Document Title	Explanation of Adaptive Platform Software Architectural Decisions
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	983

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 additional information on architectural decisions made for the AUTOSAR Adaptive Platform standard.

1.1 Objectives

The main objective of this document is to provide a **documentation of architectural decisions** made for the AUTOSAR Adaptive Platform to make such decisions comprehensible and reviewable in the future and ultimately get a more maintainable standard.

1.2 Scope

This document covers decisions made for the software architecture of the AUTOSAR Adaptive Platform. The main audience of this document are architects of the AUTOSAR Adaptive Platform as well as members of other working groups.

2 Definition of Terms and Acronyms

2.1 Acronyms and Abbreviations

Abbreviation / Acronym	Description
API	Application Programming Interface
STL	Standard Template Library

2.2 Definition of Terms

Term	Description
Adaptive Application	See [1, AUTOSAR Glossary].
Execution Management	A `Functional Cluster`. See [2, EXP_SWArchitecture] for an overview.
Functional Cluster	A logical group of functionality within the AUTOSAR Adaptive Platform. Functional Clusters are the subject of the individual specification documents that make up the AUTOSAR Adaptive Platform standard. See [2, EXP_SWArchitecture] for an overview of all Functional Clusters in the AUTOSAR Adaptive Platform.
Platform Health Management	A `Functional Cluster`. See [2, EXP_SWArchitecture] for an overview.
Process	See [1, AUTOSAR Glossary].
State Management	A `Functional Cluster`. See [2, EXP_SWArchitecture] for an overview.
Software Cluster	See [1, AUTOSAR Glossary] and [2, EXP_SWArchitecture].
Thread	See [1, AUTOSAR Glossary].
Watchdog	An external component that supervises execution of the AUTOSAR Adaptive Platform. See [2, EXP_SWArchitecture] for an overview.

3 Related Documentation

This document provides an overview of the architectural decisions that have been made for the AUTOSAR Adaptive Platform and their rationale. A high-level overview of the current AUTOSAR Adaptive Platform architecture is provided in [2, EXP_SWArchitecture].

4 Overview

This chapter provides and overview of the organization and structure of decisions listed in this document. All decisions are organized in a table as described below.

Date of approval	Date when the decision has been approved by CF-CCB.
Decision	The decision itself. Impact or direct consequences (for example, changes to interfaces) of the decision are not documented in the normal case. Changes are requested during the roll-out process after the decision has been made.
Rationale	A rationale for the decision.
Architectural cluster	A list of architectural clusters that are affected by the decision.
Category	Category of the decision.
Application affected	States if the decision has an direct impact on existing applications.
Assumptions	Lists the assumptions that have been made before making the decision itself. These assumptions are documented in order to be able to review decisions in the future and check if some assumptions probably no longer hold.
Constraints	Provides an overview of the constraints that were identified to have an impact on possible solutions. The constraints are also documented in order to be reference points for future reviews of the decision.
Alternatives	Lists the alternatives that were considered and a rationale why they are worse than the decision that has been made.
Remarks	Lists remarks on the decisions.
Related requirements	Lists requirements related to the decision.

5 Architectural Decisions

This chapter lists architectural decisions that have been made for the AUTOSAR Adaptive Platform.

5.1 Dynamic memory allocation

Date of approval	2020-09-15
Decision	The use of dynamic memory allocation by Adaptive Applications and Functional Clusters is allowed and assumed upon designing the AUTOSAR Adaptive Platform standard.
Rationale	The use of dynamic memory allocation is essentially indispensable as the AUTOSAR Adaptive Platform standard employs C++ as the language for its API. As the AUTOSAR Adaptive Platform standard will be used in safety-related systems, dynamic memory allocation can cause non-deterministic behavior. Two typical issues are the fragmentation and non-deterministic allocation/de-allocation processing time. Memory allocators designed for non-safety-critical systems often exhibit such issues, as they are more or less designed for memory efficiency and/or average processing performance. These issues can be controlled by using deterministic memory allocators. Memory allocation is a well-studied area and various techniques have been reported (Refer to references below for some examples). Multiple AUTOSAR partners within the architecture group reportedly have such deterministic memory allocators implemented and have been used in mass-production systems. Note that such allocators should replace the default `malloc()/free()` implementations provided in the standard C library, that sits underneath the C++ runtime library providing `new()/delete()` and also STL that AUTOSAR Adaptive Platform also uses. This frees applications from providing its own custom deterministic allocators and installing it to custom-allocator-aware classes. Please refer to [3], [4], [5], and [6] for further information on memory fragmentation and memory allocation in real-time systems.

▽

△

Architectural cluster	• Communication Management • Adaptive Core • REST • Persistency • Execution Management • Time Synchronization • Platform Health Management • Identity and Access Management • Diagnostic Management • Log and Trace • Cryptography • State Management • Update and Configuration Management • Network Management
Category	Safety
Application affected	No
Assumptions	Platform vendors and/or compiler vendors can replace the default memory allocation/deallocation functions to use deterministic versions of those functions during critical phases of the runtime when such determinism is required for safety purposes.
Constraints	During certain phases of the runtime determinism is required. These are the phases in which the allocators need to be replaced with deterministic versions.
Alternatives	• Not using dynamic memory allocation is not an alternative for using C++. • Disallow dynamic memory allocation during certain phases of the runtime in which determinism is required. This makes it very difficult to run complex code during these phases.
Remarks	No remarks.
Related requirements	• [RS_AP_00129] Public types defined by functional clusters shall be designed to allow implementation without dynamic memory allocation

5.2 Making Adaptive Runtime classes final

Date of approval	2020-09-15
Decision	`Adaptive Runtime` types shall use the `final` specifier unless they are meant to be used as a base class.

▽

△

Rationale	Making classes `final` that are not intended to be used as base class expresses the design (in particular the class hierarchy) more explicit. This will avoid problems such as • to derive from a class that is not prepared for sub-classing, • to inadvertently create a new virtual function instead of overwriting a function from the base class due to a slightly different signature.
Architectural cluster	• `Communication Management` • `Adaptive Core` • `REST` • `Persistency` • `Execution Management` • `Time Synchronization` • `Platform Health Management` • `Diagnostic Management` • `Log and Trace` • `Cryptography`
Category	None
Application affected	No
Assumptions	A clear expression of the intended design of the public AUTOSAR Runtime for Adaptive Applications class hierarchy.
Constraints	No constraints were identified.
Alternatives	The alternative is to have a code review of the application code using AUTOSAR types. This is far out-of-scope of AUTOSAR wherefore it is not a real alternative.
Remarks	No remarks.
Related requirements	• [RS_AP_00140] Usage of "final specifier" in ara types

5.3 Usage of out parameters

Date of approval	2020-09-15
Decision	Out parameters can be used for in-place modifications but shall not be used for returning values.
Rationale	Harmonized look and feel. C++ Core Guidelines [7]: “F.20: For "out" output values, prefer return values to output parameters. [...] A return value is self-documenting, whereas a `&` could be either in-out or out-only and is liable to be misused. This includes large objects like standard containers that use implicit move operations for performance and to avoid explicit memory management.”

▽

△

Architectural cluster	• `Persistency`
Category	None
Application affected	No
Assumptions	Dynamic memory allocation is allowed for all cases in which the APIs are used, even when running time critical safety related code including ASIL D.
Constraints	In/out parameters, i.e. modifying an already existing parameter within a function is allowed. For example, a function that clears or writes to a buffer should receive that buffer as a non-const in/out parameter.
Alternatives	No alternatives were considered.
Remarks	No remarks.
Related requirements	• [RS_AP_00141] Usage of out parameters

5.4 Usage of named constructors for exception-less object creation

Date of approval	2020-10-06
Decision	Exceptionless functions for creation of objects which returns an `ara::core::Result` should use the "named constructor idiom".
Rationale	Disadvantages of constructor token approach are avoided as follows: • The constructor token type is an implementation detail of a `Class` and should thus not be specified, or even accessible from outside. This makes the use of `auto` for obtaining a token mandatory because the token type cannot be referred to in any other way. • Moving the token's content to the `SomeClass` instance has to be done very carefully to fulfill the always-successful guarantee, which can be tricky if multiple resources need to be acquired. • The token object is "destroyed" by `std::move`-ing its value into the `SomeClass` constructor (actually, it is to be in a "valid" but unspecific state according to the C++ standard), but it is easily possible to mistakenly use it again for attempting to create another instance, with undefined results.
Architectural cluster	• `Communication Management` • `Adaptive Core` • `REST` • `Persistency` • `Execution Management` • `Time Synchronization` • `Platform Health Management` ▽

▽

△

	△ • `Identity and Access Management` • `Diagnostic Management` • `Log and Trace` • `Cryptography` • `State Management` • `Update and Configuration Management` • `Network Management`
Category	Safety
Application affected	Yes
Assumptions	No assumptions were made.
Constraints	No constraints were identified.
Alternatives	• Constructor token approach. It was not considered due to the drawbacks described in the rationale of this decision. • Regular constructor calls. Regular constructor calls were not considered because regular constructors may throw exceptions and thus cannot be used in an exception-less design.
Remarks	No remarks.
Related requirements	No related requirements.

5.5 Introduction of a monotonic clock API

Date of approval	2020-10-20
Decision	The AUTOSAR Runtime for Adaptive Applications shall provide its own monotonic `std::chrono::SteadyClock` representing the power-up time of the machine. The accuracy of this clock is defined by the platform vendor. The accuracy of this clock could be used as a characteristic value of the platform so that the projects could check whether this clock meets the project-specific requirements (e.g. time synchronization requires typically a clock with higher accuracy). The system start of the machine defines the epoch of the clock. So this clock represents the power-up time of the machine. Functional Clusters dealing with timestamps or clocks should use this clock as a basis.
Rationale	The timestamps used in the time synchronization cluster should be based on `std::chrono`. Time synchronization requires a monotonic clock with special accuracy.

▽

△

Architectural cluster	• Adaptive Core • Execution Management • Time Synchronization
Category	None
Application affected	Yes
Assumptions	The time synchronization cluster is typically a daemon-based architecture due to a single communication endpoint of the time sync messages. A standardized clock with a special accuracy as a common basis is required to synchronize the daemon with the library.
Constraints	No constraints were identified.
Alternatives	The used clock could also be passed as a template argument. But a standardized clock with a special accuracy as a common basis is required anyway in case the time synchronization cluster is daemon based.
Remarks	No remarks.
Related requirements	No related requirements.

5.6 Responsibilities of State Management, Execution Management, and Platform Health Management

Date of approval	2020-10-27
Decision	`State Management`, `Execution Management`, and `Platform Health Management` are the fundament/basis of the AUTOSAR Adaptive Platform. A failure in either `State Management`, `Platform Health Management`, or `Execution Management` process will typically lead to stop triggering the watchdog. `Platform Health Management` supervises `State Management` and `Execution Management`. `Platform Health Management` controls the watchdog and is thus in turn supervised by the hardware watchdog. Triggering of a Machine reset as a last resort should not be an option at all in case of a failing of an Adaptive Application supervision (i.e. apart from `Operating System` / `Execution Management` / `State Management` / `Platform Health Management`). A supervision failure in an Adaptive Application shall be reported to `State Management`. `State Management` may forward this failure based on the criticality to `Platform Health Management` to wrongly trigger or stop triggering the serviced watchdog. `Platform Health Management` performs a logical supervision of checkpoints within a process or between processes within a `Function Group`. `Platform Health Management` reports any supervision failures to `State Management`. `State Management` is responsible to perform recovery actions including a switch of the `Function Group State`, by delegating to the Adaptive Application, or, as a last resort, by advising `Platform Health` ▽

▽

△

	△ Management to perform a hardware reset. Platform Health Management is intended for supervision of safety-critical processes. Thus, Platform Health Management is an optional part of the AUTOSAR Adaptive Platform for non safety-critical applications. Processes shall never be restarted on their own because they may have unknown runtime dependencies. The relation between a Process and a Function Group is comparable to the relation between a thread and a process. State Management should always trigger a request (Function Group State change) to restart processes even in the simplistic/non-dependent cases. Thus, Platform Health Management does not have a direct interface to Execution Management. The unrecoverable state interface of Platform Health Management shall be removed.
Rationale	The chosen solution leads to a simpler design of Platform Health Management with a single and well-defined responsibility. The chosen solution also adheres to the single responsibility principle for State Management (control system state) and Execution Management (control processes) as well. Recovery actions can be added by extension (open-closed principle) to State Management. There is no need to modify or configure Platform Health Management. Supervision failures may be handled by an Adaptive Application as well if State Management chooses to delegate recovery to the Adaptive Application.
Architectural cluster	• Execution Management • Platform Health Management • State Management
Category	Safety
Application affected	Yes
Assumptions	• State Management is a mandatory part of the AUTOSAR Adaptive Platform. • Performance impact / delay of indirect reporting of supervision failures to an Adaptive Application via State Management is negligible in comparison to execution of reasonable recovery actions (such as starting processes).
Constraints	No constraints were identified.
Alternatives	• Alternative 1 - Failure recovery coordinated by Platform Health Management Recovery in case of a systematic failure is coordinated by Platform Health Management. Several components (Adaptive Application, Execution Management, State Management, watchdog) are involved based on priorities. Platform Health Management coordinates the recovery in the following manner: ▽

▽

△

	△ 1. Platform Health Management asks the Adaptive Application to recover 2. In case of failure, Platform Health Management asks Execution Management to restart failed processes 3. In case of failure, Platform Health Management asks State Management to recover by switching the Function Group State 4. In case of failure, Platform Health Management stops triggering the watchdog and resets the Machine 5. In case of failure, Platform Health Management switches to unrecoverable state (not yet fully defined) Alternative 1 was not considered due to not adhering to the single responsibility principle because several components are responsible for recovery actions. This solution would also require Platform Health Management to have application knowledge because it has to determine the appropriate Function Group State in step 3. Restarting single processes may not be appropriate (step 2) due to runtime dependencies. • Alternative 2 - Distributed failure recovery Recovery in case of a systematic failure is coordinated by Platform Health Management and State Management. Several components (Adaptive Application, Execution Management, watchdog) are involved based on priorities. Platform Health Management and State Management coordinate the recovery in the following manner: 1. Platform Health Management asks the Adaptive Application to recover 2. In case of failure, Platform Health Management asks State Management to coordinate recovery by restarting the application 3. State Management asks Execution Management to change state / switch to degraded state or safe state 4. In case of failure, State Management asks Adaptive Application to recover 5. In case step 2 failed due to application dependencies, Platform Health Management stops triggering the watchdog and resets the Machine Alternative 2 was not considered due to not adhering to the single responsibility principle because Platform Health Management and State Management share responsibility for coordinating recovery actions.
Remarks	• According to ISO 26262, it has to be ensured that a reaction is triggered after a safety-relevant failure occurred. Therefore, Platform Health Management shall make sure that State Management receives the notification on a detected failure even if they communicate via an unreliable communication channel, for example, an inter-process communication mechanism. To achieve this, Platform Health Management should implement a timeout monitoring. If no response by ▽

▽

△

	△ State Management is received after a configurable timeout and number of tries, Platform Health Management shall trigger a reaction via hardware Watchdog. • For release R19-11 of the AUTOSAR Adaptive Platform, the configuration of Platform Health Management included rules for monitoring (PhmSupervision), arbitration and recovery actions. With this decision, Platform Health Management is only responsible for monitoring. The rules for monitoring (PhmSupervision) are unaffected. However, the responsibilities for arbitration and recovery actions are moved to State Management. In the current design, State Management is a piece of project-specific, coded software with only little configuration. The configuration for State Management should be extended to support arbitration and recovery actions as well. This will allow to validate such configurations based on standardized rules which is extremely hard to achieve on source code level.
Related requirements	No related requirements.

5.7 Use of local proxy objects for shared access to objects

Date of approval	2020-10-27
Decision	Local proxy object(s) shall be used to provide shared access to object instance(s) via the AUTOSAR Runtime for Adaptive Applications interface.
Rationale	Local proxy objects hide the implementation details of the shared access. The AUTOSAR Runtime for Adaptive Applications interface shall return a proxy object by value. The caller shall use the object as a local proxy for subsequent communication. Return by value is the most straightforward way to return data. This decision enforces harmonization of the AUTOSAR Runtime for Adaptive Applications interface. Stack vendors may freely choose how to implement the shared access inside the proxy class. An example for the use of a local proxy object by the caller is the following:

```

Result<void> myFunc() {
    Result<KeyValueStorage> kvsRes
        = KeyValueStorage::Create(KVS_ID);
    if (kvsRes) {
        KeyValueStorage kvs = std::move(kvsRes).Value();
        auto keyRes = kvs.GetAllKeys(); // Value semantics
        // ...
    } else {
        return {std::move(kvsRes).Error()};
    }
}
```

▽

△

Architectural cluster	• Communication Management • Adaptive Core • REST • Persistency • Execution Management • Time Synchronization • Platform Health Management • Identity and Access Management • Diagnostic Management • Log and Trace • Cryptography • State Management • Update and Configuration Management • Network Management
Category	None
Application affected	Yes
Assumptions	No assumptions were made.
Constraints	No constraints were identified.
Alternatives	The alternative of using proxy classes is the usage of handles. These handles would however reveal the implementation details of the shared access.
Remarks	No remarks.
Related requirements	• [RS_AP_00135] Avoidance of shared ownership

References

[1] Glossary
AUTOSAR_TR_Glossary

[2] Explanation of Adaptive Platform Software Architecture
AUTOSAR_EXP_SWArchitecture

[3] Dynamic Memory Allocation and Fragmentation
https://www.researchgate.net/publication/295010953_Dynamic_Memory_Allocation_and_Fragmentation

[4] Dynamic Memory Allocation on Real-Time Linux
https://static.lwn.net/images/conf/rtlws-2011/proc/Jianping.pdf

[5] TLSF: a new dynamic memory allocator for real-time systems
https://doi.org/10.1109/EMRTS.2004.1311009

[6] The Memory Fragmentation Problem: Solved?
https://doi.org/10.1145/286860.286864

[7] C++ Core Guidelines
https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md