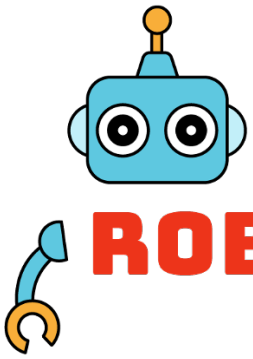


ROBOSAPIENS

D5.2

Distributed architecture for the RoboSAPIENS platform

WP5

Public Document

Grant Agreement	101133807
Project	RoboSapiens
Deliverable Number	D5.2
Version	0.5
Due Month	18
Date	September 2025

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
Avramelou Loukia, AUTH
Symeonidis Charalampos, AUTH
Tosidis Pavlos-Apostolos, AUTH
Passalis Nikolaos, AUTH
Nikolaidis Nikolaos, AUTH
Tefas Anastasios, AUTH
Thomas Wright, AU
Beatriz Sanguino, NTNU

Editors:

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

Internal reviewers:

Claudio Gomes, AU
Nikolaos Passalis, AUTH

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	27-01-2025	Sahar Nasimi Nezhad, Bert Van Acker	Initial document version
0.2	28-02-2025	Sahar Nasimi Nezhad, Bert Van Acker	Populating structure with own conent
0.3	25-03-2025	Sahar Nasimi Nezhad, Bert Van Acker	Gathering input from case study owners
0.4	30-05-2025	Sahar Nasimi Nezhad, Bert Van Acker	Content integration
0.5	20-06-2025	Sahar Nasimi Nezhad, Bert Van Acker	Internal review feedback processed

Funded by
the European Union

Abstract

This deliverable (D5.2) outlines the progress made in Work Package 5 (WP5) between month ten (M10) and month eighteen (M18). Taking deliverable D5.1 as the baseline, it highlights the additions and advancements achieved during this period, with a focus on the design and validation of the RoboSAPIENS deployment architecture. It introduces new functionalities such as support for distributed operations and the integration of trustworthiness features, and describes the robosapiensIO (RPIO) toolkit, developed to support trustworthy self-adaptive robotics systems. The deliverable also summarizes ongoing integration efforts with case studies to validate both the platform and toolkit.

Funded by
the European Union

Contents

Funded by
the European Union

Acronyms

AADL Architecture Analysis and Design Language. 9, 10, 15, 16, 26, 52

AI Artificial Intelligence. 18

DRL Deep Reinforcement Learning. 29, 30

DSL Domain-Specific Language. 19

DT Digital Twin. 7, 9, 17, 51

IMU Inertial Measurement Unit. 25

JSON JavaScript Object Notation. 19

Lidar Light Detection and Ranging. 23–26, 29, 30, 32, 39, 40, 42, 45, 47–49

LSTM Long Short-Term Memory. 27–29

M2C model-to-code. 10, 13, 14, 16

M2M model-to-model. 10, 13, 14, 16

MAPLE-K Monitor, Analysis, Plan, Legitimate, Execute and Knowledge. 9, 22, 30, 32, 34

MQTT Message Queuing Telemetry Transport. 12, 17–20, 25

RAP RoboSAPIENS Adaptive Platform. 5, 7–11, 13, 15–17, 23, 51, 52

ROS2 Robot Operating System 2. 17, 19

RPIO robosapiensIO. 5, 7–11, 13, 20, 22, 26, 51, 52

Funded by
the European Union

1 Introduction

This explanatory document outlines the progress made since the previous deliverable D5.1, highlighting the features added over the past nine months. It provides a comprehensive overview of recent advancements in the RAP, including improvements in software architecture, distributed operations, and the integration of trustworthiness mechanisms. It also presents updates on the supporting toolkit, RPIO, and describes the ongoing case studies used to validate both the RAP framework and the RPIO toolkit.

RAP is the software architecture developed in this project to support the deployment of *trustworthy self-adaptivity* to robotic applications. It provides a conceptual and structural foundation such that those robotic systems become capable of adapting to changing conditions at runtime, while maintaining trustworthiness.

In contrast, RPIO is a *software toolkit*, also developed in this project, that operationalizes the RAP architecture. It assists engineers in the *concept definition*, *design*, *configuration*, and *code generation* of the RAP-based software components. Beyond generating the functional code for RAP-based components, the RPIO toolkit also produces deployment-related code that facilitates the integration and distribution of these software components across distributed embedded computing platforms.

The work performed within WP5 aligns with the overall objectives of the project. More specifically, the conducted work has advanced the State-of-the-Art (SotA) with respect to the following project objectives:

O1 *Enable control software robotic open-ended self-adaptation in response to unprecedented system structural and environmental changes*

The layered RAP software architecture (Section 3) was developed, which provides a set of services to handle the open-ended self-adaptation in response to unprecedented events. Besides the software architecture itself, a software toolsuite, RPIO toolkit (Section 4), is developed to help engineers construct such open-ended self-adapting robotic systems.

O2 *Advance safety engineering techniques to assure robotic safety not only before but also during adaptation and after adaptation has taken place.*

The RAP software architecture was extended to incorporate both the *trustworthiness checker* and the *legitimate* components (Section 4.4). These components ensure the robot behaves in a trustworthy manner: the *legitimate* component evaluates the trustworthiness of future behavior through Digital Twin (DT) simulations before adaptation, while the *trustworthiness checker* continuously monitors behavior during and after adaptation.

Since the proposed RAP architecture is capable to include the concepts elaborated in WP1, WP2 and WP3, it therefore indirectly contributes to the project objectives O3 and O4 as well.

Funded by
the European Union

1.1 Document objectives

The objectives of the document are listed below:

- Identify the **stakeholders** of the RAP and their concerns.
- Identify high-level system goals by means of requirements.
- Provide an overview of **RAP advancements**, to show the architecture development progress.
- Provide an overview of the **supporting RPIO toolkit**, and explain its purpose and main features.
- Provide an overview of **ongoing case studies** and identify how those use cases are employed to validate the RAP architecture and the RPIO toolkit.

1.2 Document scope

This document covers both the RAP software architecture and the accompanying RPIO toolkit. It provides a short recap of the original RAP architecture described in deliverable D5.1 [ANM24], followed by key advancements made since. These include the introduction of distributed operation, integration of trustworthiness features, and several improvements based on user feedback. The RPIO toolkit is also presented as a means to support developers in building and deploying trustworthy self-adaptive robotic systems using the RAP architecture. As the platform continues to evolve, future updates will be provided in subsequent releases.

This version of the RAP software architecture introduces distributed operation and trustworthiness integration, along with several extensions added in response to user feedback. As the platform continues to evolve, it is expected that new capabilities will emerge. Any future architectural enhancements and feature roll-outs will be documented in the corresponding release notes and in follow-up documents.

1.3 Document structure

This document begins by identifying the key requirements and stakeholders relevant to this deliverable. It then provides a recap of the original RAP architecture and highlights the main advancements made to support distributed operation and trustworthiness integration. Following that, it introduces the RPIO toolkit, which supports the development of trustworthy self-adaptive robotics systems. The document concludes with an overview of ongoing case studies that serve to validate both the RAP architecture and the RPIO toolkit.

Funded by
the European Union

2 Stakeholders and requirements

This section provides an update on the architecture and toolkit requirements and identifies the primary stakeholders.

2.1 Requirements update

The set of requirements defined in Deliverable D5.1 remains largely unchanged in this deliverable. However, requirement ID_RAP_02 has been updated to better reflect the actual goals and scope of the project, based on insights gained during development and integration. In addition, two new requirements, ID_RAP_10 and ID_RAP_11, have been introduced on the link between formal verification and realization.

Requirement ID_RAP_02 The requirement, which was previously focused specifically on online learning for updating the DT model, has been generalized. It now states that the RAP shall support online updating of all software components at runtime. This broader formulation reflects the project's goal to enable continuous evolution of the Monitor, Analysis, Plan, Legitimate, Execute and Knowledge (MAPLE-K) loop and its components, including but not limited to online learning, thereby enhancing the platform's adaptability and long-term autonomy.

Toolchain construction To ensure seamless integration between formal verification and deployment phases of the self-adaptive system, a dedicated toolchain must be provided. The toolkit shall enable transformation from RoboWorld [BCCJ23], used for formal verification of the self-adaptive system model, to Architecture Analysis and Design Language (AADL), for annotating the model with deployment specifications (ID_RAP_10), and shall also support semi-automated deployment on heterogeneous compute architectures by means of code generation (ID_RAP_11).

2.2 Requirements progress

The updated RAP architecture and the accompanying RPIO toolkit address a set of requirements, some of which apply specifically to the RAP architecture, others to the RPIO toolkit, and some to both. Detailed explanations are provided below:

- **ID_RAP_01 - Distributed operation:** The updated version of the RAP architecture supports distributed operation on heterogeneous platforms, e.g., servers and edge nodes, by means of a custom middleware and RPIO client libraries.
- **ID_RAP_02 – Online updating and evolution of software components:** Initial support for online updating of software components at runtime has been introduced. This includes mechanisms to store and restore the state of MAPLE-K components, as well as remote triggering of component replacement or reconfiguration.
- **ID_RAP_03 - Flexible update and configuration:** Support for flexible update and (re-)configuration of the system is added by introducing an updated deployment package and extending the supported communication strategies within the updated RAP architecture.

Funded by
the European Union

- **ID_RAP_04-07 - Unified architecture description:** Support for describing different parts of the trustworthy self-adaptive systems is added by means of AADL models. Using these architectural and mapping models, the RPIO toolkit supports engineers in the development of trustworthy self-adaptive systems by automating the realization, system configuration and deployment.
- **ID_RAP_08 - Communication protocols:** The communication manager of the updated version of the RAP architecture is extended to support more (standardized) communication protocols. In addition, the knowledge manager is also extended to support a broader range of knowledge-sharing protocols.
- **ID_RAP_09 - Diagnosis:** The updated version of the RAP architecture supports remote monitoring of both the RAP architecture building blocks and the user-specified MAPLE-K nodes, providing a starting point for proper diagnostics.
- **ID_RAP_10 - Concept-to-design:** The RPIO toolkit contains a set of model-to-model (M2M) transformations, transforming the conceptual models, modeled within the RoboWorld ecosystem, into AADL model skeletons.
- **ID_RAP_11 - Design-to-realization:** The RPIO toolkit contains a set of model-to-code (M2C) transformations, transforming the annotated AADL models to deployment-specific code and RAP configuration, ready for deployment on one or more compute units.

2.3 Stakeholders

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

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

Funded by
the European Union

3 The RoboSAPIENS Adaptive Platform

As discussed in the related work, most self-adaptive robotic applications are implemented ad hoc, treating trustworthiness and self-adaptivity as separate concerns. The RPIO framework is designed to provide a generic, standardized end-to-end solution for developing and deploying trusted self-adaptive applications across various robotic domains. In the following sections, we extend the key tools of the RPIO framework.

As shown in Figure 1 the MAPLE-K adds trustworthiness by introducing two main components, indicated in yellow, the *legitimate* and *trustworthiness checker* components.

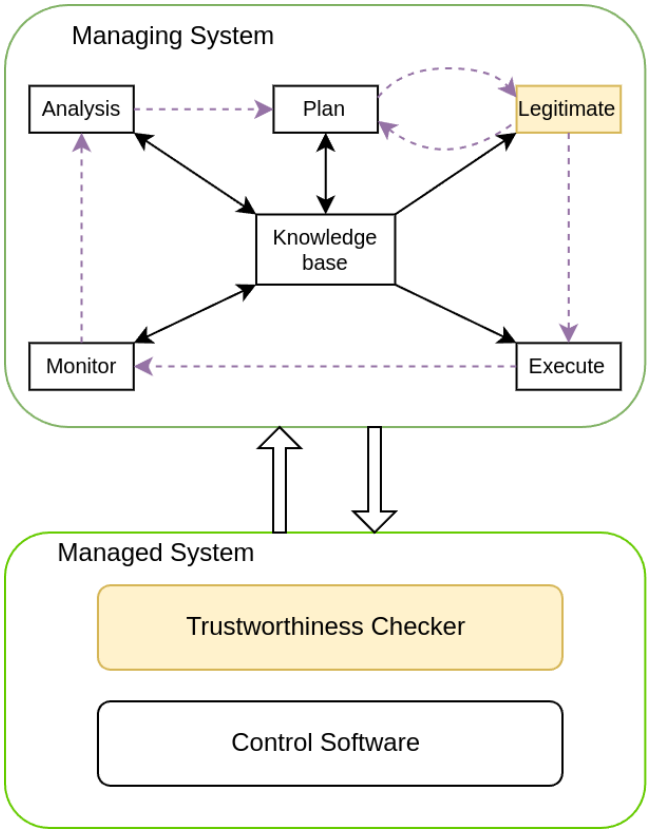


Figure 1: The trust components integrated in the architecture

3.1 RoboSAPIENS Adaptive Platform

The RAP architecture provides a software architecture for developing and managing MAPLE-K pattern while maintaining trust and reliability.

As shown in Figure 2 the RAP is designed as a service layer to fully support MAPLE-K loop development. The MAPLE-K components are running on the managing system, which is the place to run the adaptive application. Also, the probe and effector are two software components that are running on the managed system or the robot to use the standardized message protocol to send the data from the robot or environment to the adaptive application. This way, the RAP provides services to enable

Funded by
the European Union

running the trust-integrated distributed MAPLE-K pattern. The architecture is designed to work in single-robot and distributed multi-environment scenarios.

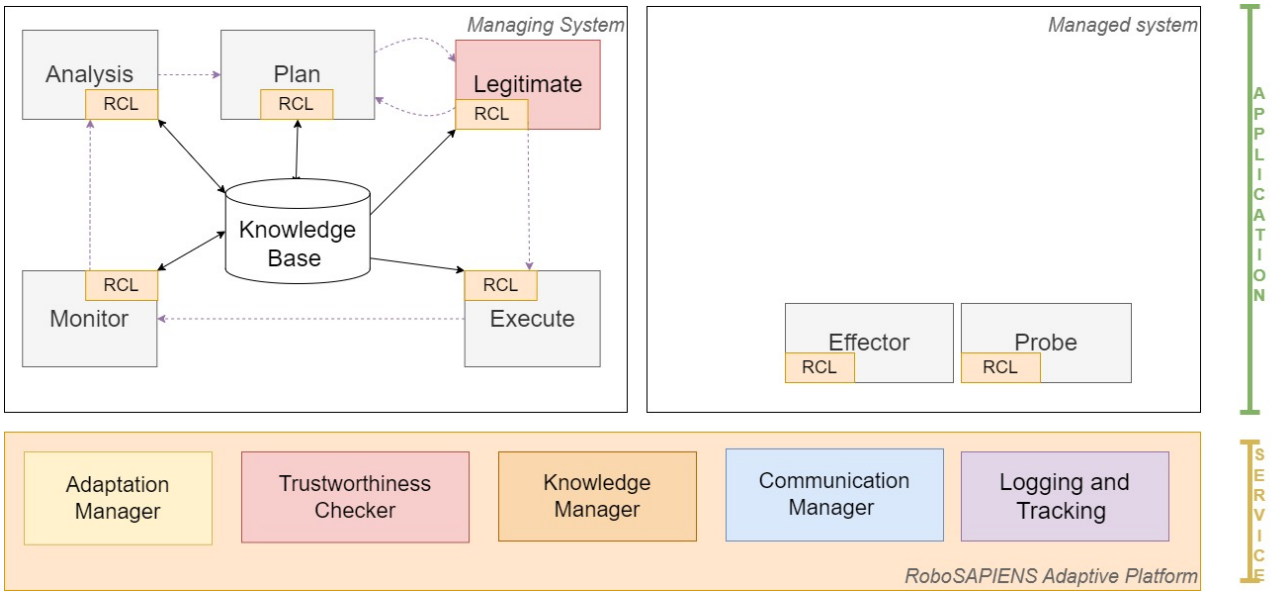


Figure 2: The RAP platform software architecture concept

As illustrated in Figure 2, the architecture consists of five main services:

1. **Adaptation Manager** orchestrates the adaptation process and all the data and event exchanges across the architecture, including event calls and receptions. It also monitors the state of the MAPE-K components and stores relevant data in the Knowledge Base to enable adaptive management.
2. **Trustworthiness Checker** assesses the trustworthiness of self-adaptation mechanisms in real time, ensuring consistent compliance with trust requirements. It also provides placeholders for custom trustworthiness checks tailored to specific use cases. The different aspects of the Trustworthiness Checker are being actively researched and discussed as part of Work Package 3.
3. **Knowledge Manager** manages all interactions with the Knowledge Base, enabling global knowledge sharing across single or multiple deployment spaces. A Redis database currently serves as its backbone for data storage and retrieval during adaptation.
4. **Communication Manager** facilitates event exchanges between architecture components using AADL-generated custom messages. It efficiently handles communication between managing and managed systems, leveraging Message Queuing Telemetry Transport (MQTT) and Redis as communication backbones, with future plans to incorporate additional protocols.
5. **Logging and tracking** ensures transparency and traceability by monitoring and recording events throughout the adaptation process.

Funded by
the European Union

3.2 RoboSAPIENS Client Library

As illustrated in Figure 2, the RoboSAPIENS client libraries are the Application Programming Interfaces (APIs) that allow users to implement their (MAPLE) software components and integrate them into the RAP in order to build adaptive system applications. Using client libraries, users gain access to RAP services, such as knowledge access, generic communication mechanisms, etc.

Currently, only a Python-based client library (rpclpy) is available, but we envisage adding client libraries in a variety of programming languages so that users may write MAPLE application code in the language that is best-suited for their application. For example, you might prefer writing the Legitimate component in Python as it makes prototyping iterations faster, while other parts of your system that are concerned with efficiency, e.g., system monitoring and analysis, the components might be better implemented in C++.

Software components written using client libraries can share knowledge and communicate with each other using custom messages, which are generated through code generators tailored to the respective programming language. In addition to the language-specific communication, client libraries expose to users the core functionality of the RAP software architecture, such as the system-wide logger or component monitor.

3.3 RoboSAPIENS staged development process

The RoboSAPIENS development process provides a comprehensive, end-to-end solution for designing and commissioning MAPE-K in robotic applications. The framework provides automated M2M and M2C transformations, enabling seamless integration of the RAP into robotic systems.

The development process takes the MAPLE-K pattern specifications as input and automatically generates the code skeleton for each MAPE-K software component. Additionally, it produces configuration files containing application-independent parameters required to run the RAP. Each of the software components uses the RAP services via the client libraries.

By combining the RAP with the RPIO toolkit for transformations, the entire process of designing and deploying a MAPLE-K loop is automated within a standardized framework, streamlining development and deployment.

Funded by
the European Union

4 robosapiensIO toolkit

The development of trustworthy, self-adaptive robotic systems requires a structured and traceable engineering process. To address this need, the RobosapiensIO provides a methodical approach that guides the development and deployment of such systems from early conceptualization to final operation. The methodology is designed to ensure safety, trustworthiness, and adaptability at every stage, while supporting iterative development and continuous refinement. By introducing clear stages, defined transitions, and integrated traceability mechanisms, RoboSAPIENS helps engineers manage complexity, mitigate risks, and maintain alignment between system requirements and implementation throughout the entire life cycle.

4.1 Overview of the RPIO framework

Figure 3 illustrates the RoboSAPIENS development and deployment process, structured into four stages: **Concept**, **Design**, **Development**, and **Deployment**. Each stage contains dedicated engineering activities that contribute to building trustworthy, self-adaptive robotic systems. Transformations, such as M2M, M2C, and various integration steps, represent transitions between activity artifacts.

A key strength of this methodology lies in its support for traceability across all stages. This traceability enhances both safety and trustworthiness by ensuring that system properties and requirements are consistently tracked throughout the development life cycle. Furthermore, the process explicitly supports both forward and backward navigation between stages, enabling iterative development and refinement. This flexibility allows system engineers to revisit earlier stages based on new insights, anomalies, or changing requirements, fostering a more robust and adaptable development process.

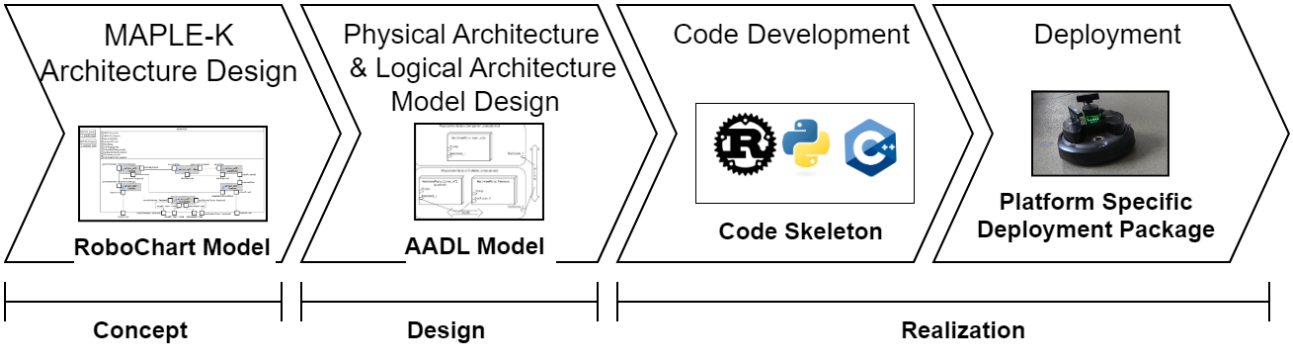


Figure 3: RoboSAPIENS development and deployment process

4.1.1 Concept stage

The purpose of this stage is verification of the MAPLE-K architecture using formal verification methods [BCCJ23] to ensure reliability and trustworthiness of the system model.

Funded by
the European Union

During the concept stage, the self-adaptive robotics system is described through both its conceptual architecture and the internal (system) and external (environment) properties that influence its behavior. The conceptual architecture is modeled using the **RoboArch notation** [BCM22], providing a compositional representation of the system's structural components. This architectural model is subsequently transformed into a **RoboChart sketch** [MRL+19], which captures the behavior of the robot's software controllers through state machines.

4.1.2 Design stage

Within the design stage, the compositional architecture serves as the foundation for defining both the operational and deployment architectures of the self-adaptive robotics system.The logical architecture can be derived through the transformation from RoboChart to AADL. The system architect then uses AADL to model the physical architectures of the system and to specify the deployment rules [FLV06]. This includes linking functional components to physical (compute) components and defining the communication matrix between these components. The configuration of the RAP software architecture, which is essential for the proper deployment and execution of the self-adaptive system components, is carried out using a custom extension of AADL. Additionally, a **Domain-specific language** is used to specify the system trustworthiness specification, an essential input for the runtime verification of trustworthiness by the **trustworthiness checker** component [BLP+25].

4.1.3 Realization stage

The realization stage consists of two important steps: user-specific application development and deployment. In the application development step, the application-independent code skeletons are transformed into application-specific MAPE-K code by the application engineers. Within the deployment step, the application code is packaged as platform-specific runnable, e.g., a Docker container, ready to be deployed and executed.

- **Application Development** within the application development stage, the application-independent software templates are transformed into application-specific skeletons, providing a foundation for application engineers to integrate their custom code for adaptive applications. These application-specific components are subsequently deployed and executed on the RAP, discussed in Section 3.
- **Deployment** the platform-specific deployment package, offers flexible options for running the code on various hardware devices through **Python launch files** or **Docker files**, with each Docker container file tailored to the specific deployment spaces defined by the physical compute units. This simplifies the distributed deployment of the framework to run each stage of the MAPLE-K in a distributed remote environment. This approach ensures that the adaptive behavior specified during earlier stages is effectively realized in the deployed system, maintaining both reliability and adaptability during runtime.

Funded by
the European Union

4.2 Transformations

The software development framework employs a model-driven development approach to streamline the transition from design to implementation. Consisting of M2M transformations and M2C transformations.

4.2.1 Concept-to-design transformation

The transformation from the concept stage to the design stage primarily involves converting RoboChart models into AADL models. This so-called **RoboChart2AADL** transformation focuses on capturing both the messages exchanged within the system and the logical architecture skeleton, ensuring that the design remains consistent with the conceptual behavior defined in RoboChart [BCCJ23]. The resulting AADL models represent the structural and behavioral aspects necessary for further refinement in the realization stage. It is our plan to validate this transformation with different RoboSAPIENS use cases.

The transformation process follows a set of well-defined transformation rules; these rules ensure that the essential semantics of RoboChart are preserved in the AADL representation. The transformation focuses on two primary areas: **behavior rules** and **connection rules**:

- The **behavior rules** concern the mapping of the RoboChart state element to corresponding constructs in AADL. In RoboChart, a state plays a dual role: it represents both a static element of the state machine (the state itself) and the dynamic behavior executed within that state.
- The **connection rules** handle the mapping of RoboChart connections, events, and variables to appropriate AADL elements. In RoboChart, events serve two primary functions: (i) *Triggering transitions between states* and (ii) *Carrying data between controllers and state machines*.

4.2.2 Design-to-Realization transformation

The transformation from the design stage to the realization stage primarily involves transforming the set of AADL models used to model the logical, physical, and deployment architecture into a **standalone deployment package**, which contains (i) application-specific software skeletons, (ii) custom messages for message-based communication, (iii) a configuration set for the RAP, and (iv) workflows to deploy and run the self-adaptive application onto the compute architecture.

Funded by
the European Union

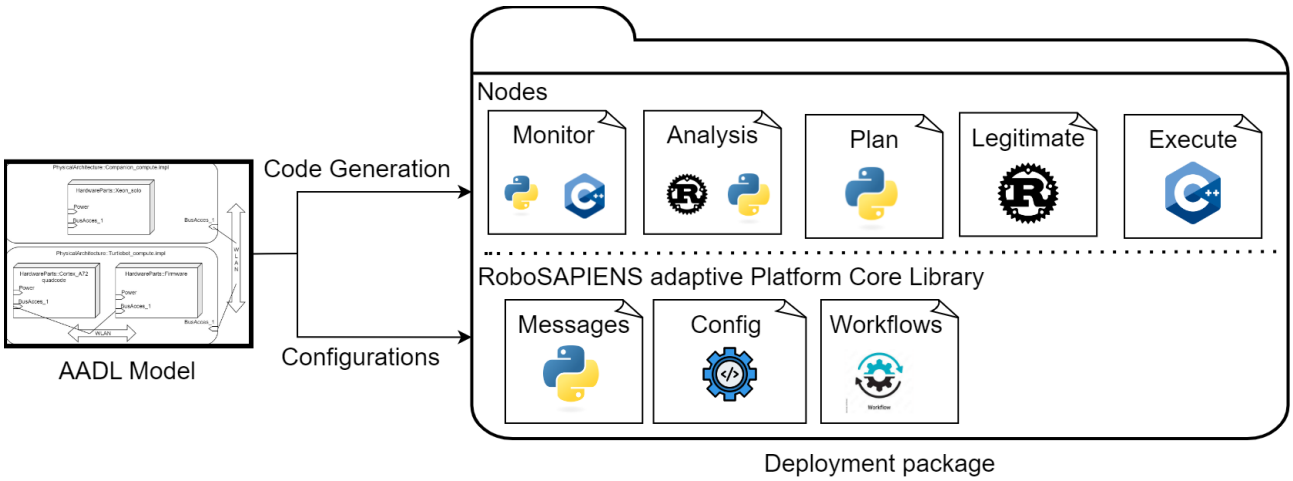


Figure 4: RAP platform-specific deployment package

By combining these elements into a single deployment package, it significantly simplifies development by providing a ready-to-use structure that aligns with the target compute architecture. Additionally, such a package enhances traceability by including references to the concept and design stages, ensuring that developers can easily track and understand the evolution of the system. This approach minimizes setup time, reduces errors, and promotes a seamless transition from design to realization.

As shown in Figure 4, the deployment package includes a standardized internal folder structure that organizes **application nodes** according to the MAPLE-K layers or managed system layers. These nodes may include Python, C/C++, and ROS2 components, among others. The package also contains **user-defined messages** specific to the platform, similar to custom messages in the ROS framework, along with platform-specific **deployment configurations**. Additionally, it provides a trustworthiness checklist and a set of automated workflows, step-by-step processes such as configuration, startup, and validation, that can be executed automatically. These workflows can be triggered using a simple **deployment tool**.

4.3 Distributed RAP software architecture

The RAP framework has transitioned from the centralized architecture introduced in the previous deliverable to a fully distributed design, enhancing scalability, modularity, and fault tolerance. This shift allows MAPE-K components to be dynamically added or removed at runtime and supports seamless deployment across multiple computational units. The framework is designed to integrate easily with widely used communication and data management tools, including MQTT, Redis, and potentially Robot Operating System 2 (ROS2), making it highly suitable for advanced DT applications.

4.3.1 Adaptation Manager

The Adaptation Manager orchestrates the adaptive behavior within the MAPE-K loop, managing the life-cycle of core components like the Communication Man-

Funded by
the European Union

ager, Knowledge Manager, and Logging and Tracking modules. This manager now supports distributed operation, allowing different components to run on separate nodes or devices. It leverages multiple communication protocols to optimize data flow and reduce latency, supporting complex distributed systems.

In this version, the Communication Manager has been split into two specialized classes. The **Event Manager** handles event-based messaging within the MAPE-K loop, typically using Redis Pub/Sub or local in-memory channels to ensure low-latency communication. The **Message Manager** is responsible for managing messages between the managing and managed systems, supporting protocols such as MQTT, TCP/IP, and RabbitMQ to enable broader compatibility.

This separation allows the MAPE-K loop to adapt to various deployment scenarios, from centralized local systems to highly distributed, edge-based architectures. To enhance reliability, each message includes a unique ID (UID) and timestamp, preventing duplicate processing and improving traceability.

4.3.2 Communication Manager

The Communication Manager has evolved from a single, shared Python class into a modular, protocol-agnostic design that supports a wide range of communication backbones. **Internal messaging** is handled through low-latency, high-throughput channels such as Redis, Kafka, or in-memory mechanisms to enable efficient inter-component coordination. For external messaging, the system supports long-distance, asynchronous communication using protocols like MQTT, UDP, and TCP/IP, with planned integration of ROS2 for robotics-specific deployments. This flexible architecture allows developers to select the most suitable protocol for each use case, ensuring optimal performance and scalability.

4.3.3 Knowledge Manager

The Knowledge Manager has similarly moved to a distributed model, replacing simple Python classes with robust, in-memory databases like Redis, Memcached, and Apache Kafka. This change supports high-speed, real-time data access and enables the storage of structured, timestamped messages. Planned features include historical data storage and model management, essential for AI-driven adaptive systems.

4.3.4 Logging and Tracking

To provide deeper insights into system behavior, the Logging and Tracking components now support multiple backends, including MQTT-based real-time logging, Redis-based event tracking, and traditional file-based logs. This multi-channel approach ensures reliable event tracing, even in highly distributed environments, and lays the groundwork for integrating live dashboards and Artificial Intelligence (AI)-driven anomaly detection.

Funded by
the European Union

4.4 Trustworthiness integration

The RoboSAPIENS project incorporates a Trustworthiness Checker as a critical component for ensuring the reliability and safety of autonomous robotic systems. This checker is implemented as a Docker container, providing flexible, modular deployment options for various robotic architectures. The checker is designed to evaluate the trustworthiness of components by monitoring their runtime behaviour and validating data streams against formally specified models.

4.4.1 Architecture overview

The trustworthiness checker operates as a standalone service, interfacing with robotic components through multiple communication protocols, including MQTT and ROS2. It supports a range of runtime models, including asynchronous and constraint-based processing, making it adaptable to different robotics use cases. The checker is configured to accept inputs from files, MQTT topics, or ROS2 topics, allowing integration with both centralized and distributed systems.

4.4.2 Data flow and model support

The trustworthiness checker is designed to allow dynamic and distributed specification of trustworthiness requirements for different MAPLE-K loops, by specifying the trustworthiness requirements for a given system in a trustworthiness specification file. The trustworthiness specification is provided in a custom Domain-Specific Language (DSL) based on the LOLA Stream-Based Verification language, meaning that it encodes trustworthiness properties as a number of monitoring rules, relating input variables pertaining to the MAPLE-K loop execution to output trustworthiness verdicts. [LMK12]

The trustworthiness checker receives each of these input variables as streams of data from the different system components. The input variables can be boolean variables, representing the firing of discrete events, or typed data variables representing other forms of data about the state of the system.

These input variables include **sequencing events**, which record the activation and completion of MAPLE-K stages; **knowledge data events**, which represent reads and writes to the knowledge base; and **stage-specific events or data**, which capture custom information generated by individual MAPLE-K stages.

Outputs can be directed to the console or published to MQTT or ROS topics, facilitating real-time monitoring and analysis. More details on trustworthiness checker can be found in [BLP+25].

4.4.3 Data types and message formats

The trustworthiness checker utilizes a standardized JavaScript Object Notation (JSON) message format for data exchange, supporting a range of data types. These include 64-bit signed integers, represented as `{"Int": 42}`; boolean values for logical judgments, such as `{"Bool": true}`; and UTF-8 encoded strings, for example `{"Str": "Hello World"}`. This consistent data representation simplifies inte-

Funded by
the European Union

gration with robotic control systems and allows for straightforward message parsing during runtime.

4.4.4 Integration with MAPE-K loop

In the context of the RoboSAPIENS project, the trustworthiness checker plays a crucial role in monitoring all phases of the MAPE-K loop. It continuously evaluates the trustworthiness of sensor data, components status and providing real-time insights to enhance the reliability and safety of autonomous operations. This integration is essential for achieving high levels of autonomy and resilience in complex, distributed robotic systems.

4.4.5 Scalability and flexibility

The checker's design allows it to scale across multiple robots or subsystems, leveraging MQTT for efficient message distribution in loosely coupled architectures. This makes it a powerful tool for real-time monitoring and decision-making in robotic fleets and heterogeneous multi-agent systems.

4.5 Feature list

Table 1 outlines the features added to the RPIO toolkit as part of this work, highlighting the advancements made since Deliverable D5.1 [ANM24]. It also presents the ongoing developments and planned enhancements targeted for inclusion in the next deliverable.

Table 1: Feature list

ID	Title	D5.1	D5.2	RPio
A-AM-1	Orchestrate adaptation	Done	-	0.3.0
A-CM-1	Communication manager-local	Done	-	0.3.0
T-D2R-1	Generate code skeleton	Done	-	0.3.0
T-D2R-2	Generate custom messages	Done	-	0.3.0
A-KM-1	Local	Done	-	0.3.0
A-LT-1	Logger-filesystem	Done	-	0.3.0
A-AM-2	Orchestrate events and messages	Todo	Done	0.4.0
A-AM-3	Add timestamp	Todo	Done	0.4.0
A-AM-4	Add unique-id	Todo	Done	0.4.0
A-CM-2	Protocol-mqtt	Todo	Done	0.3.3

Funded by
the European Union

ID	Title	D5.1	D5.2	RPio
A-CM-3	Protocol-redis	Todo	Done	0.4.0
A-CM-4	Protocol-rabbitMQ	Todo	Done	0.4.0
A-CM-5	Protocol-WebSocket	Todo	Done	0.4.0
A-CM-6	Protocol-TCP/IP	Todo	Done	0.4.0
A-CM-7	Protocol-ROS2	Todo	Todo	Later
T-C2D-1	RoboChart to AADL	Todo	Done	0.3.3
T-C2D-2	RoboChart to messages	Todo	Done	0.3.3
T-D2R-3	AADL to launch file	Todo	Done	0.3.3
T-D2R-4	AADL to main file	Todo	Done	0.3.3
T-D2R-5	AADL to docker	Todo	Done	0.3.3
T-D2R-6	RPio package using CLI	Todo	Done	0.3.3
A-KM-2	Protocol-redis	Todo	Done	0.3.3
A-KM-3	Protocol-memcached	Todo	Done	0.4.0
A-KM-4	Protocol-kafka	Todo	Done	0.4.0
A-KM-5	R/W using standard messages	Todo	Done	0.4.0
A-KM-6	R/W knowledge base-LSTM model	Todo	Todo	Later
A-KM-7	R/W historical data	Todo	Todo	Later
A-LT-2	Dashboard	Todo	Done	0.3.0
A-LT-3	Logging-redis	Todo	Done	0.4.0
A-LT-4	Add dashboard to logging	Todo	Todo	Later
A-TC-1	TC component-mqtt	Todo	Done	0.4.0
A-TC-2	TC component-redis	Todo	Done	0.4.0
M-D-1	Logical architecture	Todo	Done	0.3.3
M-D-2	Physical architecture	Todo	Done	0.3.3
M-D-3	Mapping architecture	Todo	Done	0.3.3

Funded by
the European Union

5 Use case validation of the RPIO toolkit

To validate the various features of the RPIO toolkit, we are using a range of academic and industrial use cases. These use cases serve to verify the toolkit's capabilities across different scenarios. We categorize them into four stages based on their interaction with the RPIO toolkit:

- **Exploration:** In this phase, use cases are in the process of negotiation and architectural exploration, seeking the most suitable structure for applying the MAPLE-K loop.
- **Implementation:** Here, stakeholders begin integrating the RPIO toolkit into their use cases and identifying the challenges associated with this process.
- **Evaluation:** After the first operational run of the MAPLE-K loop, we assess how effectively the RPIO toolkit supports stakeholders in implementing their MAPLE-K architecture.
- **Finalization:** In this stage, the RPIO toolkit and its automated workflow are expected to function seamlessly within the finalized use case scenario.

Case study	Case owner	Current phase
TurtleBot 4 simulator	UAntwerp	Finalization
TurtleBot 4 Lidar occlusion	AU	Evaluation
Ship motion prediction	NTNU	Implementation
Active Perception	AUTH	Implementation
Multi robot human detection	PAL	Exploration
Robot arm screw assistant	DTI	Exploration
Dynamic risk model	IFF	Exploration

Table 2: Current status of the RPIO toolkit use-cases

As you see in Table 2, the current set of use cases demonstrates a range of development stages and technical focuses, from final validation to early exploration. Some, like the TurtleBot 4 internal validation at the University of Antwerp and the Lidar occlusion tests at Aarhus University, are near completion. Others, such as the ship motion prediction led by NTNU and active perception by AUTH, are actually being integrated with the RPIO toolkit and tested in realistic environments.

Meanwhile, three use cases – multi-robot human detection (PAL), robot arm screw assistant (DTI), and dynamic risk model (IFF) – are in the *Exploration* phase. These pilots are currently being defined in close collaboration with domain experts and are expected to shape the future capabilities of the RPIO toolkit.

What follows are individual overviews of each use case, the technical problems they aim to solve, and the specific architectural features or adaptations they motivated within the RAP.

Funded by
the European Union

5.1 TurtleBot 4 simulator

Case Owner: UAntwerp *Current Phase: Finalization*

5.1.1 Description & status

This example uses a simple simulator model for the TurtleBot 4. It publishes messages that mimic the robot's Light Detection and Ranging (Lidar) output and subscribes to a navigation topic to receive updated navigation plans. The simulator allows for faster and easier testing and improvement of various features in the framework. We used it to validate key aspects of our framework. Figure 5 demonstrates the TurtleBot 4 simulator environment.

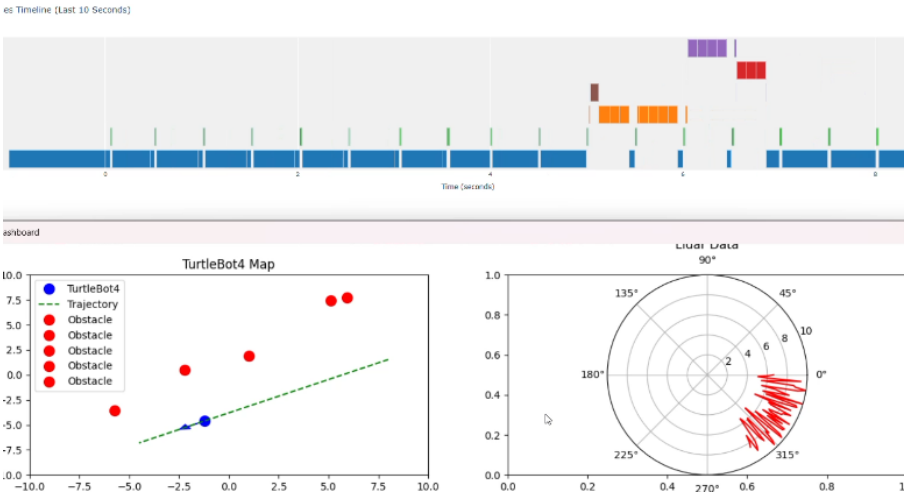


Figure 5: The TurtleBot 4 simulator for internal validations

The primary goal of the adaptation process in this example is to handle anomalies caused by deliberate occlusion of the Lidar sensor, for example, by placing an object on the robot to block its view. To manage this, we modified the navigation stack to periodically stop and rotate the robot. This behavior helps detect when the Lidar's view is blocked and adapt the navigation strategy accordingly by pausing and rotating until the obstruction is cleared.

5.1.2 Conceptual architecture

To use the RAP architecture, as Figure 6 demonstrates, we have implemented every component of the MAPLE-K loop in the following way:

- **Monitor** The monitor component collects laser scanner data to write it to the knowledge base, and then it calls the "new-data" event.
- **Analysis** The knowledge base of laser scanner data is read in order to decide whether there was an occlusion or simply an obstacle. If there was an occlusion, it writes the "Lidar-mask" data on the knowledge base and calls the "anomaly" event.
- **Plan** In case of an anomaly, the plan component triggers and reads the "Lidar-mask" data from the knowledge and then calculates the changes in the robot's

Funded by
the European Union

navigation stack, information like angle-duration of rotation. It calls the “new-plan” event, and if the plan is trusted, it writes it on the knowledge and calls the “registered-plan” event.

- **Legitimate** Listens to the “new-plan” event and checks whether the newly generated plan meets the inevitability of time limits. Then it calls the “trusted-plan” event.
- **Execute** It listens to the “registered-plan” event, and when it gets it, reads the plan from the knowledge and sends the navigation plan to the robot.

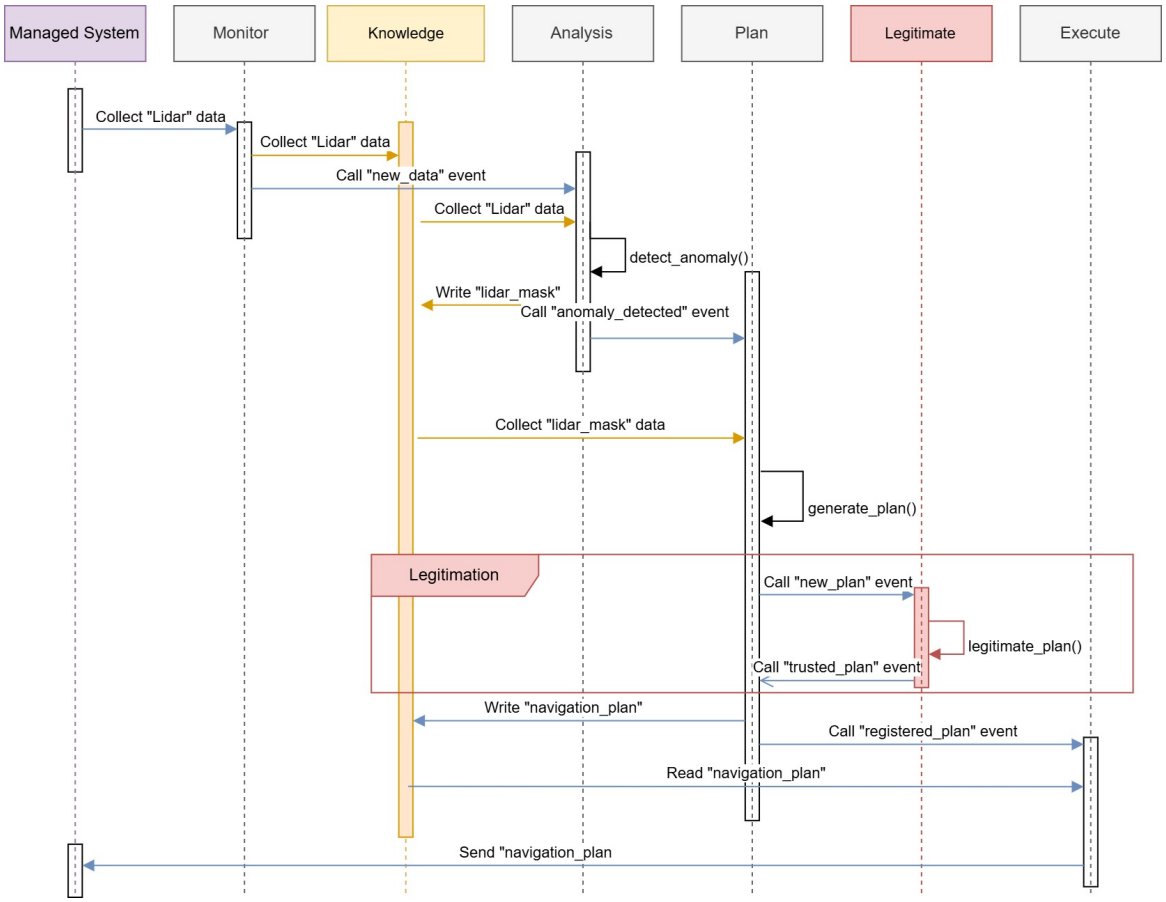


Figure 6: Sequence diagram of the TurtleBot4 use case

This TurtleBot 4 simulator is the primary environment for testing each feature of the RAP. All RAP features have been validated using this simulator.

5.2 TurtleBot 4 Lidar occlusion

Case Owner: AU *Current Phase: Evaluation*

5.2.1 Description & status

The TurtleBot4 is an advanced open source robotics platform designed for education and research. It features a Raspberry Pi 4 running ROS2, an OAK-D spatial AI stereo camera, and a 2D Lidar for robust sensor-based applications. It is built on the iRobot® Create 3 mobile base, which comes equipped with essential features

Funded by
the European Union

like an Inertial Measurement Unit (IMU), optical floor tracking, wheel encoders, and a suite of infrared sensors, all of which enable accurate localization and navigation.

The main goal of the adaptation process in this example is to handle anomalies caused by intentionally placing objects on the robot to occlude the laser scanner. The navigation stack is modified to periodically stop and rotate the robot, addressing the Lidar occlusion. The goal is to detect when the Lidar's view is obstructed and adapt the robot's movement strategy, updating the navigation plan to pause and rotate as needed to overcome the blockage.

The conceptual architecture and component development for this use case closely mirror those of the internal validation scenario; however, in this case, the implementation is applied directly to the physical TurtleBot 4 rather than the simulator.

As shown in Figure 7, the architecture was tested on an experimental setup of the TurtleBot 4 system connected with an MQTT broker and a Redis database locally. You can find the source code on GitHub.[1] In this setup, the simulator acts as the managing system, while the adaptation process runs on the managed system, hosted on a different processor, as shown in Figure 7.

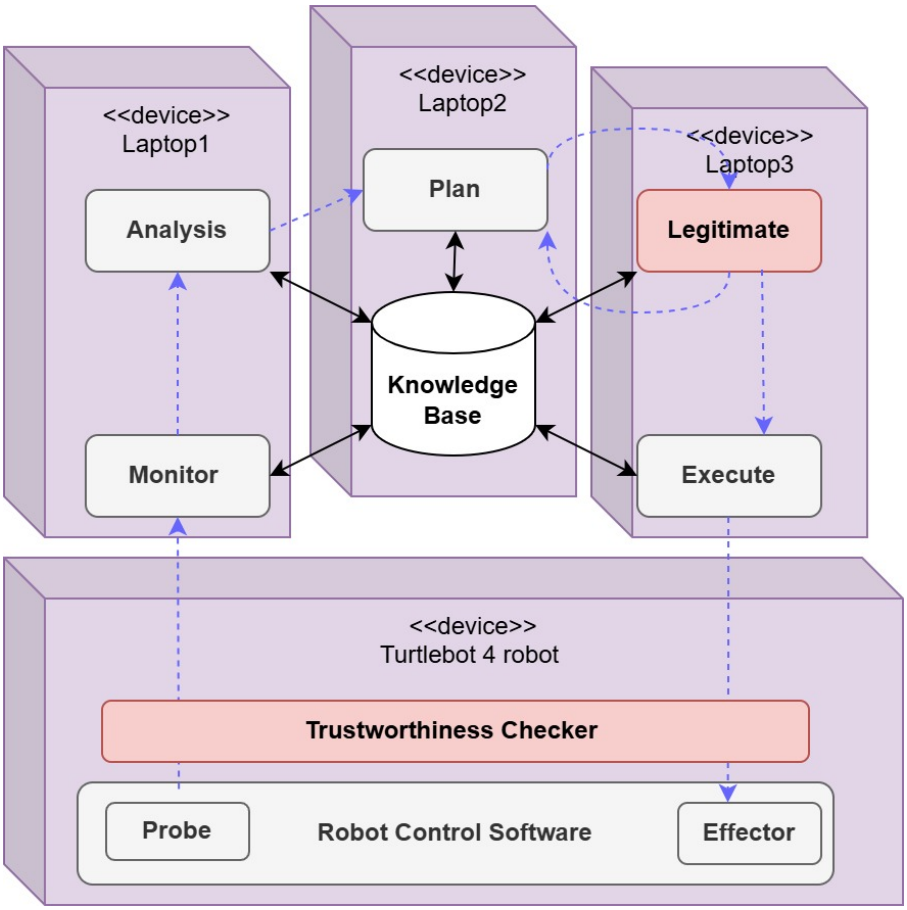


Figure 7: Distributed deployment of the TurtleBot 4 case study

[1]https://github.com/BertVanAcker/rpio-examples/tree/main/HelloWorld)

Funded by
the European Union

5.2.2 Identified challenges and lessons learned

As the first complete case study of the RPIO framework, this case study revealed a number of usability issue with initial versions of the framework which were improved in subsequent versions.

As part of this case study, custom data structures are used to represent and plan using the Lidar data. In order to facilitate serialization and storage of data within the knowledge base in a modellable way, the need to convert object to standard AADL-modelled data types became apparent.

This case study also served as a test-bed for the integration of the trustworthiness checker and the framework.

An additional issue, mentioned in Table 3, concerns the use of `MQTT` as the communication protocol. The problem was not limited to delays but also included message reordering, likely caused by the underlying broker configuration or network conditions. This behavior posed challenges for the reliable exchange of trustworthiness-related information. As a result, both the communication mechanism between the framework and the Trustworthiness Checker service and the format of trustworthiness messages were revised. These challenges significantly influenced the overall design and development of the Trustworthiness Checker [BLP+25].

Table 3 summarizes the key challenges identified in applying the RAP architecture to this case study, along with the corresponding features developed to address them.

ID	Description	Related Features
C-TB-01	Delay in logging function	A-LT-3
C-TB-02	Problem in reading some data from knowledge manager	A-KM-5, A-AM-3, A-AM-4
C-TB-03	Delay in communication manager using MQTT	A-CM-3, A-CM-4, A-AM-2

Table 3: Identified challenges in the Lidar occlusion case-study

Funded by
the European Union

5.3 Ship motion prediction

Case Owner: NTNU *Current Phase: Implementation*

5.3.1 Description and status

In the maritime domain, accurate model prediction is fundamental. The NTNU case study focuses on adapting ship motion prediction in real-time to account for disturbances and uncertainties, ensuring more precise model predictions. The primary goal of this adaptation process is to adjust the prediction model in response to anomalies, particularly environmental disturbances such as wind, waves and currents.

In our use case we present the predictions in a GUI for decision support, where we use three models to make predictions: a kinematic model, a dynamic model, and a data-driven model. The kinematic and dynamic models are mathematical models that continuously provide predictions within the decision support tool. In contrast, the data-driven Long Short-Term Memory (LSTM) model requires retraining when encountering significant environmental anomalies. During this retraining phase, the kinematic and dynamic models serve as a safe state to ensure that reliable predictions are always available. Since retraining the LSTM model requires sufficient data collection and takes time, these backup models play a crucial role in maintaining operational reliability.

For this study, a high-fidelity bridge simulator, manufactured by Kongsberg, is used as a testbed. This simulator, typically used for training nautical students and captains, offers a highly realistic environment where various conditions such as wind, waves, currents, fog, and rain can be simulated.

Currently, we have implemented the MAPLE-K loop to adapt to environmental disturbances and improve the accuracy of the LSTM-based model prediction. The model predictions are displayed in a GUI, where the LSTM model is updated when an anomaly is detected (such as high winds and waves). To ensure continuous prediction availability and maintain safety, the kinematic and dynamic models provide predictions while the LSTM model is being retrained.

The next phase involves implementing a trustworthiness checker, improving the anomaly detection, and model adaptation process.

5.3.2 Conceptual architecture

Figure 8 shows how we have implemented the MAPLE-K framework in our use case. The managed system consists of the K-Sim simulator and a framework that communicates with the simulator via WebSockets. Within this managed system, various models are used to generate predictions. Each component of the MAPLE-K framework is implemented as follows:

- **Monitor** Receives the ship status data and the variance score of the model prediction, and writes this Information in the knowledge. Then it calls the “new-data” event.
- **Analysis** Reads the variance score from the knowledge and evaluates whether

Funded by
the European Union

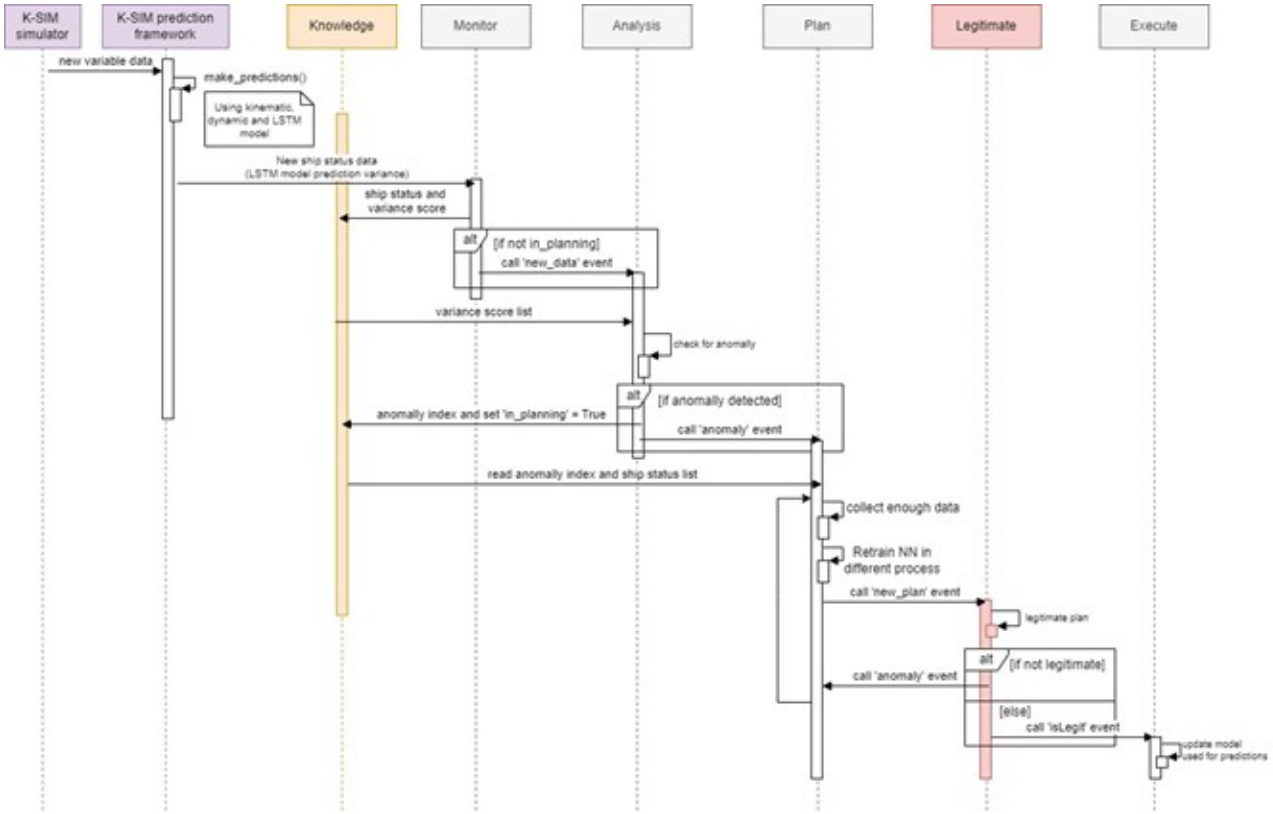


Figure 8: Sequence diagram of the NTNU use case on ship motion predictions.

it exceeds a predefined threshold for five consecutive time steps. If an anomaly is detected, it writes the time index when the anomaly was detected in the knowledge (this is crucial for the planning phase, as it determines when additional data needs to be collected for retraining the model).

- **Plan** reads the anomaly index from the knowledge component and maneuvering data for a certain period of time. After enough data is collected, the LSTM model is retrained, and when it finishes retraining it calls the event "new-plan" to legitimate the new model. Alternatively, instead of performing full retraining, AUTH is exploring the development of an anchor-based adaptation component, in which a new model learns the residuals of the initial LSTM model to refine its predictions, following the methodology presented in D2.2 [AT25].
- **Legitimate** verifies the accuracy of the newly trained model by using it to predict the vessel's trajectory based on historical data. If the variance score of this prediction exceeds the anomaly detection threshold, the model is deemed invalid, and the "anomaly" event is called to initiate a new planning phase. If the model is validated, the "IsLegit" event is called, confirming its legitimacy.
- **Execute** replaces the previous prediction model with the newly trained model by transferring it from a temporary storage location. This ensures that future predictions are based on the updated, more accurate model.

Funded by the European Union

5.3.3 Identified challenges and lessons learned

The LSTM model used to make predictions needs to be available in the managed system. However, since the retraining process occurs in the managing system, the newly trained model needs to replace the original one during the execution phase using the online learning approach. It was discussed if the models should be stored in the knowledge component. For now, the models are kept in a shared folder accessible to both the managed and managing systems, as our case does not involve a distributed system. The execute component simply transfers models from a temporary folder to the actual model folder when updating the prediction model.

Another architectural challenge was managing the waiting time in the planning component while collecting new data. This was addressed using flags stored in the knowledge base to control the process.

An initial concern was whether storing historical vessel data would create performance issues. However, the current data storage rate has proven manageable, but if the system runs for extended periods and storage becomes a bottleneck, an alternative storage approach may be necessary.

Table 4 outlines the main challenges encountered during the application of the RAP architecture in this case study, along with the solutions implemented to overcome them.

ID	Description	Related Features
C-NTNU-01	Storage of the models in the knowledge	A-KM-6
C-NTNU-02	waiting for enough collected data	A-KM-7
C-NTNU-03	integration with framework of the managed system	A-CM-6

Table 4: Identified challenges by the ship motion prediction case-study

5.4 Active perception

Case Owner: AUTH *Current Phase: Implementation*

5.4.1 Description & status

This case study builds upon and refines the initial academic active perception case study presented in [TGA+24]. It employs the TurtleBot 4 robotic platform and is developed using the Webots Robot Simulator, enabling faster and more efficient testing and validation. The TurtleBot 4 publishes data from its Lidar and camera sensors, as well as information from a SLAM system, including the robot's distance and relative angle to a predefined task objective. Leveraging this data, the robot autonomously navigates to accomplish the task using a Deep Reinforcement Learning (DRL) agent. The DRL agent processes the sensor inputs and issues the appropriate navigation commands.

Funded by
the European Union

The primary goal of the adaptation process in this case study is to handle anomalies caused by obstructions in the camera lens, such as dust clusters, mud, or other debris. This is achieved through an anomaly detection mechanism, implemented as a neural network that classifies incoming images as either normal or occluded. Upon detecting an anomaly, the robot halts and rotates to localize the obscured region in the visual field. The simulation environment is then updated in the normal cycle of the MAPLE-K to replicate the detected anomaly, and the DRL agent is fine-tuned to adapt to the altered conditions. This results in a more robust DRL agent capable of operating effectively under visual obstructions. The case study is currently being implemented in simulation, while alternative active perception strategies, alongside DRL, are also being explored.

5.4.2 Conceptual architecture

To use the RA architecture, as illustrated in Figure 9, we have designed the following MAPLE-K loop:

- **Monitor:** The monitor component collects Lidar, RGB, and localization data, writes this information to the knowledge base, and triggers the "new-data" event.
- **Analysis:** The stored camera data is preprocessed and passed to the anomaly detection network to determine whether an anomaly is present. If an anomaly is detected, the affected (anomalous) region in the camera view is inferred. The analysis component then publishes the "anomaly-area" data to the knowledge base and triggers the "anomaly" event.
- **Plan:** Upon anomaly detection, the plan component is activated. It retrieves the "anomaly-area" data from the knowledge base, simulates the anomaly in the simulator, and fine-tunes the control agent until a predefined success metric is achieved. It then triggers the "new-plan" event. If the plan meets validation criteria, it is published to the knowledge base and the "registered-plan" event is triggered.
- **Legitimate:** This component evaluates the fine-tuned agent within a simulated environment containing randomized obstacles and navigation goals. Once validated, the updated weights of the agent are transferred to the robot's control agent.
- **Execute:** This component subscribes to the "agent-command" event. Upon receiving the event, it sends the appropriate control commands to the robot for execution.

Funded by
the European Union

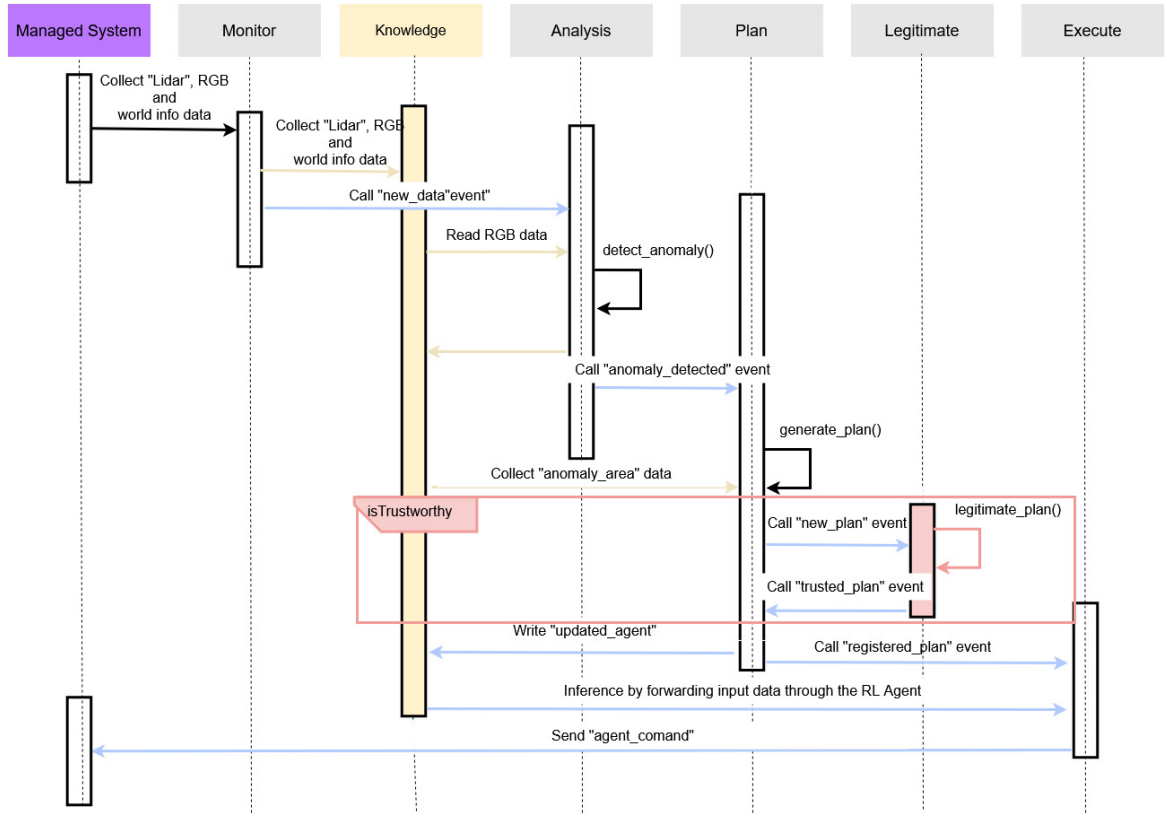


Figure 9: Sequence diagram of the Active Perception Case Study

5.5 Multi robot human detection

Case Owner: PAL *Current Phase: Exploration*

5.5.1 Description & status

In this use-case we are designing a self-adaptive, distributed robotic system using PAL Robotics' omni-directional robots. The system will operate under a distributed architecture with a shared global knowledge base and multiple autonomous agents.

- **Phase 1: TurtleBot4 scenario on omni-Based robot** To validate our architecture, we will first replicate a TurtleBot4 use case on an omni-based robot. This will help test the modular architecture, identify integration challenges, and evaluate how the self-adaptive logic transfers to PAL's omni-based platform.

Funded by the European Union

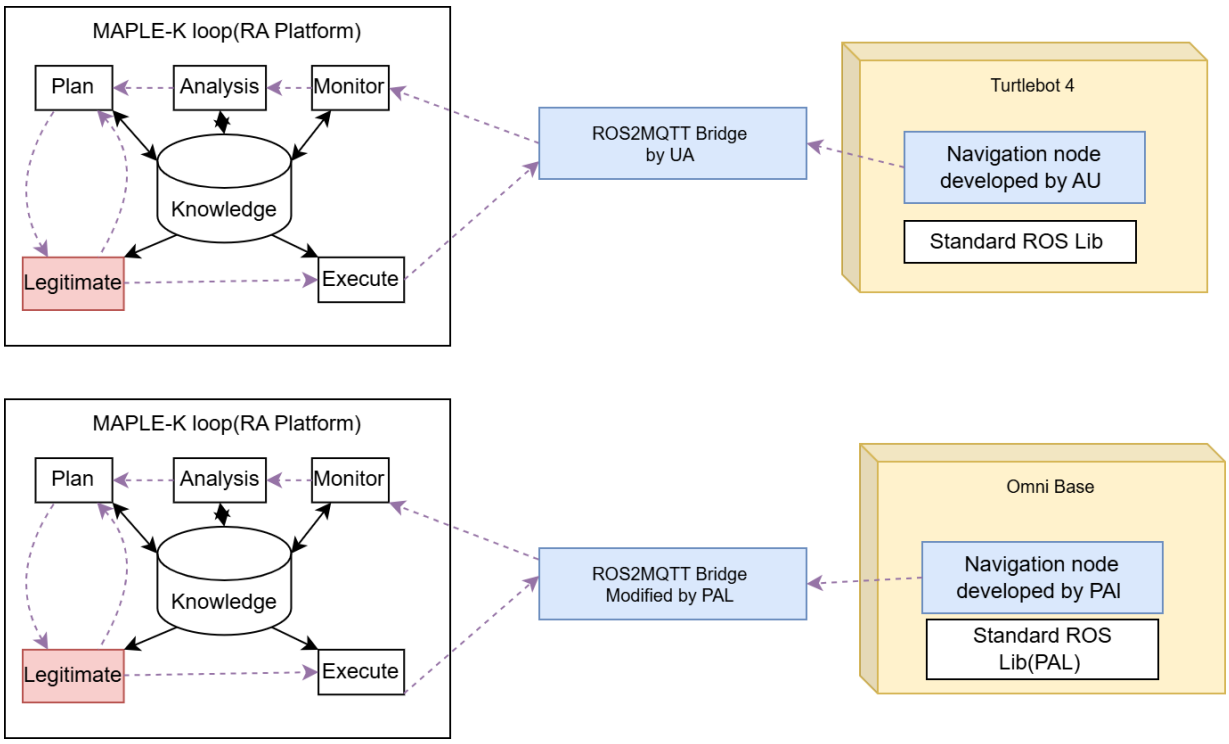


Figure 10: Lidar occlusion example integration with Omni-based robot

- **Phase 2: Warehouse use case with human detection and multi-robot coordination** The target use case is a warehouse scenario. The system will detect human presence and motion, update robot behavior dynamically, and react in real time to ensure coordinated, safe operation.

5.5.2 Conceptual architecture

As shown in Figure 11, the system follows a distributed MAPLE-K loop, with components distributed across robots and a fleet manager.

Funded by
the European Union

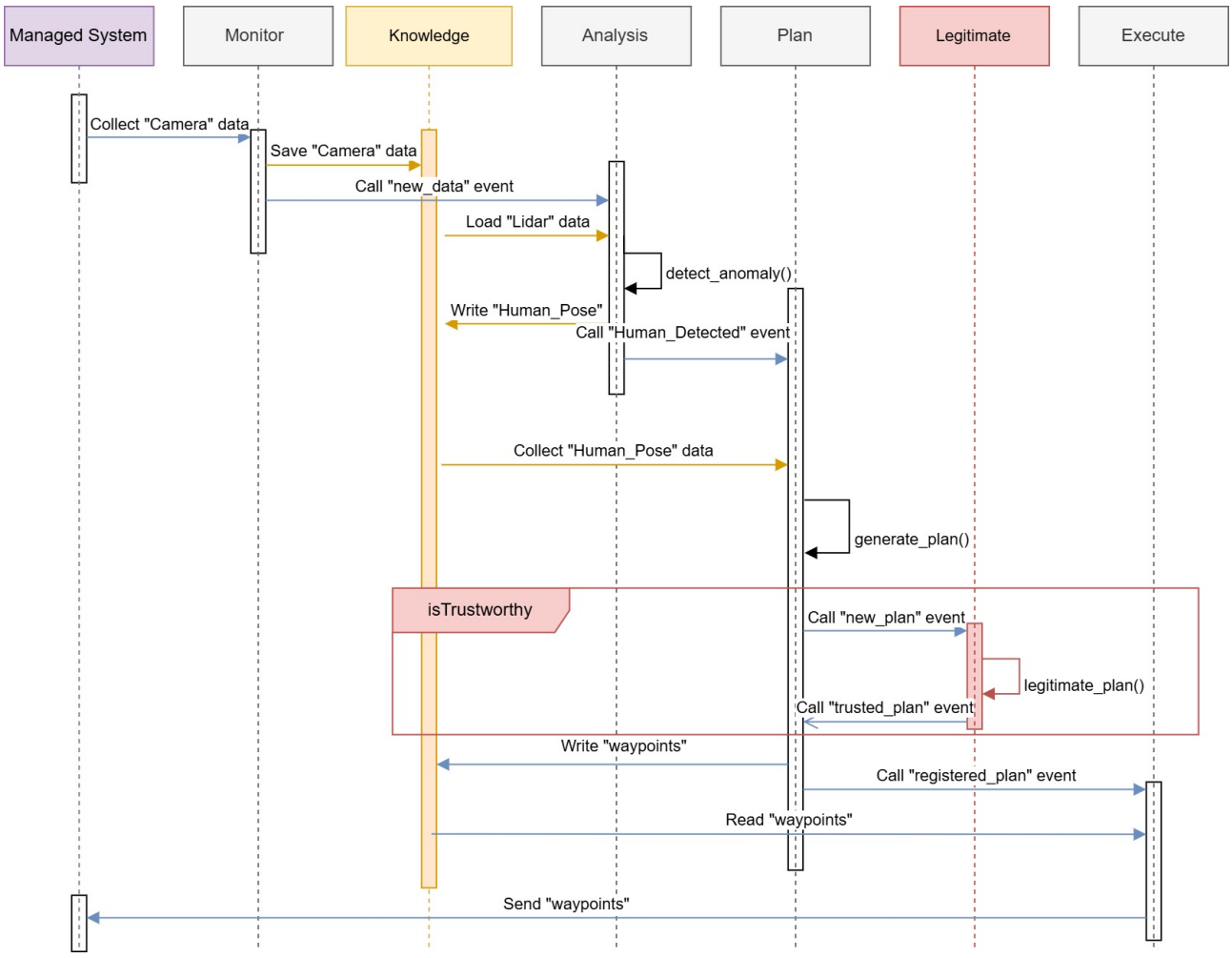


Figure 11: Sequence diagram of the human detection use-case

- **Monitor:** Each robot collects its own sensor data (e.g., camera feeds).
- **Analyze:** The fleet manager detects human position and velocity using the shared knowledge base. Human detection is initially performed in camera frames (2D space) using a lightweight variant of YOLO11 (specifically YOLO11s or YOLO11n). To ensure robust and accurate performance within the warehouse use case, the detector has been fine-tuned on lower-body human imagery. More details are provided in Section 5.6 of D2.2.
- **Plan:** The fleet manager generates and updates robot waypoints based on human movement.
- **Legitimize:** The fleet manager checks robot battery levels to ensure the plan is executable.
- **Execute:** Each robot carries out the new waypoints sent by the fleet manager.
- **Knowledge:** A shared global state is maintained across all agents via a centralized or distributed server.

Figure 12 illustrates that this setup uses decentralized execution with centralized

Funded by
the European Union

reasoning: each robot handles sensing and actuation locally, while the fleet manager coordinates planning and decision-making.

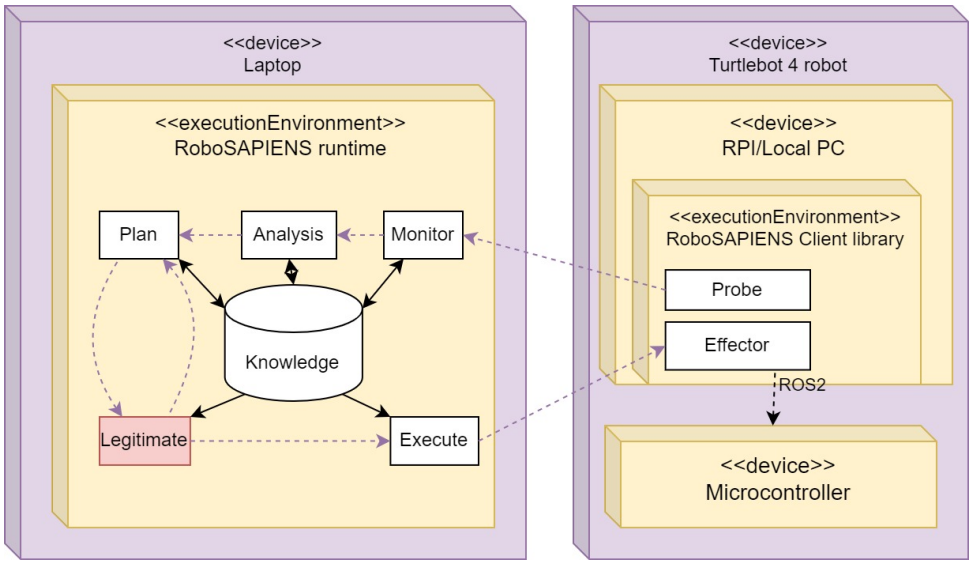


Figure 12: Distributed architecture of the MAPLE-K loop in human detection use-case

5.5.3 Challenges and focus areas

Distributed deployment means components run on different hardware and network nodes, requiring robust communication. Coordination must be timely, especially in dynamic human environments. A key challenge is keeping the shared knowledge consistent across agents. The system also needs to scale efficiently as more robots are added.

5.6 Robot arm screw assistant

Case Owner: DTI *Current Phase: Exploration*

5.6.1 Description & status

This case study focuses on assisting a robotic arm with screwing and unscrewing tasks in refurbished laptops. The initial scenario under consideration involves the robot failing to locate the exact position of a screw. In such cases, the self-adaptive system will trigger a search behavior – such as spin or random search – to find the screw's location.

We are currently in close discussion with the use-case stakeholders to better understand and define the full range of challenges. Some of the emerging issues include:

- Visual detection difficulties due to lighting changes or dust on the camera.
- Mechanical anomalies, such as unexpected load variations or motor aging.
- Potential faults that could affect precision, timing, or force application.

Funded by
the European Union

These anomalies suggest that the use of a self-adaptive architecture, powered by the MAPLE-K loop, could be extended to address a wider variety of operational uncertainties.

5.6.2 Conceptual architecture

Although the MAPLE-K loop is not fully mapped in this case yet, it is likely to evolve to include components for:

- **Monitor** camera input and robot joint states.
- **Analysis:** visual inconsistencies or deviations in robot performance.
- **Plan:** fallback behaviors (e.g., search strategies or retries).
- **Legitimize:** steps based on internal health indicators (e.g., load, temperature, or historical failure rates).
- **Execute:** adapted actions in real time.
- **Knowledge:** Storing data about failures and anomaly patterns to inform future adaptation.

This case is expected to expand as further integration challenges and edge cases are explored. It highlights how adaptive reasoning can enhance robustness in semi-structured, uncertain environments.

5.7 Dynamic risk model

Case Owner: IFFF *Current Phase: Exploration*

5.7.1 Description & status

This case study focuses on a human approaching a Universal UR10 robotic arm. The goal is to dynamically assess safety risk by using a self-adaptive system that updates human body part volumes in real time. This allows the system to respond more intelligently to proximity and motion, improving human-robot interaction safety.

5.7.2 Conceptual architecture

The MAPLE-K loop is adapted to this safety-focused context as follows:

- **Monitor:** Capture live camera data to observe human presence.
- **Analyze:** Estimate keypoints of the human body using pose detection techniques.
- **Plan:** Calculate body part volumes dynamically based on pose and proximity.
- **Legitimize:** Compare current body part angles and distances to historical data to assess risk or anomaly.
- **Execute:** Send updated volume and risk information to the robot control application for adaptive response.

Funded by
the European Union

- **Knowledge:** Store historical violation data, including timestamps and severity levels, to improve risk modeling over time.

This use case emphasizes real-time risk evaluation by adapting safety thresholds based on observed human posture and movement trends. It extends the self-adaptive architecture into the domain of safety-aware robotic systems.

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.

[ANM24] Bert Van Acker, Sahar Nasimi Nezhad, and Paul De Meulenaere. Initial architecture for the RoboSAPIENS platform. Technical report, RoboSAPIENS Deliverable, D5.1, September 2024.

[AT25] Nikolaos Passalis et al. Anastasios Tefas, Nikolaos Nikolaidis. Self-adaptation and initial safety integration. Technical report, RoboSAPIENS Deliverable, D2.2, September 2025.

[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.

[BLP+25] Yannick Bollmann, Guoyuan Li, Thomas Peyrucain, Peter Gorm Larsen, Roland Behrens, Robert Scharping, Morten Haahr Kristensen, Wright Thomas, Claudio Gomes, Lukas Esterle, Carlos Isasa, Loukia Avramelou, Dimitrios Akrivousis, Dimitrios Katsikas, Vasileios Moustakidis, Anastasios Tefas Pavlos-Apostolos, Tosidis, Nikolaos Nikolaidis, and Nikolaos Passalis. Monitorable and trustworthy verification loops. Technical report, RoboSAPIENS Deliverable, D3.2, September 2025.

[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, pages 1206–1211. IEEE, 2006.

[LMK12] Christoph Lange, Till Mossakowski, and Oliver Kutz. Lola: A modular ontology of logics, languages, and translations. In Workshop on Modular Ontologies (WoMO) 2012, page 51, 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.

[TGA+24] Anastasios Tefas, Claudio Gomes, Dimitrios Akrivousis, Dimitrios Katsikas, Erblin Isaku, Jiahui Wu, Loukia Avramelou, Nikolaos Nikolaidis, Nikolaos Passalis, Pavlos Tosidis, Shaukat Ali, Tongtong Wang, Vasileios Kanatas, Vasileios Moustakidis, and Yannick Bollmann. Active learning and system monitoring. Technical report, RoboSAPIENS Deliverable, D2.1, September 2024.

Funded by
the European Union

A Publications

Publications that resulted from the work conducted so far in WP5:

1. James Baxter, Bert Van Acker, Morten Kristensen, Thomas Wright, Ana Cavalcanti, Cláudio Gomes. "*Formal Architectural Patterns for Adaptive Robotic Software*". In *International Conference on Fundamental Approaches to Software Engineering*, 145-165. Cham: Springer Nature Switzerland, 2025.
2. Sahar Nasimi Nezhad, Bert Van Acker, and Paul De Meulenaere. "*Towards a Standardized Framework for Developing Trustworthy Self-Adaptive Robotic Systems*", *Annual Modeling and Simulation Conference (ANNSIM)*, Madrid, Spain, 2025.
3. Sahar Nasimi Nezhad, Bert Van Acker, and Paul De Meulenaere. "*Toward the Trust-Enhanced MAPE-K Loop: A Novel Robotic Software Architecture*", *International Conference on Cyber-Physical Systems (ICCPS)*, 2025.

Funded by
the European Union

B RPIO toolkit features

A-AM-1 : Orchestrate adaptation

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Adaptation Manager	Done	-	0.3.0
Description:	Orchestrate adaptation process between components the adaptation process also adds message headers to the messages that are being sent from different components			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator , TurtleBot 4 Lidar occlusion			
Linked requirements:	-			

A-CM-1 : Communication manager-local

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Communication Manager	Done	-	0.3.0
Description:	Communication between MAPLE-K components is facilitated through a modular messaging infrastructure that supports both synchronous data exchange.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator, TurtleBot 4 Lidar occlusion , Ship motion prediction			
Linked requirements:	ID_RAP_08			

T-D2R-1 : Generate code skeleton

Level	Group	D5.1 status	D5.2 status	RPIO version
Transformations	Design to Realization	Done	-	0.3.0
Description:	The system provides functionality to automatically generate software component code skeletons from AADL specifications. This process translates architectural elements such as threads, data ports, and connections into structured source code templates. These skeletons serve as a foundation for implementation, ensuring alignment with the system architecture while accelerating development and reducing the risk of structural inconsistencies.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_11			

T-D2R-2 : Generate custom messages

Funded by
the European Union

Level	Group	D5.1 status	D5.2 status	RPIO version
Transformations	Design to Realization	Done	-	0.3.0
Description:	The system supports the automatic generation of custom messages from AADL specifications. This feature extracts interface definitions and data structures from the AADL model and translates them into standardized message formats used by the runtime system. It ensures consistency between architectural design and implementation, reduces manual coding effort, and facilitates seamless integration across components.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_11			

A-KM-1 : local

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Knowledge Manager	Done	-	0.3.0
Description:	The system employs a local knowledge base as a shared knowledge layer accessible by multiple components. This local store enables fast, consistent data sharing without relying on external communication.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator, TurtleBot 4 Lidar occlusion , Ship motion prediction			
Linked requirements:	-			

A-LT-1 : Logger-filesystem

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Logging And Trancking	Done	-	0.3.0
Description:	The system incorporates a file-system-based logger to record the status of components. This logger writes status updates to structured log files, enabling persistent, local tracking of component behavior over time.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 Lidar occlusion , TurtleBot 4 simulator, Ship motion prediction			
Linked requirements:	ID_RAP_09			

A-AM-2 : orchestrate events and messages

Funded by
the European Union

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Adaptation Manager	Todo	Done	0.4.0
Description:	The Communication Manager has been refactored to separate inter-component and intra-component communication into distinct modules. The intra-component communication manager handles low-latency messaging within individual system components, often using in-memory channels. In contrast, the inter-component communication manager manages message exchanges between distributed components.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_08			

A-AM-3 : Add timestamp

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Adaptation Manager	Todo	Done	0.4.0
Description:	Messages handled by both the Communication Manager and the Knowledge Manager are now timestamped at the point of creation or transmission. This timestamping enables accurate temporal tracking of events and data exchanges across the system. It supports time-based analysis, synchronization, and auditing, enhancing the system s ability to reason about the sequence and timing of operations in distributed environments.			
Linked challenges:	C-TB-02			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	-			

A-AM-4 : Add unique-id

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Adaptation Manager	Todo	Done	0.4.0
Description:	Each event message in the system is now appended with a random unique identifier. This unique ID ensures traceability and distinction between messages, even when they carry identical content or occur in rapid succession. It enhances debugging, auditing, and correlation of events across distributed components, supporting more reliable and transparent system behavior analysis.			
Linked challenges:	C-TB-02			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	-			

A-CM-2 : protocol-mqtt

Funded by
the European Union

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Communication Manager	Todo	Done	0.3.3
Description:	Support MQTT, S lightweight publish/subscribe protocol optimized for low-bandwidth and high-latency networks, ideal for IoT devices and constrained environments.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator , TurtleBot 4 Lidar occlusion , Ship motion prediction			
Linked requirements:	ID_RAP_08			

A-CM-3 : protocol-redis

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Communication Manager	Todo	Done	0.4.0
Description:	Support Redis Pub/Sub an in-memory data store s publish/subscribe feature, providing ultra-low-latency messaging for rapid event distribution and transient notifications.			
Linked challenges:	C-TB-03			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_08			

A-CM-4 : protocol-rabbitMQ

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Communication Manager	Todo	Done	0.4.0
Description:	Support RabbitMQ (AMQP)a robust message broker implementing the Advanced Message Queuing Protocol, offering reliable message delivery, flexible routing, and complex queuing patterns.			
Linked challenges:	C-TB-03			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_08			

A-CM-5 : protocol-WebSocket

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Communication Manager	Todo	Done	0.4.0
Description:	Support WebSocket a full-duplex communication channel over a single TCP connection, enabling real-time, bidirectional data exchange between clients (e.g., browsers) and servers.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_08			

A-CM-6 : protocol-TCP/IP

Funded by
the European Union

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Communication Manager	Todo	Done	0.4.0
Description:	TCP/IP: The foundational protocol suite for reliable, ordered, and error-checked byte-stream communication between networked hosts, serving as the basis for many higher-level protocols.			
Linked challenges:	C-Ship motion prediction-02			
Validated by:	Not validated			
Linked requirements:	ID_RAP_08			

A-CM-7 : protocol-ROS2

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Communication Manager	Todo	Todo	Later
Description:	Support ROS 2 a robotics middleware built on DDS that provides decentralized, peer-to-peer publish/subscribe, services, and actions with configurable QoS settings and enabling real-time, reliable communication tailored for distributed robotic systems.			
Linked challenges:	Generic			
Validated by:	Not validated			
Linked requirements:	ID_RAP_08			

T-C2D-1 : roboChart to AADL

Level	Group	D5.1 status	D5.2 status	RPIO version
Transformations	Concept to Design	Todo	Done	0.3.3
Description:	The system enables automatic generation of an AADL logical architecture from RoboChart models. This process extracts structural and behavioral information defined in RoboChart and maps it into AADL components, connections, and data flows. By translating high-level robotic specifications into a formal architectural model, this feature ensures alignment between design and system architecture, supporting early validation, analysis, and streamlined code generation.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_10			

T-C2D-2 : roboChart to messages

Funded by
the European Union

Level	Group	D5.1 status	D5.2 status	RPIO version
Transformations	Concept to Design	Todo	Done	0.3.3
Description:	The system supports the automatic generation of standard AADL messages from RoboChart models. This feature translates formal behavioral specifications into message definitions that are compatible with AADL-based architectures. It ensures consistency between high-level robotic behavior modeling and the underlying system communication framework, facilitating seamless integration and reducing the risk of interface mismatches during implementation.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_10			

T-D2R-3 : AADL to launch file

Level	Group	D5.1 status	D5.2 status	RPIO version
Transformations	Design to Realization	Todo	Done	0.3.3
Description:	The system can automatically generate software component launch files from an AADL (Architecture Analysis and Design Language) specification. This capability ensures that each component is initialized with the correct parameters, execution context, and communication bindings as defined in the architectural model. By automating the generation of launch files, the process reduces manual configuration, aligns implementation with design, and accelerates deployment in distributed environments.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_11			

T-D2R-4 : AADL to main file

Level	Group	D5.1 status	D5.2 status	RPIO version
Transformations	Design to Realization	Todo	Done	0.3.3
Description:	The system includes functionality to automatically generate the MAPLE-K main file from an AADL (Architecture Analysis and Design Language) specification. This automation bridges model-based design with system implementation, ensuring that the generated main file reflects the defined architecture, component interactions, and data flows. It streamlines the deployment process, reduces manual errors, and promotes consistency between the architectural model and runtime behavior.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_11			

T-D2R-5 : AADL to docker

Funded by
the European Union

Level	Group	D5.1 status	D5.2 status	RPIO version
Transformations	Design to Realization	Todo	Done	0.3.3
Description:	The system supports generating a distributed deployment package using Docker. This packaging approach encapsulates each component in a container, ensuring consistent environments across different machines. It enables seamless deployment, scaling, and orchestration of services in distributed setups, while simplifying dependency management and improving portability across development, testing, and production environments.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_11			

T-D2R-6 : RPio package using CLI

Level	Group	D5.1 status	D5.2 status	RPIO version
Transformations	Design to Realization	Todo	Done	0.3.3
Description:	The system provides a command line interface for creating RPio packages. This interface allows developers to generate and configure RPio package structures efficiently, specifying required components, dependencies, and metadata directly from the terminal. It streamlines the setup process, supports automation in development pipelines, and ensures consistency across different RPio-based deployments.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_11			

A-KM-2 : protocol-redis

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Knowledge Manager	Todo	Done	0.3.3
Description:	The knowledge manager utilizes a shared, distributed knowledge base implemented with Redis. Redis is an open-source, in-memory data store that supports a variety of data structures, including strings, hashes, lists, and sets. It offers microsecond-level latency, rich data type support, built-in persistence, and publish/subscribe capabilities, making it highly suitable for real-time knowledge sharing across distributed components.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 Lidar occlusion , Ship motion prediction, TurtleBot 4 simulator			
Linked requirements:	ID_RAP_08			

A-KM-3 : protocol-memcached

Funded by
the European Union

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Knowledge Manager	Todo	Done	0.4.0
Description:	Support for Memcached. Memcached is a simple, high-performance, in-memory key-value cache that offers basic operations such as get, set, and delete. Its advantages include ultra-lightweight operation, minimal overhead, and exceptional performance for caching and high-throughput workloads.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_08			

A-KM-4 : protocol-kafka

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Knowledge Manager	Todo	Done	0.4.0
Description:	Support for Apache Kafka is included as part of the communication infrastructure. Kafka is a distributed commit-log streaming platform that treats messages as durable, replayable writes. Its key advantages include massive throughput, long-term message retention, and support for decoupled, replayable consumers, making it well-suited for high-volume, event-driven architectures.			
Linked challenges:	Generic			
Validated by:	Not validated			
Linked requirements:	ID_RAP_08			

A-KM-5 : read/write using standard messages

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Knowledge Manager	Todo	Done	0.4.0
Description:	The system supports reading and writing knowledge using standardized message formats. This approach ensures consistency and interoperability across components, enabling reliable exchange and synchronization of information within the distributed architecture.			
Linked challenges:	Generic			
Validated by:	Not validated			
Linked requirements:	ID_RAP_08			

A-KM-6 : r/w knowledgebase-LSTM model

Funded by
the European Union

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Knowledge Manager	Todo	Todo	Later
Description:	The system supports saving LSTM (Long Short-Term Memory) models directly into the knowledge base. This enables persistent storage of trained models within the distributed system, allowing components to retrieve and reuse learned behaviors or patterns as part of decision-making and adaptive control processes.			
Linked challenges:	C-Ship motion prediction-01			
Validated by:	Not validated			
Linked requirements:	ID_RAP_02			

A-KM-7 : r/w historical data

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Knowledge Manager	Todo	Todo	Later
Description:	The system enables reading and writing of historical data within the knowledge base. This capability allows components to access past system states or events, supporting tasks such as trend analysis, anomaly detection, and retrospective decision-making.			
Linked challenges:	C-Ship motion prediction-02			
Validated by:	Not validated			
Linked requirements:	ID_RAP_08			

A-LT-2 : Dashboard

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Logging And Tracking	Todo	Done	0.3.0
Description:	A visualized dashboard is integrated into the system to display the activation status of various components. This dashboard provides real-time insights into system behavior, highlighting which modules are active, idle, or experiencing issues. It enhances observability and aids in monitoring, debugging, and system validation during runtime.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 Lidar occlusion , TurtleBot 4 simulator			
Linked requirements:	ID_RAP_09			

A-LT-3 : Logging-redis

Funded by
the European Union

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Logging And Tracking	Todo	Done	0.4.0
Description:	The logging and tracking subsystem, implemented with Redis, supports a faster logging protocol to improve performance. This enhancement ensures low-latency, high-throughput capture of runtime events, enabling efficient traceability and real-time system introspection without introducing significant overhead.			
Linked challenges:	TB-01			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	ID_RAP_09			

A-LT-4 : Add dashboard to logging

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Logging And Trancking	Todo	Todo	Later
Description:	The Logging and Tracking component incorporates a visualized dashboard to present real-time status and activity of system components. This integration enables users to monitor component activation states, track events, and analyze runtime behavior through an intuitive graphical interface. By combining fast logging protocols with visual feedback, the system enhances observability, simplifies debugging, and supports effective validation during operation.			
Linked challenges:	Generic			
Validated by:	Not validated			
Linked requirements:	ID_RAP_09			

A-TC-1 : TC component-mqtt

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Trustworthiness Checker	Todo	Done	0.4.0
Description:	The first version of the trustworthiness checker has been integrated with support for MQTT-based communication. This integration allows the checker to receive and evaluate runtime data published over MQTT, enabling real-time assessment of system integrity and behavior. By leveraging MQTT's lightweight and asynchronous messaging model, the trustworthiness checker can efficiently interact with distributed components across varying network conditions.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 Lidar occlusion , TurtleBot 4 simulator			
Linked requirements:	503			

A-TC-2 : TC component-redis

Funded by
the European Union

Level	Group	D5.1 status	D5.2 status	RPIO version
Architecture	Trustworthiness Checker	Todo	Done	0.4.0
Description:	To enhance performance, the trustworthiness checker now supports faster communication using Redis. This optimization leverages Redis s low-latency, in-memory data exchange to enable rapid delivery and processing of runtime data. As a result, the checker can perform real-time trust assessments with minimal delay, improving system responsiveness and reliability.			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 simulator			
Linked requirements:	503			

M-D-1 : Logical architecture

Level	Group	D5.1 status	D5.2 status	RPIO version
Modeling	Design	Todo	Done	0.3.3
Description:	Enables robotics engineers to explicitly model the logical architecture of the self-adaptive system, by means of the MAPLE-K components and their inter- and intra-component communication			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 Lidar occlusion			
Linked requirements:	ID_RAP_04			

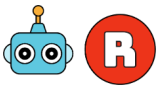
M-D-2 : Physical architecture

Level	Group	D5.1 status	D5.2 status	RPIO version
Modeling	Design	Todo	Done	0.3.3
Description:	Enables robotics engineers to explicitly model the physical architecture of the self-adaptive system, by means of the interconnected compute devices, executing (parts of) the MAPLE-K			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 Lidar occlusion			
Linked requirements:	ID_RAP_05			

M-D-3 : Mapping architecture

Level	Group	D5.1 status	D5.2 status	RPIO version
Modeling	Design	Todo	Done	0.3.3
Description:	Enables robotics engineers to explicitly model the mapping of the logical architecture on the physical architecture, enabling deployment reasoning and deployment-specific configuration			
Linked challenges:	Generic			
Validated by:	TurtleBot 4 Lidar occlusion			
Linked requirements:	ID_RAP_06			

Funded by
the European Union



Funded by
the European Union

C RAP and RPIO requirements

ID	Name	Description	RoboSAPIENS Requirement
ID_RAP_01	Supported computing platforms	The RAP shall support both resource-constraint platforms and high performance computing platforms, both local (edge node) and remote (cloud).	-
ID_RAP_02	Online updating and evolution of MAPL-K components	The RAP shall support online updating of software components at runtime. This includes the ability to incrementally adapt or replace components, such as theDT model or other runtime modules, based on newly available data, user feedback, or evolving system needs. This capability enables the continuous evolution and long-term adaptability of the system without requiring downtime or manual redeployment.	-
ID_RAP_03	Flexible data and software update	The RAP 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 RAP shall support multiple standardized commu- nication protocols for (bi-directional) inter- and/or intra-device communication with different network topologies.	-

Table 5: High-level requirements

Funded by the European Union

ID	Name	Description	RoboSAPIENS Requirement
ID_RAP_09	Transparency	The RAP shall provide diagnostics means during run-time, providing explainability on the robots data, actions and decisions in a transparent way	-
ID_RAP_10	Concept-to-design	The RPIO toolkit shall provide a mechanism to transform the RoboWorld-specific models, used in conceptual phase, into AADL models, used in design phase.	ID500
ID_RAP_11	design-to-realization	The RPIO toolkit shall provide a mechanism to transform the AADL models, describing the self-adaptive system and it's deployment, to deployment-specific code, ready to deploy on one or more compute units.	ID501;ID503

Table 6: High-level requirements(continued)

Funded by
the European Union

D 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