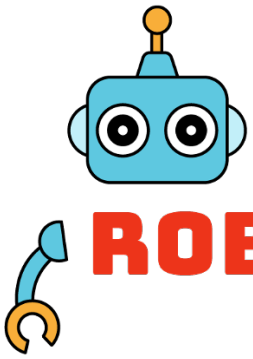

ROBOSAPIENS

D5.1

Initial architecture for the RoboSAPIENS platform

WP5

Public Document

Grant Agreement	101133807
Project	RoboSapiens
Deliverable Number	D5.1
Version	1.0
Due Month	9
Date	September 2024

http://robosapiens-eu.tech/

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or HADEA. Neither the European Union nor the granting authority can be held responsible for them.

Contributors:

Bert Van Acker, UAntwerp
Sahar Nasimi Nezhad, UAntwerp
Paul De Meulenaere, UAntwerp
Anastasios Tefas, AUTH
Nikos Nikolaidis, AUTH
Pavlos Tosidis, AUTH
Ana Cavalcanti, UoY
James Baxter, UoY
Thomas Wright, AU

Editors:

Bert Van Acker, UAntwerp
Sahar Nasimi Nezhad, UAntwerp
Paul De Meulenaere, UAntwerp

Internal reviewers:

Guoyuan Li, NTNU
Thomas Roehr, SRL

Consortium:

Aarhus University	**AU**	University of Antwerp	**UA**
Aristotle University of Thessaloniki	**AUTH**	Norwegian University of Science and Technology	**NTNU**
Danish Technological Institute	**DTI**	PAL Robotics	**PAL**
Fraunhofer IFF	**IFF**	ISDI Accelerator	**ISDI**
University of York	**UoY**	Simula Research Lab	**SRL**

Funded by
the European Union

Document Revision History:

Ver	Date	Author	Description
0.1	12-02-2024	Bert Van Acker	Initial document version
0.2	7-10-2024	Bert Van Acker, Sahar Nasimi	Document ready for internal review
0.3	30-09-2024	Bert Van Acker, Sahar Nasimi	Updated according to internal review
1.0	30-09-2024	Bert Van Acker, Sahar Nasimi	Finalized according to internal review

Funded by
the European Union

Abstract

This deliverable presents the progress made in Work Package 5 (WP5) up to month nine (M9), with a detailed focus on several key areas. Firstly, we discuss the scope and context of this document. Secondly, we provide a detailed view on the proposed architecture, using different viewpoints, addressing different concerns. Using the **building block view**, we provide a detailed description of each architecture component and its interfaces. In the **runtime view**, we provide a structured representation of the behavior of the building blocks as a series of sequential steps over time. This view depicts the workflows, the way how the building blocks should interact with each other to achieve trustworthy adaptation behavior. In the **deployment view**, we provide how the proposed architecture, building blocks and corresponding interfaces, can be deployed and executed onto one or more compute units. Lastly, we present a generic engineering workflow which enables the proper use of the proposed architecture to construct trustworthy self-adaptive systems.

Funded by
the European Union

Contents

Funded by
the European Union

1 Introduction

This explanatory document provides an overview of the software architecture of the RoboSAPIENS Adaptive Platform. The RoboSAPIENS Adaptive Platform is a software platform for constructing trustworthy self-adaptive robotics applications, defined by means of requirements and software specifications. It is highly flexible in its deployment, providing more degrees of freedom for solution design.

1.1 Objectives

This document provides both an architecture description of the RoboSAPIENS Adaptive Platform and a high-level development workflow, describing how to use the RoboSAPIENS Adaptive Platform. The architecture description of the RoboSAPIENS Adaptive Platform, closely aligned to ISO/IEC 42010 (*Systems and software engineering — Architecture description*) [EH09] has the following main objectives:

- Identify the **stakeholders** of the RoboSAPIENS Adaptive Platform and their concerns.
- Identify the **system scope** and provide **overview information** of the RoboSAPIENS Adaptive Platform.
- Provide a **building block view** to identify the architecture building blocks and their corresponding interfaces.
- Provide a **runtime view** to specify the way the architecture building blocks should interact with each other to achieve trustworthy self-adaptive behavior.
- Provide a **deployment view** on the architecture, providing exemplary deployment scenarios for an RoboSAPIENS Adaptive Platform.

The description of the high-level, generic development workflow has the following main objectives:

- Provide the **generic engineering activities** involved to construct trustworthy adaptive applications on top of the presented RoboSAPIENS Adaptive Platform.

1.2 Scope

This explanatory document applies to the RoboSAPIENS Adaptive Platform. It describes the original architectural design of the RoboSAPIENS Adaptive Platform including details on how the building blocks should interact with each other in order to achieve trustworthy self-adaptive behavior. It is recommended to get an overview of the RoboSAPIENS Adaptive Platform for all members of the RoboSAPIENS team, both application developers and use case providers.

It is important to note that this is the first iteration of the RoboSAPIENS Adaptive Platform software architecture, which currently focuses on non-distributed systems. As the platform continues to evolve, it is expected that new requirements will emerge, necessitating the extension of the architecture to support proper distributed execution. Future developments, including any architectural changes that

Funded by
the European Union

accommodate distributed systems, will be addressed in subsequent releases or iterations of this document. Additionally, trustworthiness will remain a critical focus during this evolution.

1.3 Document Structure

This document is organized as follows: In section 2 overviews and goals of the Robosapiens adaptive platform(RA) are discussed. Section 3 describes the scope and context of the document. In section 4 the solution strategy which is proposed is explained. Section 5 offers a comprehensive overview of the architectural components and their respective interfaces that make up the RA software architecture. Section 6 specifies the way the architecture components should interact to achieve trustworthy self-adaptive behavior. Section 7 provides exemplary deployment scenarios for the RA platform and section 8 provides a generic engineering workflow to develop trustworthy adaptive systems on top of the presented RA Platform. Finally section 9 lists the risks that are associated with the overall architecture of the RA platform.

Funded by
the European Union

2 Overview and Goals

2.1 Requirements overview

This section provides an overview of the basic requirements for the RoboSAPIENS Adaptive Platform that shape its architecture.The corresponding requirement identifiers are provided in round brackets. An overview of the architecture requirements is listed in Appendix, 2, including a link to the general RoboSAPIENS requirements, documented in D7.3 - Requirements Report. Some requirements are derived from the RoboSAPIENS trustworthiness definition (Appendix 10). :

Support of state-of-the-art technology
The RoboSAPIENS Adaptive Platform aims to support both structural and environmental changes including those(changes) introduced by humans. This will be addressed in a genuine way where the safety observation and validation are split into different layers deployed both on the robot and its runtime Digital Twin using cutting-edge technologies such as prediction, online monitoring and transfer learning. Therefore, the RoboSAPIENS Adaptive Platform shall support both resource-constraint platforms and high performance computing platforms, both local (edge node) and remote (cloud) (ID_RAP_01). In addition, the RoboSAPIENS Adaptive Platform shall support online learning, a technique where the (Digital Twin) model continues to incrementally learn/update its parameters as soon as the new data points are available. (ID_RAP_02).

Software update and configuration
The RoboSAPIENS Adaptive Platform shall support a flexible (configuration) data and software update. Hereby, RoboSAPIENS Adaptive Platform shall support up- and download of such update packages (ID_RAP_03) and change of communication and application software. These aspects shall be supported and executable at runtime for both the managed and managing system.

Unified architecture description
A unified way to describe adaptive systems shall be provided, which enables to describe (i) the logical architecture, the adaptive software system structure (ID_RAP_04), (ii) the physical architecture, the compute unit structure (ID_RAP_05), and (iii) the deployment and allocation of the logical architecture components on one or more compute units (ID_RAP_06), and shall provide means to describe inter- and intra-device interfaces of the entire system (ID_RAP_07).

Communication protocols
The RoboSAPIENS Adaptive Platform shall support multiple standardized communication protocols, for example, but not limited to, and protocols compatible with communication for (bi-directional) inter- and/or intra-device communication with different network topologies (ID_RAP_08).

Diagnostics
The RoboSAPIENS Adaptive Platform shall provide diagnostics means during run-

Funded by
the European Union

time, providing explainability on the robots data, actions and decisions in a transparent way (ID_RAP_09).

2.2 Stakeholders

This section lists the stakeholders of the RoboSAPIENS Adaptive Platform architecture and their main expectations regarding this architecture description document.

Role	Expectations
Project leader	Overview of technical risks in the RoboSAPIENS Adaptive Platform.
System architect	Overview of the building blocks of the RoboSAPIENS Adaptive Platform, their purpose, functionality and runtime specification. Overview of the degrees of freedom for deployment scenarios of the RoboSAPIENS Adaptive Platform.
Application engineer	Overview of the building blocks of the RoboSAPIENS Adaptive Platform, their purpose, functionality and runtime specification.
Verification engineer	Overview of the building blocks of the RoboSAPIENS Adaptive Platform, their purpose, functionality and runtime specification and the deployment solution of the realized RoboSAPIENS Adaptive Platform.

3 System Scope and Context

This chapter describes the scope and context of this document, positioning it within the RoboSAPIENS project. Figure 1 shows the envisioned high-level RoboSAPIENS architecture for constructing trustworthy self-adaptive robotics applications.

Funded by
the European Union

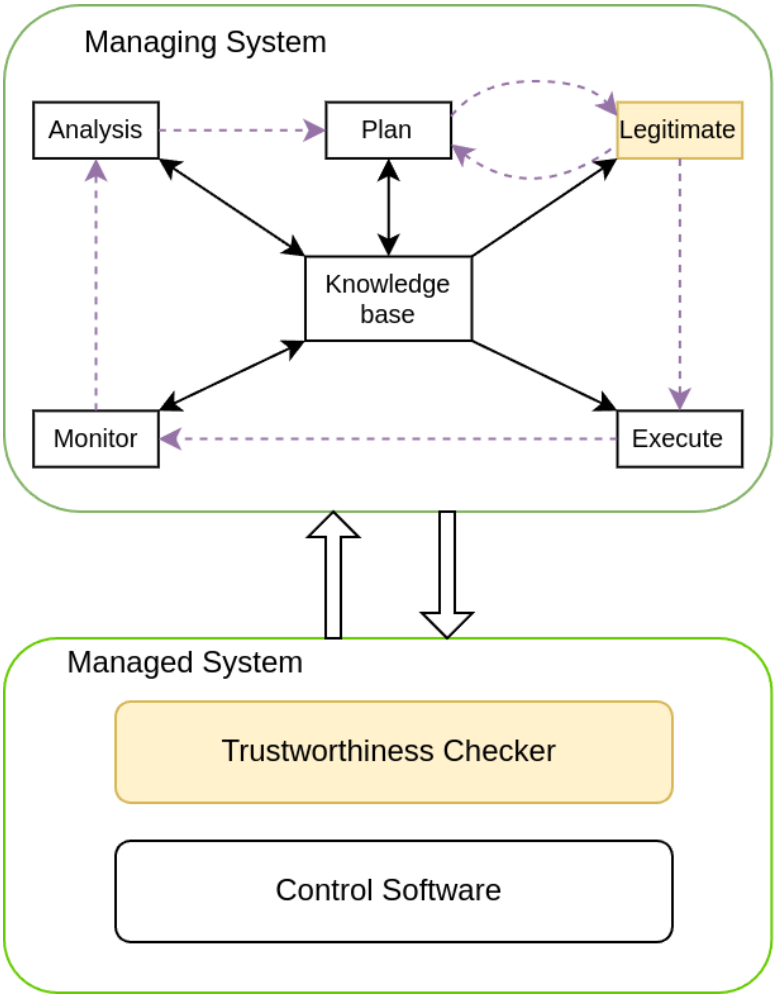

Figure 1: RoboSAPIENS approach for trustworthy self-adaptive robot applications.

3.1 Managed system

The managed system is the target of the adaptation process. It is the system that performs the core functional tasks and requires monitoring and management to ensure adaptability to changing conditions. The managing system continuously monitors the state of the managed system. The managing system collects information from the managed system using probe and sends diagnosis/adaptation actions using the effector component.

3.1.1 Control Software

Each robotic platform has a builtin control software responsible for low-level control functions such as speed regulation and trajectory tracking. This software can interact with external components through different communication protocols. Additionally, it manages the operation of sensors and drivers within the system.. The choice of control algorithms used in the software can vary depending on the application. For instance, a robot designed for industrial tasks might require precise trajectory control, while a robot used for research purposes may prioritize flexibility and adaptability. The control software ensures that the robot operates optimally in its intended environment, achieving desired performance objectives.

Funded by
the European Union

3.1.2 Trustworthiness checker

The MAPLE-K is improved by introducing a trustworthiness checker. This component plays a crucial role in ensuring the reliability and security of the self-adaptation process. The trustworthiness checker analyses the planned actions based on predefined rules. These rules can be application-independent, such as component execution time or data throughput constraints, or can be application dependent, such as robot velocity constraints or robot-specific safety constraints. By evaluating these aspects, the trustworthiness checker acts as a safeguard against potentially harmful or unreliable adaptations.

The trustworthiness checker also verifies the data collected by the Monitor component from the managed system. This ensures that the knowledge base is not polluted with inaccurate or manipulated data. Techniques like data validation and anomaly detection can be used for this purpose.

3.2 Managing system

The managing system is responsible for overseeing the managed system. It implements the MAPLE-K loop, which stands for Monitoring, Analysis, Planning, Legitimate, Execution, and Knowledge. It is an adaptation of the traditional MAPE-K loop [IW15], a well-recognised engineering approach for self-adaptive systems. The MAPLE-K adds trustworthiness, by introducing 2 main components, indicated in yellow, the *legitimate* and *trustworthiness checker* components. Adding those components on top of the traditional MAPE-K loop will ensure that the managed system operates within desired parameters and adapts to changes safely.

Key components and responsibilities of the managing system include:

- **Monitoring:** Collects data from the managed system, such as sensor data and managed system states.
- **Analysis:** Interprets the collected data to diagnose issues and detect anomalies. This step often involves sophisticated algorithms, such as machine learning or statistical analysis.
- **Planning:** Develops strategies for adapting the managed system based on the analysis. This could include re-configuring settings or adaptation actions.
- **Legitimate:** Checks the action created by the plan component based on the predefined legitimate rules. The rules can involve checking both the managed system and its surroundings to ensure the planned action is trustworthy.
- **Execution:** Implements the planned action by interacting with the managed system. This involves executing an action and ensuring its correct execution even under changing conditions.
- **Knowledge:** Maintains a repository of information that supports the other components of the MAPLE-K loop, such as sensor data, components status, models, and rules.

Funded by
the European Union

3.3 Dependencies

Dependencies vary depending on different deployment environments and use cases. For instance, the development frameworks used in robots can vary, such as ROS2 [BGGL23] or Python-based systems. Additionally, the hardware components may differ based on different use cases. We will explore dependencies in future iterations of the document.

3.4 External systems

In the context of a MAPE-K loop architecture, several external systems may interact with both the managed system and the managing system. These external systems can impact how the self-adaptive system operates and responds to changes. These external systems like browsers, client applications, Log interfaces and so on. Figure2 presents an example of an external system integrated with the RoboSAPIENS Adaptive Platform: an InfluxDB dashboard [NYZ17], which is widely utilized for time series data analysis and visualization. In this context, it is used to monitor and display the execution of the MAPLE-K loop.

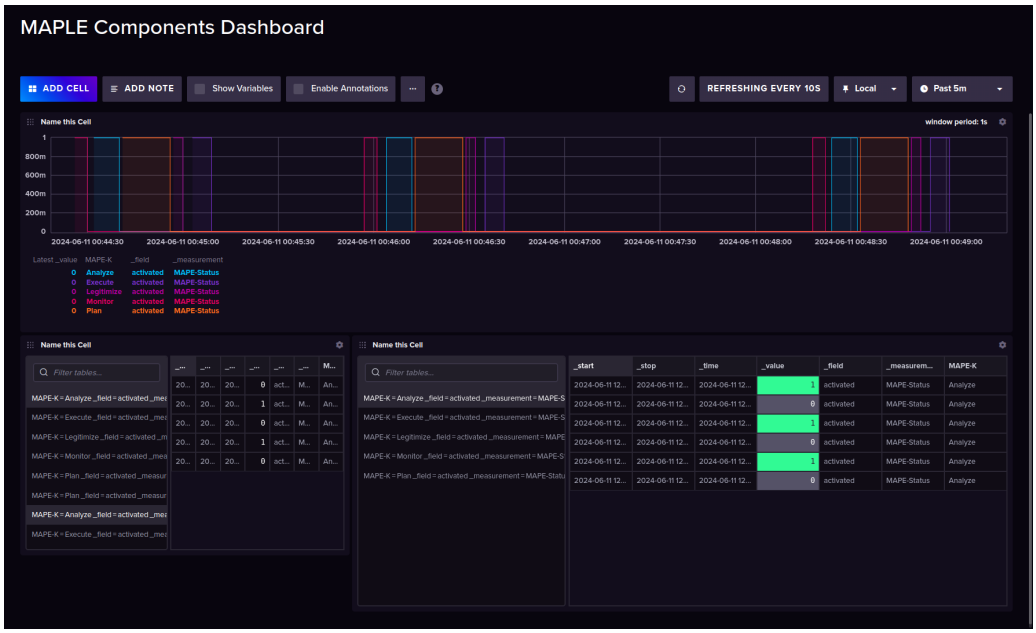

Figure 2: Monitoring the MAPLE-K Timing on InfluxDB application

Funded by
the European Union

4 Solution Strategy

The RoboSAPIENS Adaptive Platform is a reference architecture for adaptive robots. It is not a concrete implementation. We leave a degree of freedom to its implementers by defining requirements and software specifications that need to be fulfilled without specifying how.

4.1 Architectural Approach

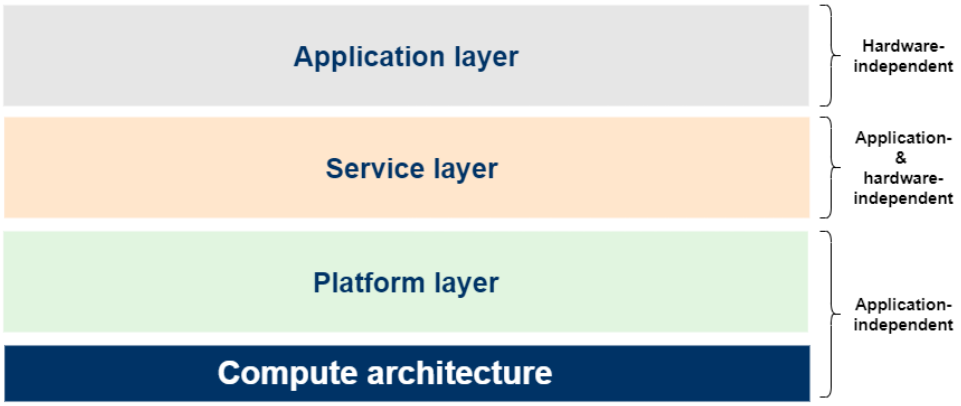

Figure 3: Layered architecture approach.

A high-level architectural design approach is delineating the separation of concerns into distinct layers to enhance modularity and manageability, as shown in Figure 3. We propose the following layers:

- **Application layer:** contains the user-specified application components to construct the behavior of the adaptive system, using the provided core services and functionalities of the RoboSAPIENS Adaptive Platform.
- **Service layer:** handling the application-independent core services and functionalities provided by the RoboSAPIENS Adaptive Platform, which enable the execution of trustworthy self-adaptive robotics applications.
- **Platform layer:** handling the core services and functionalities, including essential system services, middleware, and other foundational components, necessary for the operation and support both the *service* and *application* layers.
- **Compute architecture:** contains the physical and virtual computing resources. It encompasses the hardware and other low-level system components that provide the necessary computational power and infrastructure for the entire system, e.g. the robot controller, cloud infrastructure, etc.

Funded by
the European Union

Figure 4 illustrates a detailed view of the layered architecture, applied to a typical application setup where the managed system (A) and the managing system (B) operate on different computing architectures. For instance, this occurs when a robot (managed system) is wirelessly connected to a laptop or server (managing system) running the MAPLE-K loop. The figure clearly identifies which components are assigned to each architectural layer, specifying whether the component is independent of the application and/or hardware. It is important to note that the managed and managing systems can also run on the same computing architecture. In Section 5, these components will be discussed in detail.

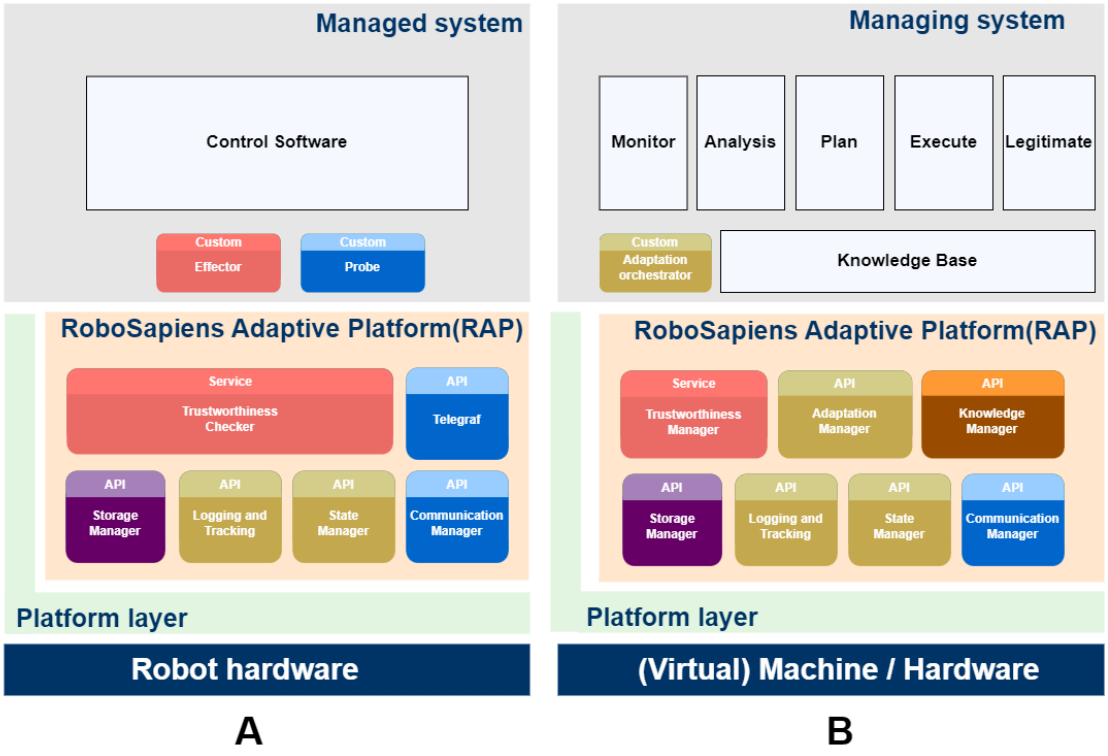

Figure 4: Detailed view on layered architecture.

4.2 Reference implementation

There is a reference implementation of the RoboSAPIENS Adaptive Platform available, demonstrating a possible instantiation of the RoboSAPIENS Adaptive Platform reference architecture. Python is the programming language of choice for the reference implementation, enabling fast and easy prototyping. Depending on the use case and performance needs, the Python-based reference implementation could (partially) be transformed into a C++-based implementation. This conversion will increase performance properties of the reference implementation, required for safety- and performance critical, real-time applications (such as typical adaptive applications). Please note that the reference implementation is not yet complete, as it does only contain placeholder software for the trustworthiness checker and legitimate components. Details on the validation of the reference implementation are discussed in subsequent section.

Funded by
the European Union

4.3 Verification and Validation

The RoboSAPIENS Adaptive Platform uses a dedicated reference implementation of the architecture to validate the requirements and to verify the (still abstract) architecture design imposed by the architecture specification. The reference implementation is currently verified on the following demonstrators:

- **Hotbox example:** simulated application, control software of managed system is regulating the inside temperature of a box, regardless of the content inside. This single-device example is used to verify the activation and interfaces of all components, except for trustworthiness checker and legitimate components.
- **Turtlebot 4 simulator:** simulated application, control software of the virtual Turtlebot 4 is executing a navigation task. This single-device example, used to prototype and verify the Python-based client library to prepare for the distributed "Hello world scenario".
- **Hello world scenario 1 (Lidar occlusion) :** real, distrubuted (academic) application, control software of the real Turtlebot 4 is executing a navigation task. This multi-device example is used to verify the remote connection and actuation of a real robot.

Please note that the reference implementation is not yet complete, therefore the verification is also not complete. Different examples are currently used to verify parts of the RoboSAPIENS Adaptive Platform. In the future, we plan to extend the verification by applying the architecture on industry-oriented case studies.

Funded by
the European Union

5 Building Blocks view

5.1 Overview

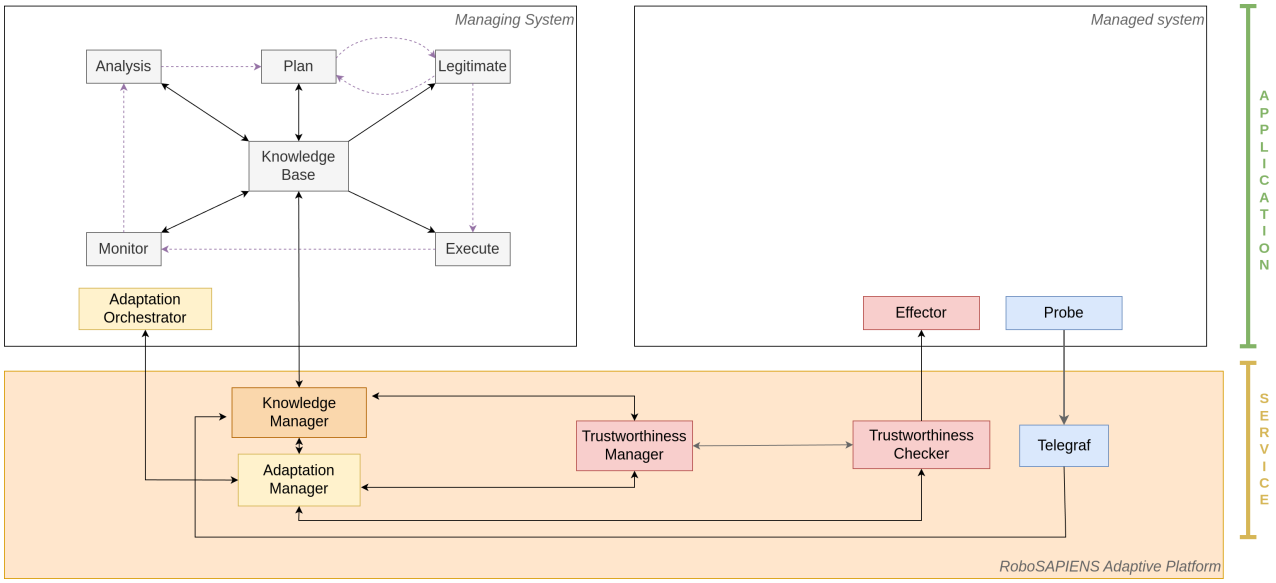

Figure 5: Overview of the RA platform components and their high-level interfaces.

Figure 5 provides a further detailed view on the layered architecture, identifying not only the main building blocks of the RoboSAPIENS Adaptive Platform, but also identifying the main interfaces between them. The colors of the architecture components does match the different categories, as shown in Figure 6.

Funded by
the European Union

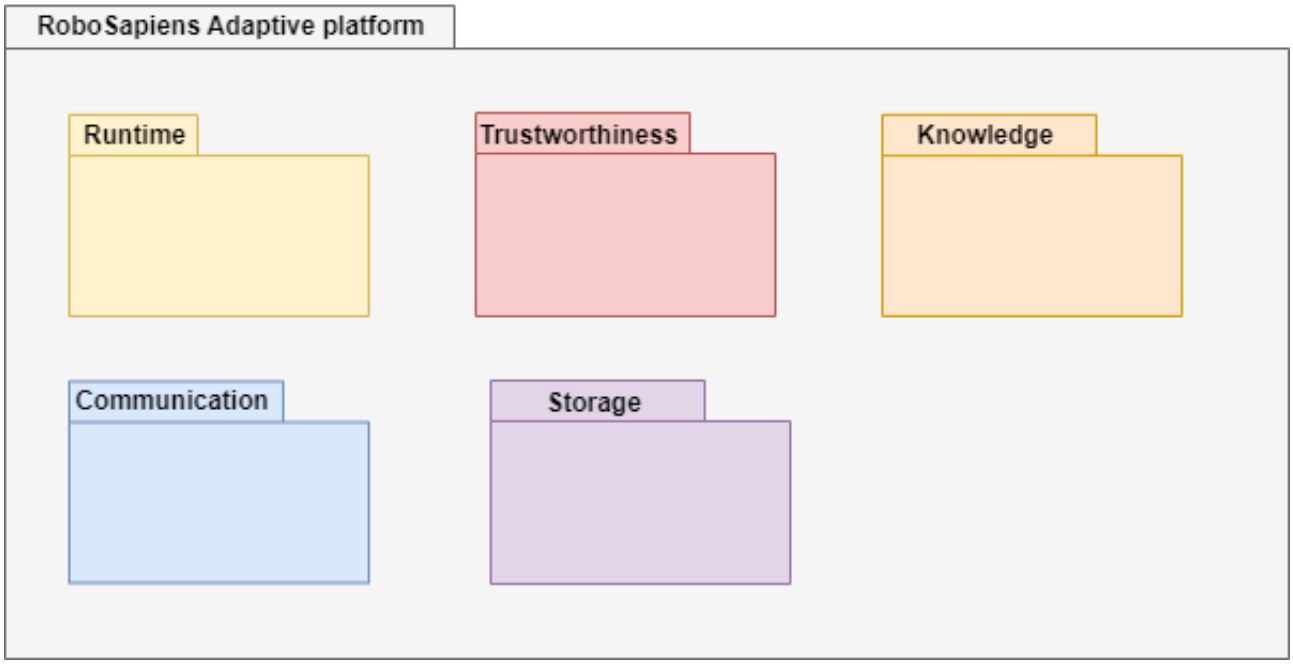

Figure 6: Overview of the RoboSapiens Adaptive Platform and its building block categories.

In subsequent sections, the building blocks are discussed in more detail. For the description of the architecture building blocks, the following structure is used:

- **Identification and responsibilities:** Describing the primary roles and responsibilities of the building block within the RoboSAPIENS Adaptive Platform.
- **Operation modes:** Describing the different operation modes the building block can function in. These modes may vary depending on the system state, environmental conditions, or user interactions. This is reflected by means of a state machine, modeled using itemis CREATE [ite], formerly known as YAKINDU [Mue11]. In general, all building blocks operates through the same distinct modes. First, in *instantiate mode*, the component is created in memory, allocating resources and preparing for use. In *configuration mode*, it is customized with the necessary parameters, followed by *initialization mode*, where dependencies and processes are set up to transition the component into a ready state. Once initialized, the component enters *initialized mode*, where it actively performs its core tasks. Finally, in *terminate mode*, the component shuts down, deallocating resources and completing any final operations before exiting. These modes ensure efficient and orderly management of the component's lifecycle.
- **Interfaces:** Describing how the building block communicates and interacts with other blocks in the system.

Funded by
the European Union

5.2 Runtime

Collection of building blocks providing services that enable the correct and transparent execution behavior of the self-adaptive application.

5.2.1 Adaptation Manager

Name:	Adaptation Manager
Short name:	am
Category:	Runtime
Deamon-based:	Yes
Responsibilities:	The `Adaptation Manager` is responsible for triggering the execution of the *managing system level trustworthiness checks*, via the `Trustworthiness Manager`, and when trustworthy, registering the adaptation action to the `Trustworthiness Checker`, ready to be executed by the managed system.

5.2.1.1 Operation Modes

The `Adaptation Manager` operation modes are shown in Figure 7. After instantiation, configuration, and initialization, the `Adaptation Manager` performs its core task, namely triggering the trustworthiness checks and depending on the trustworthiness assessment, triggering the adaptation action or drop the action.

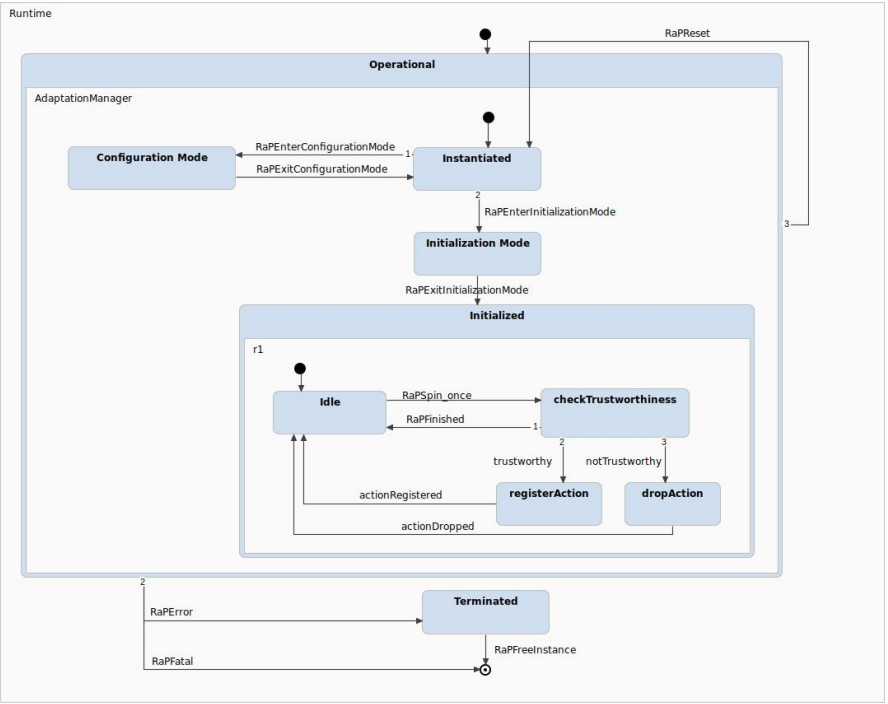

Figure 7: Operation modes of the `Adaptation Manager` component.

5.2.1.2 Interfaces

The `Adaptation Manager` interfaces are shown in Figure 8 and detailed the accompanied table.

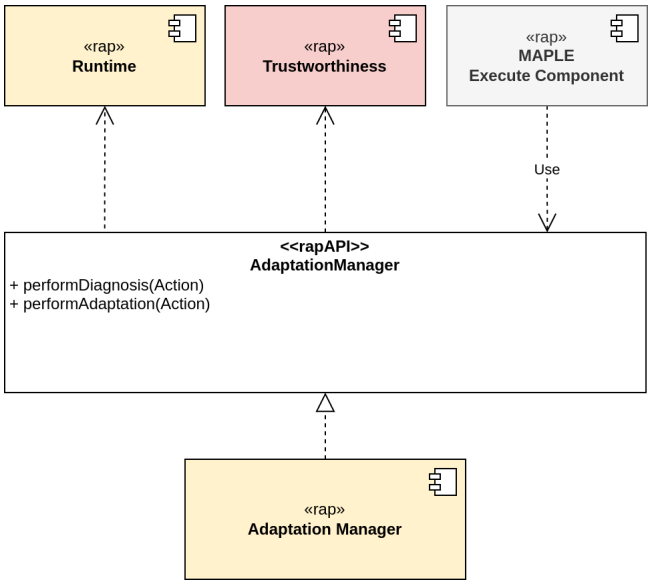

Figure 8: Interfaces for `Adaptation Manager`.

Name:	Adaptation Manager	
Technology:	Native API	
Usage:	Managing system accessible API	
Description:	This interface represents the adaptation handling mechanism. It provides functions to trigger an action, adaptation or diagnosis type, from the *execute stage* of the MAPLE-K.	
Operations:	performDiagnosis	Triggering a diagnose action.
	performAdaptation	Triggering an adaptation action.

Funded by
the European Union

5.2.2 Adaptation Orchestrator

Name:	Adaptation Orchestrator
Short name:	ao
Category:	Runtime
Deamon-based:	Yes
Responsibilities:	The `Adaptation Orchestrator` is responsible to orchestrate the adaptation, It continuously checks that if an anomaly detected and triggers the plan, legitimate and execute components.

5.2.2.1 Operation Modes

The `Adaptation Orchestrator` operation modes are shown in Figure 9. After instantiation, configuration, and initialization, the `Adaptation Orchestrator` performs its core task, as illustrated in Figure 5.2.2, the adaptation orchestrator operates based on a state machine that orchestrates the maple k loop. It continuously monitors the state, and when the anomaly detection analyzing component is triggered and the anomaly has been analyzed, then the plan spinonce function is activated. Subsequently, after the plan has been generated, the legitimate component evaluates whether the plan is legitimized, and then the planning action is executed.

Funded by
the European Union

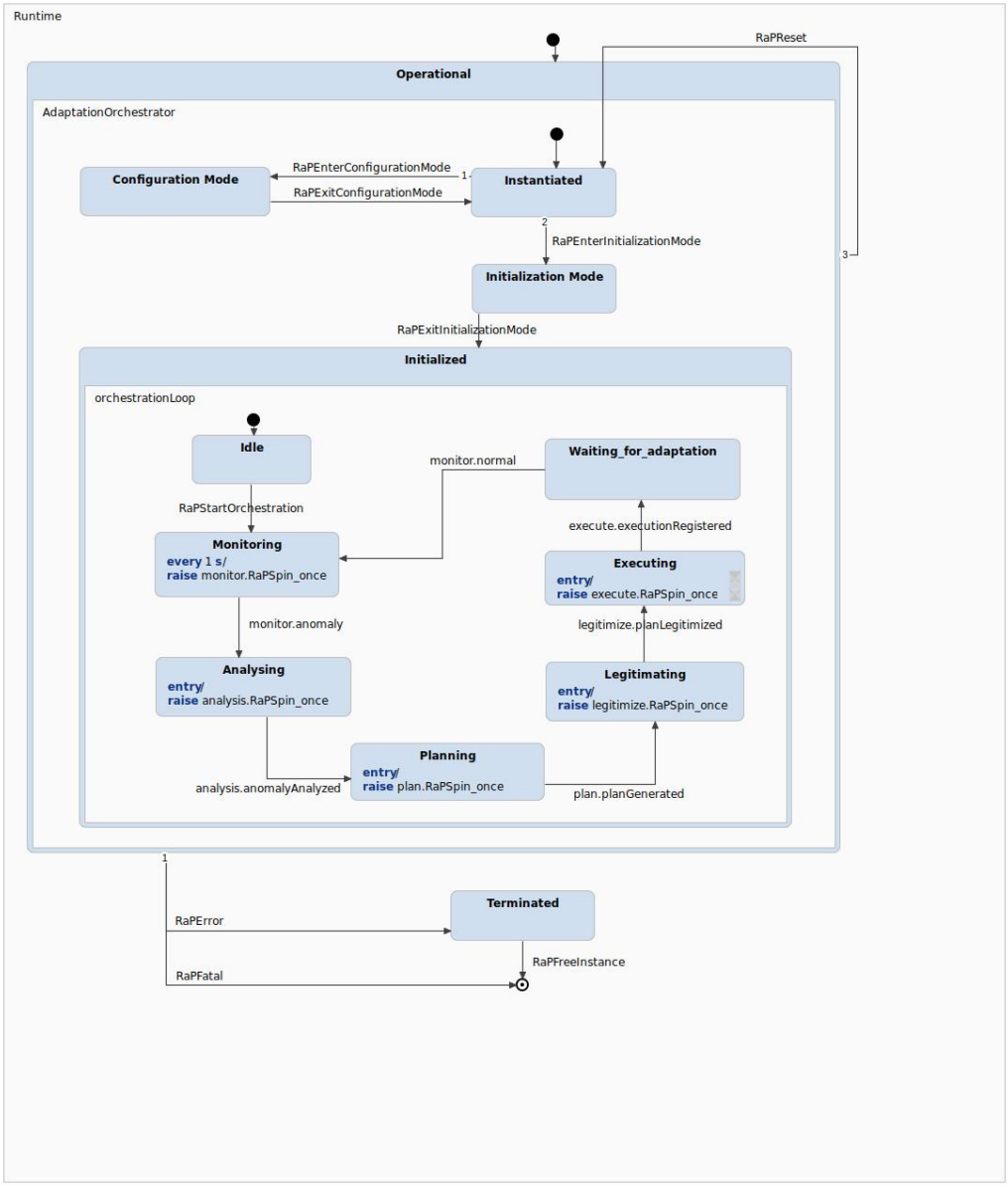

Figure 9: Operation modes of the `Adaptation Orchestrator` component.

5.2.2.2 Interfaces

The `Adaptation Orchestrator` interfaces are shown in Figure 8 and detailed the accompanied table.

Funded by
the European Union

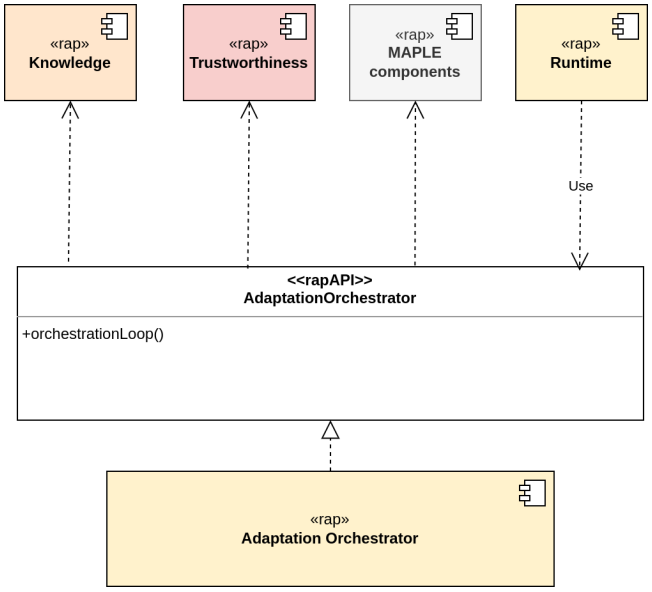

Figure 10: Interfaces for `Adaptation Orchestrator`.

Name:	Adaptation Orchestrator	
Technology:	Native API	
Usage:	Orchestrate the MAPLE loop	
Description:	This interface represents the adaptation orchestration mechanism. It works like the adaptation state machine. It is replaceable with code which is generated by state machine creators.	
Operations:	orchestrationLoop	Orchestrate the adaptation process.

Funded by
the European Union

5.2.3 State Manager

Name:	State Manager
Short name:	sm
Category:	Runtime
Deamon-based:	Yes
Responsibilities:	`State Manager` monitors and tracks the current state of the adaptive layer components (MAPLE-K) as well as the current state of the managed system, making them available to other RoboSapiens Adaptive Platform components.

5.2.3.1 Operation modes

The `State Manager` operation modes are shown in Figure 11. After instantiation, configuration, and initialization, the `State Manager` performs its core task, namely tracking the state of all managing and managed system components.

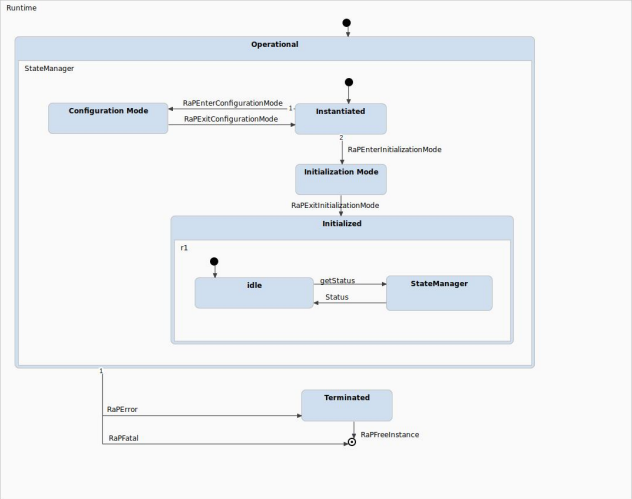

Figure 11: Operation modes of the `State Manager` component.

5.2.3.2 Interfaces

The `State Manager` interfaces are shown in Figure 12 and detailed the accompanied table.

Funded by
the European Union

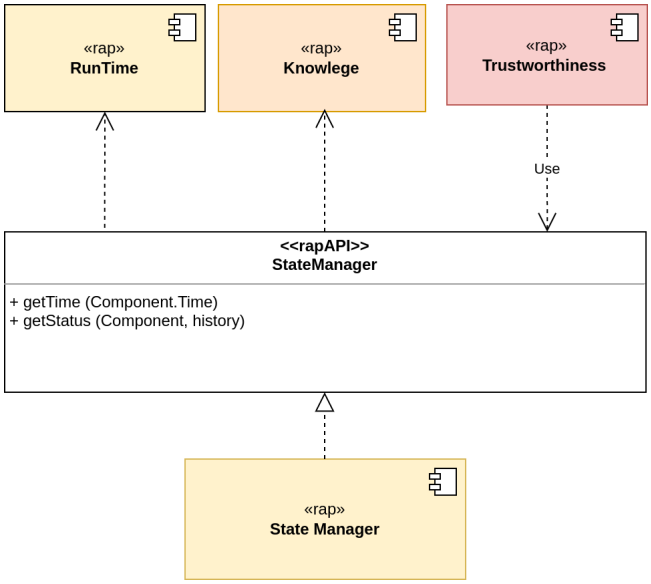

Figure 12: Interfaces for `State Manager`.

Name:	State Manager	
Technology:	Native API	
Usage:	Access component runtime status	
Description:	This interface represents the State handling mechanism `Knowledge Manager`	
Operations:	getTime	Returns the latest execution time Of the component
	getStatus	Returns the timing history or the MAPLE-K components

Funded by
the European Union

5.2.4 Logging and Tracking

Name:	Logging and Tracking
Short name:	log
Category:	Runtime
Deamon-based:	No
Responsibilities:	`Logging and Tracking` provides functionality to build and log messages of different severity. The log module can be configured to forward log messages to different sinks, e.g., to a network, the console or to non-volatile storage.

5.2.4.1 Operation modes

The `Logging and Tracking` operation modes are shown in Figure 13. After instantiation, configuration, and initialization, the `Logging and Tracking` performs its core task, processing log messages coming from different managing and managed system components.

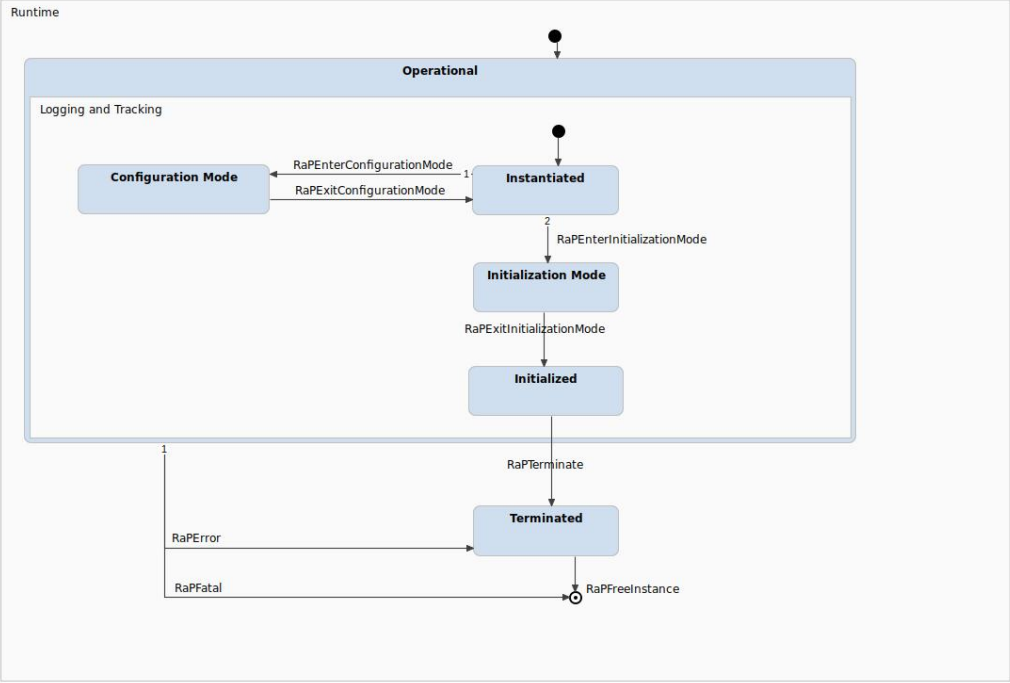

Figure 13: Operation modes of the Logger component.

5.2.4.2 Interfaces

The `Logging and Tracking` interfaces are shown in Figure 14 and detailed the accompanied table.

Funded by
the European Union

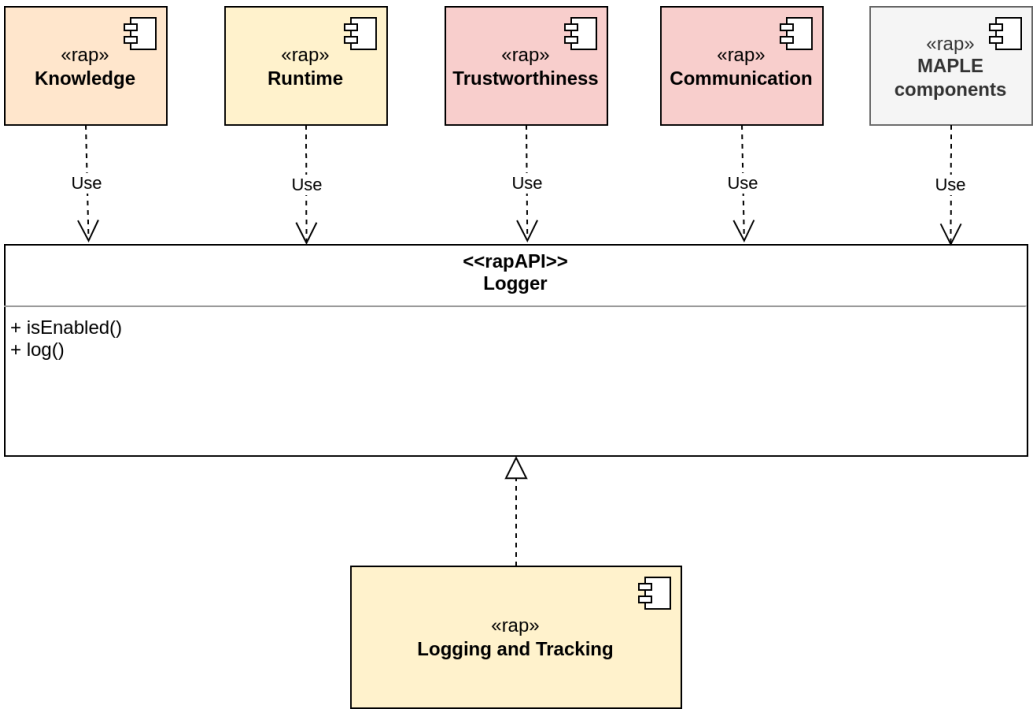

Figure 14: Interfaces for `Logging and Tracking`.

Name:	Logging and Tracking	
Technology:	Native API	
Usage:	Public API	
Description:	This interface represents a logger context. It provides functions to log using different severity levels.	
Operations:	isEnabled	Check if the logger is enabled.
	log	Logs a message.

Funded by
the European Union

5.2.5 Core

Name:	Core
Short name:	core
Category:	Runtime
Deamon-based:	No
Responsibilities:	`Core` provides functionality for initialization and de-initialization of the RoboSapiens Adaptive Platform as well as termination of Processes.

5.2.5.1 Operation modes

The `Core` operation modes are shown in Figure 15. After instantiation, configuration, and initialization, the `Core` performs its core task, The core task is always running while the RoboSapiens Adaptive Platform is active.

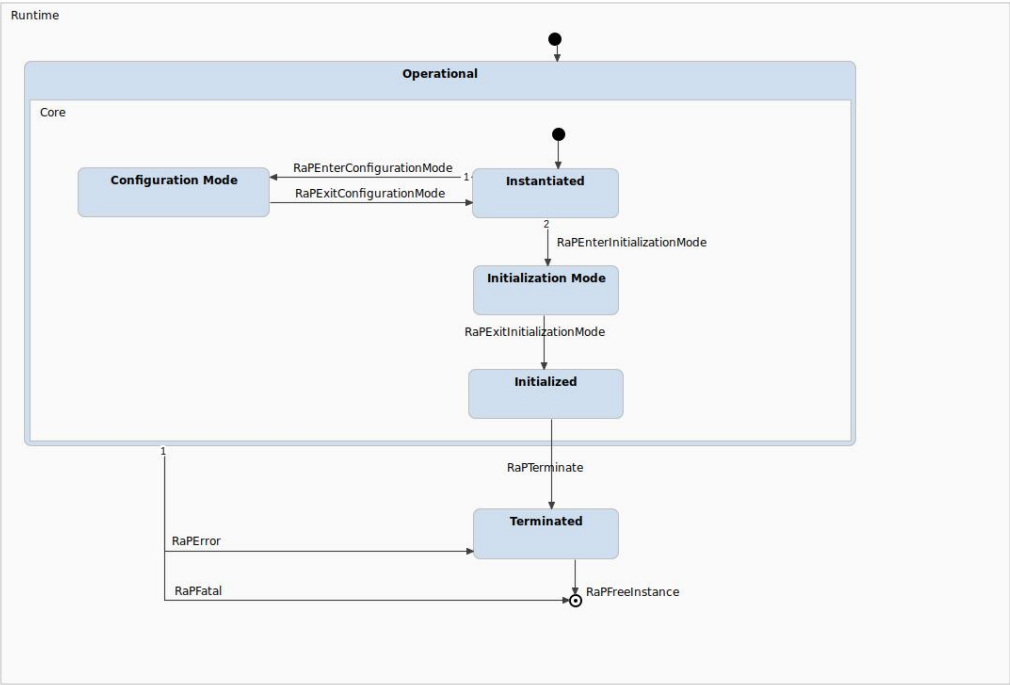

Figure 15: Operation modes of the `Core` component.

5.2.5.2 Interfaces

The Core interfaces are shown in Figure 16 and detailed the accompanied table.

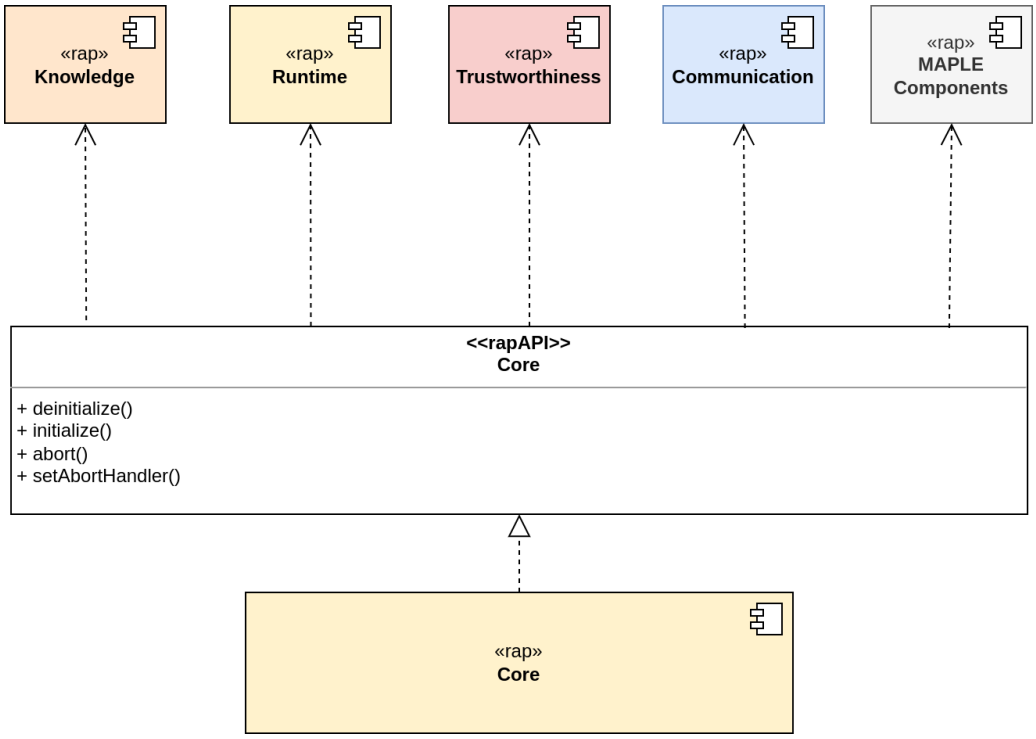

Figure 16: Interfaces of `Core`.

Funded by
the European Union

Name:	Core	
Technology:	Native API	
Usage:	Public API	
Description:	This interface provides global initialization and shutdown functions that initialize respectively deinitialize data structures and threads of the RoboSapiens Adaptive Platform.	
Operations:	deinitialize	Destroy all data structures and threads of the RoboSapiens Adaptive Platform. After this call, no interaction with the RoboSapiens Adaptive Platform is possible.
	initialize	Initialize data structures and threads of the RoboSapiens Adaptive Platform. Prior to this call, no interaction with the RoboSapiens Adaptive Platform is possible.
	abort	Abort the current process.
	setAbortHandler	Set a custom global abort handler function.

Funded by
the European Union

5.3 Trustworthiness

Collection of building blocks providing services to ensure trustworthy operation of the self-adaptive application.

5.3.1 Trustworthiness Manager

Name:	Trustworthiness Manager
Short name:	tm
Category:	Trustworthiness
Deamon-based:	Yes
Responsibilities:	The `Trustworthiness Manager` is responsible for continuously monitoring and gathering all the data of the MAPLE-K elements (adaptation layer), based on the current system state, provided by the `State Manager`, and the system knowledge, provided by the `Knowledge Manager`which is required for the actual on-device `Trustworthiness Checker`. The `Trustworthiness Manager` will also actively monitor the `Adaptation Manager` triggers, enabling trustworthy triggering of adaptation or diagnosis routines via the `Trustworthiness Checker` components.

5.3.1.1 Operation modes

The `Trustworthiness Manager` operation modes are shown in Figure 17. After instantiation, configuration, and initialization, the `Trustworthiness Manager` performs its core task, namely processing incoming trustworthiness checking requests.

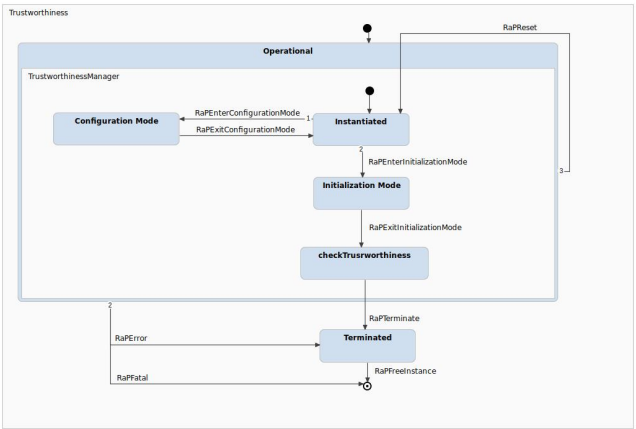

Figure 17: Operation modes of the `Trustworthiness Manager` component.

5.3.1.2 Interfaces

The `Trustworthiness Manager` interfaces are shown in Figure 18 and detailed the accompanied table.

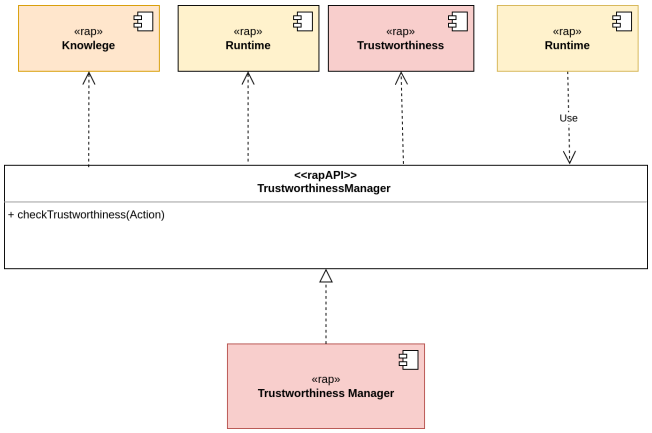

Figure 18: Interfaces for `Trustworthiness Manager`.

Name:	Trustworthiness Manager	
Technology:	Native API	
Usage:	Check trustworthiness of an action	
Description:	This interface represents the trustworthiness handling mechanism. It provides a function to monitor the managing system's status and send them to the `Trustworthiness Checker`	
Operations:	checkTrustworthiness	Return true when the action is trustworthy

5.3.2 Trustworthiness Checker

Name:	Trustworthiness Checker
Short name:	tc
Category:	Trustworthiness
Deamon-based:	Yes
Responsibilities:	The `Trustworthiness Checker` is responsible for continuously monitoring the trustworthiness of the managed system, based on the current managed system state, provided by the `State Manager`. This includes active monitoring of the normal operation and adaptation modes.

Funded by
the European Union

5.3.2.1 Operation modes

The `Trustworthiness Checker` operation modes are shown in Figure 19. After instantiation, configuration, and initialization, the `Trustworthiness Checker` performs its core task, It waits until the trustworthiness check request is called by the adaptation manager, and it checks whether the action is trustworthy to be registered.

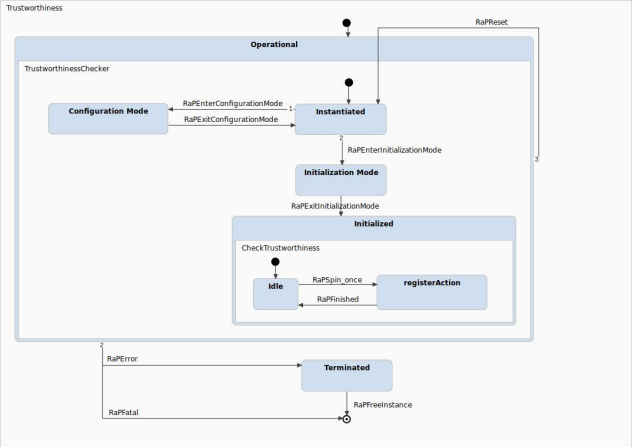

Figure 19: Operation modes of the `Trustworthiness Checker` component.

5.3.2.2 Interfaces

The `Trustworthiness Checker` interfaces are shown in Figure 20 and detailed the accompanied table.

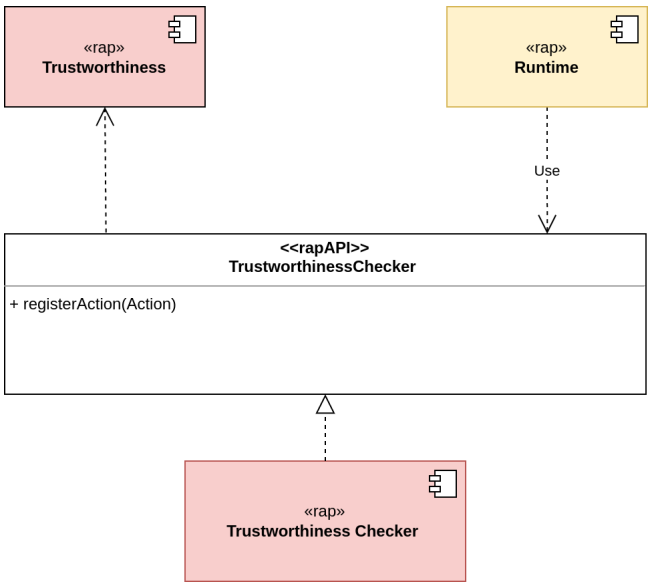

Figure 20: Interfaces for `Trustworthiness Checker`.

Funded by
the European Union

Name:	Trustworthiness Checker	
Technology:	Native API	
Usage:	Check managed system operation mode	
Description:	This function is responsible for check whether the action can be executed based on current status of the managed system	
Operations:	registerAction	True if action is applicable based on managed system operation mode.

5.4 Knowledge

Collection of building blocks providing services for the handling of (application-specific) knowledge, collected from and/or generated by the managed and/or managing system components.

5.4.1 Knowledge Manager

Name:	Knowledge Manager
Short name:	km
Category:	Knowledge
Deamon-based:	No
Responsibilities:	`Knowledge Manager` is responsible for capturing the system's knowledge, provided by the `Telegraf` component, and making the system's knowledge available for the MAPLE (adaptive layer) elements, the `Adaptation Manager` component and the `Trustworthiness Manager`.

5.4.1.1 Operation modes

The `Knowledge Manager` operation modes are shown in Figure 21. After instantiation, configuration, and initialization, the `Knowledge Manager` is always running and callable from other components to make it possible to obtain and set data on the knowledge base.

Funded by
the European Union

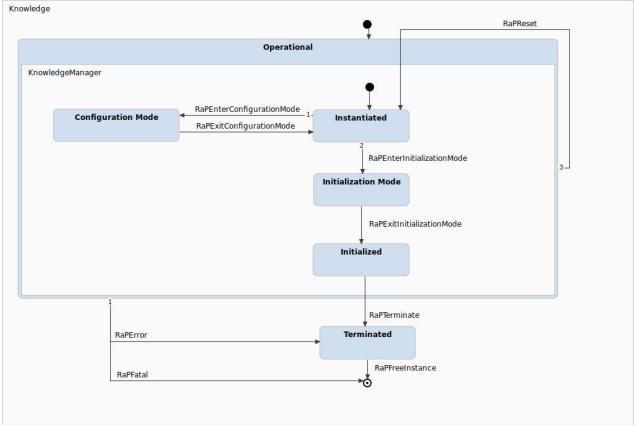

Figure 21: Operation modes of the `Knowledge Manager` component.

5.4.1.2 Interfaces

The `Knowledge Manager` interfaces are shown in Figure 22 and detailed the accompanied table.

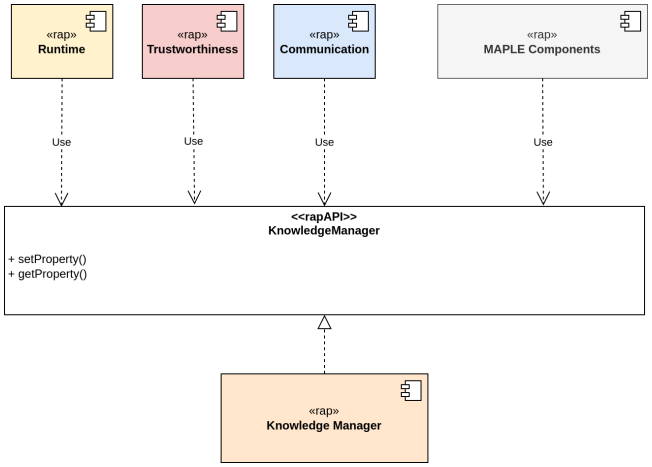

Figure 22: Interfaces for `Knowledge Manager`.

Name:	Knowledge Manager	
Technology:	Native API	
Usage:	Read and write data on the Knowledge base	
Description:	This interface represents the Knowledge management mechanism. It provides functions to read and write data on the knowledge base Data can be properties, MAPLE-K component status or actions	
Operations:	registerKnowledge	Write data on Knowledge base
	readKnowledge	Read data from knowledge base

Funded by
the European Union

5.5 Communication

Collection of building blocks providing services for the network and protocol independent communication between architecture building blocks, the managed and managing system.

5.5.1 Communication Manager

Name:	Communication Manager
Short name:	cm
Category:	Communication
Deamon-based:	Yes
Responsibilities:	`Communication Manager` is responsible for all levels of service-oriented and raw communication between the managed system application and adaptive layer elements in a distributed (real-time) embedded environment. That is, inter-process (component-level) communication and inter-machine communication. There are limited communication paths from and to the managed system, which are established only at design- and start-up-time, not at run-time.

5.5.1.1 Operation modes

The `Communication Manager` operation modes are shown in Figure 23. After instantiating, configuration, and initialization, the `Communication Manager` executes its function in response to requests for publishing messages or receiving message queues. It also keeps an ear out for orders to add publish and subscribe topics that other components are calling.

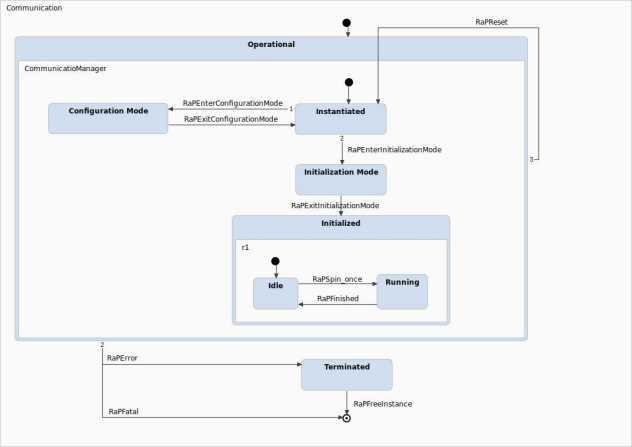

Figure 23: Operation modes of the `Communication Manager` component.

5.5.1.2 Interfaces

The `Communication Manager` interfaces are shown in Figure 24 and detailed the accompanied table.

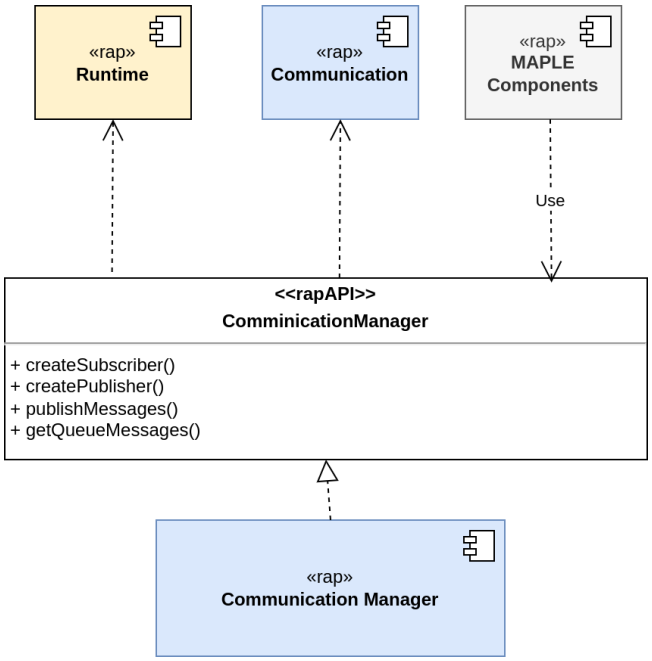

Figure 24: Interfaces for `Communication Manager`.

Name:	Communication Manager	
Technology:	Native API	
Usage:	Communication between components	
Description:	This interface represents the communication management mechanism. It provides functions to publish and subscribe on different topics Communication protocol may vary such MQTT, or local communications	
Operations:	createSubscriber	Add topic to subscribe topic list
	createPublisher	Add topic to publish topic list
	publishMessage	Publish the message
	getQueueMessage	Get the queued message in subscribed topic

Funded by
the European Union

5.5.2 Telegraf

Name:	Telegraf
Short name:	tg
Category:	Communication
Deamon-based:	Yes
Responsibilities:	`Telegraf` is responsible for collecting and sending all data, metrics, states and events from the managed system and it's running application, which is relevant to construct the system's knowledge, input for the `Knowledge Manager` component.

5.5.2.1 Operation modes

The `Telegraf` operation modes are shown in Figure 25. After instantiation, configuration, and initialization, the `Telegraf` performs its core task based on receive and send data events.

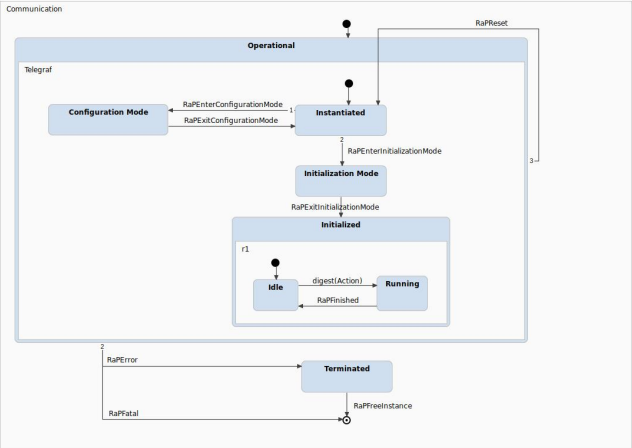

Figure 25: Operation modes of the `Telegraf` component.

5.5.2.2 Interfaces

The `Telegraf` interfaces are shown in Figure 26 and detailed the accompanied table.

Funded by
the European Union

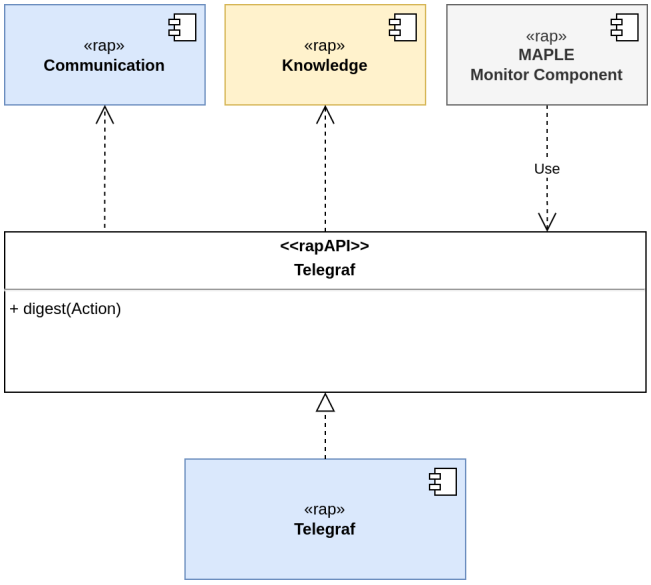

Figure 26: Interfaces for `Telegraf`.

Name:	Telegraf	
Technology:	Native API	
Usage:	Send data from managed system to the managing system	
Description:	This interface sends data from managed system It provides data for the `Knowledge Manager`	
Operations:	digest	Get managed system as input

5.6 Storage

Collection of building blocks providing services for storing (application-specific) knowledge to non-volatile memory.

5.6.1 Storage Manager

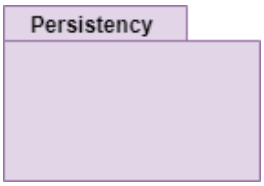

Figure 27: Overview of Storage and its building blocks.

Funded by
the European Union

Name:	Storage Manager
Short name:	stom
Category:	Storage
Deamon-based:	No
Responsibilities:	`Storage Manager` provides functionality to store and retrieve knowledge to/from non-volatile storage of a Machine.

5.6.1.1 Operation modes

The `Storage Manager` operation modes are shown in Figure 28. After instantiation, configuration, and initialization, the `Storage Manager` performs its core task based on any changes in components status, it runs continuously to monitor all components status.

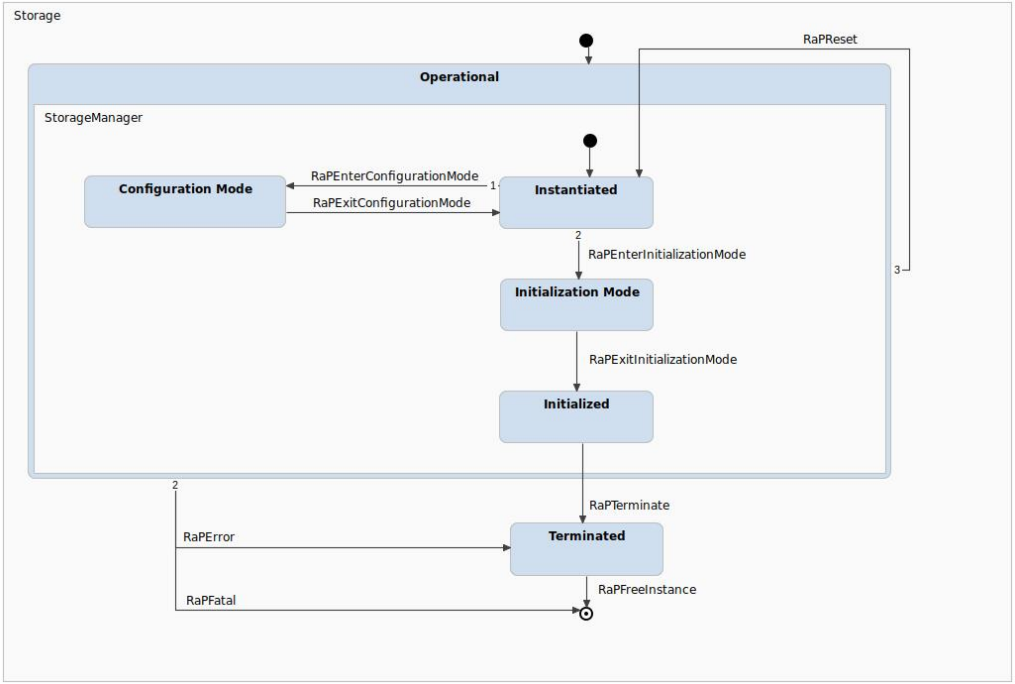

Figure 28: Operation modes of the `Storage Manager` component.

5.6.1.2 Interfaces

`Storage Manager` provides operations for handling knowledge data and recovery of persistent data of a process (cf. Figure 29).

Funded by
the European Union

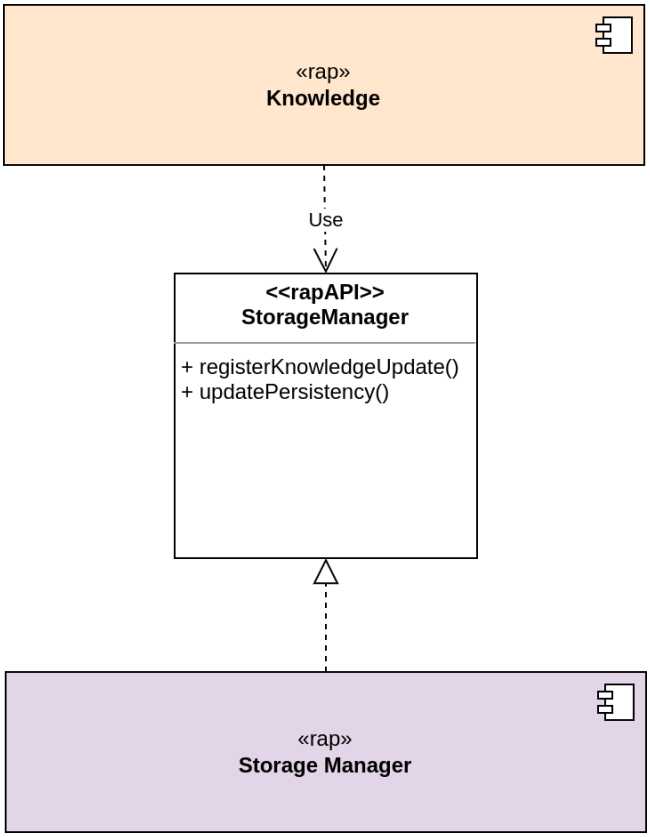

Figure 29: Interfaces for `Storage Manager`.

Name:	Storage Manager	
Technology:	Native API	
Usage:	Public API	
Description:	Provides functions to store/delete knowledge to/from non-volatile memory	
Operations:	registerKnowledgeUpdate	Registers knowledge data, prepared for pushing to non-volatile memory.
	updatePersistency	Push the registered knowledge data to non-volatime memory.

Funded by
the European Union

6 Runtime view

6.1 RA platform startup

During the startup of a machine the Operating System (OS) performs initialization steps in an implementation-specific way. These steps include starting any middleware related to the OS, including device-drivers and services handling low-level middleware. In addition, Core is started as the entry point of the RoboSAPIENS Adaptive Platform. Core controls the startup of the RoboSAPIENS Adaptive Platform internal architecture modules, e.g., Adaptation Manager. Each of these architecture modules has a dedicated startup sequence encoded within their respective operation mode state machines, which includes configuring each individual module to enable application-specific customization.

Besides the startup of the RoboSAPIENS internal architecture modules, we must also ensure correct startup of the (remote) endpoint modules, running outside the RoboSAPIENS Adaptive Platform internal architecture, such as the managed system (probes, effectors) or remote compute units (cloud/edge nodes). Facilitating correct node startup begins with the use of custom RoboSAPIENS Adaptive Platform client libraries, which ensure that each node is initialized properly and consistently according to predefined configurations. Once the nodes are correctly started, an auto-discovery mechanism, similar to the one used in ROS2, is employed to automatically identify and connect these nodes to the (centralized) managing system. This approach streamlines the integration process, allowing the managing system to recognize and incorporate new nodes seamlessly, ensuring efficient coordination and management across the network. To enable such auto-discovery mechanism, the RoboSAPIENS Adaptive Platform client libraries must provide the following functionalities:

- Advertise correct startup status
- Periodic advertising the remote node presence
- Advertise shutdown status, e.g. offline

Please note that the correct startup and discovery mechanisms for remote (compute) nodes is not yet part of the current reference implementation.

6.2 RA platform shutdown

A shutdown is requested via the Core after an application-specific event, either from within the managing or managed system. The Core will orchestrate the correct shutdown of the RoboSAPIENS Adaptive Platform internal architecture modules and will signal the shutdown status to all remote (compute) nodes, via the RoboSAPIENS Adaptive Platform client libraries. The advertising of the shutdown status is not yet part of the current reference implementation.

6.3 RA platform execution

The execution of the RA platform is organized into various flows, which are detailed in the following sections.

Funded by
the European Union

6.4 RA platform runtime update

6.4.1 Knowledge gathering from managed system

The managed system continuously sends data to the managing system. The Telegraph component [1] forwards this data to the knowledge manager using the Telegraph.Digest() function. All the collected data is then consolidated in the knowledge base.

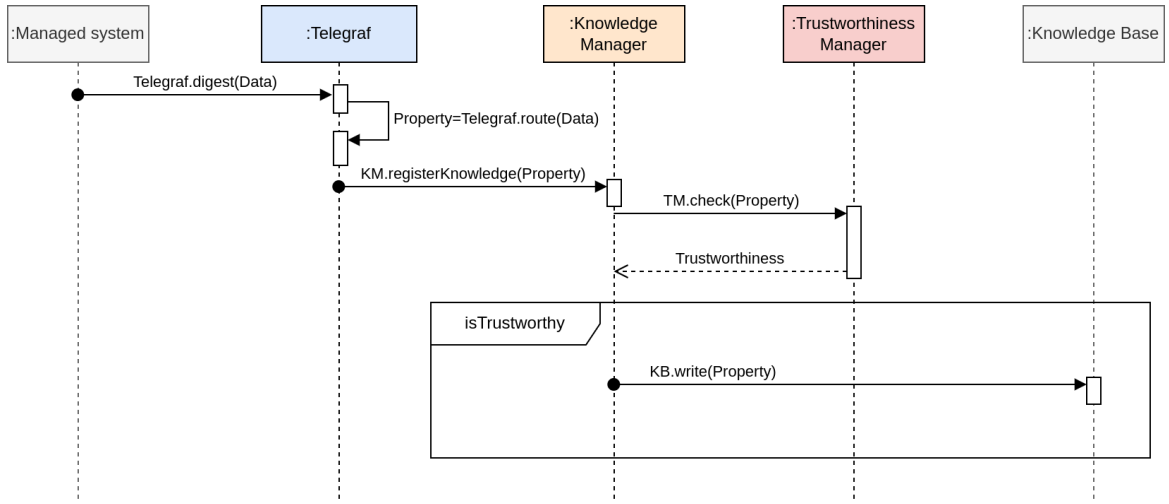

Figure 30: Registering Knowledge from the managed system

6.4.2 Monitoring

As shown in Figure 31, within the monitor component, *the spinOnce()* function retrieves the properties from the knowledge base and processes them based on predefined logic. It assesses the properties and generates the monitoring output status.

[1] Component can either be integrated direclty within the managed system application, or run as standalone component in case data needs to be collected from a 'black-box' system.

Funded by the European Union

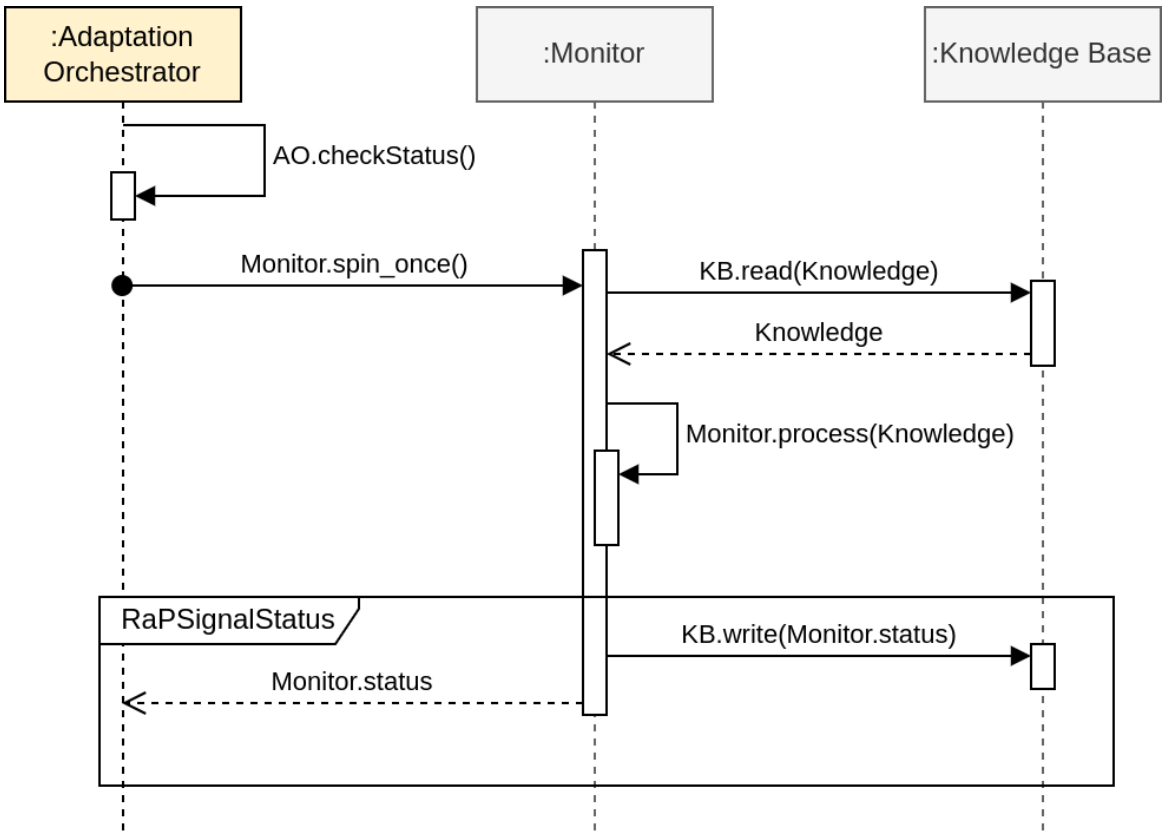

Figure 31: Monitoring knowledge

6.4.3 Planning the diagnosis/adaptation action

As shown in Figure32, when the Analysis component detects an anomaly, the Plan component's *spinOnce()* function is called. This triggers the Plan component to determine the necessary actions, such as diagnosis or adaptation. Next, the *plan-Diagnose()* function is invoked by the Legitimate component to verify the legitimacy of the plan. If the plan is deemed legitimate, the results are recorded in the KnowledgeBase. If not, additional diagnosis is required.

Funded by
the European Union

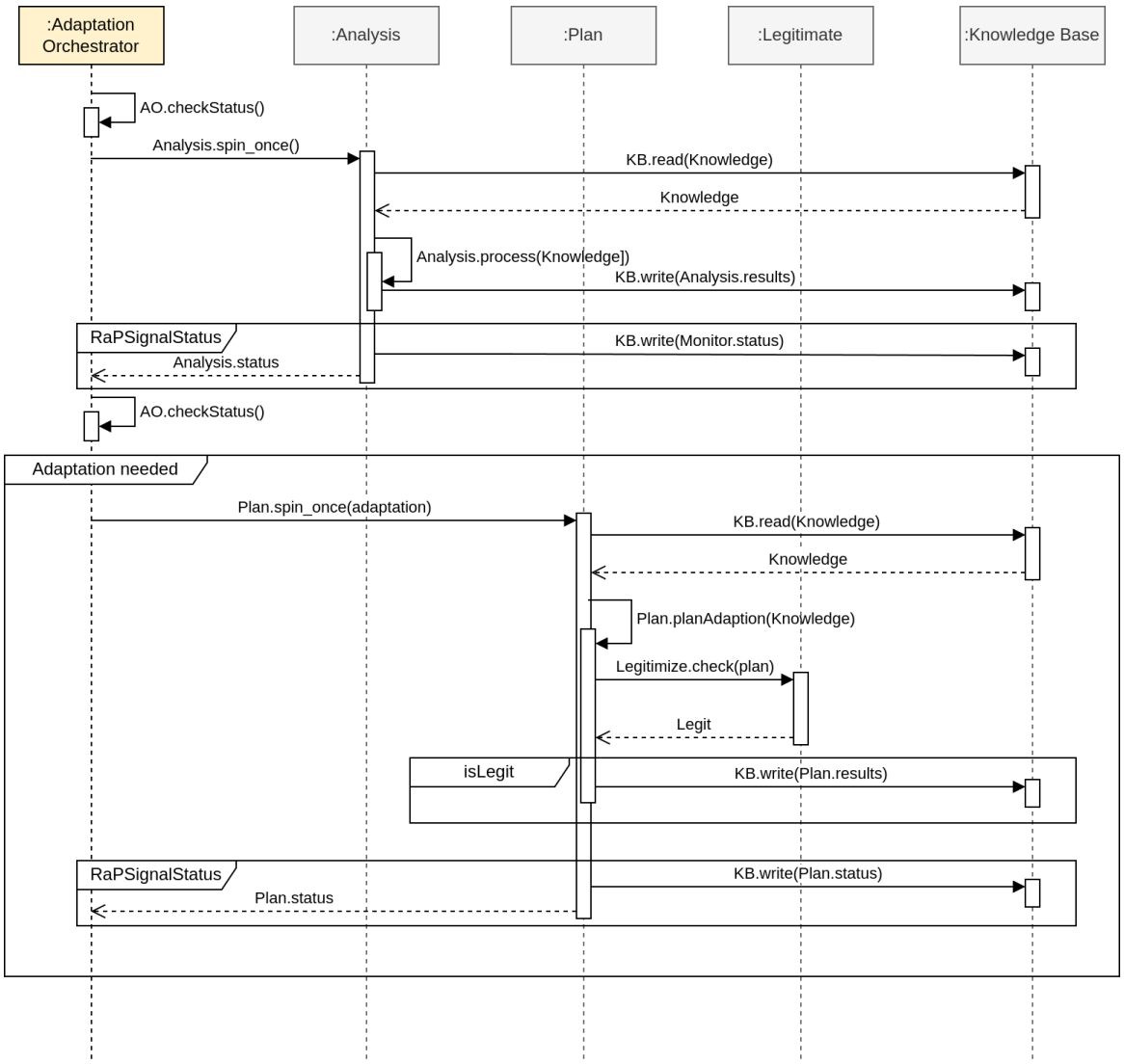

Figure 32: Planning diagnose/adaptation action

6.4.4 Register the diagnosis/adaptation action for execution

Depending on the action type, either diagnosis or adaptation, the execute block calls the *performAdaptation()* interface of the `Adaptation Manager`, providing the action to be carried out. The `Adaptation Manager` receives the action and triggers the *managing system level trustworthiness checks*, using the *checkTrustworthiness()* interface of the `Trustworthiness Manager` component. If the action is marked as trustworthy, the `Adaptation Manager` registers the action to the `Trustworthiness Checker` component, via the *registerAction()* interface, pending to be executed on the managed system. This is shown in Figure 33.

Funded by the European Union

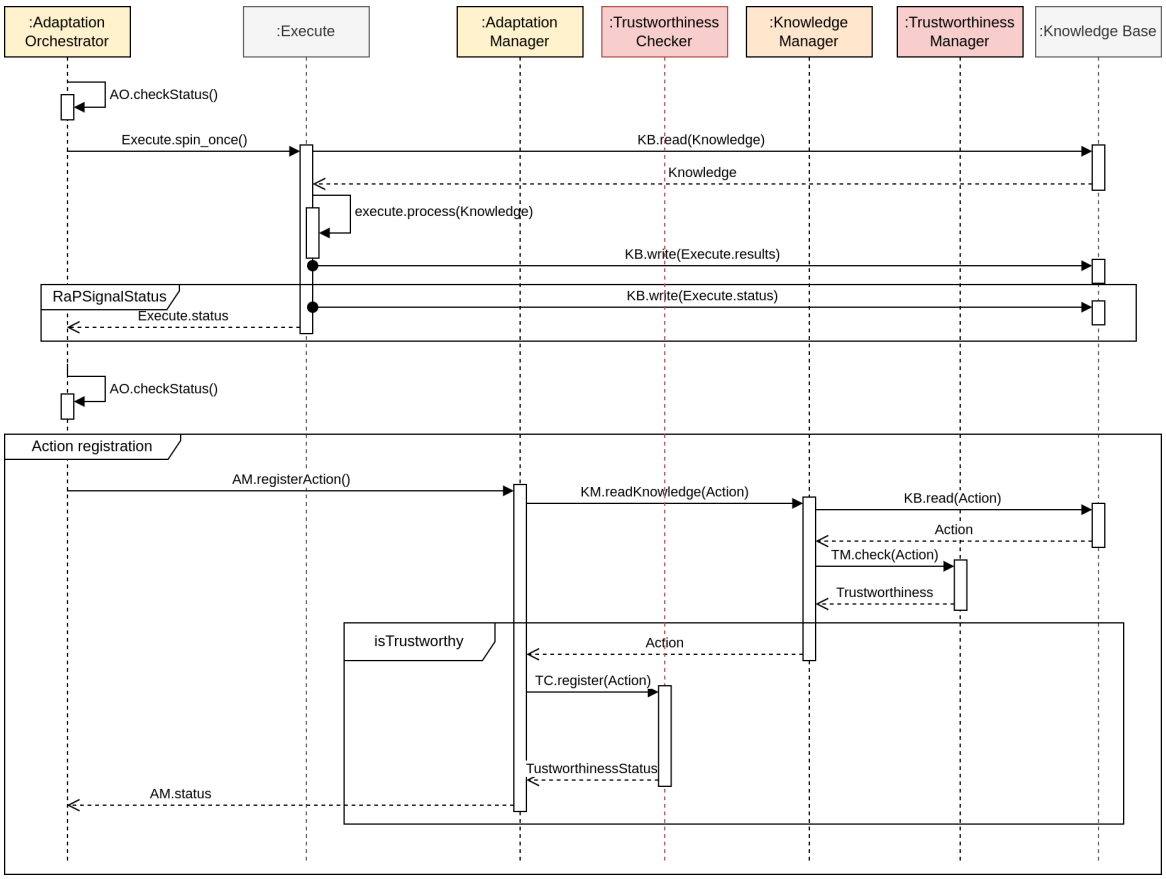

Figure 33: Registering an adaptation action from the managing system.

6.4.5 Execute the diagnosis/adaptation action

After the action/plan registered using the `Adaptation Manager` component. the **execute component** will apply the adaptation action after checking the trustworthiness check in `Trustworthiness Checker` component. This is shown in Figure 34.

Funded by
the European Union

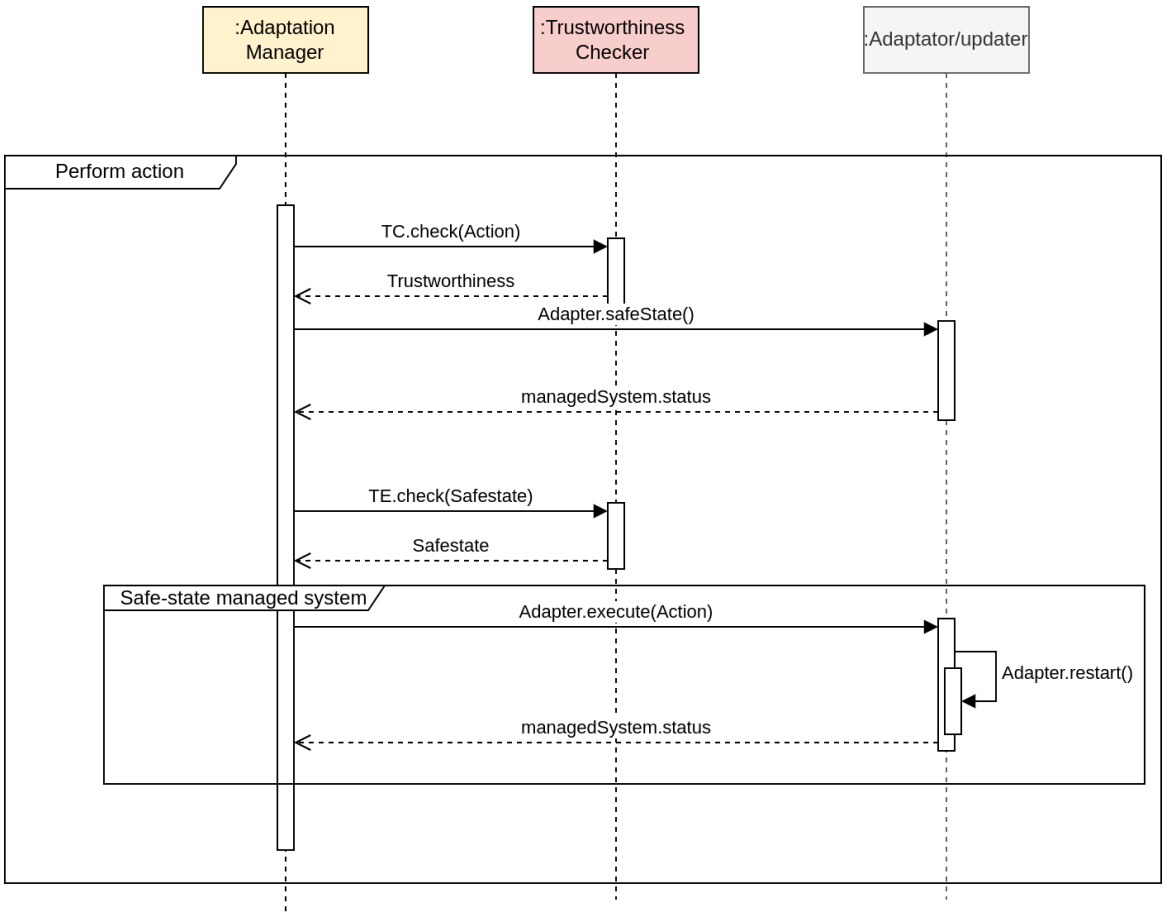

Figure 34: Executing the adaptation action.

Funded by
the European Union

7 Deployment view

This chapter provides an overview of exemplary deployment scenarios for the RoboSAPIENS Adaptive Platform. Deployment refers to the process of installing, configuring, and starting a software application on one or more computing device, for example a microcontroller, Single Board Computer (SBC), or server. Since the RoboSAPIENS Adaptive Platform is highly configurable in its deployment, we provide one representative example of a deployment scenario, namely the deployment view of a self-adaptive loop on a Turtlebot 4 robot.

7.1 TurtleBot 4 Adaptive Sensing and Navigation Scenario

We consider a preliminary use case involving adaptive navigation for the TurtleBot 4: a popular mobile robot platform that is widely used in research and education. This case study demonstrates how the RoboSAPIENS Adaptive Platform can be used to implement a MAPLE-K loop, which extends the TurtleBot 4's navigation capabilities to handle unanticipated sensor failures. Whilst simpler than many of the use cases in the full RoboSAPIENS project case studies, this example aims to provide a prototypical "hello world" scenario for the RoboSAPIENS Adaptive Platform, demonstrating the deployment view of the platform and guiding our subsequent deployments of the platform for the other case studies.

It is equipped with a variety of sensors, including a 2D LiDAR, a stereo camera, and an IMU. The robot is controlled by a Raspberry Pi 4, which runs the Robot Operating System (ROS) 2. The TurtleBot 4 is capable of performing autonomous navigation tasks using the Navigation2 (Nav2) stack, which is a popular ROS 2 package for robot navigation.

7.1.1 The TurtleBot 4 Robotic Platform

The TurtleBot 4 is a popular open-source robotic platform which utilizes the ecosystem to control a 4-wheeled mobile robot with a wide range of sensors and actuators. The TurtleBot 4 extends the capabilities of an iRobot Create 3 educational robot with additional sensors, including a 2D LiDAR, a stereo camera, and an IMU. The robot is controlled by an onboard Raspberry Pi 4 computer, which runs the Robot Operating System (ROS) 2 and interfaces with the low-level controller of the iCreate board. The TurtleBot 4 is capable of performing autonomous navigation tasks using ROS's standard Navigation2 (Nav2) stack for localization, planning, and control based on the LiDAR sensor data.

7.1.2 Navigation Scenario

We consider a task consisting of the TurtleBot 4 navigating through an environment with obstacles, using the Nav2 stack for navigation, using data from the 2D LiDAR sensor to map the environment and perform localization. During the navigation, the LiDAR sensor may experience occlusion by various forms of persistent debris attached to the robot's frame or the LiDAR scanner. This can lead to the robot's perception of the environment being obstructed, forcing the Nav2 stack to make navigation decisions due to partial or outdated maps of the environment. De-

Funded by
the European Union

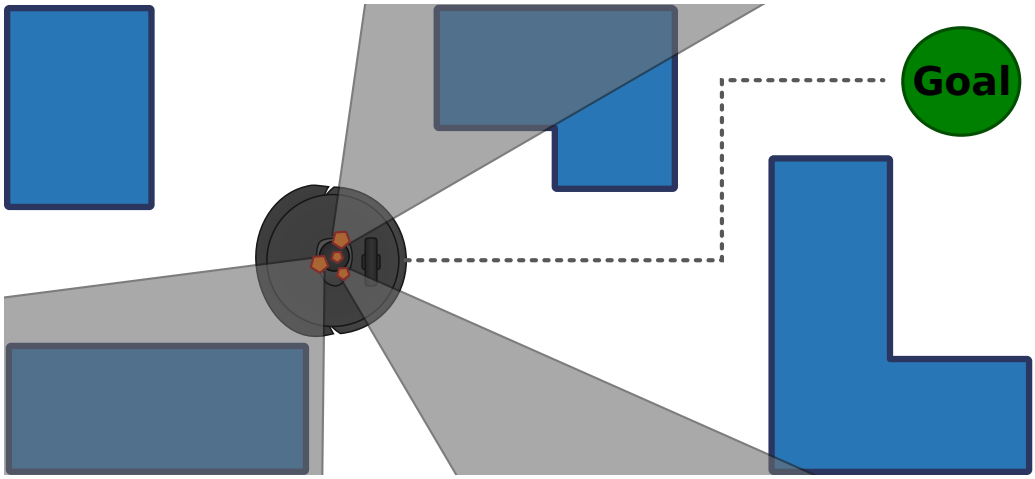

Figure 35: TurtleBox 4 navigation with LIDAR occlusion.

spite the Nav2 stacks existing error correction methods, such impaired perception can still result in navigation failures, or even collisions with obstacles, motivating a self-adaptive extension of the navigation and sensing process.

7.1.3 Self-Adaptive Navigation System

In order to address the issue of LiDAR occlusion, we propose a self-adaptive navigation system that can detect and respond to such occlusion events. This self-adaptive system combines a MAPLE-K loop deployed via the RoboSAPIENS Adaptive Platform with custom extensions to the Nav2 stack to add extensions points allowing the MAPLE-K loop to modify the navigation process. The self-adaptation scenario is defined such that the LiDAR sensor is continuously analysed to detect and model the occlusion of the LiDAR at different angles. When the LiDAR sensor detects an occlusion, then a plan is devised consisting of a sequence of stops and rotations necessary to overcome the detected occlusion and sense the environments.

The implemented algorithm uses the RoboSapiensAdaptive platform to deploy the MAPLE-K loop to the managing system (which could reside on e.g. a laptop, a cloud service, or the Raspberry Pi onboard the Turtlebot). Based on the figure 36 the managed system is sending the LiDAR data to the managing system via ROS2 protocol and a ROS2 to MQTT adapter service converts ROS messages to MQTT messages which are currently compatible with the RA architecture. Then the application dependent part of the architecture which consists of the MAPLE-K components provided by domain experts.

Funded by
the European Union

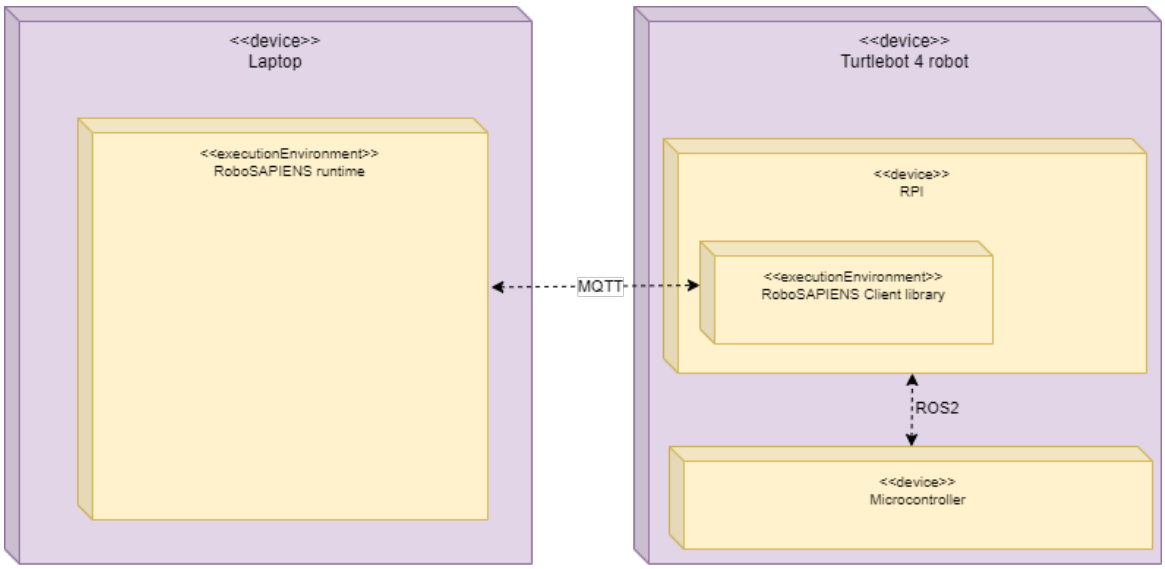

Figure 36: Exemplary TurtleBot 4 hello world case scenario.

Figure 36 depicts the deployment of the *hello world case scenario*. The managed system, the TurtleBot 4, is connected to the RoboSAPIENS Adaptive Platform over , using the RoboSAPIENS Client Library. This client library is running on the , enabling to probe the managed system and perform actions (diagnosis/adaptations) on the managing system. This includes a custom Nav2 controller which is capable of modifying the navigation of the robot based on MQTT/ROS2 messages received from the managing system. The low-level execution of the robot-specific actions, e.g., robot movement, is performed by the microcontroller running the robot firmware, connected over to the .

7.1.3.1 Monitor The MAPLE-K loop relies on LiDAR data from the TurtleBot 4 to monitor the environment and detect occlusions. This LiDAR data is continuously published on the ROS2 topic designated as "/safe-scan," and the ROS2MQTT bridge converts the ROS2 messages to standard MQTT messages. Subsequently, the monitor component writes the LiDAR data to the knowledge base using the probe function.

7.1.3.2 Analyse The analysis phase continuously processes the LiDAR data from the knowledge base to build a model of the probability of occlusion at a given angle based on a sliding window of recent samples. We then perform thresholding and spatial clustering on the LiDAR to detect discrete windows of occlusion of the LiDAR scanner's field of view. If any such windows are detected, we detect an anomaly, and the analysis phase triggers the planning phase to generate a plan to navigate around the occlusion.

7.1.3.3 Plan The planning phase generates a plan of how to insert additional stops and rotations into the TurtleBot 4's navigation plan to overcome the detected occlusions. Specifically, this plan specifies the frequency of the stops, and the angle and duration of rotation during each stop.

Funded by
the European Union

7.1.3.4 Execute Finally in the execution part the plan is being sent using the effector checks the trustworthiness of the plan and then executes the plan via the MQTT to ROS2 bridge to be transmitted to a specific ROS2 topic provided via our additional ROS2 controller plugin.

7.1.3.5 Trustworthiness Checker The trustworthiness checker is responsible for continuously monitoring both the managed system and the managing subsystem to verify a number of requirements covering the behaviour of the system before, during, and after an adaptation occurs. For example, one important requirement is the timeliness of the adaptation process: if the MAPLE-K loop takes to long to detect and respond to an occlusion, then the robot may collide with an obstacle before the adaptation is complete. Such requirements will be captured in the trustworthiness language being developed as part of the RoboSAPIENS project, and the trustworthiness checker will continuously monitor the system to ensure that these requirements are met.

7.1.3.6 Legitimate The Legitimate phase is responsible for providing inline verification of the adaptation plan before it is executed. For example, we can use inline simulation of the robot's motion to verify that the plan is feasible and will not result in a collision. In case the plan is not legitimate, the plan is rejected, and we can either perform replanning to request a different plan, or cancel the self-adaptation if this is not possible.

7.1.4 Nav2 Stack Extension

In order to enable the self-adaptive navigation system, we extend the Nav2 stack with an additional ROS 2 interface that allow the MAPLE-K loop to modify the navigation process. This consists of a custom Controller plugin which extends the Nav2 stack's existing Controller component to additionally receive a rotation plan, which introduces additional stops and rotations in the course of carrying out a motion.

We also extended the Nav2 stack with an additional ROS2 interface which is able to intercept the LiDAR data before it is processed by the Nav2 stack in order to simulate different occlusion scenarios. This makes it possible to test the impact of LiDAR occlusion and the response of the MAPLE-K loop in simulation, using the manufacturer provided Gazebo simulator for the Turtlebot 4.

Funded by
the European Union

8 Generic engineering workflow

8.1 RoboSAPIENS engineering workflow

This chapter provides a generic engineering workflow to develop trustworthy adaptive systems on top of the presented RoboSAPIENS Adaptive Platform. This engineering workflow is part of a project-wide high-level workflow, provided in Figure 37.

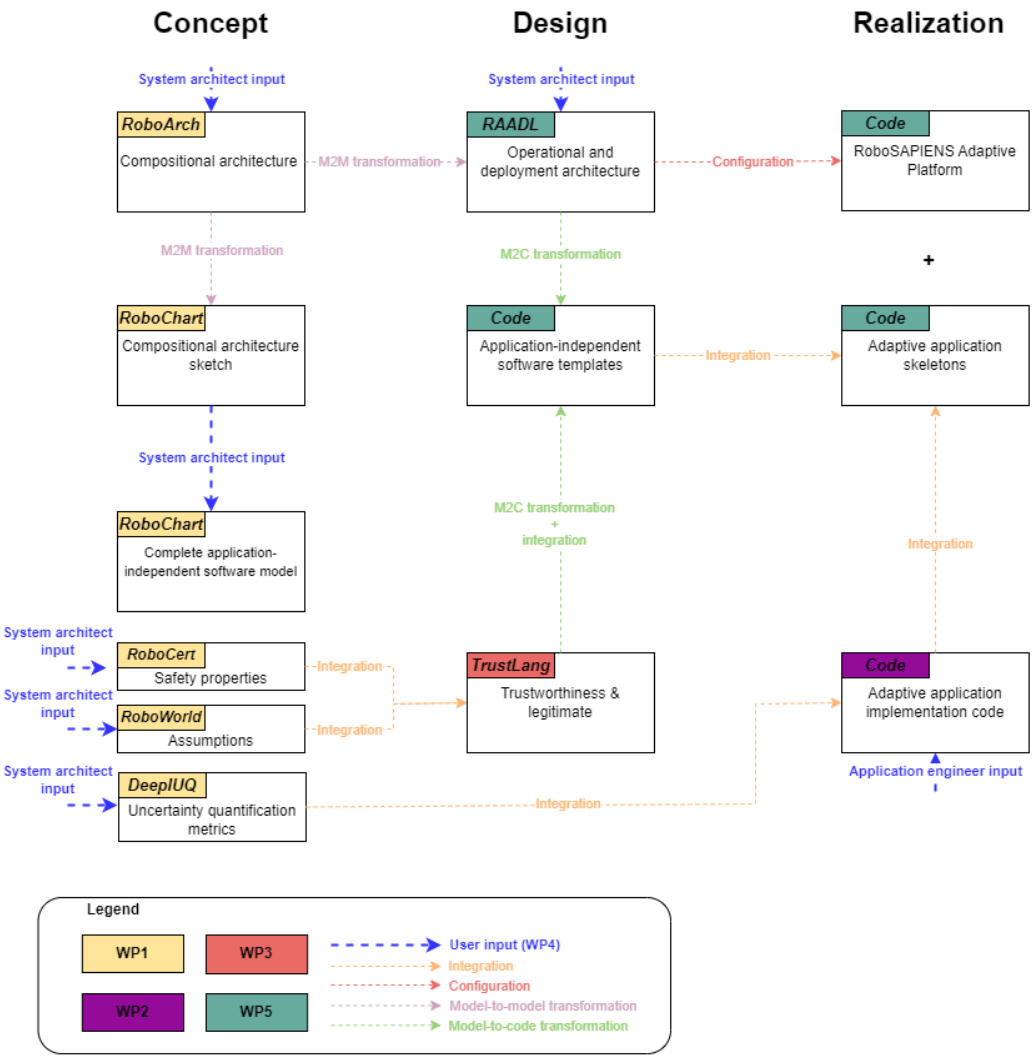

Figure 37: RoboSAPIENS big-picture high-level workflow.

Figure 37 shows the comprehensive engineering workflow structured into three distinct phases: **Concept**, **Design**, and **Realization**. Each phase comprises specific engineering activities that contribute to the overall goal of constructing trustworthy self-adaptive (robotics) systems. Different activities are linked by means of either *transformations*, e.g., model-to-model(M2M), model-to-code(M2C), integrations, (semi-)automated weaving of the activity artifacts or *human interaction*, e.g.,

Funded by
the European Union

modeling, manual coding. This methodical approach ensures a seamless transition from initial architectural concepts to final deployment of self-adaptive systems, emphasizing safety, trustworthiness, and robustness throughout the development life cycle.

8.1.1 Concept phase

Within the concept phase, both the conceptual architecture of the self-adaptive robotics system and the internal (system) and external (environment) affecting properties are described. The conceptual architecture is described using the *RoboArch* notation [BCM22], providing a compositional view on the self-adaptive robotics system. This compositional architecture is transformed into a *RoboChart* sketch [MRL+19], a behaviour model of the robot software controllers using state machines. After completion by the system architect, and annotation of the (safety) properties, specified using *RoboCert* [WC22], and (environment) assumptions, specified using *RoboWorld* [BCCJ23], this *RoboChart* model is used for verification via model checking and theorem proving.

8.1.2 Design phase

Within the design phase, the compositional architecture is used as starting point of the operational and deployment architecture of the self-adaptive robotics system. The system architect is responsible for explicitly modeling the logical and physical architecture, specifying the deployment rules, linking the functional components to the physical (compute) components and specifying the communication matrix. The trustworthiness and legitimate components are specified using the *TrustLang* specification, a specification language under development as part of the RoboSAPIENS project, using the defined property and assumption specifications from the concept phase. Application-independent software templates are then generated from the operational and deployment architecture and trustworthiness models.

8.1.3 Realization phase

Within the realization phase, the application-independent software templates are (semi-)automatically transformed into application-specific skeletons, which enable the application engineers to integrate their custom code for the adaptive applications. These application-specific components can then be deployed and executed on top of the RoboSAPIENS Adaptive platform, the runtime that facilitates the execution of trustworthy self-adaptive (robotics) systems.

8.2 Design and realization workflow

Within this document, we provide more detail on a part of the project-wide workflow, namely on the WP 5 components, shown in turquoise in Figure 37.The following main building blocks are identified:

- **Operational deployment architecture:** Result of the (architecture) modeling activity using an annotated version of Architecture Analysis and Design

Language (AADL) [FLV06], RoboSAPIENS Architecture Analysis and Design Language (RAADL), a domain-specific language for architecture design.

- **Application-independent software templates:** A set of software templates used as input for the software-independent code generation
- **RoboSAPIENS adaptive platform:** The actual implementation architecture enabling trustworthy adaptive application execution, subject of description of the first part of this document.
- **Adaptive application skeleton:** Result of the model-to-code transformation from the RAADL architecture model, combined with the software templates, executable on top of the RoboSAPIENS Adaptive Platform.
- **Adaptive application implementation code:** Application-specific code to specify the internal behavior of the different adaptive application components.

Based on the above mentioned main building blocks, and their interconnections as shown in Figure 37, we provide a more detailed view on the envolved engineering activities, depicted as a process model,in Figure 38. Using this process model, the control and data flow between engineering activities can be specified. We envision to extend these workflows to a full Formalism Transformation Graph + Process Model (FTG+PM) [MDLV12, LMD+12], adding the Formalism Transformation Graph (FTG) counterpart along the process model, specifying the used formalisms and their interrelation explicitly. Each of the identified engineering activities (round angle boxed) are discussed in detail below.

Funded by
the European Union

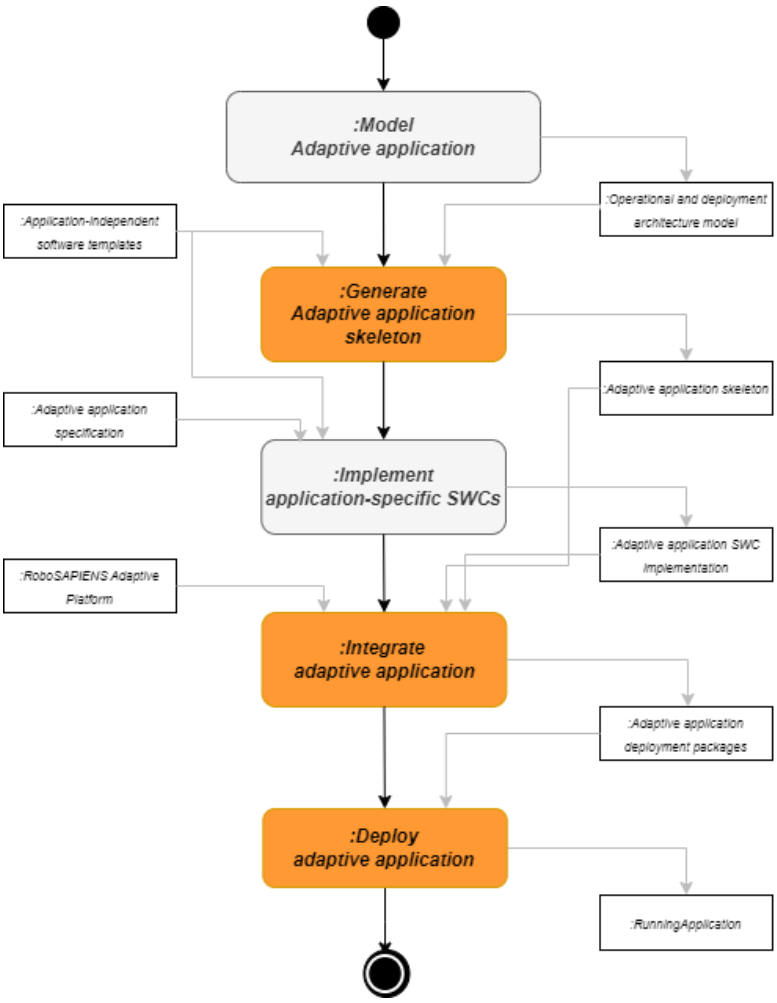

Figure 38: Identification of the in-scoop engineering activities

Funded by
the European Union

8.2.1 Model adaptive application

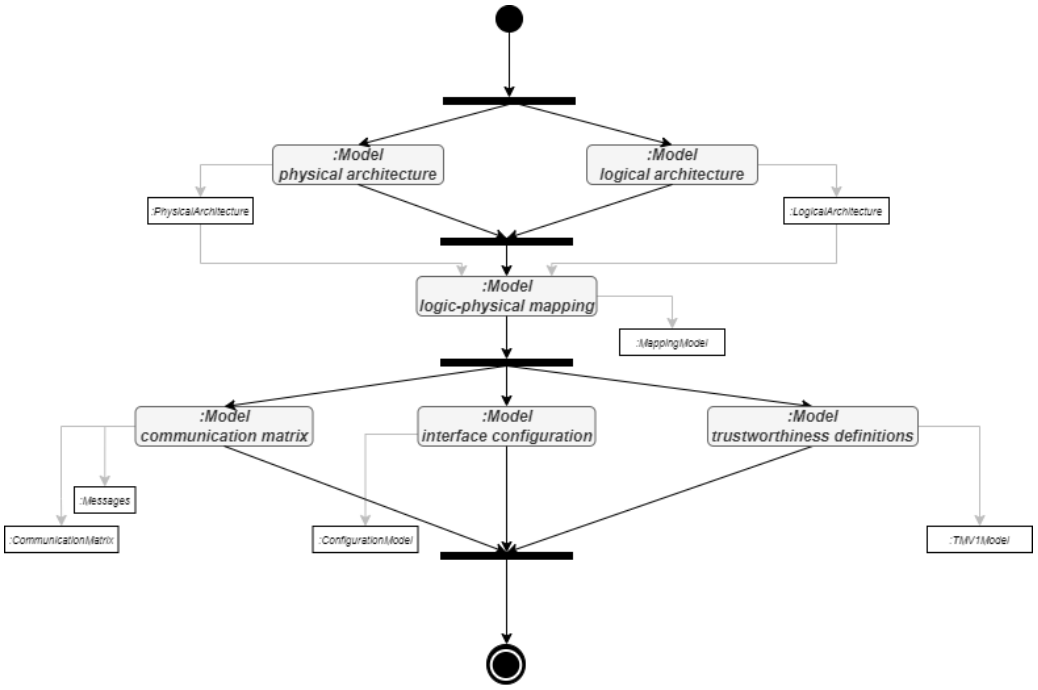

Figure 39: Identification of the modeling activities

Figure 39 shows a more detailed workflow of the modeling activity, which involves the following activities:

- **Modeling logical architecture:** Defines the system's functional components and their interactions (what the system does).
- **Modeling physical architecture:** Describes the hardware components and their connections (what the system is built with).
- **Modeling logical-physical mapping:** Links the functional components to the physical ones (how the software interacts with the hardware).
- **Modeling communication matrix:** Shows how components exchange data (who talks to whom).
- **Modeling trustworthiness definition:** Specifies security and reliability requirements (how trustworthy the system needs to be).
- **Modeling interface configuration:** Defines how components connect and share data (the communication rules).

8.2.2 Generate adaptive application skeletons

The system architect starts by building a model that defines the core of our adaptive application. Using this model as input, a code skeleton can be (semi-)automatically generated. Such code skeleton provides all application-independent functions to interact with the RoboSAPIENS Adaptive Platform, and application-specific code placeholders, ready to be implemented by the application engineer. Please note

Funded by
the European Union

that A code skeleton can be generated for various programming languages, such as C/C++ and Python, as well as for different interface protocols, like MQTT and ROS2, providing a foundational framework for developing applications across diverse platforms and systems.

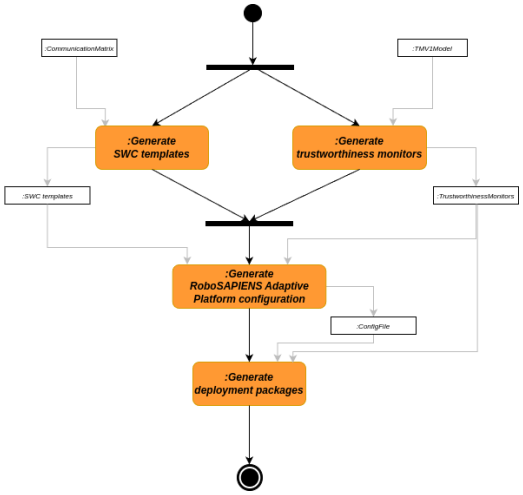

Figure 40: Generate Adaptive Aplication Skeleton Workflow

8.2.3 Implementing application-specific software components

Starting from the generated code skeletons, provided in the programming language of choose, the application engineer can integrate their application-specific logic into the code skeleton. The application engineer can easily exchange data with the RoboSAPIENS Adaptive Platform, e.g. using system knowledge or trigger an anomaly, using the provided interface functions. Figure41 illustrates the implementation workflow.

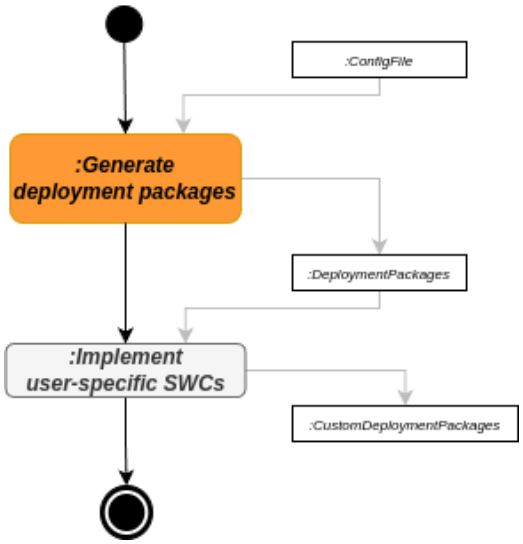

Figure 41: Implement Application Specific Component Workflow

Funded by the European Union

8.2.4 Integrate adaptive application

From the RAADL models, platform-specific deployment packages are generated. These packages, essentially a zip folder, include all the necessary elements and configurations to run the application seamlessly on the target platform[2]. In addition to application-independent and application-specific (user) code, the package contains messages that facilitate communication with other components of the RoboSAPIENS Adaptive Platform and associated processes or workflows. These workflows, serving as deployment recipes, outline a series of step-by-step actions, such as code compilation, to automate and streamline the deployment tasks, like flashing and running on a given platform, minimizing human intervention, enhancing both efficiency and consistency.

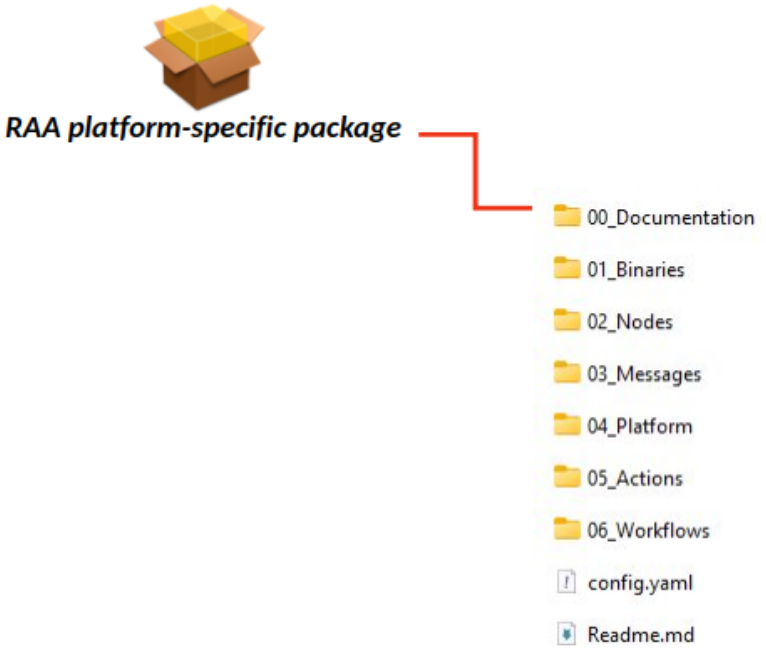

Figure 42: Implement Adaptive Application

Depicted from Figure 42, An RAA platform-specific package comes with a standardized internal folder structure. The folder structure looks as follows:

- **00_Documentation:** Component-specific (non-structured) documentation
- **01_Binaries:** Platform-dependent binaries
- **02_Nodes:** Collection of application nodes,classified within the MAPLE-K layers or managed system layers. Example application nodes are, but are not limited to, Python nodes, C/C++ nodes, ROS2 nodes.

[2]Target platforms, or deployment targets, can range from resource-constrained microcontrollers to powerful SBCs with various architectures, such as amd64, arm64, and aarch, as well as full-scale cloud servers.

Funded by
the European Union

- **03_Messages:** User-defined messagesfor given platform, similar to the custom messages within the ROS framework.
- **04_Platform:** Platform-specific configuration for deployment
- **05_Actions:** Actions, e.g. code compile, flash, deploy, validate, as executable code or scripts. These actions can be triggered from a simple deployment tool, under development as part of the RoboSAPIENS project
- **06_Workflows:** Trustworthiness checklist + (semi-) automated workflows, a series of step-by-step actions, e.g. configuration and startup, validation, that can be automatically executed. Execution of these workflows can be triggered from a simple deployment tool, under development as part of the RoboSAPIENS project

8.2.5 Deploy adaptive application

The deployment packages can be deployed, using the contained deployment workflow(s), and executed on various hardware configurations, including popular choices like Raspberry Pi, robot controller MCUs, and microcontroller boards. This deployment can be triggered from a simple deployment tool, under development as part of the RoboSAPIENS project. Figure 43 shows the generic workflow.

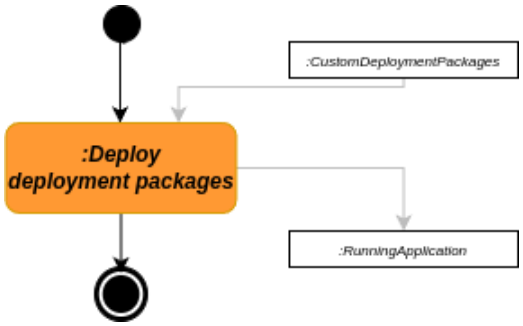

Figure 43: The Generic Engineering Workflow

Funded by
the European Union

9 Risk analysis

This chapter lists and rates risks associated with the overall architecture of the RoboSAPIENS Adaptive Platform.

9.1 Risks

9.1.1 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 9.1.1.

Table 1: Risk severity matrix

	Probability				
Impact	*very low*	*low*	*medium*	*high*	*very high*
very low	low	low	low	medium	medium
low	low	medium	medium	high	high
medium	low	medium	high	high	high
high	medium	high	high	extreme	extreme
very high	medium	high	high	extreme	extreme

Funded by
the European Union

9.1.2 Risk list

Risk ID	Description	Probability	Impact	Mitigation Strategy
R_01	It is possible that the current RoboSAPIENS Adaptive Platform reference implementation does not meet the performance requiments of one or more use cases	medium	medium	Current reference implementation is implemented in Python. If it fails to meet defined performance requirements, we can optimize performance, either by optimizing the current implementation or change to a C/C++ based implementation.
R_02	Currently, there is an incomplete understanding of the RoboChart models and the integration with the RAADL models.	medium	medium	The integration of RoboChart within the overall engineering workflow is currently under investigation. Further investigation will reduce or eliminate this risk.
R_03	It is possible that the deployment of the RoboSAPIENS Adaptive Platform components is very hard or even impossible on some hardware architectures.	low	low	We will first focus on the hardware architectures used within the industrial and lab-scale use cases. This will enable us to better understand the deployment worflows and requirements and to identify the supported hardware infrastructures.

Funded by
the European Union

References

[aiR] Ethics guidelines for trustworthy AI. https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai. Accessed: 2024-9-10.

[BCCJ23] James Baxter, Gustavo Carvalho, Ana Cavalcanti, and Francisco Rodrigues Júnior. Roboworld: Verification of robotic systems with environment in the loop. Formal Aspects of Computing, 35(4):1–46, 2023.

[BCM22] Will Barnett, Ana Cavalcanti, and Alvaro Miyazawa. Architectural modelling for robotics: Roboarch and the cortex example. Frontiers in Robotics and AI, 9:991637, 2022.

[BGGL23] Andrea Bonci, Francesco Gaudeni, Maria Cristina Giannini, and Sauro Longhi. Robot operating system 2 (ros2)-based frameworks for increasing robot autonomy: A survey. applied sciences, 13(23):12796, 2023.

[EH09] David Emery and Rich Hilliard. Every architecture description needs a framework: Expressing architecture frameworks using iso/iec 42010. In 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture, pages 31–40. IEEE, 2009.

[FLV06] Peter H Feiler, Bruce A Lewis, and Steve Vestal. The sae architecture analysis & design language (aadl) a standard for engineering performance critical systems. In 2006 ieee conference on computer aided control system design, 2006 ieee international conference on control applications, 2006 ieee international symposium on intelligent control, pages 1206–1211. IEEE, 2006.

[ite] itemis create - documentation. https://www.itemis.com/en/products/itemis-create/documentation/user-guide. Accessed: 2024-9-23.

[IW15] Didac Gil De La Iglesia and Danny Weyns. Mape-k formal templates to rigorously design behaviors for self-adaptive systems. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 10(3):1–31, 2015.

[LMD+12] Levi Lúcio, Sadaf Mustafiz, Joachim Denil, Bart Meyers, and Hans Vangheluwe. The formalism transformation graph as a guide to model driven engineering. School Comput. Sci., McGill Univ., Tech. Rep. SOCS-TR2012, 1, 2012.

[MDLV12] Sadaf Mustafiz, Joachim Denil, Levi Lúcio, and Hans Vangheluwe. The ftg+ pm framework for multi-paradigm modelling: An automotive case study. In Proceedings of the 6th International Workshop on Multi-paradigm Modeling, pages 13–18, 2012.

[MRL+19] Alvaro Miyazawa, Pedro Ribeiro, Wei Li, Ana Cavalcanti, Jon Timmis, and Jim Woodcock. Robochart: modelling and verification of the functional behaviour of robotic applications. Software & Systems Modeling, 18:3097–3149, 2019.

[Mue11] Andreas Muelder. Yakindu. Yakindu Statechart Modeling Tools, 2011.

Funded by
the European Union

[NYZ17] Syeda Noor Zehra Naqvi, Sofia Yfantidou, and Esteban Zimányi. Time series databases and influxdb. Studienarbeit, Université Libre de Bruxelles, 12:1–44, 2017.

[WC22] Matt Windsor and Ana Cavalcanti. Robocert: property specification in robotics. In International Conference on Formal Engineering Methods, pages 386–403. Springer, 2022.

10 Appendix

RoboSAPIENS Adaptive Platform Requirements

Funded by
the European Union

ID	Name	Description	RoboSAPIENS Requirement
ID_RAP_01	Supported computing platforms	the RoboSAPIENS Adaptive Platform shall support both resource-constraint platforms and high performance computing platforms, both local (edge node) and remote (cloud).	-
ID_RAP_02	Online learning support	he The RoboSAPIENS Adaptive Platform shall support online learning, a technique where the (Digital Twin) model continues to incrementally learn/update its parameters as soon as the new data points are available.	-
ID_RAP_03	Flexible data and software update	The RoboSAPIENS Adaptive Platform shall support flexible updating of configuration data and software packages.	-
ID_RAP_04	Logical architecture description	A unified way to describe adaptive systems shall be provided, which enables to describe the logical architecture, the adaptive software system structure.	ID500
ID_RAP_05	Physical architecture description	A unified way to describe adaptive systems shall be provided, which enables to describe the physical architecture, the compute unit structure.	ID500
ID_RAP_06	Allocation and deployment description	A unified way to describe adaptive systems shall be provided, which enables to describe the deployment and allocation of the logical architecture components on one or more compute units.	ID500;ID501
ID_RAP_07	Device interface description	A unified way to describe adaptive systems shall be provided, which enables to describe the inter- and intra-device interfaces of the entire system	ID500
ID_RAP_08	Communication protocol support	The RoboSAPIENS Adaptive Platform shall support multiple standardized communication protocols for (bi-directional) inter- and/or intra-device communication with different network topologies.	-
ID_RAP_09	Transparency	The RoboSAPIENS Adaptive Platform shall provide diagnostics means during runtime, providing explainability on the robots data, actions and decisions in a transparent way	-

Table 2: RoboSAPIENS Adaptive Platform requirements

Funded by
the European Union

RoboSAPIENS trustworthiness definition

RoboSAPIENS trustworthiness definition

Trustworthiness in the context of RoboSAPIENS refers to the degree to which robots featuring the MAPLE-K architecture are perceived as robust, safe, and capable of performing tasks as expected during runtime. This includes their compliance to ethical or legal boundaries and their inability to cause harm to humans, living creatures, or the environment. The concept entails the following aspects that will be integrated into the RoboSAPIENS' MAPLE-K loop as internalized norms that are tightly linked to the ethics guidelines for trustworthy AI of the European Union [aiR]:

Technical robustness and safety: The robot is resilient and thus able to consistently perform its designated tasks accurately and effectively over time. Additionally, a trustworthy robot can reliably adapt the execution of its designated task and operation in volatile and uncertain environments, changes in conditions, or recover from failures or errors. The robot's behavior prevents accidents, injuries to humans and living creatures, damage to itself and its environment.

Transparency: A robot's data, actions and decisions are made transparent in a way that humans understand. Traceability mechanisms help to achieve this. Additionally, decisions, a robot's capabilities and limitation will be explained in a manner adapted to the robot users concerned.

Diversity, non-discrimination and fairness: The robot avoids unfair bias and any form of discrimination. Its actions and decisions will uphold diversity to make it accessible to all humans, regardless of any disability, and will involve relevant stakeholders.

Privacy and Data Governance: The data collected, processed and stored by the robot are securely managed while privacy is maintained. Adequate data governance mechanisms ensure the quality and integrity of the data, and ensuring legitimized access to data.

The following aspects are equally relevant, but not particularly considered in RoboSAPIENS:

Accountability: Mechanisms should be put in place to ensure responsibility and accountability for the robot and its actions.

User experience and social well-being: The robot is easy to use and responsive to user inputs. It is beneficial for all users and is environmental friendly. The social and societal impact is carefully considered.

Human agency and oversight: The robot should empower human beings, allowing them to make informed decisions and fostering their fundamental rights. At the same time, proper oversight mechanisms need to be ensured, which are achieved through human-in-the-loop, human-on-the-loop, and human-in-command approaches

Building trustworthiness involves multiple robot dimensions such as its physical design, interfaces etc. The aspects listed above can also be applied to these dimensions. They are, however, not considered in RoboSAPIENS and thus not for the trustworthiness checkers of the MAPLE-K loop.

Funded by
the European Union