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

Document Status	published
Part of AUTOSAR Standard	Foundation
Part of Standard Release	R21-11

Document Change History

Date	Release	Changed by	Description
2021-11-25	R21-11	AUTOSAR Release Management	• Add Application Interfaces for SystemHealthMonitoring • Add Mode Dependent Configuration
2020-11-30	R20-11	AUTOSAR Release Management	• Change document type from SWS to ASWS • Remove arbitration rules and actions • Remove HealthChannel supervision • Add SystemHealthMonitoring • Remove spec item numbers from API chapter
2019-11-28	R19-11	AUTOSAR Release Management	• Clarifications in specification of supervisions • Deleted parameter "‘number of instances"’ from HealthChannel and SupervisedEntity • Removed SWS_HM_00071 • Changed Document Status from Final to published

2019-03-29	1.5.1	AUTOSAR Release Management	• Updated acronyms table • Added chapter with not applicable requirements • Added SWS_HM_00460 and SWS_HM_00461 • Updated traceability to requirements of RS Health Monitoring • Moved figures out of requirement trace items
2018-10-29	1.5.0	AUTOSAR Release Management	• Added API for retrieving supervision status • Clarified error recovery actions • Modified parameter configuration • Several editorial changes
2018-03-29	1.4.0	AUTOSAR Release Management	• Initial release as "draft"

Disclaimer

Table of Contents

1 Introduction and functional overview

1.1 Input documents and related standards and norms

This document specifies the functionality on the Health Monitoring and System Health Monitoring.

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

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].

The System Health Monitoring introduces platform agnostic health monitoring. It aims to abstract the health monitoring on a system level by sharing of health information between different Adaptive, Classic or non-AUTOSAR platforms. The health information shall be shared between different platforms using a standardized format of Health Indicators. The abstract interfaces for exchanging the health information across several platforms are provided in this document.

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 Indication	An indication of a Supervised Entity to signal its aliveness by calling a checkpoint used for Alive Supervision.
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.
Final Checkpoint	The ending Checkpoint of a Graph. There can be zero or more Final Checkpoints for each Graph.
Global Supervision Status	Status that summarizes the Local Supervision Status of all Supervised Entities of a software subsystem.
Graph	A set of Checkpoints connected through Transitions, where at least one of Checkpoints is an Initial Checkpoint. There is a path (through Transitions) between any two Checkpoints of the Graph.
Health Channel	Channel providing information about the health status of a (sub)system. This might be the Global Supervision Status of an application, the result any test routine or the status reported by a (sub)system (e.g. voltage monitoring, OS kernel, ECU status, ...).
Health Channel Supervision	Kind of supervision that checks if the health indicators registered by the supervised software are within the tolerances/limits.
Health Monitoring	Supervision of the software behaviour for correct timing and sequence.
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	Check if the health indicators registered by the supervised software are within the tolerances/limits.
Initial Checkpoint	The starting Checkpoint of a Graph. There can be one or more Initial Checkpoints for each Graph.
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.
Machine	see [3] AUTOSAR Glossary
Platform Health Management	`Health Monitoring` for the Adaptive Platform

Supervised Entity	A whole or part of a software component type which is included in the supervision. A Supervised Entity denotes a collection of Checkpoints within the corresponding software component type. A software component type can include zero, one or more Supervised Entities. 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 or state of a Function Group (in case of Adaptive Platform). Modes are mutually exclusive. A mode can be e.g. Startup, Shutdown, Low power.
Health Indicator	Health Indicator provides an evaluation metric of current system performance with regard to safety requirements.
System Health Monitor(SHM)	System Health Monitor is responsible for monitoring the health of a (Sub)-system. It provides Health Indicators that can be used for system wide error handling across several Classic, Adaptive and any third party platforms.
Local Health Monitor	Local Health Monitor gathers health information of the platform on which it is deployed.
Health Indicator Interface	Health Indicator Interface is an interface used for communication of Health Indicators using a standardized service field.
SE	Supervised Entity.
SOTIF	Safety Of The Intended Functionality [4].
Performance	The Performance rates the performance with respect to malfunctioning behavior.
Reliability	Reliability evaluates how much to trust the system due to uncertainties.

Table 2.1: Acronyms

3 Related documentation

References

[1] ISO 26262:2018 (all parts) – Road vehicles – Functional Safety
http://www.iso.org

[2] Requirements on Health Monitoring
AUTOSAR_RS_HealthMonitoring

[3] Glossary

AUTOSAR_TR_Glossary

[4] ISO/PAS 21448:2019 – Road vehicles – Safety of the intended functionality
http://www.iso.org

[5] Explanation of System Health Monitoring
AUTOSAR_EXP_SystemHealthMonitoring

[6] Specification of Watchdog Manager
AUTOSAR_SWS_WatchdogManager

[7] Specification of Platform Health Management
AUTOSAR_SWS_PlatformHealthManagement

4 Constraints and assumptions

4.1 Limitations and conditions of use

- The logic for determination of Health Indicator values is not standardized as a part of AUTOSAR.
- Deadline Supervision across Supervised Entitys is not completely specified. It is not specified to which Local Supervision Status (Local Supervision Status corresponds to a Supervised Entity) shall its result contribute to (either to one corresponding to Source Checkpoint/Target Checkpoint/ both).

4.2 Applicability to car domains

No restrictions.

5 Requirements Tracing

Requirement	Description	Satisfied by
[RS_HM_09125]	Health Monitoring shall provide an Alive Supervision	[ASWS_HM_00074] [ASWS_HM_00083] [ASWS_HM_00098]
[RS_HM_09163]	Health Monitoring shall provide configurable tolerances for detected errors and configurable delays of error reactions.	[ASWS_HM_00075] [ASWS_HM_00079]

Requirement	Description	Satisfied by
[RS_HM_09222]	Health Monitoring shall provide a Logical Supervision	[ASWS_HM_00252] [ASWS_HM_00271] [ASWS_HM_00273] [ASWS_HM_00295] [ASWS_HM_00296] [ASWS_HM_00297] [ASWS_HM_00331]
[RS_HM_09235]	Health Monitoring shall provide a Deadline Supervision	[ASWS_HM_00228] [ASWS_HM_00229] [ASWS_HM_00294] [ASWS_HM_00299] [ASWS_HM_00354]
[RS_HM_09242]	Health Monitoring shall support the supervision within and across Supervised Entities.	[ASWS_HM_00460]
[RS_HM_09243]	Health Monitoring shall support the supervision of concurrent and parallel Supervised Entities.	[ASWS_HM_00461]
[RS_HM_09249]	Health Monitoring shall support building safety-related systems.	[ASWS_HM_00074] [ASWS_HM_00083] [ASWS_HM_00098] [ASWS_HM_00228] [ASWS_HM_00229] [ASWS_HM_00252] [ASWS_HM_00271] [ASWS_HM_00273] [ASWS_HM_00294] [ASWS_HM_00295] [ASWS_HM_00296] [ASWS_HM_00297] [ASWS_HM_00299] [ASWS_HM_00331] [ASWS_HM_00354] [ASWS_HM_00460] [ASWS_HM_00461]
[RS_HM_09300]	System Health Monitor shall transmit Health Indicators as standardized service events	[ASWS_HM_00510]
[RS_HM_09301]	SHM shall receive relevant health information from local health monitors	[ASWS_HM_00501] [ASWS_HM_00513]
[RS_HM_09302]	Communication between SHM and local health monitors shall be E2E protected	[ASWS_HM_00503]

Requirement	Description	Satisfied by
[RS_HM_09303]	SHM shall be platform agnostic	[ASWS_HM_00501] [ASWS_HM_00502] [ASWS_HM_00503] [ASWS_HM_00504] [ASWS_HM_00505] [ASWS_HM_00506] [ASWS_HM_00509] [ASWS_HM_00510] [ASWS_HM_00511] [ASWS_HM_00512] [ASWS_HM_00513] [ASWS_HM_00514] [ASWS_HM_00515] [ASWS_HM_00516] [ASWS_HM_00517] [ASWS_HM_00518] [ASWS_HM_00519] [ASWS_HM_00520] [ASWS_HM_00521] [ASWS_HM_00522] [ASWS_HM_00523]
[RS_HM_09304]	SHM shall determine Health Indicators.	[ASWS_HM_00501] [ASWS_HM_00504]
[RS_HM_09305]	SHM should support redundancy concepts	[ASWS_HM_00504] [ASWS_HM_00505]
[RS_HM_09308]	Communication between SHM instances shall be E2E protected	[ASWS_HM_00506]
[RS_HM_09309]	Cyclic communication between SHM and local health monitors shall be used for aliveness checks	[ASWS_HM_00502] [ASWS_HM_00509]
[RS_HM_09310]	Cyclic communication between SHM instances shall be used for aliveness checks	[ASWS_HM_00509]

6 Functional specification

6.1 Functional Overview

This section presents black-box functional overview of 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.

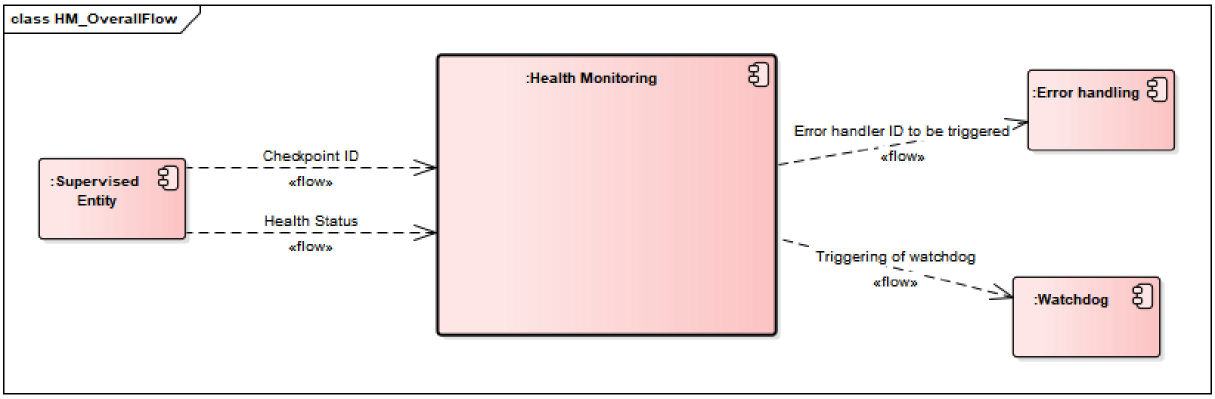

Figure 6.1: Scope of Health Monitoring

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.

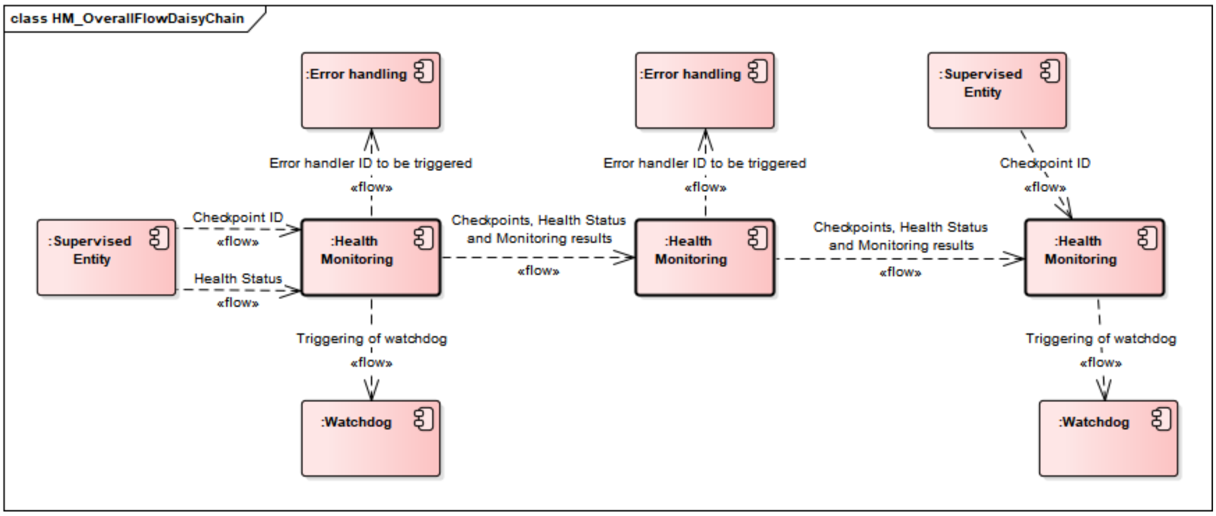

Figure 6.2: Scope of Health Monitoring Daisy Chain example

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.

A SystemHealthMonitor is responsible for combining health information of different platforms and calculate Health Indicators on different abstraction levels. These Health Indicators can then be used within the platform for stabilizing the system or enhancing services with some kind of Health of Service. The SystemHealthMonitor is defined as a platform agnostic component which could be deployed anywhere in the system.

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.

[ASWS_HM_00460] ⌈The Health Monitoring shall supervise graphs with checkpoints belonging to the same or different Supervised Entitys.⌋*(RS_HM_09242, RS_HM_09249)*

[ASWS_HM_00461] ⌈The Health Monitoring shall simultaneously supervise graphs of Supervised Entitys preempeted by other Supervised Entitys.⌋*(RS_HM_09243, RS_HM_09249)*

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 (one Source Checkpoint and one Target Checkpoint) of a Supervised Entity (for detection of early arrivals and delays), and elapsed time after the Source Checkpoints (for detection of timeouts). 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.

6.1.4 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.

6.1.4.1 Rule Pocessing

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

6.1.4.2 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.

6.1.4.3 Error Handling

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

6.1.5 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

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

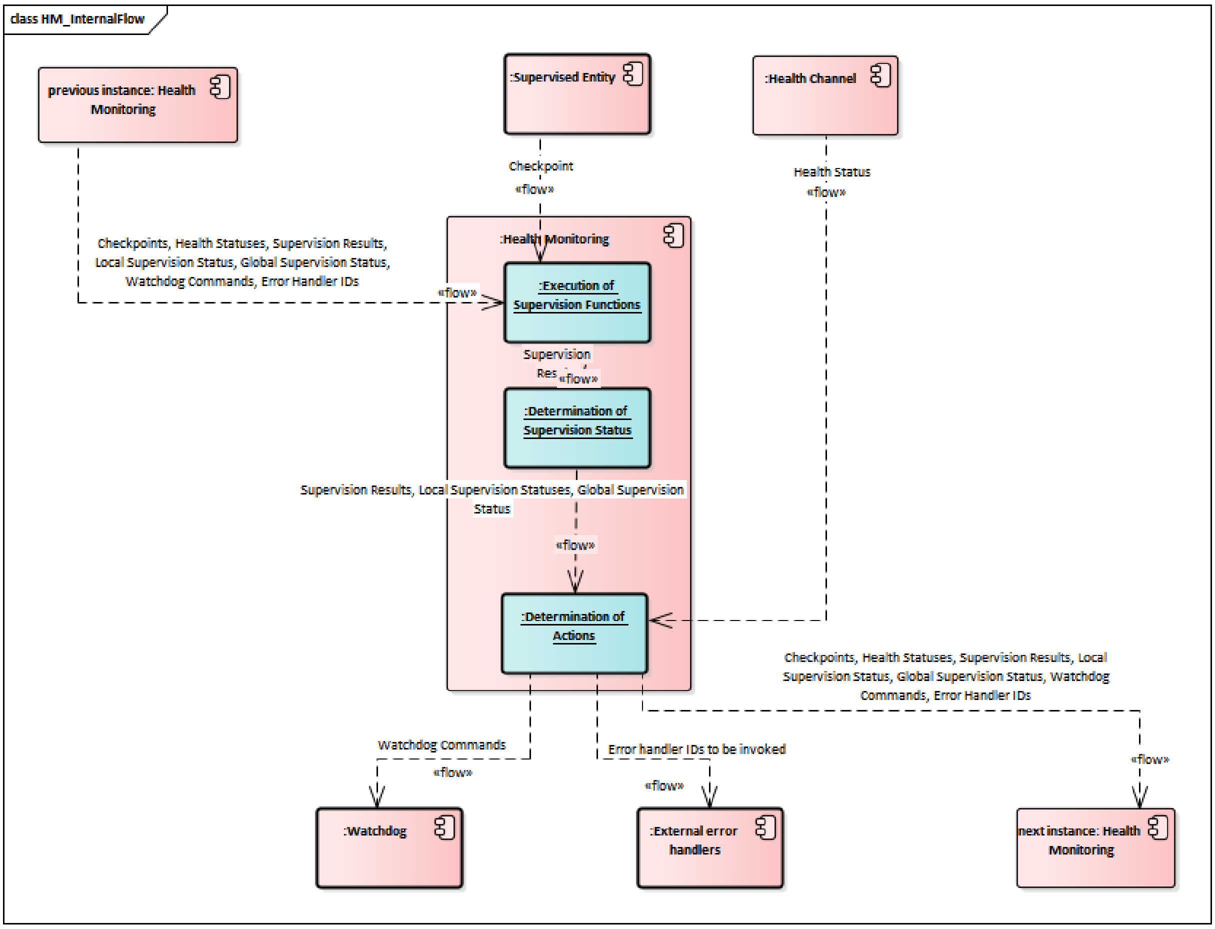

Figure 6.3: Main functions of Health Monitoring

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.

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

Based on the Global Supervision Status, the error handling and watchdog handling take place.

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), the Checkpoints 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)

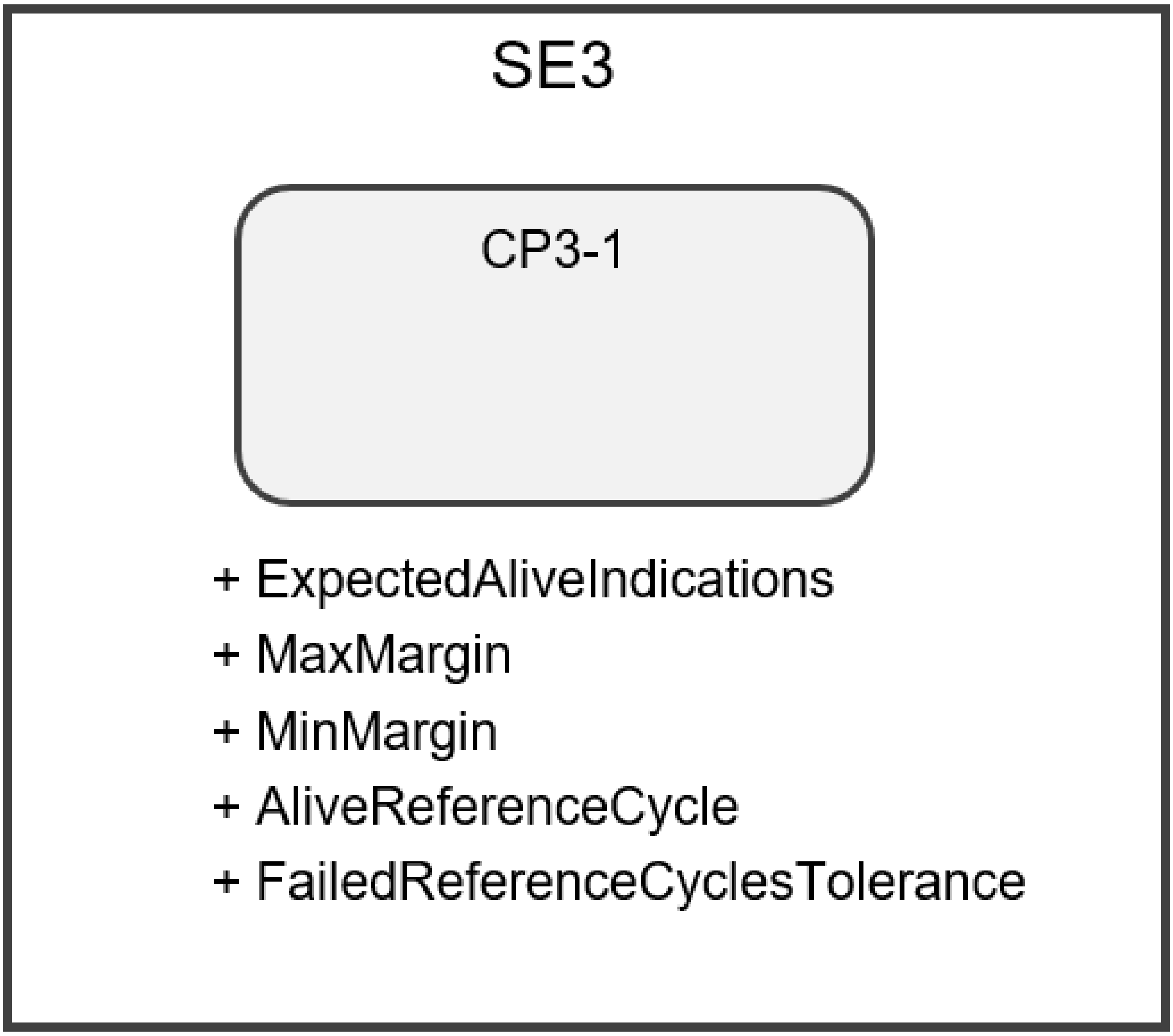

Figure 6.4: Simplest Alive Supervision Checkpoint Configuration for a given Supervision Mode

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

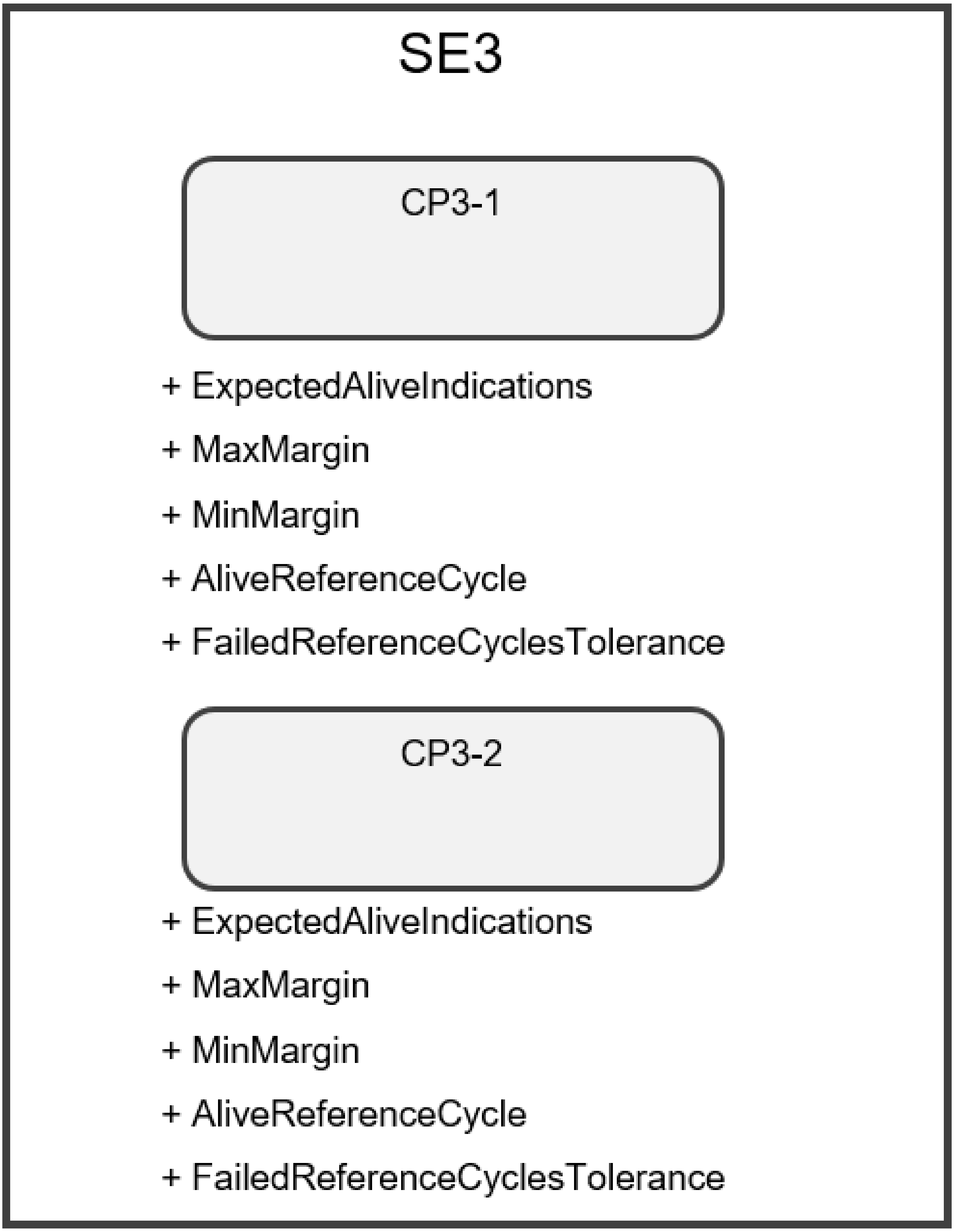

Figure 6.5: Multiple Checkpoints for Alive Supervision in one SupervisedEntity for a given Supervision Mode

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 of Checkpoints.

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 of Checkpoints 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.

[ASWS_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, RS_HM_09249)*

[ASWS_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, RS_HM_09249)*

If any Checkpoint of a SupervisedEntity fails the examination, then the result of Alive Supervision at this AliveReferenceCycle for the SupervisedEntity is set to incorrect. Otherwise, it is set to correct.

[ASWS_HM_00075] ⌈On examination of the Alive Counter, if the result of Alive Supervision is determined to be incorrect then, counter for failed alive supervision reference cycles shall be incremented unless it exceeds (is not greater than) configured

Failure Tolerance (see configuration parameter FailedReferenceCyclesTolerance).⌋(RS_HM_09163)

[ASWS_HM_00079] ⌈On examination of the Alive Counter, if the result of Alive Supervision is determined to be correct then, counter for failed alive supervision reference cycles shall be decremented unless it is zero.⌋(RS_HM_09163)

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

[ASWS_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, RS_HM_09249)

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

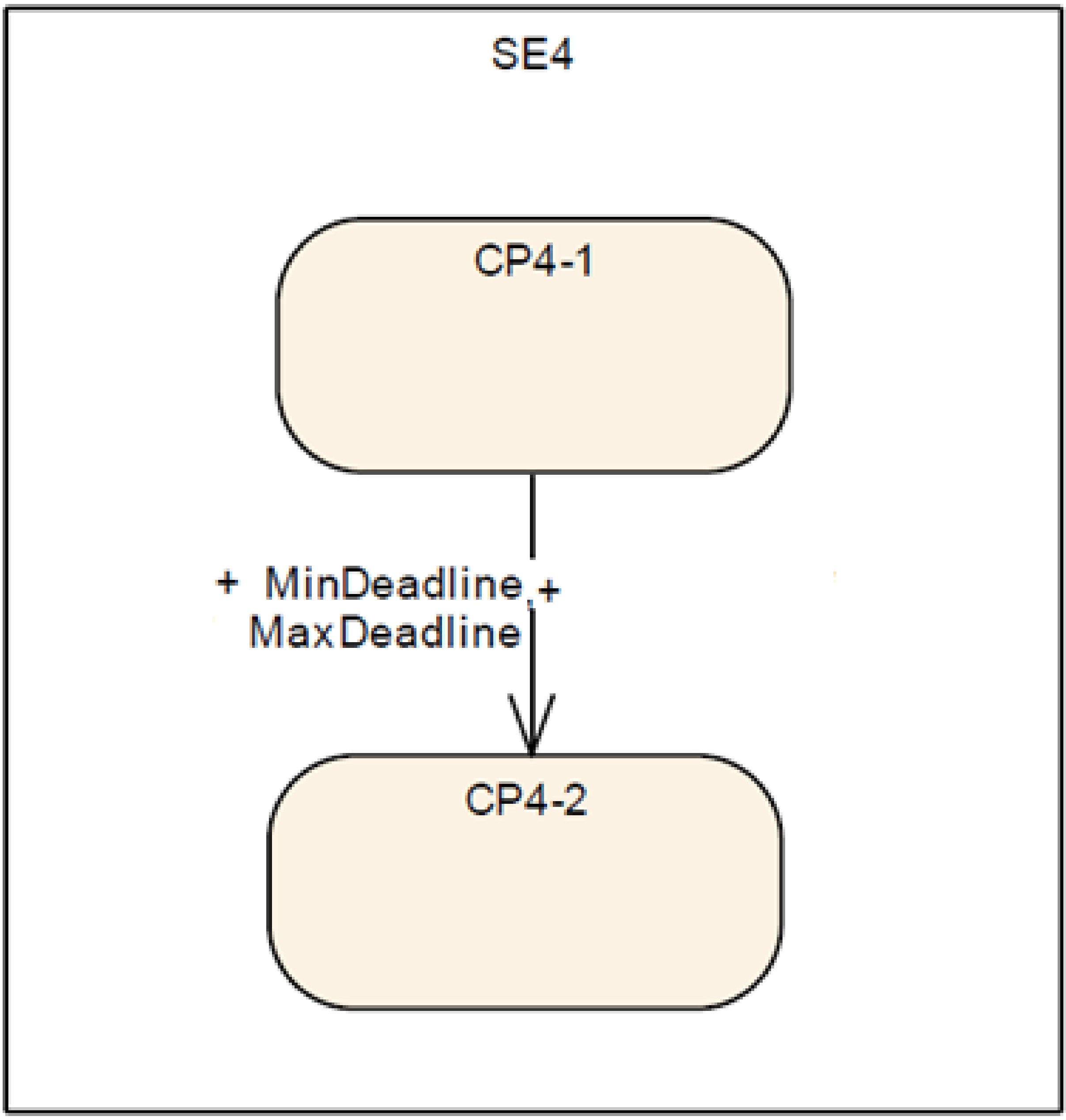

Figure 6.6: Simplest Deadline Supervision Configuration for a given Supervision Mode

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

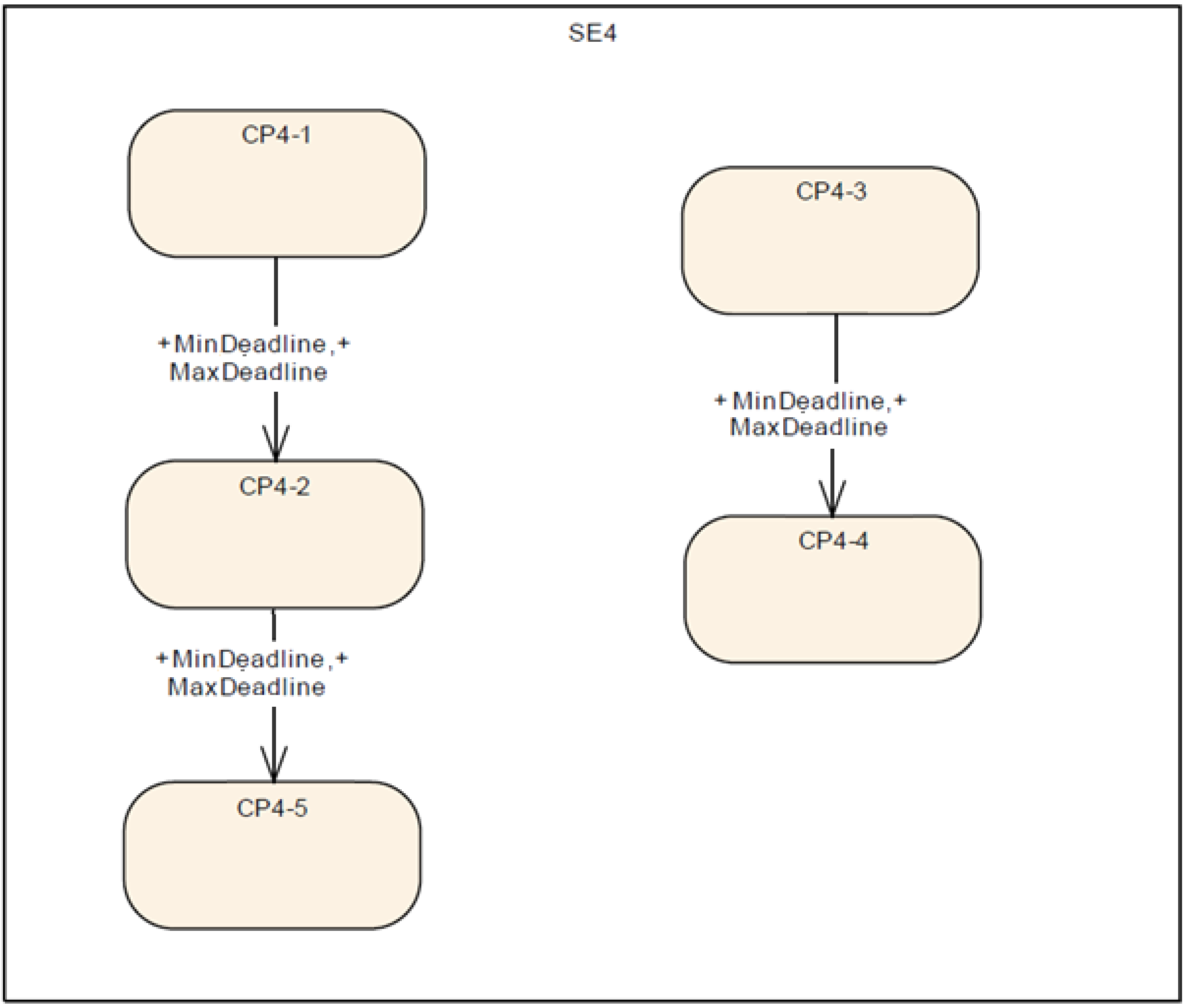

Figure 6.7: Multiple Transitions for `Deadline Supervision` in one `Supervised Entity` for a given Supervision Mode

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 Source `Checkpoint` and Target `Checkpoint` (see `DeadlineSupervision`). A Transition has minimum and maximum time `MinDeadline`, `MaxDeadline`.

6.2.2.2 Deadline Supervision Algorithm

When a Source Checkpoint (i.e. the Source Checkpoint referenced by the CheckpointTransition, see DeadlineSupervision) or a Target Checkpoint is reached, a SupervisedEntity invokes the function ReportCheckpoint, which will calculate the time expired between the Source Checkpoint and the Target Checkpoint.

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

[ASWS_HM_00294] ⌈If the time difference between the Target Checkpoint and the Source Checkpoint is not within the minimum and the maximum limits (that is, the time difference is either less than MinDeadline or greater than MaxDeadline), then the result of DeadlineSupervision for this SupervisedEntity shall be defined as incorrect. Otherwise, it shall be defined as correct.⌋*(RS_HM_09235, RS_HM_09249)*

[ASWS_HM_00228] ⌈If the Target Checkpoint is not reached even though the time since reaching the Source Checkpoint has crossed the maximum limit (that is, the time elapsed since reaching Source Checkpoint is greater than MaxDeadline), then the result of DeadlineSupervision for this SupervisedEntity shall be defined as incorrect.⌋*(RS_HM_09235, RS_HM_09249)*

[ASWS_HM_00229] ⌈When a given Source Checkpoint is reached two or more times on or before the expiration of the maximum limit without reaching the corresponding Target 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, RS_HM_09249)*

[ASWS_HM_00354] ⌈When a given Target Checkpoint is reached before the occurrence of the corresponding Source 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, RS_HM_09249)*

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

[ASWS_HM_00299] ⌈For any reported Checkpoint that is neither a Source Checkpoint nor a Target Checkpoint , the function ReportCheckpoint shall ignore this Checkpoint and not update the result of the Deadline Supervision for the Supervised Entity.⌋*(RS_HM_09235, RS_HM_09249)*

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 `Checkpoint`s 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 `Graph` are not possible.

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

```
CP0-0  initialize();
CP0-1  while (subsystem is running) {
CP0-2      if (condition_A)
CP0-3          run subtask_A;
           else
CP0-4          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.

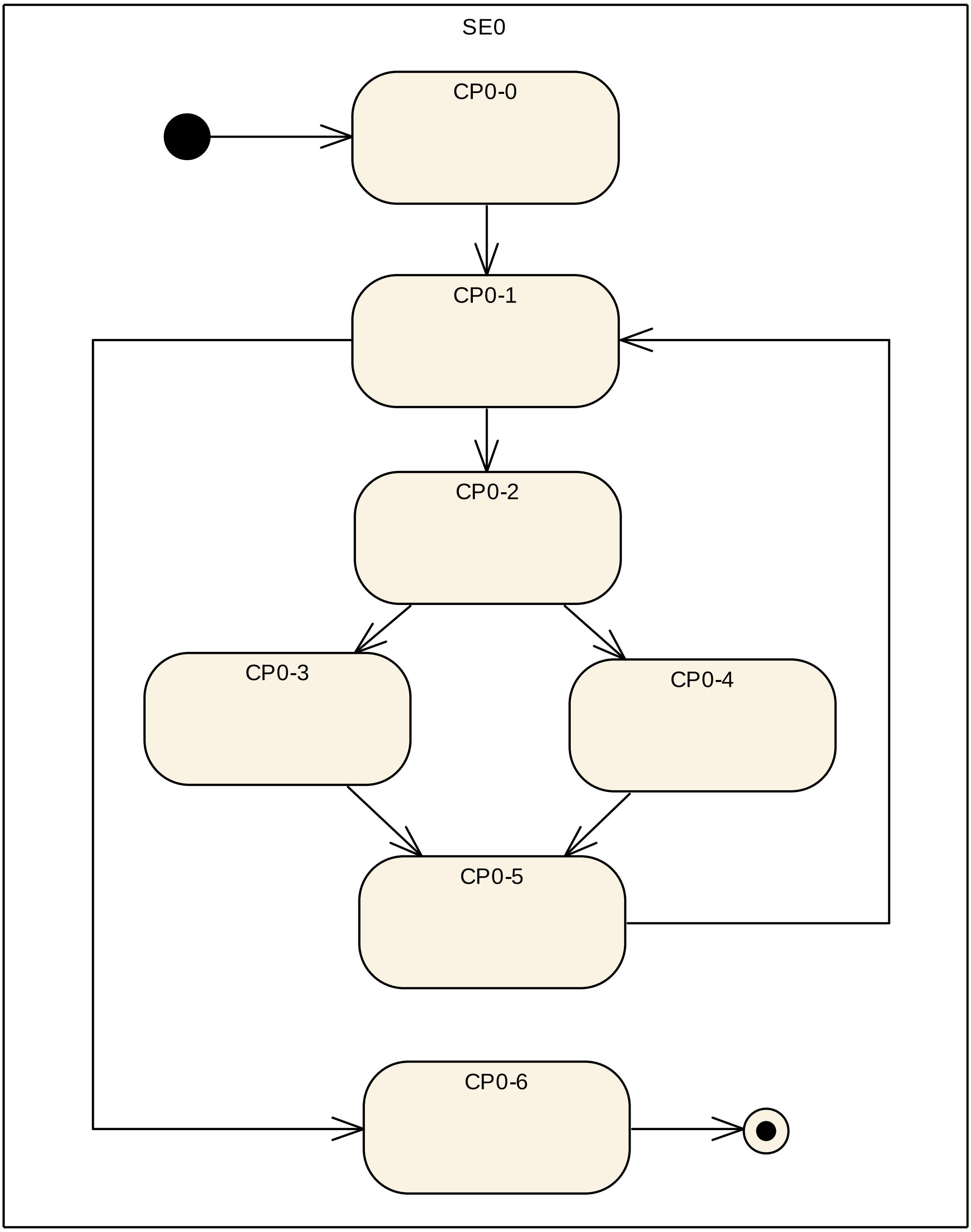

Figure 6.9: Example Control Flow Graph

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.

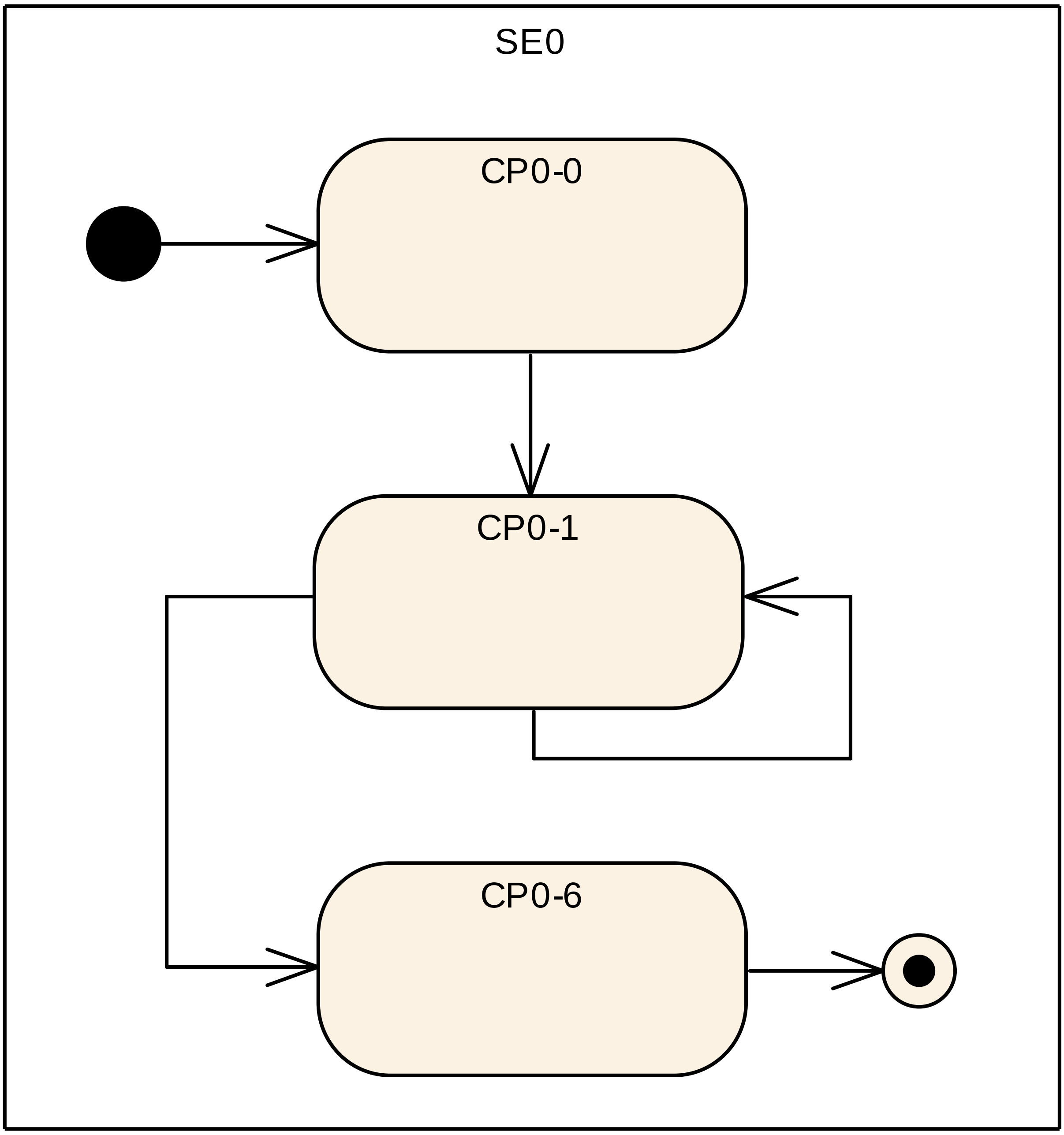

Figure 6.10: Abstracted Example Control Flow Graph

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

[ASWS_HM_00271] ⌈The Health Monitoring shall mantain the activity status of each Graph.⌋*(RS_HM_09222, RS_HM_09249)*

[ASWS_HM_00296] ⌈At the initialization, the Health Monitoring shall consider each Graph as inactive.⌋*(RS_HM_09222, RS_HM_09249)*

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, which results with execution of Logical Supervision algorithm.

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

[ASWS_HM_00295] ⌈The function ReportCheckpoint shall identify to which one Graph a reached Checkpoint belongs.⌋*(RS_HM_09222, RS_HM_09249)*

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

[ASWS_HM_00252] ⌈The function ReportCheckpoint 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 LogicalSupervision), then the result of this Logical Supervision within the SupervisedEntity of the reported Checkpoint is correct, otherwise incorrect.

2. Else if the Graph is active and all previously called Checkpoints of this Graph were called in the right sequence, 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 Target), then the result of this Logical Supervision for SupervisedEntity of the reported Checkpoint is correct, otherwise incorrect.

3. Else (i.e. the Graph is active, but at least one Checkpoint in this Graph was previously called in a wrong sequence):

   a. The result of this Logical Supervision of the Supervised Entity keeps 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, RS_HM_09249)*

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.

[ASWS_HM_00331] ⌈If the result of the Logical Supervision triggered by ReportCheckpoint is correct and the Checkpoint is defined as a final one, then the function ReportCheckpoint shall set Graph as inactive. After a final checkpoint, only initial checkpoints are possible.⌋*(RS_HM_09222, RS_HM_09249)*

[ASWS_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, RS_HM_09249)*

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

[ASWS_HM_00273] ⌈If the function ReportCheckpoint 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, RS_HM_09249)*

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 AliveSupervisions, result of DeadlineSupervisions, result of LogicalSupervisions involving Checkpoints corresponding to the Supervised Entity.

Details of determination of Local Supervision status is Platform Specific. Hence, it is not described in this document.

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.

Details of determination of Global Supervision status is Platform Specific. Hence, it is not described in this document.

6.4 System Health Monitoring

The previous chapters described Health Monitoring on platform level. In a distributed system using different platforms AP, CP, Non-AUTOSAR, a global monitor is necessary for evaluating and sharing health information on a vehicle level.

A standardized format for Health Indicator will be introduced for sharing health information of platforms, features, domains or even vehicles. These Health Indicator can either be used for platform level recovery actions, or to enhance services with a Health of Service, similar to Quality of Service (QoS).

Abstract interfaces for System Health Monitor to local health monitors shall be specified, allowing platform agnostic health management of several Adaptive, Classic and third-party platforms.

6.4.1 System Health Monitoring Architecture

The SystemHealthMonitor is intended for platform agnostic safety monitoring. For this reason the SystemHealthMonitor is introduced as an abstract component according to AUTOSAR_TPS_AbstractPlatformSpecification. A SystemHealthMonitor gathers health information of abstract LocalHealthMonitors. These LocalHealthMonitors are deployed on platform level and collect the health information of the platform itself. The LocalHealthMonitor on platform level might be implemented as a client SystemHealthMonitor as seen in the [5, EXP-SHM], or some functional cluster. The local information might include monitoring results of Platform Health Monitor(in AP)/Watchdog Manager(in CP), State Manager(in AP)/Basic Software Mode Manager(in CP) or hardware information e.g highTemp. Components like the State Manager are highly project specific and it can thus not be fully standardized which information the `LocalHealthMonitor` reports.

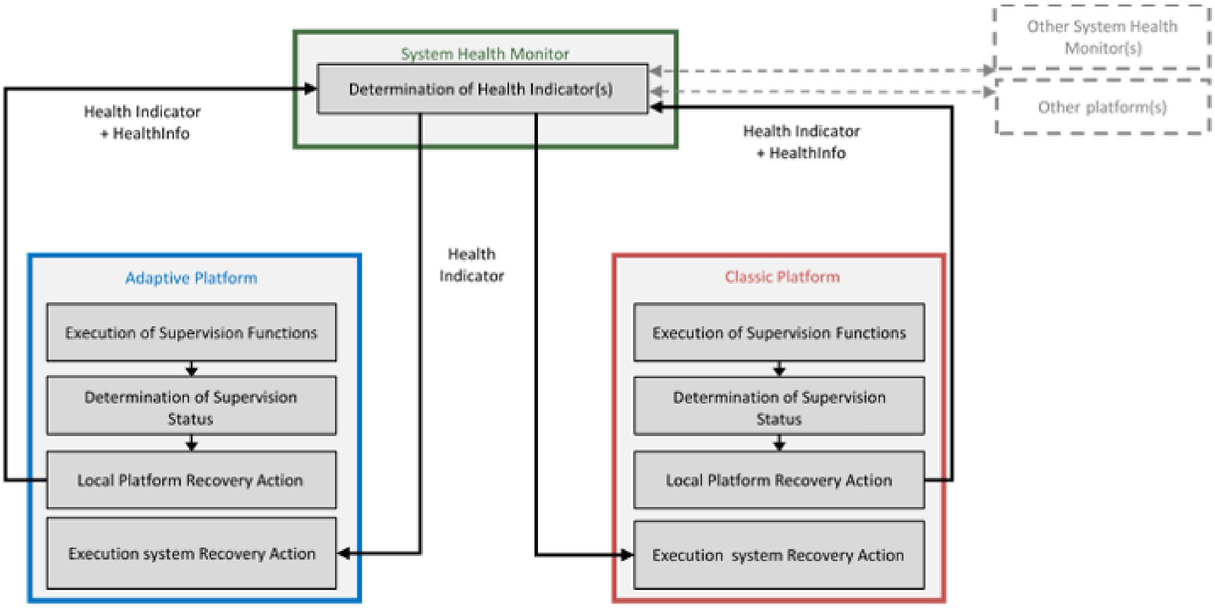

Figure 6.11: Overview of Health Information exchange between different platforms

The collected information can be used to create a platform `Health Indicator`, giving an overall estimation of the platform health.

[ASWS_HM_00501]{DRAFT} ⌈The `LocalHealthMonitor` shall create a platform `Health Indicator`, based on the locally reported health information.⌋*(RS_HM_09301, RS_HM_09304, RS_HM_09303)*

Information exchanged with SHM is considered safety relevant. Therefore, communication between SHM instance and local monitors and between multiple SHM instances shall be cyclic. Safety mechanisms like E2E protection shall be used to detect possible message loss, delay, alteration etc. The detectable errors depend on the chosen E2E profile and are project specific. Cycle exchange of Health Indicators can be used as periodical heart beat, giving an indication on the availability of the platforms and of SHM. A missed message means no confidence of correct behavior and should be considered in `Health Indicator` determination on SHM side and for recovery action on platform level.

[ASWS_HM_00502]{DRAFT} ⌈The platform `Health Indicator` and the local health information shall be cyclically reported to the `SystemHealthMonitor`.⌋*(RS_HM_09309, RS_HM_09303)*

[ASWS_HM_00509]{DRAFT} ⌈The `Health Indicator` calculated by SHM shall be reported cyclically to subscribers.⌋*(RS_HM_09309, RS_HM_09310, RS_HM_09303)*

[ASWS_HM_00503]{DRAFT} ⌈Information exchange between `LocalHealthMonitor` and `SystemHealthMonitor` shall be E2E protected.⌋*(RS_HM_09302, RS_HM_09303)*

[ASWS_HM_00504]{DRAFT} ⌈The SystemHealthMonitor shall gather and evaluate health information of all LocalHealthMonitors in its subsystem. Together with Health Indicators of other SystemHealthMonitors the subsystem information can be used to create Health Indicators at a higher level of abstraction.⌋*(RS_HM_09305, RS_HM_09304, RS_HM_09303)*

As one SystemHealthMonitor poses the threat of a single point of failure for its subsystem, multiple SystemHealthMonitors might receive the local health information, but only one of them should be actively calculating and providing the Health Indicators.

[ASWS_HM_00505]{DRAFT} ⌈A dedicated/particular Health Indicator shall be provided by only one SystemHealthMonitor at a given point of time.⌋*(RS_HM_09305, RS_HM_09303)*

6.4.2 Concept of Health Indicator

Health Indicators provide an evaluation metric of current system performance with regard to safety requirements. Health information of safety monitors is analyzed and used to determine Health Indicators on different abstraction levels. The Health Indicator is defined as a tuple of ID, Performance, Reliability, Timestamp and SubsystemState. The Performance rates the performance with respect to malfunctioning behavior. Reliability evaluates how much to trust the system due to uncertainties. SubsystemState is a systemspecific Health status of the Subsystem Sub = $\{sub_1,..., sub_n\}$. Different SubsystemStates are based on availability and availability requirements. Health Indicators can be results of supervisions on hardware, software, user, or the vehicle's environment. Combining monitoring results with well-defined safety properties, a corresponding health triple is determined. The three core parameters of the Health Indicator are supposed to capture different safety aspects required by different safety standards. The Degradation parameter is operating at the most abstract level. Only based on binary availability indications an overall degradation state is determined. ISO-26262 [1] and ISO-21448 [4] take further aspects into consideration than just the availability. ISO-26262 focuses on hazards arising from malfunctioning of E/E Systems whereas SOTIF refers to hazards caused by performance limitations. To this end, the scope of SOTIF demands including the vehicle's interaction with its environment, users, and other cars to capture uncertainties introduced by them. To include ISO-26262 and SOTIF into the Health Indicator, the Performance and Reliability parameters are used.

The timestamp can be used to store information when the HealthIndicator was created. A unique HealthIndicatorID shall be used to distinguish Health Indicators and assign them to a specific subsystem (e.g feature,platform,domain).

SystemHealthMonitors can operate on different abstraction levels. Monitoring results on platform level can be grouped on the level of functional features. Functional features might then be grouped in domains and all of this might give an health indication for the vehicle. These abstraction levels are not standardized and just given as an example. Each SystemHealthMonitor can handle multiple subsystems at different abstraction levels and thus provides multiple HealthIndicators.

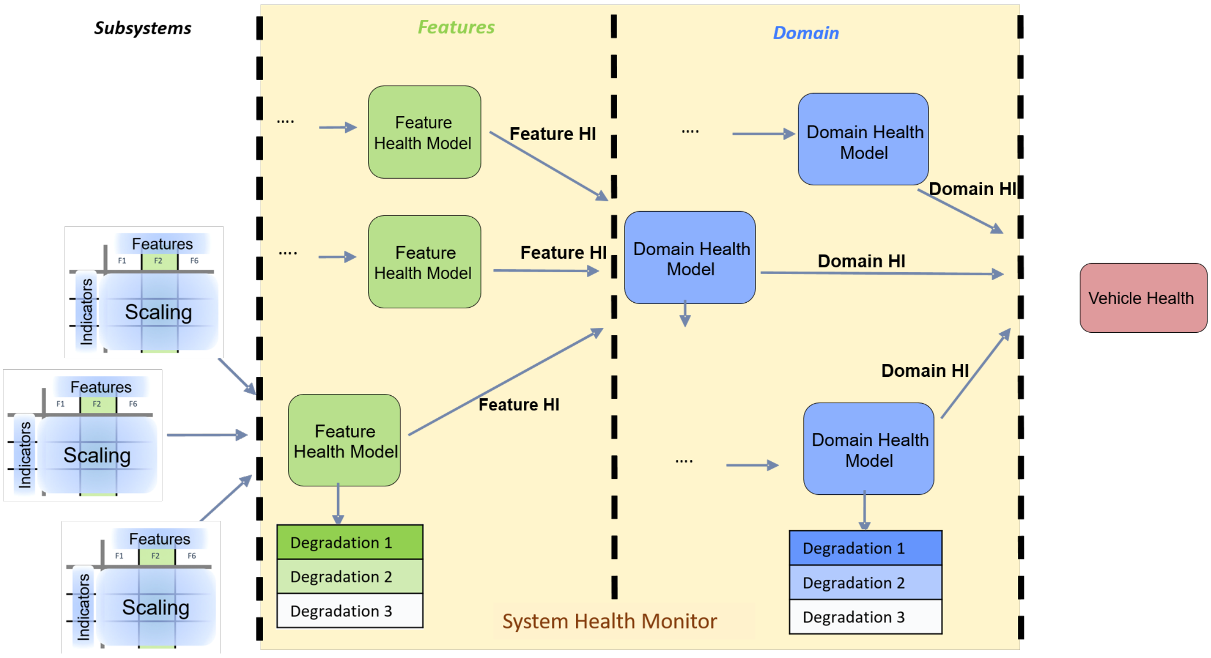

Figure 6.12: Example abstraction levels for Health Indicators

Health Indicators of subsystems can be used to build Health Indicators on feature level. These can then be combined to build Health Indicators on domain level and finally on vehicle level. Further explanation how these Health Indicators could look like for their respective domain can be found in the EXP_SHM.

[ASWS_HM_00506]{DRAFT} ⌈Reporting of Health Indicators from SHM to subscribers shall be E2E protected.⌋*(RS_HM_09308, RS_HM_09303)*

6.4.3 Application interfaces

For reporting the actual health information a standardized interface shall be used. The platform HealthIndicator can be provided over the HealthIndicatorInterface and local health information over the HealthInfoInterface. Local health information can contain health information from functional clusters e.g. supervision results from PHM/SM or external monitors (e.g voltage monitor). These interfaces are described as service interfaces in chapter 9

6.4.4 Usage of HealthIndicators

Health Indicators can be used for directly exchanging health information of subsystems. Each consumer interested in a specific Health Indicator can access it over the HealthIndicatorInterface. Local platform managers (State Manager/Basic Software Mode Manager) could use the HIs of other platforms to degrade their own platform or activate backup functions, for platforms with bad health. This would allow decentralized system degradation across multiple platforms. Similarly applications might want to know the HI of features providing them with input, in order to decide whether to trust this information.

7 Health Monitoring API specification

This chapter specifies the API of Health Monitoring that is referred in other document parts. It is defined in generic/abstract way, so that it can be implemented on different platforms. For exact API name and semantics please refer to corresponding Platform specific documents ([6] in case of Classic Platform and [7] in case of Adaptive platform).

7.1 Provided API

7.1.1 Reporting Checkpoints

Health Monitoring provides a method to report the current code location, represented by a Checkpoint

```
ReportCheckpoint(CheckpointID id)
```

7.1.2 Reporting health status

Health Monitoring provides a method to report the health status information

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

7.1.3 Forwarding information between health monitoring components

Health Monitoring provides 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.

```
ReportHealthMonitoring(HealthMonitoring montoringData)
```

7.1.4 Init / DeInit

Health Monitoring provides a method to initialize the service.

```
Init()
```

Health Monitoring provides a method to deinitialize the service.

```
DeInit()
```

7.1.5 Retrieving Supervision Status from application

Health Monitoring provides a method to report the Local Status of a Supervised Entity to the application.

```
GetLocalStatus(LocalStatusType* LocalStatus)
```

Health Monitoring provides a method to report the Global Status to which the specified Supervised Entity belongs to the application.

```
GetGlobalStatus(GlobalStatusType* GlobalStatus)
```

7.2 Assumed API

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

7.2.1 Triggering error handling

Health Monitoring provides a method to trigger a defined error handler, providing the identifier of this error.

```
TriggerErrorHandler(ErrorID id)
```

7.2.2 Controlling watchdog

Health Monitoring provides a method to control the watchdog drivers.

```
ControlWatchdog(ControlData control)
```

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 `Machine` (representing MCU, virtual machine, partition) is split into two categories:

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

It means all supervision configuration is fully mode-dependent.

A system is made of several `Machines`. Therefore, `Health Monitoring` 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) are assumed to be provided externally (by other specifications) therefore they are only referenced here.

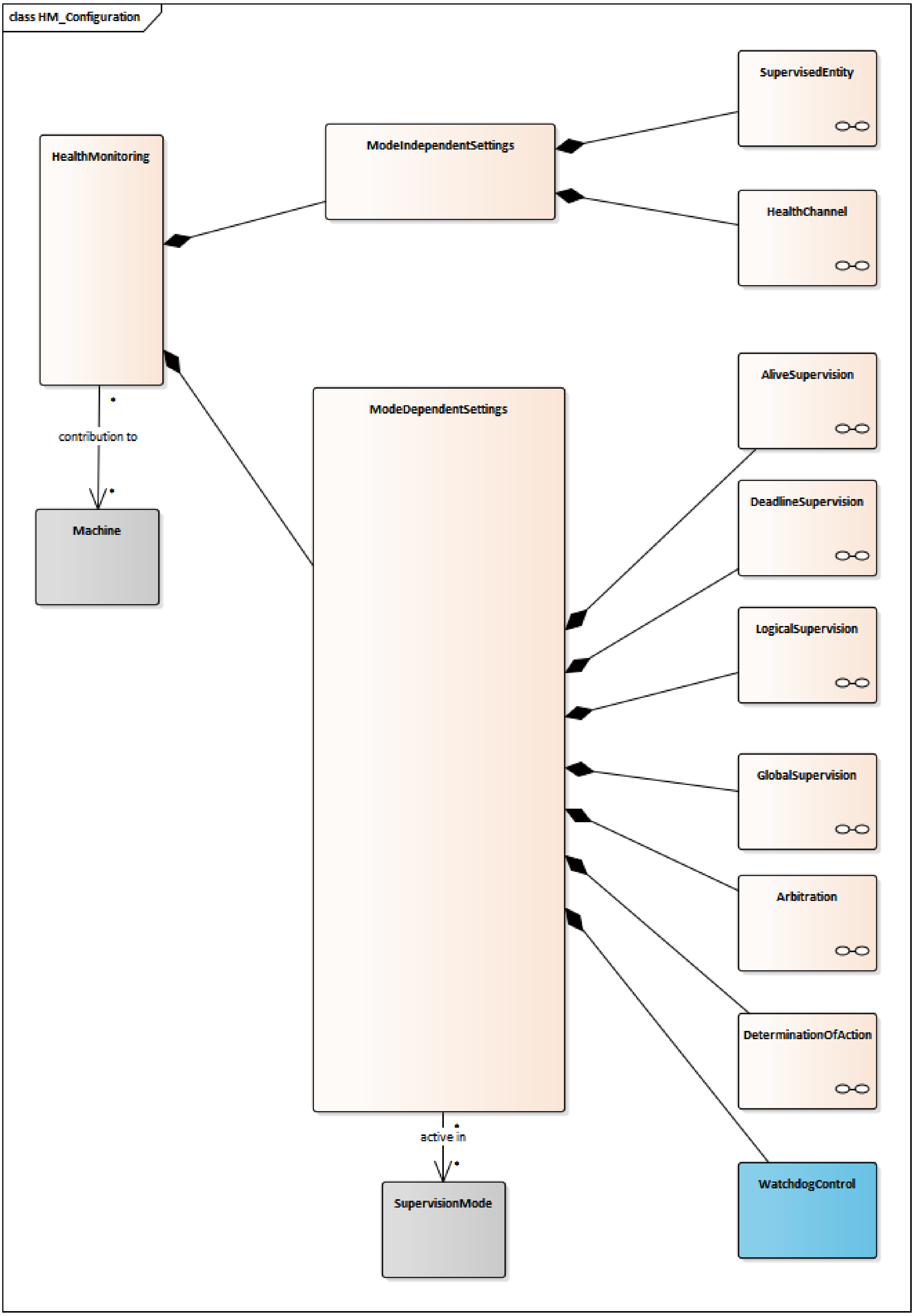

Figure 8.1: Overall configuration

8.2 Mode-independent settings

ModeIndependentSettings contain static information: what are possible SupervisedEntitys and possible Health Channel.

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

8.2.1 Supervised Entity

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

A SupervisedEntity has the following options:

1. Name: Globally unique name identifier, used by Applications
2. ID: Globally unique identifier (number)

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

A has the following options:

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

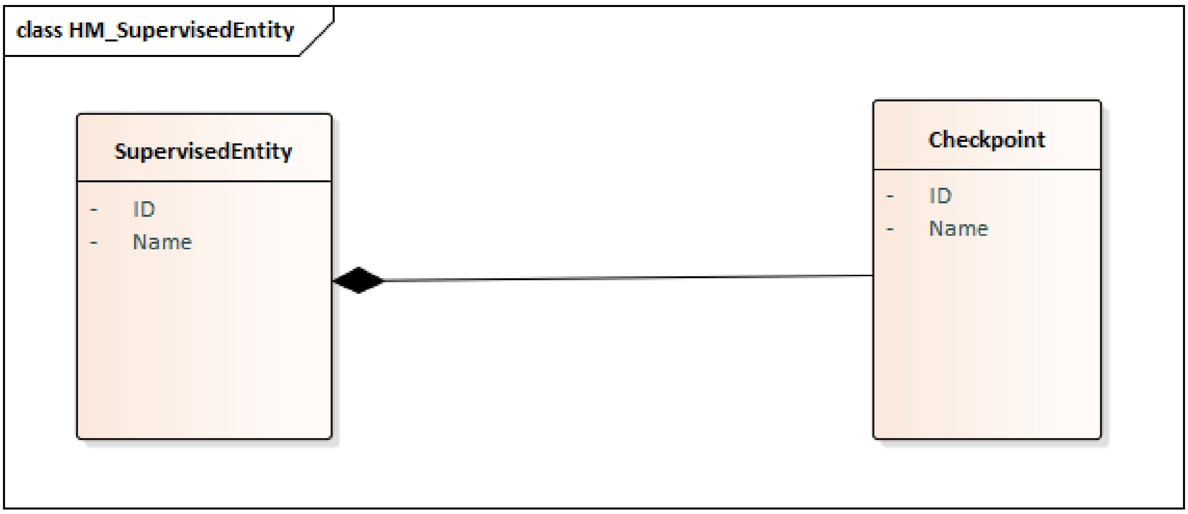

Figure 8.2: Supervised Entity

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

8.3 Mode-dependent settings

ModeDependentSettings 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

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

AliveSupervision has the following options:

1. AliveReferenceCycle: time period at which the Alive Supervision mechanism compares the amount of received Alive Indications of the Checkpoint against the expected/configured amount.
2. ExpectedAliveIndications: the amount of expected alive indications of the Checkpoint within AliveReferenceCycle
3. MaxMargin: amount of acceptable missing alive indications within AliveReferenceCycle
4. MinMargin: amount of acceptable additional alive indications within AliveReferenceCycle
5. FailedReferenceCyclesTolerance: acceptable amount of AliveReferenceCycles with incorrect/failed alive supervision

A Checkpoint 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.

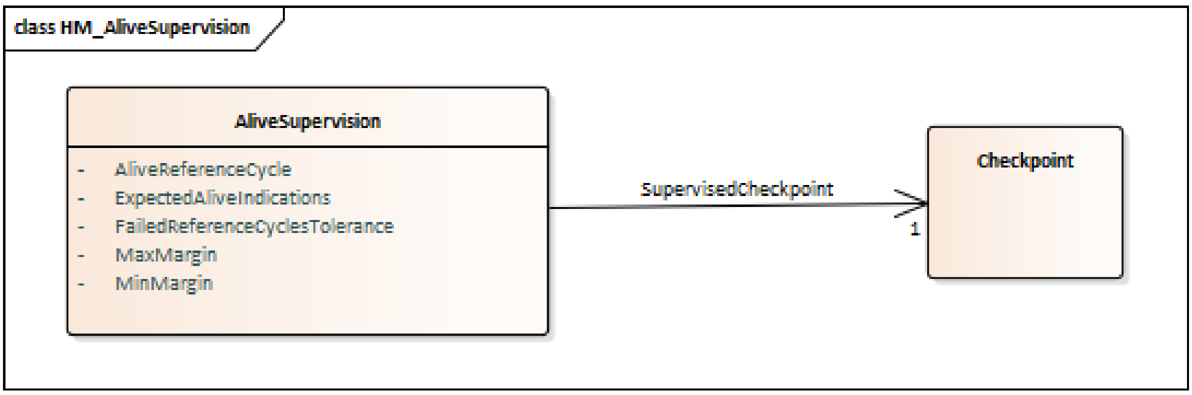

Figure 8.3: Alive Supervision

8.3.2 Deadline Supervision

DeadlineSupervision has the following options:

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

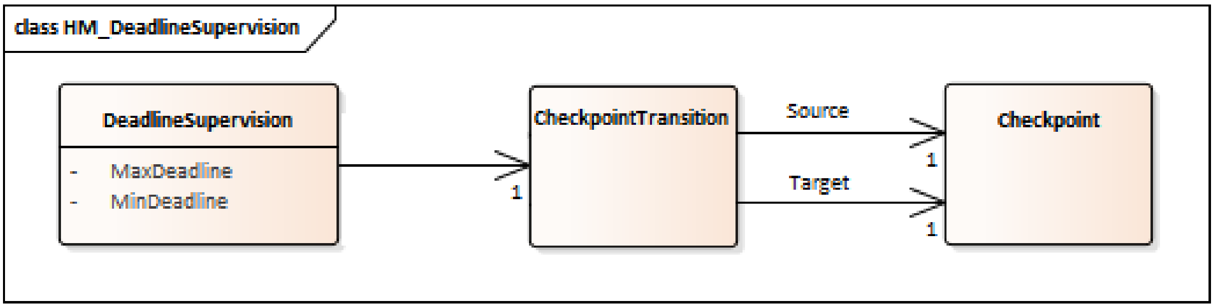

Figure 8.4: Deadline Supervision

8.3.3 Logical Supervision

LogicalSupervision is a collection of CheckpointTransitions.

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 CheckpointTransition has its Source and Target 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).

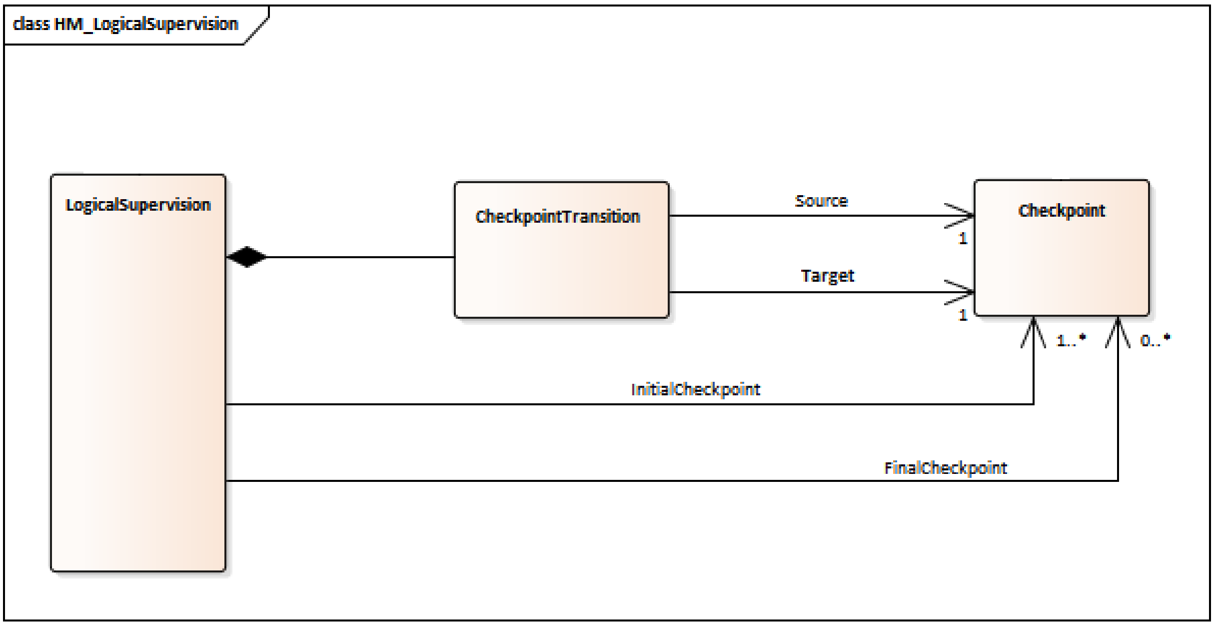

Figure 8.5: `Logical Supervision`

8.3.4 Global Supervision

A `GlobalSupervision` is an overall state of a software subsystem. There can be one or a few `GlobalSupervision`s per `Machine`.

`GlobalSupervision` is a "worst-of" of all contained `LocalSupervision`s.

`LocalSupervision` represents the state of a `SupervisedEntity`. It comprises of all AliveSupervisions, DeadlineSupervisions and LogicalSupervisions pertaining to a SupervisedEntity.

9 Service Interfaces

9.1 Type definitions

[ASWS_HM_00511]{DRAFT} ⌈

Name	HealthIndicator
Kind	STRUCTURE
Subelements	HealthIndicatorID uint8_t Timestamp uint32_t (optional) Performance int16_t (optional) Reliability int16_t (optional) SubsystemState enum [uint8_t] (optional)
Derived from	-
Description	Health Indicator provides an evaluation metric of current system performance with regard to safety requirements

⌋*(RS_HM_09303)*

[ASWS_HM_00515]{DRAFT} ⌈

Name	HealthInfo
Kind	STRUCTURE
Subelements	GlobalSupervisionInfoVector (optional) HealthChannelInfoVector (optional) FunctionGroupInfoVector (optional) LocalSupervisionInfoVector (optional) BswMModeName string (optional)
Derived from	-
Description	Structure containing different Health Information pairs [Shortname+Value].

⌋*(RS_HM_09303)*

[ASWS_HM_00516]{DRAFT} ⌈

Name	GlobalSupervisionInfo
Kind	STRUCTURE
Subelements	Name string Status enum[uint8_t]
Derived from	-
Description	Structure containing Global Supervision Status information.

⌋*(RS_HM_09303)*

[ASWS_HM_00517]{DRAFT} ⌈

Name	GlobalSupervisionInfoVector
Kind	VECTOR
Subelements	GlobalSupervisionInfo
Derived from	-
Description	A list of Global Supervision Status Information

⌋*(RS_HM_09303)*

[ASWS_HM_00518]{DRAFT} ⌈

Name	FunctionGroupInfo
Kind	STRUCTURE
Subelements	Name string State string
Derived from	-
Description	Structure containing a Function Group State.

⌋*(RS_HM_09303)*

[ASWS_HM_00519]{DRAFT} ⌈

Name	FunctionGroupInfoVector
Kind	VECTOR
Subelements	FunctionGroupInfo
Derived from	-
Description	A list of Function Group State Information

⌋*(RS_HM_09303)*

[ASWS_HM_00520]{DRAFT} ⌈

Name	HealthChannelInfo
Kind	STRUCTURE
Subelements	Name string Status string
Derived from	-
Description	A structure containing a Health Channel Status information.

⌋*(RS_HM_09303)*

[ASWS_HM_00521]{DRAFT} ⌈

Name	HealthChannelInfoVector
Kind	VECTOR
Subelements	HealthChannelInfo

▽

△

Derived from	-
Description	A list of Health Channel Status information.

⌋(*RS_HM_09303*)

[ASWS_HM_00522]{DRAFT} ⌈

Name	LocalSupervisionInfo
Kind	STRUCTURE
Subelements	Name string Status enum[uint8_t]
Derived from	-
Description	Structure containing a Local Supervision Status

⌋(*RS_HM_09303*)

[ASWS_HM_00523]{DRAFT} ⌈

Name	LocalSupervisionInfoVector
Kind	VECTOR
Subelements	LocalSupervisionInfo
Derived from	-
Description	A list of Local Supervision Status Information

⌋(*RS_HM_09303*)

Note: Following Health Information are supported in Adaptive Platform:

- GlobalSupervisionInfo
- HealthChannelInfo
- FunctionGroupInfo

Following Health Information are supported in Classic Platform:

- GlobalSupervisionInfo
- LocalSupervisionInfo
- BswMModeName

9.2 Provided Service Interfaces

9.2.1 HealthIndicator

Port **[ASWS_HM_00510]**{DRAFT} ⌈

Name	HealthIndicatorInterface		
Kind	ProvidedPort	Interface	HealthIndicatorInterface
Description	Report HealthIndicator		
Variation			

⌋(*RS_HM_09300*, *RS_HM_09303*)

Service Interface **[ASWS_HM_00512]**{DRAFT} ⌈

Name	HealthIndicator
Events	HealthIndicatorEvent
Description	The reported Health Indicator.
Type	HealthIndicator

⌋(*RS_HM_09303*)

9.2.2 HealthInfo

Port **[ASWS_HM_00513]**{DRAFT} ⌈

Name	HealthInfoInterface		
Kind	ProvidedPort	Interface	HealthInfoInterface
Description	Report HealthInfo		
Variation			

⌋(*RS_HM_09301*, *RS_HM_09303*)

Service Interface **[ASWS_HM_00514]**{DRAFT} ⌈

Name	HealthInfo
Events	HealthInfoEvent
Description	The reported Health Information
Type	HealthInfo

⌋(*RS_HM_09303*)