Document Title	Specification of Health Monitoring
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	850

Document Status	Final
Part of AUTOSAR Standard	Foundation
Part of Standard Release	1.4.0

Document Change History			
Date	**Release**	**Changed by**	**Description**
2018-03-29	1.4.0	AUTOSAR Release Management	• Initial release as "draft"

Disclaimer

Table of Contents

1 Introduction and functional overview 5
1.1 Input documents and related standards and norms 5
2 Acronyms and abbreviations 5
3 Related documentation 6
4 Constraints and assumptions 7
4.1 Limitations and conditions of use 7
4.2 Applicability to car domains 7
5 Requirements Tracing 7
6 Functional specification 11
6.1 Functional Overview 11
6.1.1 Functional Interfaces 11
6.1.2 Basic concepts - Supervised Entitys, Checkpoints, Graphs, Supervision Mode 12
6.1.3 Execution of Supervision Functions 13
6.1.3.1 Alive Supervision 13
6.1.3.2 Deadline Supervision 13
6.1.3.3 Logical Supervision 13
6.1.3.4 Health Channel Supervision 14
6.1.4 Determination of Supervision Status 14
6.1.5 Determination of Actions 14
6.1.5.1 Rule Pocessing 14
6.1.5.2 Watchdog Control 14
6.1.5.3 Error Handling 15
6.1.6 Functional Decomposition 15
6.2 Execution of Supervision Functions and Determination of Supervision Results 17
6.2.1 Alive Supervision 17
6.2.1.1 Alive Supervision Configuration 18
6.2.1.2 Alive Supervision Algorithm 20
6.2.2 Deadline Supervision 21
6.2.2.1 Deadline Supervision Configuration 21
6.2.2.2 Deadline Supervision Algorithm 24
6.2.3 Logical Supervision 24
6.2.3.1 Logical Supervision Configuration 25
6.2.3.2 Logical Supervision Algorithm 28
6.3 Determination of Supervision Status 29
6.3.1 Determination of Local Supervision Status 29
6.3.2 Determination of Global Supervision Status 33
6.3.3 Effect of changing Mode 37

6.4 Determination of Actions based on Supervision Status 38
6.4.1 Concept of HealthChannel and HealthStatus 38
6.4.2 Arbitration of HealthChannels 39
6.4.2.1 Arbitration Rules 39
6.4.2.2 Conditions and LogicalExpressions. 40
6.4.2.3 Requirements of Arbitration 40
6.4.2.4 Arbitration Behavior after Initialization 41
6.4.3 Determination of ActionList 41
6.4.3.1 Triggered and Conditional Execution 42
6.4.3.2 Available ActionItems 42
6.4.3.3 Behavior of ActionList execution after Initialization 43
7 Watchdog API specification 44
7.1 Provided API 44
7.1.1 Reporting Checkpoints and Health Status 44
7.1.2 Reporting health status 44
7.1.3 Forwarding information between health monitoring components 44
7.1.4 Init / DeInit 45
7.2 Assumed API 45
7.3 Triggering error handling 45
7.4 Controlling watchdog 45
8 Configuration Parameters 45
8.1 Overall configuration 46
8.2 Mode-independent settings 48
8.2.1 Supervised Entity 48
8.2.2 Health Channel 49
8.3 Mode-dependent settings 49
8.3.1 Alive Supervision 49
8.3.2 Deadline Supervision 50
8.3.3 Logical Supervision 51
8.3.4 Global Supervision 51
8.3.5 Arbitration 52
8.3.6 Determination of Action 54

1 Introduction and functional overview

1.1 Input documents and related standards and norms

This document specifies the functionality on the Health Monitoring.

For this release, this document applies to Adaptive Platform only: alignment with Classic Platform will be done in a subsequent release.

Health Monitoring is required by [1, ISO 26262] (under the terms control flow monitoring, external monitoring facility, watchdog, logical monitoring, temporal monitoring, program sequence monitoring) and this specification is supposed to address all relevant requirements from this standard.

Health monitoring has the following error detection functions:

1. Alive Supervision - checking if Checkpoints happens with a correct frequency
2. Deadline Supervision - checking the delta time between two Checkpoints
3. Logical Supervision - checking for correct sequence of execution of Checkpoints
4. Health Status Supervision - checking if Health Status is valid

The Health Monitoring is supposed to be implemented by AUTOSAR classic platform and AUTOSAR adaptive platform. It may be implemented by other platforms as well.

The Health Monitoring requirements are specified in [2, RS HealthMonitoring].

2 Acronyms and abbreviations

The glossary below includes acronyms and abbreviations relevant to Health Monitoring that are not included in the AUTOSAR Glossary [3].

Abbreviation / Acronym:	Description:
Alive Supervision	Kind of supervision that checks if a Supervised Entity executed in a correct frequency.
Checkpoint	A point in the control flow of a Supervised Entity where the activity is reported.
Deadline Supervision	Kind of supervision that checks if the execution time between two Checkpoints is within minimum/maximum time limit.

Global Supervision Status	Status that summarizes the Local Supervision Status of all Supervised Entities.
Health Status	A set of states that are relevant to the supervised software (e.g. a Voltage State, an application state, the result of a RAM monitoring algorithm).
Health Status Supervision	Kind of supervision that checks if the health indicators registered by the supervised software are within the tolerances/limits.
Logical Supervision	Kind of online supervision of software that checks if the software (Supervised Entity or set of Supervised Entities) is executed in the sequence defined by the programmer (by the developed code).
Local Supervision Status	Status that represents the current result of Alive Supervision, Deadline Supervision and Logical Supervision of a single Supervised Entity.
Supervised Entity	A software entity which is included in the supervision. A Supervised Entity denotes a collection of Checkpoints within a software component. There may be zero, one or more Supervised Entities in a Software Component. A Supervised Entity may be instantiated multiple times, in which case each instance is independently supervised.
Supervision Mode	An overall state of a microcontroller or virtual machine. Modes are mutually exclusive and all Supervised Entities are in the same Supervision Mode. A mode can be e.g. Startup, Shutdown, Low power.
SE	Supervised Entity.

Table 2.1: Acronyms

3 Related documentation

References

[1] ISO 26262 (Part 1-10) – Road vehicles – Functional Safety, First edition
http://www.iso.org

[2] Requirements on Health Monitoring
AUTOSAR_RS_HealthMonitoring

[3] Glossary
AUTOSAR_TR_Glossary

4 Constraints and assumptions

4.1 Limitations and conditions of use

The specification status is set to "draft" in release 1.4.0.

4.2 Applicability to car domains

No restrictions.

5 Requirements Tracing

Requirement	Description	Satisfied by
[RS_HM_09028]	Health Monitoring shall support multiple watchdogs	[SWS_HM_00451]
[RS_HM_09125]	Health Monitoring shall provide an Alive Supervision	[SWS_HM_00074] [SWS_HM_00076] [SWS_HM_00077] [SWS_HM_00078] [SWS_HM_00083] [SWS_HM_00098] [SWS_HM_00115] [SWS_HM_00117] [SWS_HM_00200] [SWS_HM_00201] [SWS_HM_00202] [SWS_HM_00203] [SWS_HM_00204] [SWS_HM_00205] [SWS_HM_00206] [SWS_HM_00207] [SWS_HM_00208] [SWS_HM_00209] [SWS_HM_00213] [SWS_HM_00214] [SWS_HM_00215] [SWS_HM_00216] [SWS_HM_00217] [SWS_HM_00218]

Requirement	Description	Satisfied by
		[SWS_HM_00221] [SWS_HM_00268] [SWS_HM_00269] [SWS_HM_00285] [SWS_HM_00286] [SWS_HM_00291] [SWS_HM_00300] [SWS_HM_00387] [SWS_HM_00440] [SWS_HM_00441] [SWS_HM_00455] [SWS_HM_00456]
[RS_HM_09159]	Health Monitoring shall be able to report supervision errors.	[SWS_HM_00068] [SWS_HM_00069] [SWS_HM_00079] [SWS_HM_00449] [SWS_HM_00450]
[RS_HM_09163]	Health Monitoring shall provide configurable tolerances for detected errors and configurable delays of error reactions.	[SWS_HM_00077] [SWS_HM_00117] [SWS_HM_00202] [SWS_HM_00203] [SWS_HM_00204] [SWS_HM_00205] [SWS_HM_00206] [SWS_HM_00215] [SWS_HM_00216] [SWS_HM_00219] [SWS_HM_00220] [SWS_HM_00300]
[RS_HM_09169]	Health Monitoring shall be able to trigger microcontroller reset.	[SWS_HM_00072]
[RS_HM_09222]	Health Monitoring shall provide a Logical Supervision	[SWS_HM_00076] [SWS_HM_00077] [SWS_HM_00078] [SWS_HM_00117] [SWS_HM_00200] [SWS_HM_00201] [SWS_HM_00202] [SWS_HM_00203] [SWS_HM_00204] [SWS_HM_00205] [SWS_HM_00206] [SWS_HM_00207] [SWS_HM_00208] [SWS_HM_00209] [SWS_HM_00213] [SWS_HM_00214] [SWS_HM_00215] [SWS_HM_00216] [SWS_HM_00217] [SWS_HM_00218] [SWS_HM_00221] [SWS_HM_00252] [SWS_HM_00268] [SWS_HM_00269]

Requirement	Description	Satisfied by
		[SWS_HM_00271] [SWS_HM_00273] [SWS_HM_00285] [SWS_HM_00286] [SWS_HM_00291] [SWS_HM_00295] [SWS_HM_00296] [SWS_HM_00297] [SWS_HM_00300] [SWS_HM_00331] [SWS_HM_00387] [SWS_HM_00440] [SWS_HM_00441] [SWS_HM_00455] [SWS_HM_00456]
[RS_HM_09226]	Health Monitoring shall be able to wrongly trigger the serviced watchdogs.	[SWS_HM_00451]
[RS_HM_09235]	Health Monitoring shall provide a Deadline Supervision	[SWS_HM_00076] [SWS_HM_00077] [SWS_HM_00078] [SWS_HM_00117] [SWS_HM_00200] [SWS_HM_00201] [SWS_HM_00202] [SWS_HM_00203] [SWS_HM_00204] [SWS_HM_00205] [SWS_HM_00206] [SWS_HM_00207] [SWS_HM_00208] [SWS_HM_00209] [SWS_HM_00213] [SWS_HM_00214] [SWS_HM_00215] [SWS_HM_00216] [SWS_HM_00217] [SWS_HM_00218] [SWS_HM_00221] [SWS_HM_00228] [SWS_HM_00229] [SWS_HM_00268] [SWS_HM_00269] [SWS_HM_00285] [SWS_HM_00286] [SWS_HM_00291] [SWS_HM_00294] [SWS_HM_00299] [SWS_HM_00300] [SWS_HM_00354] [SWS_HM_00387] [SWS_HM_00440] [SWS_HM_00441] [SWS_HM_00455] [SWS_HM_00456]

Requirement	Description	Satisfied by
[RS_HM_09244]	Health Monitoring shall support timeout watchdogs.	[SWS_HM_00073] [SWS_HM_00075] [SWS_HM_00451]
[RS_HM_09245]	Health Monitoring shall support window watchdogs.	[SWS_HM_00451]
[RS_HM_09246]	Health Monitoring shall support question-answer watchdogs.	[SWS_HM_00451]
[RS_HM_09247]	Health Monitoring shall support modes of the hardware watchdogs.	[SWS_HM_00451]
[RS_HM_09248]	Health Monitoring shall support different watchdog realizations.	[SWS_HM_00451]
[RS_HM_09251]	Health Monitoring shall be able to request a restart a Supervised entity.	[SWS_HM_00071]
[RS_HM_09253]	Health Monitoring shall support mode-dependent behavior of Supervised Entities and it shall support the supervision on the transitions between Checkpoints belonging different Supervision Modes.	[SWS_HM_00139] [SWS_HM_00182] [SWS_HM_00207] [SWS_HM_00208] [SWS_HM_00209] [SWS_HM_00291] [SWS_HM_00315] [SWS_HM_00316]
[RS_HM_09254]	Health Monitoring shall provide an interface to Supervised Entities to report the currently reached Checkpoint.	[SWS_HM_00447]
[RS_HM_09255]	Health Monitoring shall provide a Health Channel Supervision	[SWS_HM_00051] [SWS_HM_00052] [SWS_HM_00053] [SWS_HM_00054] [SWS_HM_00055] [SWS_HM_00056] [SWS_HM_00057] [SWS_HM_00058] [SWS_HM_00059] [SWS_HM_00060] [SWS_HM_00062] [SWS_HM_00063] [SWS_HM_00064] [SWS_HM_00065] [SWS_HM_00066] [SWS_HM_00067] [SWS_HM_00080] [SWS_HM_00455] [SWS_HM_00456]
[RS_HM_09257]	Health Monitoring shall provide an interface to Supervised Entities for report their health status.	[SWS_HM_00050] [SWS_HM_00448]

6 Functional specification

6.1 Functional Overview

This section presents black-box functional overview the Health Monitoring. It does not define any requirements nor details on the functionality.

6.1.1 Functional Interfaces

The Health Monitoring supervises the execution of a configurable number of Supervised Entitys and it also supervises their Health Status. When it detects a violation of the configured temporal and/or logical constraints on program execution or a violation of the configured health constraints, it triggers the appropriate error handlers. Health Monitoring controls also the Watchdogs correspondingly, see Figure 6.1.

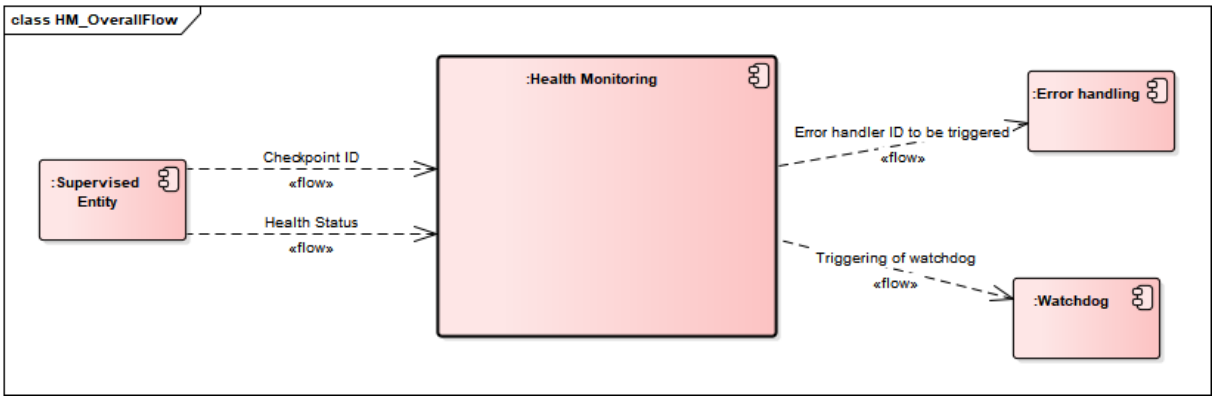

Figure 6.1: Scope of Health Monitoring

The Health Monitoring function can be split as a daisy chain. Each Health Monitoring instance has the same interface to Supervised Entitys, Error handling and Watchdog. In addition, the interface between the instances of Health Monitoring is standardized as well - it carries the results of Health Monitoring as well as "raw data" (Checkpoint IDs, Health Status together with necessary context information). Each instance adds some context-specific data to Checkpoints (e.g. process/task id).

In the example below (Figure 6.2), there are three instances of Health Monitoring, each having different usage scenarios.

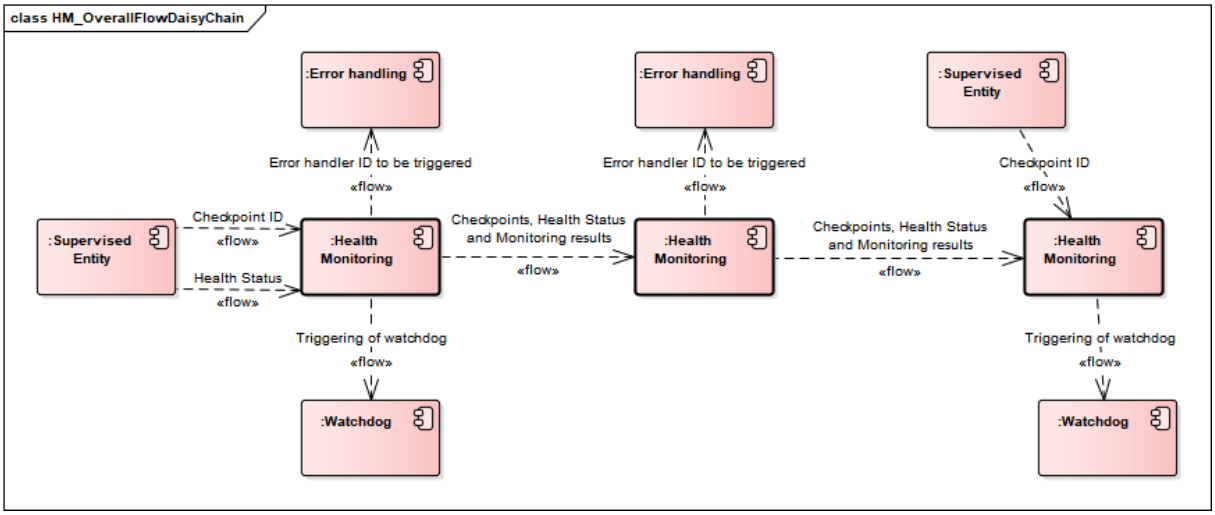

Figure 6.2: Scope of `Health Monitoring` Daisy Chain example

The data exchanged between `Health Monitoring` instances is configurable.

These are known use cases for `Health Monitoring` instances:

- The first instance is typically the same process/executable/application as the `Supervised Entity`.
- Further instance(s) can be realized as services/daemons on the microcontroller
- Further or final instance can be realized on a remote machine.

6.1.2 Basic concepts - `Supervised Entitys`, `Checkpoints`, `Graphs`, `Supervision Mode`

The `Health Monitoring` supervises the execution of software. The logical units of supervision are `Checkpoints` that belong to `Supervised Entitys`. There is no fixed relationship between `Supervised Entitys` and the architectural building blocks software, but typically a `Supervised Entity` may represent one software component.

The `Checkpoints` and Transitions between the `Checkpoints` form a Graph. The `Checkpoints` of a graph can belong to the same `Supervised Entity` or to different `Supervised Entitys`.

A `Graph` may have one or more initial `Checkpoints` and one or more final `Checkpoints`. Any sequence of starting with any `Initial Checkpoint` and finishing with any `Final Checkpoint` is correct (assuming that the checkpoints belong to the same `Graph`). After the final `Checkpoint`, any initial `Checkpoint` can be reported.

At runtime, `Health Monitoring` verifies if the configured Graphs are executed. This is called `Logical Supervision`. `Health Monitoring` verifies also the timing of `Checkpoints` and Transitions. The mechanism for periodic `Checkpoints` is called

Alive Supervision and for aperiodic Checkpoints it is called Deadline Supervision.

The granularity of Checkpoints is not fixed by the Health Monitoring. Few coarse-grained Checkpoints limit the detection abilities of the Health Monitoring. For example, for an application with only one Checkpoint the Health Monitoring is only capable of detecting that this application (or one part of this application) is cyclically running and check the timing constraints. In contrast, if that application has Checkpoints at each block and branch, the Health Monitoring may also detect failures in the control flow of that application. Fine granularity of Checkpoints causes a complex and large configuration of the Health Monitoring.

Health Monitoring allows the definition of different Supervision Modes. Different behavior of supervision functions can be configured for each Supervision Mode.

6.1.3 Execution of Supervision Functions

Health Monitoring offers Alive Supervision, Deadline Supervision, Logical Supervision and Health Channel Supervision. All supervision functions can be invoked independently.

6.1.3.1 Alive Supervision

Periodic Supervised Entitys have constraints on the number of times they are executed within a given time span. By means of Alive Supervision, The Health Monitoring checks periodically if the Checkpoints of a Supervised Entity have been reached within the given limits. This means that Health Monitoring checks if a Supervised Entity is run not too frequently or not too rarely.

6.1.3.2 Deadline Supervision

Non-cyclic Supervised Entitys have individual constraints on the timing between two Checkpoints. By means of Deadline Supervision, Health Monitoring checks the time span of transitions between two Checkpoints of a Supervised Entity. This means that Health Monitoring checks if some steps in a Supervised Entity take a time that is within the configured minimum and maximum limits.

6.1.3.3 Logical Supervision

Logical Supervision is a fundamental technique for checking the correct execution of embedded system software. Please refer to the safety standards (IEC 61508

or ISO26262) when Logical Supervision is required. Logical Supervision focuses on control flow errors, which cause a divergence from the valid (i.e. coded/-compiled) program sequence during the error-free execution of the application. An incorrect control flow occurs if one or more program instructions are processed either in the incorrect sequence or are not even processed at all. Control flow errors can lead to data corruption, microcontroller resets, or fail-silence violations.

For the control flow graph this implies that every time the Supervised Entity reports a new Checkpoint, it must be verified that there is a Transition configured between the previous Checkpoint and the reported one.

6.1.3.4 Health Channel Supervision

Using Health Channel Supervision the system integrator can hook external and debounced supervision results to the Health Monitoring. The platform integrator can create rules to resolve the dependencies between these supervision results and derive at actions when these rules evaluate to true or false. External supervision can be routines like RAM test, ROM test, kernel status, Voltage monitoring etc.

6.1.4 Determination of Supervision Status

Based on the results of the Alive, Deadline and Logical supervision functions, the Local Supervision Status of Supervised Entitys and a Global Supervision Status is calculated. Each status is determined by a state machine.

The Local Supervision Status is calculated for each Supervised Entity and a Global Supervision Status is calculated based on the Local Supervision Status of all Supervised Entitys.

6.1.5 Determination of Actions

6.1.5.1 Rule Pocessing

Based on the results of supervision functions, Health Monitoring determines the corresponding reaction.

6.1.5.2 Watchdog Control

Health Monitoring controls the hardware watchdog. When the Supervised Entitys are not correctly evaluated due to a programming error or memory failure in the watchdog protocol itself, it may still happen that the watchdog protocol erroneously sets the triggering condition and no watchdog reset will be caused. Therefore, it may

be needed to use Supervised Entitys and Checkpoints (or some other internal supervision mechanism) within watchdog protocol itself, while avoiding recursion in watchdog protocol.

6.1.5.3 Error Handling

Depending on the Local Supervision Status of each Supervised Entity and on the Global Supervision Status, the Health Monitoring initiates a number of mechanisms to recover from supervision failures. These range from local error recovery within the Supervised Entity to a global reset of the ECU.

6.1.6 Functional Decomposition

The Health Monitoring has the following logical steps:

1. Execution of all Supervision Functions - see 6.2
2. Determination of Supervision Status - see 6.3
3. Determiniation of Actions - see 6.4

The behavior of Health Monitoring is mode-dependent (see description of supervision mode in 6.1.2 and [2]).

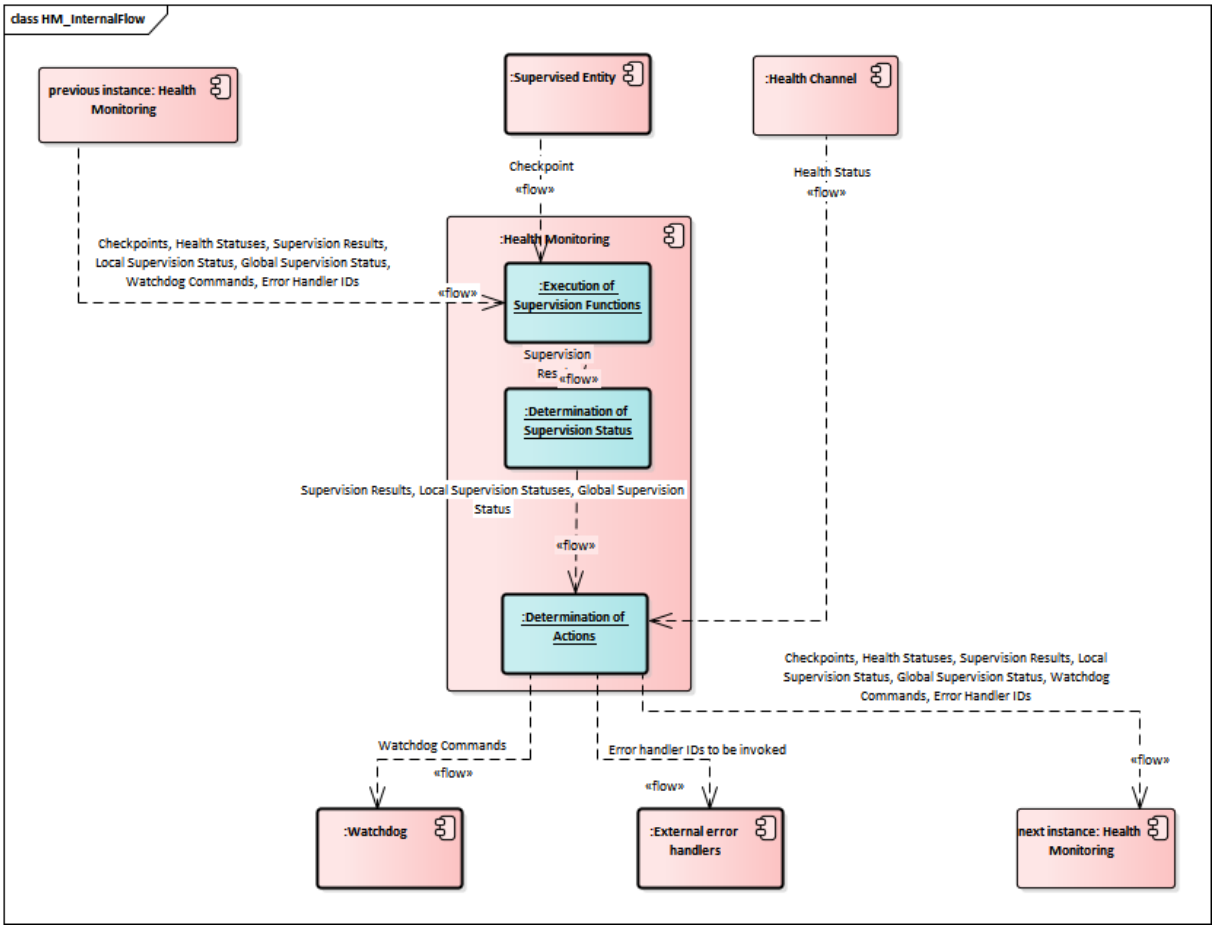

Figure 6.3: Main functions of `Health Monitoring`

The Alive, Deadline and Logical supervision mechanisms supervise each `Supervised Entity`. A `Supervised Entity` may have between one and three mechanisms enabled. Based on the results from each of enabled mechanisms, the status of the `Supervised Entity` (called Local Status) is computed.

When the status of each `Supervised Entity` is determined, then based on each `Local Supervision Status`, the status of all `Supervised Entitys` is determined (called `Global Supervision Status`).

Based on the results of Supervisions Functions (correct/incorrect), the Local Status of each `Supervised Entity` is determined by means of the `Local Supervision Status` state machine (6.11).

Based on `Local Supervision Status` of each `Supervised Entity`, the `Global Supervision Status` is determined by means of `Global Supervision Status` state machine (6.12).

Based on the `Global Supervision Status`, the error handling and watchdog handling take place (see Chapter 6.4).

6.2 Execution of Supervision Functions and Determination of Supervision Results

Supervised Entitys are the units of supervision for the Health Monitoring. Each Supervised Entitys (SupervisedEntity) can be supervised by a different supervision function or a combination of them.

The following three supervision functions are executed at this stage:

- Alive Supervision (see 6.2.1)
- Deadline Supervision (see 6.2.2)
- Logical Supervision (see 6.2.3)

Each of three Supervision Functions results with a list of Results of Supervision Function for each Supervised Entity (SupervisedEntity) (highlighted in Blue on Figure 6.3), where each Result is either correct or incorrect.

At Health Monitoring initialization, all the Results are set to correct. This means that for every Supervised Entity (SupervisedEntity) there are three partial results (one from Alive Supervision, one from Deadline Supervision and one from Logical Supervision).

In a given mode, each Supervised Entity (SupervisedEntity) may have zero, one or more Alive Supervisions (AliveSupervision), each having one correct/incorrect result.

In a given mode, each Supervised Entity (SupervisedEntity) may have zero, one or more Deadline Supervisions (DeadlineSupervision), each having one correct/incorrect result.

In a given mode, each Supervised Entity (SupervisedEntity) may have zero, one or more Logical Supervisions (LogicalSupervision) (i.e. graphs) configured, each having one correct/incorrect result.

In case there are zero active supervisions in a given mode, then Health Monitoring sees no EXPIRED local stati, so the watchdog trigger condition can be invoked.

6.2.1 Alive Supervision

The Alive Supervision (AliveSupervision) offers a mechanism to periodically check the execution reliability of one or several Supervised Entitys. This mechanism supports a check of cyclic timing constraints of independent Supervised Entitys.

6.2.1.1 Alive Supervision Configuration

To provide Alive Supervision (AliveSupervision), theCheckpoints and their timing constraints need to be configured. The simplest configuration for AliveSupervision is one Checkpoint without any Transitions, as shown in Figure 6.4)

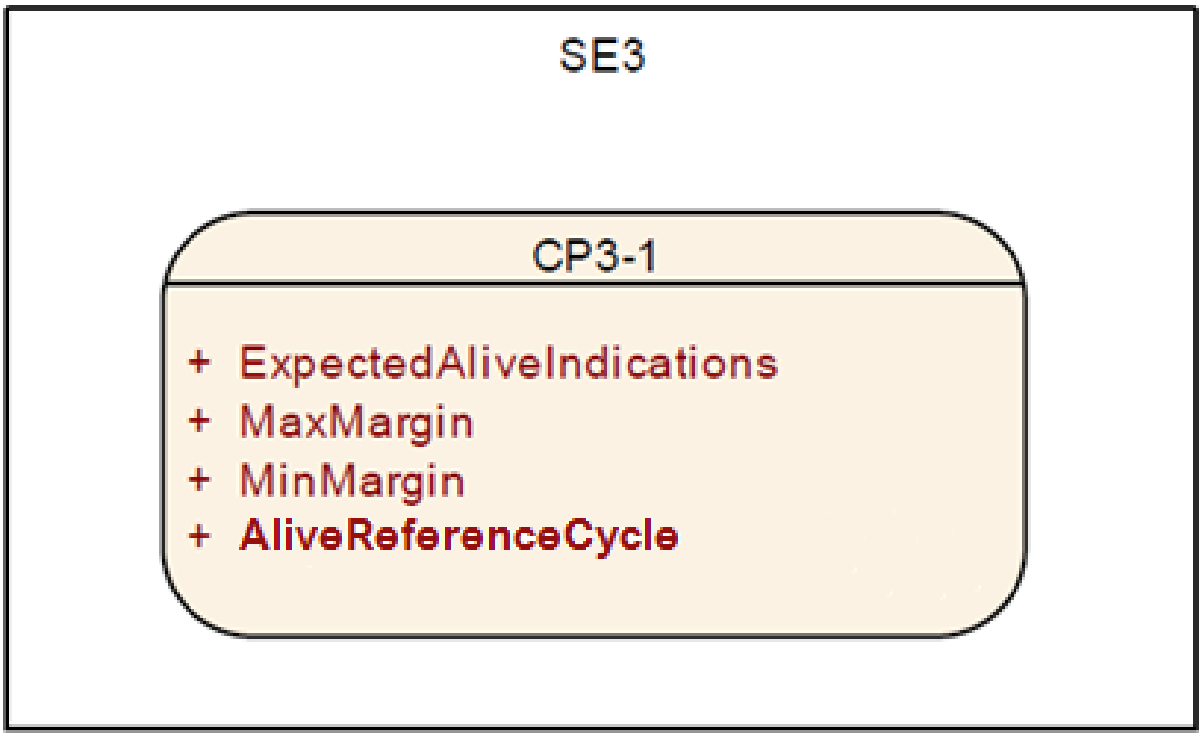

Figure 6.4: Simplest Alive Supervision Checkpoint Configuration for a given Supervision Mode

Moreover, it is also possible to have more than one Checkpoint as shown in Figure 6.5)

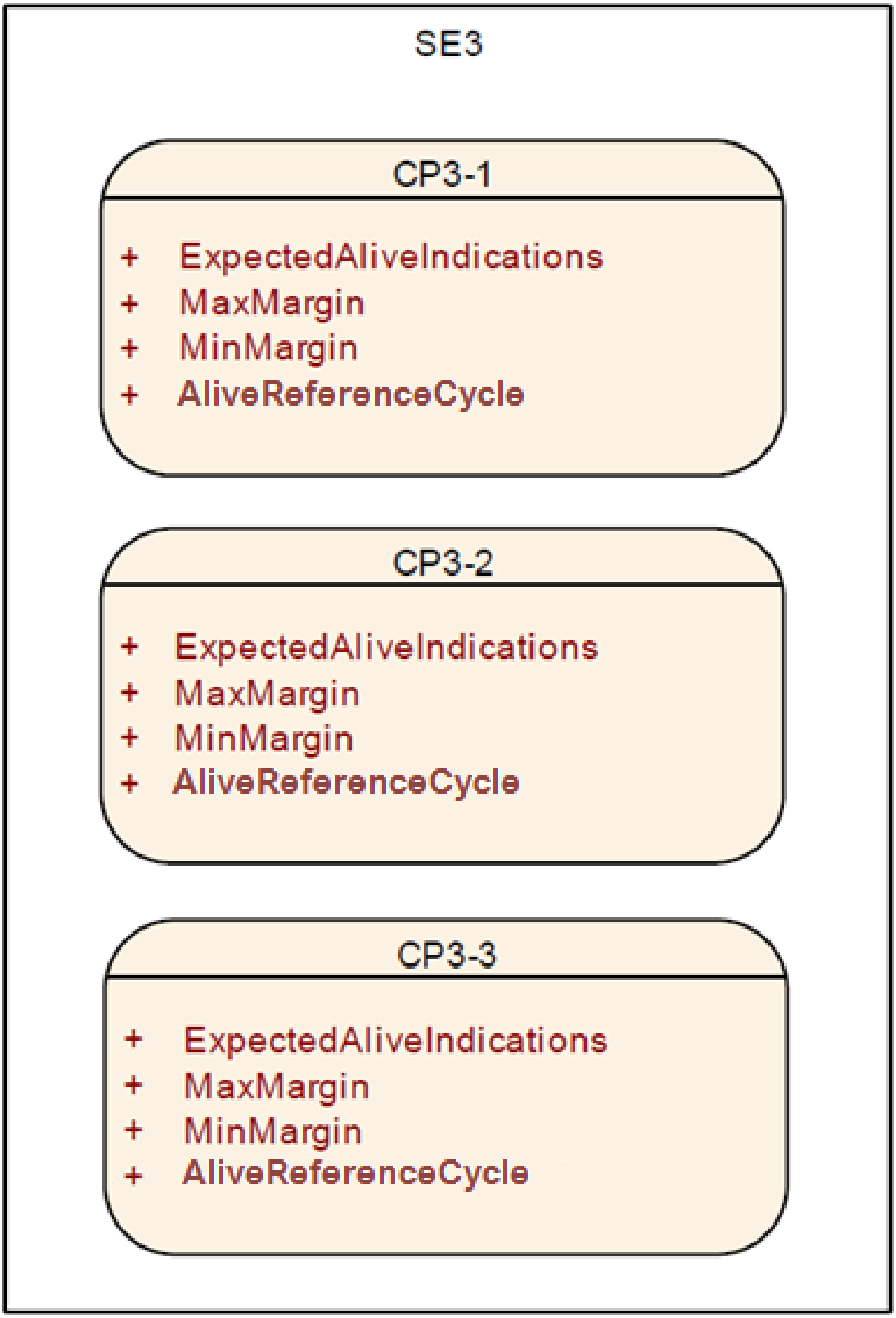

Figure 6.5: Multiple `Checkpoints` for `Alive Supervision` in one `SupervisedEntity` for a given Supervision Mode

Each Checkpoint can have its own set of AliveSupervision Parameters. Transitions are not used by AliveSupervision. Although each Checkpoint has its own parameters, it is the SupervisedEntity for which status is determined based on the frequency ofCheckpoints.

The parameters of the AliveSupervision depend on the Supervision Mode and are defined per Checkpoint (and not globally for the whole SupervisedEntity).

None, some, or all of theCheckpoints of a SupervisedEntity can be configured for AliveSupervision in a given Mode. Moreover, in each Mode the AliveSupervision options ofCheckpoints can be different.

The ExpectedAliveIndications (EAI) specifies the amount of expected alive indications from a given Checkpoint, within a fixed period of supervision cycles. The period length is defined by AliveReferenceCycle.

An acceptable negative variation (MinMargin) and acceptable positive variation (MaxMargin) can be configured.

The Health Monitoring has to support a configurable amount of independent Supervised Entitys.

6.2.1.2 Alive Supervision Algorithm

To send an Alive Indication, a Supervised Entity (SupervisedEntity) invokes the function ReportCheckpoint, which results with incrementation of an Alive Counter for the Checkpoint.

The periodic examination of the Counter of each Checkpoint of a SupervisedEntity by the Health Monitoring happens at every AliveReferenceCycle.

The Alive Reference Cycle (see AliveReferenceCycle) is the property of an AliveSupervision of a Checkpoint in a given Supervision Mode.

[SWS_HM_00098] ⌈ The Health Monitoring shall perform for each Alive Supervision (AliveSupervision) configured in the active Mode, the examination of the Alive Counter of each Checkpoint of the SupervisedEntity. The examination shall be done at the period AliveReferenceCycle of the corresponding Alive Supervision (AliveSupervision). ⌋*(RS_HM_09125)*

[SWS_HM_00074] ⌈The Health Monitoring shall examine an Alive Counter by checking if it is within the allowed tolerance (Expected - Min Margin; Expected + Max Margin) (see ExpectedAliveIndications, MinMargin, MaxMargin). ⌋*(RS_HM_09125)*

If any Checkpoint of a SupervisedEntity fails the examination, then the result of Alive Supervision for the SupervisedEntity is set to incorrect.

[SWS_HM_00115] ⌈ If the Health Monitoring detects a deviation between the counted Alive Indications and the expected amount of alive indications (including tolerance margins), for any Checkpoint of a SupervisedEntity, then Alive Supervision at

this AliveReferenceCycle for this SupervisedEntity shall be defined as incorrect. Otherwise, it shall be defined as correct. ⌋*(RS_HM_09125)*

Health Monitoring only checks the Checkpoints that are configured for the current Supervision Mode.

[SWS_HM_00083] ⌈The Health Monitoring shall not perform the examination of the Alive Counter of a Checkpoint if no corresponding Alive Supervision (AliveSupervision) is defined in the current Supervision Mode. ⌋*(RS_HM_09125)*

6.2.2 Deadline Supervision

Deadline Supervision (DeadlineSupervision) checks the timing constraints of non-cyclic Supervised Entitys. In these Supervised Entitys, a certain event happens and a following event happens within a given time span. This time span can have a maximum and minimum deadline (time window).

6.2.2.1 Deadline Supervision Configuration

For every DeadlineSupervision, twoCheckpoints connected by a Transition are configured. The Deadline is attached to the Transition from the start Checkpoint to the end Checkpoint. The simplest DeadlineSupervision configuration contains two Checkpoints and one Transition, as shown in Figure 6.6)

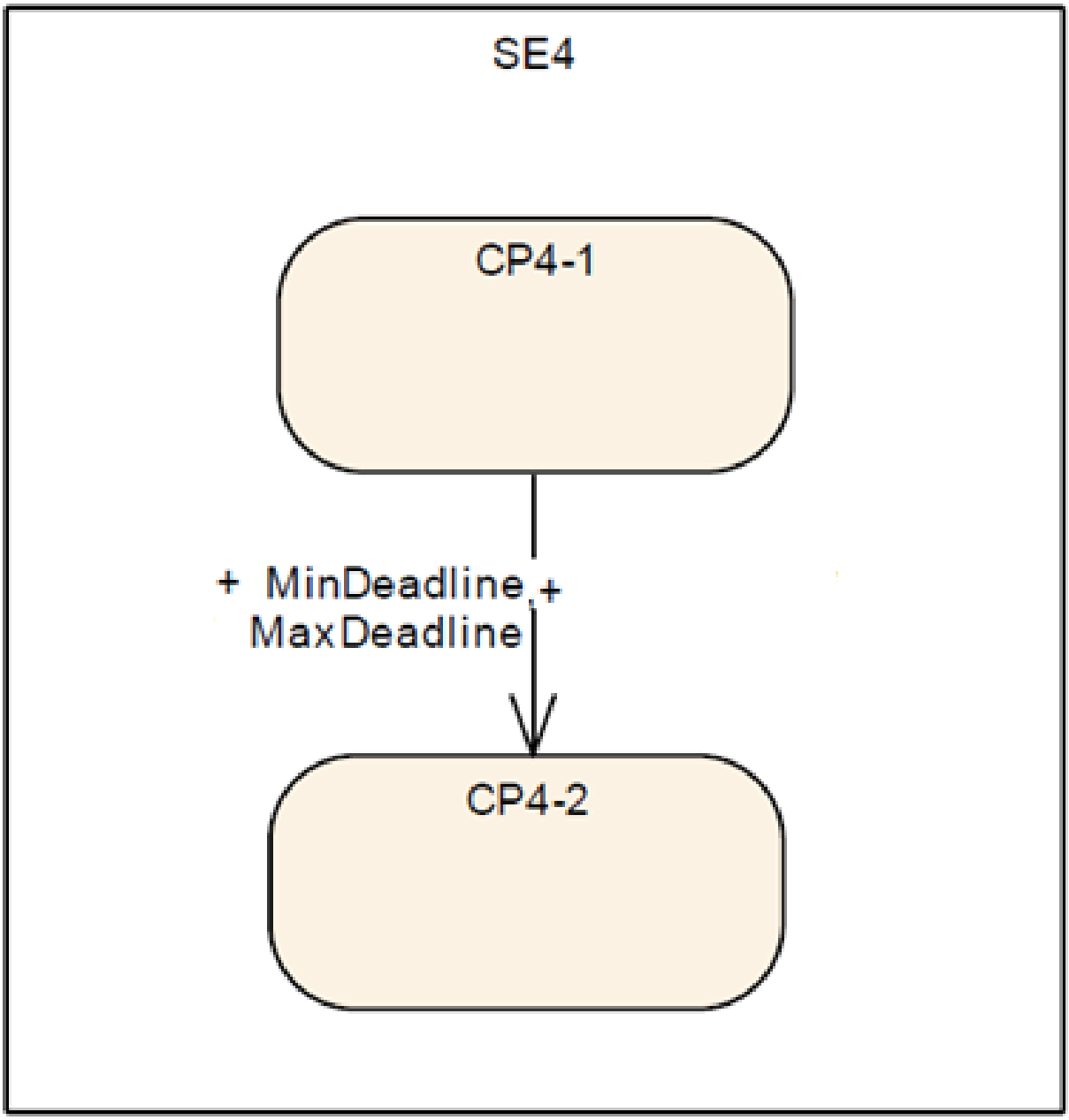

Figure 6.6: Simplest `Deadline Supervision` Configuration for a given Supervision Mode

More than one Transition can be defined in a `SupervisedEntity`. The Transitions and`Checkpoint`s do not have to form a closed graph. Since only the start and end`Checkpoint`s are considered by this Supervision Function, there can be independent graphs, as shown in Figure 6.7). Moreover, the`Checkpoint`s can be chained.

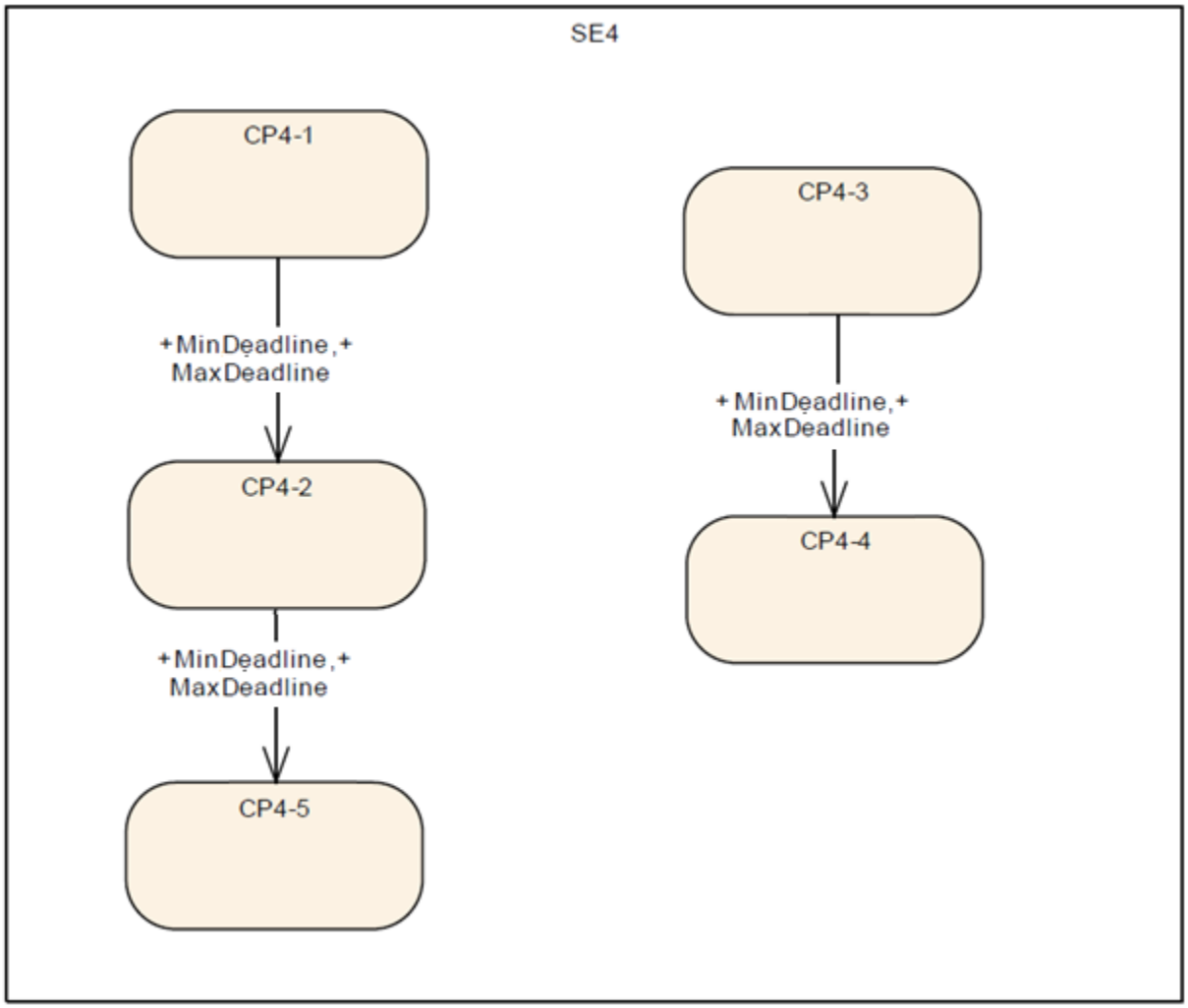

Figure 6.7: Multiple Transitions for `Deadline Supervision` in one `Supervised Entity` for a given Supervision Mode

The configuration of `DeadlineSupervision` is similar to the one of `AliveSupervision`.

The parameters of the `Deadline Supervision` (see `DeadlineSupervision`) depend on the Supervision Mode (`ModeDependentSettings`) and are defined for per a set of two`Checkpoint`s. None, some, or all of the`Checkpoint`s of a `SupervisedEntity` can be configured for `DeadlineSupervision` in a given Mode.

A `DeadlineSupervision` is defined as a set of Transitions with time constraints. A Transition is defined as two references to two`Checkpoint`s, called Deadline Start `Checkpoint` and Deadline End `Checkpoint` (DeadlineStart and DeadlineEnd, see `DeadlineSupervision`). A Transition has minimum and maximum time `MinDeadline`, `MaxDeadline`.

6.2.2.2 Deadline Supervision Algorithm

When a Deadline Start Checkpoint (i.e. the Checkpoint referenced by DeadlineStart, see DeadlineSupervision) is reached, a SupervisedEntity invokes the function ReportCheckpoint (see [SWS_HM_00447]), which results with the execution of DeadlineSupervision.

The Deadline Supervision algorithm will calculate the time expired between the Deadline Start Checkpoint and the Deadline End Checkpoint.

The calculation is performed either at the occurrence of the Deadline End Checkpoint or at the moment the elapsed time after Deadline Start Checkpoint is above the maximum limit (MaxDeadline).

[SWS_HM_00294] ⌈ If the time difference between the Deadline End Checkpoint and the Deadline Start Checkpoint is not within the minimum and the maximum limits (MinDeadline and MaxDeadline), then the result of DeadlineSupervision for this SupervisedEntity shall be defined as incorrect. Otherwise, it shall be defined as correct. ⌋*(RS_HM_09235)*

[SWS_HM_00228] ⌈ If the Deadline End Checkpoint is not reached before the maximum limit (MaxDeadline), then the result of DeadlineSupervision for this SupervisedEntity shall be defined as incorrect. ⌋*(RS_HM_09235)*

[SWS_HM_00229] ⌈ When a given Deadline Start Checkpoint is reached two or more times before the expiration of the maximum limit without reaching the corresponding Deadline End Checkpoint, this shall be considered as an error and the result of the DeadlineSupervision for this SupervisedEntity shall be considered as incorrect. ⌋*(RS_HM_09235)*

[SWS_HM_00354] ⌈ When a given Deadline End Checkpoint is reached before the occurrence of the corresponding Deadline Start Checkpoint, the function ReportCheckpoint [SWS_HM_00447] shall ignore this Checkpoint and not update the result of the Deadline Supervision for the Supervised Entity. ⌋*(RS_HM_09235)*

This means also that it is not considered as an error by DeadlineSupervision if a given Deadline End Checkpoint is reached several times in a sequence.

[SWS_HM_00299] ⌈ For any reported Checkpoint that is neither a Deadline Start Checkpoint nor a Deadline End Checkpoint , the function ReportCheckpoint (see [SWS_HM_00447]) shall ignore this Checkpoint and not update the result of the Deadline Supervision for the Supervised Entity. ⌋*(RS_HM_09235)*

6.2.3 Logical Supervision

Logical Supervision checks if the code of Supervised Entitys is executed in the correct sequence.

6.2.3.1 Logical Supervision Configuration

For every Logical Supervision (LogicalSupervision), there is a graph of Checkpoints connected by Transitions. The graph abstracts the behavior of the SupervisedEntity. There is a 1 to 1 correspondance between a Graph and the LogicalSupervision container.

In addition, a Checkpoint shall belong to maximum one Graph, overlapping Graphs are not possible.

As an example for a SupervisedEntity, let us consider the following code fragment, which contains the Checkpoints CP0-0 to CP0-6.

```
CP0-0  initialize();
CP0-1  while (subsystem is running) {
CP0-2      if (condition_A)
CP0-3          run subtask_A;
CP0-4      else
               run subtask_B;
CP0-5      run subtask_C
CP0-6  }
```

Figure 6.8: Example of Checkpoints

This SupervisedEntity can be represented by the Graph shown in Figure 6.9.

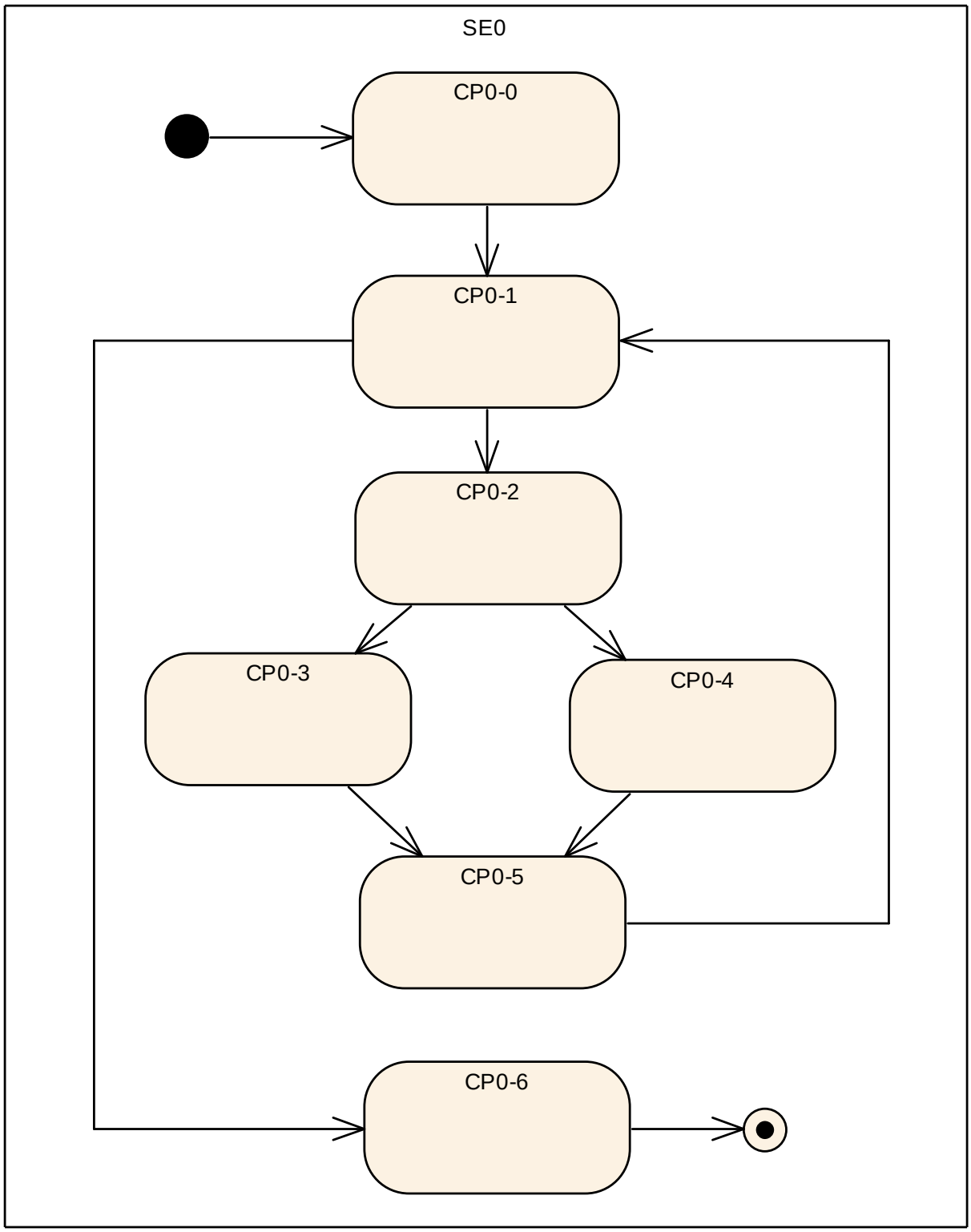

Figure 6.9: Example Control Flow Graph

A more abstract view of the `SupervisedEntity` is given by the Graph shown in Figure 6.10), where the `Checkpoint` CP0-1 represents the complete while loop.

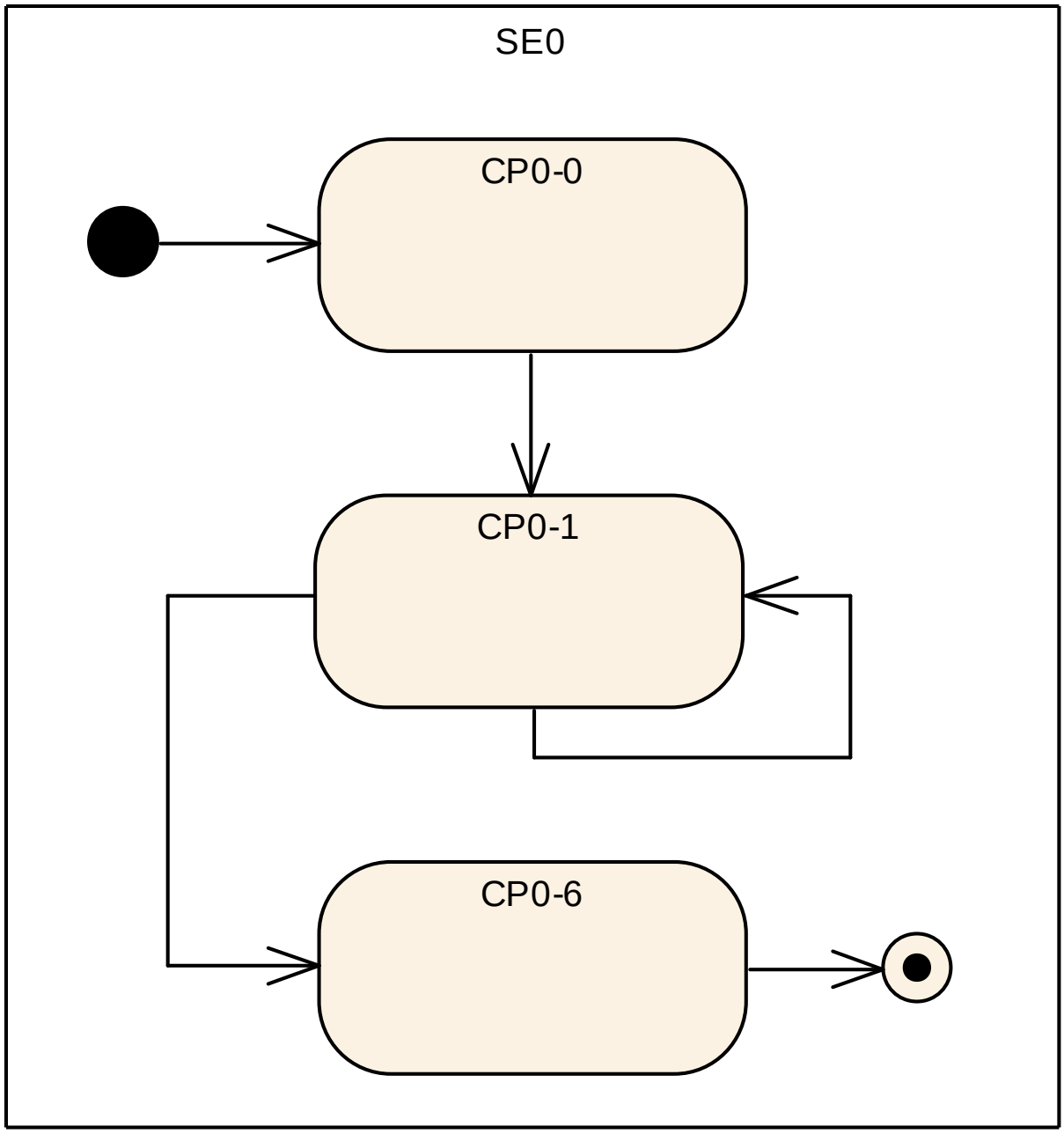

Figure 6.10: Abstracted Example Control Flow Graph

In a Graphs, Checkpoints can belong to the same SupervisedEntity or to different Supervised Entitys, no restriction is imposed. The transitions between Checkpoints in a Graph are dependent on the Supervision Mode.

The parameters of the Graphs (see LogicalSupervision) are the Transitions that are contained in a Supervision Mode (see ModeDependentSettings). Each Transition connects two Checkpoints. The Checkpoints exist irrespective if they are connected by any transitions.

6.2.3.2 Logical Supervision Algorithm

Immediately after initialization of the Health Monitoring, there has not yet been a Checkpoint reported, i.e. all the Supervised Entitys are passive. Each Graph is considered as inactive.

Each Graph represents one LogicalSupervision, but it may spans across possibly several Supervised Entitys. Assuming N Graphs that cross a Supervised Entity, this implies N results from the LogicalSupervision for the SupervisedEntity

[SWS_HM_00271] ⌈The Health Monitoring shall mantain the activity status of each Graph. ⌋*(RS_HM_09222)*

[SWS_HM_00296] ⌈At the initialization, the Health Monitoring shall consider each Graph as inactive. ⌋*(RS_HM_09222)*

Each Graph may have one or more Initial Checkpoints. Initial Checkpoints are Checkpoints with which a Graph can start.

To notify reaching a Checkpoint, a SupervisedEntity invokes the function ReportCheckpoint (see [SWS_HM_00447]), which results with execution of Logical Supervision algorithm.

Because a Checkpoint can belong to only one Graph, the function ReportCheckpoint [SWS_HM_00447] is able to identify to which Graph a Checkpoint belongs.

[SWS_HM_00295] ⌈ The function ReportCheckpoint [SWS_HM_00447] shall identify to which one Graph a reached Checkpoint belongs. ⌋*(RS_HM_09222)*

If a Graph is active, the function ReportCheckpoint [SWS_HM_00447] checks for each new Checkpoint if the Transition between the stored Checkpoint and the newly reported Checkpoint is allowed.

[SWS_HM_00252] ⌈ The function ReportCheckpoint [SWS_HM_00447] shall verify if the reported Checkpoint belonging to a Graph is a correct one by the following checks:

1. If the Graph of the reported Checkpoint is inactive, then:

   a. If the Checkpoint is an Initial Checkpoint (see LogicalSuperision), then the result of this Logical Supervision within the SupervisedEntity of the reported Checkpoint is correct, otherwise incorrect.

2. Else (i.e. the Graph is active), then:

   a. If the reported Checkpoint is a successor of the stored Checkpoint within the Graph of the reported Checkpoint (this means there is a Transition with Source and Destination), then the result of this Logical Supervision for SupervisedEntity of the reported Checkpoint is correct, otherwise incorrect. The above requirement means that in case of an incorrect transition, the

SupervisedEntity that is considered as erroneous is the one that reported the incorrect Checkpoint.

⌋(*RS_HM_09222*)

If a Checkpoint is one of the initial Checkpoints of a Graph, then the Graph is set as active.

Note that if a Graph contains multiple initial Checkpoints, either of them are allowed to be entered when the Graph is inactive: when an initial Checkpoint is reported, the corresponding Graph becomes active, so another initial Checkpoint is allowed only if a Transition is configured from the first Checkpoint to the second one as a Graph can have only one active checkpoint at a specific time.

[SWS_HM_00331] ⌈ If the result of the Logical Supervision triggered by ReportCheckpoint [SWS_HM_00447] is correct and the Checkpoint is defined as a final one, then the function ReportCheckpoint [SWS_HM_00447] shall set Graph as inactive. After a final checkpoint, only initial checkpoints are possible. ⌋ *(RS_HM_09222)*

[SWS_HM_00297] ⌈ For any reported Checkpoint that does not belong to any Graph, the function ReportCheckpoint shall ignore it and not update the result of the Logical Supervision for the SupervisedEntity. ⌋*(RS_HM_09222)*

This is because the checkpoint may be used by other Supervision Functions (Alive or Deadline).

[SWS_HM_00273] ⌈ If the function ReportCheckpoint [SWS_HM_00447] determines that the result of the Logical Supervision for the given Checkpoint is true, and the Checkpoint is the initial one (see LogicalSupervision), then the Graph corresponding to the Checkpoint shall be considered as active. ⌋ *(RS_HM_09222)*

6.3 Determination of Supervision Status

Based on the Supervision Results determined in section 6.2, the Local Supervision Status and Global Supervision Status (see LocalSupervision and GlobalSupervision) is determined.

6.3.1 Determination of Local Supervision Status

The Local Supervision Status state machine determines the status of the SupervisedEntity. This is done based on the following:

1. Previous value of the Local Supervision Status,
2. Current values of: result of AliveSupervision, result of DeadlineSupervision, result of LogicalSupervision.

The change in the Local Status state machine is done at the time defined in 6.2.1.2, 6.2.2.2 and 6.2.3.2. The state machine is initialized at the initialization of the Health Monitoring.

[SWS_HM_00200] ⌈The `Health Monitoring` shall track the `Local Supervision Status` of each `SupervisedEntity`. ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

Figure 6.11. shows the state machine for `Local Supervision Status` of a `SupervisedEntity` with all possible states.

[SWS_HM_00441] ⌈ The `Health Monitoring` shall have the local statuses defined in Figure 6.11. ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

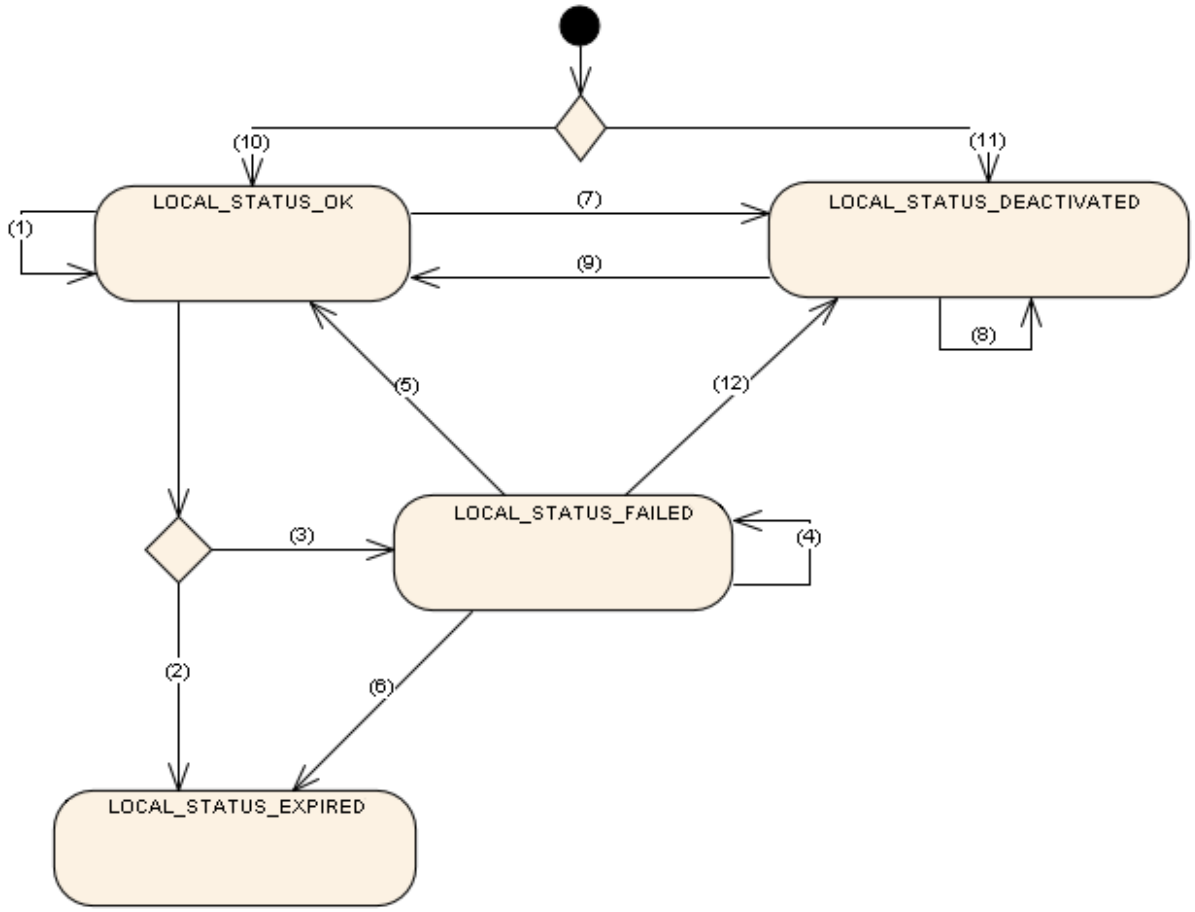

Figure 6.11: `Local Supervision Status`

For the transitions between the states of the `Local Supervision Status` the following rules apply:

[SWS_HM_00268] ⌈ If Health Monitoring successfully initialized, then for each `SupervisedEntity` that is referenced from the Initial Supervision Mode (InitialMode) (i.e. each `SupervisedEntity` that is activated in the initial mode), the Health Monitoring shall set the `Local Supervision Status` for this `SupervisedEn-`

tity to LOCAL_STATUS_OK. (see Transition 10 in Figure 6.11). ⌋(*RS_HM_09222, RS_HM_09125, RS_HM_09235*)

[SWS_HM_00269] ⌈If Health Monitoring successfully initialized, then for each SupervisedEntity that is not referenced from the Initial Mode (InitialMode), the Health Monitoring shall set the Local Supervision Status for this SupervisedEntity to LOCAL_STATUS_DEACTIVATED (see Transition 11 in Figure 6.11) ⌋(*RS_HM_09222, RS_HM_09125, RS_HM_09235*)

If Health Monitoring successfully initialized and the parameter InitialMode of this SupervisedEntity in InitialMode is not configured to LOCAL_STATUS_OK then the Health Monitoring shall set the Local Supervision Status for this SupervisedEntity to LOCAL_STATUS_DEACTIVATED. (see Transition 11 in Figure 6.11).

[SWS_HM_00201] ⌈If all values in three sets of results of Supervision (results of AliveSupervision, results of DeadlineSupervision, results of LogicalSupervision) for the SupervisedEntity are correct and the SupervisedEntity was in Local Supervision Status LOCAL_STATUS_OK, then the Health Monitoring shall leave the SupervisedEntity in the Local Supervision Status LOCAL_STATUS_OK (see Transition 1 in Figure 6.11). ⌋(*RS_HM_09222, RS_HM_09125, RS_HM_09235*)

[SWS_HM_00202] ⌈If the SupervisedEntity was in Local Supervision Status LOCAL_STATUS_OK AND:

1. (At least one result of AliveSupervision of the SupervisedEntity is incorrect and a Failure Tolerance of zero is configured (see configuration parameter FailedSupervisionCyclesTolerance) OR
2. If the result of at least one DeadlineSupervision of the SupervisedEntity or the result of at least one Logical Supervision of the SupervisedEntity is incorrect),

THEN the Health Monitoring shall change the Local Supervision Status to LOCAL_STATUS_EXPIRED (see Transition (2) in Figure 6.11). ⌋(*RS_HM_09222, RS_HM_09125, RS_HM_09235, RS_HM_09163*)

The below requirements shows the important difference of AliveSupervision versus DeadlineSupervision and LogicalSupervision: the AliveSupervision has an error tolerance for failed reference cycles.

[SWS_HM_00203] ⌈If the Supervised Entity was in Local Supervision Status LOCAL_STATUS_OK AND:

1. (If the result of at least one AliveSupervision of the SupervisedEntity is incorrect and a Failure Tolerance greater than zero is configured (see configuration parameter FailedSupervisionCyclesTolerance) AND
2. If all the results of DeadlineSupervision of the SupervisedEntity and all results of Logical Supervision of the SupervisedEntity are correct),

THEN the Health Monitoring shall change the Local Supervision Status to LOCAL_STATUS_FAILED and increment the counter for failed supervision reference cycles (see Transition (3) in Figure 6.11). ⌋(*RS_HM_09222*, *RS_HM_09125*, *RS_HM_09235*, *RS_HM_09163*)

[SWS_HM_00204] ⌈ If the SupervisedEntity was in Local Supervision Status LOCAL_STATUS_FAILED AND:

1. (If the result of at least one AliveSupervision is incorrect and the counter for failed supervision reference cycles does not exceed the configured Failure Tolerance (see parameter FailedSupervisionCyclesTolerance) AND
2. If all the results of Deadline Supervisions of the SupervisedEntity and all the result of LogicalSupervision of the SupervisedEntity are correct),

THEN the Health Monitoring shall keep the Local Supervision Status in LOCAL_STATUS_FAILED and increment the counter for failed supervision reference cycles (see Transition (4) in Figure 6.11). ⌋(*RS_HM_09222*, *RS_HM_09125*, *RS_HM_09235*, *RS_HM_09163*)

[SWS_HM_00300] ⌈If the SupervisedEntity was in Local Supervision Status LOCAL_STATUS_FAILED AND:

1. (If all the results of AliveSupervision of the SupervisedEntity are correct and the counter for failed supervision reference cycles is > 1) AND
2. If all the result of DeadlineSupervision of the SupervisedEntity and all the result of Logical Supervision of the SupervisedEntity are correct),

THEN the Health Monitoring shall keep the Local Supervision Status in LOCAL_STATUS_FAILED and decrement the counter for failed supervision reference cycles (see Transition (4) in Figure 6.11). ⌋(*RS_HM_09222*, *RS_HM_09125*, *RS_HM_09235*, *RS_HM_09163*)

[SWS_HM_00205] ⌈ If the SupervisedEntity was in Local Supervision Status LOCAL_STATUS_FAILED AND:

1. (If all the results of AliveSupervision of the SupervisedEntity are correct and the counter for failed supervision reference cycles equals 1) AND
2. If all the results of Deadline Supervisions of the SupervisedEntity and all the results of Logical Supervision of the SupervisedEntity are correct),

THEN the Health Monitoring shall change the Local Supervision Status to LOCAL_STATUS_OK and decrement the counter for failed supervision reference cycles (see Transition (5) in Figure 6.11). ⌋(*RS_HM_09222*, *RS_HM_09125*, *RS_HM_09235*, *RS_HM_09163*)

[SWS_HM_00206] ⌈ If the SupervisedEntity was in Local Supervision Status LOCAL_STATUS_FAILED AND:

1. (If at least one result of AliveSupervision is incorrect and the counter for failed supervision reference cycles exceeds the configured Failure Tolerance (see configuration parameter FailedSupervisionCyclesTolerance) OR
2. If at least one result of DeadlineSupervision of the SupervisedEntity or at least one the result of Logical Supervision of the SupervisedEntity is incorrect),

THEN the Health Monitoring shall change the Local Supervision Status to LOCAL_STATUS_EXPIRED (see Transition (6) in Figure 6.11). ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235, RS_HM_09163)*

[SWS_HM_00207] ⌈ If the SupervisedEntity was in Local Supervision Status LOCAL_STATUS_OK and there is a switch to a mode which deactivates the SupervisedEntity, then the Health Monitoring shall change the Local Supervision Status to LOCAL_STATUS_DEACTIVATED (see Transition (7) in Figure 6.11). ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235, RS_HM_09253)*

[SWS_HM_00291] ⌈If the SupervisedEntity was in Local Supervision Status LOCAL_STATUS_FAILED and there is a switch to a mode in which the SupervisedEntity is Deactivated, then the Health Monitoring shall change the Local Supervision Status to LOCAL_STATUS_DEACTIVATED (see Transition (12) in Figure 6.11). ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235, RS_HM_09253)*

Note that the above requirement is only applicable for the LOCAL_STATUS_FAILED status, but not for LOCAL_STATUS_EXPIRED.

[SWS_HM_00208] ⌈ If the SupervisedEntity was in the Local Supervision Status LOCAL_STATUS_DEACTIVATED, the functions ReportCheckpoint [SWS_HM_00447] and the Health Monitoring shall not perform any Supervision Functions for this Supervised Entity and leave the Local Supervision Status in the state LOCAL_STATUS_DEACTIVATED. (see Transition (8) in Figure 6.11) ⌋ *(RS_HM_09222, RS_HM_09125, RS_HM_09235, RS_HM_09253)*

[SWS_HM_00209] ⌈ If the SupervisedEntity was in Local Supervision Status LOCAL_STATUS_DEACTIVATED and there is a switch to a mode in which the SupervisedEntity is active, then the Health Monitoring shall change the Local Supervision Status to LOCAL_STATUS_OK. (see Transition (9) in Figure 6.11) ⌋ *(RS_HM_09222, RS_HM_09125, RS_HM_09235, RS_HM_09253)*

6.3.2 Determination of Global Supervision Status

Based on the Local Supervision Status of all Supervised Entitys of a software subsystem, the Global Supervision Status is computed. There may be one or few Global Supervision Status on the whole software (but only one Global Supervision Status for a Classic Platform).

The Global Supervision Status has similar values as the Local Supervision Status. The main differences are the addition of the GLOBAL_STATUS_STOPPED value. Figure in Figure 6.12) shows the values and Transitions between them.

[SWS_HM_00440] ⌈ The Health Monitoring shall have the global statuses specified in Figure 6.12. ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

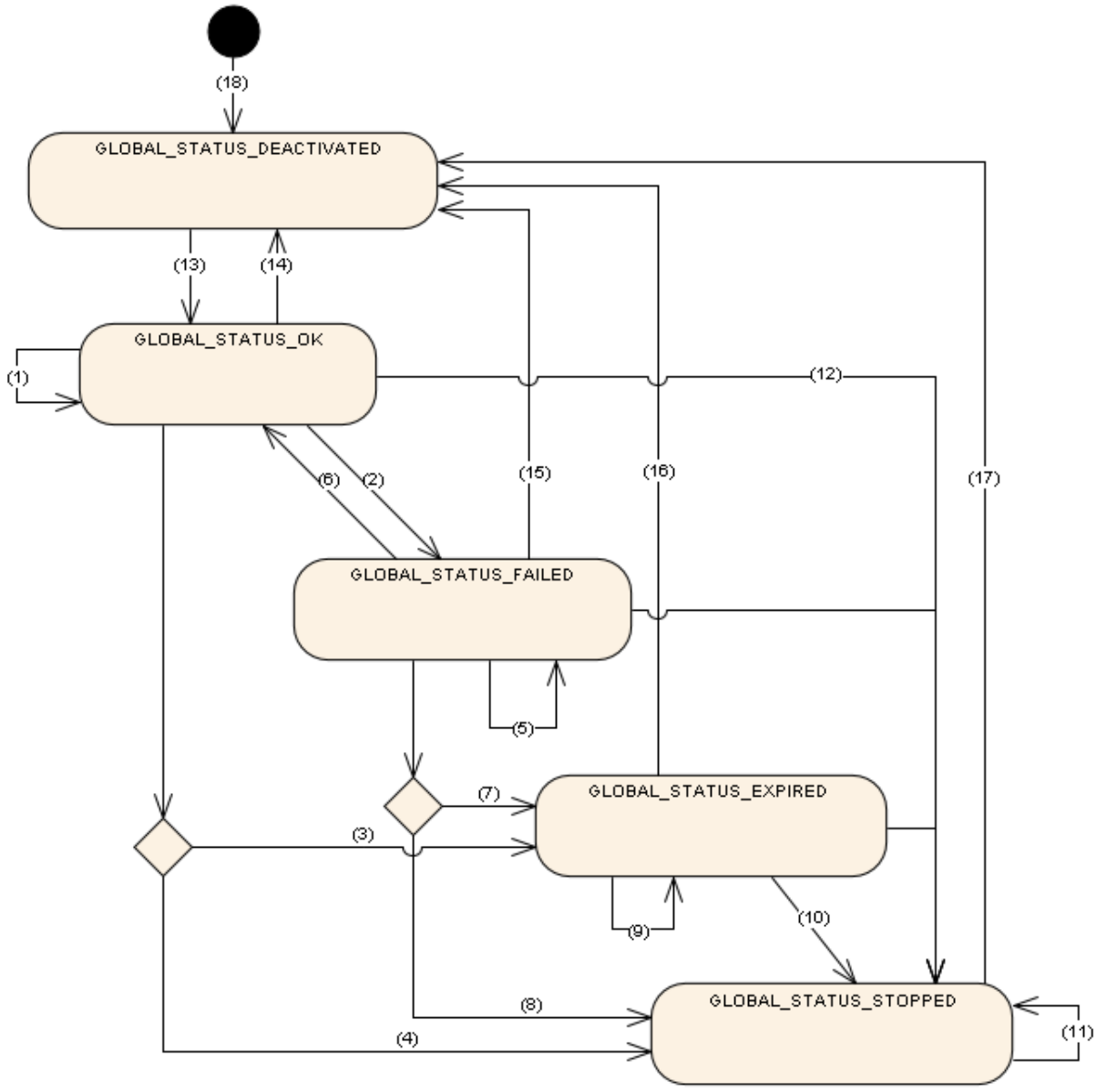

Figure 6.12: Global Supervision Status

[SWS_HM_00213] ⌈The Health Monitoring shall have one Global Supervision Status for a software subsystem. ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

[SWS_HM_00387] ⌈ The Global Supervision Status shall be statically initialized with GLOBAL_STATUS_DEACTIVATED (see Transition (18) in Figure 6.12). ⌋ *(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

The Health Monitoring provides a feature to postpone the error reaction (the error reaction being not setting a correct trigger condition) for a configurable amount of time measured in multiples of the Supervision Cycle (Supervision cycle is the period at which the Health Monitoring is performed), named Expired Supervision Tolerance (see configuration parameter ExpiredSupervisionCyclesTolerance). The Expired Supervision Tolerance is implemented within the state machine of the Global Supervision Status. The defined state machine is in the state GLOBAL_STATUS_EXPIRED while the blocking is postponed.

[SWS_HM_00214] ⌈The Health Monitoring shall calculate the Global Supervision Status in every Supervision cycle. The function shall compute the Global Supervision Cycle after it computed every Local Supervision Status.

The cyclic update of Global Supervision Status is necessary to trigger the timely transition from GLOBAL_STATUS_EXPIRED to GLOBAL_STATUS_STOPPED. ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

[SWS_HM_00285] ⌈ If the Health Monitoring was successfully initialized, the Global Supervision Status shall be set to GLOBAL_STATUS_OK (see Transition (13) in Figure 6.12). ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

[SWS_HM_00286] ⌈ If the Global Supervision Status was GLOBAL_STATUS_OK and the Health Monitoring is deactivated, then the Global Supervision Status shall be set to GLOBAL_STATUS_DEACTIVATED (see Transition (14), (15), (16) and (17) in Figure 6.12) ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

It has to be considered carefully that a deactivation of Health Monitoring when it is in states GLOBAL_STATUS_EXPIRED or GLOBAL_STATUS_STOPPED can hinder error reporting or error reaction.

[SWS_HM_00078] ⌈ If the Global Supervision Status was GLOBAL_STATUS_OK and the Local Supervision Status of all Supervised Entitys are either LOCAL_STATATUS_OK or LOCAL_STATUS_DEACTIVATED then the Health Monitoring shall keep the Global Supervision Status GLOBAL_STATUS_OK (see Transitions(1) in Figure 6.12) ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

[SWS_HM_00076] ⌈If the Global Supervision Status was GLOBAL_STATUS_OK, the Local Supervision Status of at least one Supervised Entity is LOCAL_STATUS_FAILED, and no SupervisedEntity is in Local Supervision Status LOCAL_STATUS_EXPIRED, then the Health Monitoring shall change the Global Supervision Status to GLOBAL_STATUS_FAILED (see Transition (2) in Figure 6.12) ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

The Health Monitoring supports a feature to delay the error reaction (switching to LOCAL_STATUS_EXPIRED) for a configurable amount of time. This could be used to allow clean-up activities before a watchdog reset, e.g. writing the error cause, writing NVRAM data.

[SWS_HM_00215] ⌈ If the Global Supervision Status was GLOBAL_STATUS_OK, the Local Supervision Status of at least one SupervisedEntity is LOCAL_STATUS_EXPIRED, and the Expired Supervision Tolerance is configured to a value larger than zero (see configuration parameter ExpiredSupervisionCyclesTolerance), then the Health Monitoring shall change the Global Supervision Status to GLOBAL_STATUS_EXPIRED (see Transition (3) in Figure 6.12) ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235, RS_HM_09163)*

[SWS_HM_00216] ⌈ If the Global Supervision Status was GLOBAL_STATUS_OK, the Local Supervision Status of at least one SupervisedEntity is LOCAL_STATUS_EXPIRED, and the Expired Supervision Tolerance is configured to zero (see configuration parameter ExpiredSupervisionCyclesTolerance), then the Health Monitoring shall change the Global Supervision Status to GLOBAL_STATUS_STOPPED (see Transition (4) in Figure 6.12) ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235, RS_HM_09163)*

[SWS_HM_00217] ⌈ If the Global Supervision Status was GLOBAL_STATUS_FAILED, the Local Supervision Status of at least one SupervisedEntity is LOCALCAL_STATUS_FAILED, and no SupervisedEntity is in Local Supervision Status LOCAL_STATUS_EXPIRED, then the Health Monitoring shall remain in Global Supervision Status GLOBAL_STATUS_FAILED. (see Transition (5) in Figure 6.12) ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

[SWS_HM_00218] ⌈If the Global Supervision Status was GLOBAL_STATUS_FAILED and the Local Supervision Status of all Supervised Entitys is either LOCAL_STATUS_OK or LOCAL_STATUS_DEACTIVATED then the Health Monitoring shall change the Global Supervision Status to GLOBAL_STATUS_OK (see Transition (6) in Figure 6.12) ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235)*

[SWS_HM_00077] ⌈ If the Global Supervision Status was GLOBAL_STATUS_FAILED, the Local Supervision Status of at least one SupervisedEntity is LOCAL_STATUS_EXPIRED, and the Expired Supervision Tolerance is configured to a value larger than zero (see configuration parameter ExpiredSupervisionCyclesTolerance), then the Health Monitoring shall change the Global Supervision Status to GLOBAL_STATUS_EXPIRED (see Transition (7) in Figure 6.12) ⌋*(RS_HM_09222, RS_HM_09125, RS_HM_09235, RS_HM_09163)*

[SWS_HM_00117] ⌈ If the Global Supervision Status was GLOBAL_STATUS_FAILED, the Local Supervision Status of at least one SupervisedEntity is LOCAL_STATUS_EXPIRED, and the Expired Supervision Tolerance is configured to zero (see configuration parameter ExpiredSupervi-

sionCyclesTolerance), then the Health Monitoring shall change the Global Supervision Status to GLOBAL_STATUS_STOPPED (see Transition (8) in Figure 6.12) ⌋(*RS_HM_09222, RS_HM_09125, RS_HM_09235, RS_HM_09163*)

[SWS_HM_00219] ⌈ If the Global Supervision Status was GLOBAL_STATUS_EXPIRED, the Local Supervision Status of at least one SupervisedEntity is LOCAL_STATUS_EXPIRED, and the Expired Cycle Counter is less or equal to the configured Expired Supervision Tolerance (see configuration parameter ExpiredSupervisionCyclesTolerance), then the Health Monitoring shall keep Global Supervision Status GLOBAL_STATUS_EXPIRED and increment the Expired Cycle Counter (see Transition (9) in Figure 6.12) ⌋(*RS_HM_09163*)

[SWS_HM_00220] ⌈ If the Global Supervision Status was GLOBAL_STATUS_EXPIRED, the Local Supervision Status of at least one SupervisedEntity is LOCAL_STATUS_EXPIRED, and the Expired Cycle Counter is larger than the configured Expired Supervision Tolerance (see configuration parameter ExpiredSupervisionCyclesTolerance), then the Health Monitoring shall change the Global Supervision Status to GLOBAL_STATUS_STOPPED (see Transition (10) in Figure 6.12) ⌋(*RS_HM_09163*)

[SWS_HM_00221] ⌈ If the Global Supervision Status was GLOBAL_STATUS_STOPPED, then the Health Monitoring shall remain in Global Supervision Status GLOBAL_STATUS_STOPPED (see Transition (11) in Figure 6.12) ⌋(*RS_HM_09222, RS_HM_09125, RS_HM_09235*)

6.3.3 Effect of changing Mode

The modes are statically configured and contained in the Health Monitoring configuration set. A mode switch changes the supervision parameters of the Supervised Entitys.

[SWS_HM_00182] ⌈ If the current global status GLOBAL_STATUS_OK or GLOBAL_STATUS_FAILED then for each SupervisedEntity that is activated in the new mode, the Health Monitoring shall retain the current state of the SupervisedEntity. Switching to the mode where a SupervisedEntity is deactivated clears also errors that had resulted with the GLOBAL_STATUS_FAILED status. ⌋(*RS_HM_09253*)

[SWS_HM_00315] ⌈ If the current global status is GLOBAL_STATUS_OK or GLOBAL_STATUS_FAILED then for each SupervisedEntity that is deactivated in the new mode, the Health Monitoring shall change the state of the SupervisedEntity to LOCAL_STATUS_DEACTIVATED; It shall set its Results of Active, Deadline and Logical Supervision to correct; It shall also clear its failed reference cycle counter to 0. ⌋(*RS_HM_09253*)

Executing a mode switch is possible when the Health Monitoring is in the state GLOBAL_STATUS_OK or GLOBAL_STATUS_FAILED. In other modes, changing the Supervision Mode has no effect.

[SWS_HM_00316] ⌈ If the current global status is not GLOBAL_STATUS_OK nor GLOBAL_STATUS_FAILED then the Health Monitoring shall not perform any actions at the Supervision Mode change. ⌋*(RS_HM_09253)*

[SWS_HM_00139] ⌈ If changing the supervision mode fails, the Health Monitoring shall assume a global supervision failure and set the `Global Supervision Status` to GLOBAL_STATUS_STOPPED. (see Transition (12) in Figure 6.12) ⌋*(RS_HM_09253)*

6.4 Determination of Actions based on Supervision Status

6.4.1 Concept of `HealthChannel` and `HealthStatus`

A `HealthChannel` can be the Global supervision status of the software under supervision.

A `HealthChannel` can be the result of an environment monitoring algorithm. Eg: Voltage Monitoring, Temperature Monitoring

A `HealthChannel` can be the result of a memory integrity test routine. Eg: RAM test, ROM test

A `HealthChannel` can be the status of the operating system or Kernel. Eg: Os Status, Kernel Status

A `HealthChannel` can be the status of another platform instance or Virtual Machine or ECU.

`HealthStatus` of a `HealthChannel` is the abstract format of the information that a `HealthChannel` provides to the `Health Monitoring`. Two different `HealthChannel`s may have same `HealthStatus` names to represent its result. Eg: High, low, normal

The various external monitoring routines shall report their result or status in the form of defined `HealthStatus` to the `Health Monitoring`. The `Health Monitoring` shall initiate the `Arbitration` using the indicated `HealthStatus` according to the configuration parameters defined in the configuration file. The `Arbitration` is composed of `Condition`s, `LogicalExpression`s and `Rule`s. The `Rule`s shall lead to execution of an `ActionList`.

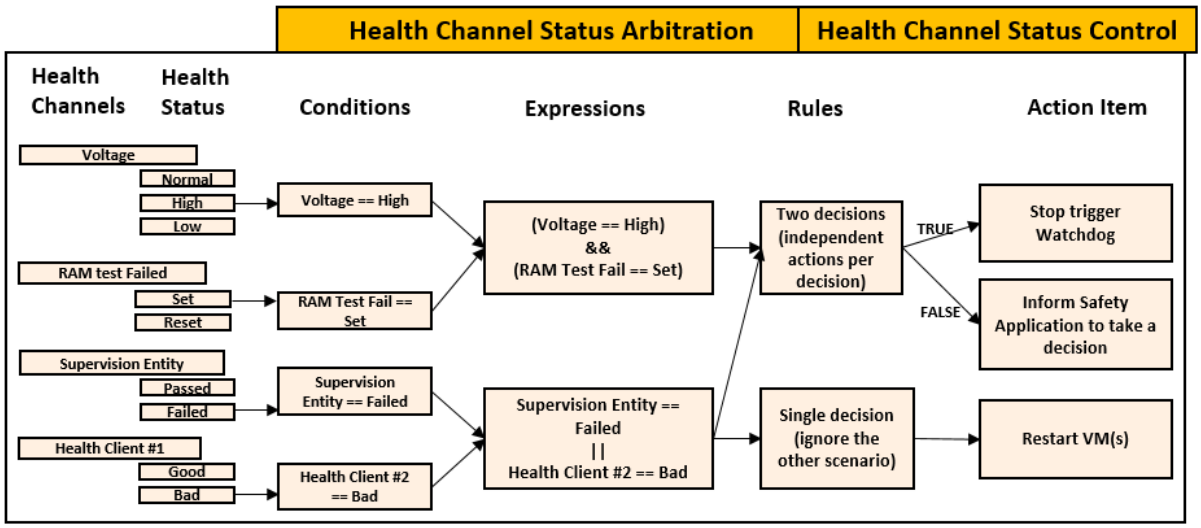

Figure 6.13: Determination of action based on the received `HealthStatus`

As mentioned above, a `HealthChannel` can be Internal (Example: Global Supervision Status) or External (Example: Voltage, RAM Test, Kernel Status). The `ActionList` is determined in two steps. First step is called `Arbitration` - see 6.4.2, which consists of `Condition`s, `LogicalExpression`s and `Rule`s. The second step is called `DeterminationOfActions` - see 6.4.3, where an `ActionList` is triggered based on if the `Rule` evaluates to TRUE or FALSE.

6.4.2 `Arbitration` of `HealthChannel`s

`Arbitration` performed by the `Health Monitoring` is simple and rule-based. The `Rule`s used for `Arbitration` are specified in the configuration of the `Health Monitoring` module.

The `Rule`s are composed of trivial `LogicalExpression`s and the `Arbitration` is thus expected to have a low runtime impact.

In order to know what `ActionList` to execute, the `Health Monitoring` is required to detect changes in `Arbitration` results from previous `Rule` evaluation. How this is done, and the memory needed to store results, is implementation specific and not described in this document.

6.4.2.1 `Arbitration Rule`s

A `Rule` is a `LogicalExpression` that is composed of a set of `Condition`s. The `Rule`s are evaluated when the `HealthStatus` of a `HealthChannel` is changed, or during the execution of the `Health Monitoring` main function. The result of the evaluation (True or False) is used to decide about execution of the corresponding `ActionList`.

6.4.2.2 Conditions and LogicalExpressions.

The LogicalExpression that comprises a Rule can use different operators such as AND, OR, XOR, NOT and NAND. Each term in the LogicalExpression corresponds to a Condition. Each defined HealthChannel when informs the Health Monitoring about its HealthStatus, the Condition will verify if a requested HealthStatus is EQUAL or NOT_EQUAL to a certain HealthStatus.

An example Rule with two Conditions is shown in Figure 6.14. The Rules and the set of available logical operations are defined as a part of the Health Monitoring configuration.

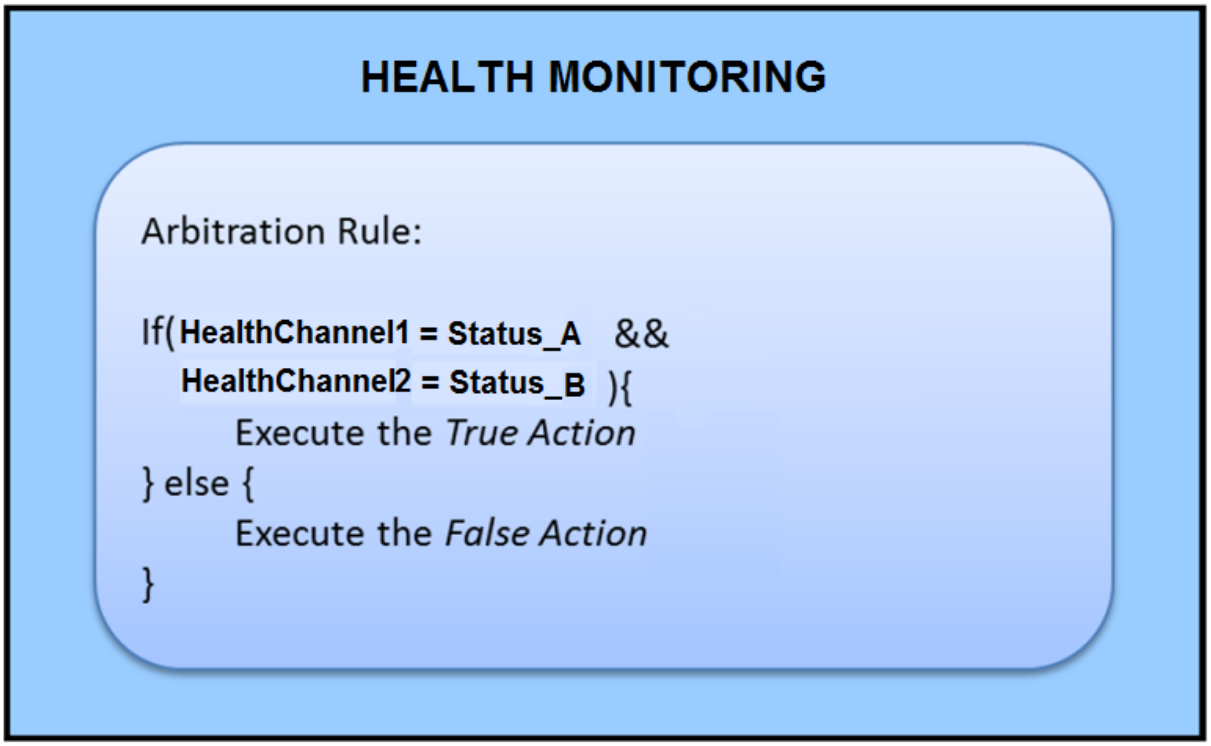

Figure 6.14: Pseudocode representation of an example rule with two conditions.

6.4.2.3 Requirements of Arbitration

The Health Monitoring accepts HealthStatus requests as input for the Arbitration. HealthStatus requests normally originate from the application but may also originate from other Platform services such as the DM, EM.

[SWS_HM_00050] ⌈ The Application shall request HealthStatus using the Health Monitoring interface ReportHealthStatus. ⌋*(RS_HM_09257)*

[SWS_HM_00051] ⌈ The Health Monitoring shall perform Arbitration based on incoming HealthStatus requests. ⌋*(RS_HM_09255)*

[SWS_HM_00052] ⌈ All HealthStatus requests are handled in the same way by the Health Monitoring. They are configured by selection of the corresponding HealthStatus condition type in the configuration. ⌋(*RS_HM_09255*)

[SWS_HM_00053] ⌈ The Health Monitoring shall perform Arbitration using configured Rules. ⌋(*RS_HM_09255*)

[SWS_HM_00054] ⌈ The Arbitration Rules shall be configurable using the module configuration parameters. ⌋(*RS_HM_09255*)

[SWS_HM_00055] ⌈ Health Monitoring is not allowed to use results of previous Arbitration Rule evaluations as input for the LogicalExpression. ⌋ (*RS_HM_09255*)

6.4.2.4 Arbitration Behavior after Initialization

The behavior of the Arbitration of Health Monitoring after initialization is controlled by the configuration container HealthStatusInitValue. This parameter may be configured once for each HealthChannel in the configuration.

[SWS_HM_00056] ⌈ If the container HealthStatusInitValue does not exist or the HealthChannel does not already have an initial value, the Health Monitoring shall treat the corresponding HealthStatus Condition as undefined and not use it for Arbitration until the corresponding HealthChannel has been updated for the first time. ⌋(*RS_HM_09255*)

6.4.3 Determination of ActionList

The DeterminationOfActions part of Health Monitoring performs the execution of the ActionList based on the results of the Arbitration. An ActionList is executed by the Health Monitoring when triggered by a Rule.

[SWS_HM_00057] ⌈ The Health Monitoring is not required to store or react on any Platform module specific return values on its performed ActionItems. ⌋ (*RS_HM_09255*)

[SWS_HM_00058] ⌈ The Health Monitoring shall contain a single ActionList or no ActionList, if Rule evaluates to True. ⌋(*RS_HM_09255*)

[SWS_HM_00059] ⌈ The Health Monitoring shall contain a single ActionList or no ActionList, if Rule evaluates to False. ⌋(*RS_HM_09255*)

[SWS_HM_00060] ⌈ The Health Monitoring evaluates its Rules in the context of the Health Monitoring interface ReportHealthStatus. ⌋(*RS_HM_09255*)

The corresponding ActionList is executed according to the selected execution method (see 6.4.3.1).

[SWS_HM_00062] ⌈ For each Rule of the Arbitration, Health Monitoring shall be able to execute different ActionList based on if the Rule evaluates to True or False. ⌋*(RS_HM_09255)*

[SWS_HM_00063] ⌈ The ActionList associated with Rules evaluated in the context of the Health Monitoring interface ReportHealthStatus shall be executed by Health Monitoring synchronously. Care must be taken that it is not blocking the Health Monitoring for too long. ⌋*(RS_HM_09255)*

6.4.3.1 Triggered and Conditional Execution

There are two ways that an ActionList may be executed based on evaluation of Rules. Either it is executed every time the Rule is evaluated with the corresponding result, or only when the evaluation result has changed from the previous evaluation. The execution method for an ActionList is configured using the ActionExecution parameter (within the ActionList container).

[SWS_HM_00064] ⌈ If a True ActionList is configured for triggered execution, the Health Monitoring shall only execute it when the evaluation of the corresponding Rule changes from False to True. ⌋*(RS_HM_09255)*

[SWS_HM_00065] ⌈ If a False ActionList is configured for triggered execution the Health Monitoring shall only execute it when the evaluation of the corresponding Rule changes from True to False. ⌋*(RS_HM_09255)*

[SWS_HM_00066] ⌈ If a True ActionList is configured for conditional execution, the Health Monitoring shall execute it every time the corresponding Rule is evaluated to True. ⌋*(RS_HM_09255)*

[SWS_HM_00067] ⌈ If a False ActionList is configured for conditional execution, the Health Monitoring shall execute it every time the corresponding Rule is evaluated to False. ⌋*(RS_HM_09255)*

6.4.3.2 Available ActionItems

The set of ActionItems that are available to use is predefined. The reason for this is to ease ECU configuration.

[SWS_HM_00068] ⌈ Health Monitoring shall be able to execute the predefined ActionItem defined by configuration container ActionItem. ⌋*(RS_HM_09159)*

[SWS_HM_00069] ⌈ The Health Monitoring shall execute a standard ActionItem mentioned below to recover from the failure. ⌋*(RS_HM_09159)*

6.4.3.2.1 Terminate or Restart Application

[SWS_HM_00071] ⌈ Health Monitoring shall invoke the standard interface defined in Execution Management module to Terminate or Restart an Application. ⌋(*RS_HM_09251*)

6.4.3.2.2 Reset Platform instance

[SWS_HM_00072] ⌈ Health Monitoring shall make use of a project specific driver interface or Hypervisor interface to reset the system, based on the complexity of the system. ⌋(*RS_HM_09169*)

6.4.3.2.3 Hardware Watchdog Reset

[SWS_HM_00073] ⌈ Health Monitoring shall use appropriate interface to periodically refresh the watchdog to prevent a watchdog reset. ⌋(*RS_HM_09244*)

[SWS_HM_00075] ⌈ When the Health Monitoring fails to refresh the watchdog, this automatically shall cause a watchdog reset (after the time needed to refresh the hardware watchdog elapse). ⌋(*RS_HM_09244*)

6.4.3.2.4 Callback Notification Function

[SWS_HM_00079] ⌈ Health Monitoring shall invoke the configured callback function to notify the Safety application to take an appropriate action. ⌋(*RS_HM_09159*)

6.4.3.3 Behavior of ActionList execution after Initialization

The behavior of the ActionList execution after initialization of the Health Monitoring is configured by the RuleInitState parameter (within the Rule container). It defines the "previous evaluation result" to be used when deciding on what ActionList to execute after the first evaluation of a Rule after initialization. The configuration parameter ActionExecution (within the ActionList container) also affects the ActionList execution after initialization.

[SWS_HM_00080] ⌈ The Health Monitoring shall act according to what is stated in Table 6.1 when a Rule is evaluated for the first time after initialization. ⌋ (*RS_HM_09255*)

RuleInitState	*ActionExecution*	*Rule evaluated to "true"*	*Rule evaluated to "false"*
UNDEFINED	TRIGGER	Execute "true" action	Execute "false" action
TRUE	TRIGGER	Do nothing	Execute "false" action
FALSE	TRIGGER	Execute "true" action	Do nothing

UNDEFINED	CONDITION	Execute "true" action	Execute "false" action
TRUE	CONDITION	Execute "true" action	Execute "false" action
FALSE	CONDITION	Execute "true" action	Execute "false" action

Table 6.1: Usage of the RuleInitState configuration parameter

Note: The "true" and "false" action are optional for each rule

7 Watchdog API specification

This chapter specifies the API of `Platform Health Management` that is referred in other document parts. It is defined in generic/abstract way, so that it can be implemented on different platforms. In particular, the data types are not defined.

7.1 Provided API

7.1.1 Reporting `Checkpoints` and `Health Status`

[SWS_HM_00447] ReportCheckpoint ⌈ `Health Monitoring` shall provide a method to report the current code location, represented by a Checkpoint

```
1 ReportCheckpoint(CheckpointID id)
```

⌋*(RS_HM_09254)*

7.1.2 Reporting health status

[SWS_HM_00448] ⌈ `Health Monitoring` shall provide a method to report the health status information

```
1 ReportHealthStatus(HealthStatusID id, HealthStatus status)
```

⌋*(RS_HM_09257)*

7.1.3 Forwarding information between health monitoring components

[SWS_HM_00449] ReportHealthMonitoring ⌈ `Health Monitoring` shall provide a method to report the information collected and determined by one `Health Monitoring` component, so that they can be forwarded to another `Health Monitoring` component.

```
1 ReportHealthMonitoring(HealthMonitoring montoringData)
```

⌋*(RS_HM_09159)*

7.1.4 Init / DeInit

[SWS_HM_00455] Init ⌈ `Health Monitoring` shall provide a method to initialize the service.

```
1 Init()
```

⌋(*RS_HM_09222*, *RS_HM_09125*, *RS_HM_09235*, *RS_HM_09255*)

[SWS_HM_00456] DeInit ⌈ `Health Monitoring` shall provide a method to deinitialize the service.

```
1 DeInit()
```

⌋(*RS_HM_09222*, *RS_HM_09125*, *RS_HM_09235*, *RS_HM_09255*)

7.2 Assumed API

This section specified an API that is used by `Health Monitoring`.

7.3 Triggering error handling

[SWS_HM_00450] TriggerErrorHandler ⌈ `Health Monitoring` shall provide a method to trigger a defined error handler, providing the identifier of this error.

```
1 TriggerErrorHandler(ErrorID id)
```

⌋(*RS_HM_09159*)

7.4 Controlling watchdog

[SWS_HM_00451] SetWatchdogCondition ⌈ `Health Monitoring` shall provide a method to control the watchdog drivers.

```
1 ControlWatchdog(ControlData control)
```

⌋(*RS_HM_09244*, *RS_HM_09245*, *RS_HM_09246*, *RS_HM_09247*, *RS_HM_09248*, *RS_HM_09028*, *RS_HM_09226*)

8 Configuration Parameters

This chapter specifies a configuration model of `Health Monitoring`. The options defined here are referenced/used in chapter 6.

This configuration, which is abstract and platform-independent is supposed to be implemented/instantiated by the specific platforms, e.g. by AUTOSAR AP.

8.1 Overall configuration

The configuration of a (representing MCU, virtual machine, partition) is split into two categories:

1. `ModeIndependentSettings` - containing only static information: what are possible `SupervisedEntity`s and possible `HealthChannel`s
2. `ModeDependentSettings` - containing all supervision function configurations.

It means all supervision configuration is fully mode-dependent.

A system is made of several `Machines`. Therefore, `HealthMonitoring` is allocated to a specific `Machine`.

It is possible that there are several independent suppliers of software for the same `Machine`. Therefore, each of suppliers can supply any part of the configuration, for any configuration classes.

`ModeDependentSettings` contains also the configuration of watchdogs - but this part is not standardized (marked in blue).

The definitions of `Machines` (machines/virtual machines/partitions) and `SupervisionModes` are assumed to be provided externally (by other specifications) therefore they are only referenced here.

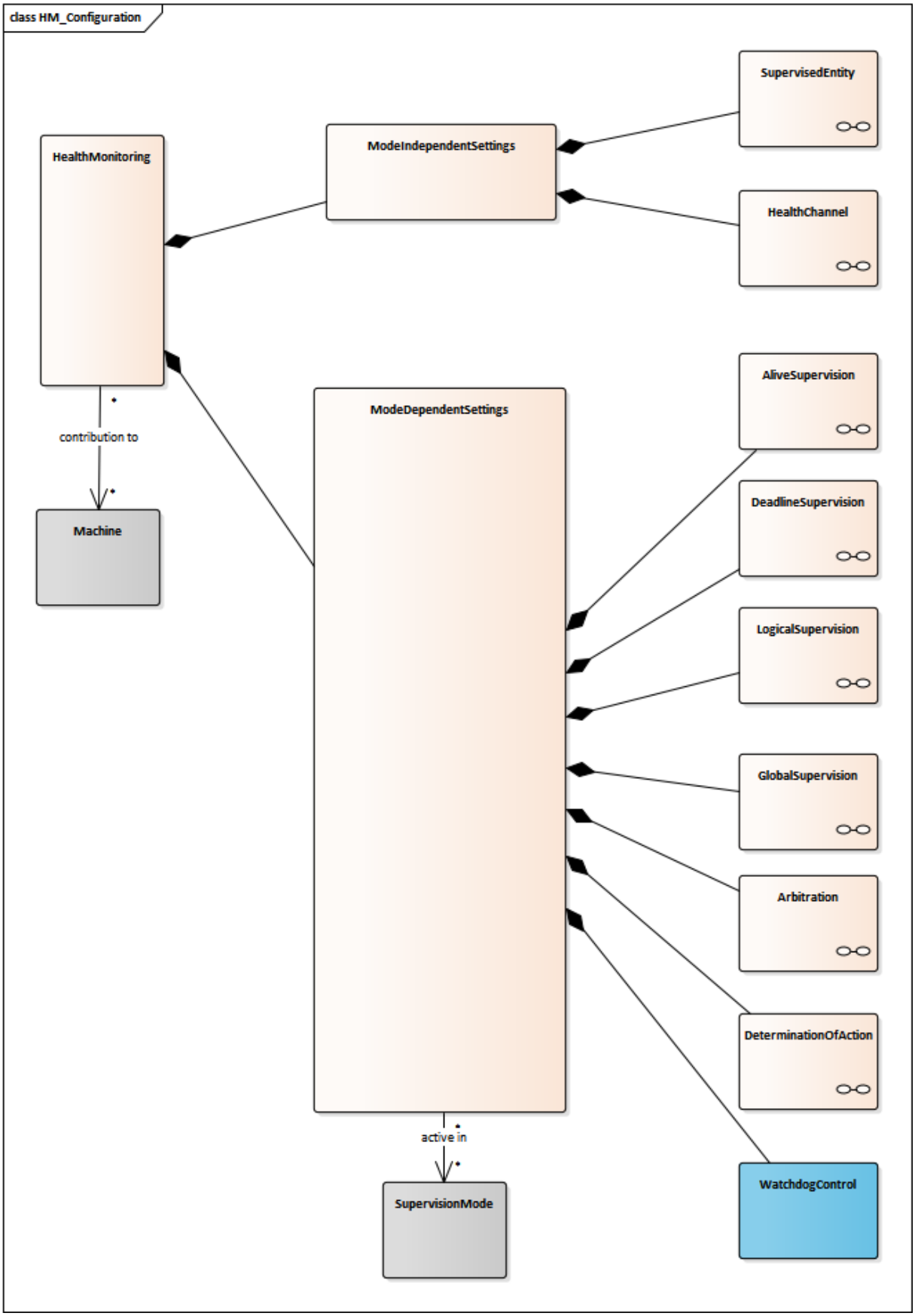

Figure 8.1: Overall configuration

8.2 Mode-independent settings

contain static information: what are possible SupervisedEntitys and possible HealthChannel.

Implementation hint: This part of configuration is typically used to generate the type-safe API to Applications.

8.2.1 Supervised Entity

A is a collection of Checkpoints that can occur during the runtime of a software.

A has the following options:

1. : Globally unique name identifier, used by Applications
2. : Globally unique identifier (number)
3. : number of instances of this SupervisedEntity.

Note that on AUTOSAR AP, the uniqueness of the name can be ensured by using a namespace as a part of the identificaiton.

A Checkpoint has the following options:

1. : Name, used by Applications, unique within the SupervisedEntity.
2. : Identifier of the Checkpoint, unique within the SupervisedEntity.

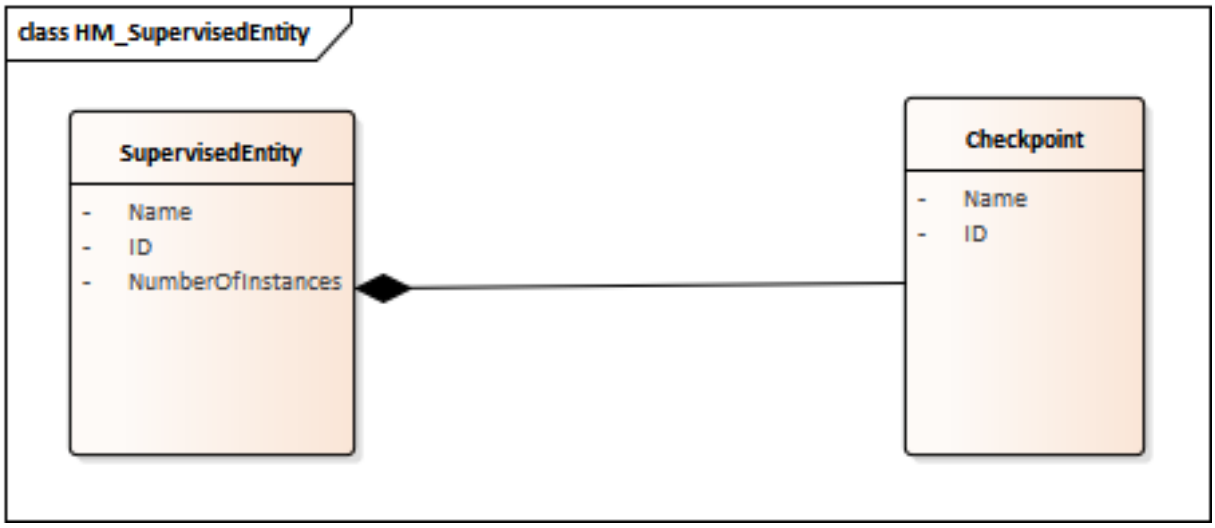

Figure 8.2: Supervised Entity

Note: On AUTOSAR AP, a Supervised Entity results with an enum, named after the Supervised Entity's namespace and name, with the enumerations corresponding to the checkpoints.

8.2.2 Health Channel

is an information that is being monitored that has already been determined by the supplier and classified according to the possible health status values.

represents a possible value of the HealthChannel.

For example, a HealthChannel can be a "Motor Temperature" its HealthChannels: "HH", "'H", "OK".

A HealthChannel has the following options:

1. : Globally unique name identifier, used by Applications.
2. : Globally unique identifier (number)
3. : number of instances of this HealthChannel.

A HealthStatus has the following options:

1. : Name, used by Applications, unique within the HealthChannel
2. : Identifier of the HealthStatus, unique within the HealthChannel.

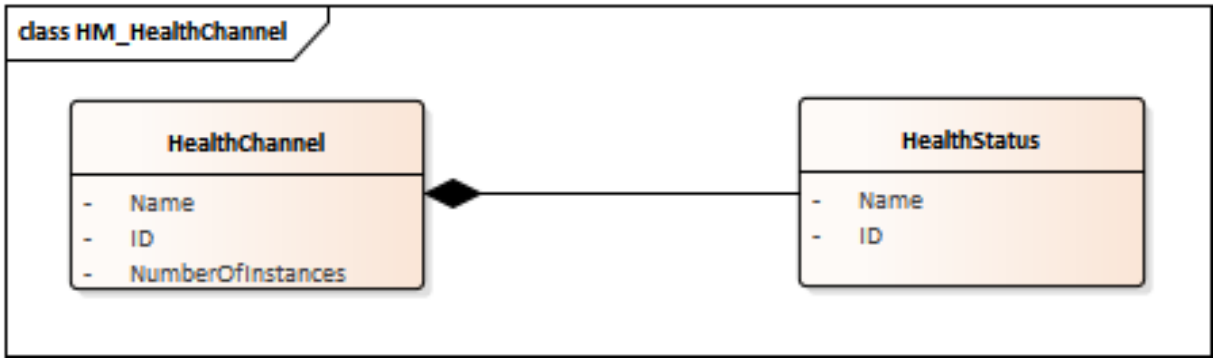

Figure 8.3: Health Channel

8.3 Mode-dependent settings

contain all supervision function configurations.

Implementation hint: This part of configuration is typically used by non-generated code to perform the supervision at runtime.

8.3.1 Alive Supervision

checks the amount of reported alive indications within the AliveReferenceCycle, which is to be within ExpectedAliveIndications - MaxMargin and ExpectedAliveIndications - MaxMargin.

AliveSupervision has the following options:

1. : time period at which the Alive Supervision mechanism compares the amount of received Alive Indications of the Checkpoint against the expected/configured amount.
2. : the amount of expected alive indications of the Checkpoint within AliveReferenceCycle
3. : amount of acceptable missing alive indications within AliveReferenceCycle
4. : amount of acceptable additional alive indications within AliveReferenceCycle

A uniquely identifies a specific location in source code. Different executions of the same code (e.g. due to multithreading or running the same application in several instances) share the same Checkpoint identification.

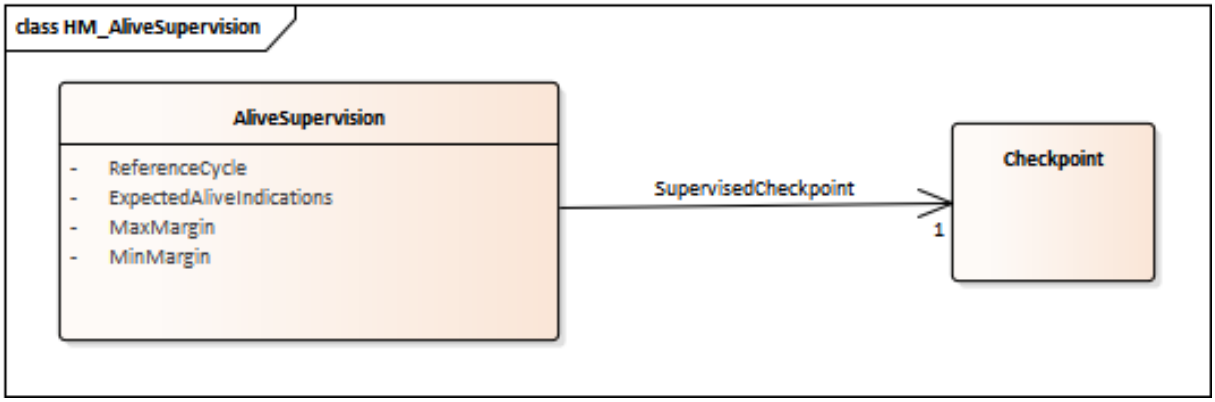

Figure 8.4: Alive Supervision

8.3.2 Deadline Supervision

has the following options:

1. : longest time span allowed.
2. : shortest time span allowed.

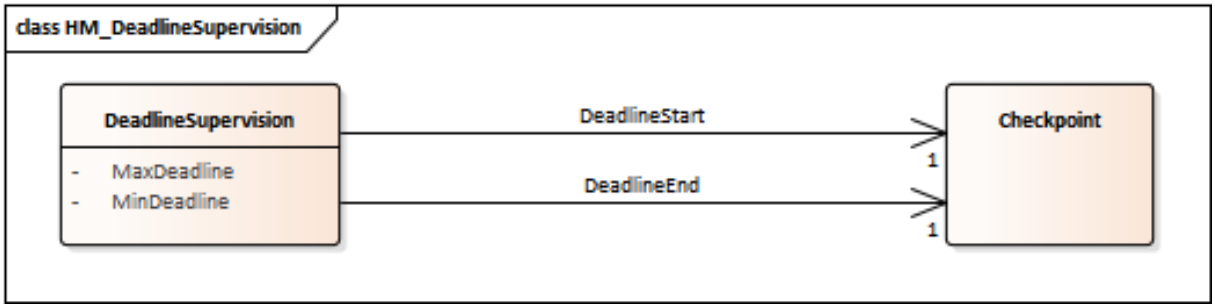

Figure 8.5: Deadline Supervision

8.3.3 Logical Supervision

is a collection of TransitionSupervisions.

A LogicalSupervision can be seen one graph.

As LogicalSupervision represents a graph, so it is possible to configure the initial and/or the final Checkpoints by referring to those Checkpoints.

A has its and Checkpoint. One Checkpoint can have multiple Transitions - this way it is possible to configure merges and forks in the graph (e.g. from A you can go to B or to C).

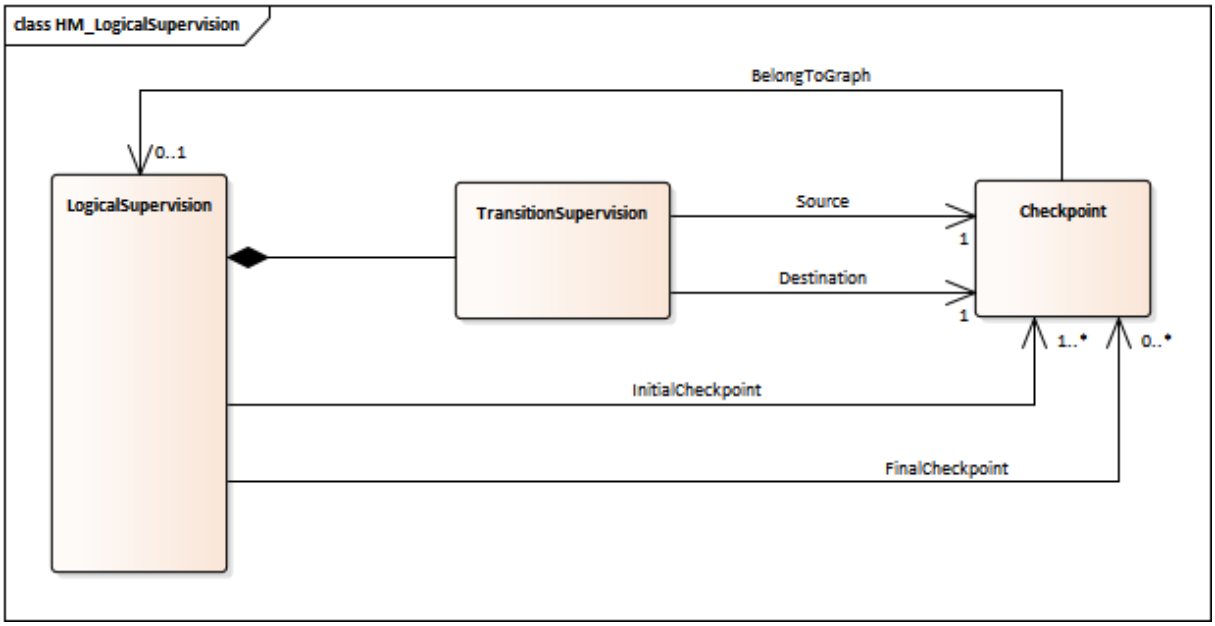

Figure 8.6: Logical Supervision

8.3.4 Global Supervision

A is an overall state of a software subsystem. There can one or a few Global Supervisions per Machine. It has the following options:

1. : at which cycle the GlobalSupervision and its contained LocalSupervisions are executed (i.e. at which cycle the new state is determined)
2. : maximum acceptable amount SupervisionCycles in the global state GLOBAL_STATUS_EXPIRED before it is considered LOCAL_STATUS_STOPPED.

Global Supervision is a "worst-of" of all contained LocalSupervisions.

represents the state of a a group of s, which have the same as their parent Global Supervisions. It has the following option:

1. : maximum acceptable amount SupervisionCycles in the local state LOCAL_STATUS_FAILED before it is considered LOCAL_STATUS_EXPIRED.

Note that the option FailedSupervisionCyclesTolerance is used only for AliveSupervision.

There is no fixed relation between SupervisedEntity on one side and LocalSupervision or GlobalSupervision on another side - it is fully configurable.

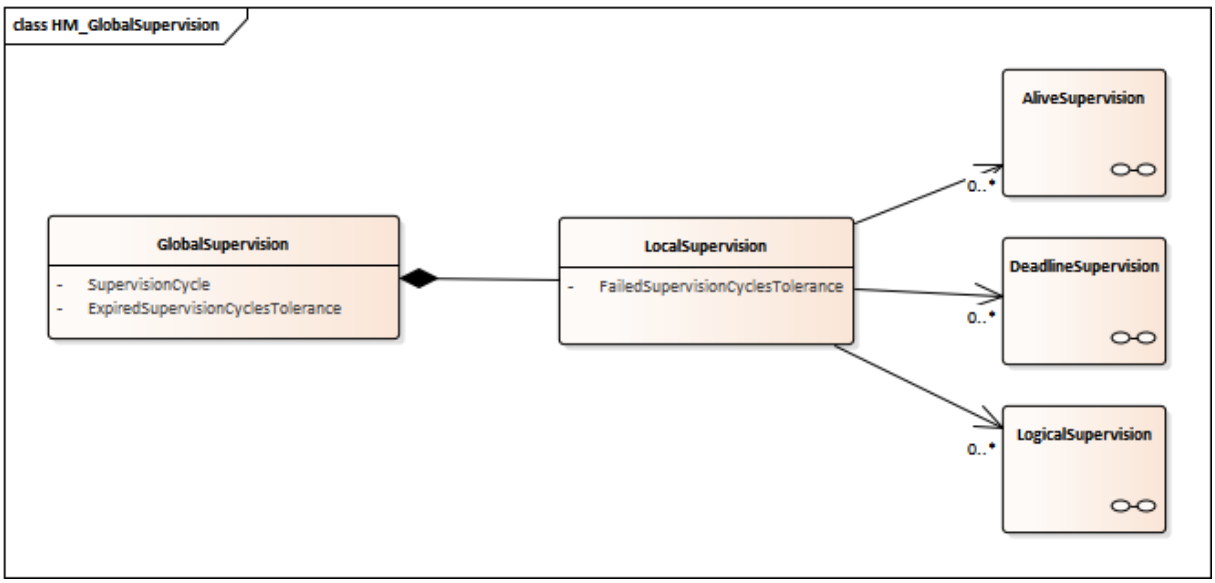

Figure 8.7: Global Supervision Status

8.3.5 Arbitration

has the following option:

1. : time period at which the Arbitration is performed.

The Arbitration has three parts:

1. Condition: boolean conditions representing if a condition is fulfilled or not
2. LogicalExpression: logical expressions built on Conditions or other LogicalExpressions
3. Rule: it takes the result of one LogicalExpression and it defines the reaction based on it.

One is a boolean and it is built by a reference to two elements:

1. HealthChannel + HealthStatus value or
2. AliveSupervision + LocalSupervisionStatus value or
3. DeadlineSupervision + LocalSupervisionStatus value or

4. `LogicalSupervision` + `LocalSupervisionStatus` value or
5. `GlobalSupervision` + `LocalSupervisionStatus` value.

Example: condition c1 meaning that "MotorEngine == HH"

One is built on top of `Condition`s or/and other `LogicalExpression`s, connected with `LogicalOperator`s. It is a boolean equation, e.g. `c1 or c2`. It has the following option:

1. a boolean operator used to connect `Condition`s and/or s

has the following option:

1. that defines the value before the `Condition`s are available. The `Rule` is considered true (active) if its `Condition` is true.

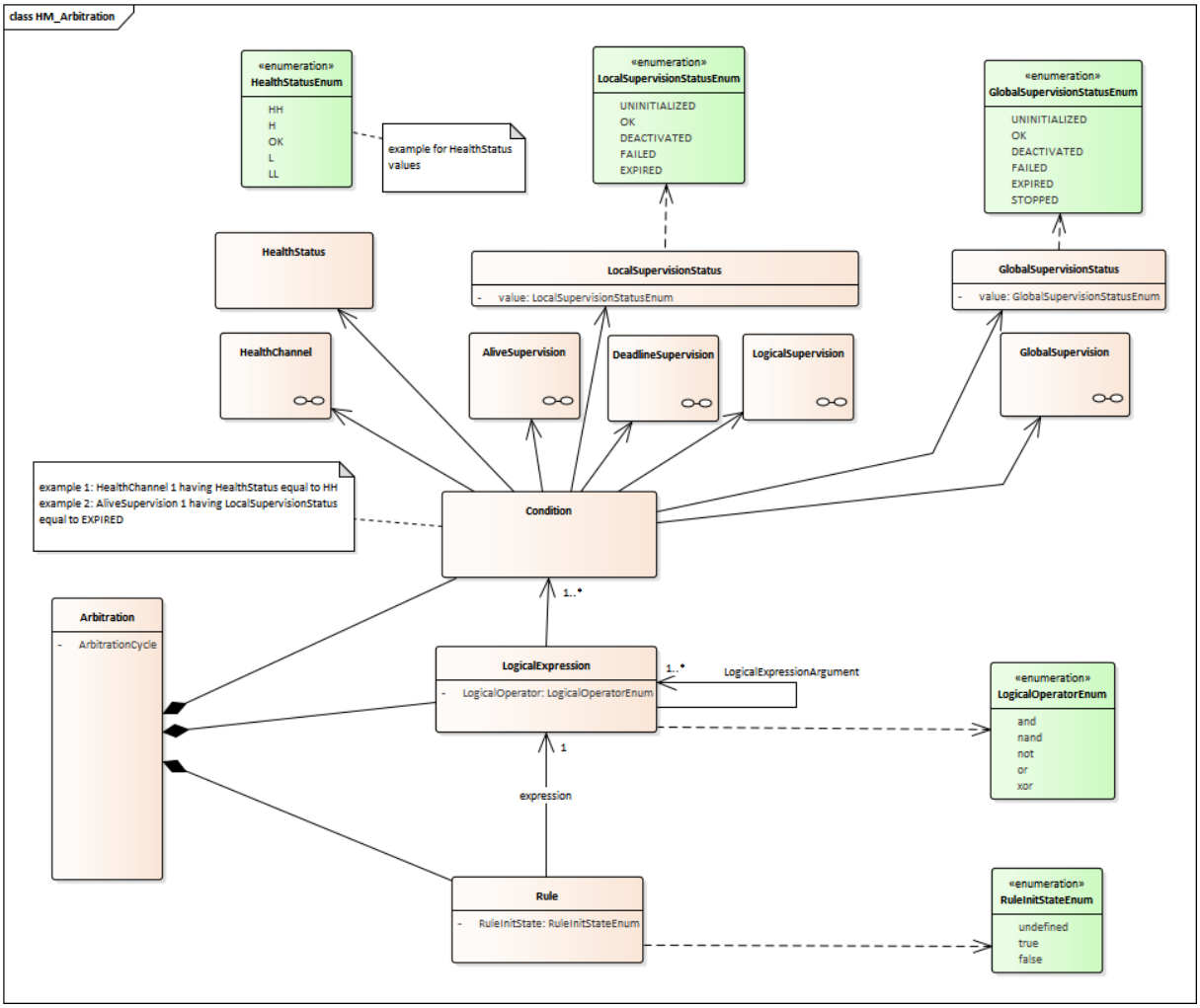

Figure 8.8: `Arbitration`

8.3.6 Determination of Action

is a collection of ActionLists to be performed.

A Rule takes the result of exactly one LogicalExpression and defines the handling of a reaction based on the result of the LogicalExpression:

1. if the LogicalExpression evaluates to true the ActionList referenced in the role will be indicated for execution
2. if the LogicalExpression evaluates to false the ActionList referenced in the role will be indicated for execution

The collects an ordered list of ActionItems to be executed when the ActionList is executed.

The definition of is not standardized (marked in blue). It could be e.g.:

1. request to switch to another mode
2. request to re-establish (reenter) the current mode, possibly by doing the reset
3. request to do soft reboot
4. immedate reset by direct invocation of low-level drivers
5. wrong triggering of HW watchdogs, resulting with immediate reset.

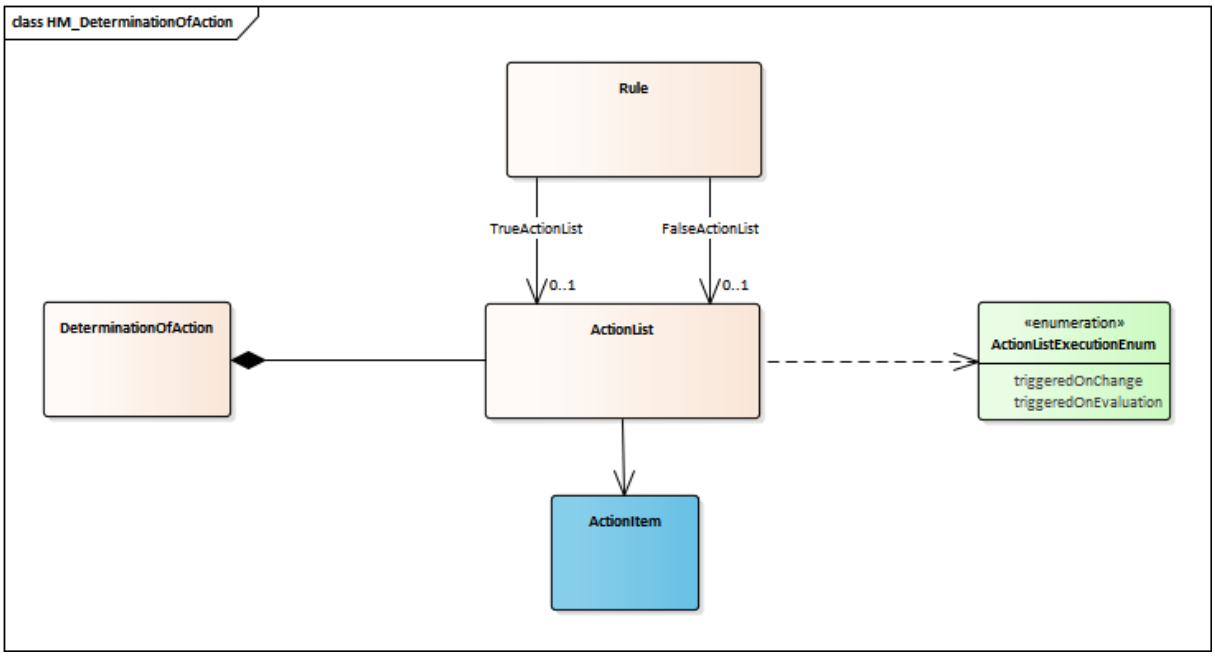

Figure 8.9: Determination of Action