Document Title	Explanation of Adaptive Platform Software Architecture
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	982

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R20-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2020-11-30	R20-11	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Introduction

This explanatory document provides an overview of the software architecture of the AUTOSAR Adaptive Platform standard.

1.1 Objectives

This document is an architecture description of the AUTOSAR Adaptive Platform in accordance to [1, ISO/IEC 42010] and has the following main objectives:

- Identify the **stakeholders** of the AUTOSAR Adaptive Platform and their **concerns**.
- Identify the **system scope** and provide **overview information** of the AUTOSAR Adaptive Platform.
- Provide definitions for all used **architecture viewpoints** and a **mapping of all stakeholder concerns to those viewpoints**.
- Provide an **architecture view** and its **architecture models** for each architecture viewpoint used in this architecture description.
- Provide **correspondence rules** and **correspondences** among the contents of this architecture description.
- Provide an **architecture rationale** (explanation, justification, reasoning for decisions made) on a high level. A more in-depth documentation of decisions is provided in [2, EXP_SWArchitecturalDecisions].
- Provide a **record of known inconsistencies** among the architecture description.

Please note that the AUTOSAR Adaptive Platform standard is defined by means of requirements and software specification documents. Those documents deliberately lack specifications of dependencies and interfaces between the building blocks of the AUTOSAR Adaptive Platform to provide more degrees of freedom for stack vendors in their solution design. This document describes the original architectural design of the AUTOSAR Adaptive Platform including details how the building blocks should interact with each other. It is an example how an implementation of the standard should work internally. However, a stack vendor is free to choose another design as long as it complies with the binding AUTOSAR Adaptive Platform standard.

1.2 Scope

This explanatory document applies to the AUTOSAR Adaptive Platform. It is recommended to get an overview of the AUTOSAR Adaptive Platform for all members of the working groups, stack vendors, and application developers.

1.3 Document Structure

This document is organized as follows. Section 4 provides an overview of the main requirements for the AUTOSAR Adaptive Platform, the top quality goals of its architecture, and a list of stakeholders that are affected by it. Section 5 lists requirements that constrain design and implementation decisions or decisions about the development process.

Section 6 is the base for discovering trade-offs and sensitivity points in the architecture of the AUTOSAR Adaptive Platform by introducing a quality attribute tree followed by the most important quality scenarios. The system context in which the AUTOSAR Adaptive Platform is intended to be used is outlined in section 7. Section 8 summarizes the fundamental decisions and solution strategies, that shape the architecture of the AUTOSAR Adaptive Platform such as technology decisions or architectural patterns to be used.

Sections 9 to 11 explain the software architecture from different view points. First, section 9 explains the decomposition of the AUTOSAR Adaptive Platform into Functional Clusters and their interdependencies. Then, section 10 demonstrates how the main use cases are realized using the Functional Clusters in the AUTOSAR Adaptive Platform. Section 11 shows different scenarios how applications based on the AUTOSAR Adaptive Platform may be deployed.

Section 12 provides an overview of concepts and patterns used by the AUTOSAR Adaptive Platform. Section 13 lists and rates risks associated with the architecture of the AUTOSAR Adaptive Platform and technical debt.

2 Definition of Terms and Acronyms

2.1 Acronyms and Abbreviations

Abbreviation / Acronym	Description
DoIP	Diagnostics over Internet Protocol
POSIX	Portable Operating System Interface
SecOC	AUTOSAR Secure Onboard Communication
TLS	Transport Layer Security
UML	Unified Modeling Language

2.2 Definition of Terms

This section lists terms that are specific to this document. A list of general terms for AUTOSAR is provided in the [3, glossary].

Term	Description
Functional Cluster	A logical group of functionality within the AUTOSAR Adaptive Platform. `Functional Cluster`s are the second level of abstraction in the building block view (cf. Chapter 9). They are also subject of the individual specification documents that make up the AUTOSAR Adaptive Platform standard.
Function Group	A set of modeled `Process`es. See Section 12.2 for details.
Thread	The smallest sequence of instructions the can be managed independently by a scheduler. Multiple `Thread`s can be executed concurrently within one `Process` sharing resources such as memory.
Watchdog	An external component that supervises execution of the AUTOSAR Adaptive Platform. See Section 7.2.3 for details.

3 Related Documentation

This document provides a high-level overview of the AUTOSAR Adaptive Platform architecture. It is closely related to general requirements for AUTOSAR Adaptive Platform specified in [4, RS_Main] and [5, RS_General], and the architectural decisions documented in [2, EXP_SWArchitecturalDecisions].

The individual building blocks of the architecture (`Functional Cluster`s) are specified in separate documents. Each `Functional Cluster` defines one or more requirements specification(s) (RS document), one or more software specification(s) (SWS document) and one or more explanatory document(s) (EXP document). Please refer to these documents for any details on the AUTOSAR Adaptive Platform standard.

4 Overview and Goals

In conventional automotive systems ECUs are used to replace or augment electro-mechanical systems. Those resource constrained,deeply-embedded ECUs typically perform basic control functions by creating electrical output signals (e.g. for actors) based on input signals (e.g. from sensors) and information from other ECUs connected to the vehicle network. Much of the control software is specifically designed and implemented for the target vehicle and does not change significantly during vehicle lifetime. The AUTOSAR Classic Platform standard addresses the needs of these deeply-embedded systems.

Recent and future vehicle functions, such as highly automated driving, will introduce complex and computing resource demanding software that shall fulfill strict safety, integrity and security requirements. Such software performs for example, environment perception and behavior planning, and interacts with external backend and infrastructure systems. The software in the vehicle regularly needs to be updated during the life-cycle of the vehicle, due to evolving external systems, improved or added functionality, or security problems. The AUTOSAR Classic Platform standard cannot fulfill the needs of such systems. Therefore, AUTOSAR specifies a second software platform, the AUTOSAR Adaptive Platform. It provides high-performance computing and communication mechanisms as well as a flexible software configuration, for example, to support software update over-the-air. Features that are specifically defined for the AUTOSAR Classic Platform, such as access to electrical signals and automotive specific bus systems, can be integrated into the AUTOSAR Adaptive Platform but is not in the focus of standardization.

4.1 Requirements Overview

This section provides an overview of the basic requirements for the AUTOSAR Adaptive Platform that impact its architecture. The corresponding requirement identifiers are provided in square brackets. Please refer to [4, RS_Main] and [5, RS_General] for any details, rationale or intended use-cases of these requirements.

Support of state-of-the-art Technology

The AUTOSAR Adaptive Platform aims to support resource-intensive (memory, cpu) applications on state-of-the-art hardware. Therefore, the AUTOSAR Adaptive Platform shall support high performance computing platforms [RS_Main_00002] as well as virtualized environments [RS_Main_00511]. The AUTOSAR Adaptive Platform shall be able to run multiple applications in parallel [RS_Main_00049], each with concurrent application internal control flows [RS_Main_00050].

Software Update and Configuration

The AUTOSAR Adaptive Platform shall support a flexible (configuration) data and software update. Hereby, AUTOSAR Adaptive Platform shall support up- and download of such update packages [RS_Main_00650] and change of communication and application software at runtime [RS_Main_00503].

AUTOSAR shall provide a unified way to describe software systems deployed to Adaptive and / or Classic platforms [RS_Main_00161]. That kind of description shall also support the deployment and reallocation of AUTOSAR Application Software [RS_Main_00150], and shall provide means to describe interfaces of the entire system [RS_Main_00160].

Security

The AUTOSAR Adaptive Platform shall support the development of secure systems [RS_Main_00514] with secure access to ECU data and services [RS_Main_00170], and secure onboard communication [RS_Main_00510].

Safety

The AUTOSAR Adaptive Platform shall support the development of safety related systems [RS_Main_00010] that are reliable [RS_Main_00011] and highly available [RS_Main_00012]. The AUTOSAR Adaptive Platform specifications shall be analyzable and support methods to demonstrate the achievement of safety related properties accordingly [RS_Main_00350].

Reuse and Interoperability

The AUTOSAR Adaptive Platform shall support standardized interoperability with non-AUTOSAR software [RS_Main_00190] as well as (source code) portability for AUTOSAR Adaptive Applications across different implementations of the platform [RS_AP_00111]. Hereby, the AUTOSAR Adaptive Platform shall provide means to describe a component model for application software [RS_Main_00080], and support bindings for different programming languages [RS_Main_00513].

Communication

The AUTOSAR Adaptive Platform shall support standardized automotive communication protocols [RS_Main_00280] for intra ECU communication [RS_Main_01001] with different network topologies [RS_Main_00230].

Diagnostics

The AUTOSAR Adaptive Platform shall provide diagnostics means during runtime for production and services purposes [RS_Main_00260].

4.2 Quality Goals

This section will list the top quality goals for the architecture whose fulfillment is of highest importance to the major stakeholders in a future version of this document. Please refer to the currently un-prioritized list of Quality Attributes in Section 6.1.

4.3 Stakeholders

This section lists the stakeholders of the AUTOSAR Adaptive Platform architecture and their main expectations.

Role	Expectation
Project Leader	Overview of technical risks and technical debt in the AUTOSAR Adaptive Platform.
Working Group Architecture	Concise yet thorough documentation of the goals and driving forces of the AUTOSAR Adaptive Platform. Documentation of the original architectural design of the AUTOSAR Adaptive Platform standard. Documentation of identified technical risks and technical debt in the AUTOSAR Adaptive Platform.
Working Group	Consolidated overview of the AUTOSAR Adaptive Platform architecture. Realization of use-cases that span multiple `Functional Cluster`s. Usage of interfaces within the AUTOSAR Adaptive Platform. Guidelines and patterns for `Functional Cluster` and interface design.
Stack Developer	Consolidated overview of the original architectural design of the AUTOSAR Adaptive Platform. Realization of use-cases that span multiple `Functional Cluster`s. Usage of interfaces within the AUTOSAR Adaptive Platform.
Application Developer	Overview of the building blocks of the AUTOSAR Adaptive Platform and their purpose and provided functionality. Explanation of the concepts used in the AUTOSAR Adaptive Platform.

Table 4.1: Stakeholder table with roles and expectations

5 Architecture Constraints

AUTOSAR is a worldwide development partnership of vehicle manufacturers, suppliers, service providers and companies from the automotive electronics, semiconductor and software industry. AUTOSAR standardizes the AUTOSAR Adaptive Platform automotive middleware. The AUTOSAR Adaptive Platform is not a concrete implementation. The AUTOSAR Adaptive Platform standard leaves a certain degree of freedom to implementers of the standard, as most standards do. On the one hand, more freedom enables competition among the different implementations and a broader choice of properties for users of the AUTOSAR Adaptive Platform. On the other hand, a more strict standardization makes the different implementations compatible and exchangeable (within the standardized scope). Naturally, those attributes are in conflict. It is usually a choice of the standardization organization to evaluate the attributes and define the desired level of strictness.

The AUTOSAR Classic Platform is rather strict in that sense by specifying a detailed layered software architecture imposing many constraints on its implementations. The AUTOSAR Adaptive Platform launched with a less strict approach. That less strict approach puts constraints on the AUTOSAR Adaptive Platform architecture as outlined below.

5.1 Internal Interfaces

An important architectural constraint is that only interfaces that are intended to be used by applications or interfaces that are used to extend the functionality of the AUTOSAR Adaptive Platform shall be standardized. Internal interfaces between the building blocks of the AUTOSAR Adaptive Platform shall not be standardized. This approach leaves a lot of freedom to design and optimize the internals of an AUTOSAR Adaptive Platform stack. However, it also imposes constraints on how the AUTOSAR Adaptive Platform architecture can be defined and described in this document. Also, this means that it might not be possible to use different functional clusters from different AUTOSAR Adaptive Platform stack vendors.

First, the existence of internal interfaces and their usage by other building blocks is in most cases a recommendation and reflects the original design approach of the authors of the standard. The same applies to any interactions described in this document that make use of such internal interfaces.

Second, some quality attributes may be hard to ensure in general by the architecture of the standard. Additional measures like security or safety considerations lack well-defined inputs such as data flows or specifications of interdependencies. Consequently, a more thorough design phase is required when an AUTOSAR Adaptive Platform stack is implemented.

5.2 Distributed Work

Standardization of the AUTOSAR Adaptive Platform is a worldwide distributed effort. The individual building blocks are specified by dedicated working groups in separate documents to be able to scale in that distributed setup. This impacts the way the AUTOSAR Adaptive Platform architecture is described in this document.

First, this document shows interfaces on an architectural level only. This document does not specify details of interfaces such as individual operations. This keeps redundancies and thus dependencies between this document and the documents actually specifying the individual building blocks manageable. Another consequence is that the interactions shown in this document are not based on actual operations specified in the interfaces but rather on an architectural level as well.

Second, this document aims to provide guidance for the working groups in specifying the individual building blocks by defining patterns and concepts to solve common problems. This guidance should help to build a uniform and consistent standard from ground up.

6 Quality Requirements

Quality requirements define the expectations of AUTOSAR Adaptive Platform stakeholders for the quality and the attributes of the AUTOSAR Adaptive Platform standard that indicate whether the quality factors are satisfied or not. Section 6.1 starts by listing the quality attributes that, in the end, are used to assess whether the AUTOSAR Adaptive Platform and its software architecture satisfies the expected quality factors or not. Section 6.2 then provides quality scenarios that operationalize quality attributes and turn them into measurable quantities by describing the reaction of the system to a stimulus in a certain situation.

6.1 Quality Attributes

The AUTOSAR Adaptive Platform has many stakeholders with different concerns. Thus, this document uses the following three quality attribute categories that correspond to the three main stakeholder groups in order to make the requirements and their impact on the architecture more comprehensible:

- **AUTOSAR Adaptive Platform Standard:** Quality requirements for the AUTOSAR standard itself. These requirements may directly affect the architecture of the AUTOSAR Adaptive Platform.
- **AUTOSAR Adaptive Platform Stack:** Quality requirements for an implementation of the AUTOSAR standard as an AUTOSAR stack. These requirements may indirectly affect the architecture of the AUTOSAR Adaptive Platform.
- **AUTOSAR Adaptive Application:** Quality requirements for an application based on an AUTOSAR stack. These requirements may transitively affect the architecture of the AUTOSAR Adaptive Platform.

The quality attributes are organized according to the Architecture Tradeoff Analysis Method (ATAM) [6] as a tree, one for each of the quality attribute categories. The leafs of those trees are the individual quality attributes.

6.1.1 AUTOSAR Adaptive Platform Standard

- Functional suitability
  - The software architecture shall reflect the project objectives (POs) and be the consistent source for all specifications (here: completeness with respect to the PO; see also usability below).
  - The standard shall not contain elements that are not traceable to POs, change requests (CRs), or concepts.

  - The standard shall contain at least one element derived from each PO, CR, or concept.

- Performance efficiency
  - The specification shall allow for a run-time efficient implementation. Run-time efficiency refers to all resource consumption, CPU, RAM, ROM.
- Compatibility
  - The standard shall retain older versions of its elements in the face of change.
  - The standard shall be interoperable with pre-existing standards, especially the AUTOSAR Classic Platform. Pre-existing standards means network protocols and the like.
  - The standard shall adopt new versions of pre-existing standards only after an impact analysis.
- Usability
  - The use of the standard shall be as easy as possible for suppliers and application developers. Easy means: not much material and resources required.
  - The holistic approach shall not be broken (avoid different approaches in one standard).
  - The standard shall contain application sample code for all its elements.
  - The standard shall contain documentation of the use cases for its elements.
  - The standard shall document the semantics of its elements.
  - The standard shall document its decisions, consequences, and implementation restrictions (both for stack & apps) including their rationale.
  - The standards elements shall be easy to use and hard to misuse.
  - The standard shall stick to pre-existing standards, as far as no functional requirements are compromised.
  - The standard shall be as stable as possible.
  - AUTOSAR standards shall not change disruptive but rather evolve evolutionary (for example, backward-compatibility can be a help).
  - The software architecture shall reflect the PO and be the consistent source for all specifications (here: consistency; see also functional suitability above).
- Reliability
  - The standard shall classify its elements with respect to safety relevance (that is, functional clusters shall be marked if they participate in safety critical operations of the platform).

  - The standard shall specify control flow restrictions between its elements in order to achieve freedom from interference.
  - The standard shall contain use case driven argumentation for safety scenarios that can be used to build a safety case. (This should help the stack implementers in getting a certification, if they follow the standard.)
- Security
  - The standard shall specify data flow restrictions between its elements, and between applications.
  - The standard shall classify its elements with respect to security sensitivity (that is, functional clusters shall be marked if they handle sensitive data.)
  - The standard shall contain use case driven argumentation for security scenarios that can be used to build a security case. (This should help the stack implementers in getting a certification, if they follow the standard.)
- Maintainability
  - It shall be possible in an efficient way to maintain AUTOSAR Adaptive Platform without preventing the introduction of new technologies (efficient in terms of effort on the modification of the standard).
  - The impact set of a change shall be available.
  - The standard shall be structured in a way that minimizes change impact.
  - It shall be possible to drop/deprecate elements of the standard.
  - It shall be easy to add new features/needs without breaking the maintainability or the need to redesign the software architecture. Easy means quick, with low effort, local changes only and no heavy side effects.
  - The maturity of parts of the standard shall be visible.
  - The process shall enforce an architectural impact analysis in a very early stage of the change process.
  - The process shall enforce minimizing changes, that is not adding similar functionality multiple times.
- Portability
  - Applications shall be portable between different stack implementations and different machines.
  - It shall be possible to scale the software architecture to the given project needs.

6.1.2 AUTOSAR Adaptive Platform Stack

- Compatibility
  - An AUTOSAR Adaptive Platform stack implementation shall be capable to offer multiple versions of the same service.
  - An instance of an AUTOSAR Adaptive Platform stack implementation shall be able to co-exist with other instances on different machines, within the same vehicle.
- Usability
  - An AUTOSAR Adaptive Platform stack implementation shall explicitly document restrictions on the application development that go beyond the standard.
- Maintainability
  - An AUTOSAR Adaptive Platform stack implementation shall be traceable to the contents of the standard.
  - An AUTOSAR Adaptive Platform stack implementation shall support multiple versions of the same service.
- Portability
  - An AUTOSAR Adaptive Platform stack shall be portable to a different custom hardware.
  - An AUTOSAR Adaptive Platform stack shall provide mechanisms to replace parts.

6.1.3 AUTOSAR Adaptive Application

- Usability
  - *No Goal:* An application developer shall be able to supply custom implementation for pre-defined platform functionality.
- Maintainability
  - An application shall explicitly state which parts of the standard it uses.
- Portability
  - An application entirely based on AUTOSAR Adaptive Platform (i.e. without custom extensions) shall be portable to another AUTOSAR Adaptive Platform stack of the same version without modifications to the application source code itself (source code compatibility).

6.2 Quality Scenarios

There are currently no quality scenarios defined for the AUTOSAR Adaptive Platform.

7 System Scope and Context

This chapter provides an overview of the AUTOSAR Adaptive Platform system context by separating the AUTOSAR Adaptive Platform and its communication partners (e.g., external systems). Considering Figure 7.1, there are three categories of communication partners for the AUTOSAR Adaptive Platform.

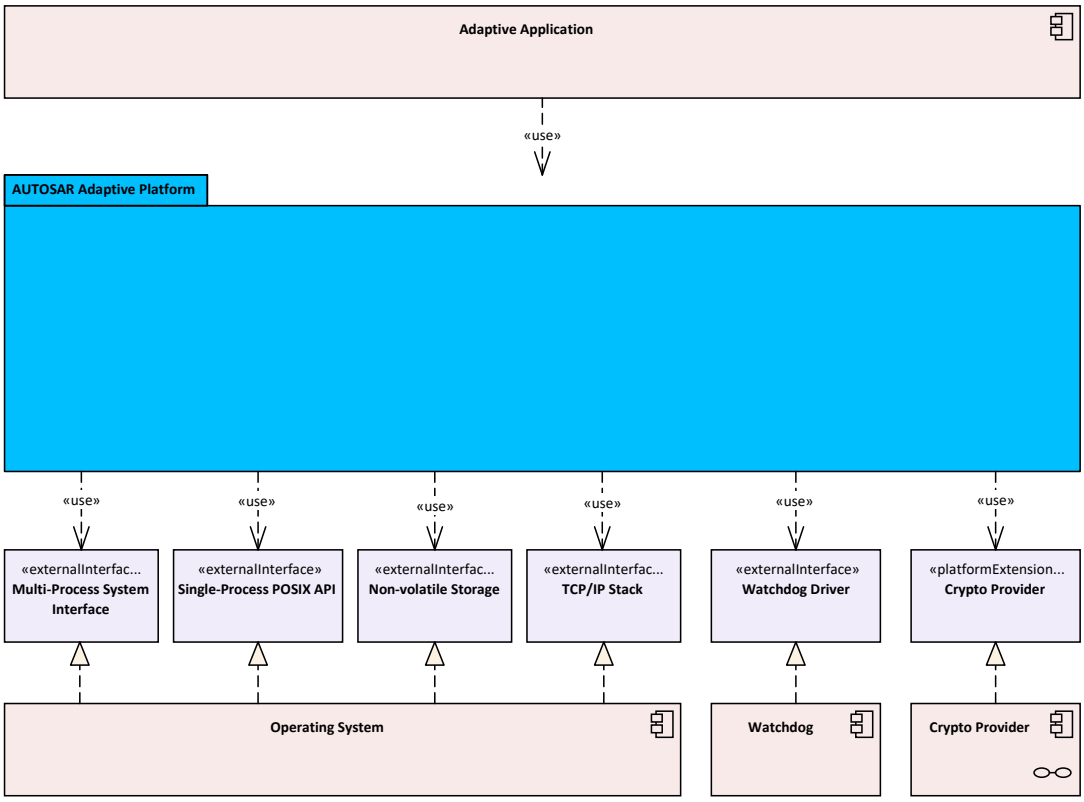

Figure 7.1: Overview of AUTOSAR Adaptive Platform and its context

The AUTOSAR Adaptive Platform is conceptually a middleware. AUTOSAR Adaptive Platform provides services to `Adaptive Application`s (cf. Section 7.1) beyond those available from the underlying operating system, drivers, and extensions (cf. Section 7.2). Section 7.3 describes the third category that are external systems communicating with (an `Adaptive Application` via) AUTOSAR Adaptive Platform.

7.1 Adaptive Application

`Adaptive Application`s are built on the functionality provided by the AUTOSAR Adaptive Platform. They directly use the various interfaces provided by the individual building blocks of AUTOSAR Adaptive Platform described in more detail in chapter 9.

7.2 Dependencies

7.2.1 Crypto Provider

`Crypto Provider` is a component that provides implementations of cryptographic routines and hash functions to the AUTOSAR Adaptive Platform.

7.2.2 Operating System

The `Operating System` is the main component that AUTOSAR Adaptive Platform uses to provide its services. The `Operating System` controls processes and threads, and provides inter-process communication facilities. The `Operating System` also provides access to network interfaces, protocols like `TCP/IP`, and access to non-volatile storage.

7.2.3 Watchdog

The `Watchdog` is a component to control the hardware watchdog of the machine an AUTOSAR Adaptive Platform runs on.

7.3 External Systems

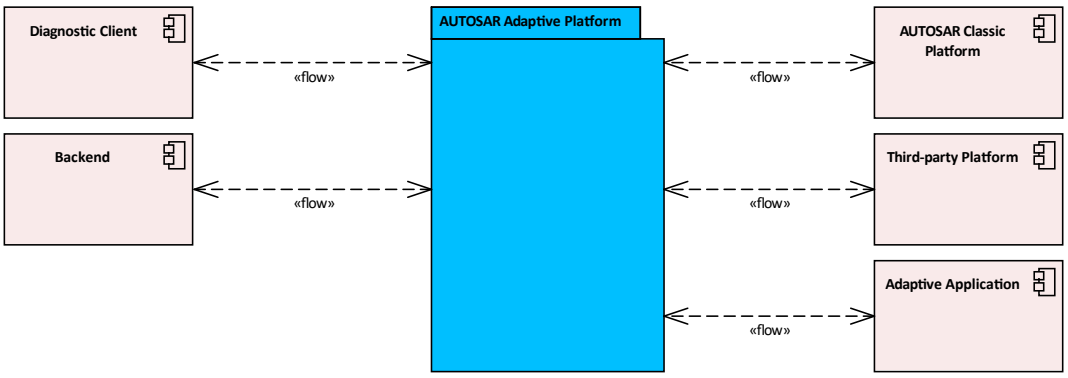

Figure 7.2: Overview of the AUTOSAR Adaptive Platform and external systems

The AUTOSAR Adaptive Platform supports applications that are operated in heterogeneous environments. This section lists the external systems that AUTOSAR Adaptive Platform is intended to interface.

7.3.1 AUTOSAR Adaptive Application

There may be many Adaptive Applications deployed in a vehicle on different Machines. An Adaptive Application that does not run on the current instance of the AUTOSAR Adaptive Platform is therefore an external system to the AUTOSAR Adaptive Platform. Such Adaptive Applications may exchange data such as sensor or status information. During a software update of the entire vehicle, the update of the individual AUTOSAR Adaptive Platforms could be coordinated by a central Adaptive Application that makes use of the UCM Master addon to UCM.

7.3.2 AUTOSAR Classic Platform

The AUTOSAR Classic Platform is the main platform for deeply-embedded applications such as sensor/actor systems. Adaptive Applications may require access for example to sensor data provided by an AUTOSAR Classic Platform ECU and vice versa.

7.3.3 Third-party Platform

Besides the both platforms (AUTOSAR Adaptive Platform and AUTOSAR Classic Platform) provided by AUTOSAR, there may be ECUs in a vehicle and other systems that are built on different platforms that need to communicate with an Adaptive Application via AUTOSAR Adaptive Platform.

7.3.4 Diagnostic Client

A Diagnostic Client uses the diagnostic services provided by the AUTOSAR Adaptive Platform.

7.3.5 Backend

A Backend system provides Software Packages for download and controls the update process via Update and Configuration Management.

8 Solution Strategy

The AUTOSAR Adaptive Platform is a standard for an automotive middleware. It is not a concrete implementation. The AUTOSAR Adaptive Platform standard leaves a certain degree of freedom to its implementers by defining requirements and software specifications that need to be fulfilled without specifying how.

8.1 Architectural Approach

To support the complex applications, while allowing maximum flexibility and scalability in processing distribution and compute resource allocations, AUTOSAR Adaptive Platform follows the concept of a service-oriented architecture (SOA). In a service-oriented architecture a system consists of a set of services, in which one may use another in turn, and applications that use one or more of the services depending on its needs. Often service-oriented architectures exhibit system-of-system characteristics, which AUTOSAR Adaptive Platform also has. A service, for instance, may reside on a local ECU that an application also runs, or it can be on a remote ECU, which is also running another instance of AP. The application code is the same in both cases - the communication infrastructure will take care of the difference providing transparent communication. Another way to look at this architecture is that of distributed computing, communicating over some form of message passing. At large, all these represent the same concept. This message passing, communication-based architecture can also benefit from the rise of fast and high-bandwidth communication such as Ethernet.

8.2 Decomposition Strategy

The building blocks of the AUTOSAR Adaptive Platform architecture are refined step-by-step in this document according to the model depicted in figure 8.1. The top-level categories are chosen to give an overview from a users perspective what kind of functionality the AUTOSAR Adaptive Platform provides. A category contains one or more Functional Clusters. The Functional Clusters of the AUTOSAR Adaptive Platform are defined to group a specific coherent technical functionality. Functional Clusters themselves specify a set of interfaces and components to provide and realize that technical functionality. The building block view also contains information of the Functional Clusters interdependencies based on interfaces from other Functional Clusters they use. However, note that these interdependencies are recommendations rather than strict specifications because they would constrain implementations.

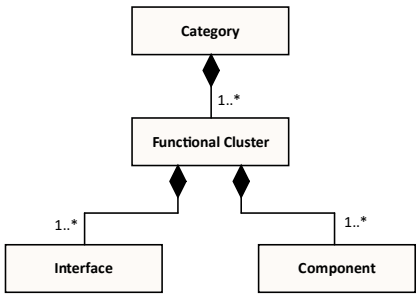

Figure 8.1: Type model of building blocks

8.3 Technology

8.3.1 Implementation Language

C++ is the programming language of choice for the AUTOSAR Adaptive Platform and `Adaptive Application`s. C++ was chosen due to its safer programming model (compared to C) and availability of certified compilers that produce highly optimized machine code. Such properties are especially important for safety- and performance-critical, real-time applications (such as typical `Adaptive Application`s) where C++ has become more and more popular in the software industry and academics.

8.3.2 Parallel Processing

Although the design for AUTOSAR Adaptive Platform as a service-oriented architecture inherently leverages parallel processing, the advancement of (heterogeneous) many-core processors offers additional opportunities. The AUTOSAR Adaptive Platform is designed to scale its functionality and performance as (heterogeneous) many-core technologies advance. Hardware and platform interface specifications are one part of that equation. However, advancements in operating system and hypervisor technologies as well as development tools (for example automatic parallelization) are also crucial and are to be fulfilled by AUTOSAR Adaptive Platform providers, the software industry, and academics.

8.4 Design Principles

The architecture of the AUTOSAR Adaptive Platform is based on several design principles that are outlined below.

8.4.1 Leveraging existing standards

AUTOSAR Adaptive Platform aims to leverage existing standards and specifications wherever possible. For example, AUTOSAR Adaptive Platform ist built on the existing and open C++ standard (cf. Section 8.3.1) to facilitate a faster development of the AUTOSAR Adaptive Platform itself and benefiting from the eco-systems of such standards. It is, therefore, a critical focus in developing the AUTOSAR Adaptive Platform specification not to casually introduce a new replacement functionality that an existing standard already offers. For instance, no new interfaces are casually introduced just because an existing standard provides the functionality required but the interface is superficially hard to understand.

8.4.2 SOLID principles

The SOLID principles [7] are a central part of the design principles of AUTOSAR. While these five principles are all valid, only the Single-responsibility Principle, the Interface Segregation Principle and the Dependency Inversion Principle are relevant on the abstraction level of this document. Therefore, they are elaborated in the following.

8.4.2.1 Single-responsibility Principle

The single-responsibility principle (`SRP, SWEBOK3`) [7] states that a component or class should be responsible for a single part of the overall functionality provided by the software. That responsibility should be encapsulated by the component or class. The services provided by the component or class (via its interface(s)) should be closely aligned with its responsibility.

The single-responsibility principle minimizes the reasons (i.e. a change to the single responsibility) that require a change to its interface. Thus, it minimizes impact on clients of such an interface and leads to a more maintainable architecture (or code).

8.4.2.2 Interface Segregation Principle

The interface segregation principle (`ISP`) [7], [8] states that clients should not be forced to depend on methods that they don't use. As a consequence of the interface segregation principle, interfaces should be split up to reflect different roles of clients.

Similar to the single-responsibility principle, the segregation of interfaces reduce the impact of a change to an interface to the clients and suppliers of an segregated interface.

8.4.2.3 Dependency Inversion Principle

The dependency inversion principle (DIP) [7], [8] states that high-level building blocks should not depend on low-level building blocks. Both should depend on abstractions (e.g. interfaces). Furthermore, the dependency inversion principle states that abstractions (e.g. interfaces) should not depend on details. Details (e.g. a concrete implementation) should depend on abstractions.

The dependency inversion principle results in a decoupling of the implementations of building blocks. This is important to scale implementation efforts (cf. Section 5.2) and to perform proper integration tests.

8.4.3 Acyclic Dependencies Principle

The acyclic dependencies principle (ADP) [7], [8] states that dependencies between building blocks should form a directed acyclic graph.

The acyclic dependencies principle helps to identify participating building blocks and reason about error propagation and freedom from interference. In general, it also reduces the extend of building blocks to consider during activities such as test, build and deployment.

8.5 Deployment

The AUTOSAR Adaptive Platform supports the incremental deployment of applications, where resources and communications are managed dynamically to reduce the effort for software development and integration, enabling short iteration cycles. Incremental deployment also supports explorative software development phases. For product delivery, the AUTOSAR Adaptive Platform allows the system integrator to carefully limit dynamic behavior to reduce the risk of unwanted or adverse effects allowing safety qualification. Dynamic behavior of an application will be limited by constraints stated in the Execution Manifest (cf. Section 12.8), for example, dynamic allocation of resources and communication paths are only possible in defined ways, within configured ranges. Implementations of an AUTOSAR Adaptive Platform may further remove dynamic capabilities from the software configuration for production use. Examples of reduced behavioral dynamics might be:

- Pre-determination of the service discovery process
- Restriction of dynamic memory allocation to the startup phase only
- Fair scheduling policy in addition to priority-based scheduling
- Fixed allocation of processes to CPU cores
- Access to pre-existing files in the file-system only

- Constraints for AUTOSAR Adaptive Platform API usage by applications
- Execution of authenticated code only

8.6 Verification and Validation

The AUTOSAR Adaptive Platform standard uses a dedicated implementation of the standard (AUTOSAR Adaptive Platform Demonstrator) to validate the requirements and to verify the (still abstract) software design imposed by the individual software specifications.

9 Building Block View

This chapter provides an overview of the static structure of the AUTOSAR Adaptive Platform by describing the high-level building blocks and their inter-dependencies. Please note that the use of interfaces between `Functional Cluster`s in the AUTOSAR Adaptive Platform is currently not standardized. Some aspects, for example, access management, are also not yet fully incorporated and standardized in all `Functional Cluster`s.

9.1 Overview

Figure 9.1 provides an overview of the different categories of building blocks available in the AUTOSAR Adaptive Platform. The categories are explained in more detail in the subsequent sections.

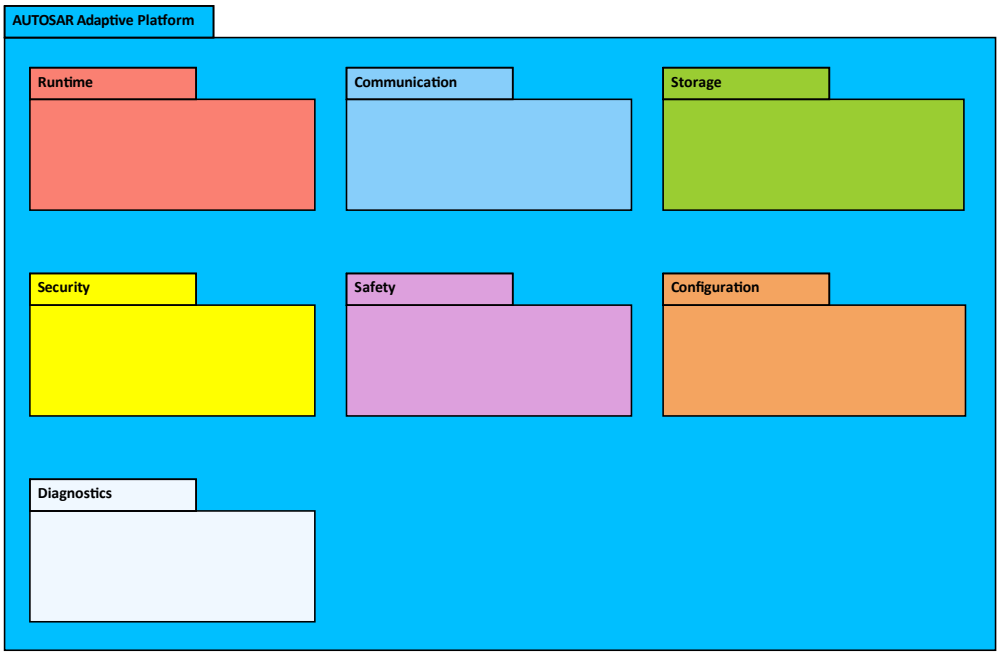

Figure 9.1: Overview of AUTOSAR Adaptive Platform and its building blocks

9.1.1 Stereotypes

The UML diagrams presented in this chapter use a UML profile to provide a more precise semantics of the elements and relationships. Table 9.1 provides an overview of the stereotypes in that profile and their semantics.

Stereotype	Metaclass	Semantics
applicationInterface	Interface	An interface that is intended to be used by Adaptive Applications directly. Components within the AUTOSAR Adaptive Platform may use such interfaces as well.
externalInterface	Interface	An interface that is provided by an external component.
internalInterface	Interface	An interface that is intended to be used only by components within the AUTOSAR Adaptive Platform itself.
platformExtension-Interface	Interface	An interface that is used to extend the functionality of the AUTOSAR Adaptive Platform. Such interfaces are not intended to be used by Adaptive Applications directly.
restrictedApplication-Interface	Interface	An applicationInterface that is restricted to be used by specific application components only. This applies in particular to interfaces used by State Management. State Management is considered to be a part of an Adaptive Application. However, State Management access to interfaces, for example, provided by Execution Management, that are not intended to be used by any other part of an Adaptive Application.

Table 9.1: Overview of Stereotypes

9.2 Runtime

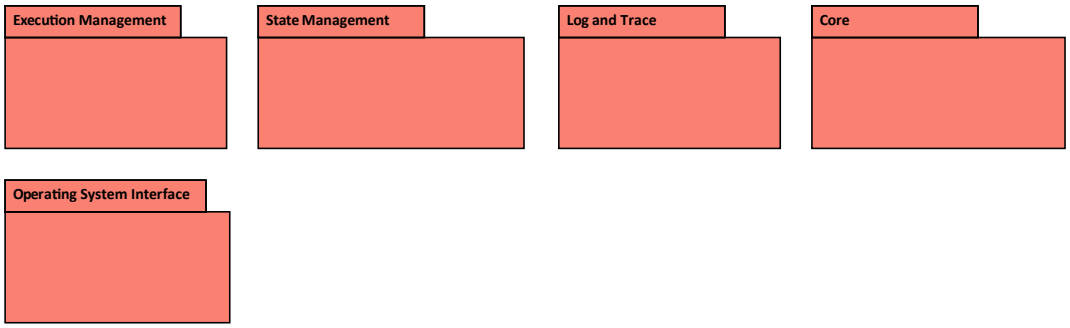

Figure 9.2: Overview of Runtime and its building blocks

9.2.1 Execution Management

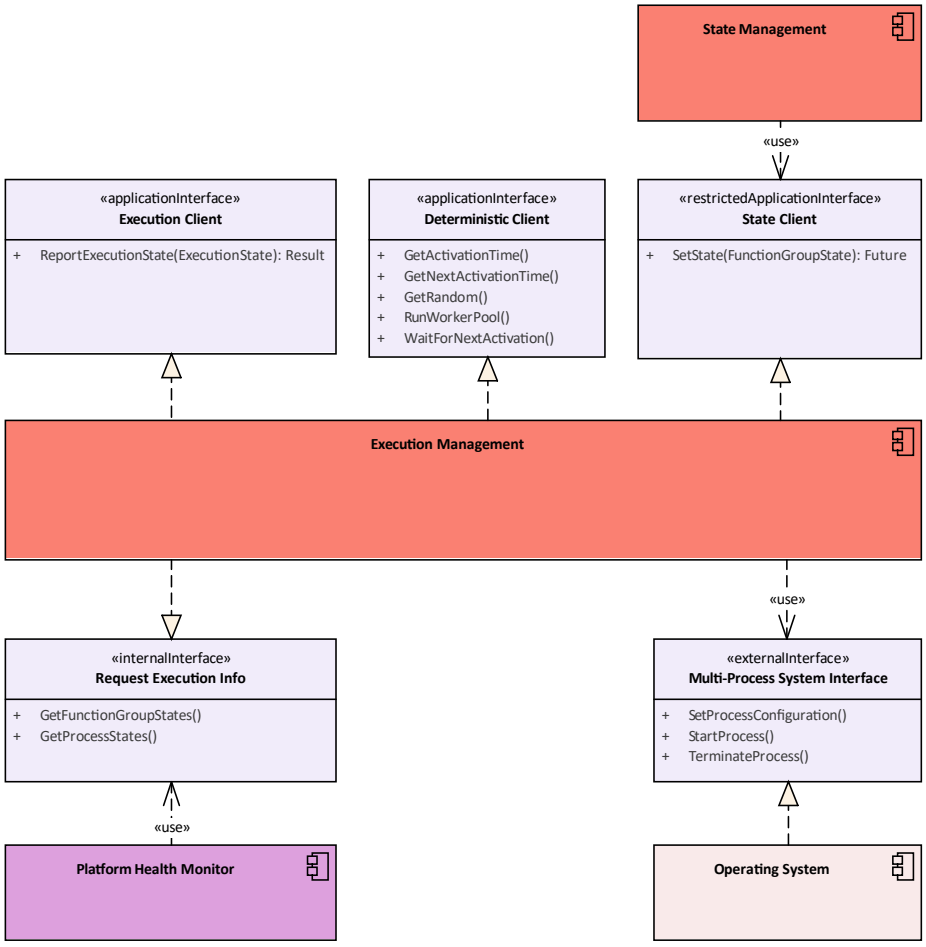

Figure 9.3: Overview of `Execution Management`

9.2.1.1 Responsibilities

`Execution Management` is responsible to control `Process`es of the AUTOSAR Adaptive Platform and `Adaptive Application`s. That is, it starts, configures, and stops `Process`es as configured in `Function Group State`s using interfaces of the `Operating System`. The `Operating System` is responsible for runtime scheduling of those `Process`. The configuration of `Process`es that `Execution Management` performs includes limiting their resource consumption (CPU time, memory) using `Resource Group`s provided by the `Operating System`.

`Execution Management` is the entry point of AUTOSAR Adaptive Platform and is started by the `Operating System` during system boot. `Execution Management` then controls the startup and shutdown of the AUTOSAR Adaptive Platform (see Section 10.2.1 for details). `Execution Management` optionally supports authenticated

startup where it maintains the chain of trust when starting from a Trust Anchor. During authenticated startup Execution Management validates the authenticity and integrity of Processes and will prevent their execution if violations are detected. Through these mechanisms, a trusted platform can be established (cf. Section 12.12).

9.2.1.2 Provided Interfaces

Deterministic Client (applicationInterface)

The Deterministic Client interface provides the functionality to run a cyclic deterministic execution.

Deterministic execution provides a mechanism such that a calculation using a given input data set always produces a consistent output within a bounded time. There is a distinction between time and data determinism. Time determinism states that the output is always produced by a fixed deadline. Data determinism refers to generating always the same output from the same input data set and internal state. In the AUTOSAR Adaptive Platform, time determinism has to be handled by the provisioning of sufficient resources. The support for data determinism is provided by Execution Management through the Deterministic Client interface.

If a software lockstep is used, Execution Management interacts with the software lockstep framework to ensure identical behavior of the redundantly executed Processes.

Execution Management also interacts with Communication Management to synchronize data handling with cycle activation.

Execution Client (applicationInterface)

The Execution Client interface provides functionality for a Process to report its execution state to Execution Management.

State Client (restrictedApplicationInterface)

The State Client interface provides functionality to request entering a Function Group State. This interface is intended to be used by State Management only.

State Management determines the desired state of the Function Groups that run on an AUTOSAR Adaptive Platform and requests corresponding state transitions via the State Client interface. Execution Management will start/stop Processes to reflect the configuration of the Function Group State made by the integrator and report the result back to State Management.

Request Execution Info (internalInterface)

The `Request Execution Info` interface provides functionality to retrieve information about `Process`es and `Function Group State`s.

9.2.1.3 Required Interfaces

Multi-Process System Interface

`Execution Management` should use this interface for starting, controlling, and stopping `Process`es via the `Operating System`.

9.2.2 State Management

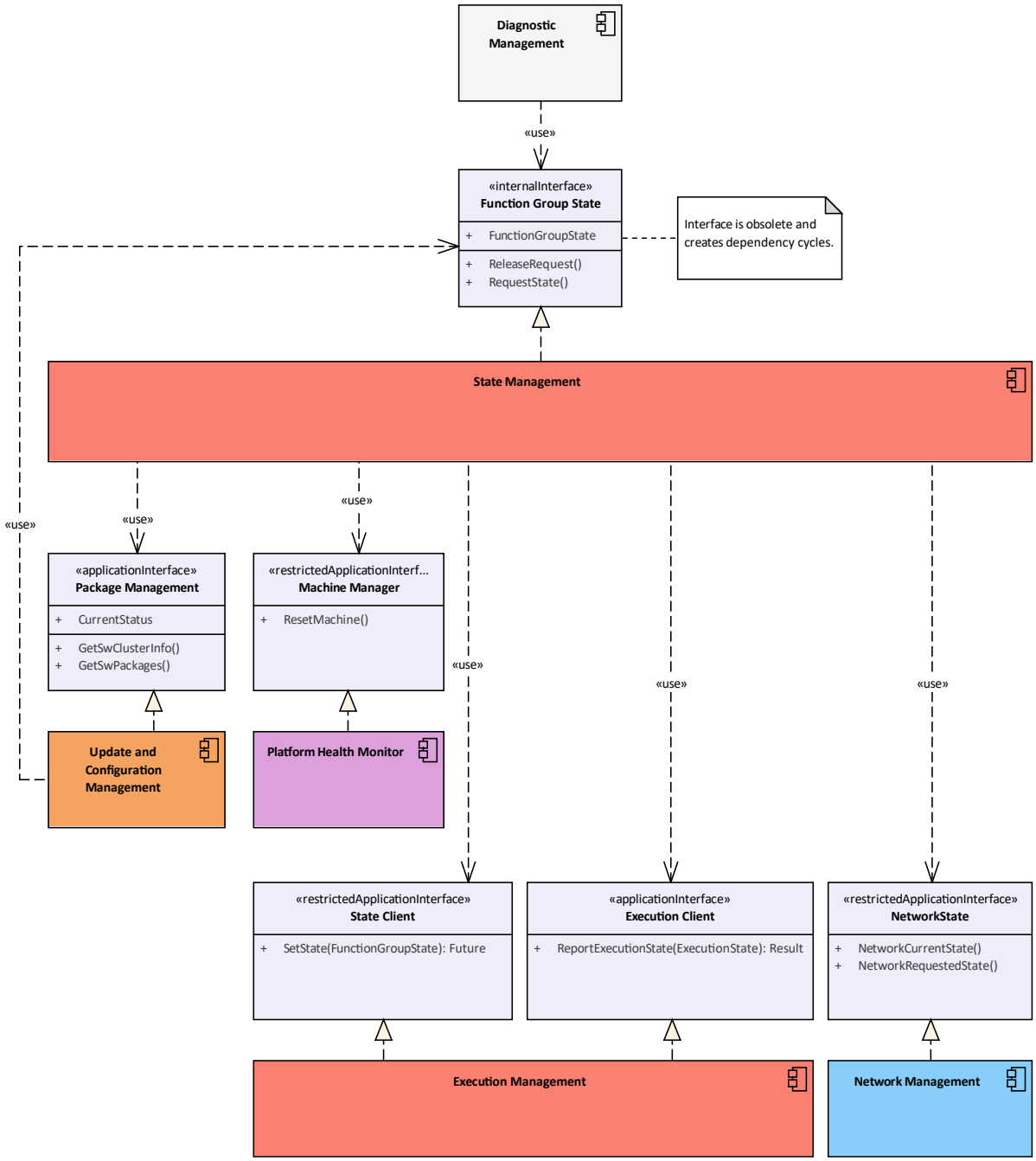

Figure 9.4: Overview of `State Management`

9.2.2.1 Responsibilities

`State Management` determines the desired target state of the AUTOSAR Runtime for Adaptive Applications based on various application-specific inputs. That target state is the set of active `Function Group State`s of all `Function Group`s running on

the AUTOSAR Runtime for Adaptive Applications. State Management delegates to Execution Management to switch the individual Function Groups to the respective Function Group States.

State Management is a unique component in the AUTOSAR Adaptive Platform because it is not part of a AUTOSAR Adaptive Platform stack. The logic of State Management currently needs to be implemented as application-specific code and then configured and integrated with an AUTOSAR Adaptive Platform stack.

9.2.2.2 Provided Interfaces

Function Group State (internalInterface)

The (obsolete) Function Group State interface provides functionality to request a switch to a Function Group State. It is still included in the current release due to compatibility reasons with Update and Configuration Management.

9.2.2.3 Required Interfaces

Execution Client

State Management should use the Execution Client interface to report its own execution state back to Execution Management.

State Client

State Management should use the State Client interface to request transitions to Function Group States.

Network State

State Management should use the Network State interface to trigger activation and deactivation of (partial) networks and to subscribe for corresponding activation and deactivation events.

Package Management

State Management should use the Package Management interface to subscribe for status changes of Update and Configuration Management. State Management shall use this information to prevent shutdown of the AUTOSAR Runtime for

Adaptive Applications while an update session is running and to reload/restart relevant parts of the application and AUTOSAR Runtime for Adaptive Applications after an update has been applied.

9.2.3 Log and Trace

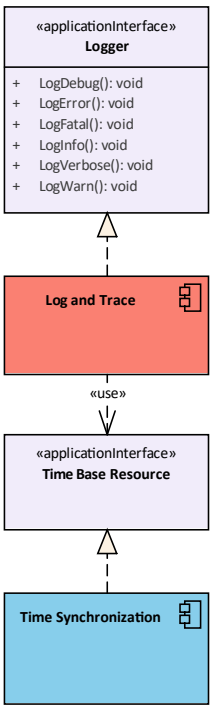

Figure 9.5: Overview of `Log and Trace`

9.2.3.1 Responsibilities

`Log and Trace` provides functionality to build and log messages of different severity. An `Adaptive Application` can be configured to forward log messages to various sinks, for example to a network, a serial bus, the console, and to non-volatile storage.

9.2.3.2 Provided Interfaces

Logger (applicationInterface)

The `Logger` interface provides functionality to log textual messages.

9.2.3.3 Required Interfaces

Time Base Resource

The `Time Base Resource` interface should be used to determine timestamps for log messages.

9.2.4 Core

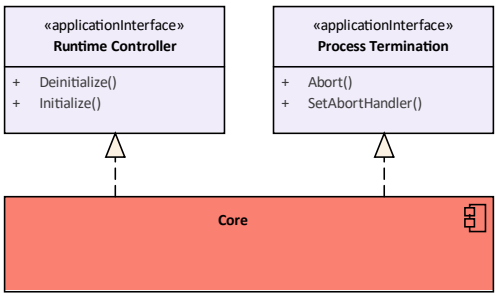

Figure 9.6: Overview of `Core`

9.2.4.1 Responsibilities

`Core` provides functionality for initialization and de-initialization of the AUTOSAR Runtime for Adaptive Applications as well as termination of `Processes`.

9.2.4.2 Provided Interfaces

Runtime Controller (applicationInterface)

The `Runtime Controller` interface provides functionality for initialization and de-initialization of the AUTOSAR Runtime for Adaptive Applications.

Process Termination (applicationInterface)

The `Process Termination` interface provides functionality to terminate the current `Process`.

9.2.4.3 Required Interfaces

`Application Initializer` does not require any standardized interfaces.

9.2.5 Operating System Interface

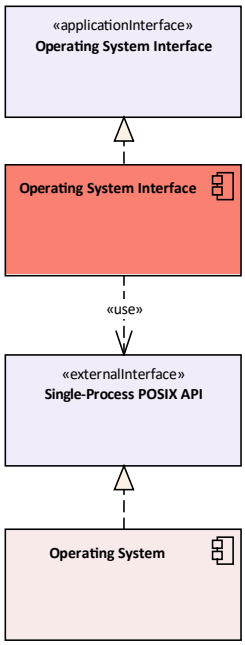

Figure 9.7: Overview of `Operating System Interface`

9.2.5.1 Responsibilities

The `Operating System Interface` provides functionality for implementing multi-threaded real-time embedded applications and corresponds to the [9, POSIX PSE51 profile]. That profile provides support to create `Thread`s that may be executed in parallel on modern multi-core processors and to control their properties such as stack memory or their scheduling. In addition, primitives for shared resource access are provided such as `Semaphore`s or memory locking. Asynchronous (real-time) signals and message passing enable inter-process communication. High resolution timers and clocks are provided to control real-time behavior precisely. Some input/output functions are provided as well but no file system APIs.

POSIX PSE51 and the `Operating System Interface` do not provide any means to execute and control `Process`es. `Process`es (of the AUTOSAR Adaptive Platform) are entirely controlled by `Execution Management` via non-standardized interfaces.

Note that a typical AUTOSAR Adaptive Platform stack will not provide an actual implementation of the `Operating System Interface` because all functionality is already provided by standard libraries of the programming language (e.g. Standard C++ Library).

9.2.5.2 Provided Interfaces

Operating System Interface (applicationInterface)

The `Operating System Interface` is used by `Adaptive Application`s to create, configure and control multiple `Thread`s with support for real-time guarantees.

9.2.5.3 Required Interfaces

Single-Process POSIX API

The `Single-Process POSIX API` is used to delegate creation, configuration and control of `Thread`s to the `Operating System`.

9.3 Communication

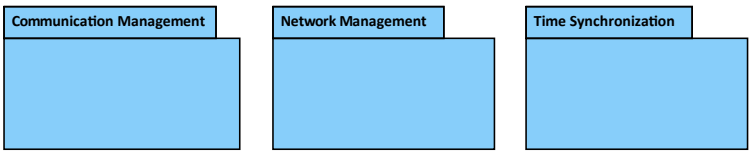

Figure 9.8: Overview of Communication and its building blocks

9.3.1 Communication Management

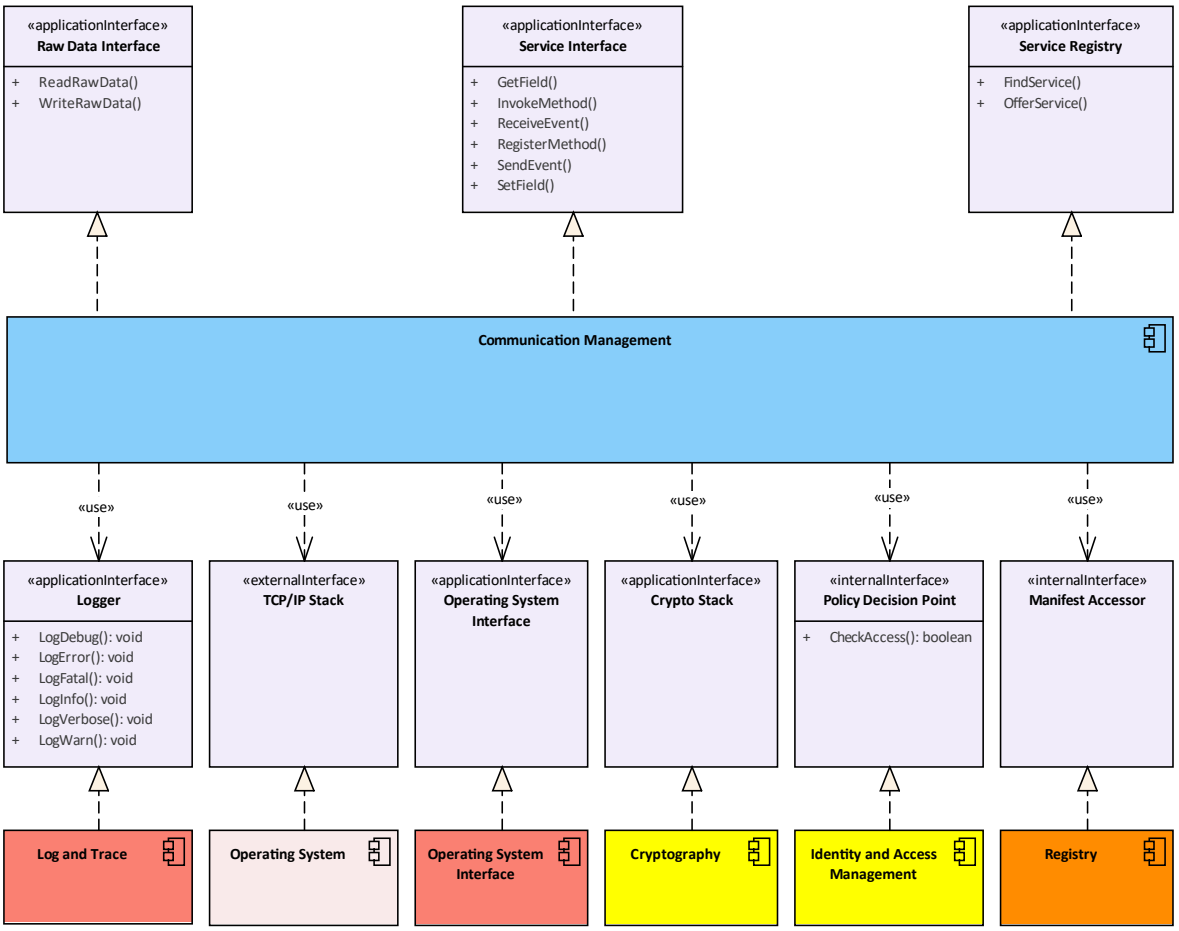

Figure 9.9: Overview of `Communication Management`

9.3.1.1 Responsibilities

`Communication Management` is responsible for all levels of service-oriented and raw communication between applications in a distributed real-time embedded environment. That is, intra-process communication, inter-process communication and inter-machine communication. The latter is also possible with AUTOSAR Classic Platforms and third-party platforms. Communication paths can be established at design-, start-up-, and run-time. `Communication Management` consists of a generic part that handles brokering and configuration as well as (potentially generated) skeletons for service providers and respective proxies for service consumers.

9.3.1.2 Provided Interfaces

Service Interface (applicationInterface)

The `Service Interface` is used for service-oriented communication.

A service consists of a combination of `Event`s, `Field`s, and `Method`s. The `Service Interface` supports both synchronous callback-based communication (e.g., service method calls) and asynchronous communication (e.g., field changes, events). Extensions are provided for secure communication and quality of service.

Raw Data Interface (applicationInterface)

The `Raw Data Interface` provides functionality to send and receive streams of raw data.

Service Registry (applicationInterface)

The `Service Registry` provides functionality to register and to discover services during runtime.

9.3.1.3 Required Interfaces

Logger

The `Logger` interface should be used to log for example failed checks.

TCP/IP Stack

The `TCP/IP Stack` interface should be used to control network connections for inter-machine communication.

Operating System Interface

The `Operating System Interface` interface should be used to control connections for intra- and inter-process communication.

Crypto Application Interface

The `Crypto Application Interface` should be used for end-to-end protection (integrity, authenticity, confidentiality) of communication channels.

Manifest Accessor

The `Manifest Accessor` should be used to read service configuration from the `Manifest`s.

Policy Decision Point

The `Policy Decision Point` interface should be used to check access.

9.3.2 Network Management

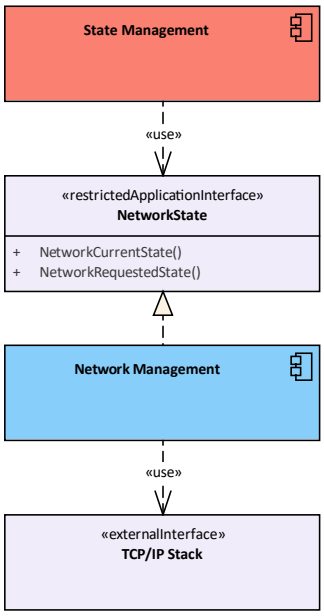

Figure 9.10: Overview of `Network Management`

9.3.2.1 Responsibilities

`Network Management` provides functionality to request and query the network states for logical network handles that can be mapped to physical or partial networks.

9.3.2.2 Provided Interfaces

Network State (restrictedApplicationInterface)

The `Network State` interface is used to request and query the network states for logical network handles. It is intended to be used by `State Management` only.

9.3.2.3 Required Interfaces

TCP/IP Stack

`Network Management` should use the functionality of the underlying `TCP/IP Stack` to send or receive `Network Management` messages on the physical networks.

9.3.3 Time Synchronization

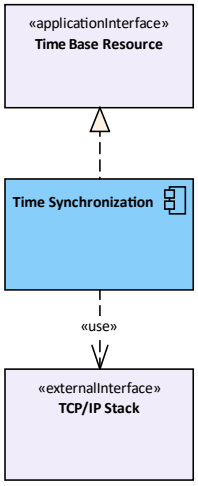

Figure 9.11: Overview of `TimeSynchronization`

9.3.3.1 Responsibilities

`Time Synchronization` provides synchronized time information in distributed applications. Synchronized time information between different applications and/or `Machine`s is of paramount importance when the correlation of different events across a distributed system is needed, either to be able to track such events in time or to trigger them at an accurate point in time.

9.3.3.2 Provided Interfaces

Time Base Resource (applicationInterface)

The `Time Base Resource` interface is used retrieve and update time information.

9.3.3.3 Required Interfaces

TCP/IP Stack

The `TCP/IP Stack` should be used to perform synchronization via network.

9.4 Storage

Figure 9.12: Overview of Storage and its building blocks

9.4.1 Persistency

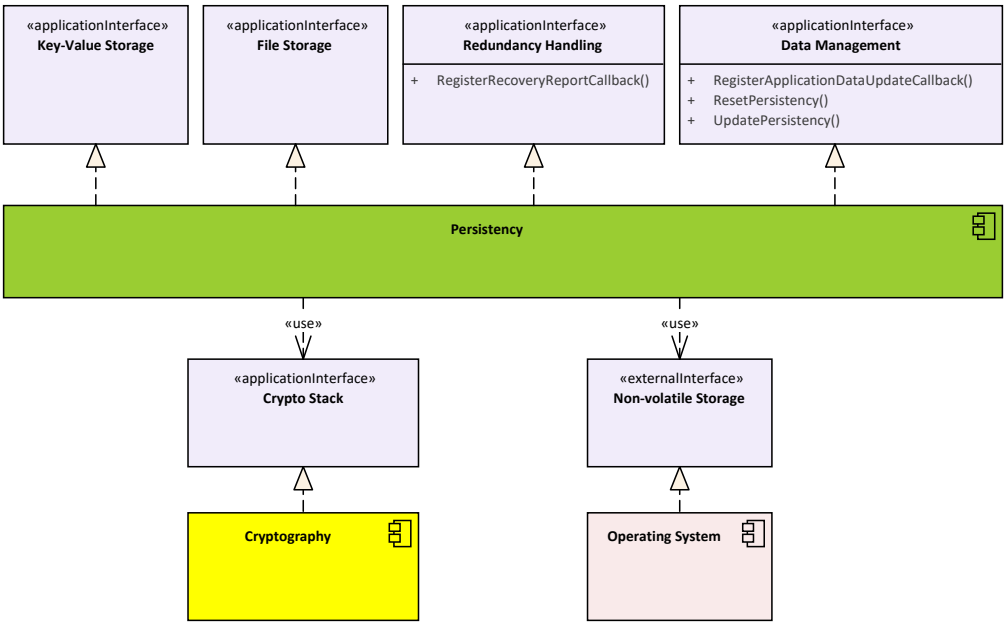

Figure 9.13: Overview of `Persistency`

9.4.1.1 Responsibilities

`Persistency` provides functionality to store and retrieve information to/from non-volatile storage of a `Machine`.

Persistent data is always private to one `Process` and is persisted across boot and ignition cycles. There is no mechanism available to share data between different `Processes` using `Persistency` to prevent a second path of data exchange besides `Communication Management`. However, `Persistency` supports concurrent access from multiple threads of the same application running in the context of the same `Process`.

`Persistency` offers integrity of the stored data and provides error detection and correction schemes. `Persistency` also offers confidentiality of the stored data using encryption.

`Persistency` offers statistics, for example, the number of used resources.

9.4.1.2 Provided Interfaces

File Storage (applicationInterface)

The `File Storage` interface provides read and write access to plain files that may be used to store arbitrary data.

Key-Value Storage (applicationInterface)

The `Key-Value Storage` interface provides read and write access to data structured as key-value pairs. It supports strings as keys and all primitive types supported by AUTOSAR Adaptive Platform as values. Besides the plain types, `Persistency` shall store serialized binary data which are given by `ara::core::Vector` of `ara::core::Byte` as well as arbitrary `CppImplementationDataTypes`.

Redundancy Handling (applicationInterface)

As `Persistency` supports redundant storage of data and files, an error in the stored data can be fixed implicitly by using the redundantly stored data. Only if this fails, an error will occur. The `Redundancy Handling` interface provides a way to track whether storage errors have been fixed using the available redundancy.

Data Management (applicationInterface)

The `Data Management` interface provides functionality to trigger data migration (e.g., after an software update) and data reset.

9.4.1.3 Required Interfaces

Crypto Stack

The `Crypto Stack` interface should be used to ensure integrity and confidentiality of the stored data.

Non-volatile Storage

The `Non-volatile Storage` interface should be used to access non-volatile storage.

9.5 Security

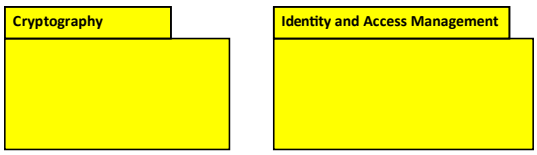

Figure 9.14: Overview of Security and its building blocks

9.5.1 Cryptography

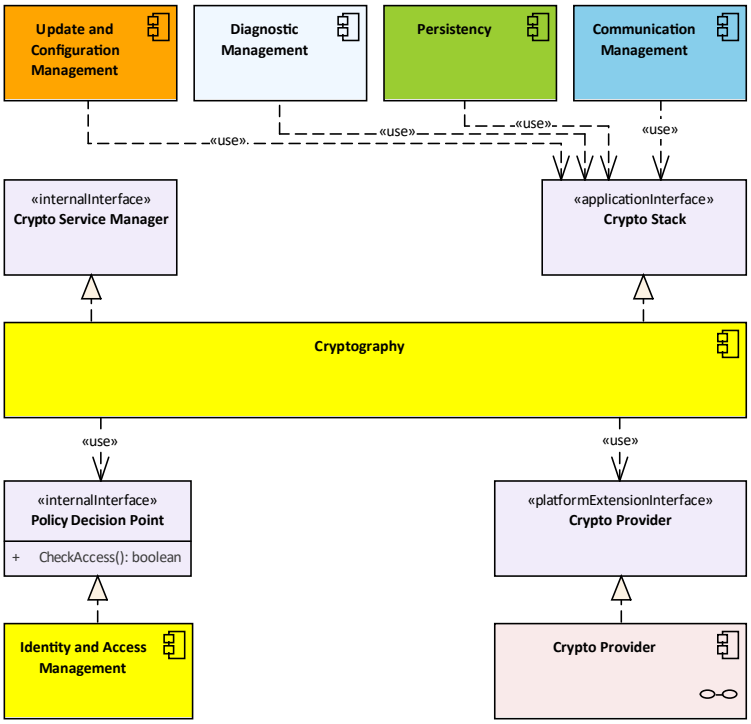

Figure 9.15: Overview of `Cryptography`

9.5.1.1 Responsibilities

`Cryptography` provides various cryptographic routines to ensure confidentiality of data, to ensure integrity of data (e.g., using hashes), and auxiliary functions for example key management and random number generation. `Cryptography` is designed to support encapsulation of security-sensitive operations and decisions in a separate component, such as a Hardware Security Module (HSM). Additional protection of keys and key usage can be provided by constraining keys to particular usages (e.g., decrypt-only), or limiting the availability of keys to individual applications as reported by `Identity and Access Management`.

Depending on application support, `Cryptography` can also be used to protect session keys and intermediate secrets when processing cryptographic protocols such as TLS and SecOC.

9.5.1.2 Provided Interfaces

Crypto Stack (applicationInterface)

The `Crypto Stack` provides cryptographic routines and auxiliary functions to `Adaptive Application`s.

Crypto Service Manager (internalInterface)

The `Crypto Service Manager` provides internal functionality for access management and certificate storage.

9.5.1.3 Required Interfaces

Crypto Provider

`Cryptography` should use the `Crypto Provider` interface to access the actual implementation of cryptographic routines and auxiliary functions provided by external libraries or hardware drivers.

Policy Decision Point

`Cryptography` should use the `Policy Decision Point` interface to make access control decisions, for example on keys.

9.5.2 Identity and Access Management

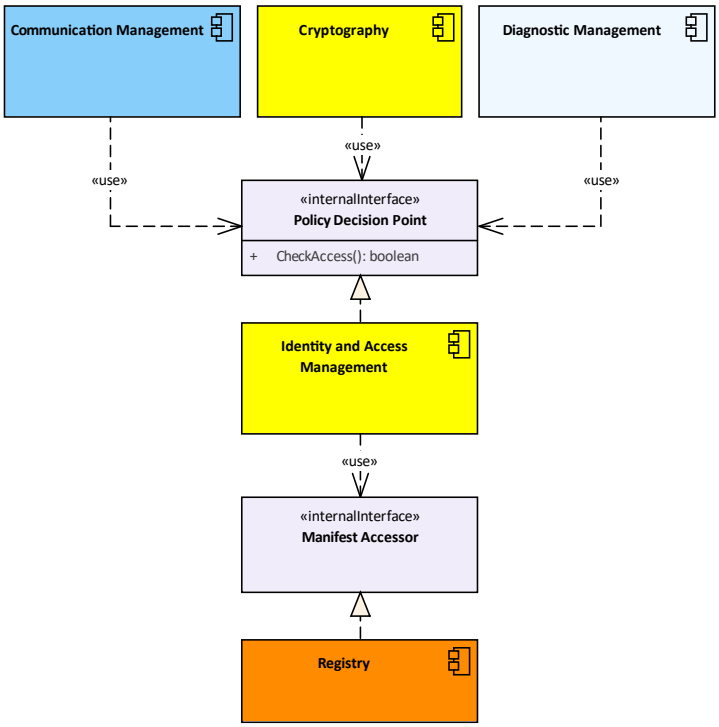

Figure 9.16: Overview of `Identity and Access Management`

9.5.2.1 Responsibilities

`Identity and Access Management` checks access to resources of the AUTOSAR Adaptive Platform, for example, on `Service Interfaces` and `Functional Clusters`. `Identity and Access Management` hereby introduces access control for `Adaptive Applications` and protection against privilege escalation in case of attacks. In addition, `Identity and Access Management` enables integrators to verify access on resources requested by `Adaptive Applications` in advance during deployment.

9.5.2.2 Provided Interface

Policy Decision Point (internalInterface)

The `Policy Decision Point` interface provides functionality to make an access control decision.

9.5.2.3 Required Interfaces

Manifest Accessor

Intents of `Adaptive Application`s are stored in the application manifest and need to be extracted by `Identity and Access Management`.

9.6 Safety

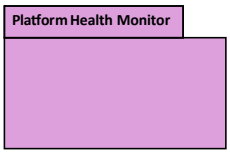

Figure 9.17: Overview of Safety and its building blocks

9.6.1 Platform Health Management

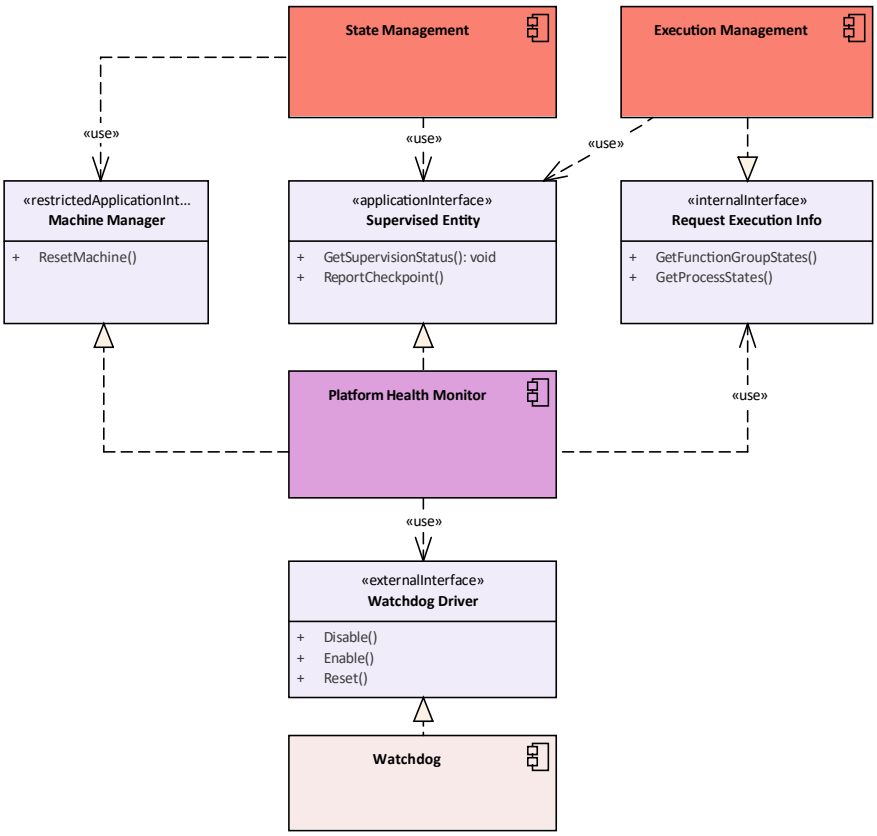

Figure 9.18: Overview of `Platform Health Management`

9.6.1.1 Responsibilities

Platform Health Management performs (aliveness, logical, and deadline) supervision of Processes in safety-critical setups and reports failures to State Management. Platform Health Management also controls the Watchdog that in turn supervises the Platform Health Management.

Alive Supervision checks that a supervised entity is not running too frequently and not too rarely. Deadline Supervision checks that steps in a supervised entity are executed within the configured minimum and maximum time. Logical Supervision checks that the control flow during execution matches the designed control flow. All types of supervision can be used independently and are performed based on reporting of Checkpoints by the supervised entity.

State Management and Execution Management are the fundamental Functional Clusters of the AUTOSAR Adaptive Platform and need to run and work properly in any case. Therefore, Platform Health Management shall always supervise the corresponding Processes for State Management and Execution Management. Supervision failures in these Processes shall be recovered by a reset of the Machine since the normal way of error recovery (via Execution Management and State Management) is no longer reliable.

9.6.1.2 Provided Interfaces

Machine Manager (restrictedApplicationInterface)

The Machine Manager interface provides functionality to trigger a reset of the Machine.

Supervised Entity (applicationInterface)

The Supervised Entity interface provides functionality to report Checkpoints to Platform Health Management, for example that a certain milestone in the control flow has been reached.

9.6.1.3 Required Interfaces

Watchdog Driver

Platform Health Management shall use the Watchdog Driver to control the Watchdog.

Request Execution Info

`Platform Health Management` should use the `Request Execution Info` to retrieve information about active `Process`es and `Function Group State`s and enable / disable its supervisions accordingly.

9.7 Configuration

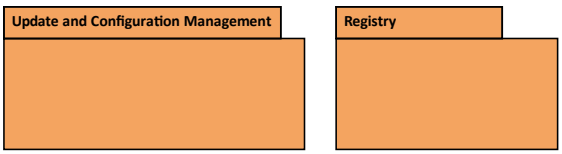

Figure 9.19: Overview of Configuration and its building blocks

9.7.1 Update and Configuration Management

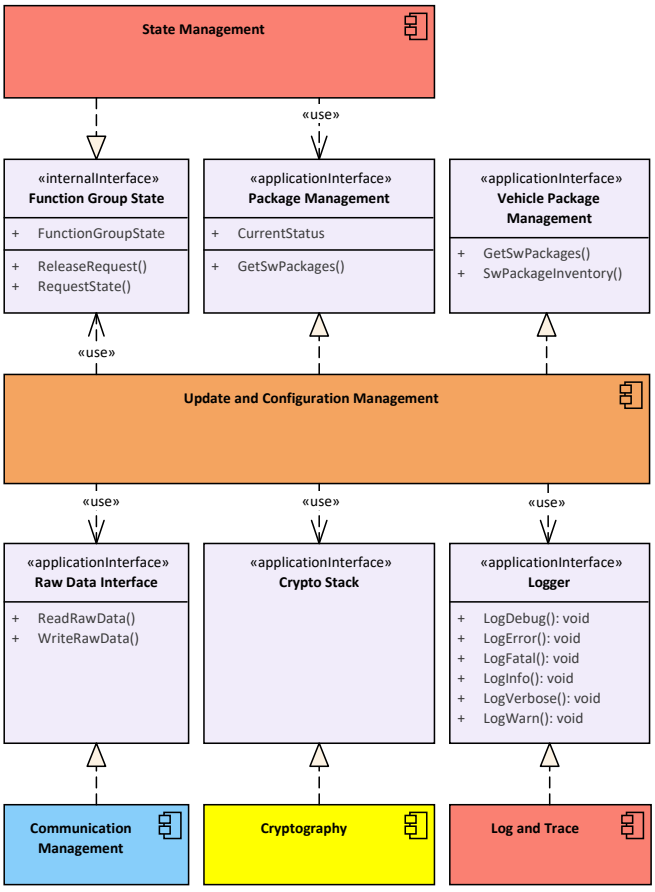

Figure 9.20: Overview of `Update and Configuration Management`

9.7.1.1 Responsibilities

`Update and Configuration Management` is responsible for updating, installing, removing and keeping a record of the software on an AUTOSAR Adaptive Platform in a safe and secure way. Hereby, `Update and Configuration Management` enables to update the software and its configuration flexibly through over-the-air updates (OTA).

9.7.1.2 Provided Interfaces

Package Management (applicationInterface)

The `Package Management` interface provides provides functionality to download, process, activate and remove `Software Packages`. Additionally, `Package Management` provides the status of ongoing updates to `State Management` in order to prevent switching into unsafe states during updates and shutdowns of the system during the update process.

Vehicle Package Management (applicationInterface)

The `Vehicle Package Management` interface provides functionality methods to download, process, activate and remove `Vehicle Packages`.

9.7.1.3 Required Interfaces

Raw Data Interface

`Update and Configuration Management` should use the `Raw Data Interface` for raw data transfers, for example transfer of `Software Packages`.

Crypto Stack

`Update and Configuration Management` should check the integrity and authenticity of software packages using functionality of the `Crypto Stack`.

Logger

`Update and Configuration Management` should use the `Logger` interface to write log messages.

Function Group State

`Update and Configuration Management` should use the `Function Group State` interface to bring the AUTOSAR Runtime for Adaptive Applications into a state that is safe before applying a software update.

9.7.2 Registry

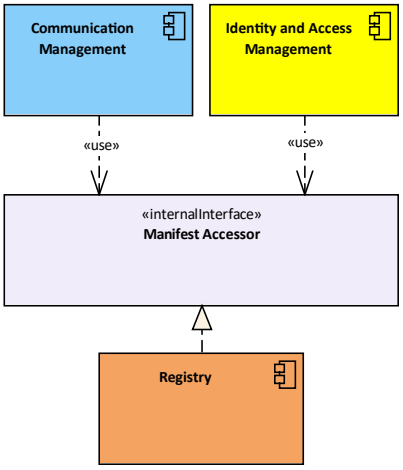

Figure 9.21: Overview of `Registry`

9.7.2.1 Responsibilities

The `Registry` is an internal component of the AUTOSAR Adaptive Platform that provides access the information stored in `Manifest`s. It is not intended to be used by `Adaptive Application`s directly.

9.7.2.2 Provided Interfaces

Manifest Accessor (internalInterface)

The `Manifest Accessor` interface is used to access data stored in `Manifest`s, for example configuration settings.

9.7.2.3 Required Interfaces

This component does not require any standardized interfaces.

9.8 Diagnostics

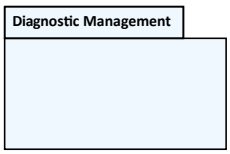

Figure 9.22: Overview of Diagnostics and its building blocks

9.8.1 Diagnostic Management

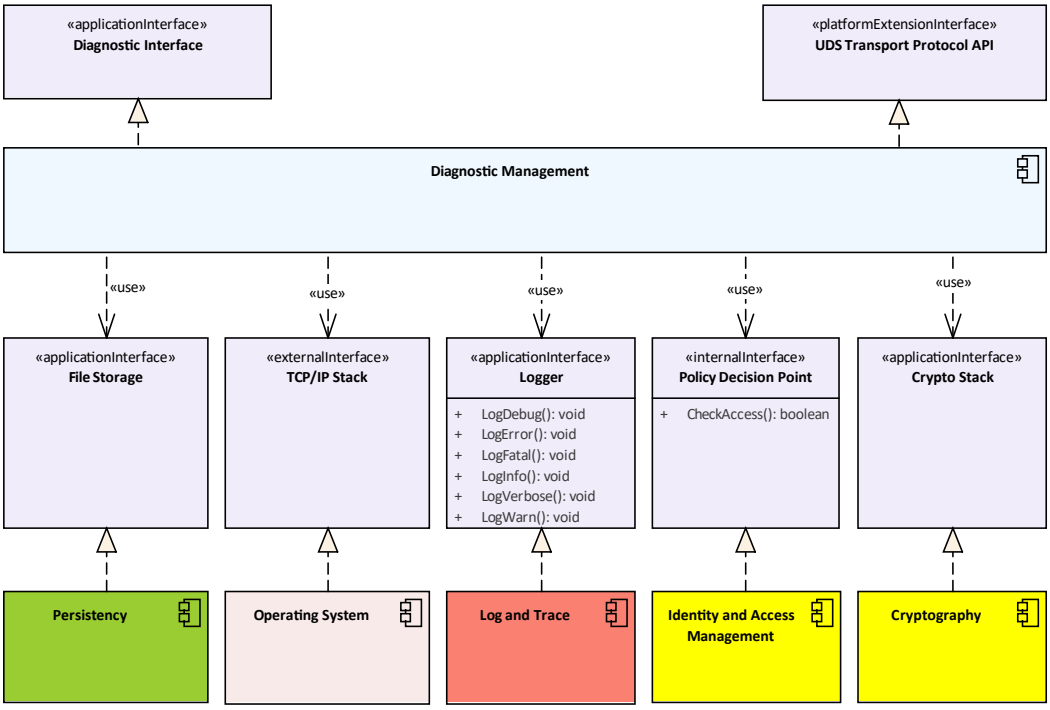

Figure 9.23: Overview of `Diagnostic Management`

9.8.1.1 Responsibilities

`Diagnostic Management` is responsible for handling diagnostic events produced by the individual `Process`es running in an AUTOSAR Runtime for Adaptive Applications. `Diagnostic Management` stores such events and the associated data persistently according to rendition policies. `Diagnostic Management` also provides access to diagnostic data for external `Diagnostic Client`s via standardized network protocols (ISO 14229-5 (UDSonIP) which is based on the ISO 14229-1 (UDS) and ISO 13400-2 (DoIP)).

9.8.1.2 Provided Interfaces

Diagnostic Interface (applicationInterface)

The `Diagnostic Interface` provides functionality to create diagnostic events and store them persistently.

UDS Transport Protocol API (platformExtensionInterface)

The `UDS Transport Protocol API` provides functionality to extend the AUTOSAR Adaptive Platform with UDS transport layer implementations, for example with an OEM specific implementation.

9.8.1.3 Required Interfaces

File Storage

`DM` should use the `File Storage` interface to store diagnostic data persistently.

TCP/IP Stack

The `TCP/IP Stack` is used to accept and control network connections from external `Diagnostic Client`s using the UDSonIP protocol.

Logger

`DM` should use the `Logger` interface to log errors and diagnostic events.

Policy Decision Point

`DM` should use the `Policy Decision Point` interface to check access of `Diagnostic Client`s.

Crypto Stack

`DM` should use the `Crypto Stack` interface for example to perform authentication of diagnostic sessions.

10 Runtime View

This chapter shows the original design approach of the AUTOSAR Adaptive Platform for implementing selected use cases. The presented use cases currently cover just a small part of the functionality of the AUTOSAR Adaptive Platform. More use cases will be added in future versions of this document. Please note that individual implementations of the AUTOSAR Adaptive Platform may always choose a different design internally. Thus, interaction partners, the type of messages, and their order may differ.

10.1 Overview

The use cases are categorized in the subsequent sections. Section 10.2 groups the use cases that control the lifecycle of an AUTOSAR Runtime for Adaptive Applications. Section 10.3 lists use cases for communication with external systems. Section 10.4 demonstrates how Adaptive Applications can be installed and how they and the AUTOSAR Runtime for Adaptive Applications can be updated.

10.2 AUTOSAR Runtime for Adaptive Applications Lifecycle

10.2.1 Machine Startup

During the startup of a machine the `Operating System` performs initialization steps in an implementation-specific way. These steps include starting any middleware related to the `Operating System`, including device-drivers and services handling low-level middleware. In addition, `Execution Management` is started as the entry point of the AUTOSAR Runtime for Adaptive Applications. `Execution Management` then controls the startup of the AUTOSAR Runtime for Adaptive Applications by starting `State Management` and `Platform Health Management` `Process`es.

After `State Management` and `Platform Health Management` are started, `State Management` takes control over the initialization of the AUTOSAR Adaptive Platform by requesting a transition to the standardized `Machine State Startup` from `Execution Management`. After the rest of the AUTOSAR Adaptive Platform has been initialized, `State Management` requests application-specific states for the other `Function Group`s on the `Machine` from `Execution Management` in the same way. `Platform Health Management` always supervises the `Processes` of `Execution Management` and `State Management` with a (probably fixed) implementation-specific set of rules. `Platform Health Management` itself is supervised by the `Watchdog`. In addition, `Platform Health Management` supervises application `Process`es according to the configuration in the `Machine Manifest`.

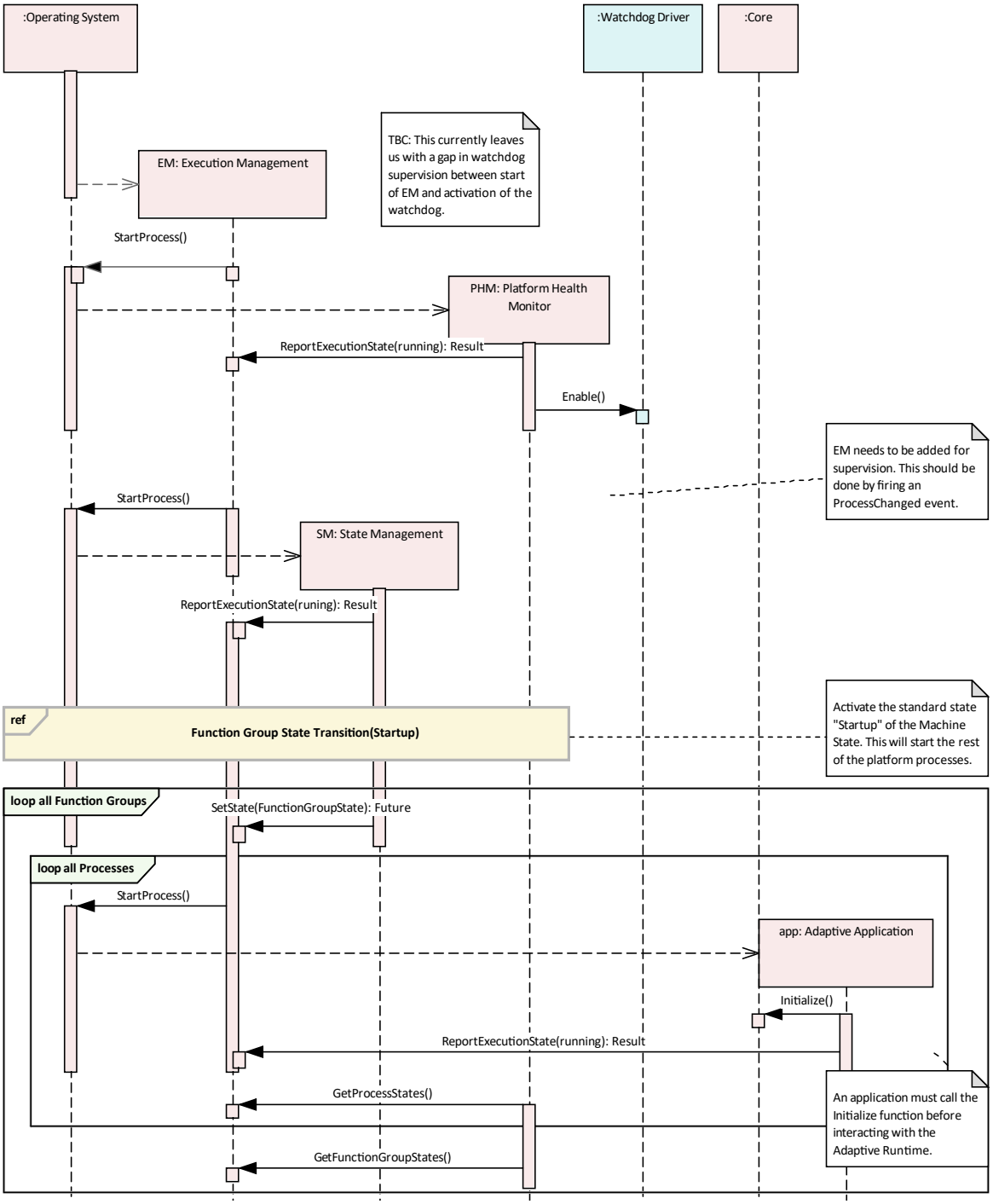

Figure 10.1: Startup of the AUTOSAR Runtime for Adaptive Applications

10.2.2 Machine Shutdown

A shutdown is requested by `State Management` after an application-specific event. First, the application `Function Group`s may be brought to a corresponding state (not

shown). Afterwards, State Management triggers a transition to the Shutdown Machine State. The shutdown procedure is controlled by Execution Management. Execution Management stops all platform Processes. Then, Execution Management terminates the State Management and Platform Health Management Processes and requests a shutdown of the underlying operating system.

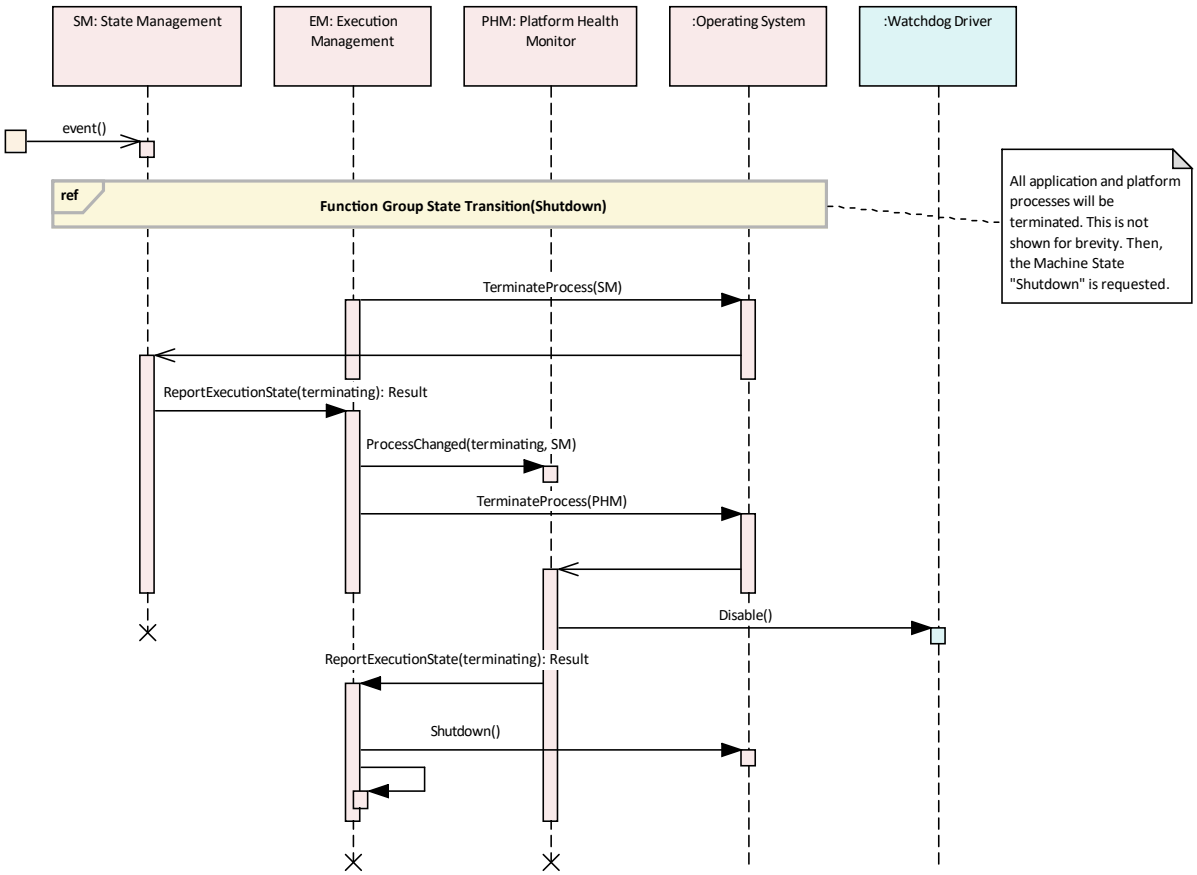

Figure 10.2: Shutdown of the AUTOSAR Runtime for Adaptive Applications

10.2.3 Function Group State Transition

A switch to another Function Group State is requested by State Management based on its inputs and internal state. The transition to the new Function Group State is controlled by Execution Management. First, all Processes are terminated that are either not active in the target state, or do have a different Startup Configuration in the target state. The latter may also include different startup dependencies. Then, all Processes are started by Execution Management in the order imposed by their dependencies. During the state transition, Execution Management notifies Platform Health Management on any change of the state of the Processes. Platform Health Management adapts its supervisions accordingly.

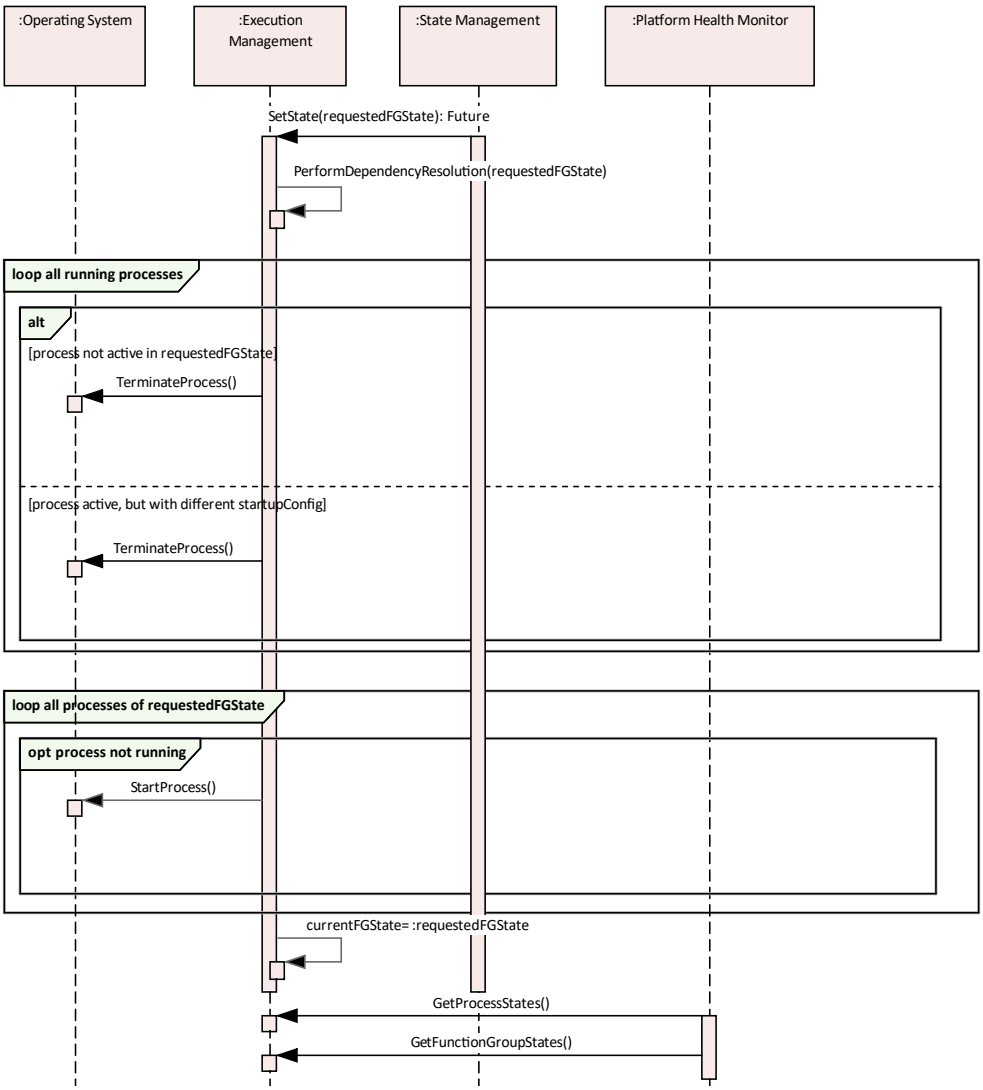

Figure 10.3: Transition to another `Function Group State`

10.2.4 Failure Recovery

In case `Platform Health Management` detects a failure in an entity it supervises it informs `Execution Management` about the supervision failure. `Execution Management` maps the supervised `Process` to the corresponding `Function Group` and delegates to `State Management` to handle that failure in the `Function Group` and perform recovery actions. `State Management` is an application-specific component that, depending on its various inputs, internal state etc., may decide upon actions to be taken to recover from a failure. There are two main possibilities:

- recover by switching `Function Group State` to another `Function Group State` (e.g., for degradation)
- recover by re-entering the same `Function Group State` and essentially restarting all `Process`es in the `Function Group`

- as a last resort, advise `Platform Health Management` to reset the `Machine`

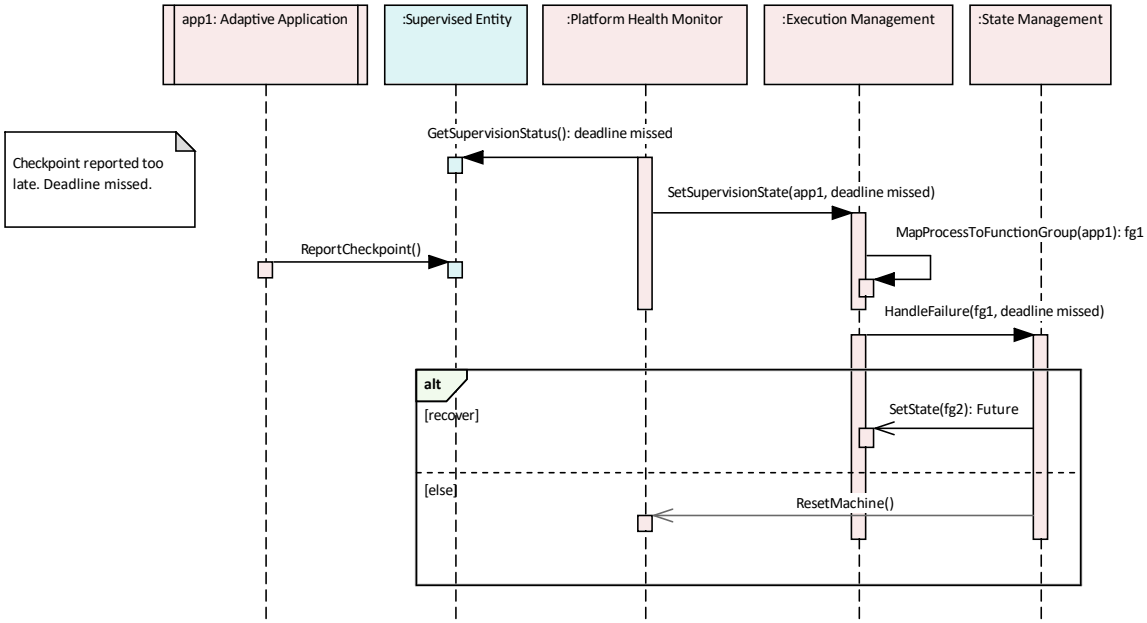

Figure 10.4: Failure recovery scenarios

10.3 Communication

Service-oriented communication in the AUTOSAR Adaptive Platform is guarded by `Identity and Access Management` that provides access control. All service-requests are handled by `Communication Management`. `Communication Management` determines the `Adaptive Application Identity` (AAID) of the sender `Process` using `Execution Management`. Then, `Communication Management` requests an access control decision from `Identity and Access Management` using the identity of the sender and information about the called service. `Communication Management` enforces the access control decision by forwarding the request to the service in case the access was granted or dropping the request in case the access was denied.

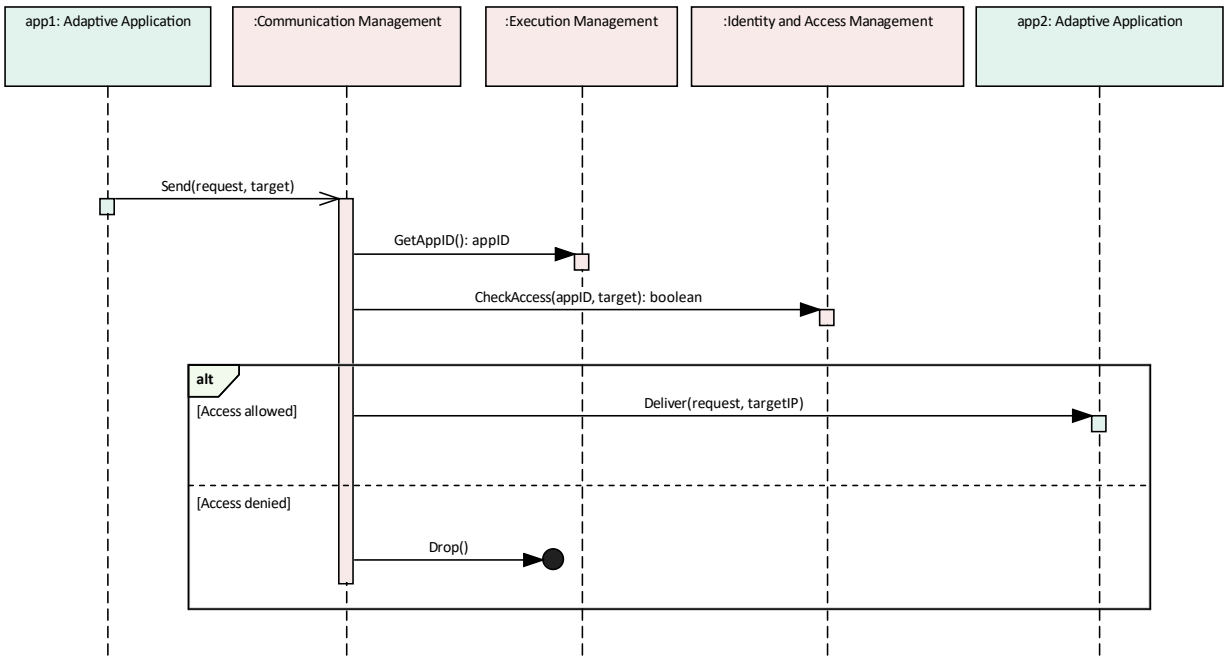

Figure 10.5: Access control in service-oriented communication

10.4 Update and Configuration Management

10.4.1 Update of an Adaptive Application

When an `Adaptive Application` has to be updated, the new version of the application first has to be transferred to the `Update and Configuration Management` instance. The state of the application as well as `Update and Configuration Management` are monitored by `State Management`. Only if there is nothing else running (state `kIdle`), the actual update process starts.

`Update and Configuration Management` then receives the signal to start processing the transferred data. After successful processing, the update enters activation phase and `Update and Configuration Management` checks dependencies in the transferred `Software Package`s. After successful activation, the `Adaptive Application` is restarted and after successful restart, running the updated software.

The last step for `Update and Configuration Management` is to finish the update process by cleaning up and deleting temporary data, old software version or stored data which are no longer required for the execution of the `Adaptive Application`.

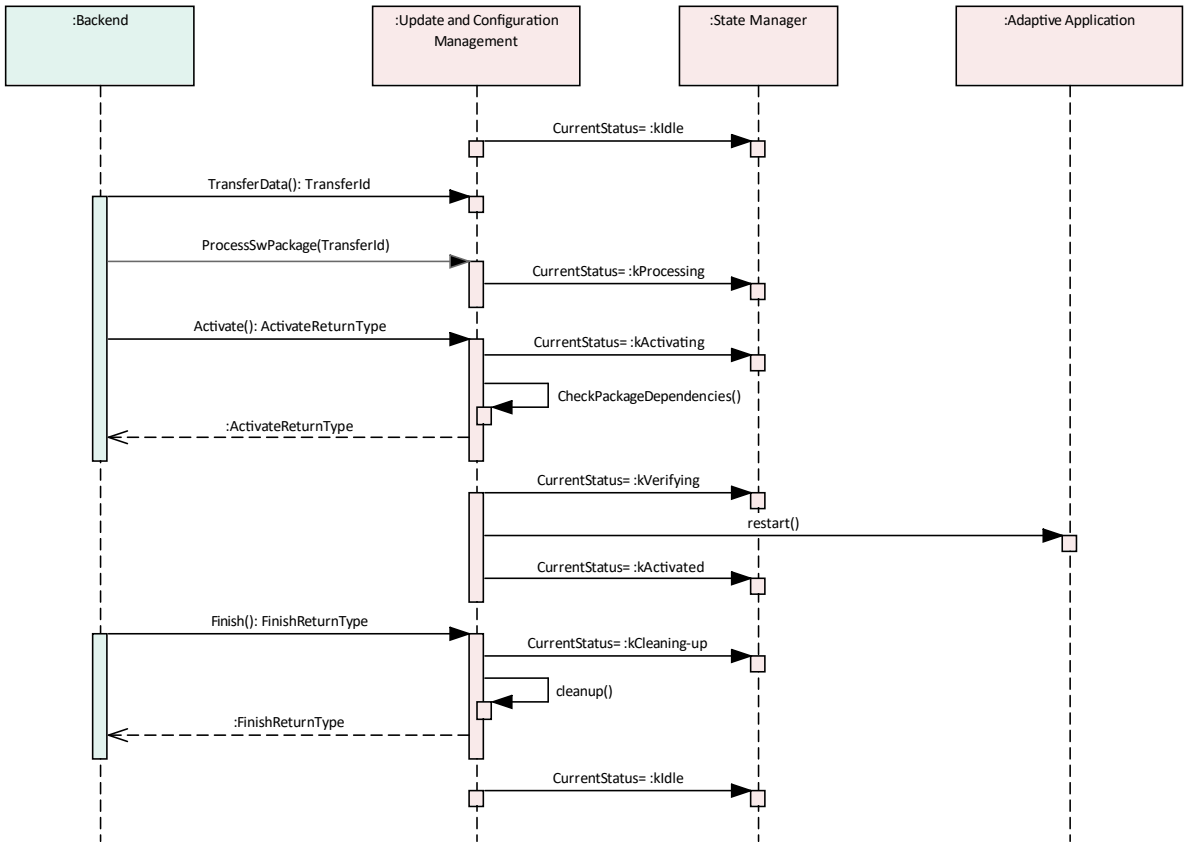

Figure 10.6: Successful update of an Adaptive Application

11 Deployment View

This chapter provides an overview of exemplary deployment scenarios for an AUTOSAR Adaptive Platform. Since the AUTOSAR Adaptive Platform is highly configurable in its deployment, this section rather provides constraints on supported deployments and a selection of relevant deployment scenarios.

11.1 Vehicle Software Deployment

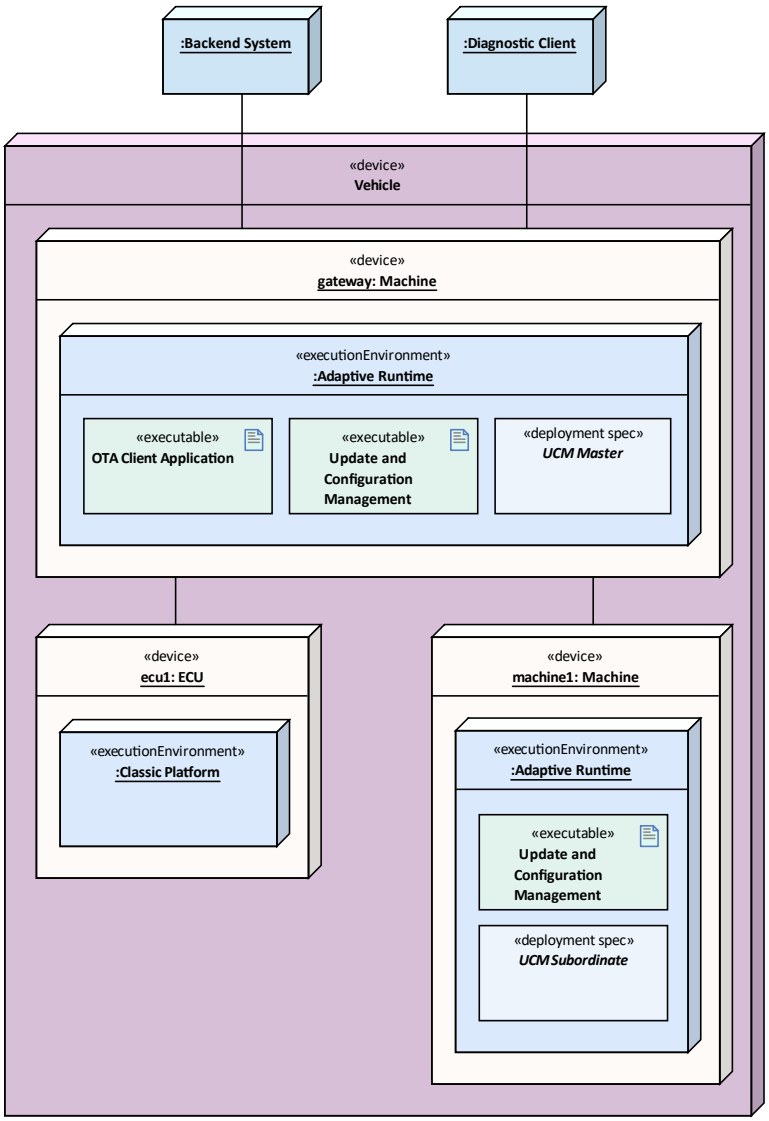

Figure 11.1: Exemplary vehicle software update scenario

`Update and Configuration Management` allows to install and update software on the AUTOSAR Adaptive Platform and AUTOSAR Classic Platform. For the AUTOSAR Adaptive Platform, `Update and Configuration Management` also allows to remove software. The software packages can be received either from a `Diagnostic Client` or from a specific `Backend System` for over-the-air updates. In a vehicle, one `Adaptive Application` takes the role of a master that controls the update process in the vehicle and distributes individual software packages to the `Machine`s and `ECU`s within a vehicle.

12 Cross-cutting Concepts

This section provides an overview of cross-cutting concepts and patterns used in the AUTOSAR Adaptive Platform.

12.1 Overview of Platform Entities

The AUTOSAR Adaptive Platform defines design entities that several `Functional Cluster`s depend on. Figure 12.1 provides an overview of these entities, their logical relationships, and the `Functional Cluster`s that depend on them. For the sake of brevity, this overview uses simplifications and abstractions over the actual specifications in the [10, manifest specification].

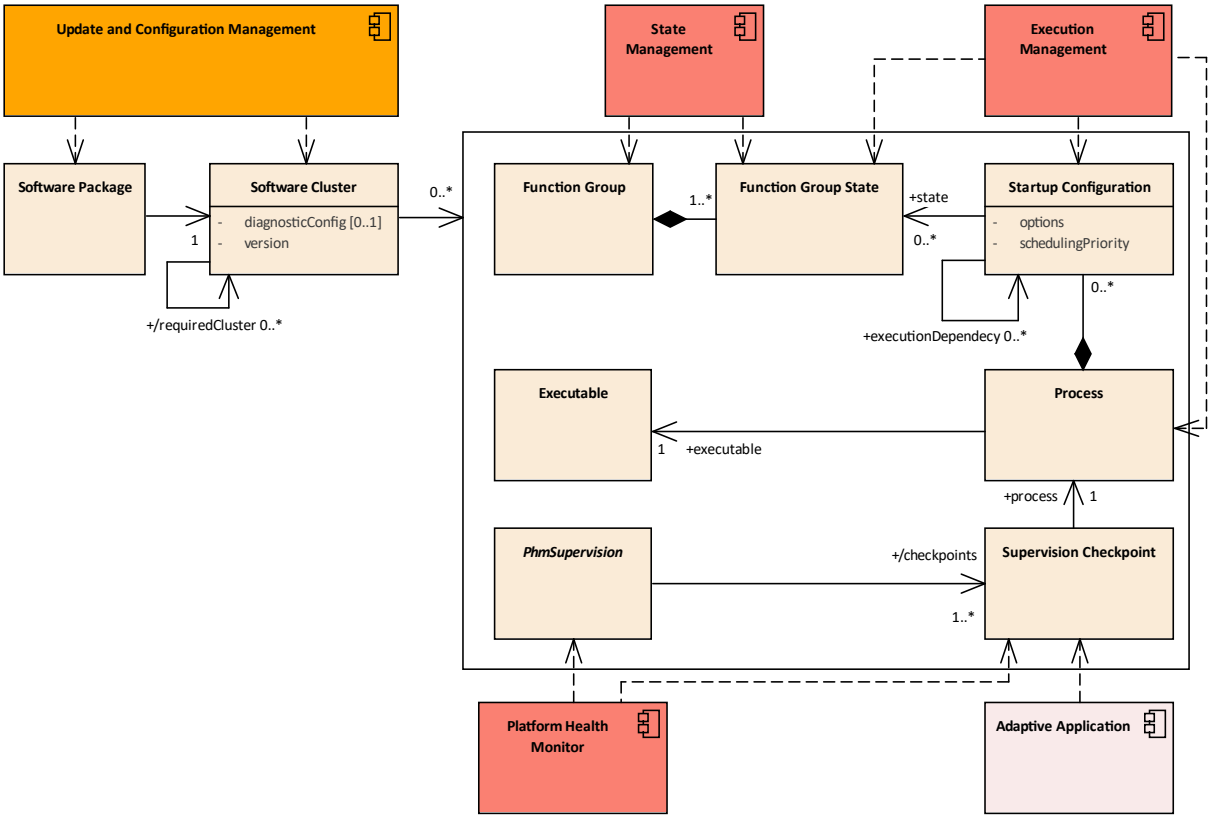

Figure 12.1: Overview of platform entities and their logical relationships

A `Software Package` is a digitally signed package that can be installed/uninstalled via `Update and Configuration Management`. A `Software Package` contains exactly one `Software Cluster` (see Section 12.4 for details). A `Software Cluster` refers to a set of `Executable`s (and other files). The corresponding executable file then holds the executable code for the `Machine` that the AUTOSAR Adaptive Platform runs on.

Additionally, a Software Cluster configuration collects a set of Processes (cf. Section 12.4) and related entities. A Process refers to an Executable and provides different Startup Configuration values, for example parameters, a scheduling priority, and resource constraints. A Startup Configuration of a Process applies to one or more Function Group States. Function Group States belong to a Function Group.

During runtime, State Management requests to enter a Function Group State from Execution Management. Execution Management then terminates and starts the Processes accordingly using the underlying Operating System.

For safety-critical systems, Platform Health Management performs supervision of Processes according to rules (logical sequence, deadlines) defined in PhmSupervisions. A PhmSupervision refers to a number of Supervision Checkpoints. During runtime, a process reports whenever it has reached such a checkpoint in its control flow.

12.2 Function Group

A Function Group is (logically) made up a set of modeled Processes that provide a certain functionality. For example, a Function Group could be an application, or a service. A special Function Group is the Machine State that groups the Process of the AUTOSAR Adaptive Platform itself. A Function Group contains a set of Function Group States.

12.3 Function Group State

A Function Group State defines which Processes of a Function Group with what configuration parameters shall be running or not. The Machine State (that refers to the Processes of the AUTOSAR Adaptive Platform itself) defines at least the following Function Group States: Off, Startup, Shutdown, and Restart.

12.4 Software Cluster

A Software Cluster configuration refers to a set of modeled Processes. Those Processes are (transitively) used by one or more Function Group(s). Hereby, a Function Group and its associated entities shall be part of only one Software Cluster. In other words, Function Groups that span several Software Clusters are invalid. A Software Cluster is packaged into one Software Package - the atomic installable/updateable unit managed by Update and Configuration Management. A Software Cluster may depend on other Software Clusters. Such dependencies are expressed by version constraints. A Software Cluster may also specify structural dependencies to Sub Software Clusters in order to

build larger installable units. The top of such a structural dependency hierarchy is called a `Root Software Cluster`. Please note that a `Software Cluster` is only used to configure deployment aspects. A `Software Cluster` is not a runtime entity.

A `Root Software Cluster` may specify a diagnostic configuration, in particular, a diagnostic address. In contrast, a `Sub Software Cluster` may depend on a diagnostic configuration of its `Root Software Cluster`. The diagnostic configuration applies to `Process`es that are (transitively) contained in a `Root Software Cluster` and its `Sub Software Cluster`(s). That means, at runtime, any diagnostic event produced by those `Process`es will be associated with the diagnostic address.

An exemplary `Software Cluster` during application design is shown in Figure 12.2. The application `Software Cluster`(s) are modeled/configured in the same way as the platform `Software Cluster` by defining `Function Group`s with `Function Group State`s and associating `StartupConfiguration`s of `Process`es to them.

A `Software Cluster` serves as a grouping entity during application design. As a result, entities within a `Software Cluster`, in particular the `Function Group`s, do not need to have a unique (simple) name within the overall model because their path is still unique. This allows to design `Software Cluster`s independently, for example, by external suppliers.

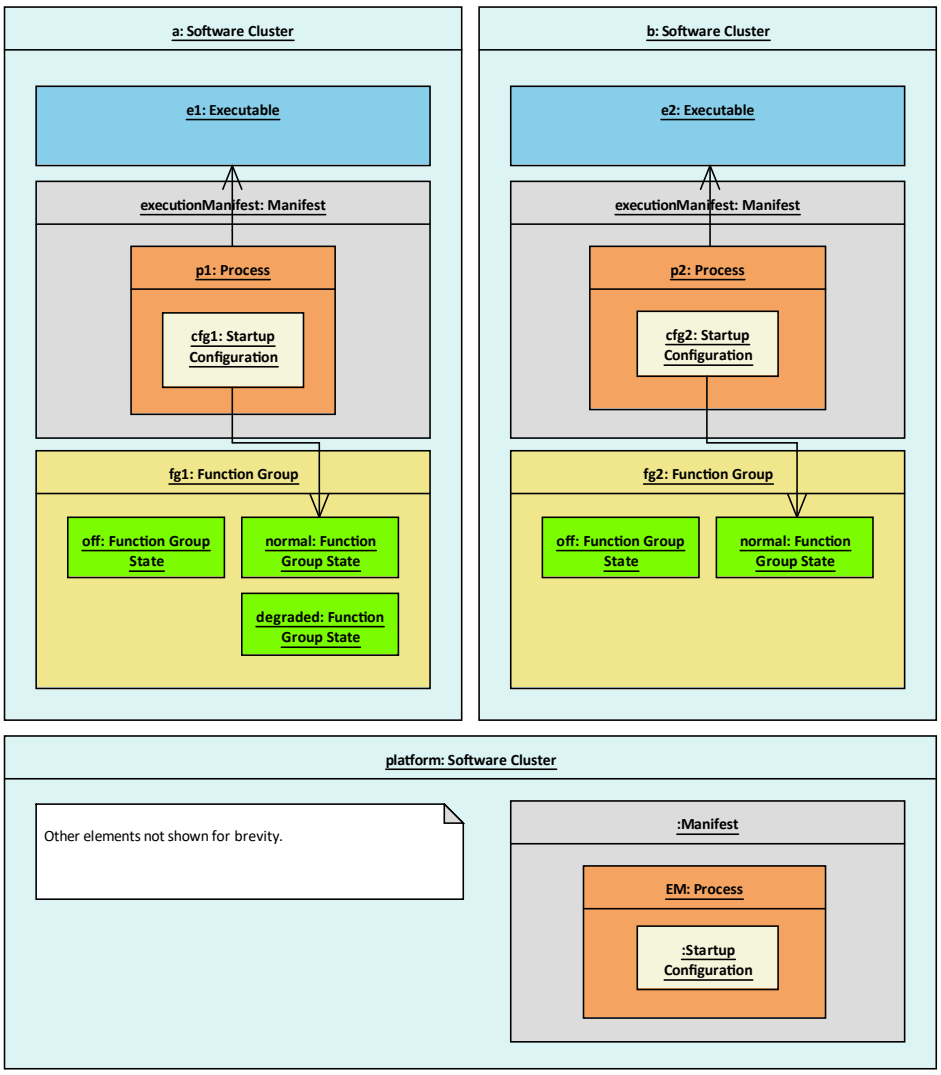

Figure 12.2: Exemplary `Software Cluster` during application design

From such a standardized model, an equivalent implementation-specific configuration for `Execution Management` is derived that is used during runtime (see Figure 12.3). That configuration advises `Execution Management` to start and configure processes accordingly when a `Function Group State` is requested. Hereby, `Execution Management` (logically) merges configurations contributed by all installed `Software Package`s. Other `Functional Cluster`s that depend on configuration provided in the `Manifest`s merge the configurations contributed by all installed `Software Package`s in the same way. Please also note that there is no corresponding runtime entity for a `Software Cluster` (see Figure 12.3).

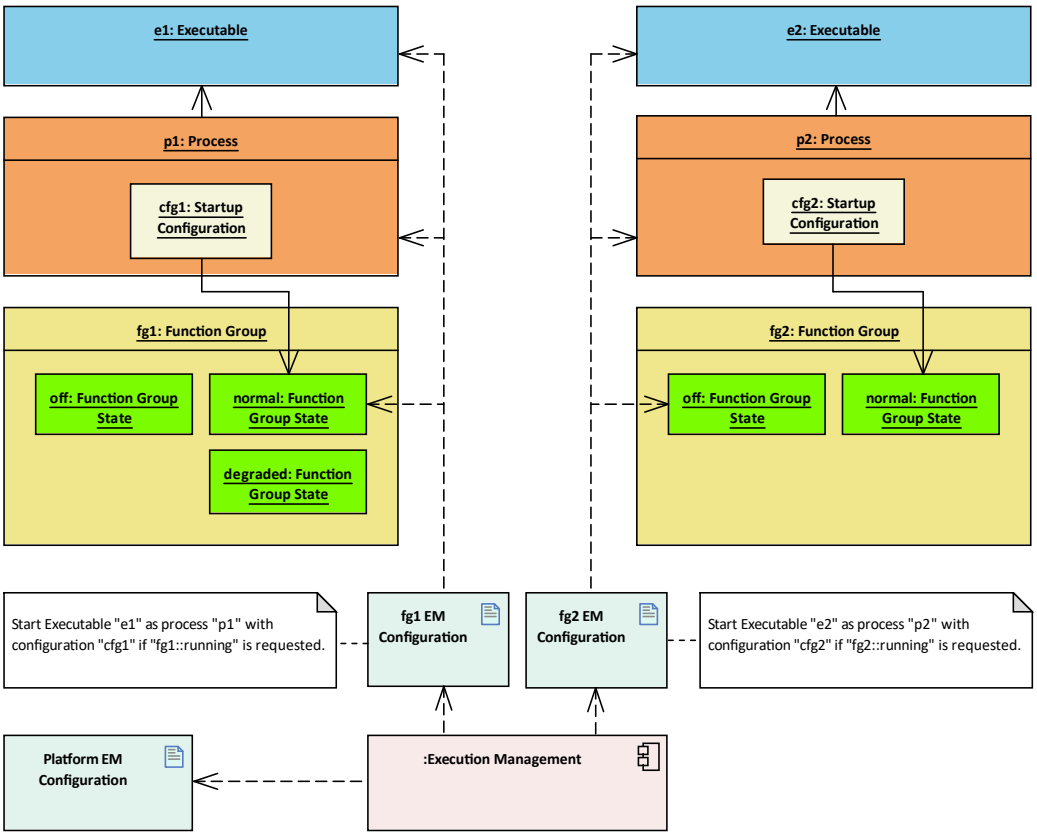

Figure 12.3: Impact of exemplary `Software Cluster` during runtime

All `Process`es related to the `Functional Cluster`s of the AUTOSAR Adaptive Platform should be referenced only in `Software Cluster`s of category `PLATFORM_CORE` or `PLATFORM`. This allows for platform-independent development of `Software Cluster`s of category `APPLICATION_LAYER`.

In case a `Functional Cluster` may need multiple logical instances (for example, `Diagnostic Management` has a logical instance per diagnostic address), an implementation of the `Functional Cluster` should still use a single physical (daemon) process.

An AUTOSAR Adaptive Platform vendor may deviate from this design guide but should provide additional countermeasures to keep `Adaptive Application`s portable.

12.5 Machine

The AUTOSAR Adaptive Platform regards hardware it runs on as a `Machine`. The rationale behind that is to achieve a consistent platform view regardless of any virtualization technology which might be used. The `Machine` might be a real physical machine, a fully-virtualized machine, a para-virtualized OS, an OS-level-virtualized container or any other virtualized environment.

On hardware, there can be one or more Machine, and only a single instance of AUTOSAR Adaptive Platform runs on a machine. It is generally assumed that this hardware includes a single chip, hosting a single or multiple Machines. However, it is also possible that multiple chips form a single Machine if the AUTOSAR Adaptive Platform implementation allows it.

12.6 Manifest

A Manifest represents a piece of AUTOSAR model description that is created to support the configuration of an AUTOSAR Adaptive Platform product and which is uploaded to the AUTOSAR Adaptive Platform product, potentially in combination with other artifacts (like binary files) that contain executable code to which the Manifest applies. Please note that a typical Adaptive Application will make use of several distinct but interrelated Manifests. Hereby, the individual Manifests contribute information to the complete application model. For example, each Software Cluster may contribute a self-contained set of Manifests to configure its functionality.

The usage of a Manifest is limited to the AUTOSAR Adaptive Platform. This does not mean, however, that all ARXML produced in a development project that targets the AUTOSAR Adaptive Platform is automatically considered a Manifest. In fact, the AUTOSAR Adaptive Platform is usually not exclusively used in a vehicle project. A typical vehicle will most likely be also equipped with a number of ECUs developed on the AUTOSAR Classic Platform and the system design for the entire vehicle will, therefore, have to cover both, ECUs built on top of the AUTOSAR Classic Platform and Machines created on top of the AUTOSAR Adaptive Platform.

In principle, the term Manifest could be defined such that there is conceptually just one "Manifest" and every deployment aspect would be handled in this context. This does not seem appropriate because it became apparent that Manifest-related model-elements exist that are relevant in entirely different phases of a typical development project.

This aspect is taken as the main motivation that next to the application design it is necessary to subdivide the definition of the term Manifest in three different partitions:

Application Design This kind of description specifies all design-related aspects that apply to the creation of application software for the AUTOSAR Adaptive Platform. It is not necessarily required to be deployed to the adaptive platform machine, but the application design aids the definition of the deployment of application software in the Execution Manifest and Service Instance Manifest. See Section 12.7 for details.

Execution Manifest This kind of Manifest is used to specify the deployment-related information of applications running on the AUTOSAR Adaptive Platform. An Execution Manifest is bundled with the actual executable code to support the integration of the executable code onto the machine. See Section 12.8 for details.

Service Instance Manifest This kind of `Manifest` is used to specify how service-oriented communication is configured in terms of the requirements of the underlying transport protocols. A `Service Instance Manifest` is bundled with the actual executable code that implements the respective usage of service-oriented communication. See Section 12.9 for details.

Machine Manifest This kind of `Manifest` is supposed to describe deployment-related content that applies to the configuration of just the underlying machine (i.e. without any applications running on the machine) that runs an AUTOSAR Adaptive Platform. A `Machine Manifest` is bundled with the software taken to establish an instance of the AUTOSAR Adaptive Platform. See Section 12.10 for details.

The temporal division between the definition (and usage) of different kinds of `Manifest` leads to the conclusion that in most cases different physical files will be used to store the content of the three kinds of `Manifest`. In addition to the Application Design and the different kinds of `Manifest`, the AUTOSAR Methodology supports a System Design with the possibility to describe Software Components of both AUTOSAR Platforms that will be used in a System in one single model. The Software Components of the different AUTOSAR platforms may communicate in a service-oriented way with each other. But it is also possible to describe a mapping of Signals to Services to create a bridge between the service-oriented communication and the signal-based communication.

12.7 Application Design

The application design describes all design-related modeling that applies to the creation of application software for the AUTOSAR AP. Application Design focuses on the following aspects:

- Data types used to classify information for the software design and implementation
- Service interfaces as the pivotal element for service-oriented communication
- Definition how service-oriented communication is accessible by the application
- Persistency Interfaces as the pivotal element to access persistent data and files
- Definition how persistent storage is accessible by the application
- Definition how files are accessible by the application
- Definition how crypto software is accessible by the application
- Definition how the Platform Health Management is accessible by the application
- Definition how Time Bases are accessible by the application
- Serialization properties to define the characteristics of how data is serialized for the transport on the network

- REST service interfaces as the pivotal element to communicate with a web service by means of the REST pattern
- Description of client and server capabilities
- Grouping of applications in order to ease the deployment of software.

The artifacts defined in the application design are independent of a specific deployment of the application software and thus ease the reuse of application implementations for different deployment scenarios.

12.8 Execution Manifest

The purpose of the execution manifest is to provide information that is needed for the actual deployment of an application onto the AUTOSAR AP. The general idea is to keep the application software code as independent as possible from the deployment scenario to increase the odds that the application software can be reused in different deployment scenarios. With the execution manifest the instantiation of applications is controlled, thus it is possible to

- instantiate the same application software several times on the same machine, or to
- deploy the application software to several machines and instantiate the application software per machine.

The Execution manifest focuses on the following aspects:

- Startup configuration to define how the application instance shall be started. The startup includes the definition of startup options and access roles. Each startup may be dependent on machines states and/or function group states.
- Resource Management, in particular resource group assignments.

12.9 Service Instance Manifest

The implementation of service-oriented communication on the network requires configuration which is specific to the used communication technology (e.g. SOME/IP). Since the communication infrastructure shall behave the same on the provider and the requesters of a service, the implementation of the service shall be compatible on both sides.

The Service Instance Manifest focuses on the following aspects:

- Service interface deployment to define how a service shall be represented on the specific communication technology.

- Service instance deployment to define for specific provided and required service instances the required credentials for the communication technology.
- The configuration of E2E protection
- The configuration of Security protection
- The configuration of Log and Trace

12.10 Machine Manifest

The machine manifest allows to configure the actual adaptive platform instance running on specific hardware (machine).

The Machine Manifest focuses on the following aspects:

- Configuration of the network connection and defining the basic credentials for the network technology (e.g. for Ethernet this involves setting of a static IP address or the definition of DHCP).
- Configuration of the service discovery technology (e.g. for SOME/IP this involves the definition of the IP port and IP multi-cast address to be used).
- Definition of the used machine states.
- Definition of the used function groups.
- Configuration of the adaptive platform functional cluster implementations (e.g. the operating system provides a list of OS users with specific rights).
- The configuration of the Crypto platform Module.
- The configuration of Platform Health Management.
- The configuration of Time Synchronization.
- Documentation of available hardware resources (e.g. how much RAM is available; how many processor cores are available).

12.11 Error Handling

Proper handling of errors during runtime is an important aspect to build safe and secure systems. The AUTOSAR Adaptive Platform does provide means for raising and handling of such errors on different levels in the platform.

`Platform Health Management` detects errors (errors in the logical control flow, missed deadlines, and missed liveness reporting) at the level of `Processes` and performs recovery actions (for example, degradation) according to rules defined in the `Manifest. Execution Management` detects unexpected termination of `Processes` and reports to `State Management` for handling of such errors.

During execution of a Process of an Adaptive Application, different abnormal conditions might be detected and need to be handled and/or reported. The following types of unsuccessful operations are distinguished within the AUTOSAR Adaptive Platform:

- An Error is the inability of an AUTOSAR Runtime for Adaptive Applications API function to fulfill its specified purpose. An Error it is often a consequence of invalid and/or unexpected input data. An Error is considered to be recoverable and therefore shall be handled by applications.
- A Violation is the consequence of failed pre- or post-conditions of internal state of the AUTOSAR Runtime for Adaptive Applications. A Violation is considered to be non-recoverable.
- A Corruption is the consequence of the corruption of a system resource, e.g. stack or heap overflow, or a hardware memory flaw (for example, a detected bit flip). A Corruption is considered to be non-recoverable.
- A failed default allocation is the inability of the AUTOSAR Runtime for Adaptive Applications's default memory allocation mechanism to satisfy an allocation request (for example, there is not enough free memory available).

It is expected that a Violation or Corruption will not be experienced by a user of the AUTOSAR Adaptive Platform (i.e. an application developer), unless there is something seriously wrong in the overall system. For example, faulty hardware may lead to a Corruption. A Violation may occur if basic assumptions about resource requirements are violated, or the user runs the AUTOSAR Runtime for Adaptive Applications in a configuration that is not supported by its vendor.

12.12 Trusted Platform

To guarantee the correct function of the system, it is crucial to ensure that the code executed on the AUTOSAR Adaptive Platform is unaltered (integrity) and has legitimate origin (authenticity). Keeping this property allows the integrator to build a Trusted Platform. A key property of a system that implements a Trusted Platform is a Trust Anchor (also called Root of Trust). A Trust Anchor is often realized as a public key that is stored in a secure environment, e.g. in non-modifiable persistent memory or in an Hardware Security Module. A system designer is responsible to ensure that the system starts beginning with a Trust Anchor and that the chain of trust is kept until the Execution Management is launched. Depending on the mechanism that is chosen by the system designer to establish the chain of trust, the integrity and authenticity of the entire system (including all executables) may be checked during system start-up. Alternatively, the system designer might only ensure that the already executed software has been checked regarding integrity and authenticity and Execution Management takes over responsibility on continuing the chain of trust when it takes over control of the system. In the latter case, the system integrator is responsible to ensure that the Execution Management is configured accordingly.

Passing trust requires that a trusted entity checks (using trusted functionality) that the entity down the chain is authentic. The Trust Anchor (the first entity in the chain) is authentic by definition. An example of such a chain of trust could look like this: The Trust Anchor authenticates the bootloader before the bootloader is being started. In each subsequent step in the boot process, the to-be-started executable shall be authenticated first, for example by the executable started previously or by some external entity like an Hardware Security Module. After the relevant parts of the Operating System have been authentically started, it shall launch Execution Management as one of its first processes in the same manner passing trust to the AUTOSAR Adaptive Platform. Then, Execution Management takes over the responsibility of authenticating Adaptive Applications before launching them.

As stated above, if authenticity is not checked by the functionality of the Trust Anchor itself, which is authentic by definition, the functionality that shall be applied to verify authenticity of an executable has to be authenticated as well before it is applied. For instance, if the Crypto Functional Cluster shall be used to verify authenticity of executables, the Crypto Functional Cluster itself shall be authenticated by some trusted entity before it is used.

12.13 Secure Communication

AUTOSAR supports different protocols that provide communication security over a network. Integrity of messages can be ensured by the end-to-end protection offered by the [11, AUTOSAR E2E library]. End-to-end protection assumes that safety- and security-related data exchange shall be protected at runtime against the effects of faults within the communication link. Such faults include random hardware faults (e.g. corrupt registers of a transceiver), interference (e.g. electromagnetic interference), and systematic faults in the communication stack. The configuration of end-to-end-protection is done via Service Instance Manifest on level of Service events, methods, and fields (notifier, get, and set methods). Confidentiality and authenticity of messages can be ensured by dedicated configurations for the individual transport protocols (e.g. TLS, SecOC) in the Service Instance Manifest on level of Service events, methods, and fields (notifier, get, and set methods).

13 Risks and Technical Debt

This chapter lists and rates risks associated with the overall architecture of the AUTOSAR Adaptive Platform in Section 13.1. These risks usually might cause that some of the quality attributes of the AUTOSAR Adaptive Platform are not (fully) meat. Section 13.2 lists technical debt of the AUTOSAR Adaptive Platform that may impact its maintainability.

13.1 Risks

13.1.1 Risk Assessment

This document categorizes risks according to their severity. The severity is a function of the probability and the impact of a risk. The probabilities are categorized as follows:

- **very low** - probability is less than 1 thousandth
- **low** - probability is between 1 thousandth and 1 percent
- **medium** - probability is between 1 percent and 10 percent
- **high** - probability is between 10 percent and 50 percent
- **very high** - probability is more than 50 percent

The impact of a risk is categorized as follows:

- **very low** - at most one quality scenario will take additional significant effort to be satisfied
- **low** - more than one quality scenario will take additional significant effort to be satisfied
- **medium** - at most one quality scenario is not satisfied with small gaps
- **high** - at most one quality scenario is not satisfied with big gaps
- **very high** - more than one quality scenario is not satisfied with big gaps

The final severity of a risk is then calculated according to table 13.1.

	Probability				
Impact	very low	low	medium	high	very high
very low	low (1)	low (2)	low (3)	medium (4)	medium (5)
low	low (2)	medium (4)	medium (6)	high (8)	high (10)
medium	low (3)	medium (6)	high (9)	high (12)	high (15)
high	medium (4)	high (8)	high (12)	extreme (16)	extreme (20)
very high	medium (5)	high (10)	high (15)	extreme (20)	extreme (25)

Table 13.1: Risk Severity Matrix

13.1.2 Risk List

No architectural risks were identified yet.

13.2 Technical Debt

No technical debt has been identified yet.

References

[1] ISO 42010:2011 – Systems and software engineering – Architecture description
http://www.iso.org

[2] Explanation of Adaptive Platform Software Architectural Decisions
AUTOSAR_EXP_SWArchitecturalDecisions

[3] Glossary
AUTOSAR_TR_Glossary

[4] Main Requirements
AUTOSAR_RS_Main

[5] General Requirements specific to Adaptive Platform
AUTOSAR_RS_General

[6] ATAM[SM]: Method for Architecture Evaluation
https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13706.pdf

[7] Agile Software Development: Principles, Patterns, and Practices

[8] Guide to the Software Engineering Body of Knowledge, Version 3.0
www.swebok.org

[9] API standards for Open Systems
http://www.opengroup.org/austin/papers/wp-apis.txt

[10] Specification of Manifest
AUTOSAR_TPS_ManifestSpecification

[11] Specification of SW-C End-to-End Communication Protection Library
AUTOSAR_SWS_E2ELibrary