . . . 18

6 Requirements Tracing 19

7 Functional specification 23
7.1 State Management Responsibilities . . . 24
7.1.1 Machine State . . . 25
7.1.2 Function Group State . . . 26
7.1.3 State Management Architecture . . . 27
7.2 Interaction with Platform Health Management . . . 28
7.3 Interaction with Update and Configuration Management . . . 29
7.4 Interaction with Network Management . . . 33
7.5 Interaction with Suspend-to-RAM Functionality . . . 34
7.5.1 S2R Hub . . . 35
7.5.2 S2R Satellite . . . 37
7.5.3 Platform Extension Power Mode . . . 38
7.5.4 Coordinated `Suspend Mode` Management for Virtual Machines . . . 39
7.6 Interaction with Execution Management . . . 39
7.7 StateManagement StateMachine . . . 41
7.7.1 StateMachine introduction . . . 41
7.7.2 Controlling application for StateMachine States . . . 42
7.7.3 StateMachine design considerations . . . 45
7.7.4 StateMachine general conditions . . . 47
7.7.5 StateMachine state changes . . . 49
7.7.6 StateMachine ActionLists . . . 53
7.7.7 StateMachine ActionListItems . . . 53
7.7.8 Controlling multiple StateMachine Instances . . . 58
7.7.9 ActionListItem Sleep . . . 64
7.7.10 ActionListItem SetNetworkHandle . . . 65
7.7.11 StateMachine State notification . . . 66

7.7.12 StateMachine ActionListTimeout . . . 68
7.7.13 StateMachine ErrorCode configuration and handling . . . 69
7.7.14 StateMachine support for Update and Configuration Management . 76
7.7.15 StateMachine support for Suspend-to-RAM . . . 95
7.7.15.1 Autonomous wake-up from suspend state . . . 102
7.8 Functional cluster life-cycle . . . 107
7.8.1 Startup . . . 107
7.8.2 Shutdown . . . 107
7.8.3 Restart . . . 108
7.8.4 Suspended . . . 108
7.8.4.1 Suspend-to-RAM tolerant . . . 108
7.8.4.2 Suspend-to-RAM not supported . . . 109
7.8.4.3 Suspend-to-RAM aware . . . 109
7.8.5 Daemon crash . . . 109
7.9 Reporting . . . 109
7.9.1 Security Events . . . 109
7.9.2 Log Messages . . . 110
7.9.3 Violation Messages . . . 110
7.9.4 Production Errors . . . 113
8 API specification 114
8.1 Header: ara/sm/sm_error_domain.h . . . 115
8.1.1 Non-Member Types . . . 115
8.1.1.1 Enumeration: SmErrc . . . 115
8.1.2 Non-Member Functions . . . 116
8.1.2.1 Other . . . 116
8.1.3 Class: SmErrorDomain . . . 117
8.1.3.1 Public Member Types . . . 117
8.1.3.2 Public Member Functions . . . 118
8.1.4 Class: SmException . . . 120
8.1.4.1 Public Member Functions . . . 121
8.2 Header: ara/sm/s2r/S2RHub.h . . . 121
8.2.1 Class: S2RHub . . . 121
8.2.1.1 Public Member Functions . . . 122
8.3 Header: ara/sm/s2r/S2RSatellite.h . . . 126
8.3.1 Class: S2RSatellite . . . 126
8.3.1.1 Public Member Functions . . . 127
9 Service Interfaces 134
9.1 Implementation Data Types . . . 134
9.1.1 Data types for Update And Configuration Management interaction . 134
9.1.2 Data types for StateMachine interaction . . . 134
9.1.3 Data types for StateMachine notification . . . 135
9.1.4 Data types for UpdateAllowed service interface . . . 135

9.1.5 Data types for ResetMachineNotifier . . . 135
9.2 Provided Service Interfaces . . . 137
9.2.1 UpdateRequest . . . 137
9.2.2 StateMachine service . . . 141
9.2.3 StateMachine UpdateAllowed service . . . 143
9.3 Required Service Interfaces . . . 144
9.4 Application Errors . . . 145
9.4.1 StateManagement Error Domain . . . 145
10 Configuration 146
10.1 Default Values . . . 146
10.2 Semantic Constraints . . . 146
A Mentioned Manifest Elements 147
B Demands and constraints on Base Software (normative) 162
C Platform Extension Interfaces (normative) 163
C.1 Header: apext/sm/power_state_interface.h . . . 163
C.1.1 Non-Member Types . . . 163
C.1.1.1 Enumeration: PowerState . . . 163
C.1.1.2 Type Alias: WakeUpHandler . . . 164
C.1.2 Class: PowerStateInterface . . . 164
C.1.2.1 Public Member Functions . . . 165
D Not implemented requirements 167
E History of Constraints and Specification Items 168
E.1 Constraint and Specification Item Changes between AUTOSAR Release R24-11 and R25-11 . . . 168
E.1.1 Added Specification Items in R25-11 . . . 168
E.1.2 Changed Specification Items in R25-11 . . . 170
E.1.3 Deleted Specification Items in R25-11 . . . 171
E.1.4 Added Constraints in R25-11 . . . 171
E.1.5 Changed Constraints in R25-11 . . . 172
E.1.6 Deleted Constraints in R25-11 . . . 173
E.2 Constraint and Specification Item Changes between AUTOSAR Release R23-11 and R24-11 . . . 173
E.2.1 Added Specification Items in R24-11 . . . 173
E.2.2 Changed Specification Items in R24-11 . . . 174
E.2.3 Deleted Specification Items in R24-11 . . . 176
E.2.4 Added Constraints in R24-11 . . . 177
E.2.5 Changed Constraints in R24-11 . . . 178
E.2.6 Deleted Constraints in R24-11 . . . 179
E.3 Constraint and Specification Item Changes between AUTOSAR Release R22-11 and R23-11 . . . 179
E.3.1 Added Specification Items in R23-11 . . . 179

E.3.2 Changed Specification Items in R23-11 . . . 180
E.3.3 Deleted Specification Items in R23-11 . . . 181
E.3.4 Added Constraints in R23-11 . . . 181
E.3.5 Changed Constraints in R23-11 . . . 181
E.3.6 Deleted Constraints in R23-11 . . . 182
E.4 Constraint and Specification Item Changes between AUTOSAR Release R21-11 and R22-11 . . . 182
E.4.1 Added Specification Items in R22-11 . . . 182
E.4.2 Changed Specification Items in R22-11 . . . 183
E.4.3 Deleted Specification Items in R22-11 . . . 183
E.4.4 Added Constraints in R22-11 . . . 184
E.4.5 Changed Constraints in R22-11 . . . 184
E.4.6 Deleted Constraints in R22-11 . . . 184
E.5 Constraint and Specification Item Changes between AUTOSAR Release R20-11 and R21-11 . . . 184
E.5.1 Added Specification Items "in R21-11" . . . 184
E.5.2 Changed Specification Items "in R21-11" . . . 186
E.5.3 Deleted Specification Items "in R21-11" . . . 186
E.5.4 Added Constraints "in R21-11" . . . 186
E.5.5 Changed Constraints "in R21-11" . . . 186
E.5.6 Deleted Constraints "in R21-11" . . . 186
E.6 Constraint and Specification Item Changes between AUTOSAR Release R19-11 and R20-11 . . . 186
E.6.1 Added Specification Items in R20-11 . . . 186
E.6.2 Changed Specification Items in R20-11 . . . 188
E.6.3 Deleted Specification Items in R20-11 . . . 188
E.6.4 Added Constraints in R20-11 . . . 188
E.6.5 Changed Constraints in R20-11 . . . 188
E.6.6 Deleted Constraints in R20-11 . . . 188
E.7 Constraint and Specification Item Changes between AUTOSAR Release R19-03 and R19-11 . . . 188
E.7.1 Added Specification Items in 19-11 . . . 188
E.7.2 Changed Specification Items in 19-11 . . . 189
E.7.3 Deleted Specification Items in 19-11 . . . 189
E.7.4 Added Constraints in 19-11 . . . 189
E.7.5 Changed Constraints in 19-11 . . . 189
E.7.6 Deleted Constraints in 19-11 . . . 189
E.8 Constraint and Specification Item Changes in AUTOSAR Release R19-03 189
E.8.1 Added Specification Items in 19-03 . . . 189
E.8.2 Changed Specification Items in 19-03 . . . 190
E.8.3 Deleted Specification Items in 19-03 . . . 190
E.8.4 Added Constraints in 19-03 . . . 190
E.8.5 Changed Constraints in 19-03 . . . 190
E.8.6 Deleted Constraints in 19-03 . . . 191

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.
Suspend-to-RAM	Suspend-to-RAM is an energy-saving mode in which the computer is almost completely switched off, but the RAM remains active so that the current system status can be quickly restored.
S2R Hub	The S2R Hub is in charge to coordinate from the State Management the communication with the S2R satellites. The S2R Hub could either be used via StateMachine approach or via a C++ API (e.g. in a SMControlApplication)
S2R Satellite	The S2R Satellite can be used in S2R-Aware applications to register as S2R satellite and are then informed before a Suspend-to-RAM is entered and after the wake-up again that they can resume their 'normal' operation.
S2R-Aware application	Application is notified before S2R and can prepare accordingly. e.g., stop offering services, persist data
S2R-Tolerant application	Application seamlessly handles S2R, no action required. e.g., all used resources support S2R (monotonic clock,...)
S2R-Unsupported application	Application does not support S2R at all, needs to be terminated and restarted. e.g., HW requirement
Suspend Mode	Suspend Mode is an internal application state in that manages degradations such as stopping service offerings or safety functions before entering Suspend-to-RAM and after resuming from it.

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] Explanation of 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] Explanation of Identity and Access Management
AUTOSAR_AP_EXP_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

[14] Specification of Execution Management
AUTOSAR_AP_SWS_ExecutionManagement

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

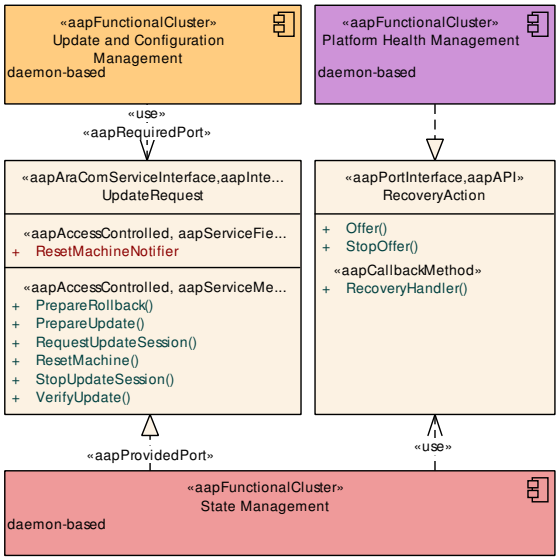

Figure 5.1: Interfaces provided by State Management to other Functional Clusters

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*
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

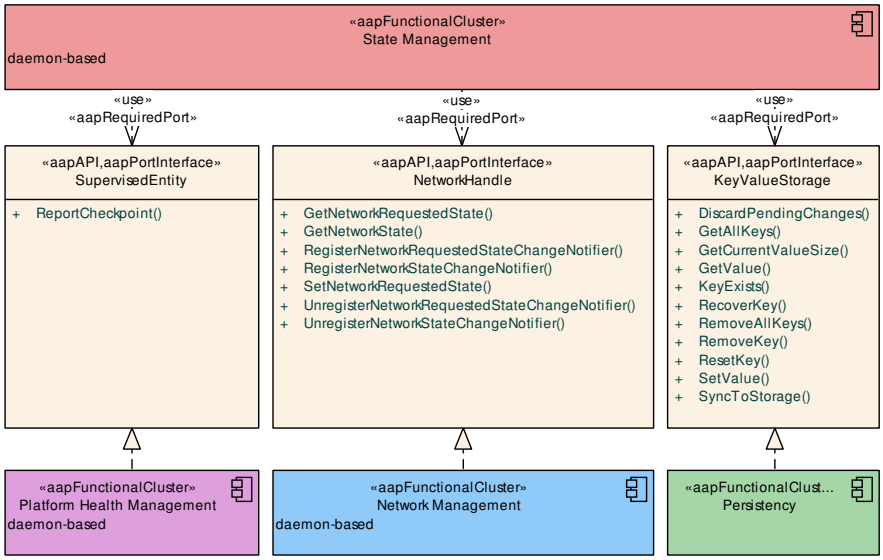

Figure 5.2: Interfaces required by State Management from other Functional Clusters

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

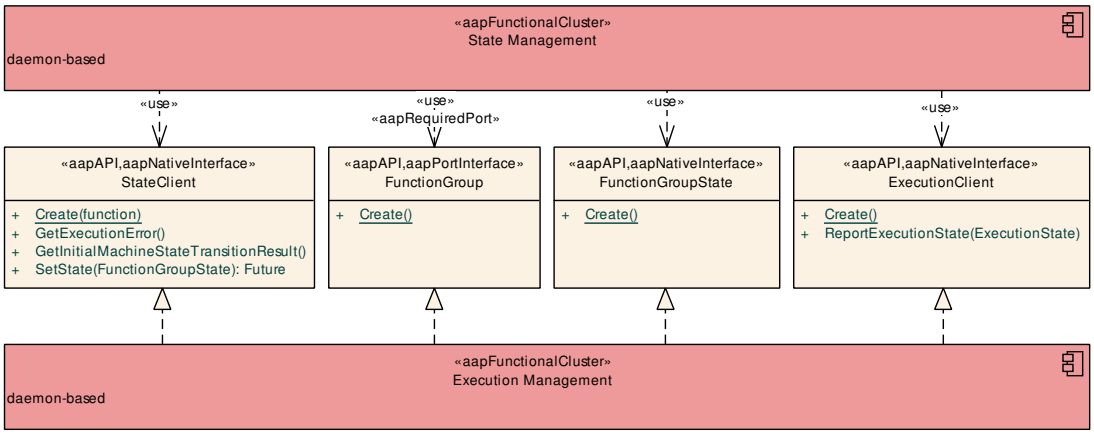

Figure 5.3: Interfaces required by State Management from Execution Management

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 `FunctionGroup State`s.

△

Functional Cluster	*Interface*	*Purpose*
Execution Management	FunctionGroupState	This interface shall be used to request `FunctionGroup State` transitions.
Execution Management	StateClient	This interface shall be used to request `FunctionGroup State` 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	RecoveryAction	`Platform Health Management` uses this interface to trigger failure recovery.
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

5.3 Suspend-to-RAM - Security Considerations

The `Suspend-to-RAM` low-power state relies on the security capabilities of the underlying platform. In automotive applications, where cybersecurity and functional safety are paramount, it is the responsibility of the system integrator to ensure that `Suspend-to-RAM` is correctly configured and that all relevant security mechanisms are fully implemented and validated.

In the context of automotive ECUs – especially those handling sensitive data – `Suspend-to-RAM` must only be used if its security implications have been thoroughly assessed and mitigated through appropriate platform and OS-level measures.

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_81240] [SWS_SM_81241] [SWS_SM_81242] [SWS_SM_81243] [SWS_SM_81244] [SWS_SM_81245] [SWS_SM_81246] [SWS_SM_81247] [SWS_SM_81248] [SWS_SM_81249] [SWS_SM_81250] [SWS_SM_81251] [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_AP_00159]	usage of "noexcept" specifier	[SWS_SM_81243] [SWS_SM_81244] [SWS_SM_81247] [SWS_SM_81248] [SWS_SM_81249] [SWS_SM_81251]
[RS_Ids_00810]	Basic SW security events	[SWS_SM_70000] [SWS_SM_70001]
[RS_SM_00001]	State Management shall coordinate and control multiple sets of Applications.	[SWS_SM_00203] [SWS_SM_00210] [SWS_SM_00214] [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]

△

Requirement	Description	Satisfied by
		△ [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_00667] [SWS_SM_00668] [SWS_SM_00669] [SWS_SM_00670] [SWS_SM_00671] [SWS_SM_00672] [SWS_SM_00673] [SWS_SM_00674] [SWS_SM_00675] [SWS_SM_00676] [SWS_SM_00677] [SWS_SM_00678] [SWS_SM_00679] [SWS_SM_00685] [SWS_SM_00686] [SWS_SM_00687] [SWS_SM_00688] [SWS_SM_91016] [SWS_SM_91017] [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_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_00214] [SWS_SM_00680] [SWS_SM_00681] [SWS_SM_00682] [SWS_SM_00683] [SWS_SM_00684] [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]

▽

△

Requirement	Description	Satisfied by
[RS_SM_00005]	State Management internal states.	[SWS_SM_00203] [SWS_SM_00210] [SWS_SM_00214] [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_00667] [SWS_SM_00668] [SWS_SM_00669] [SWS_SM_00670] [SWS_SM_00671] [SWS_SM_00672] [SWS_SM_00673] [SWS_SM_00674] [SWS_SM_00675] [SWS_SM_00676] [SWS_SM_00677] [SWS_SM_00678] [SWS_SM_00679] [SWS_SM_00680] [SWS_SM_00681] [SWS_SM_00682] [SWS_SM_00683] [SWS_SM_00684] [SWS_SM_00685] [SWS_SM_00686] [SWS_SM_00687] [SWS_SM_00688]
[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]
[RS_SM_00402]	Coordination of System Sleep States	[SWS_SM_00214] [SWS_SM_00680] [SWS_SM_00681] [SWS_SM_00682] [SWS_SM_00683] [SWS_SM_00684] [SWS_SM_00685] [SWS_SM_00686] [SWS_SM_00687] [SWS_SM_00688] [SWS_SM_00689] [SWS_SM_00690] [SWS_SM_00691] [SWS_SM_00692] [SWS_SM_00701] [SWS_SM_00702] [SWS_SM_00703] [SWS_SM_00705] [SWS_SM_00706] [SWS_SM_00707] [SWS_SM_00710] [SWS_SM_00711] [SWS_SM_00712] [SWS_SM_00713] [SWS_SM_00714] [SWS_SM_00715] [SWS_SM_00716] [SWS_SM_00717] [SWS_SM_71000] [SWS_SM_71001] [SWS_SM_71002] [SWS_SM_71003] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_SM_71004] [SWS_SM_71005] [SWS_SM_81002] [SWS_SM_81003] [SWS_SM_81004] [SWS_SM_81005] [SWS_SM_81006] [SWS_SM_81007] [SWS_SM_81008] [SWS_SM_81009] [SWS_SM_81010] [SWS_SM_81011] [SWS_SM_81012] [SWS_SM_81013] [SWS_SM_81014] [SWS_SM_81102] [SWS_SM_81103] [SWS_SM_81105] [SWS_SM_81106] [SWS_SM_81107] [SWS_SM_81108] [SWS_SM_81109] [SWS_SM_81110] [SWS_SM_81111] [SWS_SM_81112]
[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 Management's NetworkHandle.

State Management also supports the Suspend-to-RAM functionality, enabling the system to enter a low-power state while preserving operational context in memory. This capability is achieved through coordinated interactions between applications, functional clusters, and the underlying operating system. By managing transitions into and out of Suspend-to-RAM, State Management ensures that system integrity and responsiveness are maintained, facilitating seamless resumption of services upon wake-up.

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 explains how State Management coordinates Suspend-to-RAM functionality
- Section 7.6 describes how Execution Management is used to change Function Group State or Machine State.
- Section 7.7 explains how State Management coordinates state changes in Function Groups, NmNetworkHandles and Adaptive Applications via StateMachines state changes.

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 Management's 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.
- Suspend-to-RAM handling, where State Management can coordinate via Function Group State transitions, that S2R-Unsupported applications are terminated before entering low-power mode and are restarted during the wake-up phase to restore full system functionality.

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 service interfaces 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

- Adaptive Application state related to Suspend-to-RAM. See Section 7.5

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.

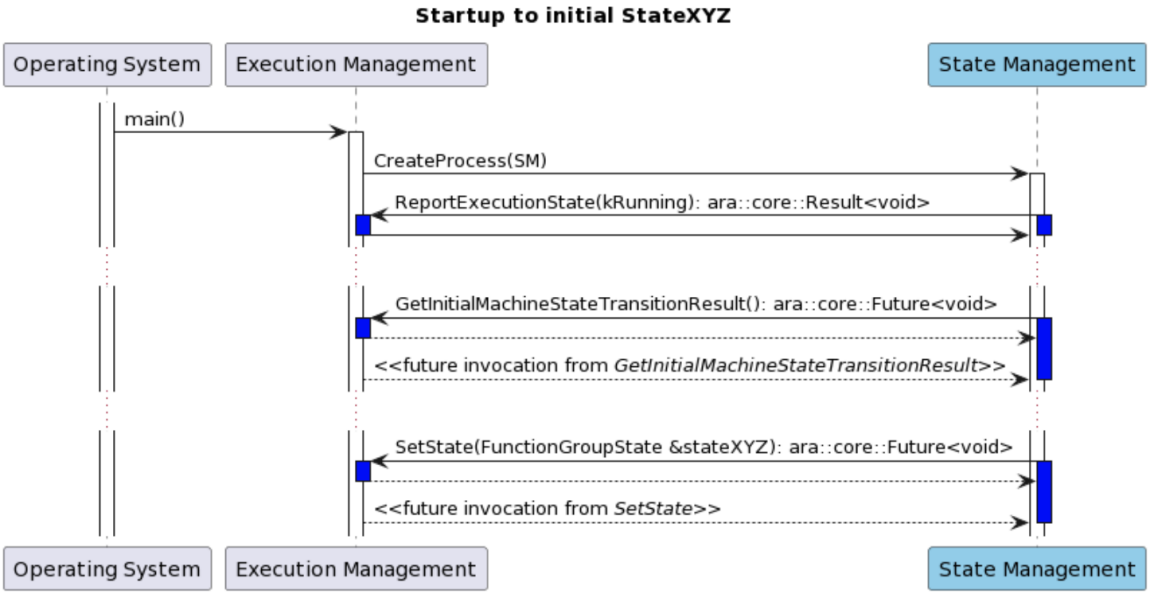

Figure 7.1: Start-up Sequence – from Startup to initial running state Driving

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.

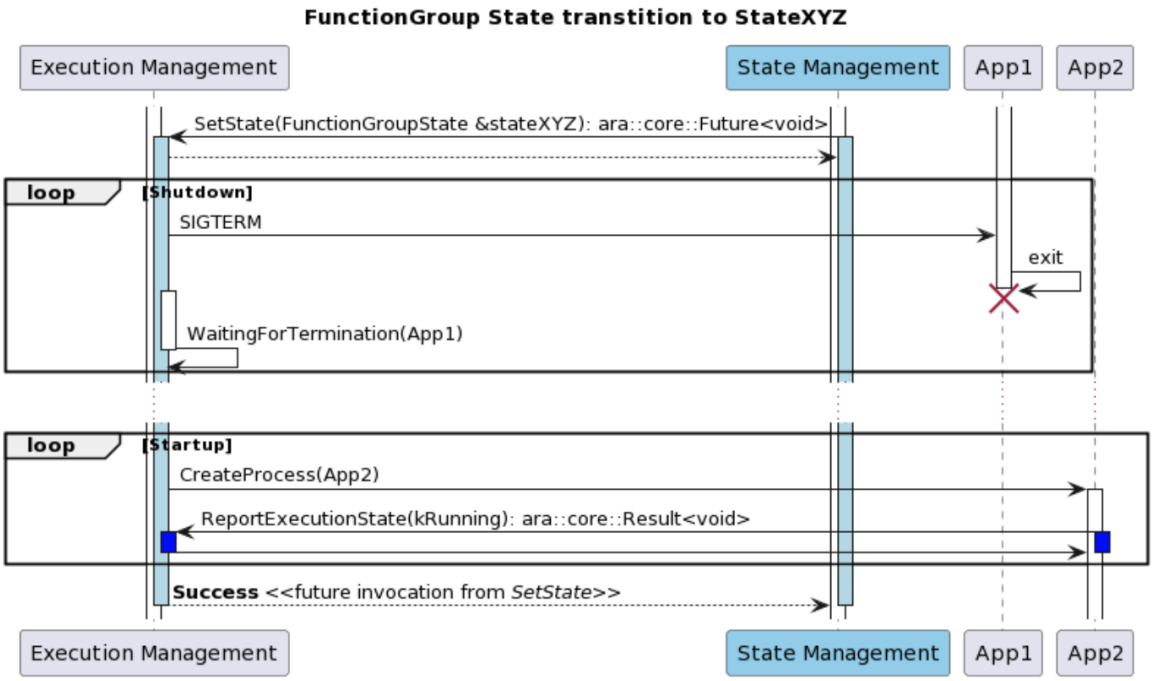

Figure 7.2: State Change Sequence – Transition to machine state `StateXYZ`

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 kExpired state, Platform Health Management will notify State Management via C++ API provided by Platform Health Management. 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 during a performed recovery action a new issue in the same StateMachine is reported. 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 provide the same result to all RecoveryHandler() calls related to the same affected StateMachine based on the result when the latest recovery action issued by the last RecoveryHandler() call is finished.⌋

Note: A "nested recovery" is very project specific and therefore an implementation detail. For StateMachine approach this detail is explained in chapter 7.7.5.

Note: Platform Health Management monitors the return of the RecoveryHandler() with a configurable timeout. If State Management gives no response to RecoveryHandler() Platform Health Management will do its own countermeasures by 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 7.7.5.

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.

To ensure update integrity and avoid potential inconsistencies, State Management has to reject any Suspend-to-RAM request once an update session request has been granted. This safeguard prevents the system from entering a low-power state that could interrupt or compromise the update process.

[SWS_SM_00214] Reject Suspend-to-RAM during active update session

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00004, RS_SM_00005, RS_SM_00402

⌈If a Suspend-to-RAM request is received while an update session is active [SWS_SM_00210], State Management shall reject the request.⌋

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 7.7.14.

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 7.7.10 and in section 7.7.5.

7.5 Interaction with Suspend-to-RAM Functionality

To optimize the shutdown and startup times, the machine could use Suspend-to-RAM instead of a complete shutdown and initial boot.

This includes before the OS is entering any suspend state

- a (partial) shutdown of all Function Groups including Executables where Executable.suspendToRamAwareness is set to suspendToRamNotSupported. This means that either all affected Function Groups are in "Off" state, or at least in a state where these affected Processs are not included/executed anymore.
- a notification to all Processs where their Executable.suspendToRamAwareness is set to suspendToRamAware. This can either be achieved by the StateMachine approach (see section 7.7.15) or by an instance of ara::sm::s2r::S2RHub in the SMControlApplication (see section 7.5.1). However under consideration of [SWS_SM_00684].

State Management provides a special mechanisms to coordinate Suspend-to-RAM operations for S2R-Aware applications. These Adaptive Applications require a state synchronization before the operating system enters the suspend state. To achieve this, State Management offers an S2R Hub that manages communication with S2R Satellites. Satellites can dynamically register with the hub. The S2R Hub can be accessed either through the StateMachine approach (see section 7.7.15) or via the C++ API (ara::sm::s2r::S2RHub) (see section 7.5.1). These options are mutually exclusive. To ensure consistent system behavior and avoid conflicting requests, external applications shall not use the ara::sm::s2r::S2RHub API when the StateMachine approach is chosen (see [SWS_SM_00684]). The StateMachine approach integrates the ara::sm::s2r::S2RHub logic internally.

For this interaction, the State Management offers S2R Satellites which can register dynamically at the hub as shown in this figure:

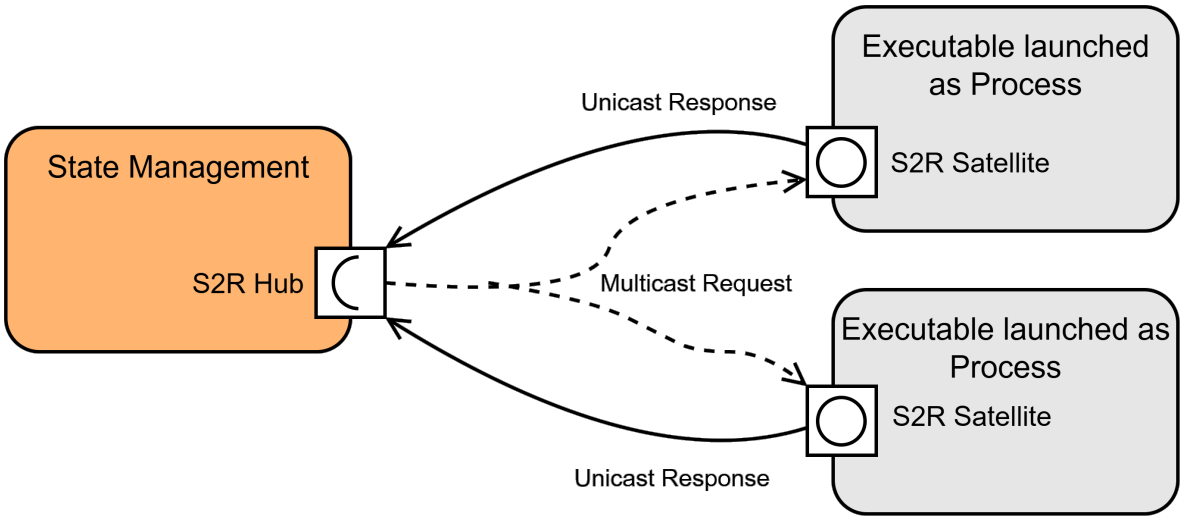

Figure 7.3: Interaction between S2RSatellite and S2RHub

7.5.1 S2R Hub

The `S2R Hub` coordinates communication with `S2R Satellites`. It can be accessed through either the `StateMachine` approach (see section 7.7.15) or a C++ API (e.g. in an `SMControlApplication`) in a mutual exclusive manner (see [SWS_SM_00684]).

[SWS_SM_00706] IAM check

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈`S2R Hub` shall verify during the `ara::sm::s2r::S2RHub::S2RHub` constructor that the current `Process` context matches the mapped `SuspendToRamHubMapping.process`. In case not, the Violation `InsufficientPermissionsViolation` and the SecurityEvent `SEV_ACCESS_CONTROL_SM_IAM_ACCESS_DENIED` shall be triggered.⌋

[SWS_SM_00714] IAM demand

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈`S2R Hub` shall restrict processes that can trigger the suspend to RAM by using a `ara::sm::s2r::S2RHub::S2RHub` identified by a `SuspendToRamModuleInstantiation` to `Process`es referenced by a `SuspendToRamHubMapping` in the role `process` which also references the `SuspendToRamModuleInstantiation` in the role `moduleInstantiation`.⌋

[SWS_SM_00715] Restrict to only authenticated S2R Satellites

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈`S2R Hub` shall restrict that only IAM authenticated `S2R Satellites` can register. This means that the registration shall be rejected, if the `ara::sm::s2r::S2RSatellite` process context does not match the `Process` referenced by a `SuspendToRamSatelliteMapping` in the role `process`.⌋

[SWS_SM_00701] Request satellites to enter Suspend Mode

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈When `ara::sm::s2r::S2RHub::RequestToEnterSuspendMode` is invoked, the `S2R Hub` shall send a message to all actively registered `S2R Satellite` instances (see [SWS_SM_00710]) to enter the `Suspend Mode` by calling the `ara::sm::s2r::S2RSatellite::EnterSuspendMode` callback.⌋

[SWS_SM_00702] Returning EnterSuspendState

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈`ara::sm::s2r::S2RHub::RequestToEnterSuspendMode` shall return after all addressed `S2R Satellite` instances have either responded, deregistered, or are in timeout.

- The overall result is positive, if all `S2R Satellite` instances respond positively or deregister via `ara::sm::s2r::S2RSatellite::StopOffer`.
- If any `ara::sm::s2r::S2RSatellite` respond negatively with `kRejected`, the result shall be `kAtLeastOneRejected`.
- Communication is monitored with the timeout `timeout`. If a satellite does not respond in time, the ErrorCode `kCommunicationTimeout` shall be returned.
- Any internal communication error shall result in the ErrorCode `kCommunicationFailed`.

⌋

Note: The deregistration of a `S2R Satellite` is considered a positive result, as it is assumed that the process will then be terminated.

[SWS_SM_00705] Request OS to enter suspend state

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈To enter the OS suspend state, `ara::sm::s2r::S2RHub::EnterSuspendToRamOs` shall forward the call to `apext::sm::PowerStateInterface::SetPowerState` with `state` set to `kPmSuspendToRam`.⌋

[SWS_SM_00703] Request satellites to leave Suspend Mode

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈To leave the `Suspend Mode`, `ara::sm::s2r::S2RHub::RequestToLeaveSuspendMode` shall send a message to all actively registered `S2R Satellite` instances (see [SWS_SM_00710]) to invoke `ara::sm::s2r::S2RSatellite::LeaveSuspendMode`).⌋

[SWS_SM_00707] Returning LeaveSuspendState

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈`ara::sm::s2r::S2RHub::RequestToLeaveSuspendMode` shall return after all addressed `S2R Satellite` instances have either responded, deregistered, or are in timeout.

- The overall result is positive, if all `S2R Satellite` instances respond positively or deregister via `ara::sm::s2r::S2RSatellite::StopOffer`.
- If any `ara::sm::s2r::S2RSatellite` respond negatively with `kRejected`, the result shall be `kAtLeastOneHadIssuesToLeave`.
- Communication is monitored with the timeout `timeout`. If a satellite does not respond in time, the ErrorCode `kCommunicationTimeout` shall be returned.
- Any internal communication error shall result in the ErrorCode `kCommunicationFailed`.

⌋

7.5.2 S2R Satellite

`S2R Satellite` are used in `S2R-Aware applications` to register with the `S2R Hub` and receive notifications before entering and after leaving the `Suspend Mode`.

[SWS_SM_00712] IAM check in S2R Satellite

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈`S2R Satellite` shall verify during the `ara::sm::s2r::S2RSatellite::S2RSatellite` constructor that the current `Process` context matches the mapped `SuspendToRamSatelliteMapping.process`. In case not, the Violation `InsufficientPermissionsViolation` and the SecurityEvent `SEV_ACCESS_CONTROL_SM_IAM_ACCESS_DENIED` shall be triggered.⌋

In projects without IAM requirements on `S2R Satellites`, the named constructor `ara::sm::s2r::S2RSatellite::Create` can be used.

[SWS_SM_00716] Forced IAM check in S2R Satellite

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈`ara::sm::s2r::S2RSatellite::Create` shall return the error `kAuthenticationRequired` if the *"Force IAM switch" is set to true*.⌋

[SWS_SM_00710] Registration of S2R Satellites at S2R Hub

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈An `S2R Satellite` instance shall register with the hub via `ara::sm::s2r::S2RSatellite::Offer`. It is considered actively registered until `ara::sm::s2r::S2RSatellite::StopOffer` is called.⌋

[SWS_SM_00711] Trigger to enter and leave Suspend Mode

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈On receiving a suspend trigger, the `S2R Satellite` shall invoke `ara::sm::s2r::S2RSatellite::EnterSuspendMode`.

On receiving a resume trigger, it shall invoke `ara::sm::s2r::S2RSatellite::LeaveSuspendMode`.⌋

[SWS_SM_00713] Deregistration of S2R Satellites at S2R Hub

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈`ara::sm::s2r::S2RSatellite::StopOffer` shall deregister the `S2R Satellite` and block until any ongoing callback execution completes.⌋

The `ara::sm::s2r::S2RSatellite::StopOffer` has either to be called from the application before the destruction of `ara::sm::s2r::S2RSatellite`, or within the user defined destructor in the inherited S2RSatellite class.

7.5.3 Platform Extension Power Mode

`State Management` requires communication with the OS, which is facilitated by `apext::sm::PowerStateInterface`.

[SWS_SM_00717] Instantiation of apext::sm::PowerStateInterface

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈If `ara::sm::s2r::S2RHub` is instantiated, the `State Management` shall also create an instance of the `apext::sm::PowerStateInterface`.⌋

7.5.4 Coordinated Suspend Mode Management for Virtual Machines

To ensure a consistent and scalable approach to managing the suspend state of multiple virtual machines (VMs), the coordination mechanism shall be handled externally. Each VM is responsible for initiating its own shutdown process upon receiving a designated trigger. This means that:

- There is no centralized coordinator communicating directly with the applications inside each VM.
- Instead, an external system signals the suspend event (e.g. to the edge State Management via ara::com), and each VM reacts independently based on its internal logic and configuration.
- This design promotes modularity, resilience, and ease of integration, as each VM encapsulates its own shutdown behavior without relying on external orchestration at the application level.

By decentralizing the shutdown logic, the system avoids tight coupling between the coordinator and individual VM internals, leading to a more robust and maintainable architecture.

AUTOSAR State Management does not provide standardized external management interfaces.

The AUTOSAR Network Management can alternatively be used to coordinate the suspend or shutdown of a VM. For the wake-up the VM must be configured to resume or start on network activities.

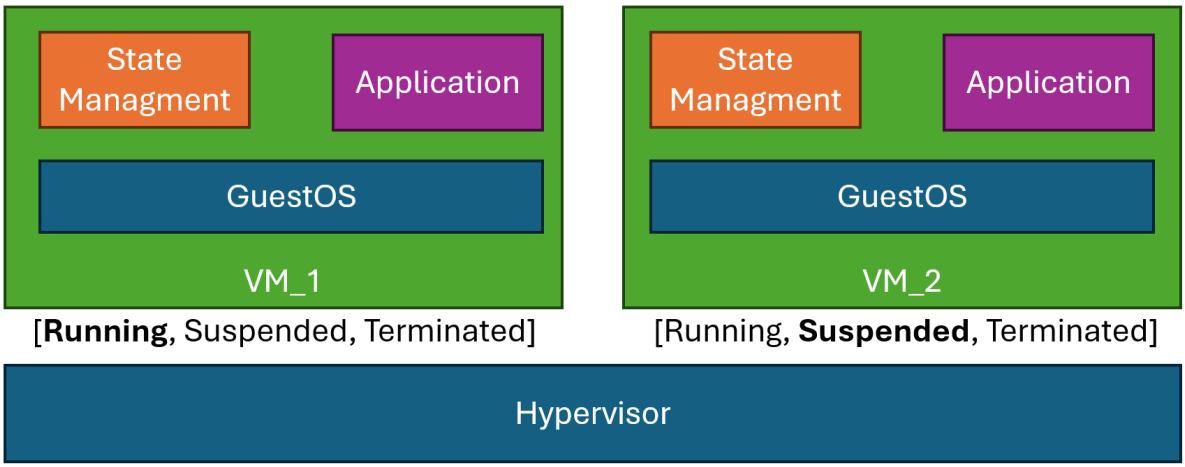

Figure 7.4: Virtualized environments

7.6 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.

[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.

7.7 StateManagement StateMachine

7.7.1 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).
- Suspend-to-RAM, (represented by meta-class StateManagementEnterSuspendToRamActionItem, StateManagementEnterSuspendToRamOsActionItem and StateManagementLeaveSuspendToRamActionItem) to control the Suspend Mode in S2R-Aware applications.

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.
- Suspend-to-RAM handling can trigger StateMachine State changes when transitioning the Controller into low-power suspend state or during wake-up.

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:

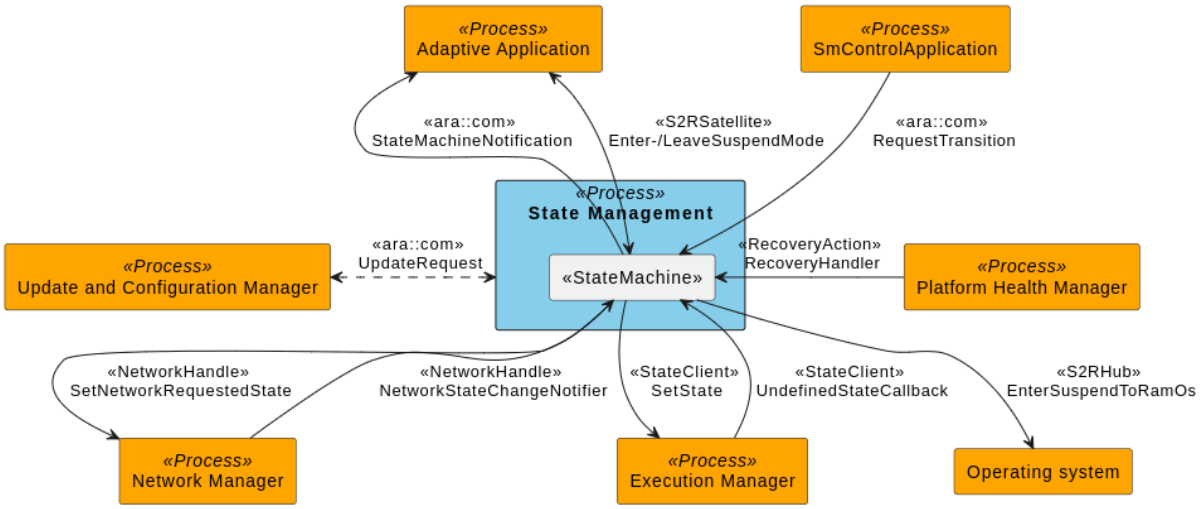

Figure 7.5: Interactions with StateMachine

StateMachines are an optional element of State Management, so projects can decide to implement State Management fully by its own. In this case, implementation is not bound by requirements in Chapter 7.7. This is achieved by keeping interfaces towards State Management public.

7.7.2 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.

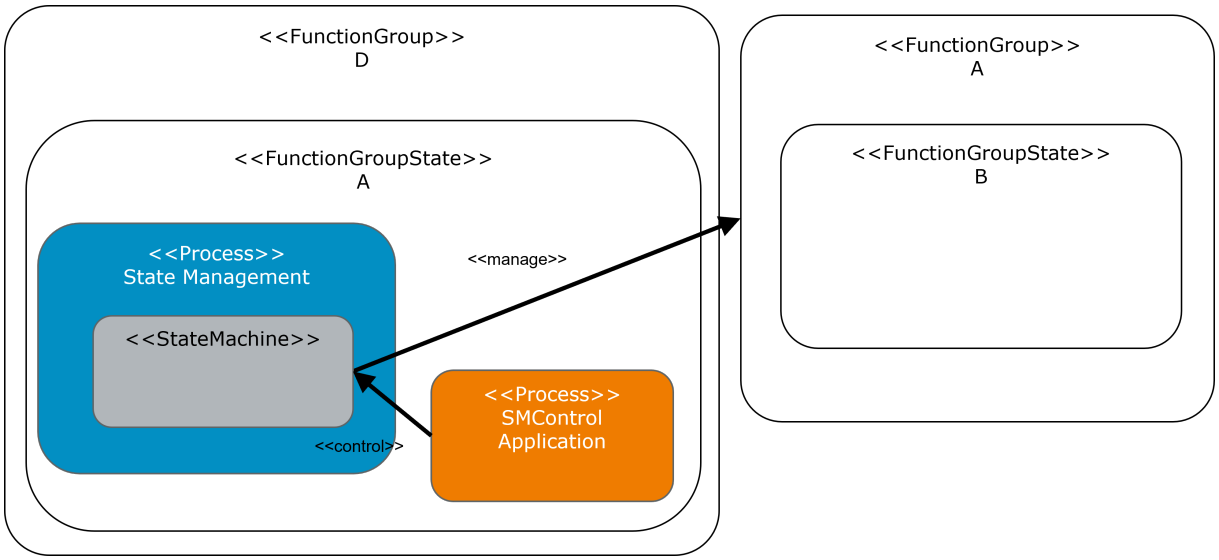

Figure 7.6: SMControlApplication and StateManagement Process started together

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

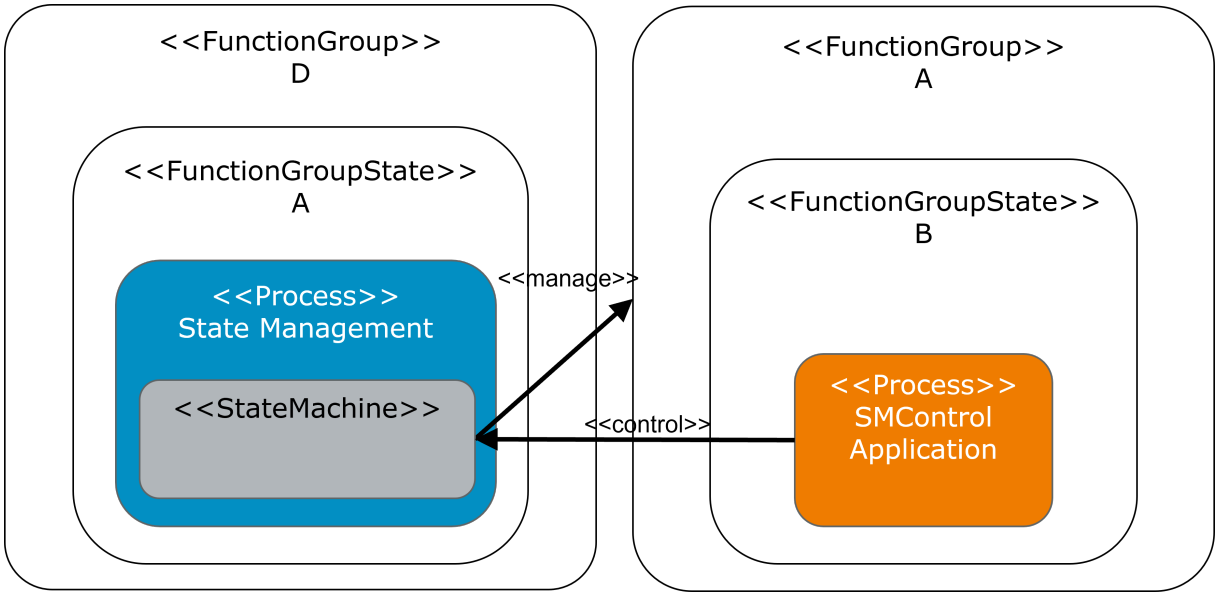

Figure 7.7: SMControlApplication started in initial State of StateManagements StateMachine

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.

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 7.8: TransitionRequestTable

Please note that "Examples" section for State Management 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.⌋

7.7.3 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.

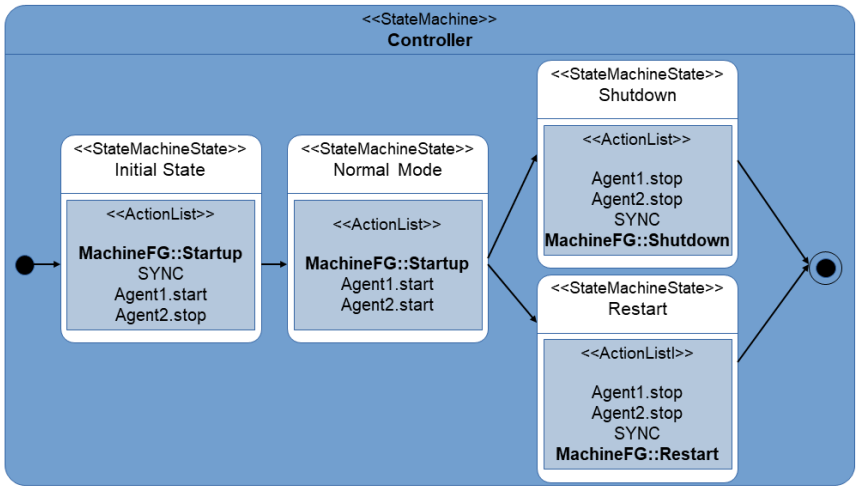

Figure 7.9: Example for Controller StateMachineStates with MachineFG

When Suspend-to-RAM is supported the Controller configuration needs further considerations. They are explained in detail in section 7.7.15.

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.⌋

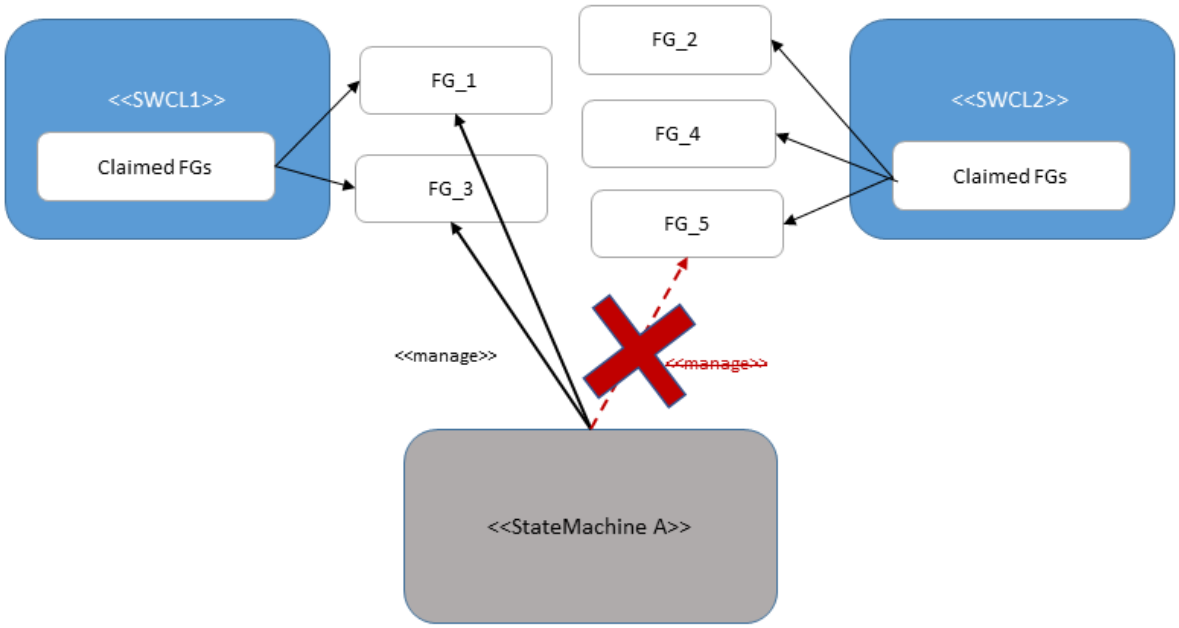

Figure 7.10: Agent - FunctionGroup relation

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.

7.7.4 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.

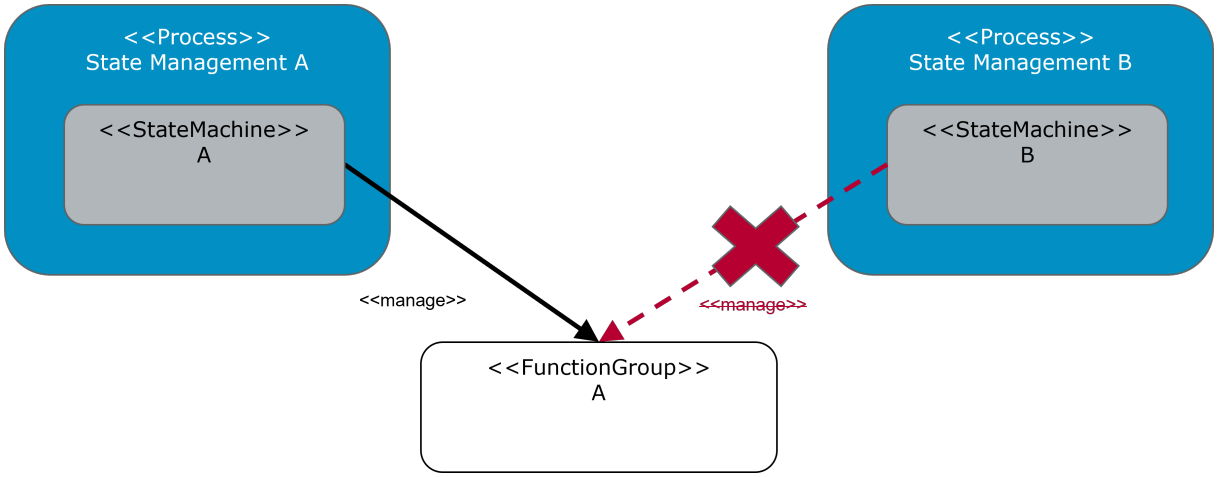

Figure 7.11: Function Group controlled by single StateMachine

[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.⌋

7.7.5 StateMachine state changes

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

[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 (see [SWS_SM_00601]) and shall cease any further processing of the request.⌋

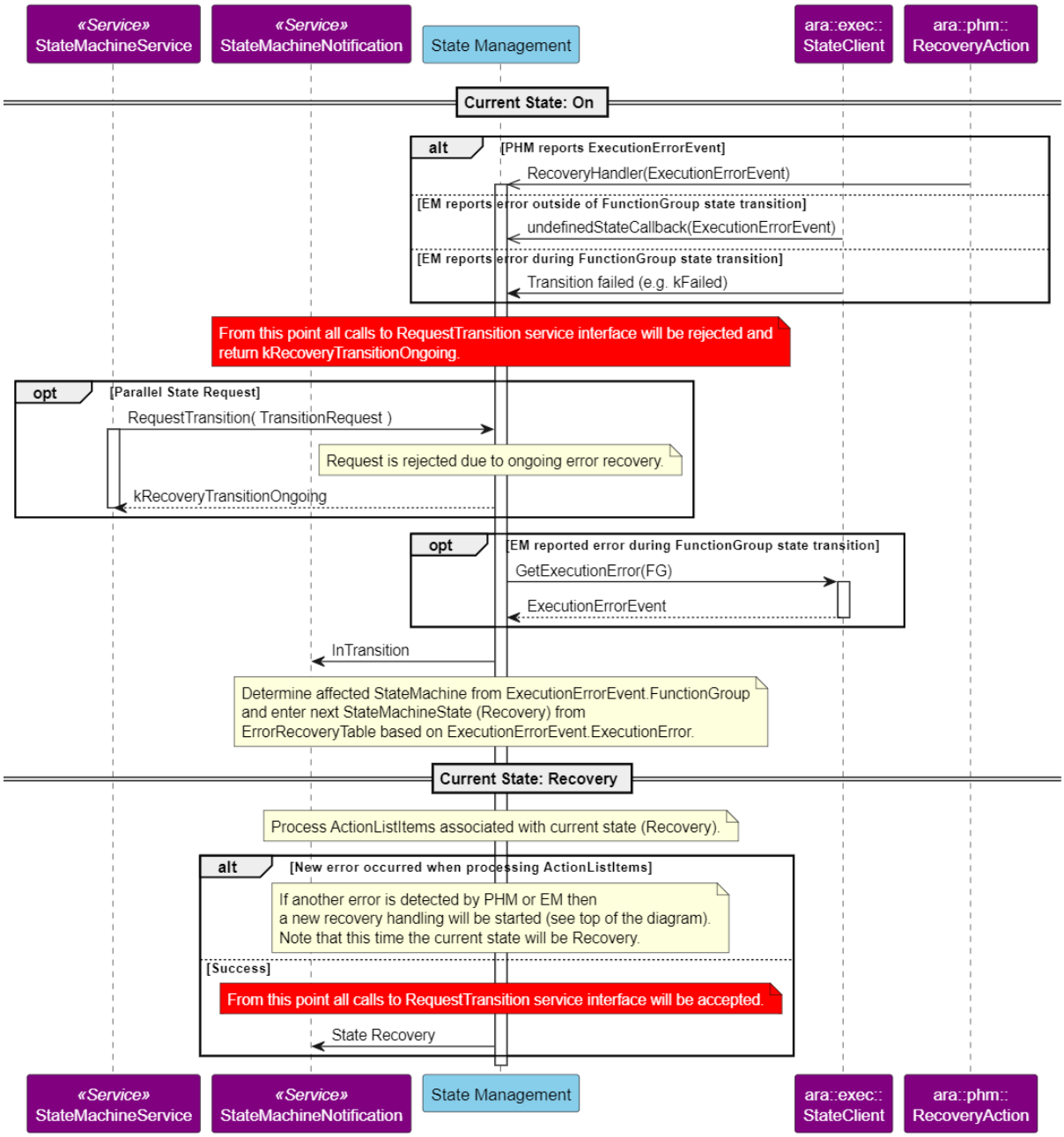

Figure 7.12: Error Recovery

[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_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`.

⌋

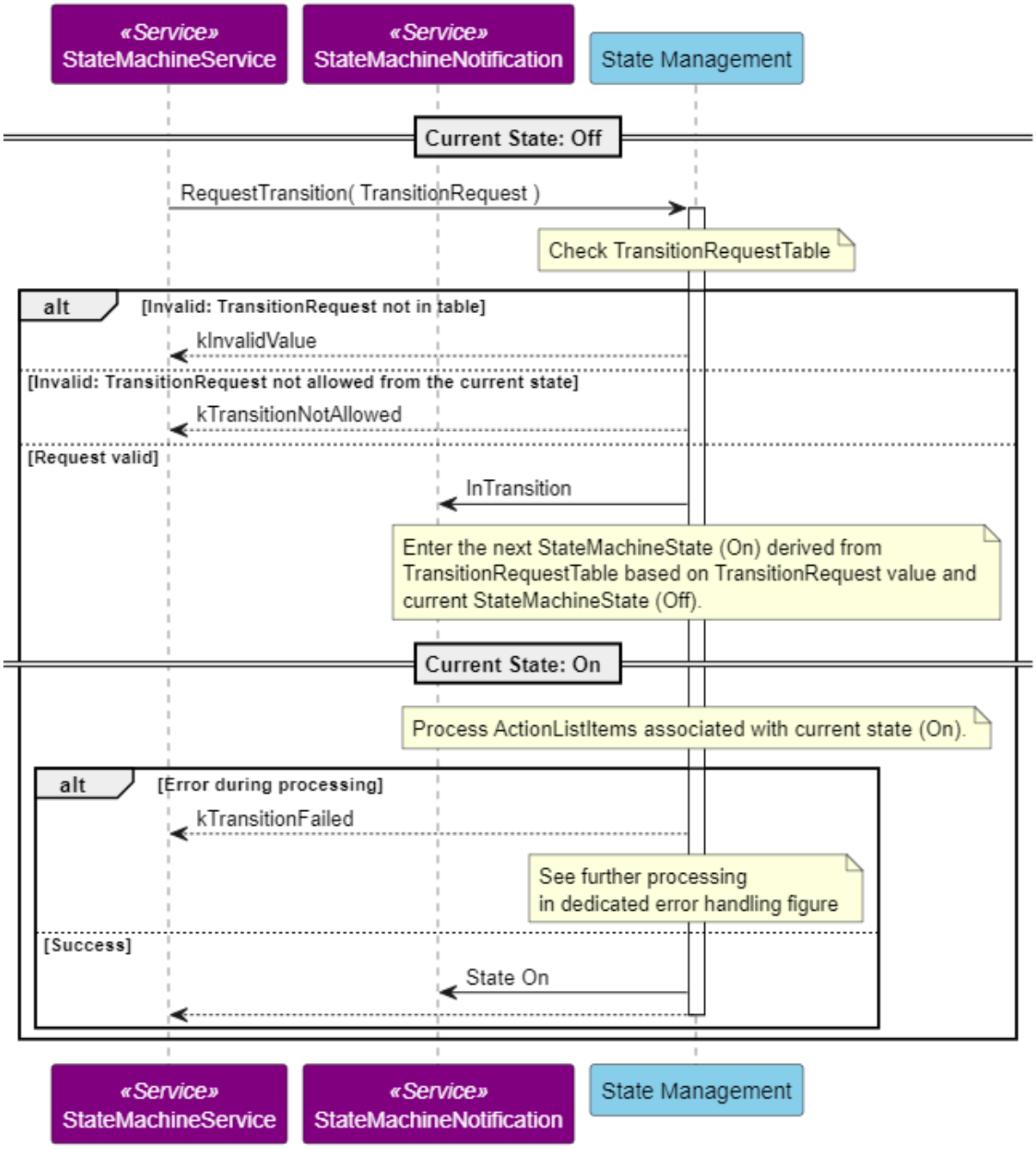

Figure 7.13: StateMachine change

[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: Network Management 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 7.14: 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.

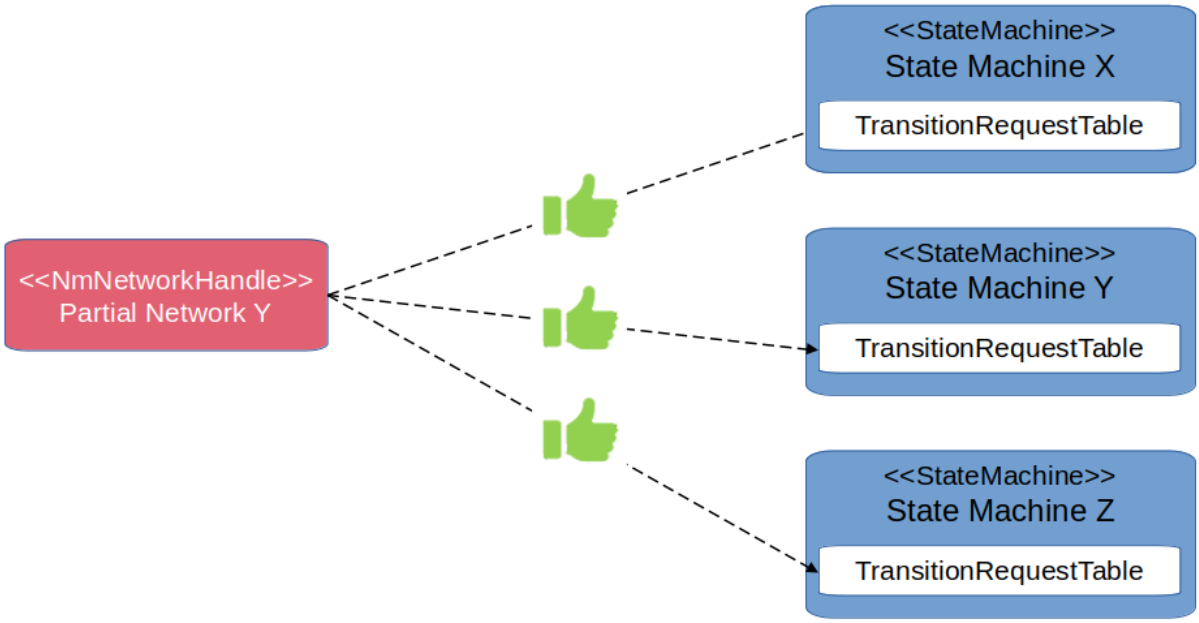

Figure 7.15: Example of one NmNetworkHandle influencing multiple StateMachines

7.7.6 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.

7.7.7 StateMachine ActionListItems

There are multiple kinds of ActionListItems:

- Requesting a Function Group State (corresponding to StateManagementSetFunctionGroupStateActionItem)
- Start a StateMachine with optional parameter state (corresponding to startAgent)
- Stop a StateMachine (corresponding to stopAgent)

- SYNC to sync between different ActionListItems (corresponding to StateManagementSyncActionItem)
- Sleep to delay processing the next ActionListItems (corresponding to StateManagementSleepActionItem)
- SetNetworkHandle switches the provided NetworkHandle to the configured state (NoCom or FullCom) (corresponding to StateManagementNmActionItem)
- EnterSuspendToRam used for requesting S2R-Aware applications to prepare for Suspend-to-RAM (corresponding to StateManagementEnterSuspendToRamActionItem)
- EnterSuspendToRamOS used to request the OS to enter in Suspend-to-RAM (corresponding to StateManagementEnterSuspendToRamOsActionItem)
- LeaveSuspendToRam used for requesting S2R-Aware applications to leave their Suspend-to-RAM state (corresponding to StateManagementLeaveSuspendToRamActionItem)

[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

⌈Processing of the ActionListItems in the ActionLists shall be started 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

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

In order to ensure that an `ActionList` is fully processed, an implicit SYNC command is assumed at the end of every `ActionList`.

[SWS_SM_00667] Finalization of ActionList processing

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈An `ActionList` is successfully processed as soon as all `ActionListItems`, configured for the current `ActionList`, have been executed and all results have been collected.⌋

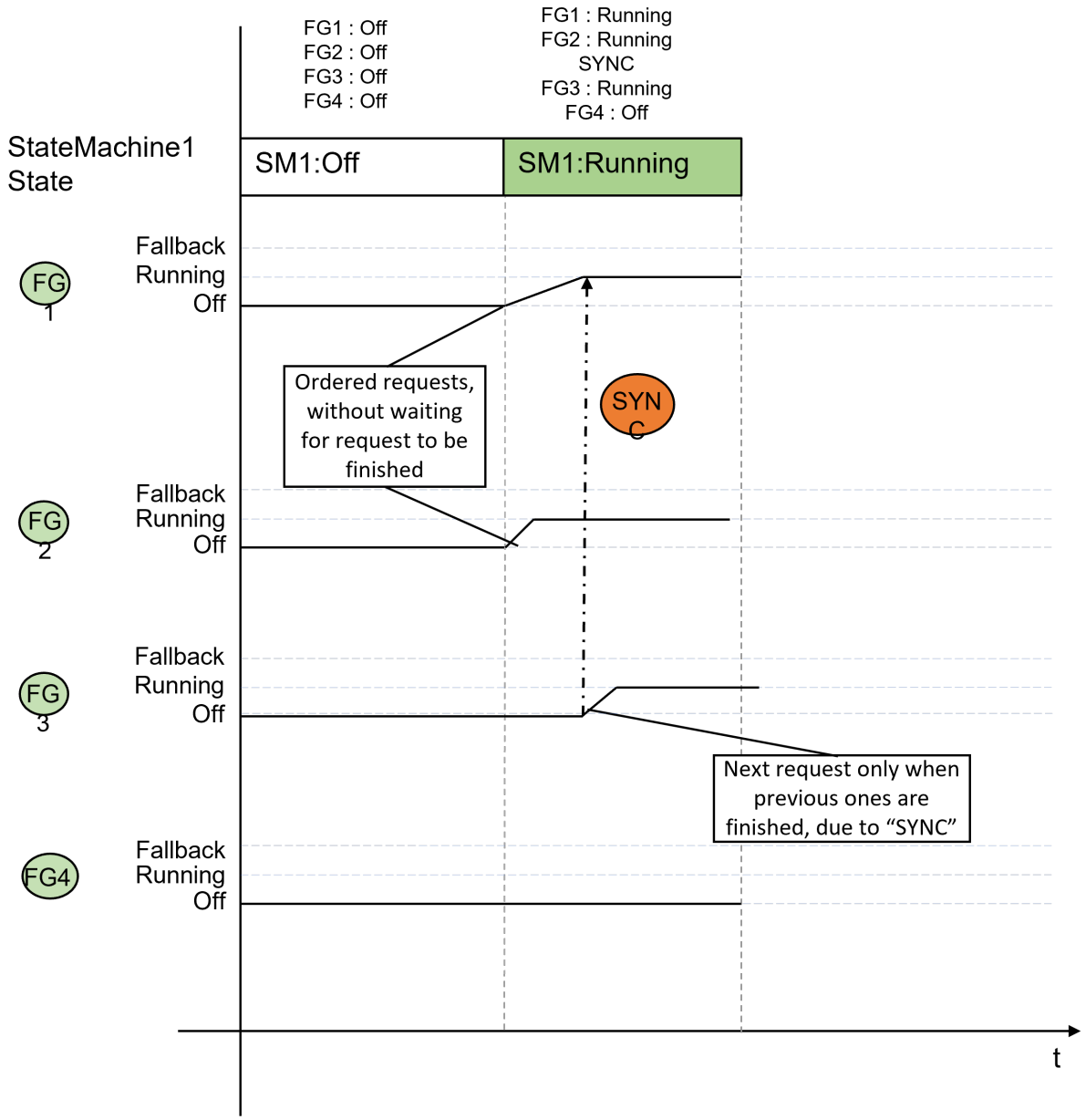

Figure 7.16: Parallel ActionListItem execution and SYNC

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.

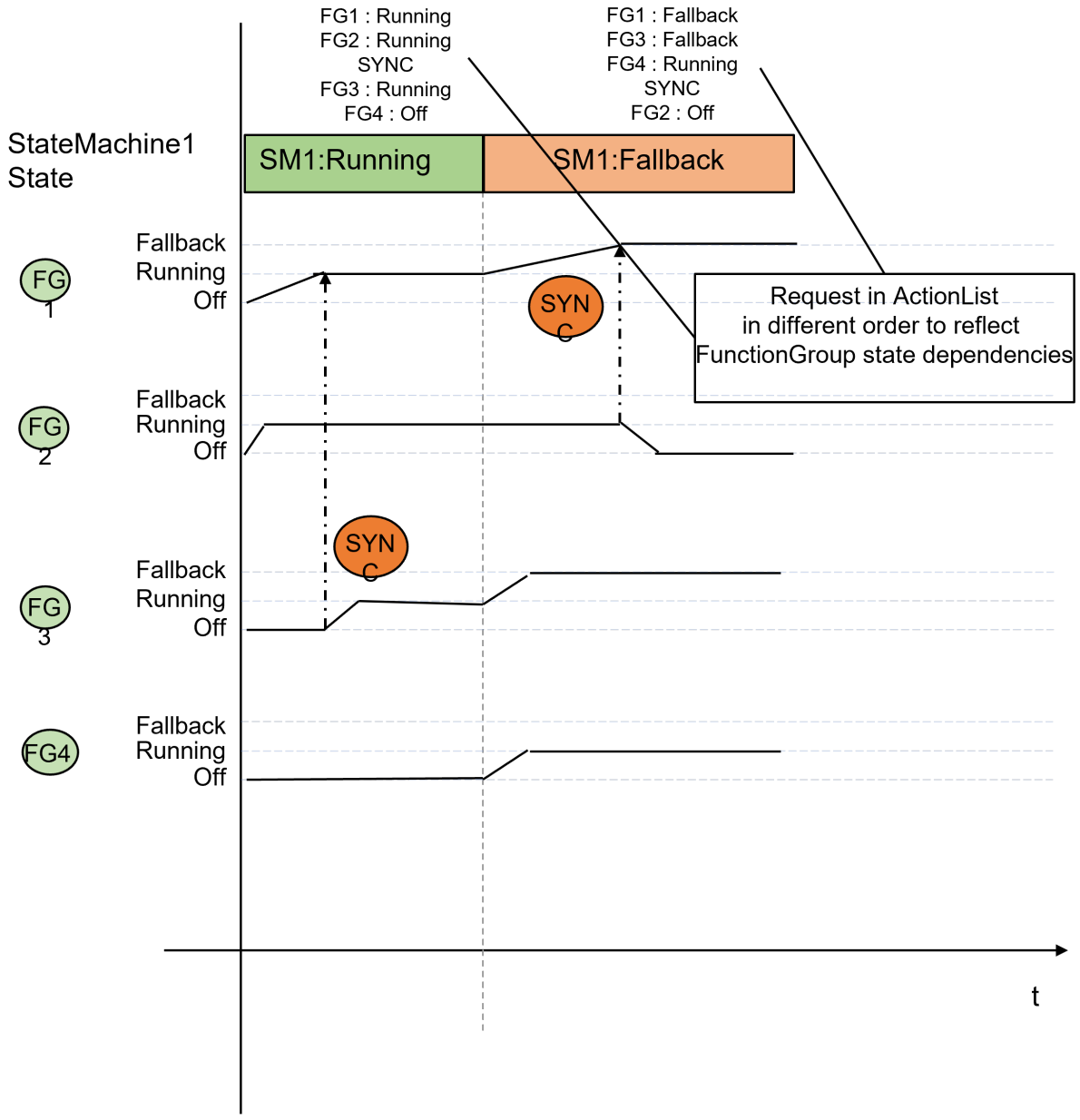

Figure 7.17: Arbitrary order for ActionListItems

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.⌋

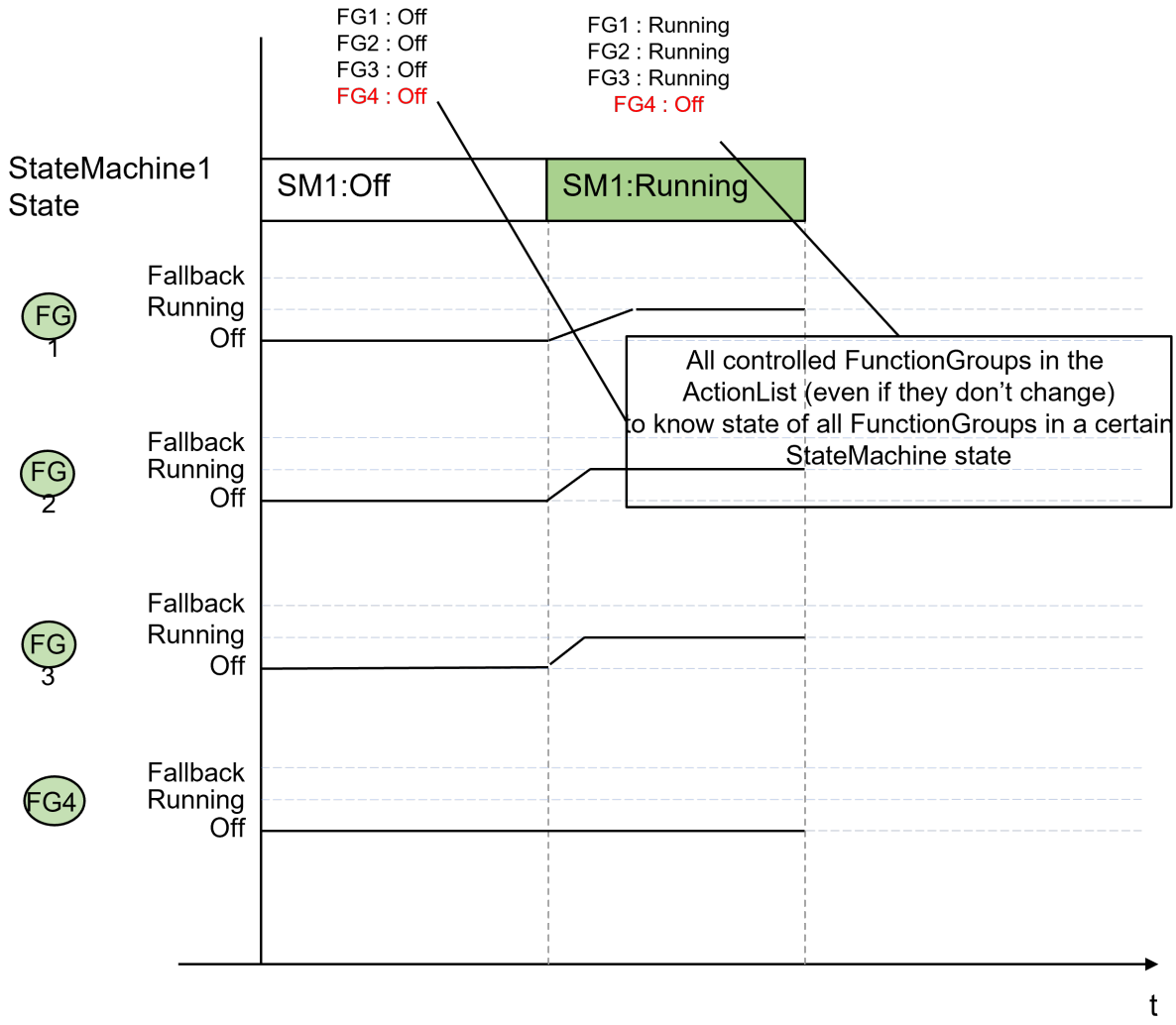

Figure 7.18: Completeness of controlled Function Groups

7.7.8 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 a StateMachines of type Controller, shall reference the complete set of Agents that are controlled by the Controller.⌋

The process of starting a StateMachine is a protracted operation that involves multiple steps. On the one hand, the configured interfaces, such as the StateMachineService and the StateMachineNotification, associated with the specific StateMachine must be initialized and provided to ensure that the StateMachine is accessible from the application layer. On the other hand, the StateMachine itself must initiate the processing of the ActionList that corresponds to the requested initial StateMachine State. This processing is governed by the specifications outlined in [SWS_SM_00612] or [SWS_SM_00622].

From the perspective of the Controller, a StateMachine is considered started once it exists and its associated interfaces are offered. It is important to note that the Controller does not require the completion of the initial ActionList processing to regard the StateMachine as started. This distinction is critical, as waiting for the completion of the initial ActionList would compromise time determinism. The duration required to process an ActionList is influenced by its size and complexity, making such a wait non-deterministic. This could, in turn, introduce delays in the Controller initial mode transition or recovery actions, which are expected to be executed within a short and predictable time frame.

[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.⌋

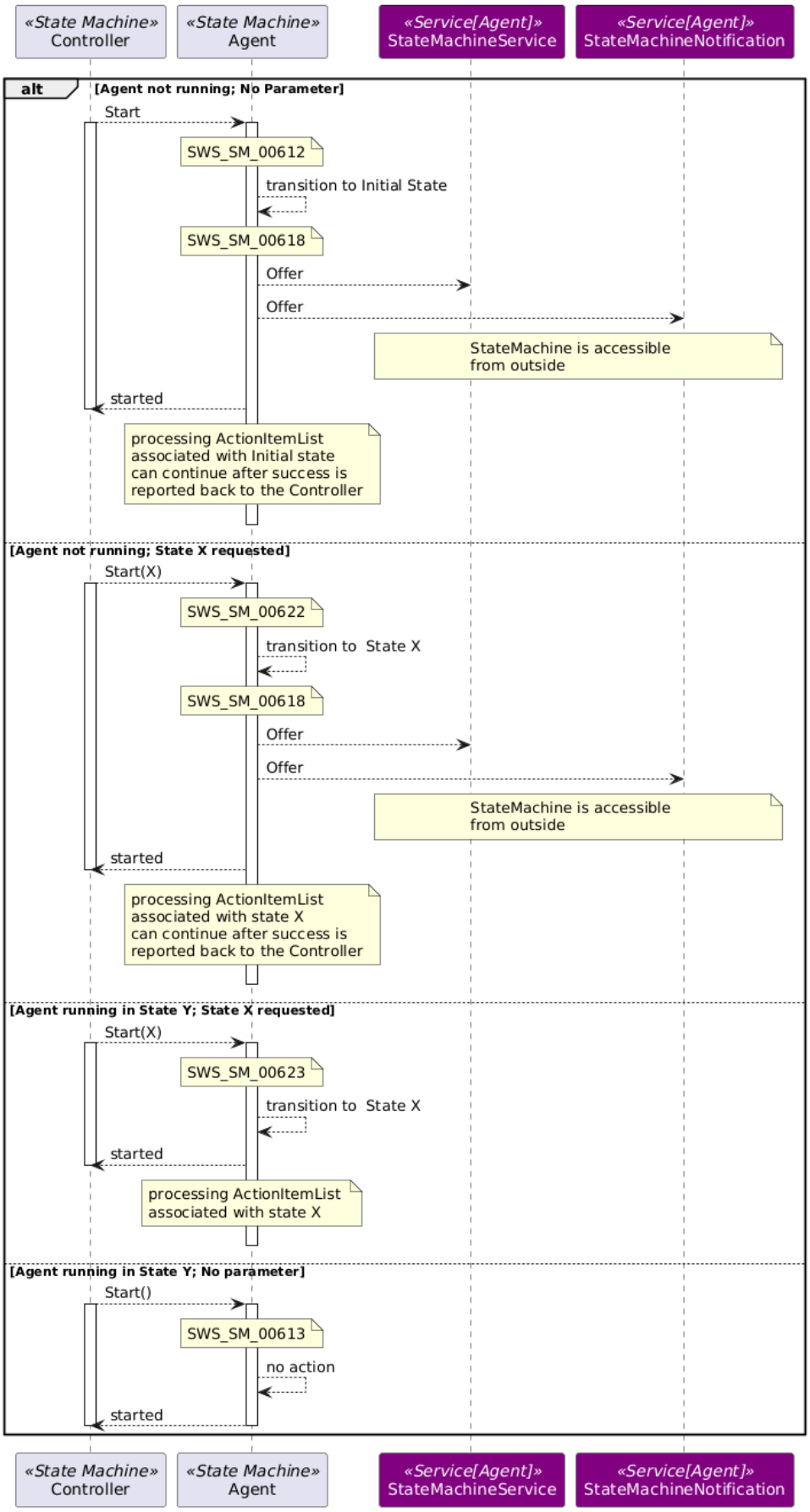

Figure 7.19: ActionListItem Start StateMachine

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].

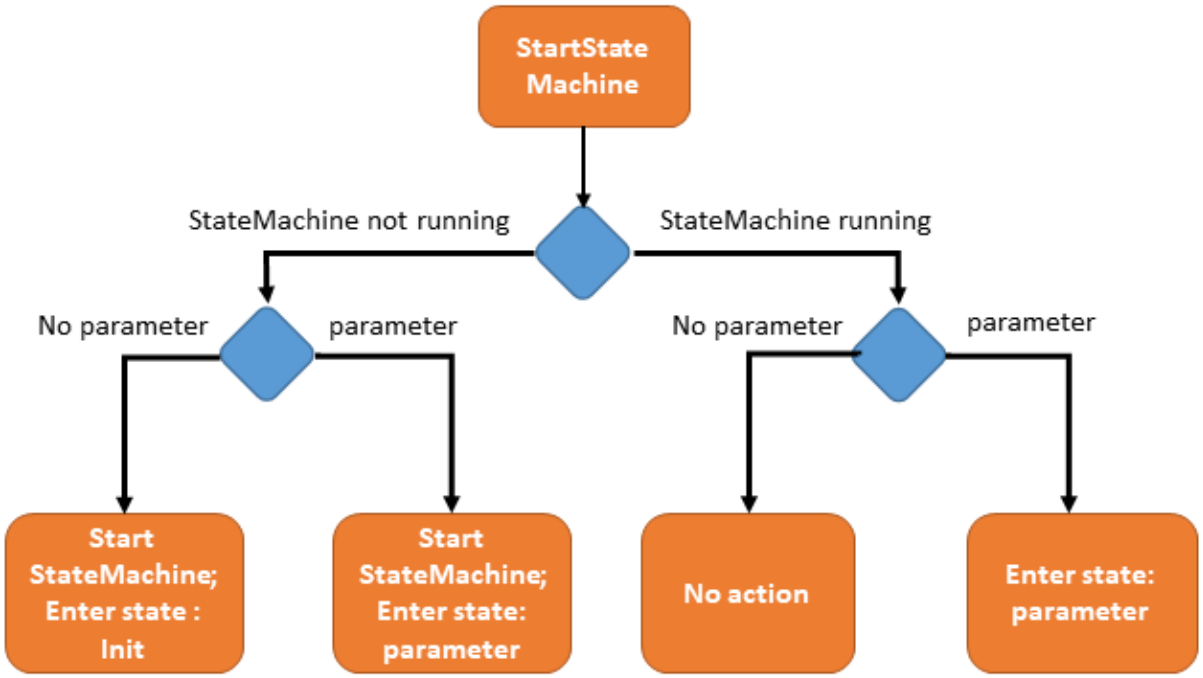

Figure 7.20: StartStateMachine decision tree

[SWS_SM_00614] ActionListItem "Stop StateMachine" processing

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈At the end of the processing of StopStateMachine ActionListItem, the referenced StateMachine shall cease to exist.⌋

During update session StateManagement needs to prepare parts of Machine for update. [SWS_SM_00614] together with [SWS_SM_00619] and [SWS_SM_00651] serves an important role here. They ensure that, when StopStateMachine is executed, the StateMachine is in a state where it could be safely updated by Update and Configuration Management.

[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

⌈During the processing of `ActionListItem` "StopStateMachine" the referenced `StateMachine` shall be transitioned to the "Off" `StateMachine State`.⌋

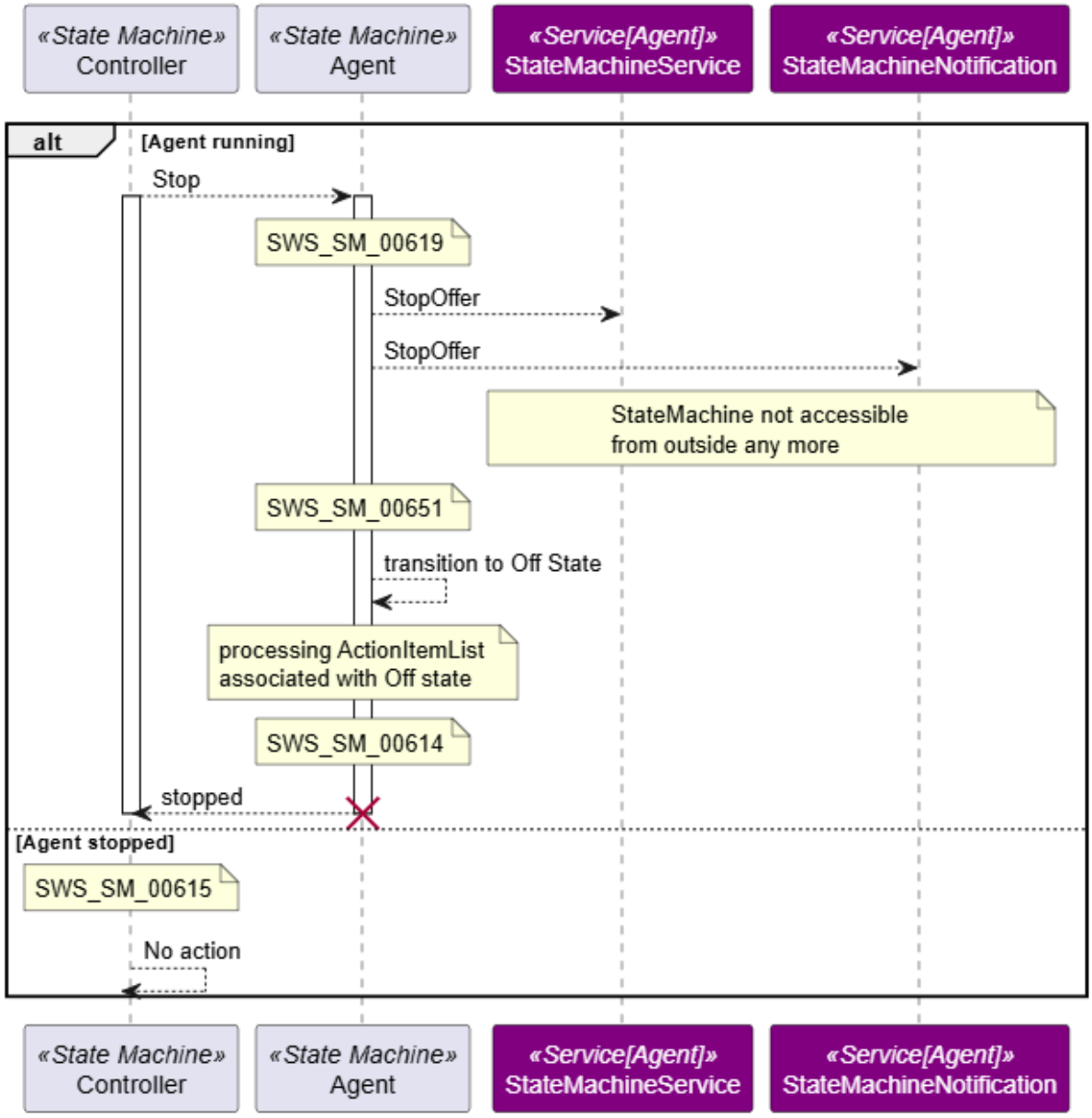

Figure 7.21: Processing ActionListItem StopStateMachine

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]). From the `Controller` perspective a stopped `StateMachine` means that the `StateMachine` does no longer exist. Therefore waiting that the Off State is reached and the configured interfaces from the `Agent` are no longer provided is mandatory

(see Figure: Processing ActionListItem StopStateMachine and [SWS_SM_00651]). 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.

7.7.9 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).

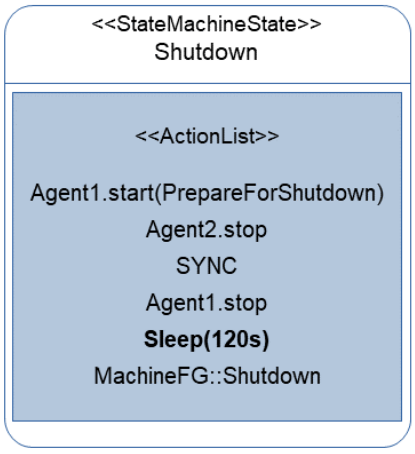

Figure 7.22: Example for an ActionList using ActionListItem Sleep

7.7.10 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.⌋

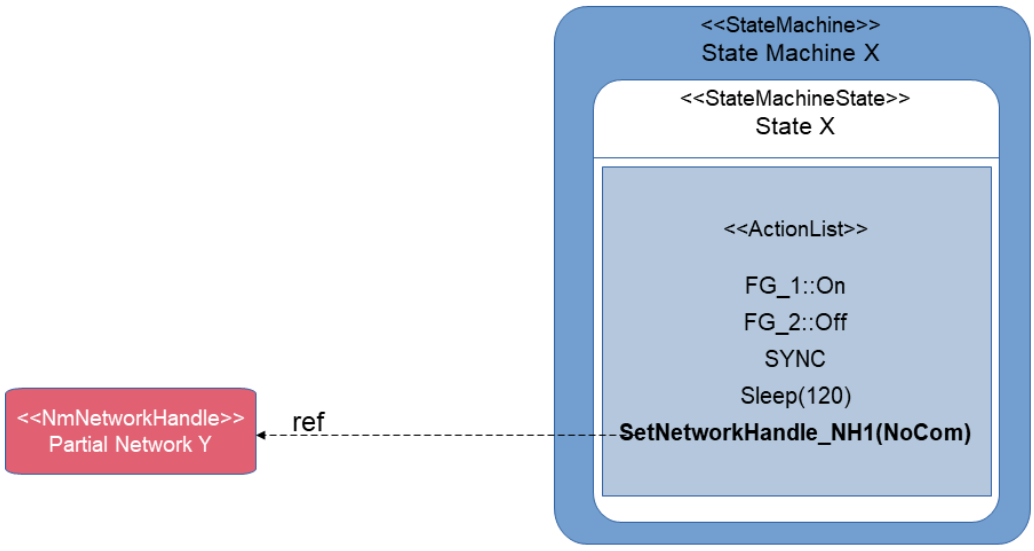

Figure 7.23: Afterrun example using the SetNetworkHandle in combination with Sleep

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`.⌋

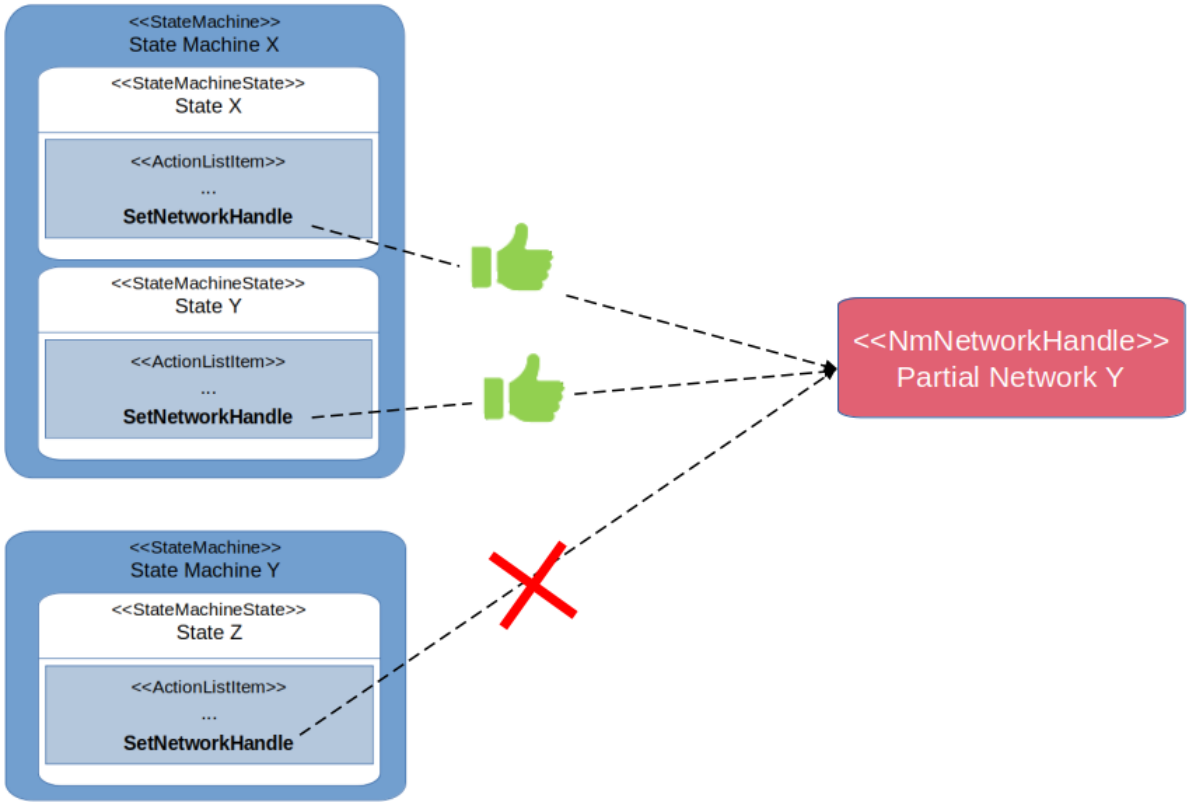

Figure 7.24: StateMachine to NmNetworkHandle restriction

7.7.11 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`.

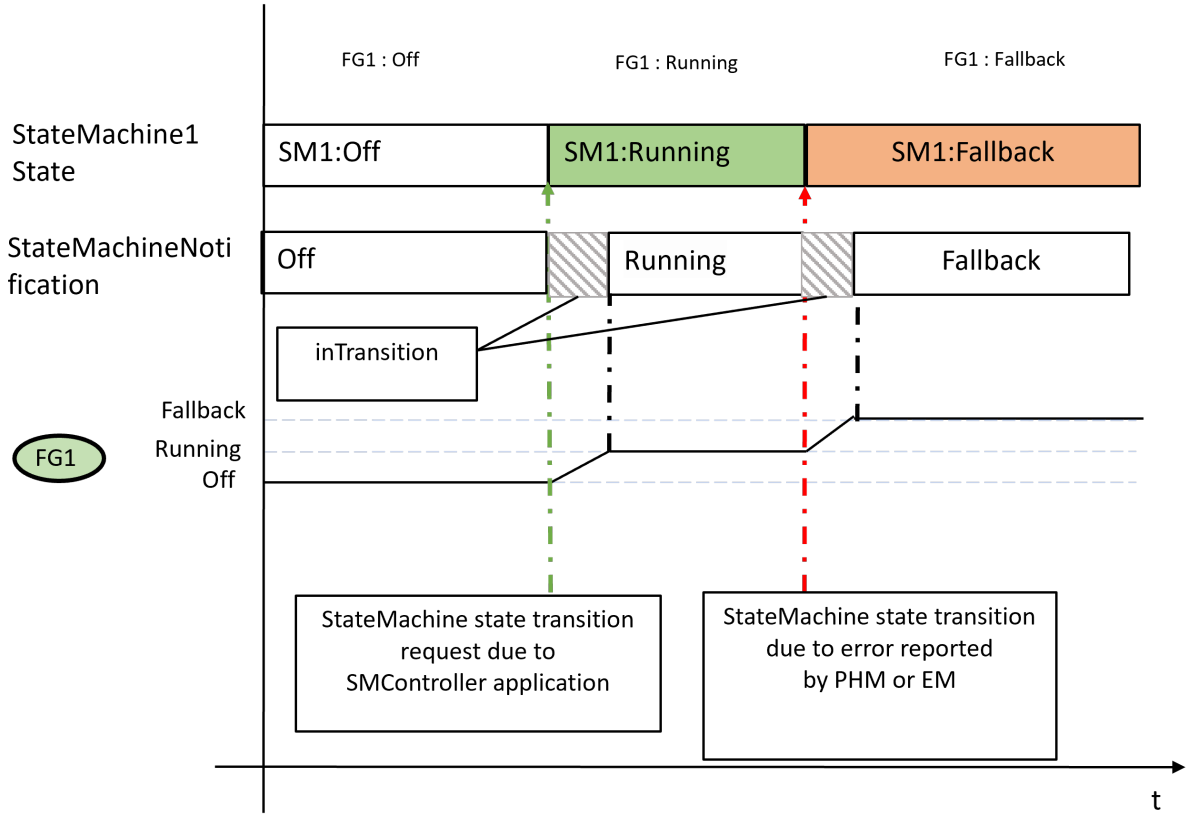

Figure 7.25: Value of configured StateMachineNotification::CurrentState field

[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".⌋

7.7.12 StateMachine ActionListTimeout

Processing an ActionList cannot be guaranteed to be free from errors. There are explicit errors which can be easily detected, like an unexpected termination of a Process or a failed transition to a Function Group State. However, there may be a situation where a StateMachine State transition, i.e. the processing of an ActionList, may take too much time even though no error is detected. Here are some examples:

- Wrong configuration, like setting too large Sleep values.
- Process is not able to start, because it has an ExecutionDependency to a self-terminating Process, which is not terminating.
- Low performance on the target making the initialization of Agents, and therefore startup of Processs, taking longer than expected.

The ActionListTimeout is a configurable value which follows a simple approach:

- Value set as default value valid for all ActionLists within a Machine.
- Value set per specific ActionList, which overrides the global value for that ActionList.

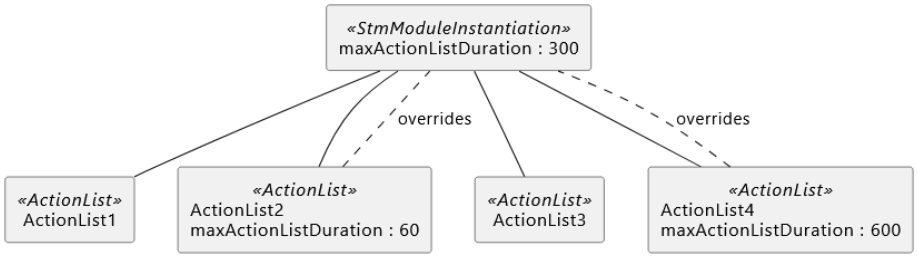

Figure 7.26: ActionList timeout

[SWS_SM_00668] Default value for ActionListTimeout

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈When StateManagementActionList.maxActionListDuration is not configured, State Management shall use StateManagementModuleInstantiation.maxActionListDuration (aka the global ActionListTimeout).⌋

[SWS_SM_00669] ActionList timeout monitoring

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈State Management shall monitor the time required to process the ActionList (the time between processing the first ActionListItem and receiving the last response associated with ActionList (see [SWS_SM_00667]).⌋

7.7.13 StateMachine ErrorCode configuration and handling

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, Execution Management or State Management internal error. To achieve this, a mapping table, the ErrorRecoveryTable is introduced, which maps ErrorCodes produced by failed processes, failed Function Group transition requests, failed NmNetworkHandle transition requests and failed StateMachine transitions to the required nextState ("recovery") StateMachine State.

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

Figure 7.27: ErrorRecoveryTable

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

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

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

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 an ErrorCode is reported and the StateMachine is not "ImpactedByUpdate", StateMachine shall:

- set internal flag that error recovery is ongoing
- evaluate the next StateMachine State configured for ErrorCode 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 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]).⌋

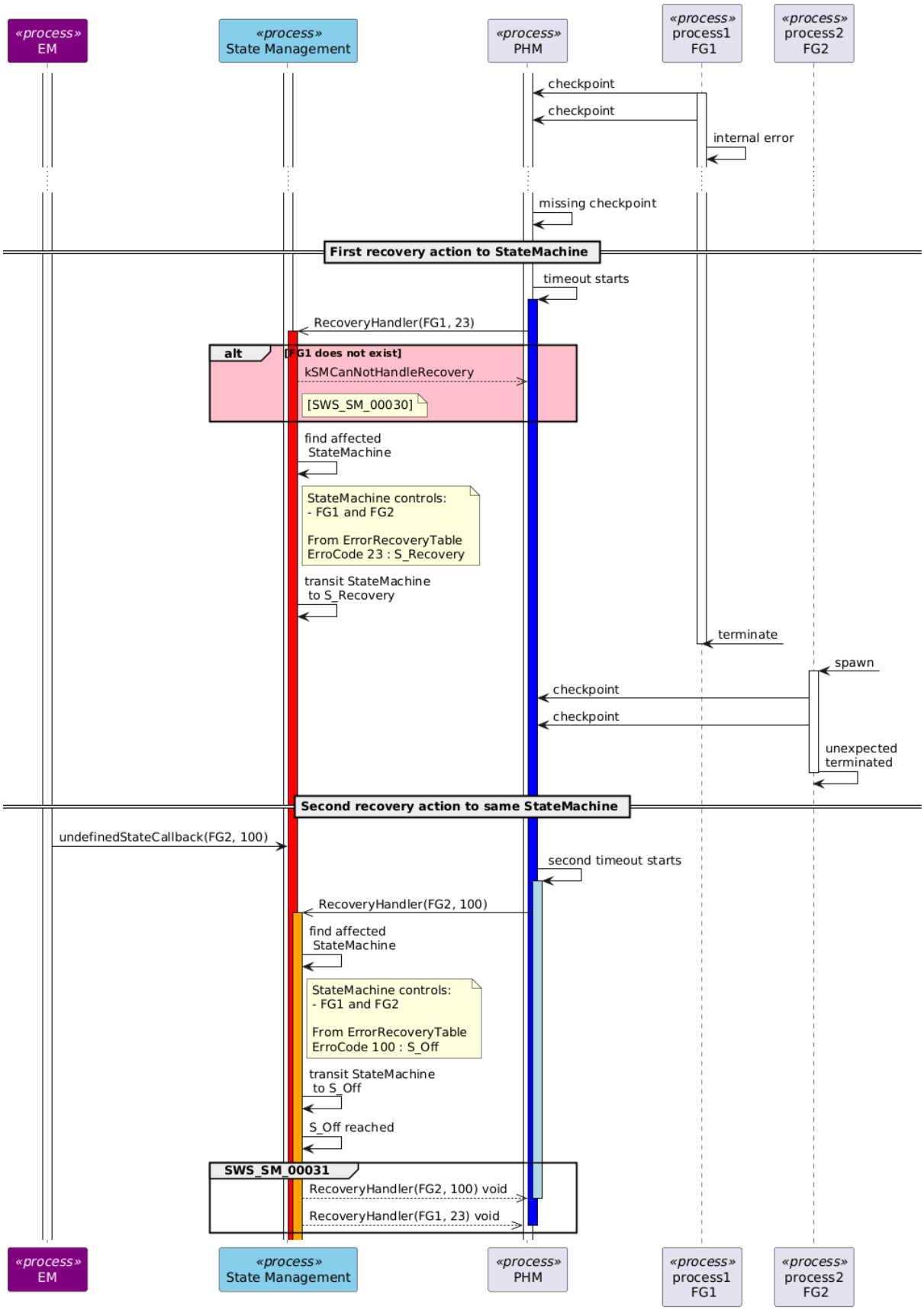

Figure 7.28: Nested recovery example

[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.⌋

The following list presents an overview of the errors, which can be detected and handled by StateManagement:

- Errors reported by Platform Health Management or Execution Management due to a detected failure at Process level (see [SWS_SM_00670]).
- Errors reported by ara::exec::StateClient related to failed or invalid requests as well as ara::exec::StateClient communication issues (see [SWS_SM_00671]).
- Errors at StateMachine transitions, where processing the complete ActionList overpassed the specified timeout (see [SWS_SM_00673]).
- Errors detected when a StateMachine of type Agent fails to start or to stop (see [SWS_SM_00675] and [SWS_SM_00677]).
- Errors from Agent transitions, where Controller recovery actions are needed (see [SWS_SM_00679]).
- Errors issued when requesting S2R-Aware applications to enter or leave the Suspend-to-RAM mode fails (see [SWS_SM_00685] and [SWS_SM_00687])

All previous errors can be configured as ErrorCodes in the ErrorRecoveryTables from the respective StateMachines.

[SWS_SM_00670] ErrorCode as reaction to a failed Process

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005

⌈The ExecutionErrorEvent::ExecutionError provided by Execution Management via StateClient::GetExecutionError or StateClient::undefinedStateCallback as well as from Platform Health Management via the RecoveryHandler shall be used as ErrorCode in the StateMachine which manages the Function Group having StartupConfigDependencies with the failed Process.⌋

There are two possible failed responses from the StateClient::SetState towards State Management:

- The requested transition was accepted and started but errors at Process level occurred and therefore the transition failed.
- The request itself failed, due to a failed communication to Execution Management or the transition request is invalid.

The first case will return the mapped ExecutionError from the failed process and a recovery action can be performed (see [SWS_SM_00670]). For the second case the ErrorCode functionGroupTransitionRequestFailedError mapped to FunctionGroupErrorMapping can be configured.

[SWS_SM_00671] ErrorCode as reaction to a failed SetState request

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005

⌈If the SetState error is "ara::exec::ExecErrc::kNoCommunication", "ara::exec::ExecErrc::kInvalidTransition" or "ara::exec::ExecErrc::kInvalidMetaModelIdentifier" the configured functionGroupTransitionRequestFailedError mapped to FunctionGroupErrorMapping shall be used as ErrorCode in the StateMachine which manages the affected Function Group.⌋

[SWS_SM_00672] Default value for functionGroupTransitionRequestFailedError

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005

⌈In case functionGroupTransitionRequestFailedError does not have an ErrorCode configured, State Management shall use the ErrorCode value 1.⌋

There is a class of errors, which are more difficult to detect. This includes but is not limited to:

- State Management can request state changes to NmNetworkHandle ([SWS_SM_00625] and [SWS_SM_00626]) there is still the possibility that the change does not take place and the that the NmNetworkHandle remains in its current state.
- Project could be deployed on a low performance hardware.
- Delays introduced by ExecutionDependencies.
- Programming errors in applications, e.g. infinite loops.

This kind of errors can be easily detected with ActionListTimeout. In case detection of such errors is needed, the maxDurationExceededError mapped to the StateManagementActionList can be used.

[SWS_SM_00673] ErrorCode as reaction to an ActionList timeout

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005

⌈If processing an ActionList results in a timeout [SWS_SM_00669], then the maxDurationExceededError shall be used as ErrorCode in the StateMachine which manages the ActionList.⌋

[SWS_SM_00674] Default value for StateManagementActionList.maxDurationExceededError

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005

⌈In case `maxDurationExceededError` does not have an ErrorCode configured, `State Management` shall use the ErrorCode value 1.⌋

If a `StateMachine` of type `Agent` fails to be started or stopped by the `StateMachine` of type `Controller`, a recovery action is needed. Please note that these errors refer to the process of creating and destroying the `StateMachine` of type `Agent` as well as its service interfaces, and not about errors during processing the `ActionLists` mapped to the InitialMode and Off States from the `StateMachine`. In order to configure these kind of errors the `startAgentError` and `stopAgentError` mapped to the `StateMachine` itself will be used.

[SWS_SM_00675] ErrorCode for failed creation of a StateMachine of type Agent

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005

⌈If the `StateMachine` of type `Controller` detects an error during creation of a `StateMachine` of type `Agent`, the configured `startAgentError` mapped to the affected `StateMachine` shall be used as ErrorCode in the `Controller ErrorRecoveryTable`.⌋

[SWS_SM_00676] Default value for startAgentError

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005

⌈In case `startAgentError` does not have an ErrorCode configured, `State Management` shall use the ErrorCode value 1.⌋

[SWS_SM_00677] ErrorCode for failed termination of a StateMachine of type Agent

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005

⌈If the `StateMachine` of type `Controller` detects an error during destruction of a `StateMachine` of type `Agent`, the configured `stopAgentError` mapped to the affected `StateMachine` shall be used as ErrorCode in the `Controller ErrorRecoveryTable`.⌋

[SWS_SM_00678] Default value for stopAgentError

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005

⌈In case `stopAgentError` does not have an ErrorCode configured, `State Management` shall use the ErrorCode value 1.⌋

When a StateMachine of type Agent detects an error, a recovery action will be started based on the Agent's ErrorRecoveryTable. Usually those errors are handled in the Agent's scope and are not visible to the StateMachine of type Controller. Sometimes however a successful processing of an ActionList, associated with an Agent StateMachine State, can be important enough that, any error detected during the execution of the mentioned ActionList, should be signaled back to the Controller. In this case the actionListProcessingFailedError can be used.

[SWS_SM_00679] ErrorCode for failure during processing of an Agent ActionList

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005

⌈If the processing of the Agent ActionList resulted in a failure and actionListProcessingFailedError is configured for that ActionList, then actionListProcessingFailedError shall be used as ErrorCode in the Controller ErrorRecoveryTable.⌋

When Suspend-to-RAM is supported and the Controller processes the ActionListItems EnterSuspendToRam and LeaveSuspendToRam, error conditions may occur during processing. A typical failure scenario arises when a S2R-Aware application either does not respond to the request or explicitly rejects the transition - refusing to enter or exit the Suspend-to-RAM state.

To support error tracing in such cases, the enterSuspendToRamError shall be used to report failures encountered during the execution of EnterSuspendToRam. Similarly, the leaveSuspendToRamError shall be used to report errors during the execution of LeaveSuspendToRam.

[SWS_SM_00685] ErrorCode for failed EnterSuspendToRam

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005, RS_SM_00402

⌈If the StateMachine of type Controller detects an error when processing the EnterSuspendToRam ActionListItem, the configured enterSuspendToRamError shall be used as ErrorCode in the Controller's ErrorRecoveryTable.⌋

[SWS_SM_00686] Default value for enterSuspendToRamError

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005, RS_SM_00402

⌈In case enterSuspendToRamError does not have an ErrorCode configured, State Management shall use the ErrorCode value 1.⌋

[SWS_SM_00687] ErrorCode for failed LeaveSuspendToRam

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005, RS_SM_00402

⌈If the `StateMachine` of type `Controller` detects an error when processing the LeaveSuspendToRam `ActionListItem`, the configured `leaveSuspendToRamError` shall be used as ErrorCode in the `Controller`'s `ErrorRecoveryTable`.⌋

[SWS_SM_00688] Default value for leaveSuspendToRamError

Status: DRAFT
Upstream requirements: RS_SM_00001, RS_SM_00005, RS_SM_00402

⌈In case `leaveSuspendToRamError` does not have an ErrorCode configured, `State Management` shall use the ErrorCode value 1.⌋

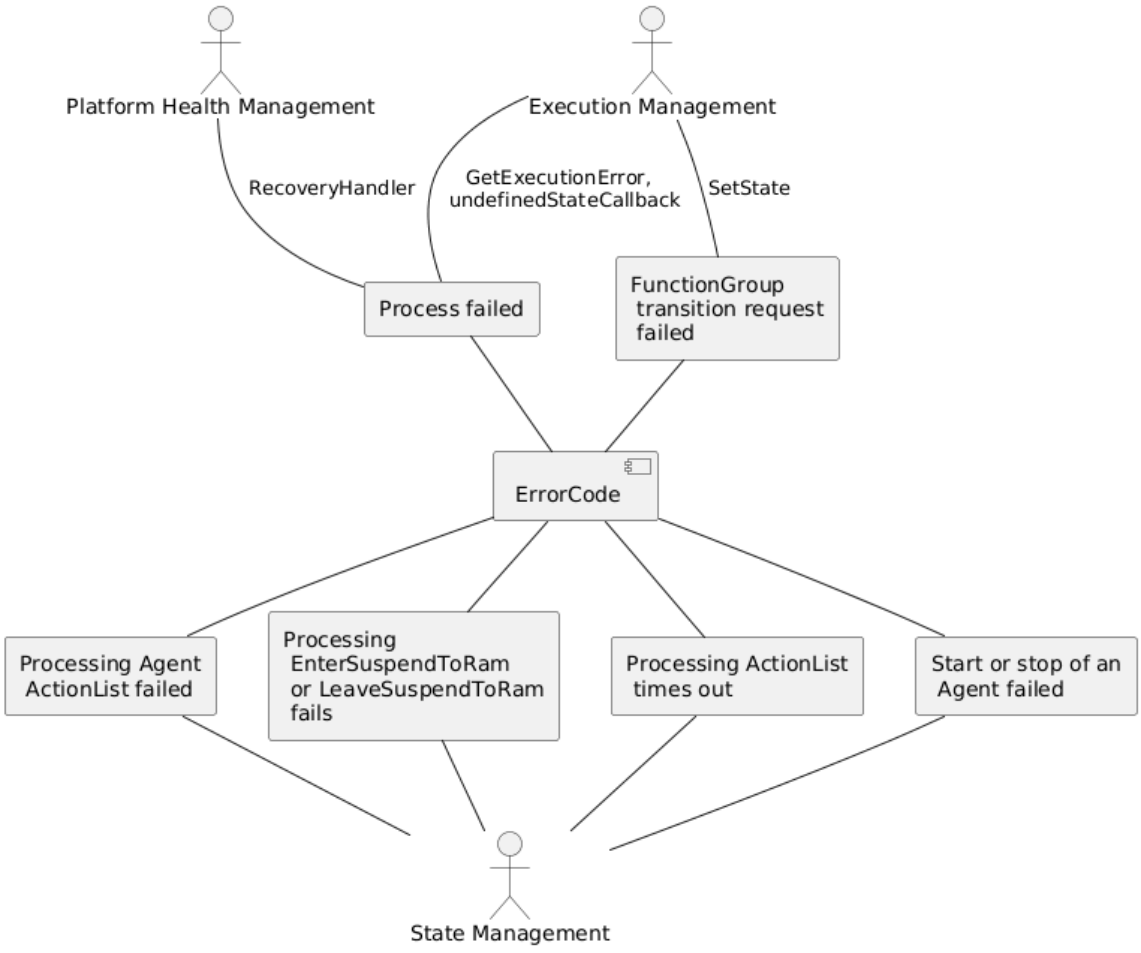

Figure 7.29: Overview for the StateMachine ErrorCodes handling

7.7.14 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.

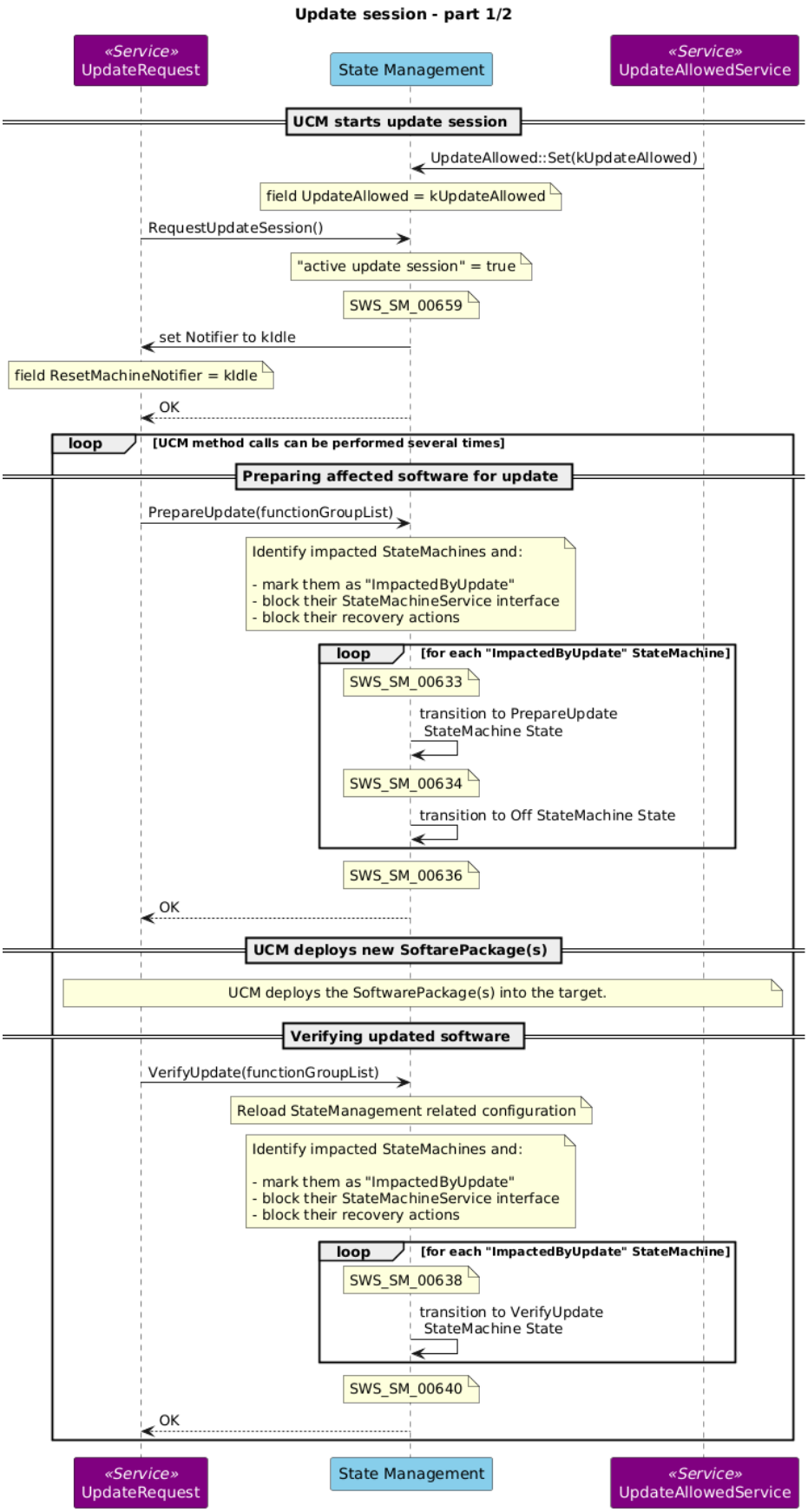

Figure 7.30: Overview of update session within StateMachine approach (part 1 of 2)

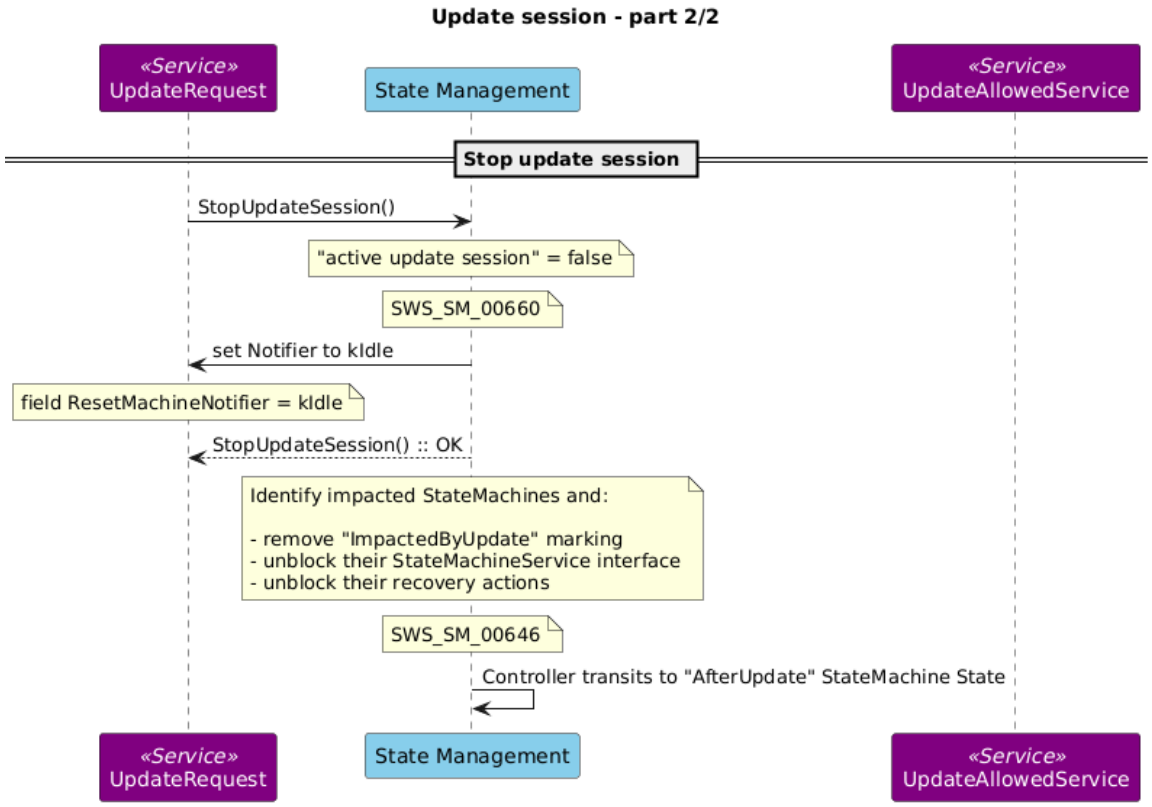

Figure 7.31: Overview of update session within StateMachine approach (part 2 of 2)

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 a Machine, there shall be at most one Port typed by the UpdateRequest interface.⌋

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

Upstream requirements: RS_SM_00001, RS_SM_00005

⌈If a Modelled Process controlling StateMachine of type Controller has a Port configured that is typed by the UpdateRequest interface, it shall 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.

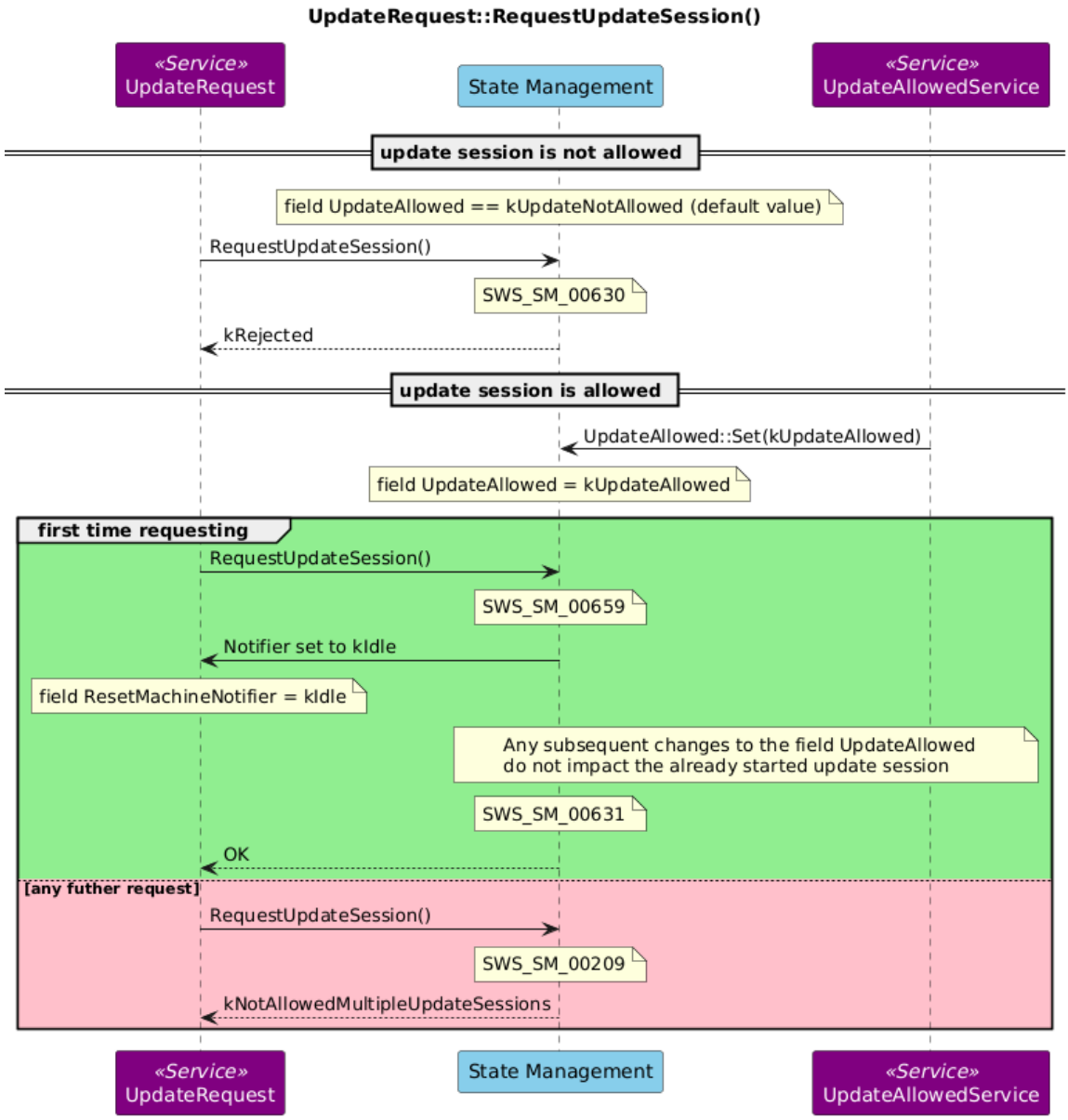

Figure 7.32: Requesting update session within StateMachine approach

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.

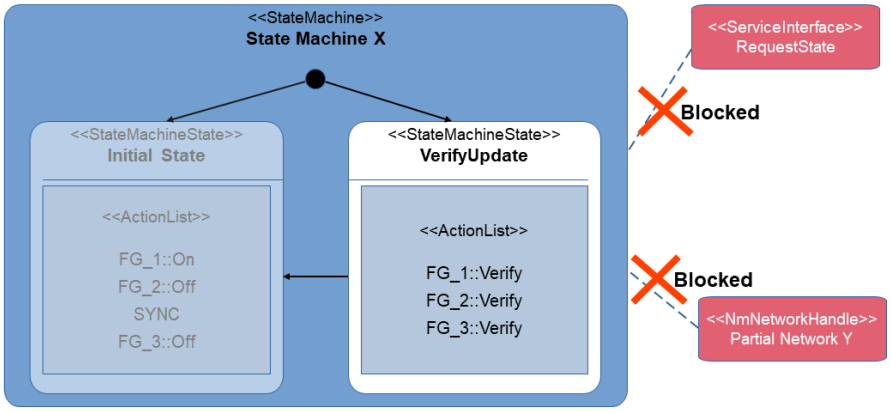

Figure 7.33: Example for StateMachineState VerifyUpdate for an Agent

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.

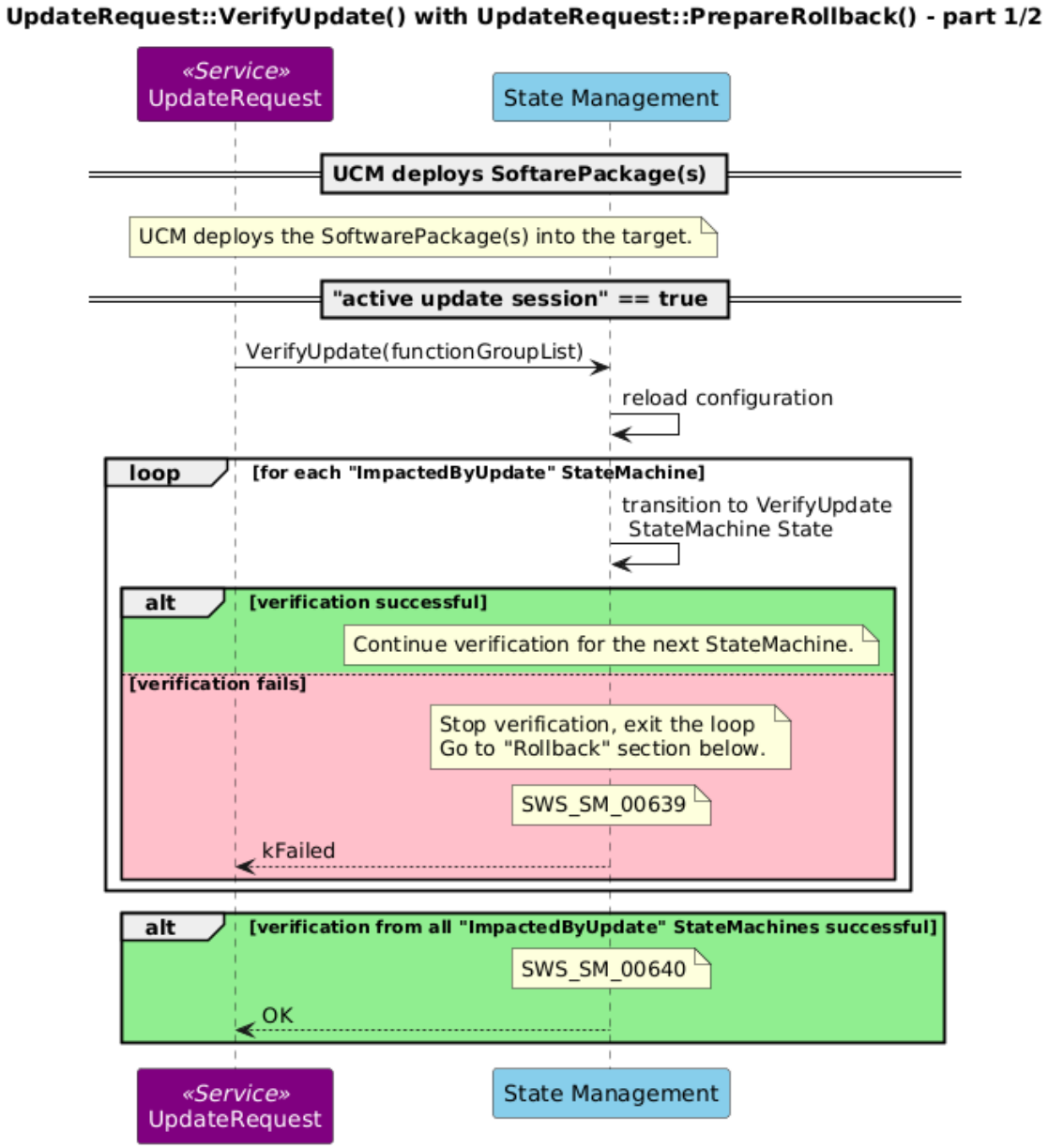

Figure 7.34: Verify and prepare rollback within StateMachine approach - part 1/2

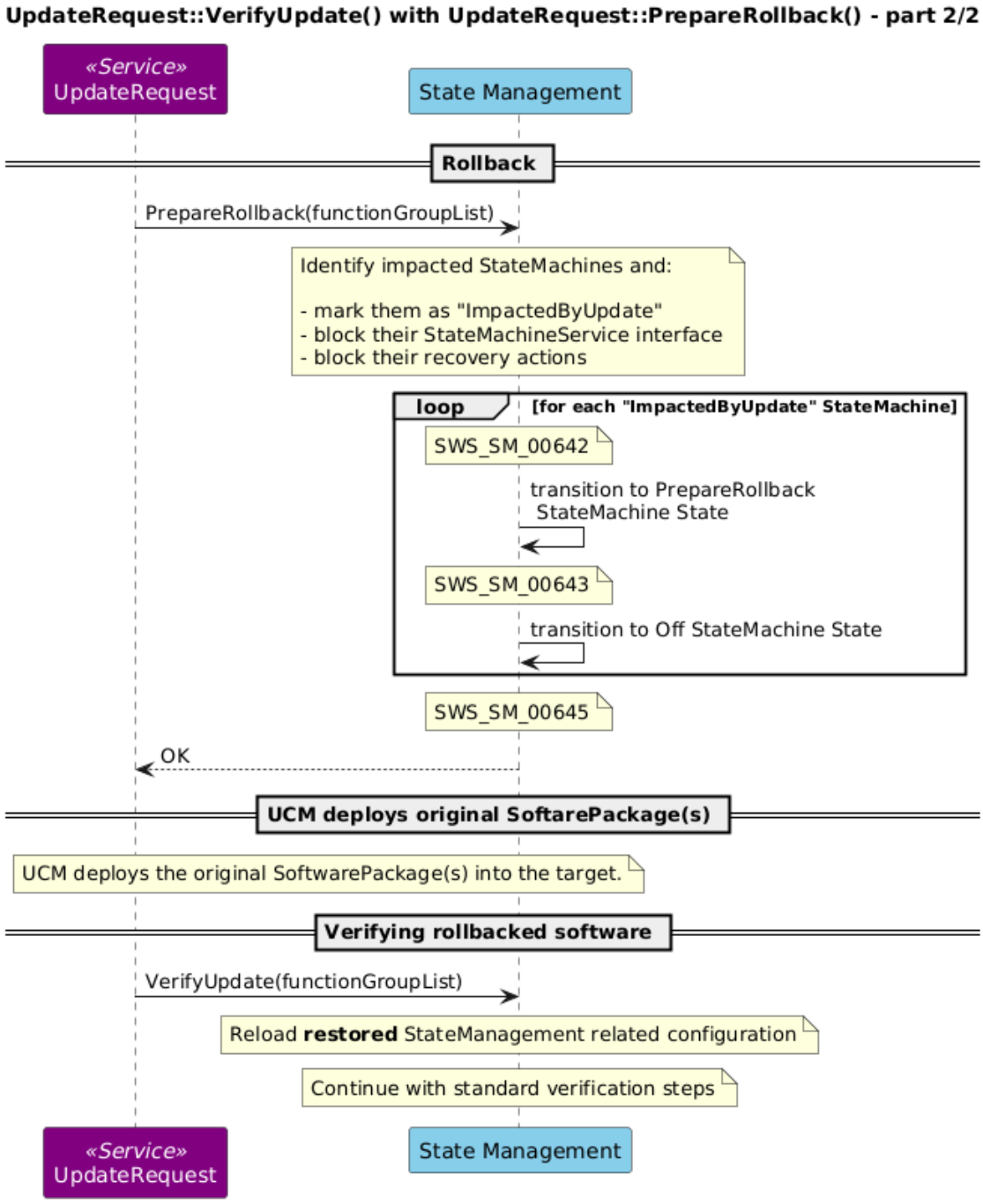

Figure 7.35: Verify and prepare rollback within StateMachine approach - part 2/2

[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.

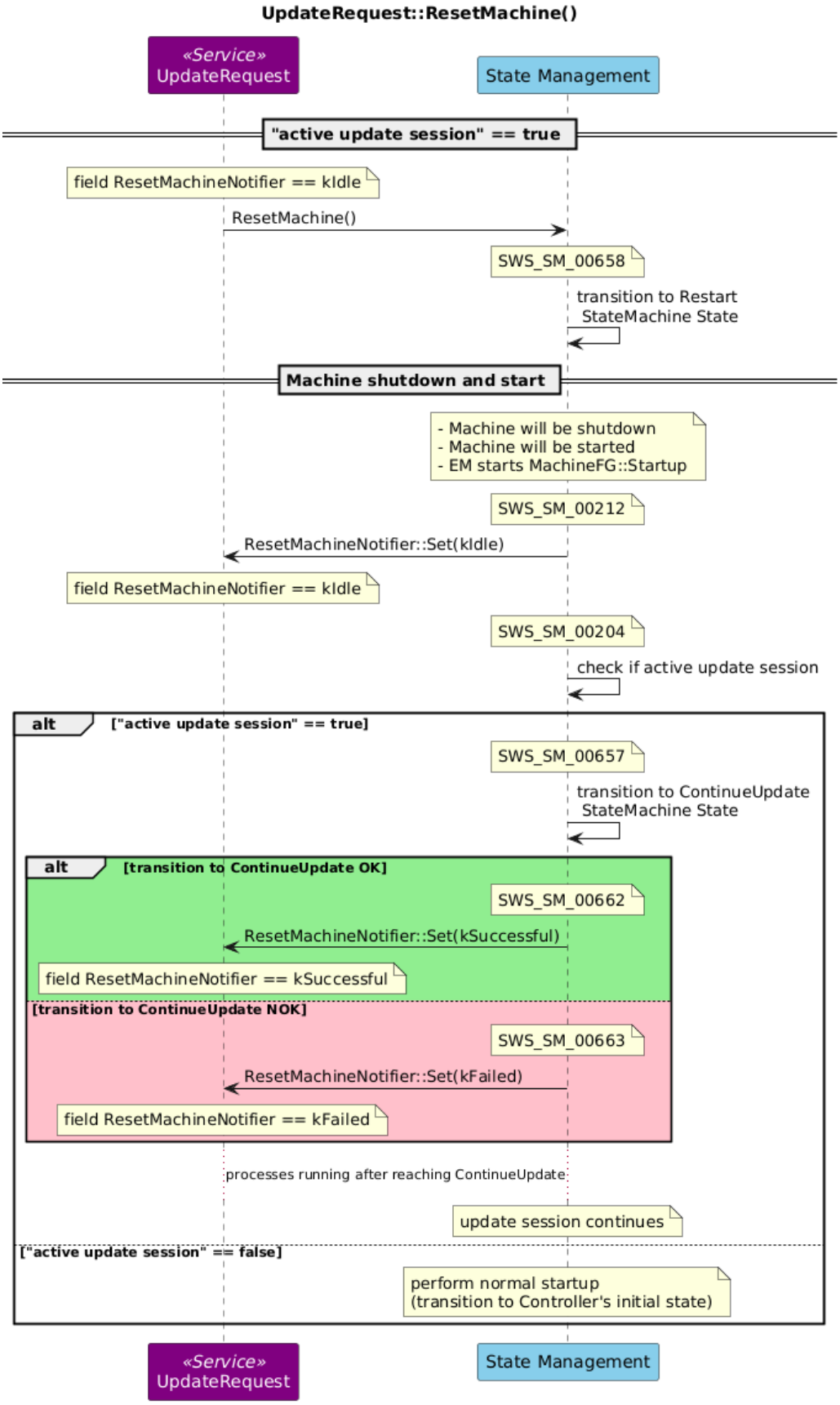

Figure 7.36: Reset machine handling within StateMachine approach

[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.⌋

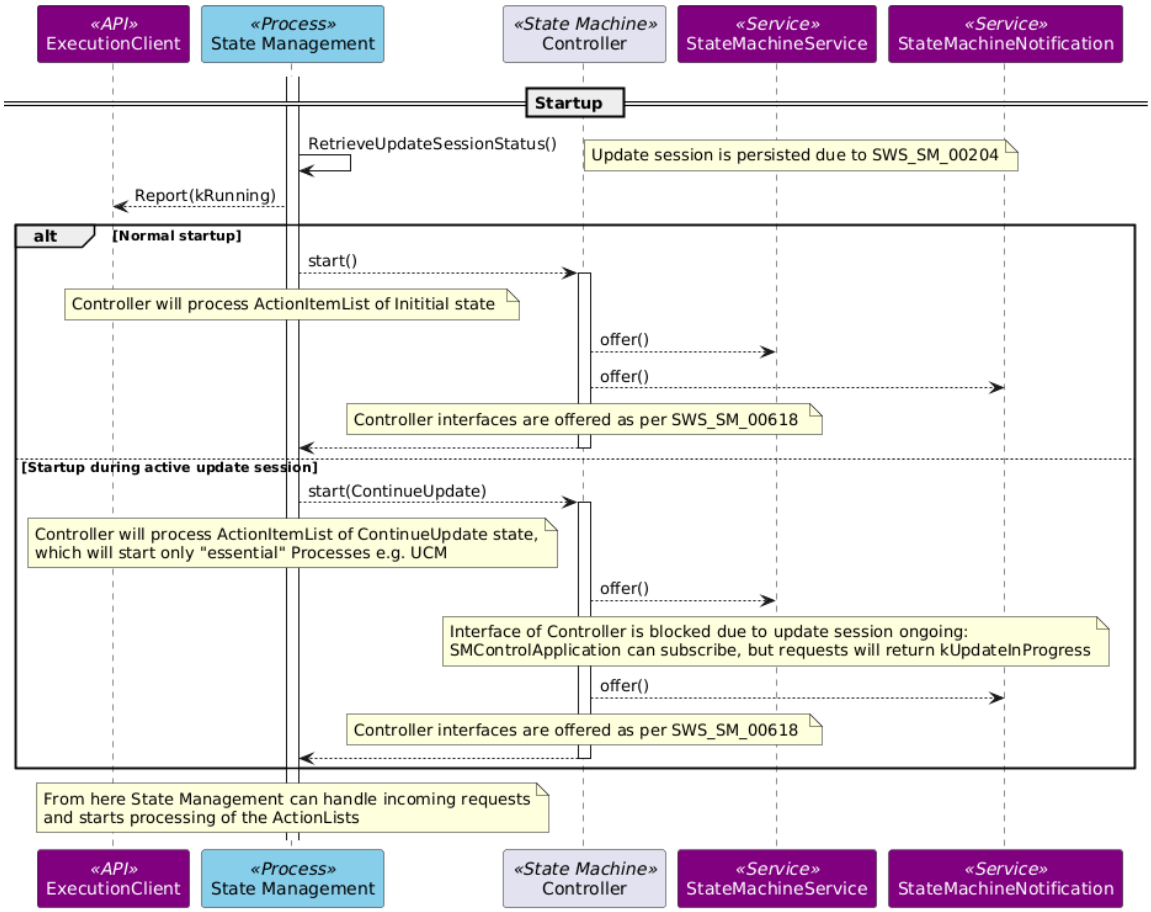

Figure 7.37: Start-up Sequence to Initial State or ContinueUpdate state

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.

7.7.15 StateMachine support for Suspend-to-RAM

The Suspend-to-RAM functionality transitions the Machine into well-defined states, where the Controller and all Agents have to move to StateMachine States configured to support suspend and wake-up scenarios. As mentioned in Chapter 7.5 the suspend and wake-up states of a Machine has to be carefully coordinated to ensure that only Adaptive Applications capable of handling suspend mode (S2R-

Aware applications) continue running before the OS enters the suspend state. Conversely, the opposite procedure is expected during OS wake-up, with Adaptive Applications returning to their normal operational mode.

In the StateMachine approach, the Controller is responsible for orchestrating state changes at the Machine level. To maintain consistent Suspend-to-RAM coordination and avoid conflicting or redundant requests, the StateMachine approach integrates the ara::sm::s2r::S2RHub functionality internally. Therefore, external applications such as the SMControlApplication are not allowed to access the ara::sm::s2r::S2RHub API when the StateMachine approach is used. This ensures centralized control and prevents race conditions or state inconsistencies during suspend and wake-up transitions.

[SWS_SM_00684] Mutual exclusivity of StateMachine approach and S2RHub API usage

Status: DRAFT
Upstream requirements: RS_SM_00004, RS_SM_00005, RS_SM_00402

⌈If the State Management follows the StateMachine approach, Adaptive Applications shall not use the C++ interface ara::sm::s2r::S2RHub. The Suspend-to-RAM coordination with Adaptive Applications and OS shall be exclusively managed by the StateMachine logic. Conversely, use of the ara::sm::s2r::S2RHub API is only permitted when the StateMachine approach is not used.⌋

Analogous to the shutdown and restart scenarios – where the SMControlApplication issues transition requests to StateMachine States configured in the Controller to perform a shutdown or restart of the Machine – an orchestrated transition into suspend mode requires that the Controller also has at least one StateMachine State properly configured.

As a logical consequence, only the Controller's StateMachine States configured with the ActionListItems EnterSuspendToRam and EnterSuspendToRamOS are permitted to initiate requests for S2R-Aware applications to enter Suspend-to-RAM mode, as well as to notify the operating system that the Suspend-to-RAM transition may begin. This coordination is achieved through the interaction between the ActionListItems EnterSuspendToRam and EnterSuspendToRamOS, along with the appropriate instantiation of ara::sm::s2r::S2RHub by State Management.
As a logical consequence, only the Controller's StateMachine States configured with the ActionListItems EnterSuspendToRam and EnterSuspendToRamOS are permitted to initiate requests for S2R-Aware applications to enter Suspend-to-RAM mode, as well as to notify the operating system that the Suspend-to-RAM transition may begin. This coordination is achieved through the interaction between the ActionListItems EnterSuspendToRam and EnterSuspendToRamOS, along with the appropriate instantiation of ara::sm::s2r::S2RHub by State Management.

[SWS_SM_00690] Instantiation of ara::sm::s2r::S2RHub

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈When EnterSuspendToRam, EnterSuspendToRamOS, or LeaveSuspendToRam ActionListItem exits, State Management shall create a instance of ara::sm::s2r::S2RHub.⌋

[SWS_SM_00689] ActionListItem - Trigger EnterSuspendState

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈When EnterSuspendToRam ActionListItem is triggered, State Management shall invoke ara::sm::s2r::S2RHub::RequestToEnterSuspendMode and use maxActionItemDuration as value for the argument timeout. The returned Result shall be used as the result for the ActionListItem.⌋

[SWS_SM_00692] ActionListItem - Trigger EnterSuspendState

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈When EnterSuspendToRamOs ActionListItem is triggered, State Management shall invoke ara::sm::s2r::S2RHub::EnterSuspendToRamOs.⌋

Multiple instances of the ActionListItem EnterSuspendToRamOs within the same Controller's Suspend-related StateMachine States are not foreseen. Additionally, it is expected that these StateMachine States contain only one ActionListItem of type EnterSuspendToRam. During ActionList processing, the first invocation transitions all S2R-Aware applications into Suspend-to-RAM mode, and the operating system is notified to initiate the suspend sequence. Repeating either ActionListItem within the same ActionList provides no functional benefit and may lead to redundant or conflicting behavior, as well as unnecessary complexity.

[SWS_SM_CONSTR_00034] Exclusive assignment of suspend-related ActionListItems in suspend-related states

Status: DRAFT

⌈Only the Controller's suspend-related StateMachine States shall be configured with at least one ActionListItem of type EnterSuspendToRam and exactly one ActionListItem of type EnterSuspendToRamOS.⌋

Coordinated exit from Suspend-to-RAM mode has also to be ensured. The integrator has to configure at least one Controller StateMachine State responsible for orchestrating the Machine's wake-up by restoring all required processes to their normal operational mode –either by starting them via Execution Management or by requesting them to exit their current suspend state. The SMControlApplication may initiate the transition request to the Controller to enter the wake-up state. The Controller then utilizes the ActionListItem LeaveSuspendToRam to execute this transition.

[SWS_SM_00691] ActionListItem - Trigger EnterSuspendState

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈When LeaveSuspendToRam `ActionListItem` is triggered, `State Management` shall invoke `ara::sm::s2r::S2RHub::RequestToLeaveSuspendMode`.⌋

[SWS_SM_CONSTR_00035] Prohibited assignment of LeaveSuspendToRam in Suspend state

Status: DRAFT

⌈The `Controller`'s suspend-related `StateMachine States` [SWS_SM_CONSTR_00034] shall not be configured with the `ActionListItem` LeaveSuspendToRam.⌋

The constraint [SWS_SM_CONSTR_00035] ensures, that the LeaveSuspendToRam `ActionListItem` is used exclusively for wake-up coordination and have to be assigned only to `Controller`'s `StateMachine States` responsible for exiting `Suspend-to-RAM` mode.

For `Controller StateMachine States` intended to handle wake-up transitions is not expected to configure more than one instance of the `ActionListItem` LeaveSuspendToRam. A single invocation is sufficient to coordinate the return of `S2R-Aware applications` from suspend mode to their normal operational state.

In a typical configuration, multiple `Controller StateMachine States` may be defined to handle wake-up scenarios following a `Suspend-to-RAM` transition (for handling update sessions see the special case 7.7.15.1. Each of these states is expected to include the `ActionListItem` LeaveSuspendToRam to ensure proper coordination with `S2R-Aware application`. However, transitions to these wake-up states may also occur for reasons unrelated to a prior `Suspend-to-RAM` request –for example, due to error recovery or manual intervention. In such cases, execution of the LeaveSuspendToRam `ActionListItem` shall proceed without adverse effects. The `ActionList` processing has to continue seamlessly, regardless of whether a `Suspend-to-RAM` was previously initiated.

[SWS_SM_00680] Robust execution of LeaveSuspendToRam ActionItem

Status: DRAFT
Upstream requirements: RS_SM_00004, RS_SM_00005, RS_SM_00402

⌈If a `Controller StateMachine State` includes the `ActionListItem` LeaveSuspendToRam, it shall be executed unconditionally during `ActionList` processing. If the `Controller` is leaving a suspend state, the result of the `ActionListItem` shall reflect the actual execution outcome. If no prior `Suspend-to-RAM` transition occurred, the `ActionListItem` shall complete without error and shall not cause the `ActionList` processing to fail.⌋

Example 7.1

The following diagrams provide an example how the `Suspend-to-RAM` can be supported by the `StateMachine` approach:

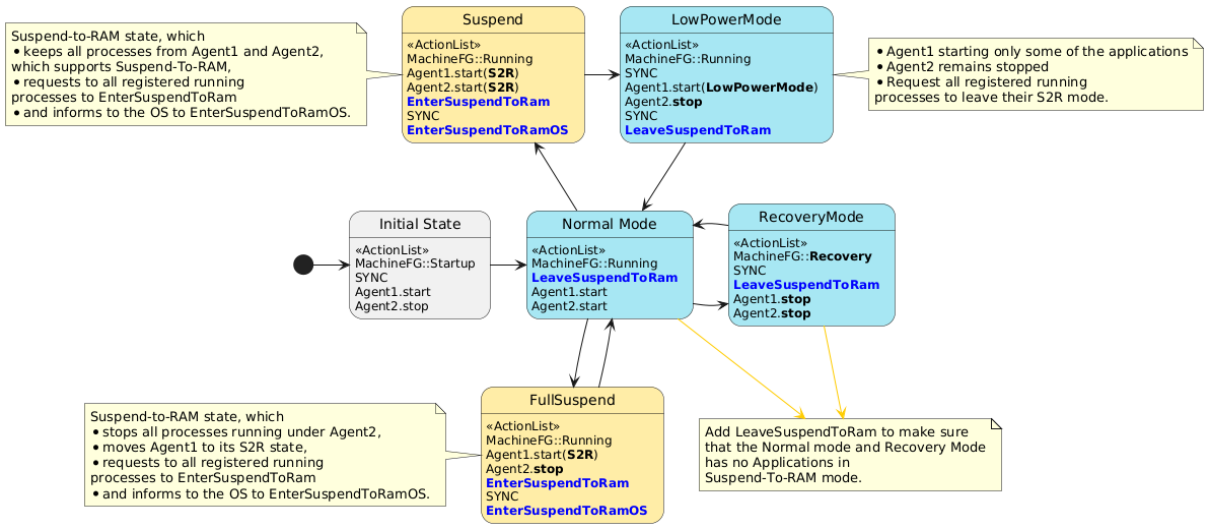

Figure 7.38: Example for the Controller StateMachineStates supporting Suspend-To-RAM.

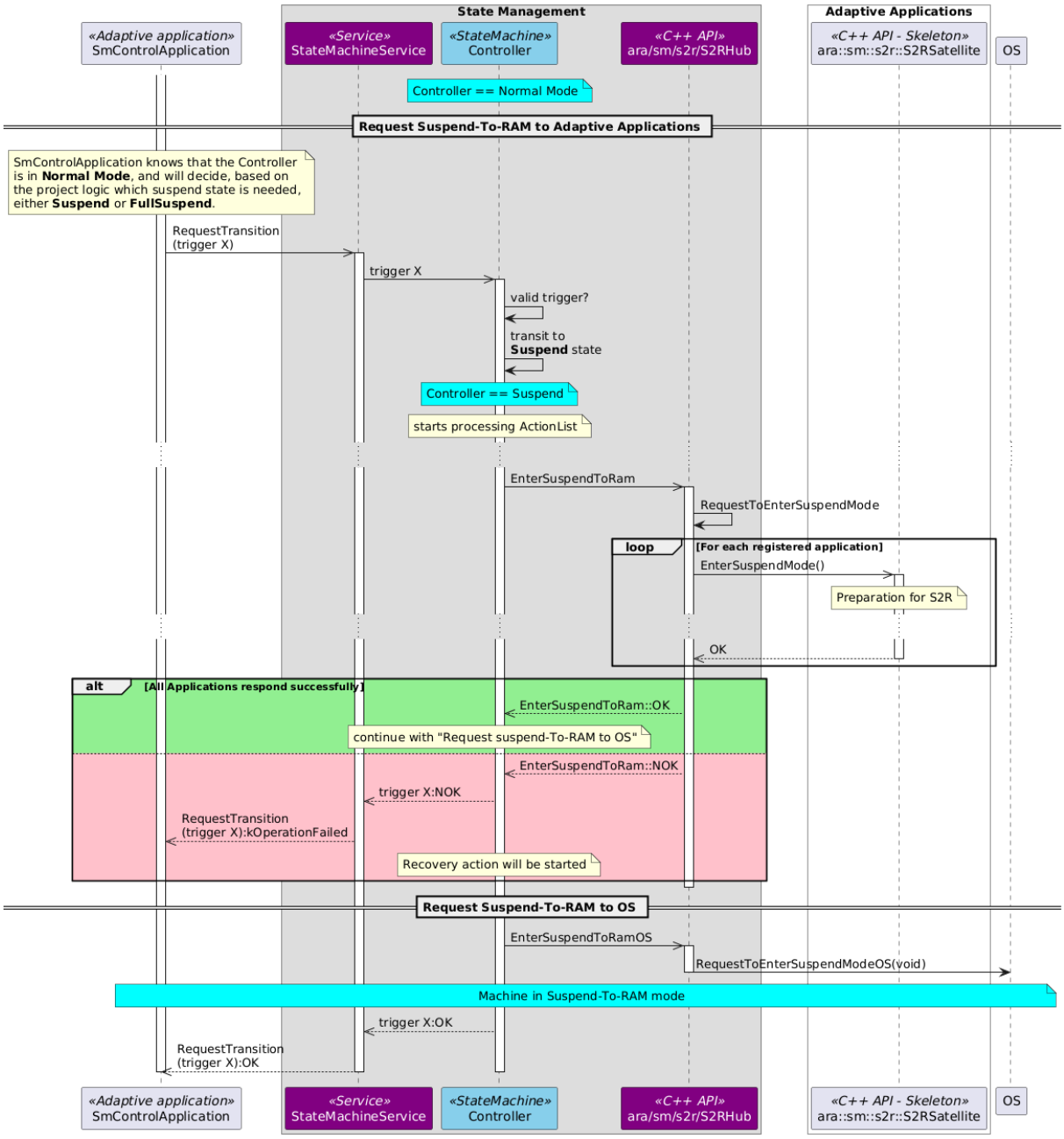

Figure 7.39: Example for entering Suspend-To-RAM mode.

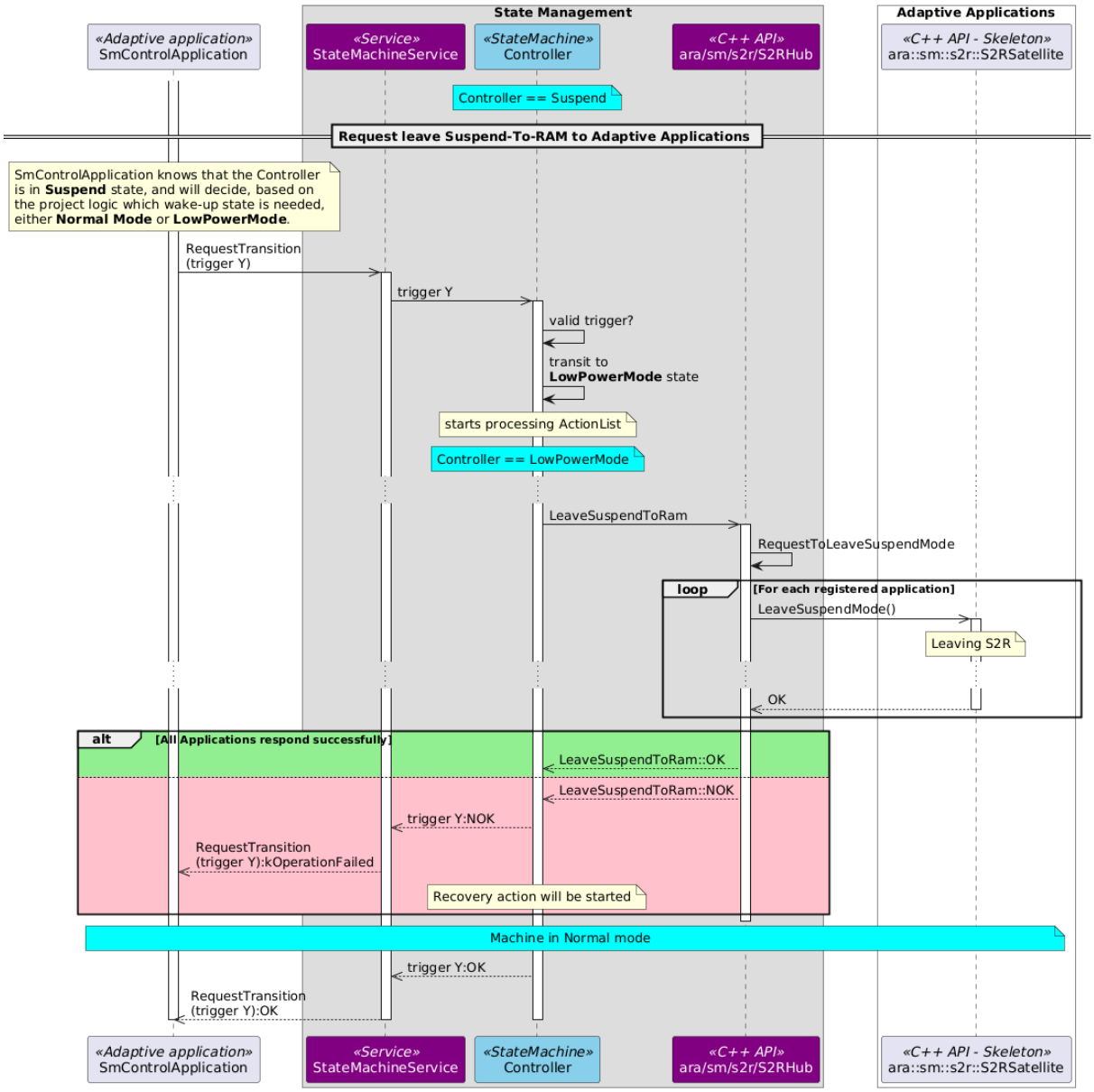

Figure 7.40: Example for leaving Suspend-To-RAM

7.7.15.1 Autonomous wake-up from suspend state

`State Management` is capable of reacting to wake-up triggers originating from multiple sources. While the `SMControlApplication` may explicitly request a transition out of `Suspend-to-RAM` mode, `State Management` can also initiate a wake-up autonomously in response to system-level or an update session request.

Specifically, if the operating system signals a wake-up via the `apext::sm::WakeUpHandler` interface (if implemented), or if the `Update and Configuration Management` issues a `RequestUpdateSession` call while the `Controller` is in a suspend-related state, `State Management` will initiate a wake-up sequence.

In these cases, State Management performs the transition by moving the Controller from its current suspend-related StateMachine State to a designated wake-up StateMachine State. This autonomous behavior ensures that the Machine can resume operation without requiring explicit coordination from the SMControlApplication. To enable this autonomous transition, a dedicated wake-up StateMachine State has to be modeled and identifiable by State Management.

[SWS_SM_CONSTR_00036] Existence of StateMachine State LeaveSuspend for StateMachine of type Controller

Status: DRAFT

⌈The configured StateMachine of type Controller shall have corresponding LeaveSuspend StateMachine State configured, at the time when the creation of the manifest is finished.⌋

[SWS_SM_CONSTR_00037] Mandatory ActionListItem in LeaveSuspend state

Status: DRAFT

⌈The Controller's StateMachine State LeaveSuspend shall include the ActionListItem LeaveSuspendToRam in its ActionList.⌋

[SWS_SM_00683] Autonomous wake-up due to OS system call

Status: DRAFT
Upstream requirements: RS_SM_00004, RS_SM_00005, RS_SM_00402

⌈When the operating system notifies system wake-up via the apext::sm::WakeUpHandler, State Management shall transition the Controller to its StateMachine State LeaveSuspend [SWS_SM_CONSTR_00036].⌋

[SWS_SM_00681] Autonomous wake-up due to update session request

Status: DRAFT
Upstream requirements: RS_SM_00004, RS_SM_00005, RS_SM_00402

⌈When Update and Configuration Management issues RequestUpdateSession, UpdateAllowed is kUpdateAllowed [SWS_SM_00631] and the Controller is in a suspend-related StateMachine State [SWS_SM_CONSTR_00034], State Management shall transition the Controller to its StateMachine State LeaveSuspend [SWS_SM_CONSTR_00036].⌋

Once the Controller has transitioned out of suspend mode and the LeaveSuspend state has successfully completed its ActionList processing, State Management can proceed with the RequestUpdateSession call. If the ActionList processing fails, the request shall be rejected with a return value of kOperationRejected. This guarantees that the system is in a fully operational state before granting update access.

[SWS_SM_00682] Conditional approval of RequestUpdateSession after autonomous wake-up

Status: DRAFT
Upstream requirements: RS_SM_00004, RS_SM_00005, RS_SM_00402

⌈Procesing the `RequestUpdateSession` request shall only continue if the `Controller`'s `StateMachine State` LeaveSuspend has successfully completed its `ActionList` processing. If the `ActionList` fails or is interrupted, the request shall return `kOperationRejected`.⌋

Example 7.2

The following diagrams show an example, how a project may configure the `Controller` to cover autonomous wake-up from the `Suspend-to-RAM` mode.

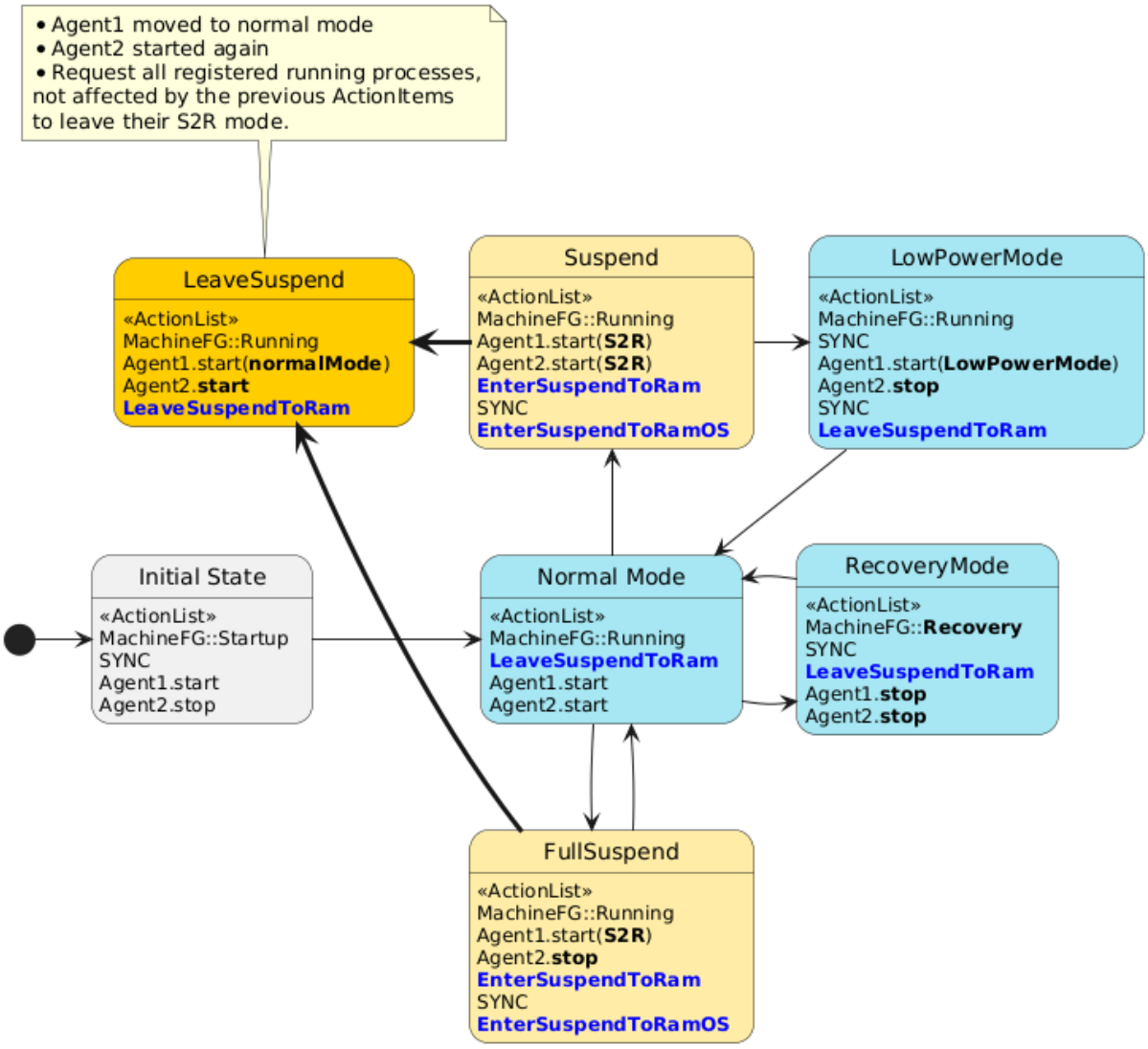

Figure 7.41: Example for the Controller configuration supporting autonomous wake-up via the LeaveSuspend StateMachine State

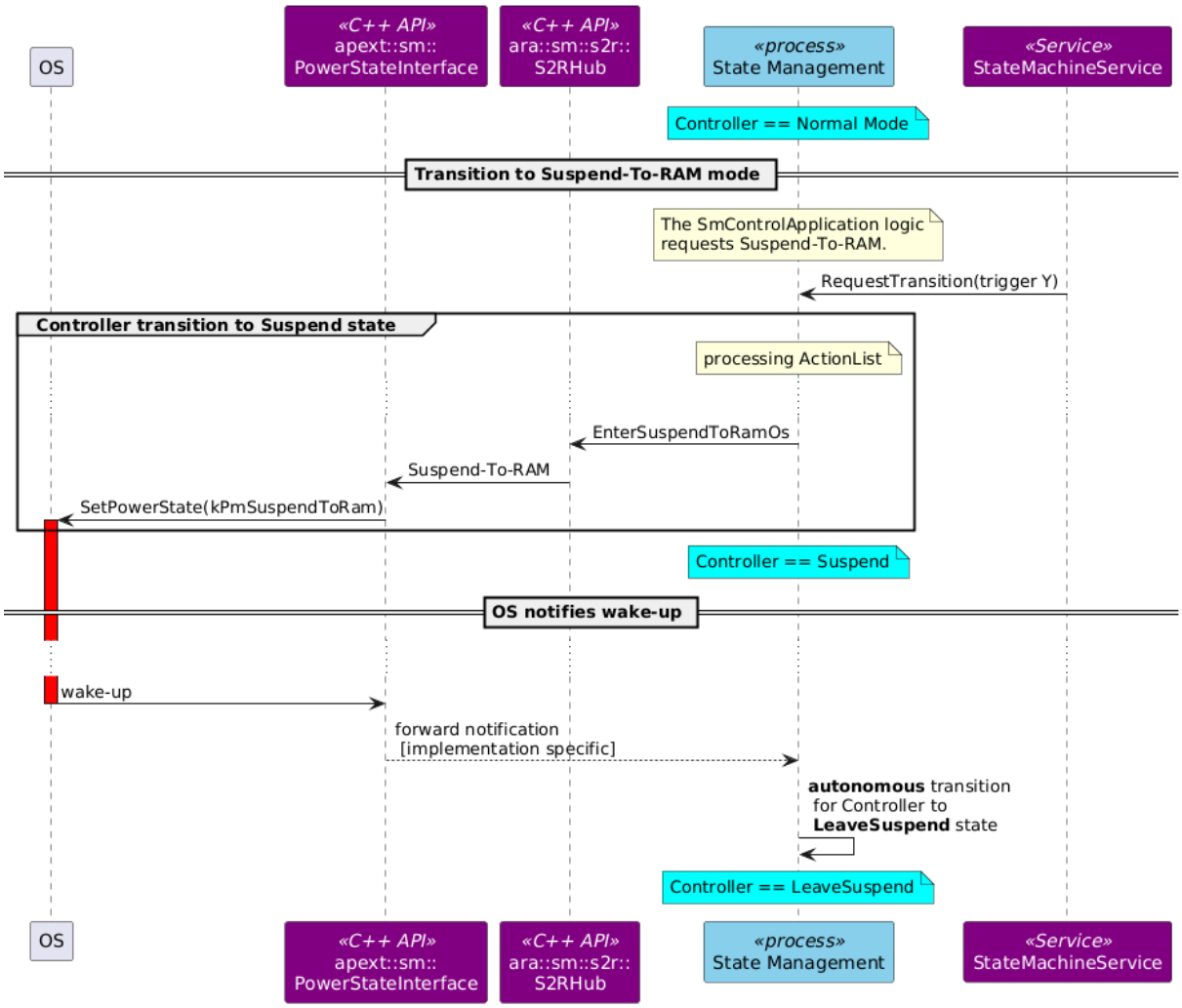

Figure 7.42: Example for autonomous wake-up due to OS system call

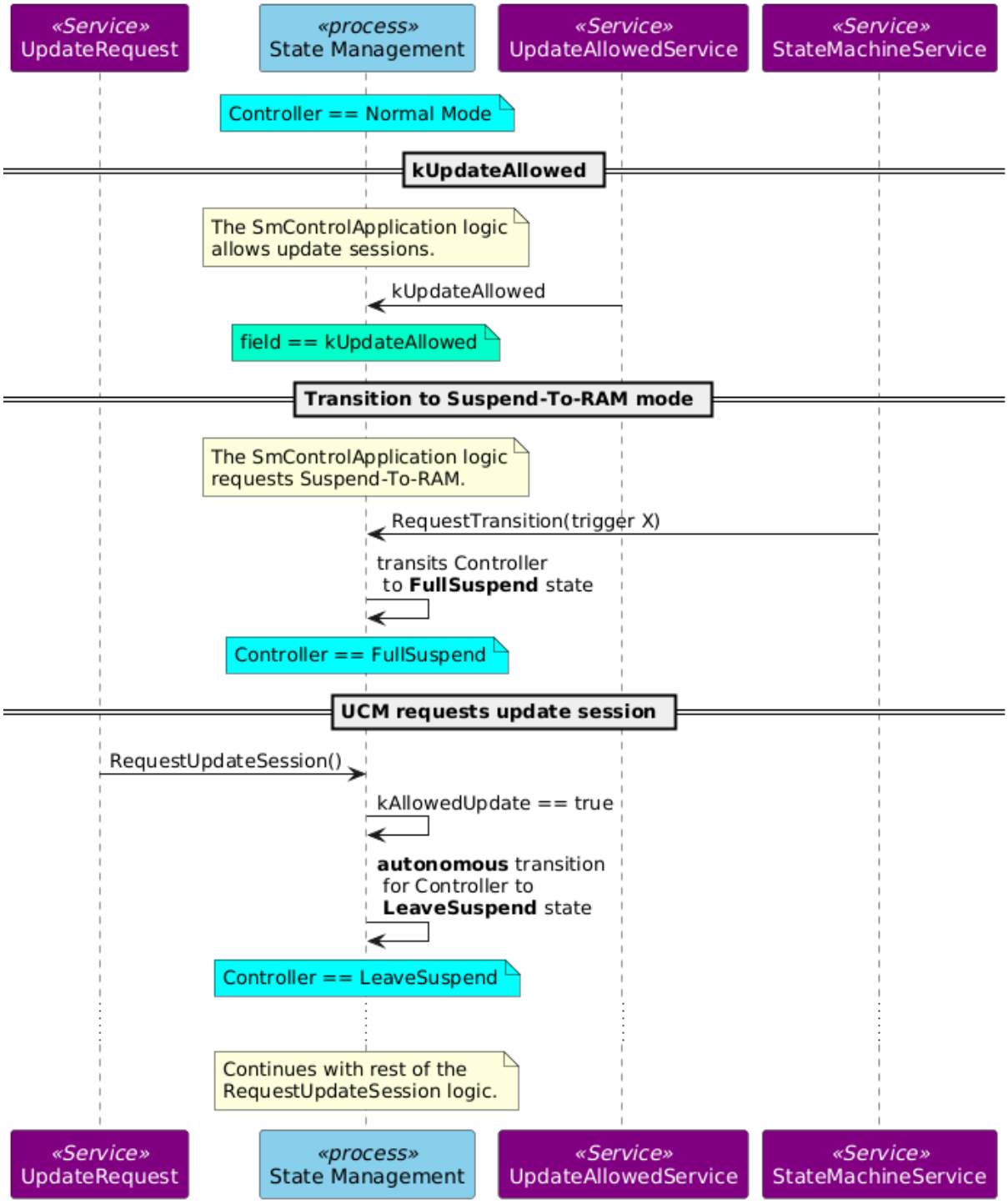

Figure 7.43: Example for autonomous wake-up due to update session request

7.8 Functional cluster life-cycle

7.8.1 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.

7.8.2 Shutdown

As mentioned in Section 7.8.1 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.

7.8.3 Restart

As mentioned in Section 7.8.1 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 7.8.2. Only difference to shutdown is, that the Machine is being restarted instead of being just shutdown.

7.8.4 Suspended

High-performance computing platforms based on microprocessors have complex startup processes that consist of many steps. However, customers expect near-instant responsiveness from systems like rear camera upon entering the car. Additionally, machines should use only necessary power and activate power-saving modes when appropriate. To address these challenges, it is essential to consider optimized use-cases such as Suspend-to-RAM and warm booting a machine, in addition to the standard cold boot process.

The State Management supports a coordinated entering into a Suspend-to-RAM (S2R) of the underlying operating system (if supported), where the machines state is saved to RAM, CPU, and peripheral devices powered down. The coordination is different for Adaptive Applications not supporting S2R, Adaptive Applications tolerating S2R, and Adaptive Applications which are S2R aware.

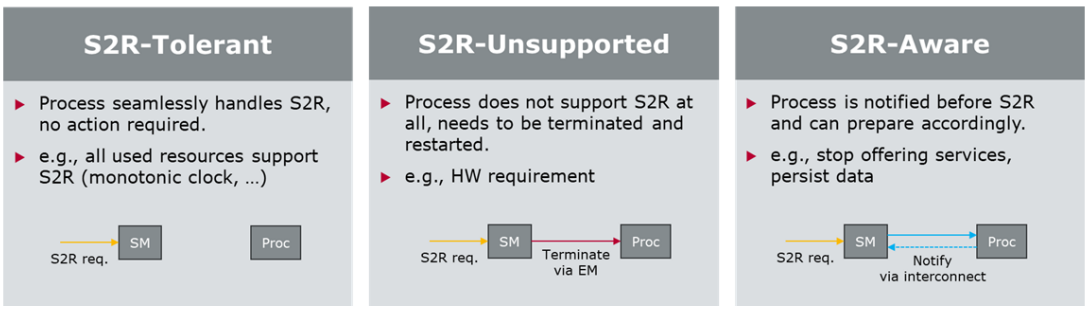

Figure 7.44: Different types of S2R support in Adaptive Applications

7.8.4.1 Suspend-to-RAM tolerant

If a Executable.suspendToRamAwareness is set to suspendToRamTolerant, the application is able to handle and survive a Suspend-to-RAM. From the State Management's point of view, there are no actions that need to be taken before the Suspend-to-RAM.

7.8.4.2 Suspend-to-RAM not supported

If a Executable.suspendToRamAwareness is set to suspendToRamNotSupported, the application is NOT able to handle or survive a Suspend-to-RAM. Therefore, from the State Management's point of view, appropriate countermeasures have to be taken by the integrator. The most common countermeasure is probably to terminate the corresponding application by State Management requesting a Function Group State transition to Execution Management, where the process is terminated.

7.8.4.3 Suspend-to-RAM aware

If a Executable.suspendToRamAwareness is set to suspendToRamAware, the application signaled that it needs a pre-synchronization to handle and survive a Suspend-to-RAM. From the State Management's point of view, therefore, appropriate measures have to be taken. The State Management offers the S2R hub and the S2R satellite for this purpose. The corresponding applications have to register as S2R satellite and are then informed before a Suspend-to-RAM is entered and after the wake-up again that they can resume their 'normal' operation. The SMControlApplication could either use the StateMachine approach (see section 7.7.15) or the C++ Interface ara::sm::s2r::S2RHub (see section 7.5.1). These options are mutually exclusive. The usage of ara::sm::s2r::S2RHub by the SMControlApplication is only applicable when State Management is not based on the StateMachine approach (see [SWS_SM_00684]).

7.8.5 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.

7.9 Reporting

7.9.1 Security Events

This section lists all security events defined by this functional cluster.

[SWS_SM_70000] Security events for State Management

Status: DRAFT
Upstream requirements: RS_Ids_00810

⌈

Name	*Description*	*ID*
SEV_ACCESS_CONTROL_SM_IAM_ACCESS_DENIED	Access of an application to a resource provided by State Management was denied.	137

⌋

[SWS_SM_70001] Security event context data definition: SEV_ACCESS_CONTROL_SM_IAM_ACCESS_DENIED

Status: DRAFT
Upstream requirements: RS_Ids_00810

⌈

SEV Name	*SEV_ACCESS_CONTROL_SM_IAM_ACCESS_DENIED*	
ID	137	
Description	Access of an application to a resource provided by State Management was denied.	
Context Data Version	1	
Context Data	*Data Type*	*Allowed Values*
UserId	uint32	

⌋

7.9.2 Log Messages

This functional cluster does not define any non-verbose log messages (i.e., modelled DLT messages).

7.9.3 Violation Messages

This section lists all violation messages (i.e., DLT messages logged for Violations according to [SWS_CORE_00021]) defined by this functional cluster.
Please note that concrete implementations might additionally implement Non-Standardized Violations (see also [SWS_CORE_00003]).

[SWS_CORE_13000]

Dlt-Message	InsufficientPermissionsViolation
Description	Sent in case the caller had insufficient permissions for the requested operation.
MessageId	0x80001fff
MessageType Info	DLT_LOG_FATAL

▽

△

Dlt-Argument	*ArgumentDescription*	*ArgumentType*	*ArgumentUnit*
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
message	Additional message that describes the cause of the access violation.	uint8 [encoding UTF-8]	

[SWS_CORE_13003]

Dlt-Message	InstanceSpecifierMappingIntegrityViolation		
Description	InstanceSpecifier either cannot be resolved in the model in the context of your executable, or it refers to a model element other than a PortPrototype.		
MessageId	0x80001ffc		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	*ArgumentDescription*	*ArgumentType*	*ArgumentUnit*
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	

[SWS_CORE_13004]

Dlt-Message	PortInterfaceMappingViolation		
Description	The type of mapping does not match the expected type of PortInterface: {portInterfaceTypeName} referenced by a {mappingTypeName}.		
MessageId	0x80001ffb		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	*ArgumentDescription*	*ArgumentType*	*ArgumentUnit*
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	

[SWS_CORE_13005]

Dlt-Message	ProcessMappingViolation		
Description	Matching InstanceRef exists, but no matching (modelled) Process found that matches the (runtime) process.		
MessageId	0x80001ffa		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	

[SWS_CORE_13006]

Dlt-Message	InstanceSpecifierAlreadyInUseViolation		
Description	Violation message that is sent in case a constructor in the ara framework was called with an Instance Specifier already in use in this process.		
MessageId	0x80001ff9		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	
location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object.	uint8 [encoding UTF-8]	
className	Name of the class that was instantiated.	uint8 [encoding UTF-8]	

[SWS_CORE_13007]

Dlt-Message	AraNotInitializedViolation		
Description	Violation message that is sent in case a constructor or function checks for an initialized ara and identifies that ara is not initialized.		
MessageId	0x80001ff8		
MessageType Info	DLT_LOG_FATAL		
Dlt-Argument	***ArgumentDescription***	***ArgumentType***	***ArgumentUnit***
modeledProcessId	Meta-model identifier of the process that caused the violation, i.e., its short name path with '/' as a separator.	uint8 [encoding UTF-8]	

▽

△

location	An implementation-defined identifier of the location where the violation was detected, for example {filename}:{linenumber}.	uint8 [encoding UTF-8]	
instanceSpecifier	InstanceSpecifier used to try to create the object. (only present in case this violation is created by a function/constructor that has an InstanceSpecifier as an argument)	uint8 [encoding UTF-8]	
functionName	Name of the function/constructor that was called that checks for an initialized ara.	uint8 [encoding UTF-8]	

7.9.4 Production Errors

This functional cluster does not define any production errors (i.e., Diagnostic Events).

8 API specification

This chapter provides a reference of the APIs defined by this functional cluster. The API is described in the following chapters in tables. Table 8.1 explains the content that is described in such an API table.

Kind:	Defines the kind of the declaration that this API table describes. The following values are supported: • class (Declaration of a class) • function (Declaration of a member or non-member function) • struct (Declaration of a structure) • type alias (Declaration of a type alias) • enumeration (Declaration of an enumeration) • variable (Declaration of a variable)	
Port Interfaces:	States that the C++ API `class` is the related C++ API binding for the given modeled sub-class of `PortInterface`	
Header File:	Defines the header file to be included according to [SWS_CORE_90001]	
Forwarding Header File:	Defines the forwarding header file to be included according to [SWS_CORE_90001]	
Scope:	Defines the scope that may be a C++ `namespace` (in case of a `class` or non-member function) or a `class` declaration (in case of a member)	
Symbol:	C++ symbol name	
Thread Safety:	Defines whether a function is thread-safe, not thread-safe, or conditional according to [SWS_CORE_13200] and [SWS_CORE_13202]	
Syntax:	Description of C++ syntax	
Template Param:	Template parameter (0..*)	Template parameter(s) used to parameterize the template
Parameters (in):	Parameter declaration (0..*)	Parameter(s) that are passed to the function
Parameters (out):	Parameter declaration (0..*)	Parameter(s) that are returned to the caller
Return Value:	Return type	Type of the value that the function returns
Exception Safety:	Defines whether a function is exception-safe, not exception safe or conditionally exception safe	
Exceptions:	List of C++ `Exception`s that may be thrown by the function	
Violations:	List of violations that may raised by the function	
Errors:	Error type (0..*)	List of defined `ara::core::ErrorCode`s that may be returned by the function with their recoverability class defined in [RS_AP_00160]. APIs can be extended with vendor-specific error codes. These are not standardized by AUTOSAR
Description:	Brief description of the function	

Table 8.1: Explanation of an API table

8.1 Header: ara/sm/sm_error_domain.h

8.1.1 Non-Member Types

8.1.1.1 Enumeration: SmErrc

[SWS_SM_81240] Definition of API enum ara::sm::SmErrc

Status: DRAFT
Upstream requirements: RS_AP_00119

⌈

Kind:	enumeration	
Header file:	#include "ara/sm/sm_error_domain.h"	
Forwarding header file:	#include "ara/sm/sm_fwd.h"	
Scope:	namespace ara::sm	
Symbol:	SmErrc	
Underlying type:	ara::core::ErrorDomain::CodeType	
Syntax:	`enum class SmErrc : ara::core::ErrorDomain::CodeType {...};`	
Values:	kOfferFailed	= 2
		Service could not be offered due to failure of communication with Sm daemon
	kCommunicationFailed	= 3
		Communication to satellites failed on lower layers.
	kCommunicationTimeout	= 4
		Timeout of communication with satellites.
	kAtLeastOneRejected	= 5
		At least one Satellite instance had issues to enter `Suspend Mode`.
	kAtLeastOneHadIssues ToLeave	= 6
		At least one Satellite instance had issues to leave `Suspend Mode`.
	kRejected	= 7
		Issue to enter or leave the `Suspend Mode`.
	kAuthenticationRequired	= 8
		`S2R Satellite` requires IAM authentication. `ara::sm::s2r::S2RSatellite::S2RSatellite` has to be used instead.
Description:	Defines an enumeration class for the State Management error codes.	

⌋

8.1.2 Non-Member Functions

8.1.2.1 Other

8.1.2.1.1 GetSmDomain

[SWS_SM_81251] Definition of API function ara::sm::GetSmDomain

Status: DRAFT
Upstream requirements: RS_AP_00119, RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/sm/sm_error_domain.h"	
Scope:	namespace ara::sm	
Syntax:	`constexpr const ara::core::ErrorDomain & GetSmDomain () noexcept;`	
Return value:	const ara::core::Error Domain &	The global SmErrorDomain object.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Returns the global SmErrorDomain object.	

⌋

8.1.2.1.2 MakeErrorCode

[SWS_SM_81244] Definition of API function ara::sm::MakeErrorCode

Status: DRAFT
Upstream requirements: RS_AP_00119, RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/sm/sm_error_domain.h"	
Scope:	namespace ara::sm	
Syntax:	`constexpr ara::core::ErrorCode MakeErrorCode (SmErrc code, ara::core::ErrorDomain::SupportDataType data) noexcept;`	
Parameters (in):	code	Error code number.
	data	Vendor defined data associated with the error.
Return value:	ara::core::ErrorCode	An ErrorCode object.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Creates an error code.	

⌋

8.1.3 Class: SmErrorDomain

[SWS_SM_81241] Definition of API class ara::sm::SmErrorDomain

Status: DRAFT
Upstream requirements: RS_AP_00119

⌈

Kind:	class
Header file:	#include "ara/sm/sm_error_domain.h"
Forwarding header file:	#include "ara/sm/sm_fwd.h"
Scope:	namespace ara::sm
Symbol:	SmErrorDomain
Base class:	ara::core::ErrorDomain
Syntax:	`class SmErrorDomain final : public ara::core::ErrorDomain {...};`
Unique ID:	As per `ara::sm::SmErrorDomain` in [SWS_CORE_90023]
Description:	Defines the error domain for State Management.

⌋

8.1.3.1 Public Member Types

8.1.3.1.1 Type Alias: Errc

[SWS_SM_81245] Definition of API type ara::sm::SmErrorDomain::Errc

Status: DRAFT
Upstream requirements: RS_AP_00119

⌈

Kind:	type alias
Header file:	#include "ara/sm/sm_error_domain.h"
Scope:	`class ara::sm::SmErrorDomain`
Symbol:	Errc
Syntax:	`using Errc = SmErrc;`
Description:	Alias for the error code value enumeration.

⌋

8.1.3.1.2 Type Alias: Exception

[SWS_SM_81246] Definition of API type ara::sm::SmErrorDomain::Exception

Status: DRAFT
Upstream requirements: RS_AP_00119

⌈

Kind:	type alias
Header file:	#include "ara/sm/sm_error_domain.h"
Scope:	`class ara::sm::SmErrorDomain`
Symbol:	Exception
Syntax:	`using Exception = SmException;`
Description:	Alias for the exception base class.

⌋

8.1.3.2 Public Member Functions

8.1.3.2.1 Special Member Functions

8.1.3.2.1.1 Default Constructor

[SWS_SM_81247] Definition of API function ara::sm::SmErrorDomain::SmErrorDomain

Status: DRAFT
Upstream requirements: RS_AP_00119, RS_AP_00159

⌈

Kind:	function
Header file:	#include "ara/sm/sm_error_domain.h"
Scope:	`class ara::sm::SmErrorDomain`
Syntax:	`SmErrorDomain () noexcept;`
Exception Safety:	exception safe
Thread Safety:	thread-safe
Description:	Creates a SmErrorDomain instance.

⌋

8.1.3.2.2 Member Functions

8.1.3.2.2.1 Message

[SWS_SM_81249] Definition of API function ara::sm::SmErrorDomain::Message

Status: DRAFT
Upstream requirements: RS_AP_00119, RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/sm/sm_error_domain.h"	
Scope:	`class ara::sm::SmErrorDomain`	
Syntax:	`const char * Message (CodeType errorCode) const noexcept override;`	
Parameters (in):	errorCode	The error code number.
Return value:	const char *	The message associated with the error code.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Returns the message associated with the error code.	

⌋

8.1.3.2.2.2 Name

[SWS_SM_81248] Definition of API function ara::sm::SmErrorDomain::Name

Status: DRAFT
Upstream requirements: RS_AP_00119, RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/sm/sm_error_domain.h"	
Scope:	`class ara::sm::SmErrorDomain`	
Syntax:	`const char * Name () const noexcept override;`	
Return value:	const char *	"Sm".
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Returns the name of the error domain.	

⌋

8.1.3.2.2.3 ThrowAsException

[SWS_SM_81250] Definition of API function ara::sm::SmErrorDomain::ThrowAsException

Status: DRAFT
Upstream requirements: RS_AP_00119

⌈

Kind:	function	
Header file:	#include "ara/sm/sm_error_domain.h"	
Scope:	`class ara::sm::SmErrorDomain`	
Syntax:	`void ThrowAsException (const ara::core::ErrorCode &errorCode) const override;`	
Parameters (in):	errorCode	The error to throw.
Return value:	None	
Exception Safety:	not exception safe	
Thread Safety:	thread-safe	
Description:	Throws the exception associated with the error code. As per [SWS_CORE_10304], this function does not participate in overload resolution when C++ exceptions are disabled in the compiler toolchain.	

⌋

8.1.4 Class: SmException

[SWS_SM_81242] Definition of API class ara::sm::SmException

Status: DRAFT
Upstream requirements: RS_AP_00119

⌈

Kind:	class
Header file:	#include "ara/sm/sm_error_domain.h"
Forwarding header file:	#include "ara/sm/sm_fwd.h"
Scope:	namespace ara::sm
Symbol:	SmException
Base class:	ara::core::Exception
Syntax:	`class SmException : public ara::core::Exception {...};`
Description:	Exception type thrown by State Management.

⌋

8.1.4.1 Public Member Functions

8.1.4.1.1 Constructors

8.1.4.1.1.1 SmException

[SWS_SM_81243] Definition of API function ara::sm::SmException::SmException

Status: DRAFT
Upstream requirements: RS_AP_00119, RS_AP_00159

⌈

Kind:	function	
Header file:	#include "ara/sm/sm_error_domain.h"	
Scope:	`class ara::sm::SmException`	
Syntax:	`explicit SmException (ara::core::ErrorCode errorCode) noexcept;`	
Parameters (in):	errorCode	The error code.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Construct a new StateManagement exception object containing an error code.	

⌋

8.2 Header: ara/sm/s2r/S2RHub.h

8.2.1 Class: S2RHub

[SWS_SM_81102] Definition of API class ara::sm::s2r::S2RHub

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	class
Header file:	#include "ara/sm/s2r/S2RHub.h"
Forwarding header file:	#include "ara/sm/sm_fwd.h"
Scope:	namespace ara::sm::s2r
Symbol:	S2RHub
Syntax:	`class S2RHub {...};`
Description:	S2RHub class.

⌋

8.2.1.1 Public Member Functions

8.2.1.1.1 Special Member Functions

8.2.1.1.1.1 Move Constructor

[SWS_SM_81106] Definition of API function ara::sm::s2r::S2RHub::S2RHub

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function	
Header file:	#include "ara/sm/s2r/S2RHub.h"	
Scope:	`class ara::sm::s2r::S2RHub`	
Syntax:	`S2RHub (S2RHub &&ra) noexcept;`	
Parameters (in):	ra	The S2RHub object to be moved.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Move constructor for S2RHub.	

⌋

8.2.1.1.1.2 Copy Constructor

[SWS_SM_81107] Definition of API function ara::sm::s2r::S2RHub::S2RHub

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function
Header file:	#include "ara/sm/s2r/S2RHub.h"
Scope:	`class ara::sm::s2r::S2RHub`
Syntax:	`S2RHub (const S2RHub &)=delete;`
Description:	The copy constructor for S2RHub shall not be used.

⌋

8.2.1.1.1.3 Copy Assignment Operator

[SWS_SM_81109] Definition of API function ara::sm::s2r::S2RHub::operator=

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function
Header file:	#include "ara/sm/s2r/S2RHub.h"
Scope:	`class ara::sm::s2r::S2RHub`
Syntax:	`S2RHub & operator= (const S2RHub &)=delete;`
Description:	The copy assignment operator for S2RHub shall not be used.

⌋

8.2.1.1.1.4 Move Assignment Operator

[SWS_SM_81108] Definition of API function ara::sm::s2r::S2RHub::operator=

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function	
Header file:	#include "ara/sm/s2r/S2RHub.h"	
Scope:	`class ara::sm::s2r::S2RHub`	
Syntax:	`S2RHub & operator= (S2RHub &&ra) noexcept;`	
Parameters (in):	ra	The S2RHub object to be moved.
Return value:	S2RHub &	The moved S2RHub object.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Move assignment operator for S2RHub.	

⌋

8.2.1.1.1.5 Destructor

[SWS_SM_81105] Definition of API function ara::sm::s2r::S2RHub::~S2RHub

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function
Header file:	#include "ara/sm/s2r/S2RHub.h"
Scope:	`class ara::sm::s2r::S2RHub`

▽

△

Syntax:	`virtual ~S2RHub () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor for S2RHub.

⌋

8.2.1.1.2 Constructors

8.2.1.1.2.1 S2RHub

[SWS_SM_81103] Definition of API function ara::sm::s2r::S2RHub::S2RHub

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function	
Header file:	#include "ara/sm/s2r/S2RHub.h"	
Scope:	`class ara::sm::s2r::S2RHub`	
Syntax:	`explicit S2RHub (const ara::core::InstanceSpecifier &instance) noexcept;`	
Parameters (in):	instance	instance specifier to the PPortPrototype of a SM...
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`InsufficientPermissionsViolation`	Sent in case the caller had insufficient permissions for the requested operation.
	`InstanceSpecifierMappingIntegrityViolation`	InstanceSpecifier either cannot be resolved in the model in the context of your executable, or it refers to a model element other than a PortPrototype.
	`PortInterfaceMappingViolation`	The type of mapping does not match the expected type of Port Interface: {portInterfaceTypeName} referenced by a {mappingType Name}.
	`ProcessMappingViolation`	Matching InstanceRef exists, but no matching (modelled) Process found that matches the (runtime) process.
	`InstanceSpecifierAlreadyInUseViolation`	Violation message that is sent in case a constructor in the ara framework was called with an InstanceSpecifier already in use in this process.
	`AraNotInitializedViolation`	Violation message that is sent in case a constructor or function checks for an initialized ara and identifies that ara is not initialized.
Description:	Creation of an S2RHub.	

⌋

8.2.1.1.3 Member Functions

8.2.1.1.3.1 EnterSuspendToRamOs

[SWS_SM_81112] Definition of API function ara::sm::s2r::S2RHub::EnterSuspendToRamOs

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function
Header file:	#include "ara/sm/s2r/S2RHub.h"
Scope:	`class ara::sm::s2r::S2RHub`
Syntax:	`void EnterSuspendToRamOs () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not-threadsafe
Description:	Function to trigger OS to enter Suspend To RAM.

⌋

8.2.1.1.3.2 RequestToEnterSuspendMode

[SWS_SM_81110] Definition of API function ara::sm::s2r::S2RHub::RequestToEnterSuspendMode

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function	
Header file:	#include "ara/sm/s2r/S2RHub.h"	
Scope:	`class ara::sm::s2r::S2RHub`	
Syntax:	`ara::core::Future< void > RequestToEnterSuspendMode (int timeout=1000) noexcept;`	
Parameters (in):	timeout	Timeout value in ms until the `S2R Satellite` have to respond.
Return value:	ara::core::Future< void >	HubResult if it is successful
Exception Safety:	exception safe	
Thread Safety:	not-threadsafe	
Errors:	SmErrc::k CommunicationFailed	rollback_semantics
		Communication to satellites failed on lower layers.
	SmErrc::k CommunicationTimeout	rollback_semantics
		Timeout of communication with satellites.
	SmErrc::kAtLeastOne Rejected	no_rollback_semantics
		Some S2RSatellites entered suspend state, but not all.
Description:	Function to Request all `S2R Satellite` to enter suspend state.	

⌋

8.2.1.1.3.3 RequestToLeaveSuspendMode

[SWS_SM_81111] Definition of API function ara::sm::s2r::S2RHub::RequestToLeaveSuspendMode

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function	
Header file:	#include "ara/sm/s2r/S2RHub.h"	
Scope:	`class ara::sm::s2r::S2RHub`	
Syntax:	`ara::core::Future< void > RequestToLeaveSuspendMode (int timeout=1000) noexcept;`	
DIRECTION NOT DEFINED	timeout	--
Return value:	ara::core::Future< void >	HubResult if it is successful
Exception Safety:	exception safe	
Thread Safety:	not-threadsafe	
Errors:	SmErrc::kCommunicationFailed	rollback_semantics
		Communication to satellites failed on lower layers.
	SmErrc::kCommunicationTimeout	rollback_semantics
		Timeout of communication with satellites.
	SmErrc::kAtLeastOneHadIssuesToLeave	no_rollback_semantics
		Some S2RSatellites left suspend state, but not all.
Description:	Function to request all registered `S2R Satellite` to leave suspend state.	

⌋

8.3 Header: ara/sm/s2r/S2RSatellite.h

8.3.1 Class: S2RSatellite

[SWS_SM_81002] Definition of API class ara::sm::s2r::S2RSatellite

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	class
Header file:	#include "ara/sm/s2r/S2RSatellite.h"
Forwarding header file:	#include "ara/sm/sm_fwd.h"
Scope:	namespace ara::sm::s2r
Symbol:	S2RSatellite
Syntax:	`class S2RSatellite {...};`
Description:	S2RSatellite abstract class.

⌋

8.3.1.1 Public Member Functions

8.3.1.1.1 Special Member Functions

8.3.1.1.1.1 Move Constructor

[SWS_SM_81006] Definition of API function ara::sm::s2r::S2RSatellite::S2RSatellite

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function	
Header file:	#include "ara/sm/s2r/S2RSatellite.h"	
Scope:	`class ara::sm::s2r::S2RSatellite`	
Syntax:	`S2RSatellite (S2RSatellite &&ra) noexcept;`	
Parameters (in):	ra	The S2RSatellite object to be moved.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Move constructor for S2RSatellite.	

⌋

8.3.1.1.1.2 Default Constructor

[SWS_SM_81004] Definition of API function ara::sm::s2r::S2RSatellite::S2RSatellite

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function
Header file:	#include "ara/sm/s2r/S2RSatellite.h"
Scope:	`class ara::sm::s2r::S2RSatellite`
Syntax:	`S2RSatellite ()=delete;`
Description:	Default constructor for S2RSatellite shall not be used.

⌋

8.3.1.1.1.3 Copy Constructor

[SWS_SM_81007] Definition of API function ara::sm::s2r::S2RSatellite::S2RSatellite

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function
Header file:	#include "ara/sm/s2r/S2RSatellite.h"
Scope:	`class ara::sm::s2r::S2RSatellite`
Syntax:	`S2RSatellite (const S2RSatellite &)=delete;`
Description:	The copy constructor for S2RSatellite shall not be used.

⌋

8.3.1.1.1.4 Move Assignment Operator

[SWS_SM_81008] Definition of API function ara::sm::s2r::S2RSatellite::operator=

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function	
Header file:	#include "ara/sm/s2r/S2RSatellite.h"	
Scope:	`class ara::sm::s2r::S2RSatellite`	
Syntax:	`S2RSatellite & operator= (S2RSatellite &&ra) noexcept;`	
Parameters (in):	ra	The S2RSatellite object to be moved.
Return value:	S2RSatellite &	The moved S2RSatellite object.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Description:	Move assignment operator for S2RSatellite.	

⌋

8.3.1.1.1.5 Copy Assignment Operator

[SWS_SM_81009] Definition of API function ara::sm::s2r::S2RSatellite::operator=

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function
Header file:	#include "ara/sm/s2r/S2RSatellite.h"
Scope:	`class ara::sm::s2r::S2RSatellite`
Syntax:	`S2RSatellite & operator= (const S2RSatellite &)=delete;`
Description:	The copy assignment operator for S2RSatellite shall not be used.

⌋

8.3.1.1.1.6 Destructor

[SWS_SM_81005] Definition of API function ara::sm::s2r::S2RSatellite::~S2RSatellite

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function
Header file:	#include "ara/sm/s2r/S2RSatellite.h"
Scope:	`class ara::sm::s2r::S2RSatellite`
Syntax:	`virtual ~S2RSatellite () noexcept;`
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Destructor for S2RSatellite.

⌋

8.3.1.1.2 Constructors

8.3.1.1.2.1 S2RSatellite

[SWS_SM_81003] Definition of API function ara::sm::s2r::S2RSatellite::S2RSatellite

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function	
Header file:	#include "ara/sm/s2r/S2RSatellite.h"	
Scope:	`class ara::sm::s2r::S2RSatellite`	
Syntax:	`explicit S2RSatellite (const ara::core::InstanceSpecifier &instance) noexcept;`	
Parameters (in):	instance	instance specifier to the PPortPrototype of a `SuspendToRamSatelliteInterface`
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Violations:	`InstanceSpecifierMappingIntegrityViolation`	InstanceSpecifier either cannot be resolved in the model in the context of your executable, or it refers to a model element other than a PortPrototype.
	`PortInterfaceMappingViolation`	The type of mapping does not match the expected type of Port Interface: {portInterfaceTypeName} referenced by a {mappingType Name}.
	`ProcessMappingViolation`	Matching InstanceRef exists, but no matching (modelled) Process found that matches the (runtime) process.
	`InstanceSpecifierAlreadyInUseViolation`	Violation message that is sent in case a constructor in the ara framework was called with an InstanceSpecifier already in use in this process.
	`AraNotInitializedViolation`	Violation message that is sent in case a constructor or function checks for an initialized ara and identifies that ara is not initialized.
Description:	Creation of an S2RSatellite.	

⌋

8.3.1.1.3 Member Functions

8.3.1.1.3.1 Create

[SWS_SM_81014] Definition of API function ara::sm::s2r::S2RSatellite::Create

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function
Header file:	#include "ara/sm/s2r/S2RSatellite.h"
Scope:	`class ara::sm::s2r::S2RSatellite`
Syntax:	`static ara::core::Result< S2RSatellite > Create () noexcept;`

▽

△

Return value:	ara::core::Result< S2 RSatellite >	a result that contains either a object or an error.
Exception Safety:	exception safe	
Thread Safety:	thread-safe	
Errors:	ara::sm::SmErrc::k AuthenticationRequired	rollback_semantics
		`S2R Satellite` requires IAM authentication. `ara::sm::s2r::S2RSatellite::S2RSatellite` has to be used instead.
Violations:	`AraNotInitialized-Violation`	Violation message that is sent in case a constructor or function checks for an initialized ara and identifies that ara is not initialized.
Description:	Creation of an conneting directly to S2RHub within the current machine.	

⌋

8.3.1.1.3.2 EnterSuspendMode

[SWS_SM_81010] Definition of API function ara::sm::s2r::S2RSatellite::EnterSuspendMode

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function	
Header file:	#include "ara/sm/s2r/S2RSatellite.h"	
Scope:	`class ara::sm::s2r::S2RSatellite`	
Syntax:	`virtual ara::core::Future< void > EnterSuspendMode ()=0;`	
Return value:	ara::core::Future< void >	void in case the `Suspend Mode` is successfully entered; or the error kRejected if there is an issue to enter `Suspend Mode`.
Exception Safety:	not exception safe	
Thread Safety:	not-threadsafe	
Errors:	SmErrc::kRejected	rollback_semantics
		There are issues to enter `Suspend Mode`.
Description:	EnterSuspendMode is called to enter `Suspend Mode`. The handler invocation needs to be enabled before by a call of Offer().	

⌋

8.3.1.1.3.3 LeaveSuspendMode

[SWS_SM_81011] Definition of API function ara::sm::s2r::S2RSatellite::LeaveSuspendMode

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function	
Header file:	#include "ara/sm/s2r/S2RSatellite.h"	
Scope:	`class ara::sm::s2r::S2RSatellite`	
Syntax:	`virtual ara::core::Future< void > LeaveSuspendMode ()=0;`	
Return value:	ara::core::Future< void >	void in case the `Suspend Mode` is left; or the error kRejected if the resume is not successful.
Exception Safety:	not exception safe	
Thread Safety:	not-threadsafe	
Errors:	SmErrc::kRejected	rollback_semantics
		There are issues to leave `Suspend Mode`.
Description:	LeaveSuspendMode is called to leave `Suspend Mode`. The handler invocation needs to be enabled before by a call of Offer().	

⌋

8.3.1.1.3.4 Offer

[SWS_SM_81012] Definition of API function ara::sm::s2r::S2RSatellite::Offer

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function	
Header file:	#include "ara/sm/s2r/S2RSatellite.h"	
Scope:	`class ara::sm::s2r::S2RSatellite`	
Syntax:	`ara::core::Result< void > Offer () noexcept;`	
Return value:	ara::core::Result< void >	A Result, being either empty or containing any of the errors defined below.
Exception Safety:	exception safe	
Thread Safety:	not thread-safe	
Errors:	SmErrc::kOfferFailed	rollback_semantics
		Service could not be offered due to failure of communication with Sm daemon
Description:	Enables potential invocations of handlers .	

⌋

8.3.1.1.3.5 StopOffer

[SWS_SM_81013] Definition of API function ara::sm::s2r::S2RSatellite::StopOffer

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function
Header file:	#include "ara/sm/s2r/S2RSatellite.h"
Scope:	`class ara::sm::s2r::S2RSatellite`
Syntax:	`void StopOffer () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not thread-safe
Description:	Disables invocations of handlers.

⌋

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 functional cluster does not define any default values for attributes specified in [13].

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	***AbstractSuspendToRamMapping*** (abstract)			
Note	This meta-class acts as an abstract base class for suspend-to-RAM-related mappings. **Tags:** atp.Status=candidate This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *Packageable Element*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Subclasses	SuspendToRamHubMapping, SuspendToRamSatelliteMapping			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
module Instantiation	SuspendToRamModule Instantiation	0..1	ref	This reference identifies the applicable suspend-to-RAM module instantiation. **Tags:** atp.Status=candidate
pPortPrototype InExecutable	PPortPrototype	0..1	iref	This reference identifies the suspend-to-RAM-related port. **Stereotypes:** atpUriDef **Tags:** atp.Status=candidate **InstanceRef implemented by:** PPortPrototypeIn ExecutableInstanceRef
process	Process	0..1	ref	This reference identifies the Process in which the Executable referenced in the role pPortPrototypeIn Executable is executed. **Tags:** atp.Status=candidate

Table A.1: AbstractSuspendToRamMapping

Class	**Executable**			
Note	This meta-class represents an executable program. **Tags:** atp.recommendedPackage=Executables This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDesignElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
implementation Props	Executable ImplementationProps	*	aggr	This aggregation contains the collection of implementation-specific properties necessary to properly build the enclosing Executable.
minimumTimer Granularity	TimeValue	0..1	attr	This attribute describes the minimum timer resolution (TimeValue of one tick) that is required by the Executable.
reporting Behavior	ExecutionState ReportingBehavior Enum	0..1	attr	this attribute controls the execution state reporting behavior of the enclosing Executable.
rootSw Component Prototype	RootSwComponent Prototype	0..1	aggr	This represents the root SwCompositionPrototype of the Executable. This aggregation is required (in contrast to a direct reference of a SwComponentType) in order to support the definition of instanceRefs in Executable context.

△

Class	**Executable**			
suspendToRam Awareness	SuspendToRam AwarenessEnum	0..1	attr	This attribute describes the type of awareness of the enclosing Executable to suspend-to-RAM functionality. **Tags:** atp.Status=candidate
version	StrongRevisionLabel String	0..1	attr	Version of the executable.

Table A.2: Executable

Class	**FunctionGroupErrorMapping**			
Note	This meta-class is used to associate an error code to an entire function group. The error code shall be used if any function group state change in the referenced function group fails.			
Base	*ARObject*, *Describable*			
Aggregated by	StateManagementModuleInstantiation.functionGroupErrorMapping			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
functionGroup	ModeDeclarationGroup Prototype	0..1	ref	This reference identifies the function group that is affected by an error raised if a request to transition a function group failed. **Tags:** atp.Status=draft This Attribute is only used by the AUTOSAR Adaptive Platform.
functionGroup Transition RequestFailed Error	ApApplicationError	0..1	ref	This reference identifies the ApApplicationError to be raised if a request to transition a function group failed. **Tags:** atp.Status=draft This Attribute is only used by the AUTOSAR Adaptive Platform.

Table A.3: FunctionGroupErrorMapping

Class	***Identifiable*** (abstract)
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, ClientIdDefinition, ClientServer Operation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEvent Group, CouplingPort, *CouplingPortAbstractShaper*, *CouplingPortStructuralElement*, CryptoCertificate, CryptoCertificateGroup, CryptoKeySlot, CryptoKeySlotDesign, CryptoKeySlotUsageDesign, Crypto Provider, *CryptoServiceMapping*, DataPrototypeGroup, DataPrototypeTransformationPropsIdent, Data Transformation, *DdsAbstractServiceInstanceElementCp*, DdsCpDomain, DdsCpPartition, DdsCpQos Profile, DdsCpTopic, DdsDomainRange, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, *DiagnosticAbstractSovdContent*, DiagnosticAuthTransmitCertificateEvaluation, DiagnosticConnected Indicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, DiagnosticExtendedDataRecord Element, DiagnosticFunctionInhibitSource, DiagnosticParameterElement, *DiagnosticRoutineSubfunction*, DiagnosticSovdMethodPrimitive, DltApplication, DltArgument, DltArgumentProps, DltMessage, DoIp Interface, DoIpLogicAddress, DoIpLogicalAddress, DoIpNetworkConfigurationDesign, DoIpRouting Activation, E2EProfileConfiguration, End2EndEventProtectionProps, End2EndMethodProtectionProps, EthernetWakeupSleepOnDatalineConfig, EventHandler, EventMapping, ExclusiveArea, *Executable Entity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeature ▽

▽

△

Class	***Identifiable*** (abstract)			
	△ MapElement, FMFeatureRelation, FMFeatureRestriction, FMFeatureSelection, FieldMapping, FireAndForgetMethodMapping, FlexrayArTpNode, FlexrayTpPduPool, *FrameTriggering*, GeneralParameter, GlobalSupervision, GlobalTimeGateway, *GlobalTimeMaster*, *GlobalTimeSlave*, *HealthChannel*, *HeapUsage*, HwAttributeDef, HwAttributeLiteralDef, HwPin, HwPinGroup, *IEEE1722TpAcfBus*, *IEEE1722TpAcfBusPart*, IPSecRule, IPv6ExtHeaderFilterList, ISignalToIPduMapping, ISignalTriggering, *IdentCaption*, ImpositionTime, InternalTriggeringPoint, Keyword, LifeCycleState, Linker, MacAddressVlanMembership, MacMulticastGroup, MacSecKayParticipant, McDataInstance, MemorySection, MemoryUsage, MethodMapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, ModeSwitchSenderComSpecProps, NetworkEndpoint, *NmCluster*, *NmNode*, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PersistencyDeploymentElement*, *PersistencyInterfaceElement*, *PhmSupervision*, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKeySlotMapping, QueuedReceiverComSpecProps, ResourceConsumption, ResourceGroup, RootSwClusterDesignComponentPrototype, RootSwComponentPrototype, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecOcJobMapping, SecOcJobRequirement, SecureCommunicationAuthenticationProps, *SecureCommunicationDeployment*, SecureCommunicationFreshnessProps, SecurityEventContextDataElement, SecurityEventContextProps, ServerComSpecProps, *ServiceInterfaceDeploymentElement*, ServiceInterfaceElementSecureComConfig, *ServiceNeeds*, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SoftwarePackageStep, SomeipEventGroup, SomeipProvidedEventGroup, SomeipTpChannel, *StackUsage*, *StateManagementActionItem*, StateManagementActionList, StateManagementStateNotification, *StateManagementStateRequest*, StaticSocketConnection, StructuredReq, SupervisionCheckpoint, SupervisionMode, SupervisionModeCondition, SwGenericAxisParamType, SwServiceArg, SwcServiceDependency, SystemMapping, *TimeBaseResource*, *TimingClock*, TimingClockSyncAccuracy, TimingCondition, *TimingConstraint*, *TimingDescription*, TimingExtensionResource, TimingModeInstance, TlsCryptoCipherSuite, TlsCryptoCipherSuiteProps, TlsJobMapping, Topic1, TpAddress, TraceableTable, TraceableText, *TracedFailure*, TransformationISignalPropsIdent, *TransformationProps*, TransformationTechnology, Trigger, UcmDescription, UcmRetryStrategy, UcmStep, UriDescription, 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	MultiLanguageOverviewParagraph	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

▽

△

Class	***Identifiable*** (abstract)			
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.4: Identifiable

Class	**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.5: ModeDeclaration

Class	**ModeDeclarationGroup**			
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.6: ModeDeclarationGroup

Class	ModeDeclarationGroupPrototype			
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.7: ModeDeclarationGroupPrototype

Class	NmInteractsWithSmMapping			
Note	This mapping represents an interaction from network management to state management. **Tags:** atp.Status=draft atp.recommendedPackage=FCInteractions This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *FunctionalClusterInteractsWithFunctionalClusterMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
nmNetworkHandle	NmNetworkHandle	0..1	ref	This reference identifies the network management handle that wants to interact with state management. **Tags:** atp.Status=draft
stateRequest	StateManagementStateRequest	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.8: NmInteractsWithSmMapping

Class	NmNetworkHandle			
Note	Group of partialNetworks and/or VLANs that can be controlled collectively. This Class is only used by the AUTOSAR Adaptive Platform.			
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 NmNetworkHandle. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=partialNetwork
vlan	EthernetCommunicationConnector	*	ref	Reference to a VLAN that is included in the NmNetworkHandle.

Table A.9: NmNetworkHandle

Enumeration	**NmStateRequestEnum**
Note	This enumeration defines the description of states that can be requested from the network management. **Tags:** atp.Status=draft This Enumeration is only used by the AUTOSAR Adaptive Platform.
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
noCom	This literal represents that case that no communication should be possible. **Tags:** atp.EnumerationLiteralIndex=0 atp.Status=draft

Table A.10: NmStateRequestEnum

Class	***PortInterface*** (abstract)			
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*, *AbstractSuspendToRamInterface*, *AbstractSynchronizedTimeBase Interface*, ClientServerInterface, *CryptoInterface*, *DataInterface*, *DiagnosticPortInterface*, FirewallState SwitchInterface, *IdsmAbstractPortInterface*, LogAndTraceInterface, ModeSwitchInterface, Network ManagementPortInterface, *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 This Attribute is only used by the AUTOSAR Adaptive Platform.

Table A.11: PortInterface

Class	**Process**			
Note	This meta-class provides information required to execute the referenced `Executable`. **Tags:** atp.recommendedPackage=Processes This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *AbstractExecutionContext*, *AtpClassifier*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *Uploadable PackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
design	ProcessDesign	0..1	ref	This reference represents the identification of the design-time representation for the Process that owns the reference.
executable	Executable	*	ref	Reference to executable that is executed in the process. **Stereotypes:** atpUriDef
functionCluster Affiliation	String	0..1	attr	This attribute specifies which functional cluster the Process is affiliated with.

▽

△

Class	**Process**			
numberOf RestartAttempts	PositiveInteger	0..1	attr	This attribute defines how often a process shall be restarted if the start fails. numberOfRestartAttempts = "0" OR Attribute not existing, start once numberOfRestartAttempts = "1", start a second time
preMapping	Boolean	0..1	attr	This attribute describes whether the executable is preloaded into the memory.
processState Machine	ModeDeclarationGroup Prototype	0..1	aggr	Set of Process States that are defined for the process. This attribute is used to support the modeling of execution dependencies that utilize the condition of process state. Please note that the process states may not be modeled arbitrarily at any stage of the AUTOSAR workflow because the supported states are standardized in the context of the SWS Execution Management [14].
stateDependent StartupConfig	StateDependentStartup Config	*	aggr	Applicable startup configurations.

Table A.12: Process

Class	**ServiceInterface**			
Note	This represents the ability to define a PortInterface that consists of a heterogeneous collection of methods, events and fields. **Tags:** atp.recommendedPackage=ServiceInterfaces This Class is only used by the AUTOSAR Adaptive Platform.			
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

▽

△

Class	**ServiceInterface**			
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.variation Point.shortLabel vh.latestBindingTime=blueprintDerivationTime xml.sequenceOffset=60

Table A.13: ServiceInterface

Class	**SmInteractsWithNmMapping**			
Note	This mapping represents an interaction from state management to network management. **Tags:** atp.Status=draft atp.recommendedPackage=FCInteractions This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *CollectableElement*, *FunctionalClusterInteractsWithFunctionalClusterMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeployment Element*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
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.14: SmInteractsWithNmMapping

Class	***StateManagementActionItem*** (abstract)			
Note	This meta-class represents an action item that is executed in response to a state change. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*			
Subclasses	StateManagementNmActionItem, StateManagementSetFunctionGroupStateActionItem, State ManagementSleepActionItem, StateManagementStateMachineActionItem, *StateManagementSuspend ToRamActionItem*, StateManagementSyncActionItem			
Aggregated by	StateManagementActionList.actionItem			
Attribute	*Type*	*Mult.*	*Kind*	*Note*
–	–	–	–	–

Table A.15: StateManagementActionItem

Class	**StateManagementActionList**
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 This Class is only used by the AUTOSAR Adaptive Platform.
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*

▽

△

Class	**StateManagementActionList**			
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
actionList Processing FailedError	ApApplicationError	0..1	ref	This reference identifies the error code for the case that the enclosing action list fails to process successfully. This reference is only relevant for state management agents. **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
maxActionList Duration	TimeValue	0..1	attr	This attribute defines the maximum duration in which the enclosing StateManagementActionList shall finish execution **Tags:** atp.Status=draft
maxDuration ExceededError	ApApplicationError	0..1	ref	This reference identifies the ApApplicationError that shall be triggered if the configured maximum duration of the execution of the action item list is exceeded. **Tags:** atp.Status=draft

Table A.16: StateManagementActionList

Class	**StateManagementEnterSuspendToRamActionItem**			
Note	This meta-class represents a state management action item to trigger S2R-aware application to enter the Suspend2Ram state. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StateManagementActionItem*, *StateManagementSuspendToRamActionItem*			
Aggregated by	StateManagementActionList.actionItem			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
enterSuspend ToRamError	ApApplicationError	0..1	ref	This reference identifies the error code for the case that entering suspend-to-RAM fails. **Tags:** atp.Status=draft

Table A.17: StateManagementEnterSuspendToRamActionItem

Class	**StateManagementEnterSuspendToRamOsActionItem**			
Note	This meta-class represents a state management action item to trigger the OS to enter the Suspend2Ram state. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StateManagementActionItem*, *StateManagementSuspendToRamActionItem*			
Aggregated by	StateManagementActionList.actionItem			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.18: StateManagementEnterSuspendToRamOsActionItem

Class	StateManagementErrorCompareRule			
Note	This meta-class represents the configuration of a compare rule for the processing of an error submission. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARObject*, *StateManagementCompareCondition*, *StateManagementCompareFormulaPart*			
Aggregated by	StateManagementCompareFormula.part			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.19: StateManagementErrorCompareRule

Class	StateManagementLeaveSuspendToRamActionItem			
Note	This meta-class represents a state management action item to trigger S2R-aware application to leave the Suspend2Ram state. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StateManagementActionItem*, *StateManagementSuspendToRamActionItem*			
Aggregated by	StateManagementActionList.actionItem			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
leaveSuspendToRamError	ApApplicationError	0..1	ref	This reference identifies the error code for the case that leaving suspend-to-RAM fails. **Tags:** atp.Status=draft

Table A.20: StateManagementLeaveSuspendToRamActionItem

Class	StateManagementModuleInstantiation			
Note	This meta-class represents the target-configuration-level configuration of the state management on the AUTOSAR adaptive platform. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARObject*, *AdaptiveModuleInstantiation*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *NonOsModuleInstantiation*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, Machine.moduleInstantiation			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
actionItemList	StateManagementActionList	*	aggr	This represents the collection of action item lists defined in the context of the enclosing state management module. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=actionItemList.shortName atp.Status=draft
functionGroupErrorMapping	FunctionGroupErrorMapping	*	aggr	This aggregation continas the collection of FunctionGroup ErrorMappings in the context of the enclosing State ManagementModuleInstantiation. **Tags:** atp.Status=draft
maxActionListDuration	TimeValue	0..1	attr	This attribute defines a global value for the maximum duration in which any enclosed StateManagementAction List shall finish execution. the value in this attribute will be superseded by the definition of attribute maxActionList Duration in the context of a specific StateManagement ActionList **Tags:** atp.Status=draft

▽

△

Class	StateManagementModuleInstantiation			
notification	StateManagementStateNotification	*	aggr	This aggregation represents the state switch notifications handled by the state manager. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=notification.shortName atp.Status=draft
request	StateManagementStateRequest	*	aggr	This aggregation represents the state requests handled by the state manager. **Stereotypes:** atpSplitable **Tags:** atp.Splitkey=request.shortName atp.Status=draft

Table A.21: StateManagementModuleInstantiation

Class	StateManagementNmActionItem			
Note	This meta-class represents a state management action item to interact with the network management. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.			
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.22: StateManagementNmActionItem

Class	***StateManagementPortInterface*** (abstract)			
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 This Class is only used by the AUTOSAR Adaptive Platform.			
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.23: StateManagementPortInterface

Class	StateManagementSetFunctionGroupStateActionItem			
Note	This meta-class represents a state management action item to set a specific state in a specific function group. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.			
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.24: StateManagementSetFunctionGroupStateActionItem

Class	StateManagementSleepActionItem			
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 This Class is only used by the AUTOSAR Adaptive Platform.			
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.25: StateManagementSleepActionItem

Class	StateManagementStateMachineActionItem			
Note	This meta-class represents a state management action item to start or stop a state machine. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.			
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
startAgent	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
startAgentError	ApApplicationError	0..1	ref	This reference identifies the error that shall be raised if the staring of an agent failed. **Tags:** atp.Status=draft
stopAgent	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
stopAgentError	ApApplicationError	0..1	ref	This reference identifies the error that shall be raised if the stopping of an agent failed. **Tags:** atp.Status=draft

Table A.26: StateManagementStateMachineActionItem

Class	StateManagementStateNotification			
Note	This meta-class represents the ability to formalize state notifications on the AUTOSAR adaptive platform. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.			
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.27: StateManagementStateNotification

Class	***StateManagementStateRequest*** (abstract)			
Note	This abstract class serves as the base class for state requests on the AUTOSAR adaptive platform. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.			
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.28: StateManagementStateRequest

Class	***StateManagementSuspendToRamActionItem*** (abstract)			
Note	This meta-class serves as an abstract base class for all suspend-to-RAM-related subclasses. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StateManagementActionItem*			
Subclasses	StateManagementEnterSuspendToRamActionItem, StateManagementEnterSuspendToRamOsAction Item, StateManagementLeaveSuspendToRamActionItem			
Aggregated by	StateManagementActionList.actionItem			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
maxActionItem Duration	TimeValue	0..1	attr	This attribute denotes the amount of time after which the execution of the action item is considered failed. **Tags:** atp.Status=draft

Table A.29: StateManagementSuspendToRamActionItem

Class	StateManagementSyncActionItem
Note	This meta-class represents a state management action item to synchronize state machines. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.

▽

△

Class	**StateManagementSyncActionItem**			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StateManagementActionItem*			
Aggregated by	StateManagementActionList.actionItem			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table A.30: StateManagementSyncActionItem

Class	**StateManagementTriggerCompareRule**			
Note	This meta-class represents the configuration of a compare rule for the processing of a trigger request. **Tags:** atp.Status=draft This Class is only used by the AUTOSAR Adaptive Platform.			
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:** ModeDeclarationInState ManagementStateNotificationInstanceRef

Table A.31: StateManagementTriggerCompareRule

Enumeration	**SuspendToRamAwarenessEnum**
Note	This enumeration provides values that describe the awareness of an Executable to suspend-to-RAM functionality. **Tags:** atp.Status=candidate This Enumeration is only used by the AUTOSAR Adaptive Platform.
Aggregated by	Executable.suspendToRamAwareness
Literal	***Description***
suspendToRam Aware	The Executable is notified of suspend-to-RAM activity and can prepare accordingly. **Tags:** atp.EnumerationLiteralIndex=2 atp.Status=candidate
suspendToRamNot Supported	The Executable does not support suspend-to-RAM. **Tags:** atp.EnumerationLiteralIndex=1 atp.Status=candidate
suspendToRam Tolerant	The Executable is not notified of suspend-to-RAM activity and seamlessly handles suspend-to-RAM activities. **Tags:** atp.EnumerationLiteralIndex=0 atp.Status=candidate

Table A.32: SuspendToRamAwarenessEnum

Class	**SuspendToRamHubMapping**
Note	This mapping associates a suspend-to-RAM hub with the applicable module instantiation. **Tags:** atp.Status=candidate atp.recommendedPackage=SuspendToRamMappings This Class is only used by the AUTOSAR Adaptive Platform.

▽

△

Class	SuspendToRamHubMapping			
Base	*ARElement*, *ARObject*, *AbstractSuspendToRamMapping*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *Uploadable PackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.33: SuspendToRamHubMapping

Class	SuspendToRamModuleInstantiation			
Note	This meta-class represents the ability to define the target-configuration of a suspend-to-RAM module instantiation. **Tags:** atp.Status=candidate This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARObject*, *AdaptiveModuleInstantiation*, *AtpClassifier*, *AtpFeature*, *AtpStructureElement*, *Identifiable*, *MultilanguageReferrable*, *NonOsModuleInstantiation*, *Referrable*			
Aggregated by	*AtpClassifier*.atpFeature, Machine.moduleInstantiation			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.34: SuspendToRamModuleInstantiation

Class	SuspendToRamSatelliteInterface			
Note	This meta-class represents a satellite-side PortInterface for the implementation of suspend-to-RAM functionality. **Tags:** atp.Status=candidate atp.recommendedPackage=SuspendToRamInterfaces This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *AbstractSuspendToRamInterface*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.35: SuspendToRamSatelliteInterface

Class	SuspendToRamSatelliteMapping			
Note	This mapping associates a suspend-to-RAM satellite with the applicable module instantiation. **Tags:** atp.Status=candidate atp.recommendedPackage=SuspendToRamMappings This Class is only used by the AUTOSAR Adaptive Platform.			
Base	*ARElement*, *ARObject*, *AbstractSuspendToRamMapping*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *Uploadable PackageElement*			
Aggregated by	ARPackage.element			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
–	–	–	–	–

Table A.36: SuspendToRamSatelliteMapping

B Demands and constraints on Base Software (normative)

This functional cluster defines no demands or constraints for the Base Software on which the AUTOSAR Adaptive Platform is running on (usually a POSIX-compatible operating system).

C Platform Extension Interfaces (normative)

This chapter provides a reference of the Platform Extension Interfaces defined by this functional cluster. Platform Extension Interfaces are intended to be used/provided by an OEM or Integrator to extend the functionality of the AUTOSAR Adaptive Platform.

C.1 Header: apext/sm/power_state_interface.h

C.1.1 Non-Member Types

C.1.1.1 Enumeration: PowerState

[SWS_SM_71002] Definition of API enum apext::sm::PowerState

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	enumeration	
Header file:	#include "apext/sm/power_state_interface.h"	
Forwarding header file:	#include "apext/sm/sm_fwd.h"	
Scope:	namespace apext::sm	
Symbol:	PowerState	
Underlying type:	std::uint32_t	
Syntax:	`enum class PowerState : std::uint32_t {...};`	
Values:	kPmSuspendToIdle	= 1
		Derived from Linux Power State PM_SUSPEND_TO_IDLE (Kernel view)
	kPmSuspendToRam	= 2
		Derived from Linux Power State PM_SUSPEND_TO_RAM (Kernel view)
	kPmHibernation	= 3
		Derived from Linux Power State PM_HIBERNATION (Kernel view)
	kPmOff	= 4
		Derived from Linux Power State PM_OFF (Kernel view)
Description:	Power State Enumeration.	

⌋

C.1.1.2 Type Alias: WakeUpHandler

[SWS_SM_71004] Definition of API type apext::sm::WakeUpHandler

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	type alias
Header file:	#include "apext/sm/power_state_interface.h"
Scope:	namespace apext::sm
Symbol:	WakeUpHandler
Syntax:	`using WakeUpHandler = std::function<void(void)>;`
Thread Safety:	not thread-safe
Description:	WakeUpHandler function.

⌋

C.1.2 Class: PowerStateInterface

[SWS_SM_71000] Definition of API class apext::sm::PowerStateInterface

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	class
Header file:	#include "apext/sm/power_state_interface.h"
Forwarding header file:	#include "apext/sm/sm_fwd.h"
Scope:	namespace apext::sm
Symbol:	PowerStateInterface
Syntax:	`class PowerStateInterface {...};`
Description:	class for power state coordination with OS

⌋

C.1.2.1 Public Member Functions

C.1.2.1.1 Member Functions

C.1.2.1.1.1 RegisterWakeUpHandler

[SWS_SM_71003] Definition of API function apext::sm::PowerStateInterface::RegisterWakeUpHandler

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function	
Header file:	#include "apext/sm/power_state_interface.h"	
Scope:	`class apext::sm::PowerStateInterface`	
Syntax:	`void RegisterWakeUpHandler (apext::sm::WakeUpHandler handler) noexcept;`	
DIRECTION NOT DEFINED	handler	--
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	not-threadsafe	
Description:	Register wake-up handler function in the OS-Wrapper to inform SM about wake-up.	

⌋

C.1.2.1.1.2 SetPowerState

[SWS_SM_71001] Definition of API function apext::sm::PowerStateInterface::SetPowerState

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function	
Header file:	#include "apext/sm/power_state_interface.h"	
Scope:	`class apext::sm::PowerStateInterface`	
Syntax:	`void SetPowerState (apext::sm::PowerState state) noexcept;`	
DIRECTION NOT DEFINED	state	--
Return value:	None	
Exception Safety:	exception safe	
Thread Safety:	not-threadsafe	
Description:	Forward the power state request to OS.	

⌋

C.1.2.1.1.3 UnregisterWakeUpHandler

[SWS_SM_71005] Definition of API function apext::sm::PowerStateInterface::UnregisterWakeUpHandler

Status: DRAFT
Upstream requirements: RS_SM_00402

⌈

Kind:	function
Header file:	#include "apext/sm/power_state_interface.h"
Scope:	`class apext::sm::PowerStateInterface`
Syntax:	`void UnregisterWakeUpHandler () noexcept;`
Return value:	None
Exception Safety:	exception safe
Thread Safety:	not-threadsafe
Description:	Unregister wake-up handler in OS-Wrapper.

⌋

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 R24-11 and R25-11

E.1.1 Added Specification Items in R25-11

Number	Heading
[SWS_SM_00214]	Reject Suspend-to-RAM during active update session
[SWS_SM_00667]	Finalization of ActionList processing
[SWS_SM_00668]	Default value for ActionListTimeout
[SWS_SM_00669]	ActionList timeout monitoring
[SWS_SM_00670]	ErrorCode as reaction to a failed Process
[SWS_SM_00671]	ErrorCode as reaction to a failed SetState request
[SWS_SM_00672]	Default value for functionGroupTransitionRequestFailedError
[SWS_SM_00673]	ErrorCode as reaction to an ActionList timeout
[SWS_SM_00674]	Default value for StateManagementActionList.maxDurationExceededError
[SWS_SM_00675]	ErrorCode for failed creation of a StateMachine of type Agent
[SWS_SM_00676]	Default value for startAgentError
[SWS_SM_00677]	ErrorCode for failed termination of a StateMachine of type Agent
[SWS_SM_00678]	Default value for stopAgentError
[SWS_SM_00679]	ErrorCode for failure during processing of an Agent ActionList
[SWS_SM_00680]	Robust execution of LeaveSuspendToRam ActionItem
[SWS_SM_00681]	Autonomous wake-up due to update session request
[SWS_SM_00682]	Conditional approval of RequestUpdateSession after autonomous wake-up
[SWS_SM_00683]	Autonomous wake-up due to OS system call
[SWS_SM_00684]	Mutual exclusivity of StateMachine approach and S2RHub API usage
[SWS_SM_00685]	ErrorCode for failed EnterSuspendToRam
[SWS_SM_00686]	Default value for enterSuspendToRamError
[SWS_SM_00687]	ErrorCode for failed LeaveSuspendToRam
[SWS_SM_00688]	Default value for leaveSuspendToRamError
[SWS_SM_00689]	ActionListItem - Trigger EnterSuspendState
[SWS_SM_00690]	Instantiation of ara::sm::s2r::S2RHub
[SWS_SM_00691]	ActionListItem - Trigger EnterSuspendState
[SWS_SM_00692]	ActionListItem - Trigger EnterSuspendState

△

Number	Heading
[SWS_SM_00701]	Request satellites to enter Suspend Mode
[SWS_SM_00702]	Returning EnterSuspendState
[SWS_SM_00703]	Request satellites to leave Suspend Mode
[SWS_SM_00705]	Request OS to enter suspend state
[SWS_SM_00706]	IAM check
[SWS_SM_00707]	Returning LeaveSuspendState
[SWS_SM_00710]	Registration of S2R Satellites at S2R Hub
[SWS_SM_00711]	Trigger to enter and leave Suspend Mode
[SWS_SM_00712]	IAM check in S2R Satellite
[SWS_SM_00713]	Deregistration of S2R Satellites at S2R Hub
[SWS_SM_00714]	IAM demand
[SWS_SM_00715]	Restrict to only authenticated S2R Satellites
[SWS_SM_00716]	Forced IAM check in S2R Satellite
[SWS_SM_00717]	Instantiation of apext::sm::PowerStateInterface
[SWS_SM_70000]	Security events for State Management
[SWS_SM_70001]	Security event context data definition: SEV_ACCESS_CONTROL_SM_IAM_ACCESS_DENIED
[SWS_SM_71000]	Definition of API class apext::sm::PowerStateInterface
[SWS_SM_71001]	Definition of API function apext::sm::PowerStateInterface::SetPowerState
[SWS_SM_71002]	Definition of API enum apext::sm::PowerState
[SWS_SM_71003]	Definition of API function apext::sm::PowerStateInterface::RegisterWakeUpHandler
[SWS_SM_71004]	Definition of API type apext::sm::WakeUpHandler
[SWS_SM_71005]	Definition of API function apext::sm::PowerStateInterface::UnregisterWakeUpHandler
[SWS_SM_81002]	Definition of API class ara::sm::s2r::S2RSatellite
[SWS_SM_81003]	Definition of API function ara::sm::s2r::S2RSatellite::S2RSatellite
[SWS_SM_81004]	Definition of API function ara::sm::s2r::S2RSatellite::S2RSatellite
[SWS_SM_81005]	Definition of API function ara::sm::s2r::S2RSatellite::~S2RSatellite
[SWS_SM_81006]	Definition of API function ara::sm::s2r::S2RSatellite::S2RSatellite
[SWS_SM_81007]	Definition of API function ara::sm::s2r::S2RSatellite::S2RSatellite
[SWS_SM_81008]	Definition of API function ara::sm::s2r::S2RSatellite::operator=
[SWS_SM_81009]	Definition of API function ara::sm::s2r::S2RSatellite::operator=
[SWS_SM_81010]	Definition of API function ara::sm::s2r::S2RSatellite::EnterSuspendMode
[SWS_SM_81011]	Definition of API function ara::sm::s2r::S2RSatellite::LeaveSuspendMode
[SWS_SM_81012]	Definition of API function ara::sm::s2r::S2RSatellite::Offer
[SWS_SM_81013]	Definition of API function ara::sm::s2r::S2RSatellite::StopOffer
[SWS_SM_81014]	Definition of API function ara::sm::s2r::S2RSatellite::Create
[SWS_SM_81102]	Definition of API class ara::sm::s2r::S2RHub

▽

△

Number	Heading
[SWS_SM_81103]	Definition of API function ara::sm::s2r::S2RHub::S2RHub
[SWS_SM_81105]	Definition of API function ara::sm::s2r::S2RHub::~S2RHub
[SWS_SM_81106]	Definition of API function ara::sm::s2r::S2RHub::S2RHub
[SWS_SM_81107]	Definition of API function ara::sm::s2r::S2RHub::S2RHub
[SWS_SM_81108]	Definition of API function ara::sm::s2r::S2RHub::operator=
[SWS_SM_81109]	Definition of API function ara::sm::s2r::S2RHub::operator=
[SWS_SM_81110]	Definition of API function ara::sm::s2r::S2RHub::RequestToEnterSuspend Mode
[SWS_SM_81111]	Definition of API function ara::sm::s2r::S2RHub::RequestToLeaveSuspend Mode
[SWS_SM_81112]	Definition of API function ara::sm::s2r::S2RHub::EnterSuspendToRamOs
[SWS_SM_81240]	Definition of API enum ara::sm::SmErrc
[SWS_SM_81241]	Definition of API class ara::sm::SmErrorDomain
[SWS_SM_81242]	Definition of API class ara::sm::SmException
[SWS_SM_81243]	Definition of API function ara::sm::SmException::SmException
[SWS_SM_81244]	Definition of API function ara::sm::MakeErrorCode
[SWS_SM_81245]	Definition of API type ara::sm::SmErrorDomain::Errc
[SWS_SM_81246]	Definition of API type ara::sm::SmErrorDomain::Exception
[SWS_SM_81247]	Definition of API function ara::sm::SmErrorDomain::SmErrorDomain
[SWS_SM_81248]	Definition of API function ara::sm::SmErrorDomain::Name
[SWS_SM_81249]	Definition of API function ara::sm::SmErrorDomain::Message
[SWS_SM_81250]	Definition of API function ara::sm::SmErrorDomain::ThrowAsException
[SWS_SM_81251]	Definition of API function ara::sm::GetSmDomain

Table E.1: Added Specification Items in R25-11

E.1.2 Changed Specification Items in R25-11

Number	Heading
[SWS_SM_00031]	Nested recovery handling
[SWS_SM_00206]	prepare update
[SWS_SM_00207]	prepare verify
[SWS_SM_00208]	prepare rollback
[SWS_SM_00601]	StateMachine error notification reaction of StateMachines not "ImpactedBy Update"
[SWS_SM_00602]	StateMachine ErrorRecoveryOngoing flag reset
[SWS_SM_00605]	StateMachine service interface RequestTransition - recovery ongoing
[SWS_SM_00607]	StateMachine transition execution

▽

△

Number	Heading
[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_00614]	ActionListItem "Stop StateMachine" processing
[SWS_SM_00617]	CurrentState value after StateMachine State transition
[SWS_SM_00620]	StateMachine transition - NetworkHandle goes to FullCom
[SWS_SM_00621]	StateMachine transition - NetworkHandle goes to NoCom
[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_00629]	Only Process controlling StateMachine of type Controller can provide UpdateRequest interface
[SWS_SM_00640]	Successful verification of updated software
[SWS_SM_00650]	StateMachine service interface RequestTransition - transition failed
[SWS_SM_00651]	Processing StopStateMachine ActionListItem
[SWS_SM_00654]	StateMachine marked as "ImpactedByUpdate"
[SWS_SM_00665]	StateMachineNotification service interface
[SWS_SM_00666]	Nested recovery

Table E.2: Changed Specification Items in R25-11

E.1.3 Deleted Specification Items in R25-11

none

E.1.4 Added Constraints in R25-11

Number	Heading
[SWS_SM_CONSTR_00034]	Exclusive assignment of suspend-related ActionListItems in suspend-related states
[SWS_SM_CONSTR_00035]	Prohibited assignment of LeaveSuspendToRam in Suspend state

▽

△

Number	Heading
[SWS_SM_CONSTR_00036]	Existence of StateMachine State LeaveSuspend for StateMachine of type Controller
[SWS_SM_CONSTR_00037]	Mandatory ActionListItem in LeaveSuspend state

Table E.3: Added Constraints in R25-11

E.1.5 Changed Constraints in R25-11

Number	Heading
[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 ErrorCode
[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_00020]	Upper Multiplicity of UpdateRequest interface
[SWS_SM_CONSTR_00025]	NmNetworkHandle shall only be controlled by single StateMachine
[SWS_SM_CONSTR_00032]	Completeness of controlled NmNetworkHandles

Table E.4: Changed Constraints in R25-11

E.1.6 Deleted Constraints in R25-11

Number	Heading
[SWS_SM_CONSTR_00010]	ActionItems in initial StateMachine State

Table E.5: Deleted Constraints in R25-11

E.2 Constraint and Specification Item Changes between AUTOSAR Release R23-11 and R24-11

E.2.1 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

▽

△

Number	Heading
[SWS_SM_91101]	Definition of Method UpdateRequest.StopUpdateSession
[SWS_SM_91102]	Definition of Method UpdateRequest.RequestUpdateSession
[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.6: Added Specification Items in R24-11

E.2.2 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

▽

△

Number	Heading
[SWS_SM_00615]	ActionListItem "Stop StateMachine" processing - StateMachine is not running
[SWS_SM_00616]	CurrentState value during StateMachine State transition
[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

▽

△

Number	Heading
[SWS_SM_91017]	Definition of ServiceInterface UpdateRequest
[SWS_SM_91018]	Definition of ImplementationDataType FunctionGroupListType
[SWS_SM_91019]	Definition of ImplementationDataType FunctionGroupNameType
[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.7: Changed Specification Items in R24-11

E.2.3 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

▽

△

Number	Heading
[SWS_SM_91007]	Definition of ServiceInterface TriggerIn
[SWS_SM_91008]	Definition of ServiceInterface TriggerOut
[SWS_SM_91009]	Definition of ServiceInterface TriggerInOut

Table E.8: Deleted Specification Items in R24-11

E.2.4 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.9: Added Constraints in R24-11

E.2.5 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.10: Changed Constraints in R24-11

E.2.6 Deleted Constraints in R24-11

Number	Heading
[SWS_SM_CONSTR_00012]	Stop running StateMachines in the final state of a StateMachine

Table E.11: Deleted Constraints in R24-11

E.3 Constraint and Specification Item Changes between AUTOSAR Release R22-11 and R23-11

E.3.1 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
[SWS_SM_00639]	Unsuccessful verification of updated software

▽

△

Number	Heading
[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.12: Added Specification Items in R23-11

E.3.2 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
[SWS_SM_91022]	Definition of ServiceInterface StateMachineService

Table E.13: Changed Specification Items in R23-11

E.3.3 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.14: Deleted Specification Items in R23-11

E.3.4 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
[SWS_SM_CONSTR_00023]	Existence of StateMachine PrepareRollback state

Table E.15: Added Constraints in R23-11

E.3.5 Changed Constraints in R23-11

none

E.3.6 Deleted Constraints in R23-11

none

E.4 Constraint and Specification Item Changes between AUTOSAR Release R21-11 and R22-11

E.4.1 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
[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.16: Added Specification Items in R22-11

E.4.2 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.17: Changed Specification Items in R22-11

E.4.3 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.18: Deleted Specification Items in R22-11

E.4.4 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.19: Added Constraints in R22-11

E.4.5 Changed Constraints in R22-11

none

E.4.6 Deleted Constraints in R22-11

none

E.5 Constraint and Specification Item Changes between AUTOSAR Release R20-11 and R21-11

E.5.1 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

▽

△

Number	Heading
[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]	
[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

▽

△

Number	Heading
[SWS_SM_NA]	Not applicable requirements

Table E.20: Added Specification Items "in R21-11"

E.5.2 Changed Specification Items "in R21-11"

none

E.5.3 Deleted Specification Items "in R21-11"

none

E.5.4 Added Constraints "in R21-11"

none

E.5.5 Changed Constraints "in R21-11"

none

E.5.6 Deleted Constraints "in R21-11"

none

E.6 Constraint and Specification Item Changes between AUTOSAR Release R19-11 and R20-11

E.6.1 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

▽

△

Number	Heading
[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
[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]	

▽

△

Number	Heading
[SWS_SM_91017]	
[SWS_SM_91018]	
[SWS_SM_91019]	
[SWS_SM_91020]	

Table E.21: Added Specification Items in R20-11

E.6.2 Changed Specification Items in R20-11

none

E.6.3 Deleted Specification Items in R20-11

none

E.6.4 Added Constraints in R20-11

none

E.6.5 Changed Constraints in R20-11

none

E.6.6 Deleted Constraints in R20-11

none

E.7 Constraint and Specification Item Changes between AUTOSAR Release R19-03 and R19-11

E.7.1 Added Specification Items in 19-11

none

E.7.2 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.22: Changed Specification Items in 19-11

E.7.3 Deleted Specification Items in 19-11

none

E.7.4 Added Constraints in 19-11

none

E.7.5 Changed Constraints in 19-11

none

E.7.6 Deleted Constraints in 19-11

none

E.8 Constraint and Specification Item Changes in AUTOSAR Release R19-03

E.8.1 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.23: Added Specification Items in 19-03

E.8.2 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.24: Changed Specification Items in 19-03

E.8.3 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
[SWS_SM_00015]	Component (states) Transitions
[SWS_SM_00102]	Component States for Reset

Table E.25: Deleted Specification Items in 19-03

E.8.4 Added Constraints in 19-03

none

E.8.5 Changed Constraints in 19-03

none

E.8.6 Deleted Constraints in 19-03

none