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	R23-11

Document Change History

Date	Release	Changed by	Description
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

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

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

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

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

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Contents

1 Introduction and functional overview 8

1.1 Interaction with AUTOSAR Runtime for Adaptive 8

2 Acronyms and Abbreviations 9

3 Further applicable specification 12

3.1 Input documents & related standards and norms 12

4 Constraints and assumptions 13

4.1 Known limitations 13
4.2 Applicability to car domains 13

5 Dependencies to other Functional Clusters 14

5.1 Provided Interfaces 14
5.2 Required Interfaces 15

6 Requirements Tracing 17

7 Functional specification 20

7.1 State Management Responsibilities 22
7.1.1 Machine State 23
7.1.1.1 Startup 25
7.1.1.2 Shutdown 25
7.1.1.3 Restart 26
7.1.2 Function Group State 26
7.1.3 State Management Architecture 27
7.2 State Management and Adaptive (Platform) Applications 28
7.2.1 Interaction between the SM and Adaptive Applications 28
7.2.2 Synchronization across multiple Adaptive Applications 30
7.2.2.1 PowerModes for Adaptive (Platform) Applications . . 32
7.2.2.2 Diagnostic Reset for Adaptive (Platform) Applications 33
7.3 Interaction with Platform Health Management 33
7.4 Interaction with Diagnostic Management 34
7.5 Interaction with Update and Configuration Management 35
7.6 Interaction with Network Management 38
7.7 Interaction with Execution Management 39
7.8 State Management in a virtualized/hierarchical environment 41
7.9 StateManagement lifecycle 42
7.9.1 Startup 42
7.9.2 Shutdown 42
7.9.3 Restart 42
7.10 Configuration 42
7.11 StateManagement StateMachine 42
7.11.1 StateMachine introduction 42
7.11.2 Controlling application for StateMachine States 44

7.11.3 StateMachine design considerations 47
7.11.4 StateMachine general conditions 49
7.11.5 StateMachine state changes 50
7.11.6 StateMachine ActionLists 53
7.11.7 StateMachine ActionListItems 53
7.11.8 Controlling multiple StateMachine Instances 56
7.11.9 ActionListItem Sleep 58
7.11.10 ActionListItem SetNetworkHandle 59
7.11.11 StateMachine State notification 60
7.11.12 StateMachine support for Update and Configuration Management 61

8 API specification 69

9 Service Interfaces 70

9.1 Type definitions 70
9.1.1 Data types for Update And Configuration Management interaction 70
9.1.2 Data types for StateMachine interaction 70
9.1.3 Data types for UpdateAllowed service interface 71
9.2 Provided Service Interfaces 72
9.2.1 State Management TriggerIn 72
9.2.2 State Management TriggerOut 73
9.2.3 State Management TriggerInOut 74
9.2.4 UpdateRequest 75
9.2.5 StateMachine service 77
9.2.6 StateMachine UpdateAllowed service 78
9.3 Required Service Interfaces 79
9.3.1 Network Management 79
9.3.1.1 NetworkManagement NetworkState 79
9.4 Application Errors 80
9.4.1 StateManagement Error Domain 80

A Interfunctional Cluster Interfaces 81

B Not applicable requirements 82

C Mentioned Manifest Elements 83

D History of Constraints and Specification Items 93

D.1 Constraint and Specification Item History of this document according to AUTOSAR Release R23-11 93
D.1.1 Added Specification Items in R23-11 93
D.1.2 Changed Specification Items in R23-11 94
D.1.3 Deleted Specification Items in R23-11 95
D.1.4 Added Constraints in R23-11 95
D.1.5 Changed Constraints in R23-11 95

D.1.6 Deleted Constraints in R23-11 96
D.2 Constraint and Specification Item History of this document according to AUTOSAR Release R22-11 96
D.2.1 Added Specification Items in R22-11 96
D.2.2 Changed Specification Items in R22-11 97
D.2.3 Deleted Specification Items in R22-11 97
D.2.4 Added Constraints in R22-11 98
D.2.5 Changed Constraints in R22-11 98
D.2.6 Deleted Constraints in R22-11 98
D.3 Constraint and Specification Item History of this document according to AUTOSAR Release R21-11 98
D.3.1 Added Specification Items "in R21-11" 98
D.3.2 Changed Specification Items "in R21-11" 100
D.3.3 Deleted Specification Items "in R21-11" 100
D.3.4 Added Constraints "in R21-11" 100
D.3.5 Changed Constraints "in R21-11" 100
D.3.6 Deleted Constraints "in R21-11" 100
D.4 Constraint and Specification Item History of this document according to AUTOSAR Release R20-11 101
D.4.1 Added Specification Items in R20-11 101
D.4.2 Changed Specification Items in R20-11 102
D.4.3 Deleted Specification Items in R20-11 102
D.4.4 Added Constraints in R20-11 102
D.4.5 Changed Constraints in R20-11 102
D.4.6 Deleted Constraints in R20-11 102
D.5 Constraint and Specification Item History of this document according to AUTOSAR Release R19-11 103
D.5.1 Added Specification Items in 19-11 103
D.5.2 Changed Specification Items in 19-11 103
D.5.3 Deleted Specification Items in 19-11 103
D.5.4 Added Constraints in 19-11 103
D.5.5 Changed Constraints in 19-11 103
D.5.6 Deleted Constraints in 19-11 103
D.6 Constraint and Specification Item History of this document according to AUTOSAR Release R19-03 104
D.6.1 Added Specification Items in 19-03 104
D.6.2 Changed Specification Items in 19-03 104
D.6.3 Deleted Specification Items in 19-03 104
D.6.4 Added Constraints in 19-03 105
D.6.5 Changed Constraints in 19-03 105
D.6.6 Deleted Constraints in 19-03 105

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.

State Management controls state of (partial networks using provided fields (NetworkHandle) of Network Management.

State Management interacts with the Execution Management to request Function Groups and the Machine State to enter specific states that are determined by project requirements. Function Group States might additionally depend on Network Managements State.

State Management provides access to its internal state via ara::com services. A particular service implements one of standardized service interfaces. The service interfaces have fields for getting current state (field "Notifier" (see section 9.2.2)) and requesting new state (field "Trigger" (see section 9.2.1)). AUTOSAR Adaptive Platform Applications or Adaptive Applications can use the fields for reacting on the system state changes or for influencing the system state(when they are configured to have write permissions).

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

1.1 Interaction with AUTOSAR Runtime for Adaptive

The set of programming interfaces to the Adaptive Applications is called AUTOSAR Runtime for Adaptive (ARA). APIs accessed by State Management using the interfunctional cluster API is described in Appendix A which is not part of ARA.

The Adaptive AUTOSAR Services are provided via mechanisms provided by the Communication Management functional cluster [1] of the Adaptive Platform Foundation

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[2].

Terms:	Description:
State Management	The element defining modes of operation for AUTOSAR Adaptive Platform. It allows flexible definition of functions which are active on the platform at any given time.
Execution Management [3]	The element of the AUTOSAR Adaptive Platform responsible for the ordered startup and shutdown of the AUTOSAR Adaptive Platform and Adaptive Applications.
Platform Health Management [4]	A Functional Cluster within the Adaptive Platform Foundation
Communication Management [1]	A Functional Cluster within the Adaptive Platform Foundation
Network Management [5]	A Functional Cluster within the Adaptive Platform Services. Part of Communication Management.
Diagnostic Management [6]	A Functional Cluster within the Adaptive Platform Services
Update And Configuration Management [7]	A Functional Cluster within the Adaptive Platform Services
Network Handle	Network Handles are provided by Network Management. A handle represents a set of (partial) networks.
process	A process refers to the OS concept of a running process. **Attention:** process is **not equal** to Modelled Process (see below). Hence each Modelled Process has at some time a related (OS) process but a process may not always have a related Modelled Process.
Modelled Process	A Modelled Process is an instance of an Executable to be executed on a Machine and has a 1:1 association with the ARXML/Meta-Model element Modelled Process. This document also uses the term process (without the “modelled” prefix) to refer to the OS concept of a running process.
Function Group	A Function Group is a set of coherent Modelled Processes which need to be controlled consistently. Depending on the state of the Function Group, processes (related to the Modelled Processes) are started or terminated. Modelled Processes can belong to more than one Function Group State (but at exactly one Function Group). "MachineFG" is a Function Group with a predefined name, which is mainly used to control Machine lifecycle and processes of platform level Applications. Other Function Groups are sort of general purpose tools used (for example) to control processes of user level Applications.
Function Group State	The element of State Management that characterizes the current status of a set of (functionally coherent) user-level Applications. The set of Function Groups and their Function Group States is machine specific and are configured in the Machine Manifest [8].
Machine State	The state of Function Group "MachineFG" with some predefined states (Startup/Shutdown/Restart).
Execution Manifest	Manifest file to configure execution of an Adaptive Application.

Machine Manifest	Manifest file to configure a Machine. The Machine Manifest holds all configuration information which cannot be assigned to a specific Executable or process.
StateMachine	Identifiable entity which consists of at least two StateMachine States.StateMachine is modeled as a ModeDeclarationGroupPrototype
StateMachine State	State of a StateMachine, which is referenced by an ActionList. StateMachine State is represented by meta-class ModeDeclaration
Initial State	StateMachine State of a StateMachine, which is automatically entered, when a StateMachine starts/ is instantiated.
Final State	StateMachine State of a StateMachine, which is automatically entered, when a StateMachine stops/ is destroyed.
ActionList	Entity which references a StateMachine State of a StateMachine. Contains an arbitrary number of ActionListItems. Entity is represented by meta-class StateManagementActionList
ActionListItem	Item of an ActionList. Items will be executed when a StateMachine State is entered. Entity is represented by meta-class StateManagementActionItem
TransitionRequestTable	Table which defines next StateMachine State, depending on current StateMachine State and on value passed via StateMachineService interface.
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	Table which defines next StateMachine State, depending on ErrorEvent value passed in StateMachine error notification.
SMControlApplication	Project-specific Adaptive Application(s) which evaluates information from the system to request StateMachine State changes from a StateMachine via StateMachineService interface.
ErrorRecoveryOngoing	StateMachine internal flag, which is set, when it receives error notification from Platform Health Management or Execution Management. Flag is reset, when all Actions to recover from this situation are successfully done.
Controller	StateMachine with the role Controller (also known as Master). Exists only once in an Adaptive machine (when StateMachine is configured). The Controller is the main StateMachine which is responsible for the machine life-cycle.
Agent	StateMachine with the role Agent (also known as Slave). Optional StateMachines (beside Controller). Life-cycle of Agents is managed by Controller. Caring about set of Function Groups.

Table 2.1: Technical Terms

The following technical terms used throughout this document are defined in the official [2] AUTOSAR Glossary or [8] TPS Manifest Specification – they are repeated here for tracing purposes.

Term	Description
Adaptive Application	see [2] AUTOSAR Glossary
Application	see [2] AUTOSAR Glossary
AUTOSAR Adaptive Platform	see [2] AUTOSAR Glossary
Adaptive Platform Foundation	see [2] AUTOSAR Glossary
Adaptive Platform Services	see [2] AUTOSAR Glossary
Manifest	see [2] AUTOSAR Glossary
Executable	see [2] AUTOSAR Glossary
Functional Cluster	see [2] AUTOSAR Glossary
Software Cluster	see [2] AUTOSAR Glossary
Diagnostic Address	see [2] AUTOSAR Glossary
Identity and Access Management	see [2] AUTOSAR Glossary
Machine	see [2] AUTOSAR Glossary
Service	see [2] AUTOSAR Glossary
Service Interface	see [2] AUTOSAR Glossary
Service Discovery	see [2] AUTOSAR Glossary

Table 2.2: Glossary-defined Technical Terms

3 Further applicable specification

3.1 Input documents & related standards and norms

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

[1] Specification of Communication Management
AUTOSAR_AP_SWS_CommunicationManagement

[2] Glossary
AUTOSAR_FO_TR_Glossary

[3] Specification of Execution Management
AUTOSAR_AP_SWS_ExecutionManagement

[4] Specification of Platform Health Management
AUTOSAR_AP_SWS_PlatformHealthManagement

[5] Specification of Network Management
AUTOSAR_AP_SWS_NetworkManagement

[6] Specification of Diagnostics
AUTOSAR_AP_SWS_Diagnostics

[7] Specification of Update and Configuration Management
AUTOSAR_AP_SWS_UpdateAndConfigurationManagement

[8] Specification of Manifest
AUTOSAR_AP_TPS_ManifestSpecification

[9] Explanation of Adaptive Platform Software Architecture
AUTOSAR_AP_EXP_SWArchitecture

[10] Requirements of State Management
AUTOSAR_AP_RS_StateManagement

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:

- Section 7.2 This document will show the basic principles of the intended functionality of State Management. To enable State Management to be portable, in future versions of this document standardized fields and values shall be introduced.
- Section 7.4 Communication Control for Diagnostic reasons this is not yet discussed with Diagnostic Management.
- Section 7.11 The introduced StateMachine feature does not yet cover how the DiagnosticReset requests from Diagnostic Management will be handled. This fact will be improved in R24-11 when the StateMachine approach is stabilized.
- Section 7.2.2 is referencing the feature CommunicationGroups, which is removed from AP-SWS-CommunicationManagement with R23-11 release. Up to now no replacement is available. The section is kept intentionally to remember to bring back this feature in upcoming release.

4.2 Applicability to car domains

If a superior State Management instance to the one from the ECU is available in a hierarchical car context, the State Management of the ECU shall also evaluate events generated by the superior instance of State Management. Section 7.8 will give further details.

5 Dependencies to other Functional Clusters

This chapter provides an overview of the dependencies to other Functional Clusters in the AUTOSAR Adaptive Platform. Section 5.1 "Provided Interfaces" lists the interfaces provided by State Management to other Functional Clusters. Section 5.2 "Required Interfaces" lists the interfaces required by State Management.

A detailed technical architecture documentation of the AUTOSAR Adaptive Platform is provided in [9].

5.1 Provided Interfaces

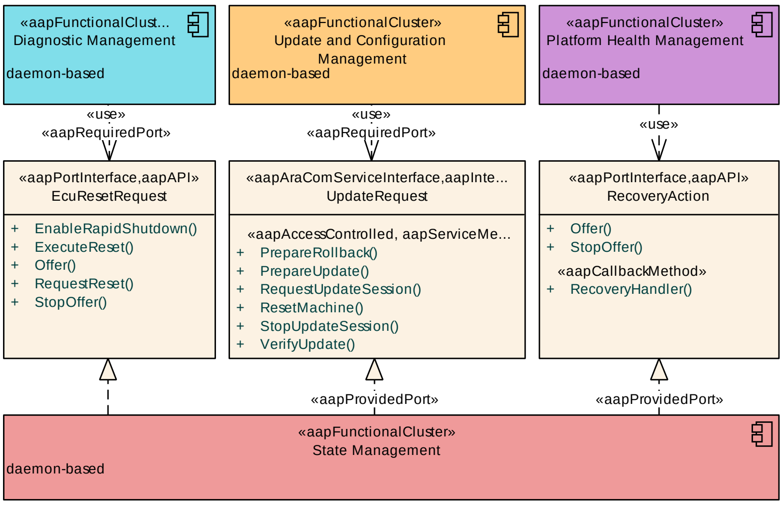


Figure 5.1: Interfaces provided by State Management to other Functional Clusters

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*
EcuResetRequest	Diagnostic Management	This interface is used to handle reset requests.
RecoveryAction	Platform Health Management	Platform Health Management uses this interface to trigger failure recovery.
UpdateRequest	Update and Configuration Management	This interface is used to interact with State Management of the Adaptive Platform during an update.

Table 5.1: Interfaces provided to other Functional Clusters

5.2 Required Interfaces

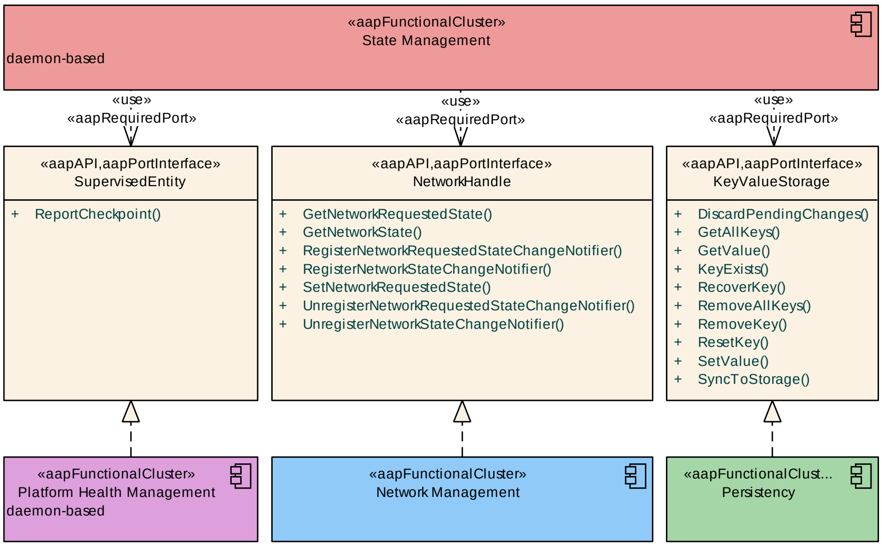


Figure 5.2: Interfaces required by State Management from other Functional Clusters

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

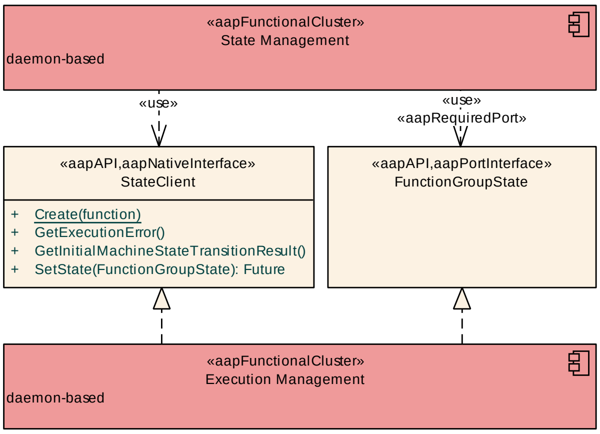


Figure 5.3: Interfaces required by State Management from Execution Management

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

Functional Cluster	***Interface***	***Purpose***
Execution Management	ExecutionClient	This interface shall be used to report the state of the `State Management` process(es).
Execution Management	FunctionGroupState	This interface shall be used to request FunctionGroupState transitions and to check their status.

▽

△

Functional Cluster	***Interface***	***Purpose***
Execution Management	StateClient	This interface shall be used to request FunctionGroupState transitions.
Log and Trace	Logger	`State Management` shall use this interface to log standardized messages.
Network Management	NetworkHandle	This interface shall be used to retrieve information about the network status of a `NetworkHandle`.
Persistency	KeyValueStorageOperations	This interface should be used to persist information (e.g. update session).
Persistency	KeyValueStorage	This interface should be used to persist information (e.g. update session).
Platform Health Management	SupervisedEntity	`State Management` shall use this interface to enable supervision of its process(es) by `Platform Health Management`.

Table 5.2: Interfaces required from other Functional Clusters

6 Requirements Tracing

The following tables reference the requirements specified in [10] 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_91004] [SWS_SM_91007] [SWS_SM_91008] [SWS_SM_91009] [SWS_SM_91017]
[RS_AP_00119]	Return values / application errors.	[SWS_SM_91010] [SWS_SM_91017]
[RS_AP_00120]	Method and Function names.	[SWS_SM_91017]
[RS_AP_00121]	Parameter names.	[SWS_SM_91017]
[RS_AP_00122]	Type names.	[SWS_SM_91018] [SWS_SM_91019]
[RS_AP_00125]	Enumerator and constant names.	[SWS_SM_91010]
[RS_AP_00142]	Handling of unsuccessful operations.	[SWS_SM_91010] [SWS_SM_91017]
[RS_AP_00149]	Guidance on error handling.	[SWS_SM_91010]
[RS_AP_00150]	Provide only interfaces that are intended to be used by AUTOSAR applications and other Functional Clusters.	[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_91016] [SWS_SM_91017] [SWS_SM_91018] [SWS_SM_91019] [SWS_SM_91021] [SWS_SM_91023] [SWS_SM_91024]
[RS_SM_00001]	`State Management` shall coordinate and control multiple sets of `Applications`.	[SWS_SM_00001] [SWS_SM_00005] [SWS_SM_00006] [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_00632] [SWS_SM_00633] [SWS_SM_00634] [SWS_SM_00635] [SWS_SM_00636] [SWS_SM_00637] [SWS_SM_00638] [SWS_SM_00639] [SWS_SM_00640] [SWS_SM_00641] [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_91016] [SWS_SM_91017] [SWS_SM_91021] [SWS_SM_91022] [SWS_SM_91023] [SWS_SM_91024] [SWS_SM_91025] [SWS_SM_91026] [SWS_SM_CONSTR_00001] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SWS_SM_CONSTR_00020] [SWS_SM_CONSTR_00021] [SWS_SM_CONSTR_00022] [SWS_SM_CONSTR_00023]
[RS_SM_00004]	`State Management` shall provide standardized interfaces.	[SWS_SM_00020] [SWS_SM_00021] [SWS_SM_00202] [SWS_SM_00204] [SWS_SM_00205] [SWS_SM_00206] [SWS_SM_00207] [SWS_SM_00208] [SWS_SM_00209] [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_91016] [SWS_SM_91017] [SWS_SM_91018] [SWS_SM_91019] [SWS_SM_91021] [SWS_SM_91022] [SWS_SM_91023] [SWS_SM_91024] [SWS_SM_91025] [SWS_SM_91026]
[RS_SM_00005]	`State Management` internal states.	[SWS_SM_00020] [SWS_SM_00021] [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_00632] [SWS_SM_00633] [SWS_SM_00634] [SWS_SM_00635] [SWS_SM_00636] [SWS_SM_00637] [SWS_SM_00638] [SWS_SM_00639] [SWS_SM_00640] [SWS_SM_00641] [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_91001] [SWS_SM_91002] [SWS_SM_91003] [SWS_SM_91007] [SWS_SM_91008] [SWS_SM_91009] [SWS_SM_CONSTR_00020] [SWS_SM_CONSTR_00021] [SWS_SM_CONSTR_00022] [SWS_SM_CONSTR_00023]
[RS_SM_00100]	`State Management` shall support ECU reset	[SWS_SM_00101] [SWS_SM_00106] [SWS_SM_00107] [SWS_SM_00203]
[RS_SM_00200]	`State Management` shall provide an interface between `State Management` instances.	[SWS_SM_00500] [SWS_SM_00501]
[RS_SM_00300]	`State Management` shall support variant handling based on calibration data.	[SWS_SM_00005] [SWS_SM_00006]

▽

△

Requirement	Description	Satisfied by
[RS_SM_00400]	State Management shall establish communication paths dynamically.	[SWS_SM_00300] [SWS_SM_00301] [SWS_SM_00303] [SWS_SM_00304] [SWS_SM_91004]
[RS_SM_00401]	State Management shall control Applications depending on dynamic communication paths .	[SWS_SM_00302] [SWS_SM_00620] [SWS_SM_00621] [SWS_SM_00625] [SWS_SM_00626]

Table 6.1: RequirementsTracing

7 Functional specification

Please note that the semantics in the following chapter is not yet fully specified.

State Management is a functional cluster contained in the Adaptive Platform Services. State Management is responsible for all aspects of Operational State Management including handling of incoming events, prioritization of these events/requests setting the corresponding internal States. Incoming events are issued when AUTOSAR Adaptive Platform or Adaptive Applications which are configured to have write access permissions change the value of "Trigger" fields provided by State Management. 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 any diagnostic or update session is active as part of State Managements internal State. State Management supervises the shutdown prevention with a project-specific timeout.

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. Network Management provides ara::com fields (NetworkHandle) where each of the fields represents a set of (partial) networks. State Management can influence these fields in dependency of Function Groups states and - vice versa - can set Function Groups to a defined state depending on the value of Network Managements NetworkHandle fields.

Adaptive Applications and AUTOSAR Adaptive Platform Applications can register to the events of the "Notifier" fields provided by State Management. They can change their internal behavior based on the value provided in the fields. Adaptive Applications and AUTOSAR Adaptive Platform Applications can influence the internal States of State Management by writing to the "Trigger" fields provided by State Management.

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 Applications.
- Section 7.2 describes how Adaptive Applications and AUTOSAR Adaptive Platform Applications could be influenced in their behavior based on provided "Notifier" fields of State Management and how they can influence the internal states of State Management by using provided "Trigger" fields.
- Section 7.4 covers several topics related to Diagnostic Management including execution of different reset types

- Section 7.5 describes how Update and Configuration Management interacts with State Management
- Section 7.6 documents support provided by Network Management to de-/activate (partial) networks in dependency of Function Group States and vice versa.
- Section 7.7 describes how Execution Management is used to change Function Group State or Machine State.
- Section 7.8 provides an introduction to how State Management will work within a virtualized/hierarchical environment.

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.

State Management is the central point where any operation event is received that might have an influence to the internal States of State Management. The State Management is responsible to evaluate these events and decide based on

- Event type (defined in project specific implementation based on project specific requirements).
- Event priority (defined in project specific implementation based on project specific requirements).
- Application identifier (Application identifier is not supported in this release. It is under discussion with FT-SEC if such an identifier could be provided by Identity and Access Management).

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

The state change request for Function Groups can be issued by several AUTOSAR Adaptive Platform Applications:

- Platform Health Management to trigger error recovery, e.g. to activate fallback Functionality.
- Diagnostic Management, to switch the system into different diagnostic states and to issue resets of the system.
- 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 NetworkHandle fields, where State Management registers to and reacts on changes of these fields issued by Network Management.

The final decision if any effect is performed is taken by State Managements internal logic based on project-specific requirements.

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

- State Management shall be monitored and supervised by Platform Health Management.

- State Management provides ara::com fields as interface to provide information about its current internal States

State Management is responsible for handling the following states:

- Machine State see 7.1.1
- Function Group State see 7.1.2

7.1.1 Machine State

A Machine State is a specific type of Function Group State (see 7.1.2). Machine States and all other Function Group States are determined and requested by the State Management functional cluster, see 7.1.3. The set of active States is significantly influenced by vehicle-wide 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 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, SM, 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.

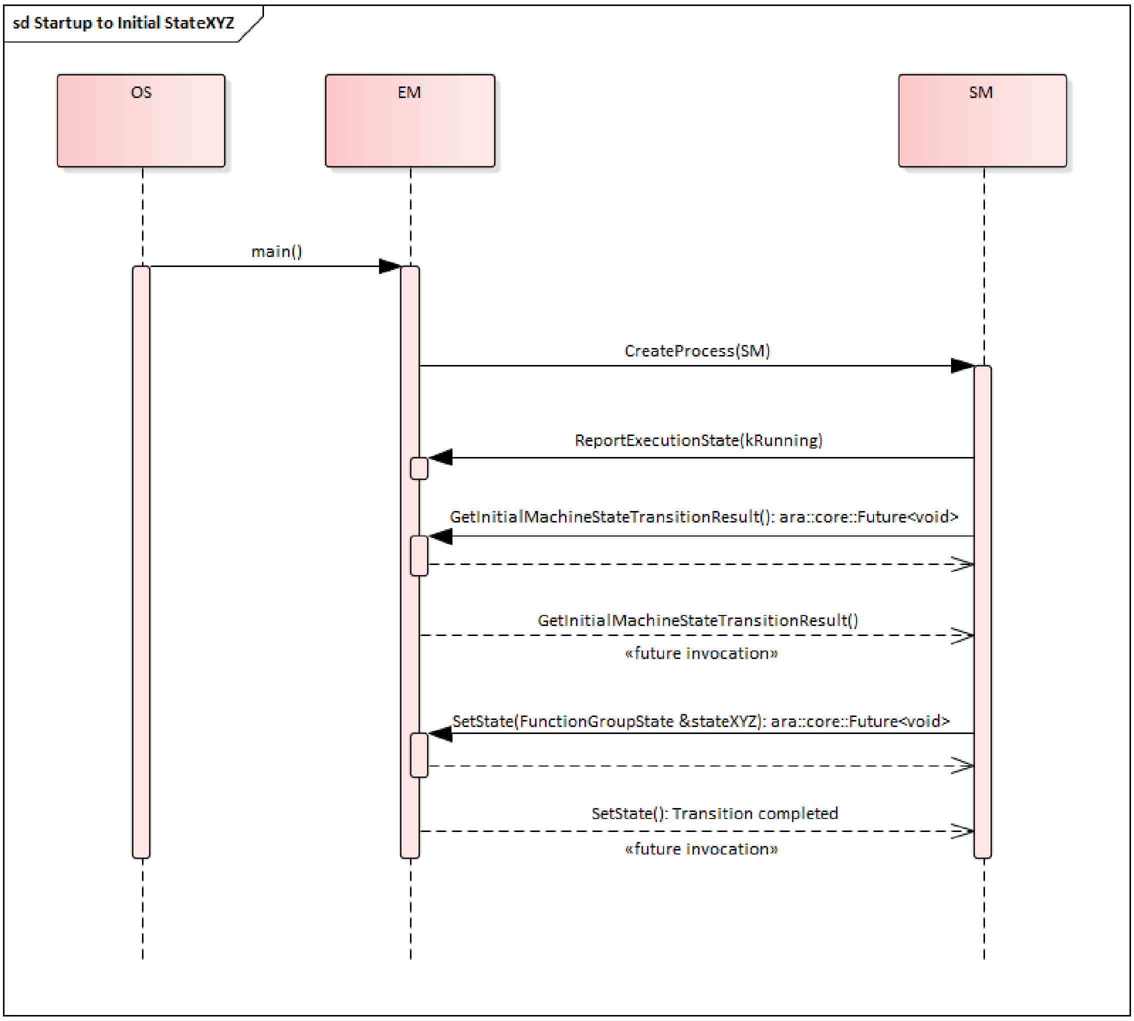

sd Startup to Initial StateXYZ
OS
EM
SM
main()
CreateProcess(SM)
ReportExecutionState(kRunning)
GetInitialMachineStateTransitionResult(): ara::core::Future<void>
GetInitialMachineStateTransitionResult()
«future invocation»
SetState(FunctionGroupState &stateXYZ): ara::core::Future<void>
SetState(): Transition completed
«future invocation»

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`.

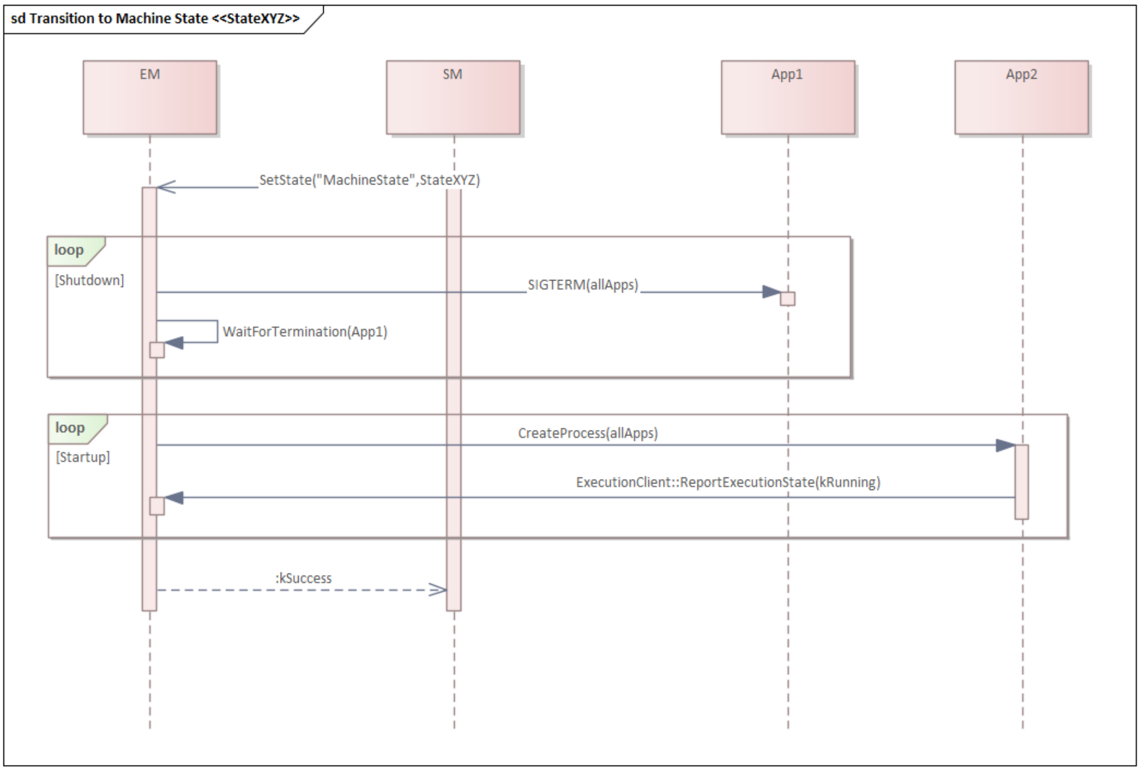

sd Transition to Machine State <<StateXYZ>>
EM
SM
App1
App2
SetState("MachineState",StateXYZ)
loop
[Shutdown]
SIGTERM(allApps)
WaitForTermination(App1)
loop
[Startup]
CreateProcess(allApps)
ExecutionClient::ReportExecutionState(kRunning)
:kSuccess

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

7.1.1.1 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.

7.1.1.2 Shutdown

As mentioned in 7.1.1.1 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.

7.1.1.3 Restart

As mentioned in 7.1.1.1 AUTOSAR assumes that State Management will be configured to run in Restart. The reasons for doing so are the same as for 7.1.1.2.

7.1.2 Function Group State

If more than one group of functionally coherent 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 diagnostic and 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 Applications.

[SWS_SM_00001]{DRAFT} **Available Function Group (states)** ⌈State Management shall obtain available Function Groups and their potential states from the Machine Manifest to set-up the Function Group specific state management.⌋ *(RS_SM_00001)*

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 [8], 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).

The system might contain calibration data for variant handling. This might include that some of the Function Groups configured in the Machine Manifest are not intended to be executed on this system. therefore State Management has to evaluate calibration data to gather information about Function Groups not configured for the system variant

[SWS_SM_00005]{DRAFT} **Function Group Calibration Support** ⌈State Management shall receive information about deactivated Function Groups from calibration data.⌋*(RS_SM_00001, RS_SM_00300)*

The storage and reception of calibration data is implementation specific.

[SWS_SM_00006]{DRAFT} **Function Group Calibration Support** ⌈State Management shall decline the request of Adaptive Applications and AUTOSAR Adaptive Platform Applications to change the Function Group State of a Function Group which is not configured to run in this variant.⌋*(RS_SM_00001, RS_SM_00300)*

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 Platforms 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.

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

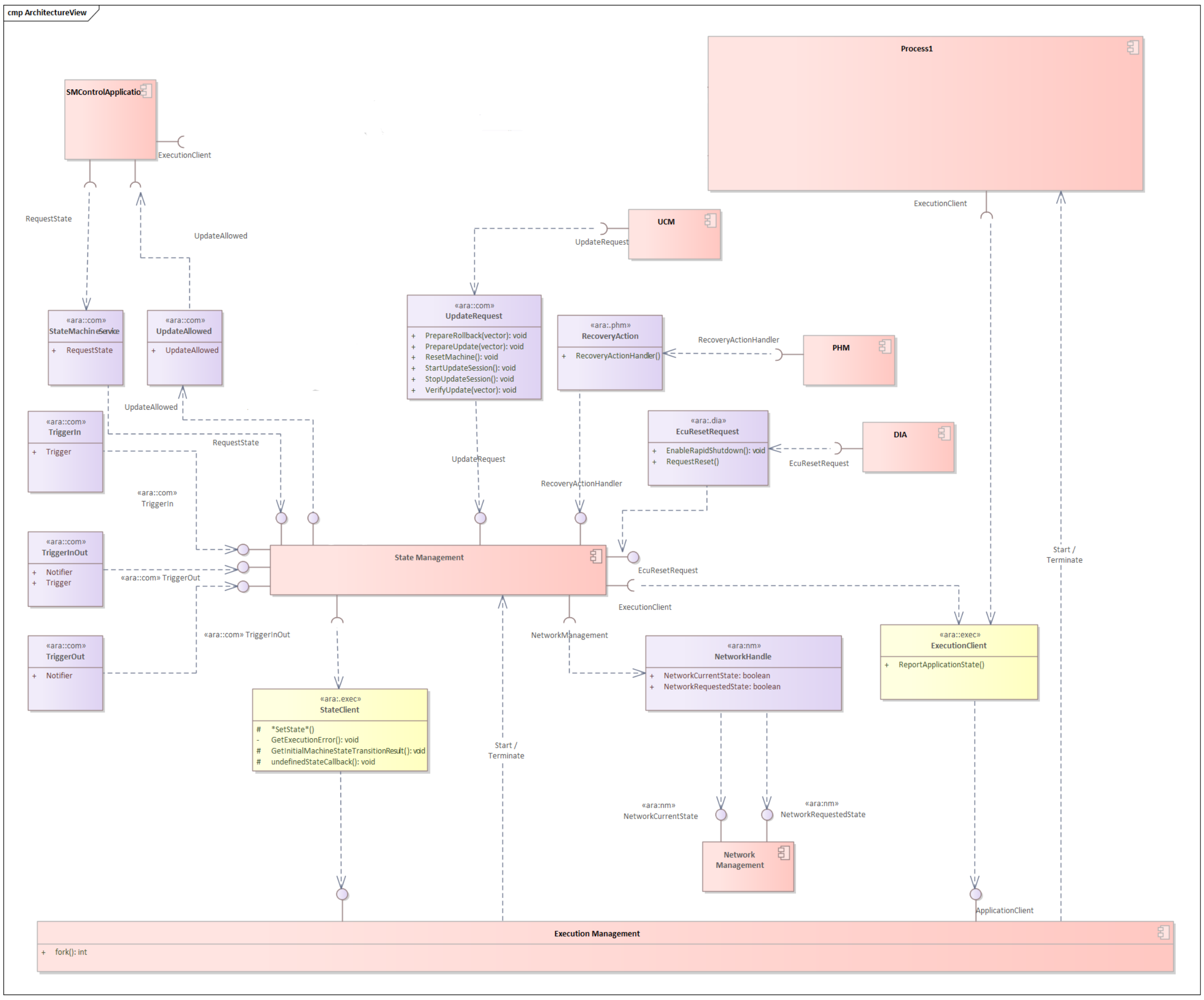


Figure 7.3: State Management Architecture

7.2 State Management and Adaptive (Platform) Applications

7.2.1 Interaction between the SM and Adaptive Applications

Some Adaptive Applications, including AUTOSAR Adaptive Platform Applications, might have the need to interact with State Management. Therefor State Management provides a service interface TriggerOut with a Notifier (see section 9.2.2) field, where each Adaptive Application can subscribe to, thus it is informed whenever a State Managements internal State changes. When an Adaptive Application recognizes the change it can carry out the appropriate action.

In the opposite way each Adaptive Application can influence the behavior of State Management by writing to the Trigger fields provided (as part of the service interface TriggerIn) by State Management. Therefore the Adaptive Application has to by configured in a way that write access to State Managements fields is granted. If the StateMachine approach is used, the Adaptive Applications can influence the behavior of State Management via the RequestState method (as part of the interface StateMachineService).

State Management provides a third service interface(TriggerInOut), where both fields are available: Trigger and Notifier. This combined field is provided with the intention that whenever the Trigger field changes the Notifier field changes as well after State Management has carried out its operation issued by the Trigger change.

Please be aware, that this interface is not compatible with the StateMachine approach and shall not be used in combination with the RequestState method from the StateMachineService interface

[SWS_SM_00020] InternalState Propagation ⌈State Management shall support implementation of multiple instances of TriggerOut with a Notifier field which reflect State Managements internal states thus Application can get State Managements states.⌋*(RS_SM_00004, RS_SM_00005)*

[SWS_SM_00021] InternalState Influence ⌈State Management shall support implementation of multiple instances of TriggerIn with a Trigger field which affect State Managements internal states thus Application can influence State Managements states.⌋*(RS_SM_00004, RS_SM_00005)*

Please note that the types (and therefore the content) of the provided fields are project-specific.

An overview of the interaction of State Management and Adaptive Applications for a non-synchronized behavior is shown in Figure 7.4.

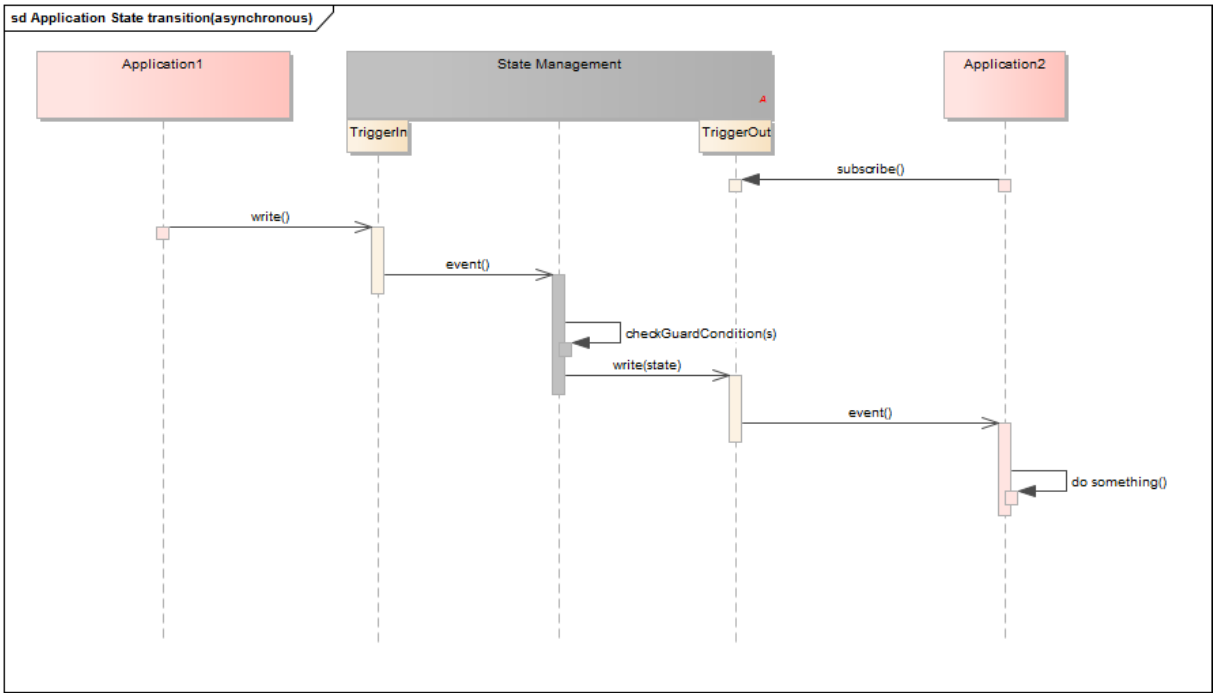


Figure 7.4: Non-Synchronized Application State handling

7.2.2 Synchronization across multiple Adaptive Applications

Removal of CommunicationGroups from AP-SWS-CommunicationManagement

The informations from this chapter are just kept to remember to bring back this feature in upcoming release. The CommunicationGroup approch is removed from AP-SWS-CommunicationManagement with R23-11 release and up to now no replacement is available

Some scenarios in AUTOSAR Adaptive Platform might require a more sophisticated handling, where a change in State Managements internal state could only be finally carried out, when related Modelled Processes have entered a dedicated 'State', which is triggered by State Management.

These triggers will be probably dedicated to a different set of Processes, depending on the functionality to be achieved. State Management sees currently two different use-cases:

- addressing all running Modelled Processes in a machine for PowerModes
- addressing running Modelled Processes for diagnostic reset reasons.

To have the possibility and flexibility to address different groups of Modelled Processes a new communication pattern called CommunicationGroups (see SWS-CommunicationManagement [1]) was introduced.

This pattern defines a kind of compound service with a proxy and a skeleton for the server as well as for the clients.

With this approach a server can:

- broadcast a message to all clients in the group
- send a message to a dedicated client in the group
- can get a list of all clients in the group
- receive the replies from all clients in the group

Conclusively a client can

- receive messages from the server
- send a reply to the server

Please note that it is essential, that a client replies to each server request, independently if the request could be fulfilled by the client or not.

To have a unique understanding of the messages and replies these will be defined as a template and the tooling will generate corresponding proxies and skeletons.(for details see SWS-CommunicationManagement)

So now State Management as a server of (multiple) CommunicationGroups can send a message to all the clients in a group and can check if

- all clients answered the request
- all clients sent the expected answer

If any of the clients did not answer or did not reply with the expected answer State Management can retry to achieve the requested state by addressing the misbehaving client directly. When the client still does not answer(or does not answer with expected reply) State Management can do further project-specific actions. Due to the asynchronous nature of CommunicationGroups it is necessary that State Management supervises the reception of the answers from all clients with a project-specific timeout.

An overview of the interaction of State Management and Adaptive Applications for a synchronized behavior is shown in Figure 7.5.

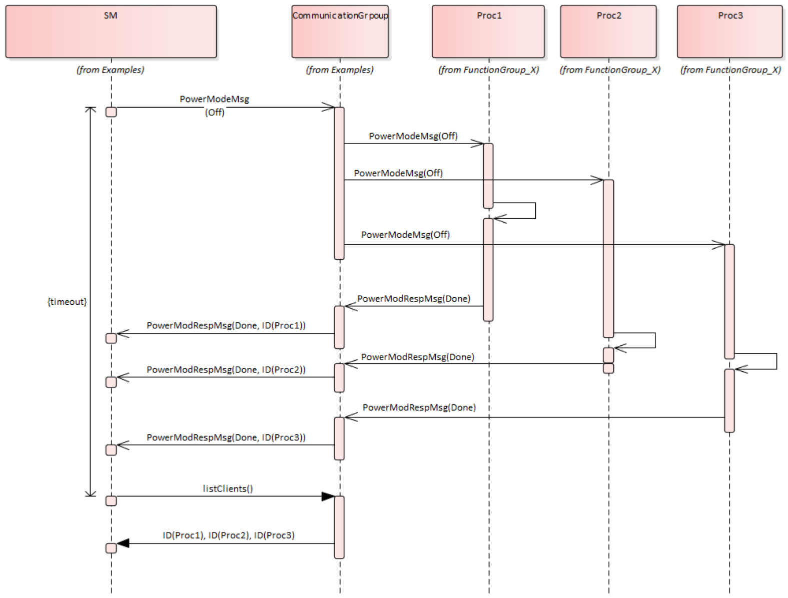


Figure 7.5: PowerModes as example of Synchronized Application State handling

7.2.2.1 PowerModes for Adaptive (Platform) Applications

The PowerModes are intended to influence the internal behavior of all Processes in the system. Currently, there are three modes supported, but there might be more modes introduced in future releases of this document.

The modes are defined as follows:

- "On" : A Modelled Process that receives this PowerMode behaves normally as it has been spawned by ExecutionManagement. It is used to "undo" the other PowerMode requests. Modelled Processes that are just spawned should behave like an "On" is requested as PowerMode.
- "Suspend" : This PowerMode is intended to be used as a signal to the Modelled Processes that the system is suspended(e.g. to RAM or to disc). The implementation of the necessary actions(e.g. setting drivers to a proprietary mode, ...) will be project-specific and might depend on the environment(e.g. used OS).
- "Off" : A Modelled Process that receives this PowerMode behaves like it receives a SIGTERM from Execution Management, beside exiting.

This PowerMode is used to realize the so called "late-wakeup", where a new wakeup reason is found during a proceeding shutdown(e.g. short-time low voltage). When the new wakeup reason is found an "On" request will be sent to the Modelled Processes, thus they can immediately continue with their "normal" work without the need to be spawned again(e.g. from the filesystem). A Modelled Process which has just received the "Off" PowerMode (and carried out the necessary actions) and receives a SIGTERM from Execution Management afterward, can perform its shutdown much faster because it has already done all the necessary steps to be prepared for exiting.

Modelled Processes that support the PowerModes are expected to behave like they would have received an "On" request when they are entering "Running" state when being spawned by Execution Management to keep compatibility with Modelled Processes which do not support the PowerModes.

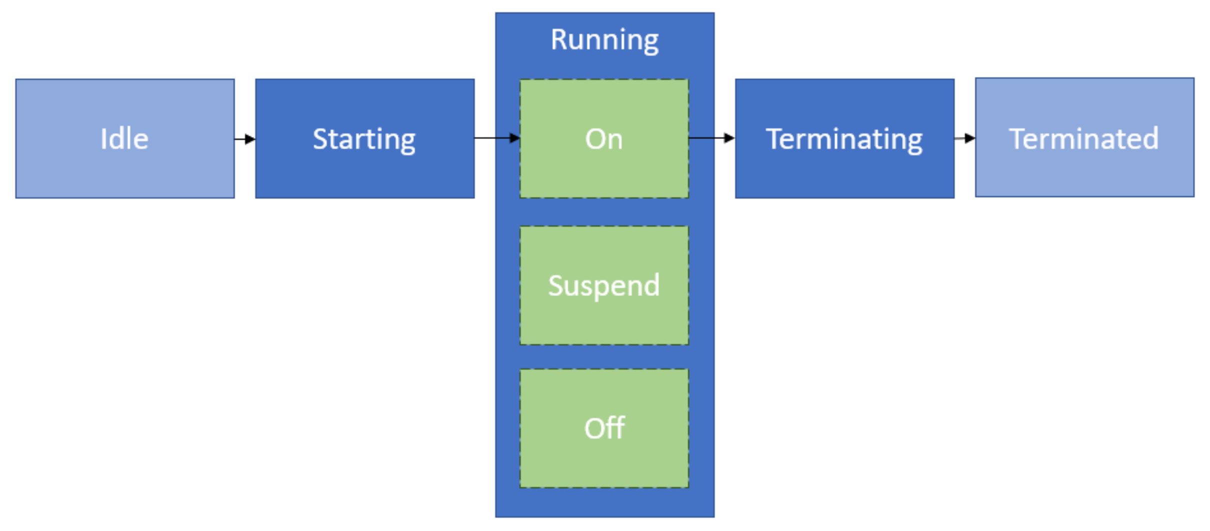


Figure 7.6: PowerModes for Adaptive (Platform) Applications

Please note that Modelled Processes that support either "Off" or "Suspend" or both of these PowerModes support the "On" PowerMode, too.

The service interface for the PowerMode, the defined messages and replies are removed in this release.

7.2.2.2 Diagnostic Reset for Adaptive (Platform) Applications

The Diagnostic Reset Service is provided for Diagnostic Reset functionality of Diagnostic Management. The rationale behind this is to change the behavior of Modelled Processes without the need to terminate and restart them. This service is intended to influence Modelled Processes that are addressed by Diagnostic Address. If all Modelled Processes or only a subset is affected depends on the system design. Therefore it is recommended to limit access to the service by IAM.

The reaction of the Adaptive (Platform) Applications to the request itself is project-specific.

Details for the complete interaction of Diagnostic Management and State Management can be found in 7.4 Interaction with Diagnostic Management.

The service interface for the Diagnostic Reset, the defined messages, and replies are removed in this release.

Please note that this interface just provides means to the developer of State Management to realize the project-specific needs for Diagnostic Reset use cases.

7.3 Interaction with Platform Health Management

Platform Health Management is responsible for monitoring supervised entities via local supervision(s) and checking the status of health channels. Failures in local super- vision(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[4]. As soon as a global supervision enters the stopped state or a health channel contains information that is relevant for State Management, Platform Health Management will notify State Management via C++ API provided by Platform Health Manager. C++ interface is provided as a class with virtual functions, which have to be implemented by State Management.

When State Management receives notification from Platform Health Management it can evaluate the information from the notification and initiate the project-specific actions to recover from the failure(e.g. request Execution Management to switch a Function Group to another Function Group State, request Execution Management for a restart of the Machine, ...).

Note: Platform Health Management monitors the return of the RecoverHandler() with a configurable timeout. If after a configurable amount of retries the State Management will still not regularly return from the RecoveryHandler() Platform Health

Management will do its own countermeasures by wrongly triggering or stop triggering the serviced watchdog.

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

7.4 Interaction with Diagnostic Management

Diagnostic Management is responsible for diagnosing, configuring and resetting Diagnostic Addresses. The relation between a Diagnostic Addresses and a Software Cluster is project specific. The interface between Diagnostic Management and State Management is provided by Diagnostic Management as C++ API. The interface is provided as a class with virtual functions, which have to be implemented by State Management.

Diagnostic Management provides the ara::diag::EcuResetRequest interface to forward ECU Reset service requests to State Management. State Management processes the request and executes the reset of the Diagnostic Address related entity.

From Diagnostic Management point of view several different reset types have to be carried out to fulfill functionality of Diagnostic Management. Because the interpretation of the reset types (defined in ISO 14229-1)

- hardReset
- keyOffOnReset
- softReset
- customReset

is done differently by each OEM, parts of the reset functionality have to be delegated by State Management to Adaptive Applications and AUTOSAR Adaptive Platform Applications.

A "keyOffOnReset" may be translated by State Managements internal logic to stop and start the Function Group which relate to the requested Diagnostic Addresses.

A "softReset" may be translated by State Managements internal logic to request Modelled Processes (within the Function Groups which relate to the requested Diagnostic Address) to perform internal functionality without the need to terminate and start them again. Therefor State Management provides a service interface in the scope of a CommunicationGroup. All Modelled Processes which should support this feature have to use the ara::com methods and fields generated from the message and reply message definition which is removed in this release.

[SWS_SM_00101]{DRAFT} **Diagnostic Reset** ⌈State Management shall implement means to receive reset requests for Diagnostic Addresses from Diagnostic Management. State Management shall carry out the project specific actions for the specific reset type.⌋*(RS_SM_00100)*

This functionality is project specific. So therefore the correct mapping has to be done by the project specific code.

When State Management does not see any reason(project specific) to keep the machine alive any longer it will normally not shutdown the machine immediately, but will keep it alive for a configurable amount of time. Under some conditions it is needed that this waitingtime is reduced as much as possible (e.g. end of line diagnostics). This has to be supported by State Management too.

[SWS_SM_00106]{DRAFT} **Enabling of rapid shutdown** ⌈State Management shall implement means to reduce the waitingtime to shutdown the machine as much as possible⌋*(RS_SM_00100)*

There might be reasons that Diagnostic Management needs to withdraw a previously enabled rapid shutdown. This usecase has to be supported by State Management too.

[SWS_SM_00107]{DRAFT} **Disabling of rapid shutdown** ⌈State Management shall implement means to set the waitingtime to shutdown the machine to the configured value⌋*(RS_SM_00100)*

7.5 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 9.2.4) 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** ⌈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.⌋*(RS_SM_00100)*

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.

[SWS_SM_00209]{DRAFT} **Preventing multiple update sessions** ⌈RequestUpdateSession shall return kNotAllowedMultipleUpdateSessions in case the method RequestUpdateSession is called during an already active Update Session⌋ *(RS_SM_00004)*

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

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

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

[SWS_SM_00204]{DRAFT} **Persist session status** ⌈State Management shall persist information about ongoing update session, thus it can be read out after any kind of Machine reset.⌋*(RS_SM_00004)*

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** ⌈State Management shall implement the service interface UpdateRequest to Update and Configuration Management with the method call ResetMachine to request a Machine reset.⌋*(RS_SM_00004)*

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 ⌈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.⌋*(RS_SM_00004)*

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 ⌈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.⌋*(RS_SM_00004)*

[SWS_SM_00207] prepare verify ⌈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.⌋*(RS_SM_00004)*

[SWS_SM_00208] prepare rollback ⌈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.⌋*(RS_SM_00004)*

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.

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.

When a Software Cluster is removed by Update and Configuration Management, VerifyUpdate will never be called by Update and Configuration Management. Contrary to that PrepareUpdate will never be called, when a new Software Cluster is installed into the Machine.

For more detail about the update process see sequence diagrams and descriptions in [7].

7.6 Interaction with Network Management

Please be aware, that the following section is not compatible with the StateMachine approach and it will be changed therefore in future releases.

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 service interface.

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

The NetworkHandles are defined in the Machine Manifest and are there assigned to a Function Group State.

An overview of the interaction of State Management, Network Management and Adaptive Applications is shown in Figure 7.8.

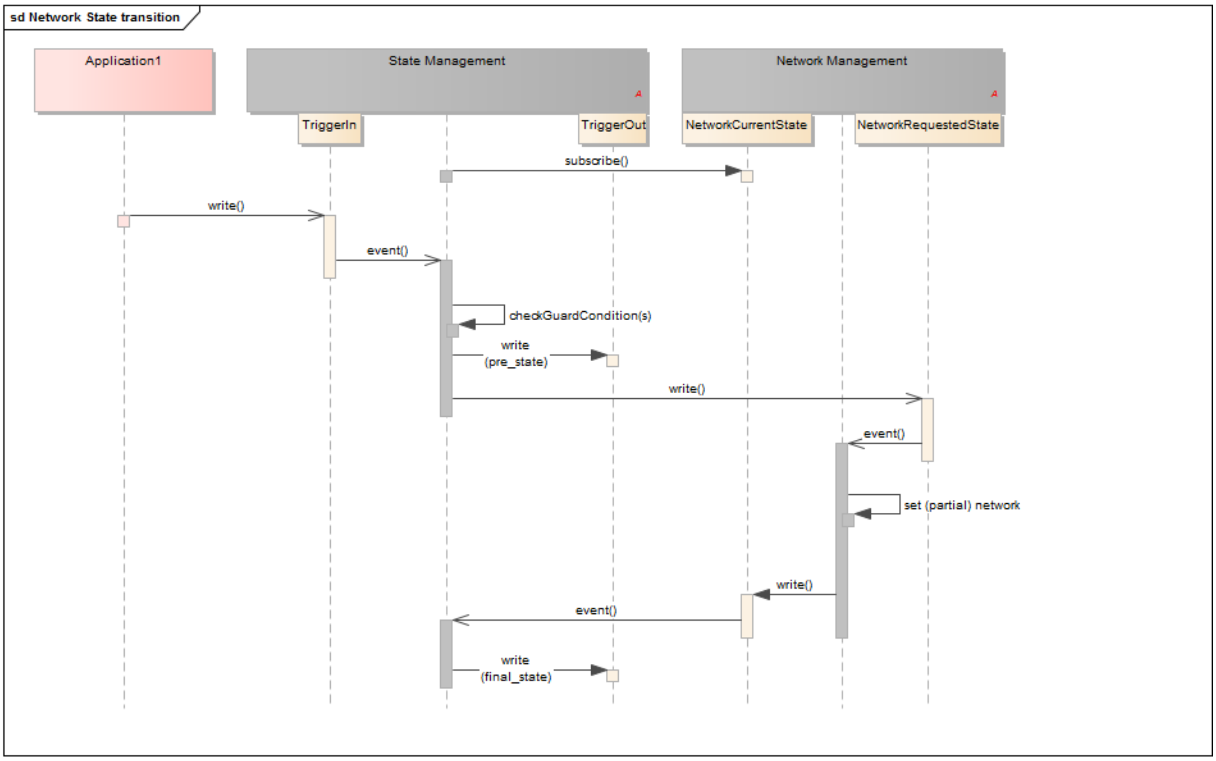


Figure 7.7: Switching Network State by "Trigger"

[SWS_SM_00300]{DRAFT} **NetworkHandle Configuration** ⌈State Management shall receive information about NetworkHandles and their associated Function Group States from Machine Manifest.⌋*(RS_SM_00400)*

Whenever (partial) networks are activated or deactivated from outside request and this set of (partial) networks is represented by a NetworkHandle in Machine Manifest Network Management will change the value of the corresponding NetworkHandle. State Management is notified about the change, because it has registered to all availabe NetworkHandle fields. When State Management recognizes a change in a fields value it sets the corresponding Function Group in the Function Group State where the NetworkHandle is configured for in the Machine Manifest.

[SWS_SM_00301]{DRAFT} **NetworkHandle Registration** ⌈State Management shall register for all NetworkHandles provided by Network Managements which are available from Machine Manifest.⌋*(RS_SM_00400)*

[SWS_SM_00302]{DRAFT} **NetworkHandle to FunctionGroupState** ⌈State Management shall set Function Groups to the corresponding Function Group State which is configured in the Machine Manifest for the NetworkHandle when it recognizes a change in NetworkHandle value.⌋*(RS_SM_00401)*

Vice versa State Managements shall change the value of the NetworkHandle when a Function Group has to change its Function Group State and an association between this Function Group State and the Network handle is available in Machine Manifest. Network Management will recognize this change and will change the state of the (partial) networks accordingly to the NetworkHandle.

[SWS_SM_00303]{DRAFT} **FunctionGroupState to NetworkHandle** ⌈State Management shall change the value of NetworkHandle when Function Groups changes its Function Group State and a NetworkHandle is associated to this Function Group State in the Machine Manifest.⌋*(RS_SM_00400)*

It might be needed that a Function Group stays longer in its Function Group State when the causing (partial) network set has been switched off or a (partial) network is longer available than the causing Function Group has been switched to Function Group State 'Off'. This is called 'afterrun'. The corresponding timeout-value has to be configured in Machine Manifest

[SWS_SM_00304]{DRAFT} **Network Afterrun** ⌈State Management shall support means to support 'afterrun' to switch off related Function Groups or (partial) networks. The timeout value for this 'afterrun' has to be read from e.g. Machine Manifest.⌋*(RS_SM_00400)*

7.7 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 Managements States (or other project specific requirements) or might be requested at State Management from an external Adaptive Application.

An overview of the interaction of State Management, Execution Management and Adaptive Applications is shown in Figure 7.8.

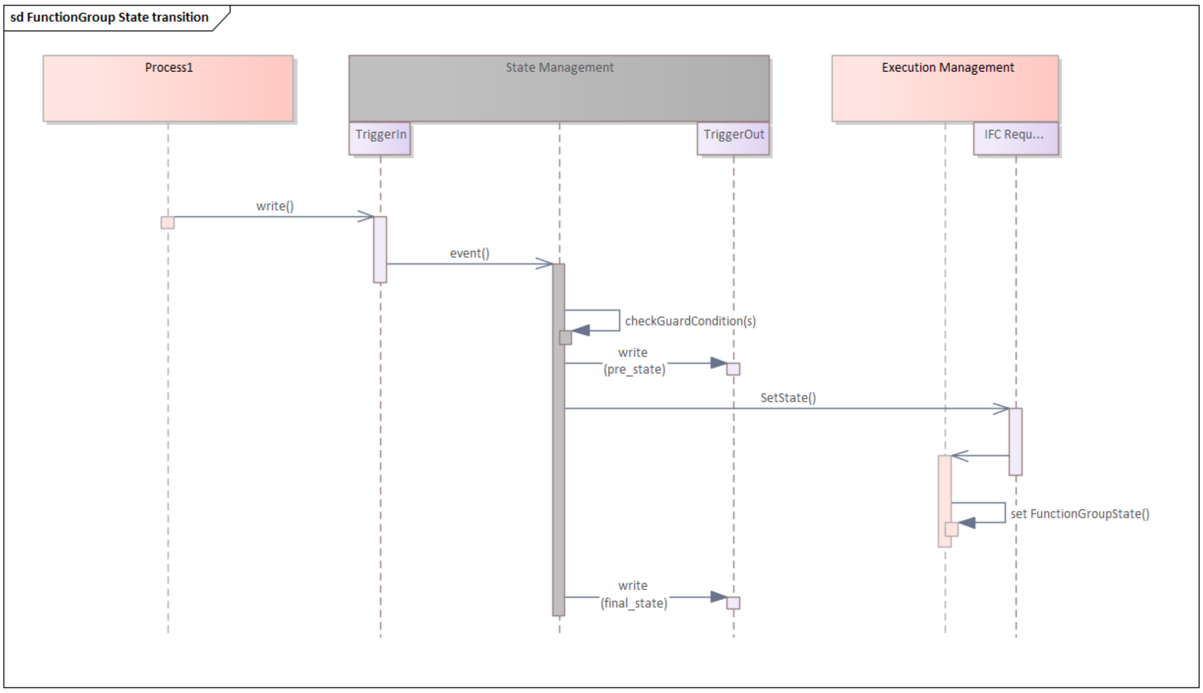


Figure 7.8: Switching FunctionGroup State by "Trigger"

[SWS_SM_00400] Execution Management ⌈State Management shall use State-Client API of Execution Management to request a change in the Function Group State of any Function Group(including Machine State).⌋*(RS_SM_00001)*

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 ⌈State Management shall evaluate the results of request to Execution Management. Based on the results State Management may do project-specific actions⌋*(RS_SM_00001)*

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
- ignore the error e.g. kInTransitionToSameState, kAlreadyInState
- 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 State Machines are 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.

7.8 State Management in a virtualized/hierarchical environment

On an ECU several machines might run in a virtualized environment. Each of the virtual machines might contain an AUTOSAR Adaptive platform. So therefore each of the virtual machines contain State Management. To have coordinated control over the several virtual machines there has to be virtual machine which supervises the whole ECU state. This is not only valid for a virtualized environment, but for a hierarchical environment, too.

[SWS_SM_00500]{DRAFT} **Virtualized/hierarchical State Management** ⌈State Management shall be able to register to the "Trigger" fields of a supervising State Management instance to receive information about the whole ECU state.⌋*(RS_SM_00200)*

[SWS_SM_00501]{DRAFT} **Virtualized/hierarchical State Management internal State** ⌈State Management shall implement means to calculate its internal States based on information from a supervising State Management instance.⌋*(RS_SM_00200)*

7.9 StateManagement lifecyle

7.9.1 Startup

State management lifecycle fully depends on machine state. Details can be found in 7.1.1.1

7.9.2 Shutdown

State management lifecycle fully depends on machine state. Details can be found in 7.1.1.2

7.9.3 Restart

State management lifecycle fully depends on machine state. Details can be found in 7.1.1.3

7.10 Configuration

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]{DRAFT} **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.⌋*(RS_SM_00001)*

7.11 StateManagement StateMachine

7.11.1 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, Final State, PrepareUpdate, VerifyUpdate and PrepareRollback. There probably will be a number of additional project-specific StateMachine States (e.g. degraded States). Each State references an ActionList, which is comprised of a set of ActionListItems. All ActionListItems in an ActionList are executed as soon as a StateMachine State of a StateMachine is entered. Currently available Types for an ActionListItem are:

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

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

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

Current StateMachine State can be published by TriggerOut::Notifier interface which is configurable.

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:

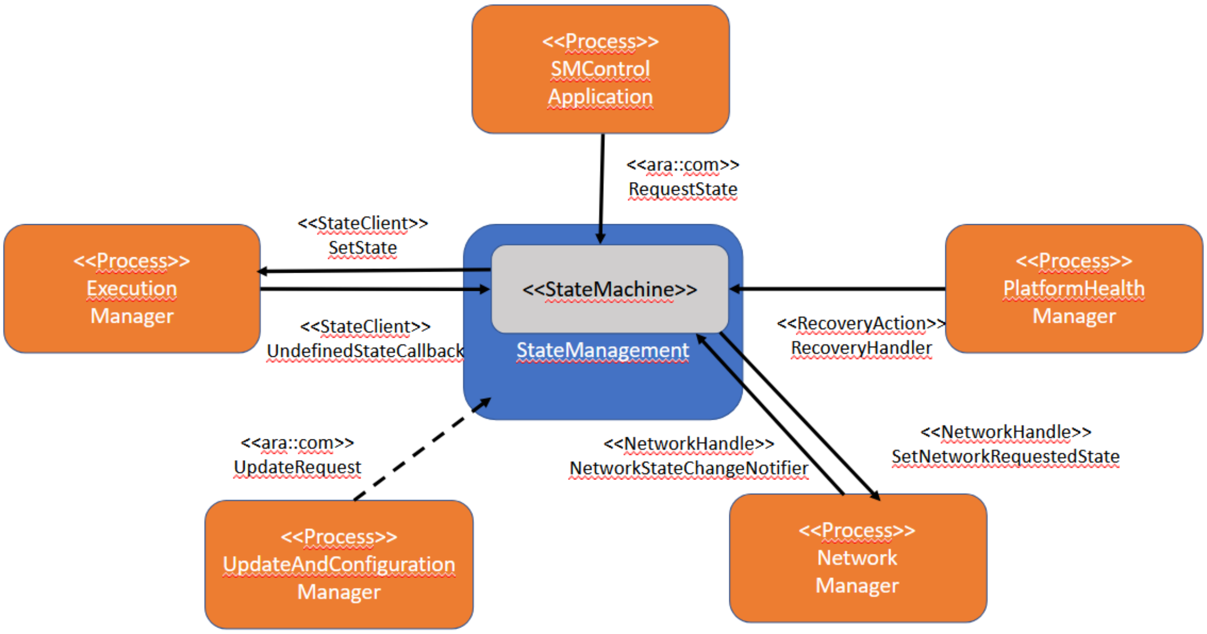


Figure 7.9: Interactions with StateMachine

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

7.11.2 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.

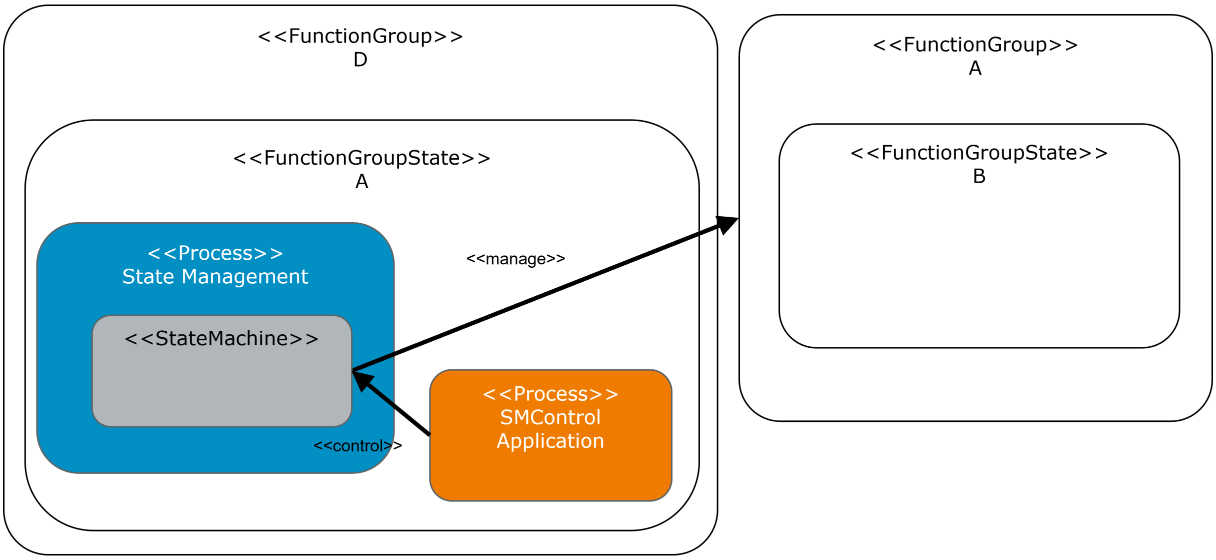


Figure 7.10: SMControllApplication and StateManagement Process started together

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

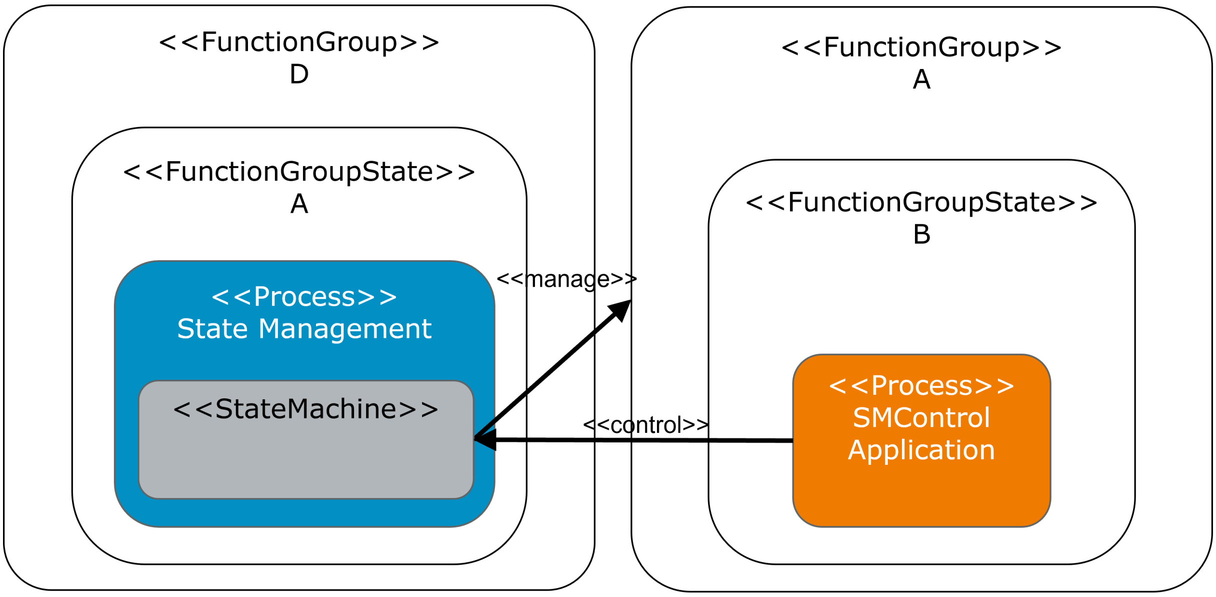


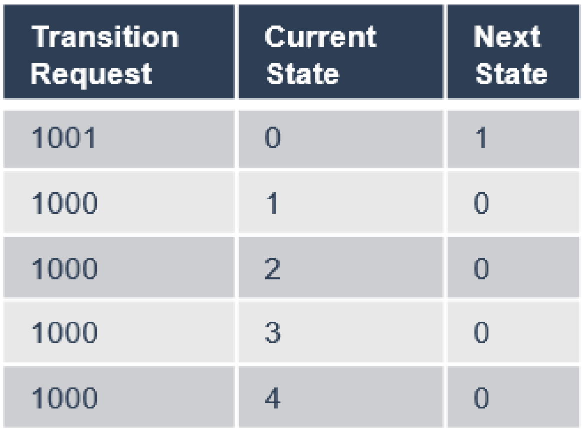
Figure 7.11: SMControllApplication started in initial State of StateManagements StateMachine

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

SMControlApplication is needed when arbitrary state changes could be requested as per StateMachine configuration. If the only functionality provided by StateMachine is the reaction to errors reported by Platform Health Management and/or Execution Management, or reaction to changes in NetworkHandles, then there is no need to have a SMControlApplication. In that case, StateMachine should start intended functionality when it enters the Initial State.

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

The SMControlApplication, uses the RequestState 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	0	1
1000	1	0
1000	2	0
1000	3	0
1000	4	0

Figure 7.12: TransitionRequestTable

Please note that appendix A.10 of TPS Manifest Specification [8] shows in detail how the TransitionRequestTable and the ErrorRecoveryTable can be build with the available meta-class elements.

[SWS_SM_00600]{DRAFT} **StateMachine service interface** ⌈State Management shall provide an ara::com based service (StateMachineService) for each instance of the StateMachine configured.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00618]{DRAFT} **StateMachine service interface - Offer** ⌈Each configured ara::com based service (StateMachineService) for a StateMachine shall be available (offered) at the time when processing of ActionListItem "StartStateMachine" is finished.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00619]{DRAFT} **StateMachine service interface - StopOffer** ⌈Each configured ara::com based service (StateMachineService) for a StateMachine shall be no longer available (offered) at the time when processing of ActionListItem "StopStateMachine" is finished.⌋*(RS_SM_00001, RS_SM_00005)*

7.11.3 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.

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)

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

[SWS_SM_00648]{DRAFT} **StateMachine of type Controller start** ⌈When Modelled Process controlling StateMachine of type Controller starts it shall start StateMachine of type Controller.⌋*(RS_SM_00001, RS_SM_00005)*

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

[SWS_SM_CONSTR_00017]{DRAFT} **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.

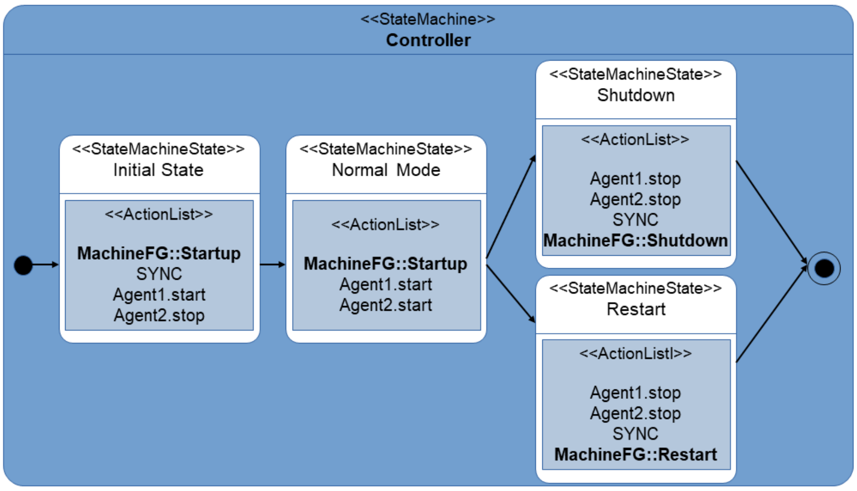


Figure 7.13: Example for Controller StateMachineStates with MachineFG

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

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

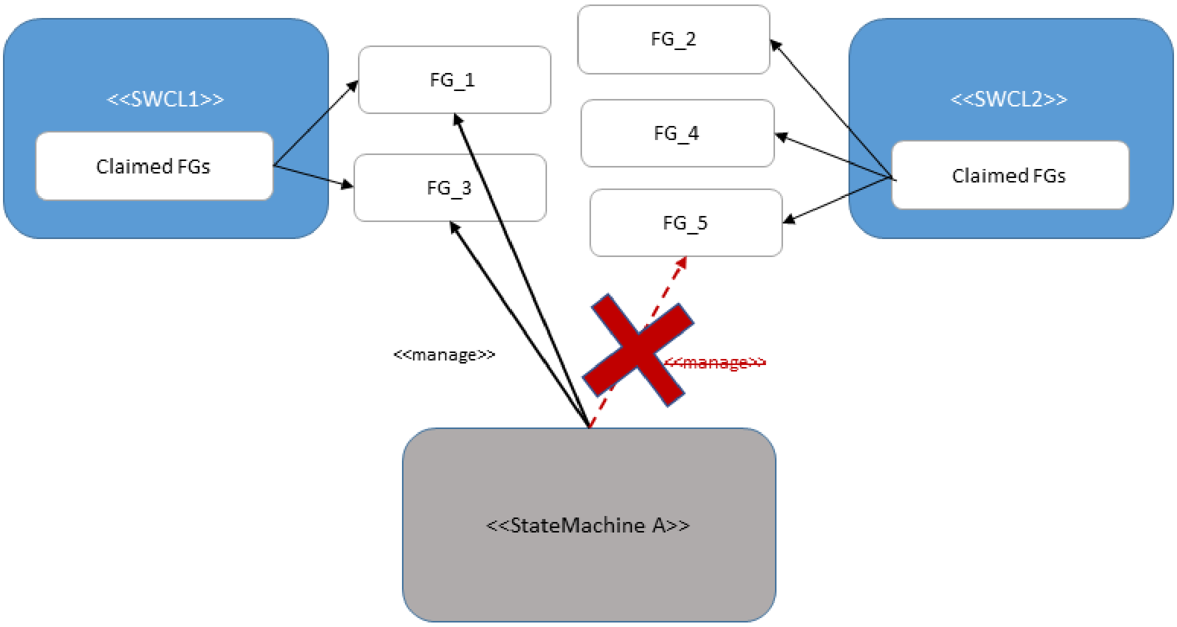


Figure 7.14: Agent - FunctionGroup relation

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.

7.11.4 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_00011]{DRAFT} **Function Group States referenced in the final state of a StateMachine** ⌈The ActionList referenced by the Final State of a StateMachine shall only contain ActionListItems that reference the "Off" state of the controlled Function Group.⌋*()*

[SWS_SM_CONSTR_00012]{DRAFT} **Stop running StateMachines in the final state of a StateMachine** ⌈When any StateMachine was started by Start StateMachine ActionListItem, and not stopped before, the Final State shall contain ActionListItems to stop all running StateMachines⌋*()*

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.

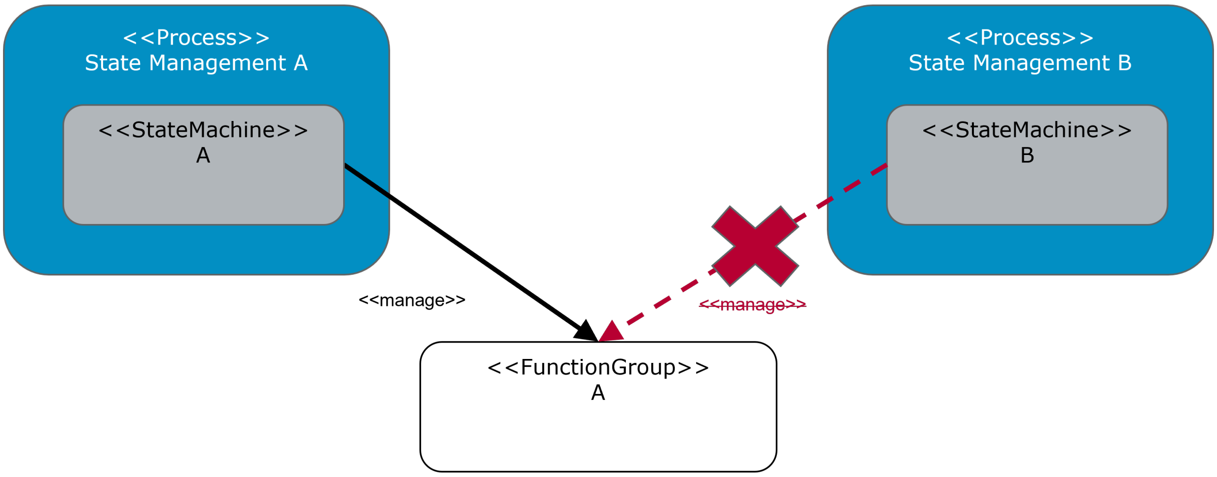
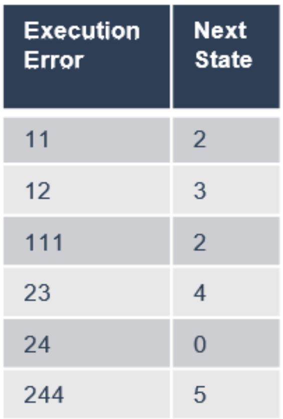


Figure 7.15: Function Group controlled by single StateMachine

[SWS_SM_CONSTR_00013]{DRAFT} **Function Group shall only be controlled by single StateMachine** ⌈A Function Group shall only be referenced by ActionListItems of exactly one StateMachine.⌋*()*

7.11.5 StateMachine state changes

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



Execution Error	Next State
11	2
12	3
111	2
23	4
24	0
244	5

Figure 7.16: ErrorRecoveryTable

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

[SWS_SM_CONSTR_00014]{DRAFT} **Handling of non-mapped ExecutionError** ⌈Each ErrorRecoveryTable shall have exactly one entry configured with value ANY as the ExecutionError⌋*()*

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

[SWS_SM_00601]{DRAFT} **StateMachine error notification reaction** ⌈When ExecutionError::EventexecutionError is reported via ara::phm::RecoveryAction::RecoveryHandler (modelled as StateManagemenPhmErrorInterface from Platform Health Management or via undefinedStateCallback or SetState from ara::exec::StateClient (modelled as StateManagementEmErrorInterface) from Execution Management, StateMachine shall

- set internal flag ErrorRecoveryOngoing
- evaluate the next StateMachine State configured for executionError from ErrorRecoveryTable
- stop processing ActionListItems from the ActionList referencing the current StateMachine State

- switch to the next StateMachine State immediately and start processing ActionListItems from the ActionList referencing this StateMachine State

⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00602]{DRAFT} **StateMachine ErrorRecoveryOngoing flag reset** ⌈The internal ErrorRecoveryOngoing flag shall be reset, when all ActionListItems of an ActionList referencing a StateMachine State, which is requested due to error reaction, are successfully processed.⌋*(RS_SM_00001, RS_SM_00005)*

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

[SWS_SM_00603]{DRAFT} **StateMachine service interface RequestState - not allowed transition** ⌈The RequestState 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.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00604]{DRAFT} **StateMachine service interface RequestState - invalid transition** ⌈The RequestState method shall return kInvalidValue if TransitionRequest value is not configured in TransitionRequestTable and shall cease any further processing of the request.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00605]{DRAFT} **StateMachine service interface RequestState - recovery ongoing** ⌈The RequestState method shall return kRecoveryTransitionOngoing if internal flag ErrorRecoveryOngoing is set and shall cease any further processing of the request.⌋*(RS_SM_00001, RS_SM_00005)*

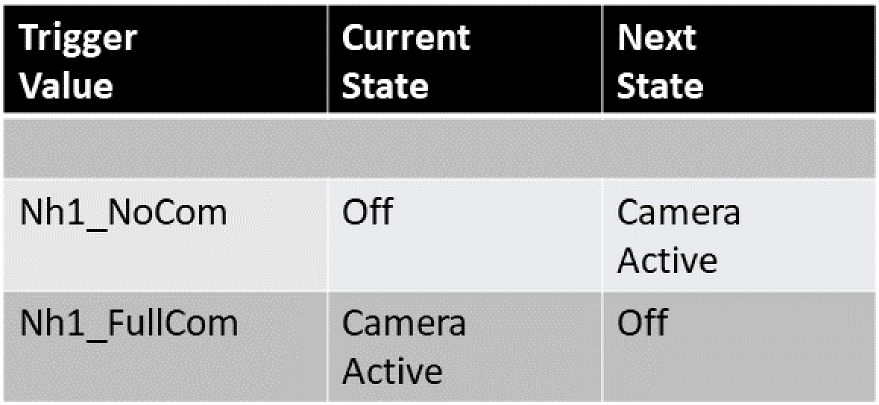
[SWS_SM_00606]{DRAFT} **Canceling ongoing state transition of StateMachine** ⌈If transition request was accepted, RequestState method shall return kCanceled to previous RequestState requests if any is still pending for the StateMachine.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00607]{DRAFT} **StateMachine transition execution** ⌈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.

⌋*(RS_SM_00001, RS_SM_00005)*

There is another source for StateMachine State change requests: Network Management NetworkHandle changes. As NetworkHandles are modelled as Port-Prototypes, 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.



Trigger Value	Current State	Next State
Nh1_NoCom	Off	Camera Active
Nh1_FullCom	Camera Active	Off

Figure 7.17: Extended transition request table

[SWS_SM_00620]{DRAFT} **StateMachine transition - NetworkHandle goes to FullCom** ⌈When StateMachine receives a change of a NetworkHandle 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.

⌋*(RS_SM_00001, RS_SM_00005, RS_SM_00401)*

[SWS_SM_00621]{DRAFT} **StateMachine transition - NetworkHandle goes to NoCom** ⌈When StateMachine receives a change of a NetworkHandle 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.

⌋*(RS_SM_00001, RS_SM_00005, RS_SM_00401)*

7.11.6 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.

7.11.7 StateMachine ActionListItems

There are multiple kinds of ActionListItems:

- Requesting a Function Group State
- Start a StateMachine with optional parameter state
- Stop a StateMachine
- SYNC to sync between different ActionListItems
- Sleep to delay processing the next ActionListItems
- SetNetworkHandle switches the provided NetworkHandle to the configured state(NoCom or FullCom)

[SWS_SM_00608]{DRAFT} **ActionListItem - Function Group State** ⌈When a Function Group State ActionListItem is found in the ActionLists, StateMachine shall request the configured Function Group State from Execution Management.⌋*(RS_SM_00001, RS_SM_00005)*

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

[SWS_SM_00609]{DRAFT} **ActionList processing order** ⌈The ActionListItems in the ActionLists shall be processed in the order they are configured.⌋*(RS_SM_00001, RS_SM_00005)*

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]{DRAFT} **processing SYNC ActionListItem** ⌈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".⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00611]{DRAFT} **processing ActionListItem** ⌈ActionListItems shall be processed in parallel unless SYNC ActionListItem is processed.⌋*(RS_SM_00001, RS_SM_00005)*

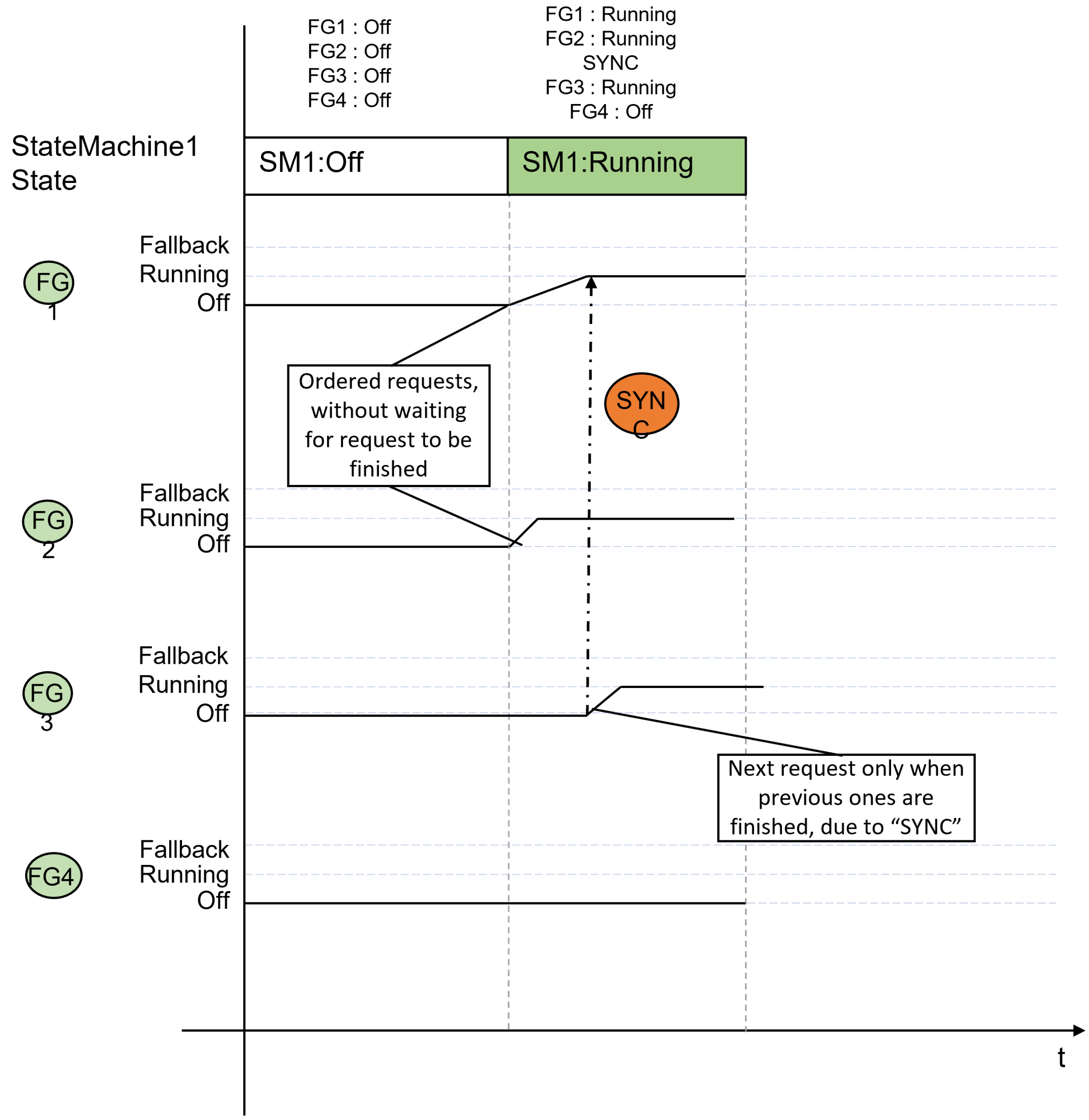


Figure 7.18: Parallel ActionListItem execution and SYNC

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.

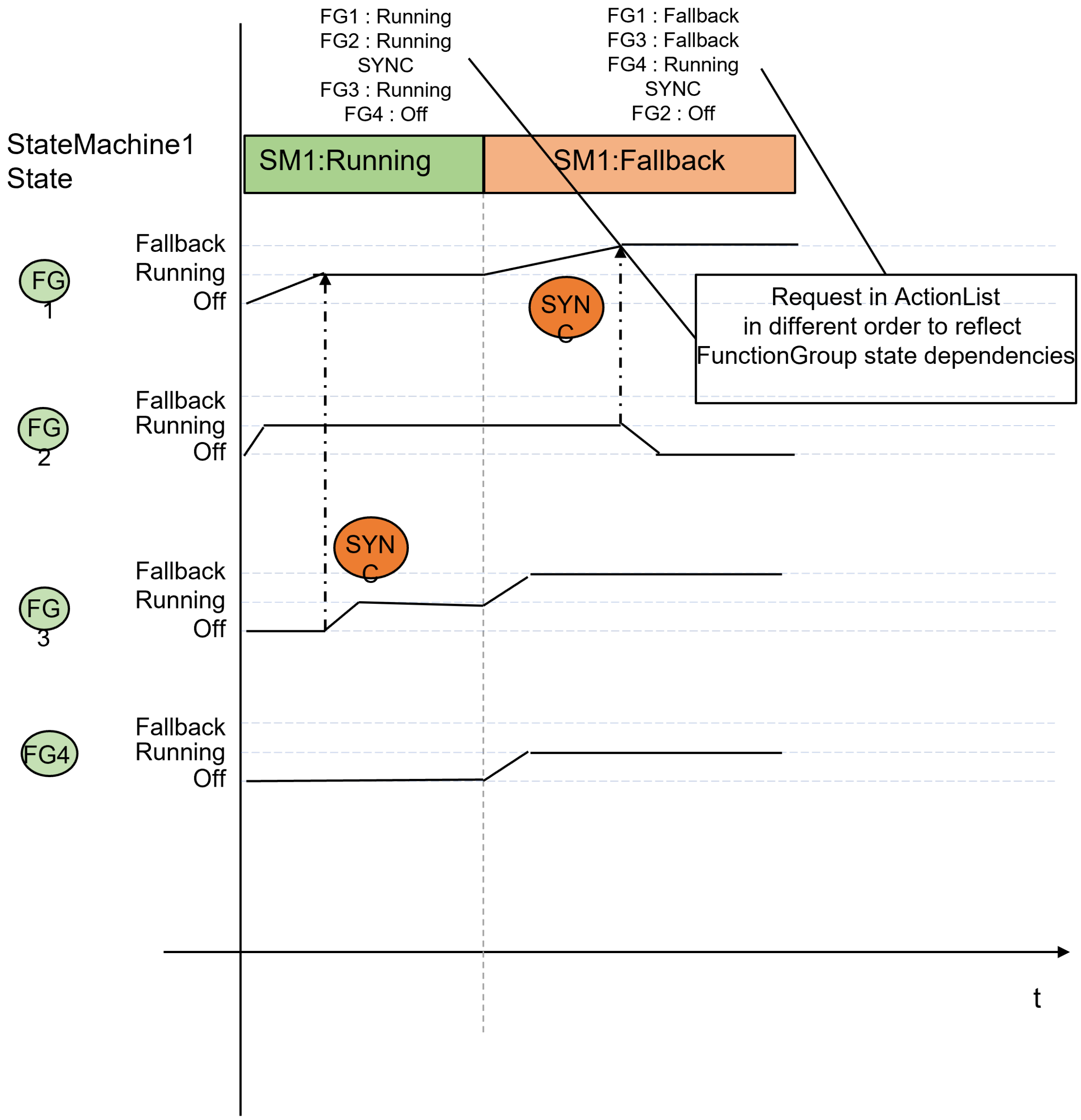


Figure 7.19: Arbitrary order for ActionListItems

To ensure that no Function Group 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, even if their state does not change from a StateMachine State to another.

[SWS_SM_CONSTR_00015]{DRAFT} **Completeness of controlled Function Groups** ⌈Each ActionList referencing different StateMachine States of the same StateMachine shall reference the same set of Function Groups.⌋*()*

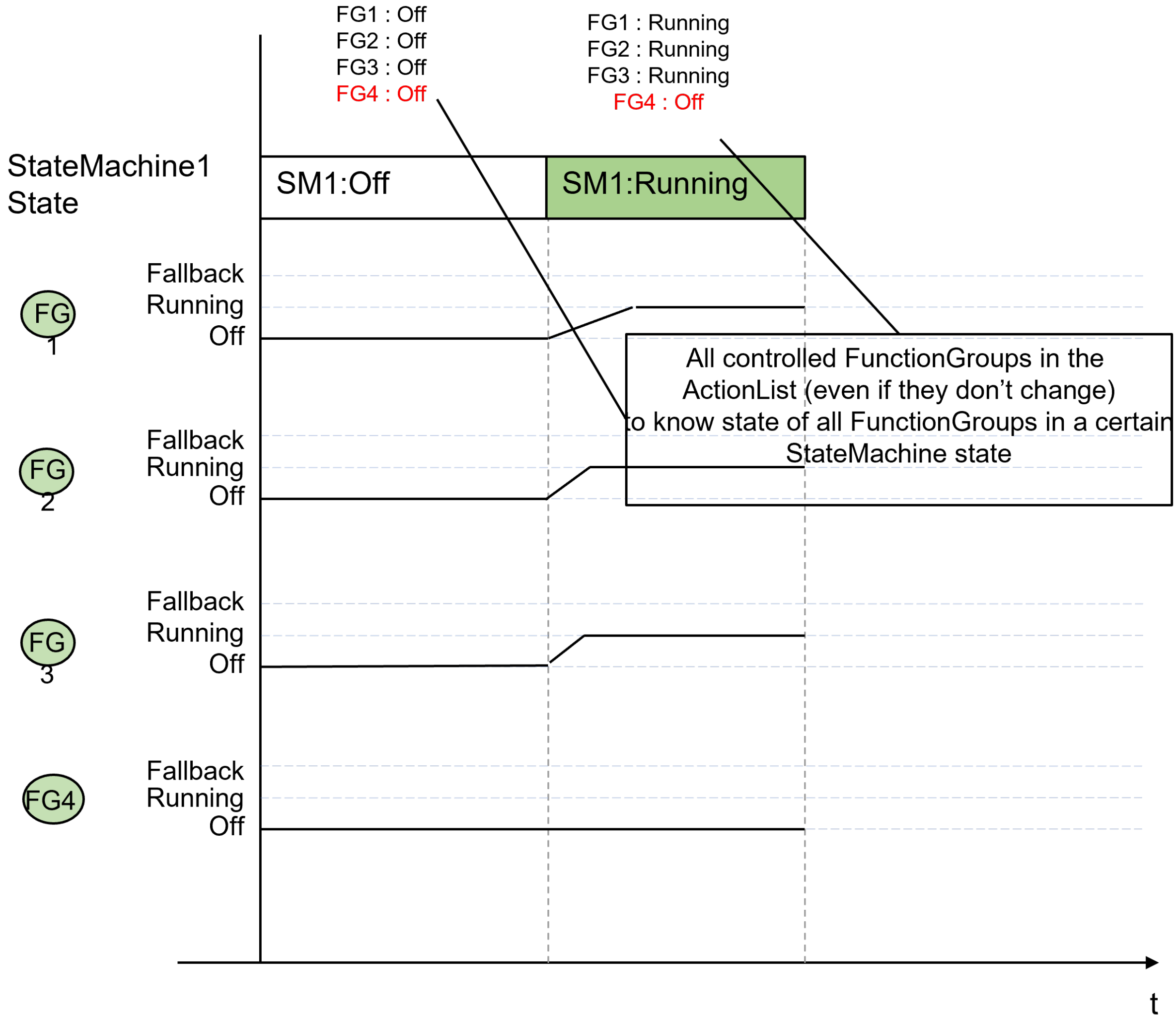


Figure 7.20: Completeness of controlled Function Groups

7.11.8 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]{DRAFT} **Usage of ActionListItem "StartStateMachine" and "StopStateMachine"** ⌈Only the StateMachine with the role Controller shall use the ActionListItem "StartStateMachine" and "StopStateMachine".⌋*()*

[SWS_SM_CONSTR_00016]{DRAFT} **Completeness of controlled StateMachines** ⌈Each ActionList referencing a StateMachine State of the same StateMachine shall reference the same set of controlled StateMachines.⌋*()*

[SWS_SM_00612]{DRAFT} **ActionListItem "Start StateMachine" without parameter, StateMachine is not running** ⌈When the ActionListItem "Start StateMachine" is processed, the referenced StateMachine shall be started. The StateMachine shall transition to the configured initial state.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00622]{DRAFT} **ActionListItem "Start StateMachine" with parameter, StateMachine is not running** ⌈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.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00613]{DRAFT} **ActionListItem "Start StateMachine" - without parameter, StateMachine is already running** ⌈When the ActionListItem "Start StateMachine" is processed, and the referenced StateMachine is already started, this processing shall be skipped.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00623]{DRAFT} **ActionListItem "Start StateMachine" - with parameter, StateMachine is already running** ⌈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.⌋*(RS_SM_00001, RS_SM_00005)*

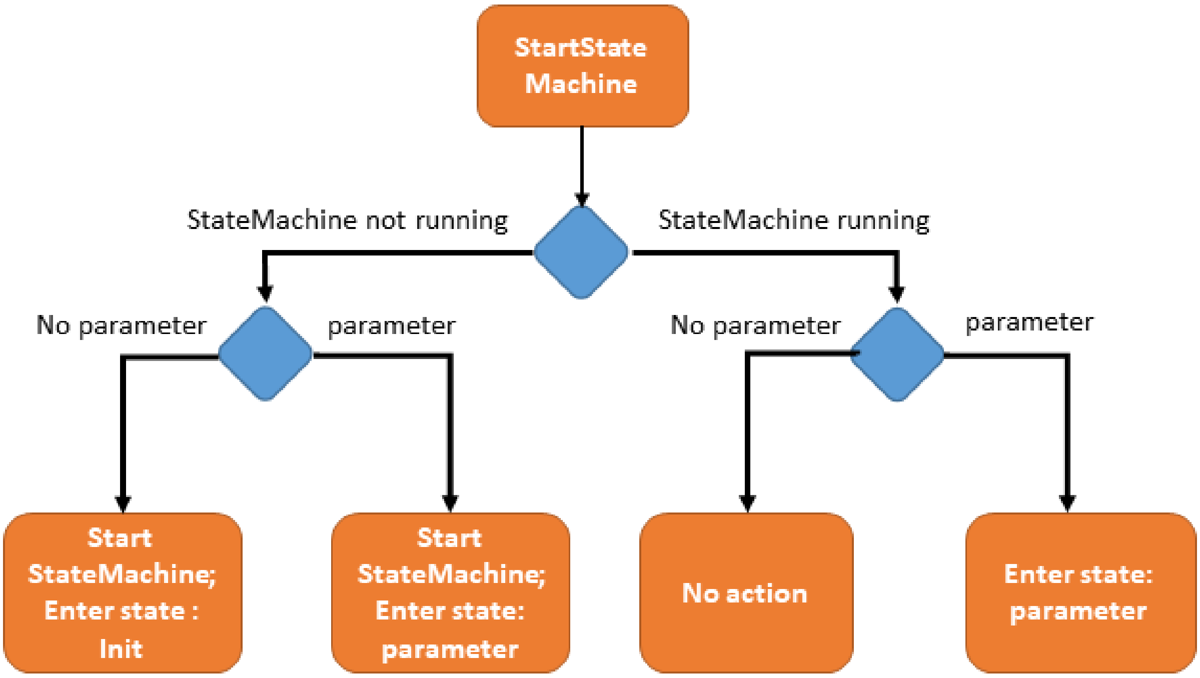


Figure 7.21: StartStateMachine decision tree

[SWS_SM_00614]{DRAFT} **ActionListItem "Stop StateMachine" processing** ⌈When the ActionListItem "Stop StateMachine" is processed, the StateMachine with the provided ID shall be stopped.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00615]{DRAFT} **ActionListItem "Stop StateMachine" processing - StateMachine is not running** ⌈When the ActionListItem "Stop StateMachine" is processed, and the StateMachine with the provided ID is not running, this processing shall be skipped.⌋*(RS_SM_00001, RS_SM_00005)*

7.11.9 ActionListItem Sleep

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

[SWS_SM_00624]{DRAFT} **ActionListItem - Sleep** ⌈When a Sleep ActionListItem is found in the ActionLists, StateMachine shall delay processing next ActionListItem on the ActionLists for the configured time.⌋*(RS_SM_00001, RS_SM_00005)*

Please note that Sleep ActionListItem will not "block" processing incoming triggers meanwhile. This means that a call to RequestState, 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).

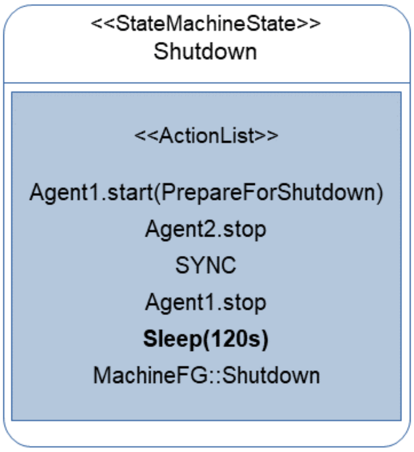


Figure 7.22: Example for an ActionList using ActionListItem Sleep

7.11.10 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]{DRAFT} **ActionListItem - SetNetworkHandle FullCom** ⌈When a SetNetworkHandle ActionListItem with parameter FullCom is found in the ActionLists, StateMachine shall set the corresponding NetworkHandle to FullCom.⌋*(RS_SM_00001, RS_SM_00005, RS_SM_00401)*

[SWS_SM_00626]{DRAFT} **ActionListItem - SetNetworkHandle NoCom** ⌈When a SetNetworkHandle ActionListItem with parameter NoCom is found in the ActionLists, StateMachine shall set the corresponding NetworkHandle to NoCom.⌋*(RS_SM_00001, RS_SM_00005, RS_SM_00401)*

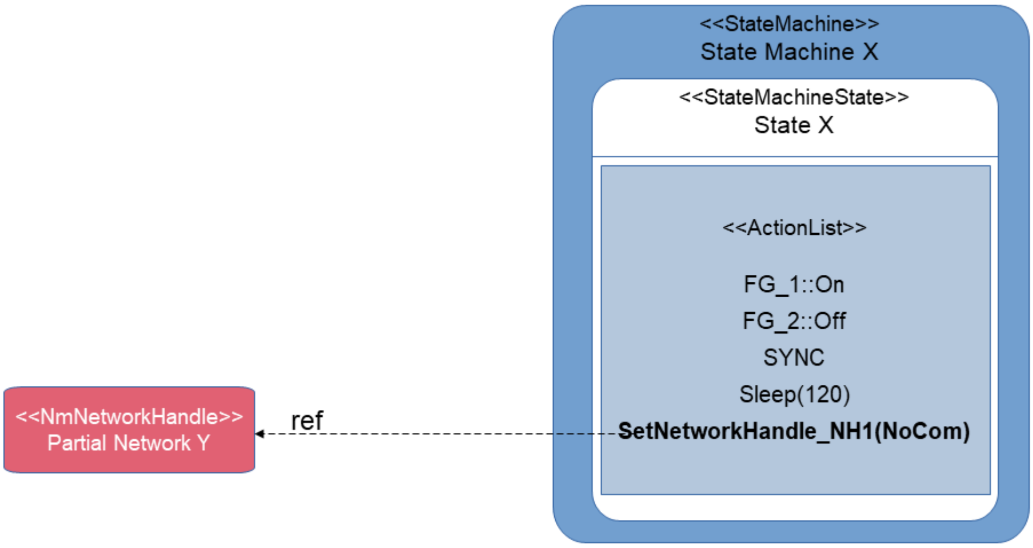


Figure 7.23: Afterrun example using the SetNetworkHandle in combination with Sleep

7.11.11 StateMachine State notification

As State Management's 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 TriggerOut::Notifier service interface (modelled as meta-class ServiceInterface) for a StateMachine.

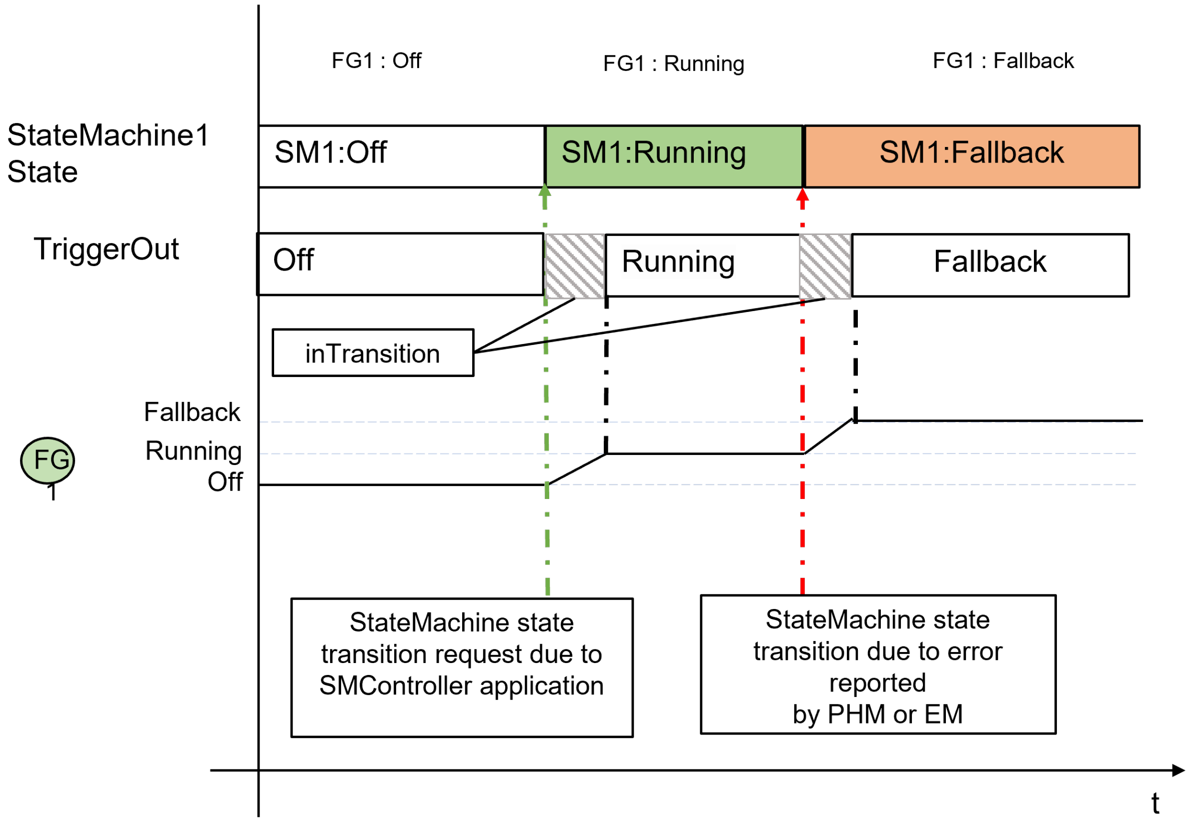


Figure 7.24: Value of configured TriggerOut::Notifier field

[SWS_SM_00616]{DRAFT} **Notifier value during StateMachine State transition** ⌈When a TriggerOut interface is configured for the StateMachine and a StateMachine State transition has been started, the value of the "Notifier" field shall be set to "inTransition".⌋*(RS_SM_00001, RS_SM_00005)*

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]{DRAFT} **Notifier value after StateMachine State transition** ⌈When a TriggerOut interface is configured for the StateMachinethe value of the "Notifier" 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.⌋*(RS_SM_00001, RS_SM_00005)*

7.11.12 StateMachine support for Update and Configuration Management

To support Update and Configuration Management [7] 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.

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]{DRAFT} **Upper multiplicity of UpdateRequest interface** ⌈In the context of Machine there shall be at most one instance of UpdateRequest interface at the time when the creation of the manifest is finished.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00629]{DRAFT} **Only Process controlling StateMachine of type Controller can provide UpdateRequest interface** ⌈Only Modelled Process controlling StateMachine of type Controller shall be able to instantiate UpdateRequest interface.⌋*(RS_SM_00001, RS_SM_00005)*

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]{DRAFT} **Rejection of update session** ⌈When UpdateAllowed is set to kUpdateNotAllowed, Modelled Process controlling StateMachine of type Controller shall return kRejected error from the RequestUpdateSession method.⌋*(RS_SM_00001, RS_SM_00005)*

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]{DRAFT} **Acceptance of update session** ⌈When UpdateAllowed is set to kUpdateAllowed, Modelled Process controlling StateMachine of type Controller shall return success from the RequestUpdateSession method.⌋*(RS_SM_00001, RS_SM_00005)*

Please note that it is deliberately left as an implementation detail when RequestState 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.

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]{DRAFT} **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.⌋*(RS_SM_00001, RS_SM_00005)*

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.

[SWS_SM_00632]{DRAFT} **Block RequestState method after PrepareUpdate call** ⌈Any call to RequestState for StateMachine of type Controller shall return kUpdateInProgress once PrepareUpdate method has been invoked.⌋*(RS_SM_00001, RS_SM_00005)*

Please note that once a call to RequestState 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]).

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]{DRAFT} **Transition affected StateMachines to PrepareUpdate state** ⌈Modelled Process controlling StateMachine of type Controller, during a call to PrepareUpdate method, shall transition every affected StateMachine to the PrepareUpdate state.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00634]{DRAFT} **Shutdown of affected StateMachines during a call to PrepareUpdate method** ⌈Modelled Process controlling StateMachine of type Controller, during a call to PrepareUpdate method, shall stop every affected StateMachine of type Agent.⌋*(RS_SM_00001, RS_SM_00005)*

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]{DRAFT} **Failing to prepare for update** ⌈If Modelled Process controlling StateMachine of type Controller fails to prepare for the update process, it shall return kFailed error from the PrepareUpdate method.⌋*(RS_SM_00001, RS_SM_00005)*

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]{DRAFT} **Successful preparation for update** ⌈When Modelled Process controlling StateMachine of type Controller successfully prepares for update, it shall return success from the PrepareUpdate method.⌋*(RS_SM_00001, RS_SM_00005)*

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]{DRAFT} **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.⌋*(RS_SM_00001, RS_SM_00005)*

Before starting verification, it is needed to block RequestState method - when not already done. Not every update session will require a call to PrepareUpdate method. This is especially true when new functionality is installed.

[SWS_SM_00637]{DRAFT} **Block RequestState method after VerifyUpdate call** ⌈Any call to RequestState for StateMachine of type Controller shall return kUpdateInProgress once VerifyUpdate method has been invoked.⌋*(RS_SM_00001, RS_SM_00005)*

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]{DRAFT} **Transition affected StateMachines to VerifyUpdate state** ⌈Modelled Process controlling StateMachine of type Controller, during a call to VerifyUpdate method, shall transition every affected StateMachine to the VerifyUpdate state.⌋*(RS_SM_00001, RS_SM_00005)*

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

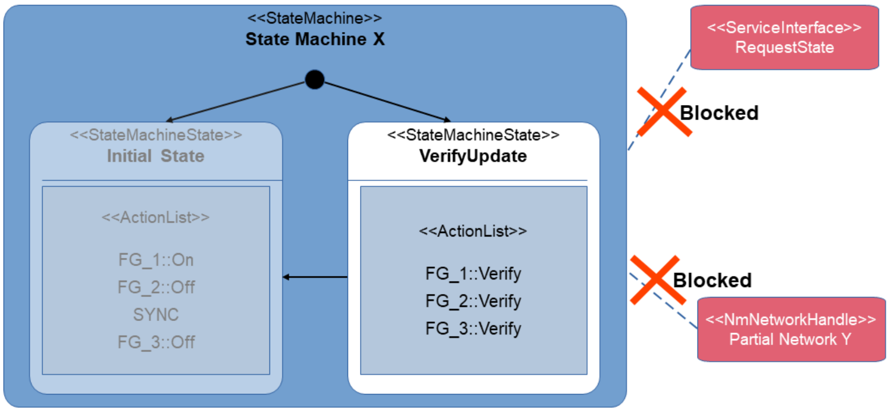


Figure 7.25: Example for StateMachineState VerifyUpdate for an Agent

To enable a StateMachine of type Agent to fulfill all steps which are needed in the StateMachine State VerifyUpdate, it is needed that the StateMachine State can't be influenced from the outside.

[SWS_SM_00649]{DRAFT} **Block RequestState method in VerifyUpdate state** ⌈Any call to RequestState for StateMachine of type Agent shall return kUpdateInProgress when current StateMachine State is VerifyUpdate.⌋*(RS_SM_00001, RS_SM_00005)*

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

[SWS_SM_00627]{DRAFT} **Evaluation of NetworkHandle changes during VerifyUpdate state** ⌈StateMachines of type Agent shall keep their StateMachine State, when current StateMachine State is VerifyUpdate and changes in a NmNetworkHandle are recognized.⌋*(RS_SM_00001, RS_SM_00005)*

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]{DRAFT} **Evaluation of NetworkHandle changes for StateMachine of type Controller** ⌈StateMachine of type Controller shall keep its StateMachine State, when the RequestState for StateMachine of type Agent returns kUpdateInProgress and changes in a NmNetworkHandle are recognized.⌋*(RS_SM_00001, RS_SM_00005)*

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]{DRAFT} **Unsuccessful verification of updated software** ⌈If Modelled Process controlling StateMachine of type Controller fails to verify updated software, it shall return kFailed error from the VerifyUpdate method.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00640]{DRAFT} **Successful verification of updated software** ⌈When Modelled Process controlling StateMachine of type Controller successfully verifies updated software, it shall return success from the VerifyUpdate method.⌋*(RS_SM_00001, RS_SM_00005)*

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.

[SWS_SM_CONSTR_00023]{DRAFT} **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.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00641]{DRAFT} **Block RequestState method after PrepareRollback call** ⌈Any call to RequestState for StateMachine of type Controller shall return kUpdateInProgress once PrepareRollback method has been invoked.⌋*(RS_SM_00001, RS_SM_00005)*

After this preparation for rollback can be started.

[SWS_SM_00642]{DRAFT} **Transition affected StateMachines to PrepareRollback state** ⌈Modelled Process controlling StateMachine of type Controller, during a call to PrepareRollback method, shall transition every affected StateMachine to the PrepareRollback state.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00643]{DRAFT} **Shutdown of affected StateMachines during a call to PrepareRollback method** ⌈Modelled Process controlling StateMachine of type Controller, during a call to PrepareRollback method, shall stop every affected StateMachine of type Agent.⌋*(RS_SM_00001, RS_SM_00005)*

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

[SWS_SM_00644]{DRAFT} **Failing to prepare for rollback** ⌈If Modelled Process controlling StateMachine of type Controller fails to prepare for the rollback process, it shall return kFailed error from the PrepareRollback method.⌋*(RS_SM_00001, RS_SM_00005)*

[SWS_SM_00645]{DRAFT} **Successful preparation for rollback** ⌈When Modelled Process controlling StateMachine of type Controller successfully prepares for rollback, it shall return success from the PrepareRollback method.⌋*(RS_SM_00001, RS_SM_00005)*

Update session ends with a call to StopUpdateSession method. At that point Machine is in an undefined state. To counter this situation Modelled Process controlling StateMachine of type Controller should restore the StateMachine State that was active for StateMachine of type Controller when update session was started.

[SWS_SM_00646]{DRAFT} **Restoring the last known state after update session** ⌈Modelled Process controlling StateMachine of type Controller, upon receiving StopUpdateSession call, shall transition StateMachine of type Controller to a StateMachine State that was active when RequestUpdateSession method was called.⌋*(RS_SM_00001, RS_SM_00005)*

After restoring a known (configured) state, requests to RequestState method can be enabled again and finally the update session will end. Even a call to RequestState for StateMachine of type Agent will shall not return kUpdateInProgress (see [SWS_SM_00649]) any longer, as the StateMachine State of StateMachine of type Agent will not be in VerifyUpdate state due to [SWS_SM_00646].

[SWS_SM_00647]{DRAFT} **Enabling RequestState method after StopUpdateSession call** ⌈Once StopUpdateSession method has been invoked any call to RequestState for StateMachine of type Controller shall not return kUpdateInProgress any longer.⌋*(RS_SM_00001, RS_SM_00005)*

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

8 API specification

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

9 Service Interfaces

9.1 Type definitions

9.1.1 Data types for Update And Configuration Management interaction

[SWS_SM_91018]{DRAFT} **Definition of ImplementationDataType FunctionGroupListType** ⌈

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

⌋*(RS_SM_00004, RS_AP_00150, RS_AP_00122)*

[SWS_SM_91019]{DRAFT} **Definition of ImplementationDataType FunctionGroupNameType** ⌈

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

⌋*(RS_SM_00004, RS_AP_00150, RS_AP_00122)*

9.1.2 Data types for StateMachine interaction

[SWS_SM_91023]{DRAFT} **Definition of ImplementationDataType TransitionRequestType** ⌈

Name	TransitionRequestType
Namespace	ara::sm
Kind	u_int32
Derived from	-
Description	a value which represents the requested state in the StateMachine.

⌋*(RS_SM_00004, RS_SM_00001, RS_AP_00150)*

9.1.3 Data types for UpdateAllowed service interface

[SWS_SM_91026]{DRAFT} **Definition of ImplementationDataType UpdateAllowedType** ⌈

Name	UpdateAllowedType	
Namespace	ara::sm	
Kind	u_int8	
Derived from	-	
Description	UpdateAllowedType	
Range / Symbol	***Limit***	***Description***
kUpdateAllowed		kUpdateAllowed
kUpdateNotAllowed		kUpdateNotAllowed

⌋*(RS_SM_00001, RS_SM_00004)*

9.2 Provided Service Interfaces

9.2.1 State Management TriggerIn

Port

[SWS_SM_91001]{DRAFT} **Definition of Port TriggerIn_{State} provided by functional cluster SM** ⌈

Name	TriggerIn_{State}		
Kind	ProvidedPort	***Interface***	`TriggerIn`
Description	To be used by Adaptive (Platform) Applications to tigger State Management to change its internal state.		
Variation			

⌋*(RS_SM_00004, RS_SM_00005, RS_AP_00150)*

Service Interface

[SWS_SM_91007]{DRAFT} **Definition of ServiceInterface TriggerIn** ⌈

Name	TriggerIn_{StateGroup}
Namespace	ara::sm

Field	Trigger
Description	Value to be evaluated by State Management in a projectspecific way.
Type	`project_specific`
HasGetter	false
HasNotifier	false
HasSetter	true

⌋*(RS_SM_00004, RS_SM_00005, RS_AP_00150, RS_AP_00115)*

9.2.2 State Management TriggerOut

Port

[SWS_SM_91002]{DRAFT} **Definition of Port TriggerOut_{State} provided by functional cluster SM** ⌈

Name	TriggerOut_{State}		
Kind	ProvidedPort	**Interface**	TriggerOut
Description	To be used by Adaptive (Platform) Applications to be informed when State Management has changed its internal state.		
Variation			

⌋*(RS_SM_00004, RS_SM_00005, RS_AP_00150)*

Service Interface

[SWS_SM_91008]{DRAFT} **Definition of ServiceInterface TriggerOut** ⌈

Name	TriggerOut_{StateGroup}
Namespace	ara::sm

Field	Notifier
Description	To be set by State Management in a projectspecific way to inform Adaptive (Platform) Applications about changes within State Management
Type	`project_specific`
HasGetter	true
HasNotifier	true
HasSetter	false

⌋*(RS_SM_00004, RS_SM_00005, RS_AP_00150, RS_AP_00115)*

9.2.3 State Management TriggerInOut

Port

[SWS_SM_91003]{DRAFT} **Definition of Port TriggerInOut_{State} provided by functional cluster SM** ⌈

Name	TriggerInOut_{State}		
Kind	ProvidedPort	**Interface**	`TriggerInOut`
Description	To be used by Adaptive (Platform) Applications to tigger State Management to change its internal state and to get information when it is carried out.		
Variation			

⌋*(RS_SM_00004, RS_SM_00005, RS_AP_00150)*

Service Interface

[SWS_SM_91009]{DRAFT} **Definition of ServiceInterface TriggerInOut** ⌈

Name	TriggerInOut_{StateGroup}
Namespace	ara::sm

Field	Trigger
Description	Value to be evaluated by State Management in a projectspecific way.
Type	`project_specific`
HasGetter	false
HasNotifier	false
HasSetter	true

Field	Notifier
Description	To be set by State Management in a projectspecific way to inform Adaptive (Platform) Applications about changes within State Management
Type	`project_specific`
HasGetter	true
HasNotifier	true
HasSetter	false

⌋*(RS_SM_00004, RS_SM_00005, RS_AP_00150, RS_AP_00115)*

9.2.4 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]{DRAFT} Definition of Port UpdateRequest provided by functional cluster SM ⌈

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			

⌋*(RS_SM_00001, RS_SM_00004, RS_AP_00150)*

Service Interface

[SWS_SM_91017] Definition of ServiceInterface UpdateRequest ⌈

Name	UpdateRequest
Namespace	ara::sm

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.	
FireAndForget	false	
Application Errors	kRejected	Requested operation was rejected due to State Managements/machines internal state.

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.	
FireAndForget	false	
Application Errors	kRejected	Requested operation was rejected due to State Managements/machines internal state.

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.	
FireAndForget	false	
Application Errors	kRejected	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.

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.	
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	kRejected	Requested operation was rejected due to State Managements/machines internal state.
Application Errors	kFailed	Requested operation failed.

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.	
FireAndForget	false	
Parameter	functionGroupList	
	Description	The list of FunctionGroups within the SoftwareCluster to be verified.
	Type	FunctionGroupListType
	Variation	
	Direction	IN
Application Errors	kRejected	Requested operation was rejected due to State Managements/machines internal state.
Application Errors	kFailed	Requested operation failed.

Method	PrepareRollback	
Description	Has to be called by Update And Configuration Management after State Management allowed to update.	
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	kRejected	Requested operation was rejected due to State Managements/machines internal state.
Application Errors	kFailed	Requested operation failed.

⌋*(RS_SM_00001, RS_SM_00004, RS_AP_00150, RS_AP_00115, RS_AP_00120, RS_AP_00142, RS_AP_00119, RS_AP_00121)*

9.2.5 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]{DRAFT} **Definition of Port StateMachineService provided by functional cluster SM** ⌈

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

⌋*(RS_SM_00001, RS_SM_00004, RS_AP_00150)*

Service Interface

[SWS_SM_91022]{DRAFT} **Definition of ServiceInterface StateMachineService** ⌈

Name	StateMachineService
Namespace	ara::sm

Method	RequestState	
Description	Has to be called by a SMControlApplication to request a change in the referenced StateMachine.	
FireAndForget	false	
Parameter	TransitionRequest	
	Description	Represents the requested state in the StateMachine.
	Type	TransitionRequestType
	Variation	
	Direction	IN
Application Errors	kInvalidValue	The provided value is not mapped to any transition.
Application Errors	kTransitionNotAllowed	Requested transition is not possible from current StateMachine state.
Application Errors	kRecoveryTransitionOngoing	Request will not be carried out, because currently recovery is ongoing.
Application Errors	kTransitionFailed	During transition to the requested state an error occurred.
Application Errors	kCanceled	The request was replaced by a newer one and therefore it was cancelled
Application Errors	kUpdateInProgress	Requested operation is not allowed as update session is in progress.

⌋*(RS_SM_00001, RS_SM_00004)*

9.2.6 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]{DRAFT} **Definition of Port UpdateAllowedService provided by functional cluster SM** ⌈

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

⌋*(RS_SM_00001, RS_SM_00004, RS_AP_00150)*

Service Interface

[SWS_SM_91025]{DRAFT} **Definition of ServiceInterface UpdateAllowedService** ⌈

Name	UpdateAllowedService
Namespace	ara::sm

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

⌋*(RS_SM_00001, RS_SM_00004)*

9.3 Required Service Interfaces

9.3.1 Network Management

9.3.1.1 NetworkManagement NetworkState

Port

[SWS_SM_91004]{DRAFT} **Definition of Port NetworkState_{NetworkHandle} required by functional cluster SM** ⌈

Name	NetworkState_{NetworkHandle}		
Kind	RequiredPort	***Interface***	`NetworkState`
Description	Provides information about network status per NetworkHandle. Intended to be only used by State Management!		
Variation	`FOR NetworkHandle : MODEL.filterType("NetworkHandle");`		

⌋*(RS_SM_00004, RS_SM_00400, RS_AP_00150, RS_AP_00115)*

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
⌈

Name	***Code***	***Description***
kCanceled	14	The request was replaced by a newer one and therefore it was cancelled
kFailed	6	Requested operation failed.
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.
kRecoveryTransitionOngoing	12	Request will not be carried out, because currently recovery is ongoing.
kRejected	5	Requested operation was rejected due to State Managements/ machines internal state.
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.

⌋*(RS_SM_00004, RS_AP_00150, RS_AP_00125, RS_AP_00142, RS_AP_00119, RS_AP_00149)*

A Interfunctional Cluster Interfaces

No IFC-Interfaces are provided by State Management.

B Not applicable requirements

[SWS_SM_NA]{DRAFT} **Not applicable requirements** ⌈These requirements are not applicable as they are not within the scope of this release.⌋*(RS_AP_00132, RS_AP_00134, RS_AP_00133, RS_AP_00153, RS_AP_00144, RS_AP_00152, 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)*

C Mentioned Manifest Elements

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

Class	***Identifiable*** (abstract)
Package	M2::AUTOSARTemplates::GenericStructure::GeneralTemplateClasses::Identifiable
Note	Instances of this class can be referred to by their identifier (within the namespace borders). In addition to this, Identifiables are objects which contribute significantly to the overall structure of an AUTOSAR description. In particular, Identifiables might contain Identifiables.
Base	*ARObject*, *MultilanguageReferrable*, *Referrable*
Subclasses	ARPackage, *AbstractDoIpLogicAddressProps*, *AbstractEvent*, *AbstractImplementationDataTypeElement*, *AbstractSecurityEventFilter*, *AbstractSecurityIdsmInstanceFilter*, *AbstractServiceInstance*, *AbstractSignalBasedToISignalTriggeringMapping*, AdaptiveSwcInternalBehavior, ApApplicationEndpoint, ApplicationEndpoint, ApplicationError, AppliedStandard, ArtifactChecksum, ArtifactLocator, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpFeature*, AutosarOperationArgumentInstance, AutosarVariableInstance, *BuildActionEntity*, BuildActionEnvironment, Chapter, CheckpointTransition, ClassContentConditional, ClientIdDefinition, ClientServerOperation, Code, *CollectableElement*, ComManagementMapping, *CommConnectorPort*, *CommunicationConnector*, *CommunicationController*, Compiler, ConsistencyNeeds, ConsumedEventGroup, CouplingPort, *CouplingPortStructuralElement*, CryptoCertificate, CryptoKeySlot, CryptoProvider, *CryptoServiceMapping*, DataPrototypeGroup, DataTransformation, DdsCpDomain, DdsCpPartition, DdsCpQosProfile, DdsCpTopic, DdsDomainRange, DependencyOnArtifact, *DiagEventDebounceAlgorithm*, DiagnosticAuthTransmitCertificateEvaluation, DiagnosticConnectedIndicator, DiagnosticDataElement, DiagnosticDebounceAlgorithmProps, DiagnosticFunctionInhibitSource, DiagnosticParameterElement, *DiagnosticRoutineSubfunction*, DiagnosticSovdMethodPrimitive, DltApplication, DltArgument, DltMessage, DoIpInterface, DoIpLogicAddress, DoIpRoutingActivation, E2EProfileConfiguration, End2EndEventProtectionProps, End2EndMethodProtectionProps, EndToEndProtection, EthernetWakeupSleepOnDatalineConfig, EventHandler, EventMapping, ExclusiveArea, *ExecutableEntity*, *ExecutionTime*, FMAttributeDef, FMFeatureMapAssertion, FMFeatureMapCondition, FMFeatureMapElement, 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, MacMulticastGroup, MacSecKayParticipant, McDataInstance, MemorySection, MemoryUsage, MethodMapping, ModeDeclaration, ModeDeclarationMapping, ModeSwitchPoint, NetworkEndpoint, *NmCluster*, *NmNode*, *PackageableElement*, ParameterAccess, PduActivationRoutingGroup, PduToFrameMapping, PduTriggering, PerInstanceMemory, *PersistencyDeploymentElement*, *PersistencyInterfaceElement*, *PhmSupervision*, *PhysicalChannel*, PortGroup, *PortInterfaceMapping*, PossibleErrorReaction, ProcessToMachineMapping, Processor, ProcessorCore, PskIdentityToKeySlotMapping, ResourceConsumption, ResourceGroup, RootSwClusterDesignComponentPrototype, RootSwComponentPrototype, RootSwCompositionPrototype, RptComponent, RptContainer, RptExecutableEntity, RptExecutableEntityEvent, RptExecutionContext, RptProfile, RptServicePoint, RunnableEntityGroup, *SdgAttribute*, SdgClass, SecOcJobMapping, SecOcJobRequirement, SecureCommunicationAuthenticationProps, *SecureCommunicationDeployment*, SecureCommunicationFreshnessProps, SecurityEventContextProps, *ServiceEventDeployment*, *ServiceFieldDeployment*, ServiceInterfaceElementSecureComConfig, *ServiceMethodDeployment*, *ServiceNeeds*, SignalServiceTranslationEventProps, SignalServiceTranslationProps, SocketAddress, SoftwarePackageStep, SomeipEventGroup, SomeipProvidedEventGroup, SomeipTpChannel, *SpecElementReference*, *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*, *TransformationProps*, TransformationTechnology, Trigger, UcmDescription, UcmRetryStrategy, UcmStep, VariableAccess, VariationPointProxy, VehicleRolloutStep, ViewMap, VlanConfig, WaitPoint

Attribute	***Type***	***Mult.***	***Kind***	***Note***

▽

△

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

Table C.1: Identifiable

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

Table C.2: ModeDeclaration

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

Table C.3: ModeDeclarationGroupPrototype

Class	**NmInteractsWithSmMapping**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			

Note	This mapping represents an interaction from network management to state management.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=FCInteractions			

Base	*ARElement*, *ARObject*, *CollectableElement*, *FunctionalClusterInteractsWithFunctionalClusterMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *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 C.4: NmInteractsWithSmMapping

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

Table C.5: NmNetworkHandle

Enumeration	**NmStateRequestEnum**
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement
Note	This enumeration defines the description of states that can be requested from the network management. **Tags:** atp.Status=draft
Aggregated by	StateManagementNmActionItem.nmStateRequest
Literal	***Description***

fullCom	This literal represents that case that full communication should be possible.
	Tags:
	atp.EnumerationLiteralIndex=1
	atp.Status=draft

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

Table C.6: NmStateRequestEnum

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

Table C.7: ProcessExecutionError

Class	**ServiceInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface			
Note	This represents the ability to define a PortInterface that consists of a heterogeneous collection of methods, events and fields. **Tags:** atp.recommendedPackage=ServiceInterfaces			
Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***

event	VariableDataPrototype	*	aggr	This represents the collection of events defined in the context of a ServiceInterface.
				Stereotypes: atpSplitable; atpVariation
				Tags:
				atp.Splitkey=event.shortName, event.variationPoint.shortLabel
				vh.latestBindingTime=blueprintDerivationTime
				xml.sequenceOffset=30

field	Field	*	aggr	This represents the collection of fields defined in the context of a ServiceInterface.
				Stereotypes: atpSplitable; atpVariation
				Tags:
				atp.Splitkey=field.shortName, field.variationPoint.shortLabel
				vh.latestBindingTime=blueprintDerivationTime
				xml.sequenceOffset=40

majorVersion	PositiveInteger	0..1	attr	Major version of the service contract. **Tags:** xml.sequenceOffset=10

method	ClientServerOperation	*	aggr	This represents the collection of methods defined in the context of a ServiceInterface.
				Stereotypes: atpSplitable; atpVariation
				Tags:
				atp.Splitkey=method.shortName, method.variationPoint.shortLabel
				vh.latestBindingTime=blueprintDerivationTime
				xml.sequenceOffset=50

minorVersion	PositiveInteger	0..1	attr	Minor version of the service contract. **Tags:** xml.sequenceOffset=20

trigger	Trigger	*	aggr	This represents the collection of triggers defined in the context of a ServiceInterface.
				Stereotypes: atpSplitable; atpVariation
				Tags:
				atp.Splitkey=trigger.shortName, trigger.variationPoint.shortLabel
				vh.latestBindingTime=blueprintDerivationTime
				xml.sequenceOffset=60

Table C.8: ServiceInterface

Class	**SmInteractsWithNmMapping**
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement

Note	This mapping represents an interaction from state management to network management.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=FCInteractions

▽

△

Class	**SmInteractsWithNmMapping**			
Base	*ARElement*, *ARObject*, *CollectableElement*, *FunctionalClusterInteractsWithFunctionalClusterMapping*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *Referrable*, *UploadableDeploymentElement*, *UploadablePackageElement*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
actionItem	StateManagementNmActionItem	0..1	ref	This reference identifies the action item with which the state management wants to interact with network management. **Tags:** atp.Status=draft
nmNetworkHandle	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 C.9: SmInteractsWithNmMapping

Class	**StateManagemenPhmErrorInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::StateManagement			

Note	This meta-class indicates that the PortPrototype that references this class is used for accepting a error submissions from the platform health management.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=StateManagementPortInterfaces			

Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*, *StateManagementErrorInterface*, *StateManagementPortInterface*, *StateManagementRequestInterface*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table C.10: StateManagemenPhmErrorInterface

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

Table C.11: StateManagementActionItem

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

Table C.12: StateManagementActionList

Class	**StateManagementEmErrorInterface**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::ApplicationDesign::PortInterface::StateManagement			

Note	This meta-class indicates that the PortPrototype that references this class is used for accepting a error submissions from the execution management.
	Tags:
	atp.Status=draft
	atp.recommendedPackage=StateManagementPortInterfaces			

Base	*ARElement*, *ARObject*, *AtpBlueprint*, *AtpBlueprintable*, *AtpClassifier*, *AtpType*, *CollectableElement*, *Identifiable*, *MultilanguageReferrable*, *PackageableElement*, *PortInterface*, *Referrable*, *StateManagementErrorInterface*, *StateManagementPortInterface*, *StateManagementRequestInterface*			
Aggregated by	ARPackage.element			
Attribute	***Type***	***Mult.***	***Kind***	***Note***
–	–	–	–	–

Table C.13: StateManagementEmErrorInterface

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

Table C.14: StateManagementNmActionItem

Class	**StateManagementSetFunctionGroupStateActionItem**			
Package	M2::AUTOSARTemplates::AdaptivePlatform::PlatformModuleDeployment::StateManagement			
Note	This meta-class represents a state management action item to set a specific state in a specific function group. **Tags:** atp.Status=draft			
Base	*ARObject*, *Identifiable*, *MultilanguageReferrable*, *Referrable*, *StateManagementActionItem*			
Aggregated by	StateManagementActionList.actionItem			
Attribute	**Type**	**Mult.**	**Kind**	**Note**
portPrototype	PPortPrototype	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:** PPortPrototypeIn ExecutableInstanceRef
setFunction GroupState	ModeDeclaration	0..1	iref	This reference identifies the funtion group step that shall become active after the action step terminates. **Tags:** atp.Status=draft **InstanceRef implemented by:** FunctionGroupStateIn FunctionGroupSetInstanceRef

Table C.15: StateManagementSetFunctionGroupStateActionItem

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

Table C.16: StateManagementSleepActionItem

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

▽

△

Class	**StateManagementStateMachineActionItem**			
startState Machine	ModeDeclarationGroup Prototype	0..1	ref	This reference identifies the state machine that shall be started when the enclosing action list item is executed. **Tags:** atp.Status=draft
stopState Machine	ModeDeclarationGroup Prototype	0..1	ref	This reference identifies the state machine that shall be stopped when the enclosing action list item is executed. **Tags:** atp.Status=draft

Table C.17: StateManagementStateMachineActionItem

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

Table C.18: StateManagementStateNotification

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

Table C.19: StateManagementStateRequest

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

Table C.20: StateManagementSyncActionItem

D History of Constraints and Specification Items

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

D.1 Constraint and Specification Item History of this document according to AUTOSAR Release R23-11

D.1.1 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
[SWS_SM_00640]	Successful verification of updated software
[SWS_SM_00641]	Block RequestState method after PrepareRollback call

▽

△

Number	Heading
[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 D.1: Added Specification Items in R23-11

D.1.2 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 D.2: Changed Specification Items in R23-11

D.1.3 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 D.3: Deleted Specification Items in R23-11

D.1.4 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 D.4: Added Constraints in R23-11

D.1.5 Changed Constraints in R23-11

none

D.1.6 Deleted Constraints in R23-11

none

D.2 Constraint and Specification Item History of this document according to AUTOSAR Release R22-11

D.2.1 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 D.5: Added Specification Items in R22-11

D.2.2 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 D.6: Changed Specification Items in R22-11

D.2.3 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 D.7: Deleted Specification Items in R22-11

D.2.4 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 D.8: Added Constraints in R22-11

D.2.5 Changed Constraints in R22-11

none

D.2.6 Deleted Constraints in R22-11

none

D.3 Constraint and Specification Item History of this document according to AUTOSAR Release R21-11

D.3.1 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

▽

△

Number	Heading
[SWS_SM_00106]	Enabling of rapid shutdown
[SWS_SM_00107]	Disabling of rapid shutdown
[SWS_SM_00202]	Reset Execution
[SWS_SM_00203]	Start update session
[SWS_SM_00204]	Persist session status
[SWS_SM_00205]	Stop update session
[SWS_SM_00206]	prepare update
[SWS_SM_00207]	prepare verify
[SWS_SM_00208]	prepare rollback
[SWS_SM_00209]	Preventing multiple update sessions
[SWS_SM_00300]	NetworkHandle Configuration
[SWS_SM_00301]	NetworkHandle Registration
[SWS_SM_00302]	NetworkHandle to FunctionGroupState
[SWS_SM_00303]	FunctionGroupState to NetworkHandle
[SWS_SM_00304]	Network Afterrun
[SWS_SM_00400]	Execution Management
[SWS_SM_00401]	Execution Management Results
[SWS_SM_00500]	Virtualized/hierarchical State Management
[SWS_SM_00501]	Virtualized/hierarchical State Management internal State
[SWS_SM_91001]	
[SWS_SM_91002]	
[SWS_SM_91003]	
[SWS_SM_91004]	
[SWS_SM_91007]	
[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]	

▽

△

Number	Heading
[SWS_SM_-CONSTR_00001]	Existence of State Management
[SWS_SM_NA]	Not applicable requirements

Table D.9: Added Specification Items "in R21-11"

D.3.2 Changed Specification Items "in R21-11"

none

D.3.3 Deleted Specification Items "in R21-11"

none

D.3.4 Added Constraints "in R21-11"

none

D.3.5 Changed Constraints "in R21-11"

none

D.3.6 Deleted Constraints "in R21-11"

none

D.4 Constraint and Specification Item History of this document according to AUTOSAR Release R20-11

D.4.1 Added Specification Items in R20-11

Number	Heading
[SWS_SM_00001]	Available Function Group (states)
[SWS_SM_00005]	Function Group Calibration Support
[SWS_SM_00006]	Function Group Calibration Support
[SWS_SM_00020]	InternalState Propagation
[SWS_SM_00021]	InternalState Influence
[SWS_SM_00100]	Prevent Shutdown due to Diagnostic Session
[SWS_SM_00101]	Diagnostic Reset
[SWS_SM_00103]	Diagnostic Reset Last Cause
[SWS_SM_00104]	Diagnostic Reset Last Cause Retrieval
[SWS_SM_00105]	Diagnostic Reset Last Cause Reset
[SWS_SM_00200]	Prevent Shutdown during to Update Session
[SWS_SM_00201]	Supervision of Shutdown Prevention
[SWS_SM_00202]	Reset Execution
[SWS_SM_00203]	Start update session
[SWS_SM_00204]	Persist session status
[SWS_SM_00205]	Stop update session
[SWS_SM_00206]	prepare update
[SWS_SM_00207]	prepare verify
[SWS_SM_00208]	prepare rollback
[SWS_SM_00300]	NetworkHandle Configuration
[SWS_SM_00301]	NetworkHandle Registration
[SWS_SM_00302]	NetworkHandle to FunctionGroupState
[SWS_SM_00303]	FunctionGroupState to NetworkHandle
[SWS_SM_00304]	Network Afterrun
[SWS_SM_00400]	Execution Management
[SWS_SM_00401]	Execution Management Results
[SWS_SM_00402]	Function Group State Change Results
[SWS_SM_00500]	Virtualized/hierarchical State Management
[SWS_SM_00501]	Virtualized/hierarchical State Management internal State
[SWS_SM_91001]	
[SWS_SM_91002]	
[SWS_SM_91003]	
[SWS_SM_91004]	

▽

△

Number	Heading
[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 D.10: Added Specification Items in R20-11

D.4.2 Changed Specification Items in R20-11

none

D.4.3 Deleted Specification Items in R20-11

none

D.4.4 Added Constraints in R20-11

none

D.4.5 Changed Constraints in R20-11

none

D.4.6 Deleted Constraints in R20-11

none

D.5 Constraint and Specification Item History of this document according to AUTOSAR Release R19-11

D.5.1 Added Specification Items in 19-11

none

D.5.2 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 D.11: Changed Specification Items in 19-11

D.5.3 Deleted Specification Items in 19-11

none

D.5.4 Added Constraints in 19-11

none

D.5.5 Changed Constraints in 19-11

none

D.5.6 Deleted Constraints in 19-11

none

D.6 Constraint and Specification Item History of this document according to AUTOSAR Release R19-03

D.6.1 Added Specification Items in 19-03

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

Table D.12: Added Specification Items in 19-03

D.6.2 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 D.13: Changed Specification Items in 19-03

D.6.3 Deleted Specification Items in 19-03

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

▽

△

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

Table D.14: Deleted Specification Items in 19-03

D.6.4 Added Constraints in 19-03

none

D.6.5 Changed Constraints in 19-03

none

D.6.6 Deleted Constraints in 19-03

none