

# ROBOSAPIENS

## D5.3

# The RoboSAPIENS Adaptive Platform - Platform Evolutions and Workflow

## WP5

**Public Document**

| | |
|---|---|
| **Grant Agreement** | 101133807 |
| **Project** | RoboSapiens |
| **Deliverable Number** | D5.3 |
| **Version** | 1.0 |
| **Due Month** | 27 |
| **Date** | February 2026 |

**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
Arkadiusz Rys, UAntwerp
Paul De Meulenaere, UAntwerp
Pavlos Tosidis , AUTH
Nikolaos Passalis , AUTH
Charalampos Symeonidis , AUTH
Anastasios Tefas , AUTH
Nikolaos Nikolaidis , AUTH
Avramelou Loukia , AUTH

## Editors:

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

## Internal reviewers:

Nikolaos Passalis, AUTH
James Baxter, UoY

## 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 | 20-10-2025 | Sahar Nasimi Nezhad, Bert Van Acker | Initial document version |
| 0.2 | 18-02-2026 | Sahar Nasimi Nezhad, Bert Van Acker, Arkadiusz Rys | Content on work-flows |
| 1.0 | 20-03-2026 | Sahar Nasimi Nezhad, Bert Van Acker, Arkadiusz Rys | Updated content according to internal review |

Funded by
the European Union

# Abstract

This deliverable (D5.3) outlines the progress made in Work Package 5 (WP5) between month 19 (M19) and month 27 (M27).

Rather than introducing major new features to the RAP software architecture or the robosapiensIO (RPIO) toolkit, the work during this period focuses on consolidating and integrating existing capabilities within the RAP Workflow. The deliverable introduces the RoboSAPIENS Workflow and its supporting workflow tool, this way presenting a coherent, executable and traceable engineering process based on the RPIO capabilities.

This deliverable also summarizes the ongoing integration efforts with case studies to validate both the platform and toolkit.


Funded by
the European Union

# Contents

**1 Introduction** **7**
1.1 Document objectives . . . 8
1.2 Document scope . . . 8
1.3 Document structure . . . 9

**2 Stakeholders and requirements** **10**
2.1 Requirements update . . . 10
2.2 Requirements progress . . . 10
2.3 Stakeholders . . . 11

**3 The RoboSAPIENS Adaptive Platform and support tooling** **13**
3.1 RoboSAPIENS Adaptive Platform . . . 13
3.2 RoboSAPIENS Client Library . . . 15
3.3 RoboSAPIENS staged development process . . . 15

**4 The RoboSAPIENS Workflow** **17**
4.1 Overview . . . 17
4.2 Workflow tool . . . 20

**5 Use case validation of the RPIO toolkit** **30**
5.1 TurtleBot 4 Lidar occlusion . . . 31
5.2 Ship motion prediction . . . 31
5.3 Multi robot human detection . . . 33
5.4 Robot arm screw assistant . . . 33

**6 Conclusion** **35**

**Appendices** **37**

**A Publications** **38**

**B RPIO toolkit features** **39**

**C Workflow framework features** **39**

**D RAP and RPIO requirements** **41**

Funded by
the European Union

# Acronyms

**AADL** Architecture Analysis and Design Language. 7, 8, 10, 11, 16, 24, 25, 28, 42

**API** Application Programming Interface. 15, 22

**DSL** Domain-Specific Language. 24

**DT** Digital Twin. 8, 41

**GUI** Graphical User Interface. 20

**IMU** Inertial Measurement Unit. 31

**Lidar** Light Detection and Ranging. 31, 33

**LSTM** Long Short-Term Memory. 32

**M2C** model-to-code. 11, 16, 18

**M2M** model-to-model. 7, 10, 11, 16, 18, 42

**MAPL-K** Monitor, Analyze, Plan, Legitimate, and Knowledge. 41

**MAPLE-K** Monitor, Analysis, Plan, Legitimate, Execute and Knowledge. 8, 10, 11, 13, 14, 16, 19, 30, 33, 34

**MQTT** Message Queuing Telemetry Transport. 14

**OAK-D** OpenCV AI Kit - Depth. 31

**RAP** RoboSAPIENS Adaptive Platform. 2–42

**REST** Representational State Transfer. 22

**ROS2** Robot Operating System 2. 31

**RPIO** robosapiensIO. 4, 5, 7–11, 13, 15, 16, 19–22, 25, 28, 30–32, 34–36, 39–42

**TCP/IP** Transmission Control Protocol/Internet Protocol. 32

Funded by
the European Union

# 1 Introduction

This explanatory document outlines the progress made since the previous deliverable D5.2, highlighting the features added over the past nine months. It provides an overview of recent developments across the RoboSAPIENS Adaptive Platform (RAP) and its supporting toolchain. Updates to the RAP software architecture in this release are limited, with no major structural changes introduced. In contrast, the robosapiensIO (RPIO) toolkit has been extended with updated model transformations, along with minor refinements to the associated code generators. In addition, the document presents the workflow toolkit and introduces the RoboSAPIENS Workflow, which is aligned with the overall RoboSAPIENS big picture. The RoboSAPIENS big picture defines a three-phase high-level workflow, from Concept to Design and Realization, guiding the systematic development and deployment of trustworthy self-adaptive robotic systems, discussed in depth in Section 4.1.

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.

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 RoboSAPIENS Workflow operationalizes the RoboSAPIENS big picture by providing engineers with a structured, step-by-step approach for constructing trustworthy self-adaptive robotics applications. The workflow captures the key engineering activities, artefacts, and decision points required to systematically design, analyse, and deploy adaptive robotic systems in a controlled and verifiable manner. A dedicated workflow tool supports the enactment of the RoboSAPIENS Workflow by coordinating both manual and automated activities. Manual activities enable human-in-the-loop interaction, allowing engineers to provide required inputs such as annotated models, while automated activities trigger RPIO functions exposed as services, including model-to-model (M2M) transformations.

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 and the RPIO toolkit (see Section 3) support the development of open-ended trustworthy self-adaptive robotic systems. As described in this deliverable, the RoboSAPIENS Workflow and its workflow tool operationalise this engineering process by enacting RPIO services, such as the RoboChart-to-Architecture Analysis and Design Language

Funded by
the European Union

(AADL) transformation, and guiding engineers through a structured, step-by-step workflow aligned with the RoboSAPIENS big picture.

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 contains the *legitimate* and *trustworthiness checker* components to support trustworthy robot behaviour. The *legitimate* component evaluates prospective adaptations using Digital Twin (DT) simulations prior to deployment, while the *trustworthiness checker* monitors behaviour during and after adaptation. As described in this deliverable, we developed the RoboSAPIENS Workflow to ensure that Monitor, Analysis, Plan, Legitimate, Execute and Knowledge (MAPLE-K) components are engineered in a trustworthy manner before being deployed in the self-adaptive system, by enforcing correct execution of the prescribed development workflow.

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.

## 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 **RPIO toolkit advancements**, to show the engineering tool progress.
- Provide an overview of the **RoboSAPIENS workflow**, and supporting workflow tool, indicating 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, the RPIO toolkit and the RoboSAPIENS workflow.

## 1.2 Document scope

This document provides a recap of deliverable D5.2 [ANM25], covering both the RAP software architecture and the accompanying RPIO toolkit. It summarises the current status of the RAP platform and the RPIO toolkit, followed by key advancements. Updates of the RAP platform are limited to small improvements and bug-fixes, while for the RPIO toolkit, we revised some transformations, in particular the transformation from RoboChart to AADL, and incremental updates to the associated code generators. The document also presents major developments in the RoboSAPIENS workflow. We introduce a workflow tool that enables workflow enactment by orchestrating RPIO toolkit services, such as model-to-code transformations, while providing workflow traceability to ensure and verify correct workflow

Funded by
the European Union

execution. Finally, we present the RoboSAPIENS workflow itself, showing its alignment with the overall RoboSAPIENS big picture. As the platform and the workflow 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, the RPIO toolkit and the high-level engineering workflow. The next section introduces the RoboSAPIENS workflow and its accompanying workflow toolkit, describing how they support the development and integration of trustworthy self-adaptive robotic systems. The document concludes with an update on the ongoing case studies that validate the RAP architecture, the RPIO toolkit, and the RoboSAPIENS workflow 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 and D5.2 remains largely unchanged in this Deliverable D5.3. The following requirements are added for the RoboSAPIENS Workflow and supporting workflow tool:

**ID_RAP_12 - RoboSAPIENS Workflow:** The RoboSAPIENS Workflow shall provide a clear, step-by-step workflow aligned with the RoboSAPIENS big picture [REF], enabling the systematic construction of trustworthy self-adaptive robotics applications.

**ID_RAP_13 - Workflow enactment:** The workflow tool shall facilitate the execution and enactment of the RoboSAPIENS Workflow, containing both manual and automated activities. Automated activities shall be able to trigger RPIO services, such as M2M transformations, while manual activities shall enable engineers to provide required activity inputs, including annotated AADL models.

**ID_RAP_14 - Workflow traceability:** The workflow tool shall provide traceability across the executed workflow by recording performed steps, associated inputs, and produced artefacts. This traceability information shall enable verification of correct workflow execution and provide a mechanism to ensure that the workflow is executed as intended, without tampering.

## 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 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:** The RAP and RPIO tool support online updating of software components at runtime. 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:** The RAP and RPIO tool support flexible updating and (re-)configuration of the system components, facilitated by the extended component communication and deployment package.
- **ID_RAP_04-07 - Unified architecture description:** The RPIO supports modeling different parts of the trustworthy self-adaptive systems by means of AADL models. Using these architectural and mapping models, the RPIO toolkit

Funded by
the European Union

supports engineers in the development of trustworthy self-adaptive systems by automating the realization, system configuration and deployment.

- **ID_RAP_08 - Communication protocols:** The RAP communication manager 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 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 M2M transformations, transforming the MAPLE-K Architecture Design (RoboChart) models into the software architecture AADL model, are updated to cover the RoboChart model updates.
- **ID_RAP_11 - Design-to-realization:** The model-to-code (M2C) transformations, transforming the annotated AADL models to deployment-specific code and RAP configuration, ready for deployment, are updated to cover the RoboChart model and RAP updates.
- **ID_RAP_12 - RoboSAPIENS Workflow:** The RoboSAPIENS big picture is transformed into the RoboSAPIENS Workflow, interpretable by the workflow tool.
- **ID_RAP_13 - Workflow enactment:** The workflow tool allows the execution of the RoboSAPIENS Workflow. It supports manual activities, triggering human interaction, and automated activities, executing corresponding RPIO services, such as M2M transformations.
- **ID_RAP_14 - Workflow traceability:** The workflow tool implements the workflow traceability, tracing the execution of the workflow, as well as the used and produced artifacts. The implemented traceability mechanism doubles as a verification mechanism, enabling verification of the correctness of both the executed workflow and the used artifacts.

## 2.3 Stakeholders

This section lists the stakeholders of the RAP architecture and their main expectations.

Funded by
the European Union

| Role | Expectations |
|---|---|
| Project leader | Overview of the building blocks of the RAP, and the corresponding technical risks. Overview of the RoboSAPIENS Workflow. |
| 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. |
| Process engineer | Overview of the RoboSAPIENS Workflow. |

Funded by
the European Union

# 3 The RoboSAPIENS Adaptive Platform and support tooling

Previously, self-adaptive robotic applications have often been developed in an ad hoc manner, with trustworthiness and self-adaptivity treated as separate concerns. To address this fragmentation, we introduced the RAP, a software architecture that provides the conceptual and structural foundation for developing and deploying trustworthy self-adaptivity in robotic systems. Complementing this architecture, the RPIO software toolkit operationalizes the RAP software architecture by providing the concrete tools and infrastructure needed to develop, integrate, and deploy all components of the trustworthy self-adaptive applications across diverse robotic domains.

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

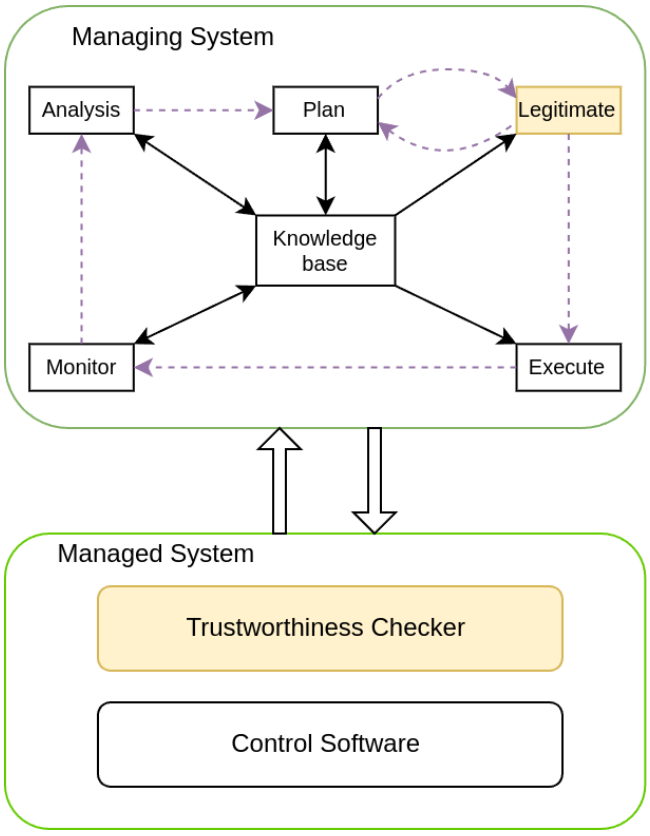



Figure 1: The trust components integrated in the architecture

## 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 developing and managing software using the MAPLE-K pattern. The MAPLE-K components are

Funded by the European Union

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

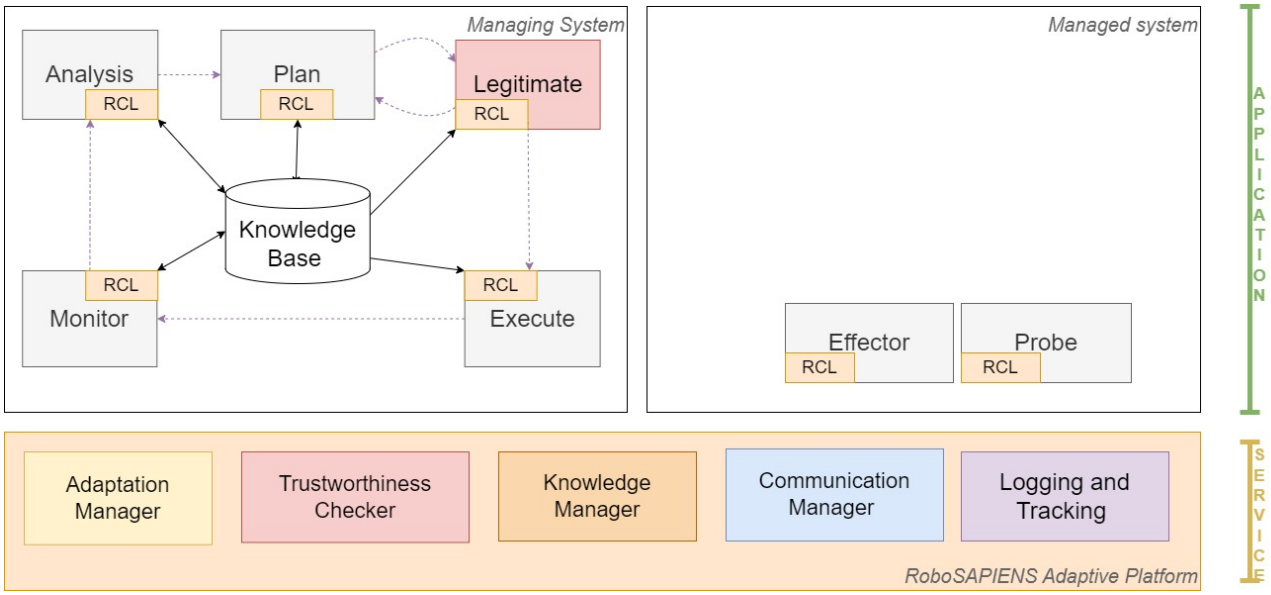



Figure 2: The RAP platform software architecture concept

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 MAPLE-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.

Funded by
the European Union

5. **Logging and tracking** ensures transparency and traceability by monitoring and recording events throughout the adaptation process.

## 3.2 RoboSAPIENS Client Library

As illustrated in Figure 2, the RoboSAPIENS client libraries are the Application Programming Interface (API) that allow users to implement their (MAPLE) software components and integrate them into the RAP in order to build adaptive system applications. Through these client libraries, software components can share knowledge and communicate with each other using custom messages, which are generated via code generators tailored to the respective programming language. In addition to language-specific communication mechanisms, the client libraries expose core functionality of the RAP software architecture to users, such as system-wide logging and component monitoring. Moreover, the use of client libraries enables distributed deployment of software components, allowing individual components to be deployed across heterogeneous, interconnected devices (e.g., edge nodes, embedded platforms, or cloud services), while maintaining seamless integration and communication through the RAP.

Currently, a Python-based client library (rpclpy) is available and serves as the primary and most mature client library. It is mainly used as a prototyping and experimentation environment, as Python enables rapid development and fast iteration cycles. In addition, prototype client libraries exist for C/C++ and Rust, both for increased performance and portability. Both prototype client libraries currently support a reduced subset of the available RAP features and provide an initial step toward supporting a broader range of programming languages and execution platforms.

Multi-formalism support provides engineers with the freedom to select the most appropriate programming language and abstraction for the problem at hand. For example, a Legitimate component may be implemented in Python to benefit from rapid prototyping and fast iteration cycles, whereas system components that are primarily concerned with efficiency—such as system monitoring and analysis—may be better implemented in C++ to meet performance and resource constraints.

## 3.3 RoboSAPIENS staged development process

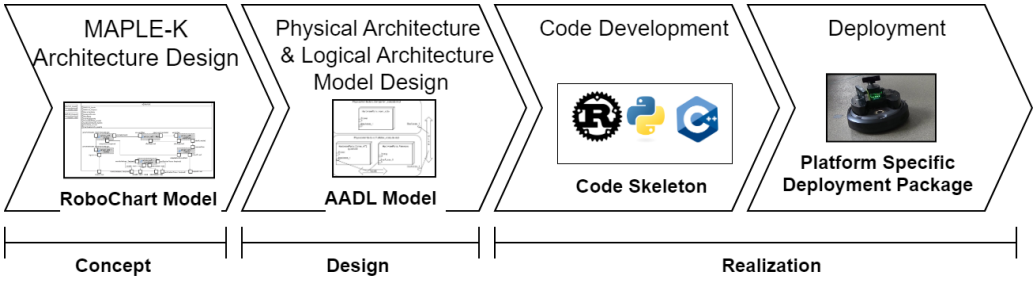



Figure 3: RPIO transformation workflow from RoboChart models to deployable adaptive robots.

Funded by
the European Union

The RoboSAPIENS development process provides a comprehensive, end-to-end solution for designing and commissioning MAPLE-K loops in robotic applications. The framework supports automated M2M and M2C transformations, enabling seamless integration of the RAP into robotic systems. The high-level overview of the overall development workflow, shown in Figure 3, starts from a MAPLE-K architecture design specified in RoboChart and culminates in a deployed and running adaptive loop that is fully integrated with and connected to the robotic system, following a series of successive model transformations and code generation steps.

In more detail, the MAPLE-K architecture design in RoboChart **(Concept phase)** is transformed into a software architecture expressed in AADL, after which the resulting architecture is extended with deployment-specific information such as target execution nodes, communication bindings, etc **(Design phase)**. Based on this enriched architecture model, the framework automatically generates code skeletons for all MAPLE-K software components, along with deployment-specific configuration files containing application-independent parameters required to execute the RAP **(Realization phase)**. Each generated software component accesses RAP services through the RAP client libraries.

The main contribution of this deliverable is a detailed description of this workflow, referred to as the RoboSAPIENS Workflow, together with its supporting toolchain that enables the execution of the workflow in practice. By combining the RAP with the RPIO toolkit for model transformations and code generation, the workflow provides a standardized and automated process that spans architectural design, software generation, configuration, and deployment of distributed MAPLE-K components.

Funded by
the European Union

# 4 The RoboSAPIENS Workflow

## 4.1 Overview

To construct the RoboSAPIENS Workflow, it is necessary to align with the overarching RoboSAPIENS big picture, which defines the high-level engineering workflow for developing trustworthy self-adaptive systems. The RoboSAPIENS Workflow is therefore derived from and structured according to this project-wide conceptual framework, illustrated in Figure 4.

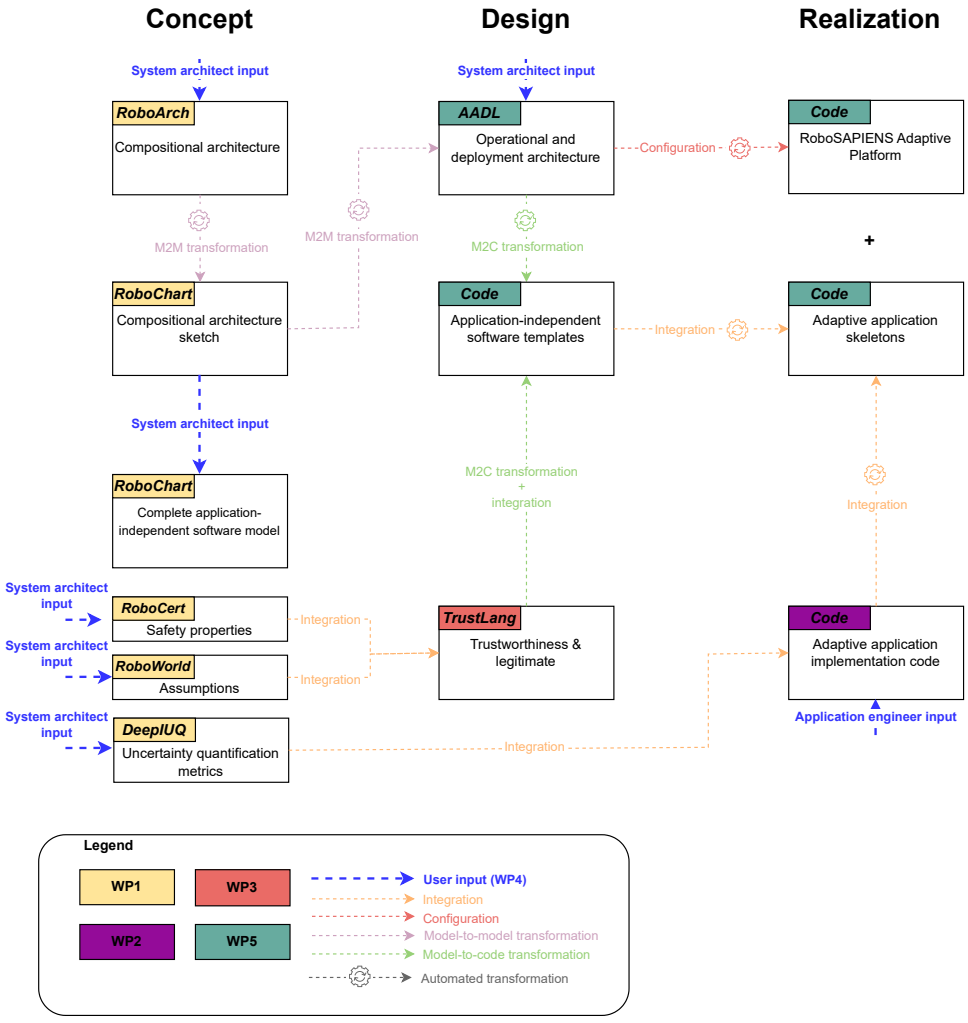



Figure 4: RoboSAPIENS big picture high-level workflow.

Figure 4 presents the comprehensive RoboSAPIENS engineering workflow, structured into three phases, *Concept*, *Design*, and *Realization*, that together enable the systematic development of trustworthy self-adaptive robotic systems. In the Concept phase, the system's compositional architecture is modeled using *RoboArch* and transformed into behavioral models in *RoboChart*, enriched with safety proper-

Funded by the European Union

ties via *RoboCert* and environmental assumptions via *RoboWorld*, enabling formal verification through model checking and theorem proving. In the Design phase, this architecture is refined into logical and physical deployment models, including component allocation, communication structures, and trustworthiness specifications expressed in *TrustLang*, from which application-independent software templates are generated. In the Realization phase, these templates are (semi-)automatically transformed into application-specific skeletons that integrate custom functionality and are deployed on the RAP runtime. Across all phases, activities are connected through automated transformations (e.g., M2M, M2C, integration, and artifact weaving) and human-driven tasks such as modeling and coding, ensuring traceability, safety, robustness, and trustworthiness throughout the lifecycle.

From this RoboSAPIENS big picture, we construct the RoboSAPIENS Workflow as a concrete refinement of the high-level engineering process. While the big picture defines the overarching Concept, Design, and Realization phases, the RoboSAPIENS Workflow introduces the specific artifacts, engineering activities, and structuring needed to guide engineers throughout the development. In doing so, it operationalizes the high-level principles and ensures their consistent realization on the RAP.

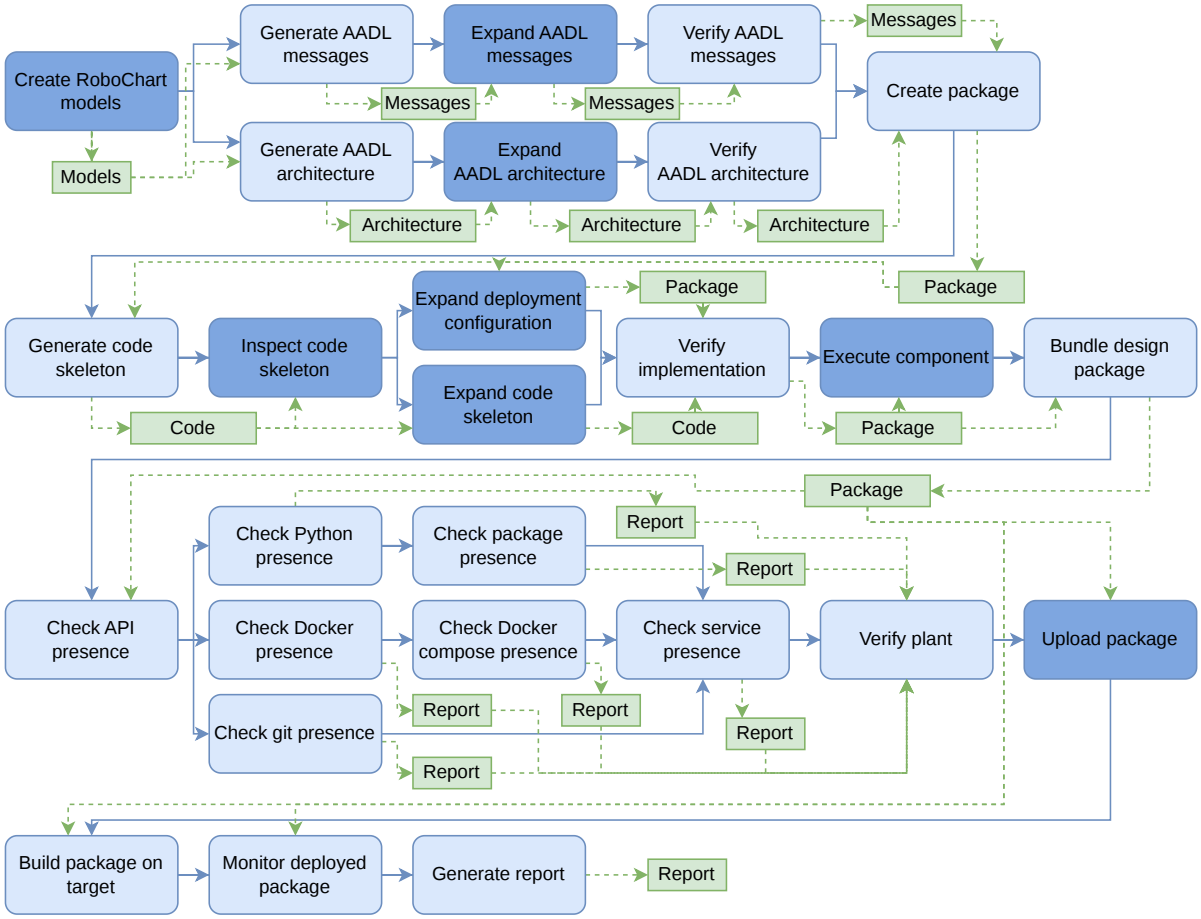



Figure 5: RoboSAPIENS workflow omitting retry logic.

Funded by
the European Union

Figure 5 illustrates the RoboSAPIENS Workflow. Engineering activities, manual (dark blue) or automated (light blue), are depicted as rounded rectangles, while artifacts are represented as rectangles. Solid lines denote control flow, indicating the sequencing of activities, whereas dashed lines represent data flow, capturing the exchange and transformation of artifacts between activities.

The starting point of the workflow is the creation of the RoboChart models which can be partially generated from the RoboArch models. Within this manual activity, the engineer provides the required input artifact as a zipped collection of *.rct* files that describe the conceptual architecture. This artifact is stored for subsequent use within the workflow, marking the completion of the Concept phase.

Once the conceptual architecture is completed, the workflow enters the Design phase and splits into two parallel branches to refine the MAPLE-K architecture. One branch focuses on generating the message templates, while the other generates the software architecture templates. In both branches, an automated transformation is triggered, which uses the conceptual architecture as input and produces the respective templates as output artifacts. Once the automated transformation activity is completed, the generated templates are provided as input to a manual activity. This step has two objectives: (i) to allow the engineer to inspect and validate the correctness of the transformation results, and (ii) to enrich the templates with additional design details, specifically by defining the compute architecture and allocating the software architecture components to the corresponding compute elements. After the engineer completes this step, the extended artifacts are subsequently processed by an automated validation activity that ensures only permissible modifications were made. At this stage, the workflow introduces a decision point that evaluates whether the validated artifacts satisfy all defined system constraints. If the artifacts do not pass the validation step, the process iterates back to the manual extension activity, requiring the engineer to revise the templates until compliance is achieved. This iterative loop continues until the constraints are fully satisfied. This and further retry loops are omitted from Figure 5 to keep it readable. Once both parallel branches successfully complete validation, the finalized logical and messaging artifacts are merged.

Once the Design phase is successfully completed, its validated output serves as the starting point for the Realization phase. The first automated activity generates the RPIO package, which encapsulates both the software configuration and the deployment configuration required to establish the execution environment on the target compute hardware. The deployment configuration may include, for example, launch files for execution within a Docker environment. Subsequently, a second automated activity generates the application-specific code skeletons, integrating the predefined structural elements and storing them within the RPIO package. This is followed by a manual activity in which the engineer inspects the generated skeletons, implements the required application-specific functionality, defines the necessary software dependencies, and, if these dependencies affect the execution environment, extends the deployment configuration accordingly. Within this activity, the engineer is also responsible for testing the implemented components on the intended target platform, to verify correct and expected behavior and to ensure that all dependencies are properly resolved. At this stage, the workflow introduces a second decision point that enables iteration over the implementation and

Funded by
the European Union

dependency configuration until the component behavior aligns with its intended specification and operates correctly on the target hardware. Now the engineer must manually execute the component within the desired execution environment to check if the component and its dependencies are working as intended. This is a simple unit test, before integration within the deployment phase.

Once the Realization phase is completed, the finalized RPIO package, containing the complete implementation of all system components, is ready for deployment on the selected target platforms. First, an automated activity is checking prerequisites and configuration of the target execution environment, to ensure alignment between the implementation and the target execution context. These checks verify the availability and correctness of required tools, packages, services, and runtime environments (e.g., installed RAP package, Docker/Docker Compose, supporting software, etc.). At this stage, the workflow introduces another decision point that enables iteration over the execution environment configuration, ensuring alignment with the designed deployment setup. If inconsistencies or missing prerequisites are detected, corrective actions are triggered, either automatically or through engineer intervention, until the environment fully conforms to the specified configuration and supports correct system operation. Secondly, within a manual activity, the engineer uploads the finalized RPIO package to the selected compute platform(s). This is followed by an automated activity that builds and runs the application on the target hardware. The automation is enabled by the RPIO package itself, which encapsulates all necessary software and deployment configuration to correctly build and execute the corresponding components on the device.

Subsequently, an automated verification activity checks whether the deployed application components are operating correctly by monitoring their heartbeat signals. Once the self-adaptive application is successfully deployed and running, a final report is generated. This report documents the executed activities within the RoboSAPIENS Workflow, as well as the consumed and produced artifacts, providing evidence of the successful and traceable execution of the entire workflow.

## 4.2 Workflow tool

To operationalize the presented RoboSAPIENS Workflow, we developed a dedicated workflow tool that translates the RoboSAPIENS workflow specification into an actionable execution process. The tool instantiates the defined activities, control logic, and data flows, enabling engineers to step through the workflow in a guided and structured manner. Through guarded progression, the tool enforces that each activity is completed successfully, based on defined validation checks and decision points, before allowing continuation to subsequent steps. For each activity, it manages the required input artifacts, their validation, and produces and stores the corresponding output artifacts. In doing so, the workflow tool ensures consistency, traceability, and controlled progression while reducing configuration drift and increasing process repeatability.

Figure 6 presents the Graphical User Interface (GUI) of the workflow tool, illustrating its look and feel as well as how the RoboSAPIENS workflow is visually represented and navigated within the tool. An in-depth discussion of the workflow tool is provided in the subsequent sections, structured into four parts, each satisfying a

Funded by
the European Union

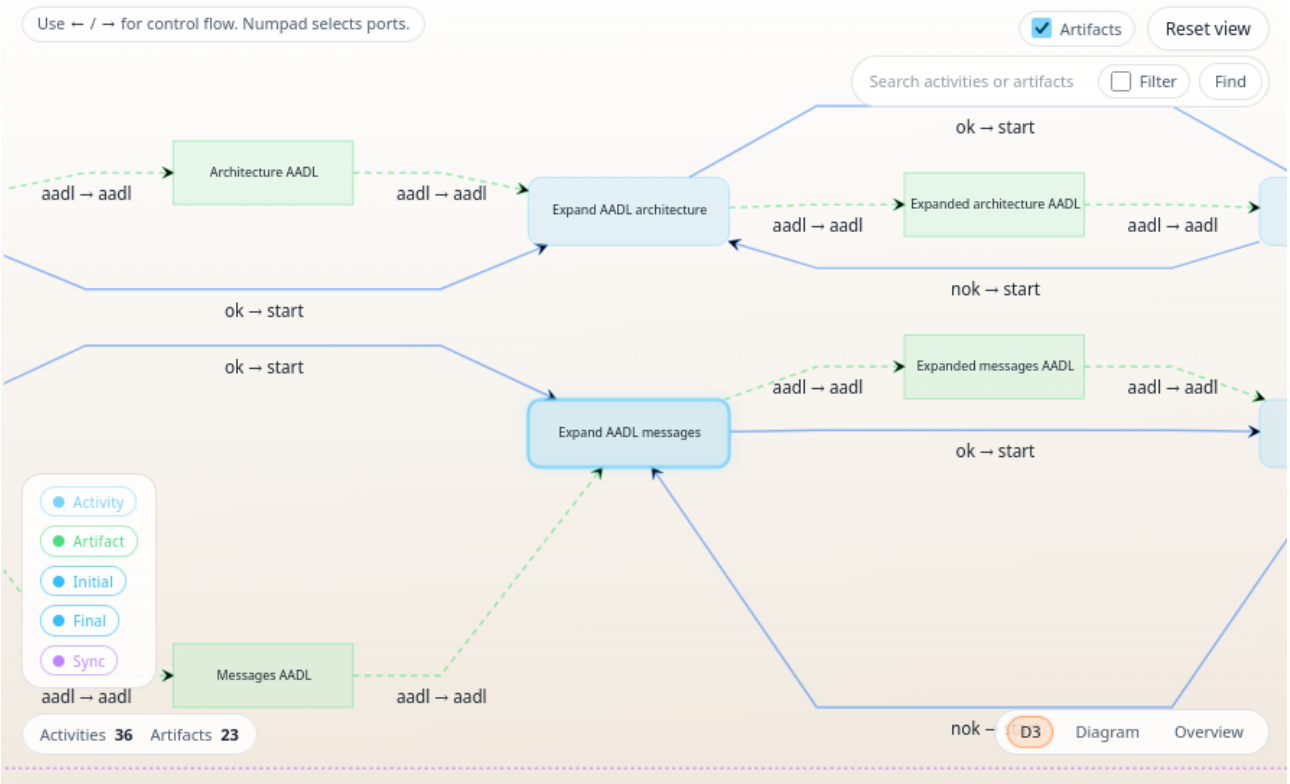



Figure 6: RoboSAPIENS design and deployment flow in the front end web interface codenamed Weaver.

subset of the requirements as denoted between parentheses:

1. **Workflow tool architecture**: presenting the architecture of the tool
2. **Workflow definition**: explaining how the RoboSAPIENS workflow is modeled within the tool (W-M-1).
3. **Workflow enactment**: explaining how the workflow is operationalized and executed (W-E-1).
4. **Workflow certification**: explaining how the executed workflow is certified (W-E-2).

### 4.2.1 Workflow tool architecture

The workflow tool is built upon a modular architecture designed to support the enactment, validation, and visualization of the RoboSAPIENS Workflow. This modular structure separates concerns such as workflow execution logic, artifact management, validation, and user interaction, ensuring extensibility and maintainability. The architecture enables seamless integration with external services, such as RPIO functions, to automate engineering activities, while also providing mechanisms to notify and guide users when manual intervention is required

Funded by
the European Union

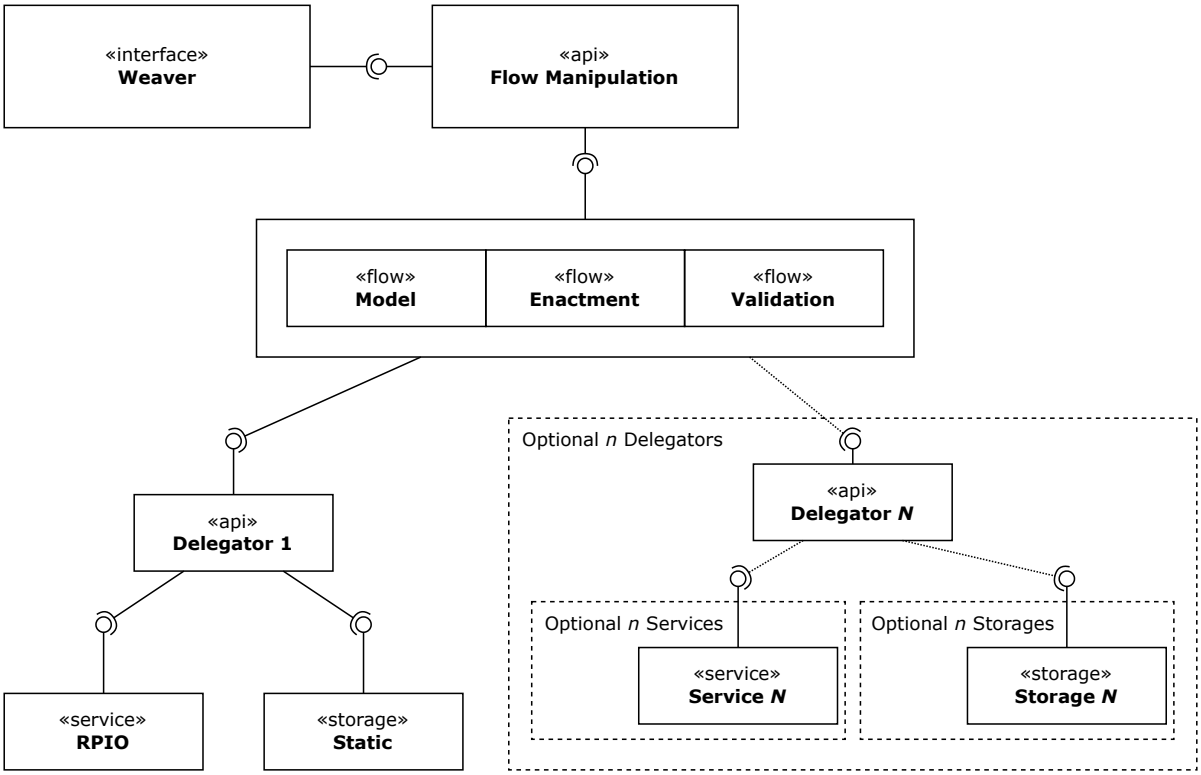



Figure 7: Architecture of Flow tooling with pre-defined RPIO and static storage services. Optional services are discoverable.

Figure 7 depicts the architecture of the workflow tooling and the interaction between its main subsystems during enactment.

At the top level, we expose the user-facing front end under the name Weaver. It provides visualization, interaction, and control over Flow definition and enactment. The Flow Manipulation API serves as the coordination layer between the front end and the core Flow engine. It translates user actions into enactment commands and propagates state updates back to the interface. Weaver communicates with the Flow Manipulation API through Representational State Transfer (REST) and server-sent events, allowing users to start enactments, step through activities, and observe execution progress in real time.

The Flow Model and Enactment engine delegates concrete work to external helper APIs. In the architecture shown in Figure 7, such helpers are: Delegator 1, … N. These services expose actions that can be bound to Activities or Artifacts in the Flow. During enactment, the Flow engine invokes these services to execute automated steps or to retrieve and store artifacts. The Flow engine remains agnostic to service implementation details and interacts with them through a uniform service interface.

Each of the Delegators may rely on one or more underlying providers (Service or Storage), discovered dynamically. For example, Delegator 1 connects to the RPIO service and static storage backend, while other Delegators connect to their own

Funded by
the European Union

provider ecosystems. This discovery-based design allows services to be substituted or extended without modifying the Flow engine or the front end.

The workflow is defined, enacted, and verified using Weaver (W-M-1). In Weaver, a Flow specifies activities, artifacts, and their relationships.

- Artifacts represent structured data.
  - Each artifact has a unique identifier.
  - Each artifact has a human-readable name.
  - Each artifact has a description providing additional detail.
  - Each artifact has a way of storing and retrieving data.
- Activities represent units of work.
  - Each activity has a unique identifier.
  - Each activity has a human-readable name.
  - Each activity has a description providing additional detail.
  - Activities include:
    - Initial activities that denote where to start the flow.
    - Final activities that signify the end of a flow.
    - Synchronization gates that indicate splitting and joining of parallel branches.
    - Automated activities that include instructions to perform operations.
    - Composite activities that consist of a sub-flow.
- Edges connect activities and artifacts.
  - Each edge has a source port and target port.
  - Ports are identified by name. Simplifying deterministic choice during execution.
  - Examples include:
    - `Act1[ok]` → `Act2[start]`,
    - `Act1[produces]` → `Art1[data]`.
  - Control flow edges define execution order.
  - Data flow edges transfer artifacts between Activities.
  - Synchronization edges coordinate parallel execution by linking pairs of corresponding Synchronization Activities.
  - Trace edges record enactment provenance.
  - Type edges enforce artifact compatibility and parameter compatibility.

Funded by
the European Union

### 4.2.2 RoboSAPIENS workflow definition

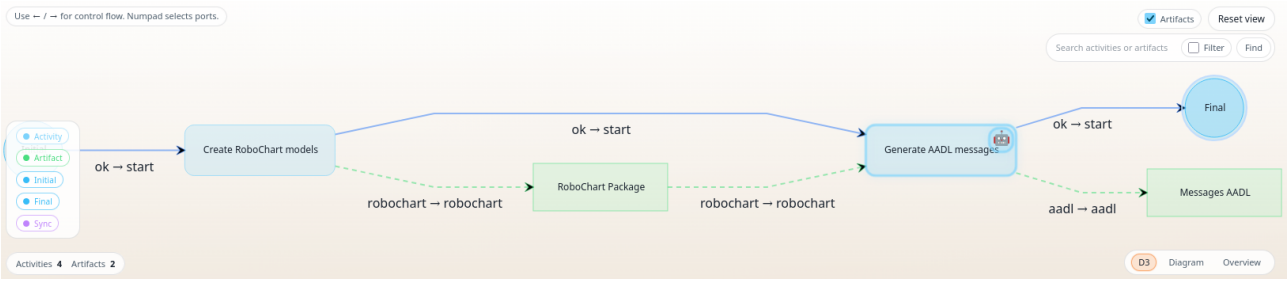



Figure 8: Example RoboChart to AADL conversion flow extracted from the main RoboSAPIENS flow.

The RoboSAPIENS workflow is currently predetermined and predefined within the workflow tool. Its construction is a manual activity, carried out using a custom Domain-Specific Language (DSL), which is tailored to express workflow structures and their associated artifacts in a consistent and tool-compatible manner. Alternative definition mechanisms are also available, but these require expert knowledge of the workflow tool's internal implementation and are therefore intended for advanced users. The defined workflow can be visualized as a graph in the workflow tool as shown in Figure 8.

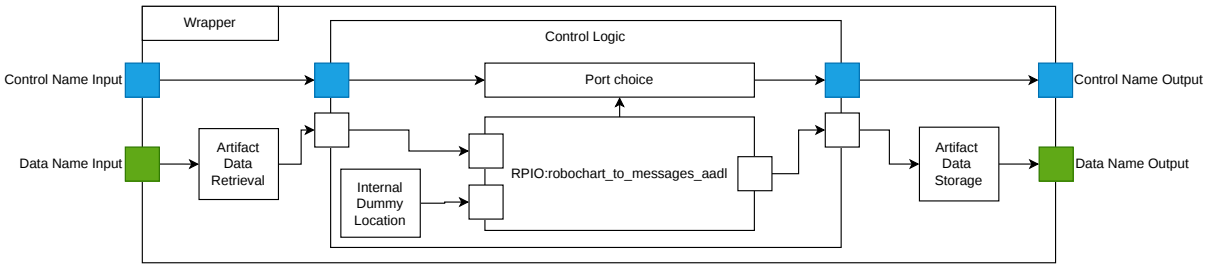



Figure 9: Interaction diagram of the services used during enactment showcasing the RoboChart to AADL conversion.

Artifacts can be stored using specialized storage mechanisms. The mechanism themselves are parametrized per flow to choose a storage path. This provides flexibility but requires specifying the storage handlers for more intricate cases where data is not stored in memory. When an activity produces or consumes an artifact the artifact's accompanying storage mechanism is called to store or retrieve the data. This interaction is visible in the example service in Figure 9. It depicts how the first step in the service wrapper for converting RoboChart to an AADL file is the artifact data retrieval. This step converts the artifact into the data it represents by using the metadata in the Storage handler to retrieve the stored RoboChart models. The RoboSAPIENS workflow specifies a remote static storage service for each of the artifacts involved. The data is sent to a server and accessed when required.

The control logic in Figure 9 depicts the inner working of Automated Activities using a service. The artifact data and control flow name are passed in. The control

Funded by
the European Union

logic pre-processes this data, which might include unzipping an archive or preparing dummy arguments. Dummy arguments are required when the method call takes parameters which are implicit in the workflow. Such as the output location of a generated file as this is handled by the storage mechanism. It then performs an action. In the example this is converting the RoboChart files into AADL messages by using the corresponding call from the RPIO library.

The RoboSAPIENS workflow mostly uses an API wrapper around RPIO for its operations. This allows calling individual methods with the correctly pre-parsed parameters. The automated activity itself only needs to define the service and its parameters, for example how to call the control logic.

### 4.2.3 Workflow enactment

Workflow enactment is the core functionality of the workflow tool, responsible for transforming the defined RoboSAPIENS Workflow into an executable and guided engineering process. It interprets the specified activities, control logic, artifacts, and decision points, and operationalizes them into structured execution steps. Through guarded progression, it ensures that activities are completed and validated before advancing, coordinates automated tasks and external service integrations, manages artifact flow, and notifies engineers when manual intervention is required, thereby enabling controlled, traceable, and consistent execution of the workflow.

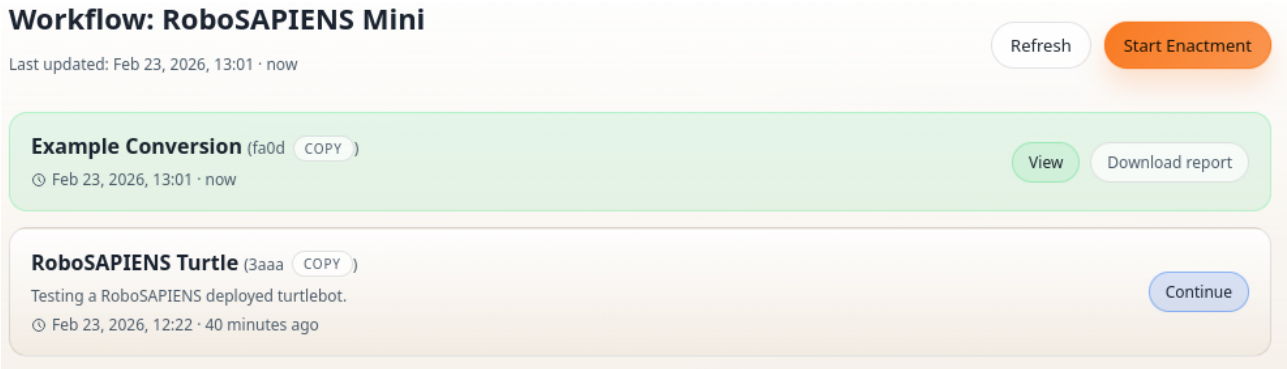



Figure 10: Enactments tab in the Weaver interface showing the current enactments for a miniature version of the RoboSAPIENS workflow where only the RoboChart to AADL transformation takes place.

Figure 10 shows the enactments overview for a part of the main RoboSAPIENS flow. In this flow only the RoboChart to AADL transformation is performed as a show case. Multiple enactments can take place at the same time. Active enactments can be resumed. Completed enactments expose their full execution record and reports.

Figure 11 shows a single enactment with its Trace. The upper half is reserved for manipulating activities. It shows the currently available activities based on satisfied prerequisites. For this enactment we are at a manual activity which asks to provide the RoboChart files. It will only ever display activities which can be started, are in progress, or can be completed. Activities that can be completed require their output artifact to be stored and a control flow choice to be made.

Funded by
the European Union

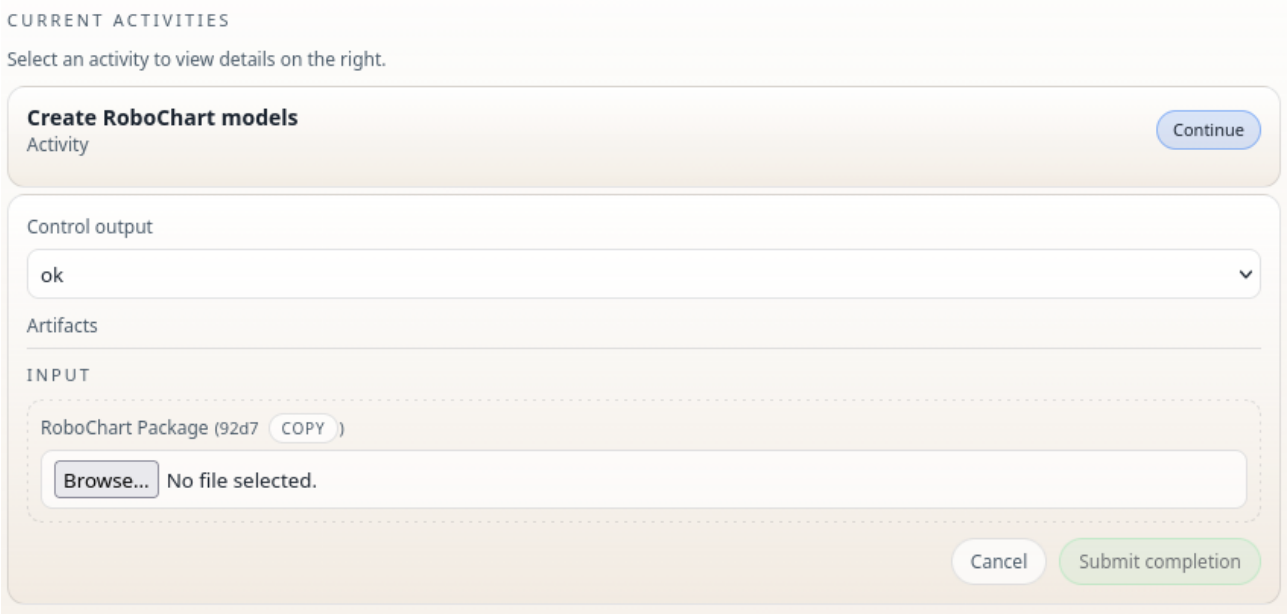



Figure 11: Cropped screenshot of the Weaver interface showing a single enactment actions pane.

The distinction between manual and automated activities is simple. Manual activities expose access to their input artifacts which can be viewed and require the user to upload output artifacts and make control decisions. Automated activities only require confirmation of completion and show their current progress.

The trace in Figure 12 shows the corresponding trace of the enactment. The initial activity has been started and completed and the RoboChart files have been uploaded. The conversion has taken place and the final activity has been reached. Each of the events is marked as verified.

Below, the details of workflow enactment within the workflow tool are provided. The enactment mechanism follows token-based, Petri-net-like semantics[Pet66]. Activities consume tokens from their incoming control edges and produce tokens on their outgoing edges upon successful completion. Decision points evaluate conditions and forward the token along exactly one outgoing control edge, while synchronization gates coordinate parallel branches and emit tokens on all outgoing edges once their conditions are satisfied. Execution begins at the designated initial activity and terminates at the final activity, ensuring a well-defined and controlled progression through the workflow.

During enactment, the workflow tool produces an **append-only trace** that records the execution history. Each trace event stores its parent element, timestamp, data hash, predecessor hash, and event type, thereby capturing both execution order and artifact provenance. Together, the defined workflow and the generated trace make the execution state explicit and enable independent verification of correctness and integrity.

Activities and artifacts can have associated services. Activity services consume input artifacts, produce output artifacts, and determine the subsequent control flow

Funded by
the European Union

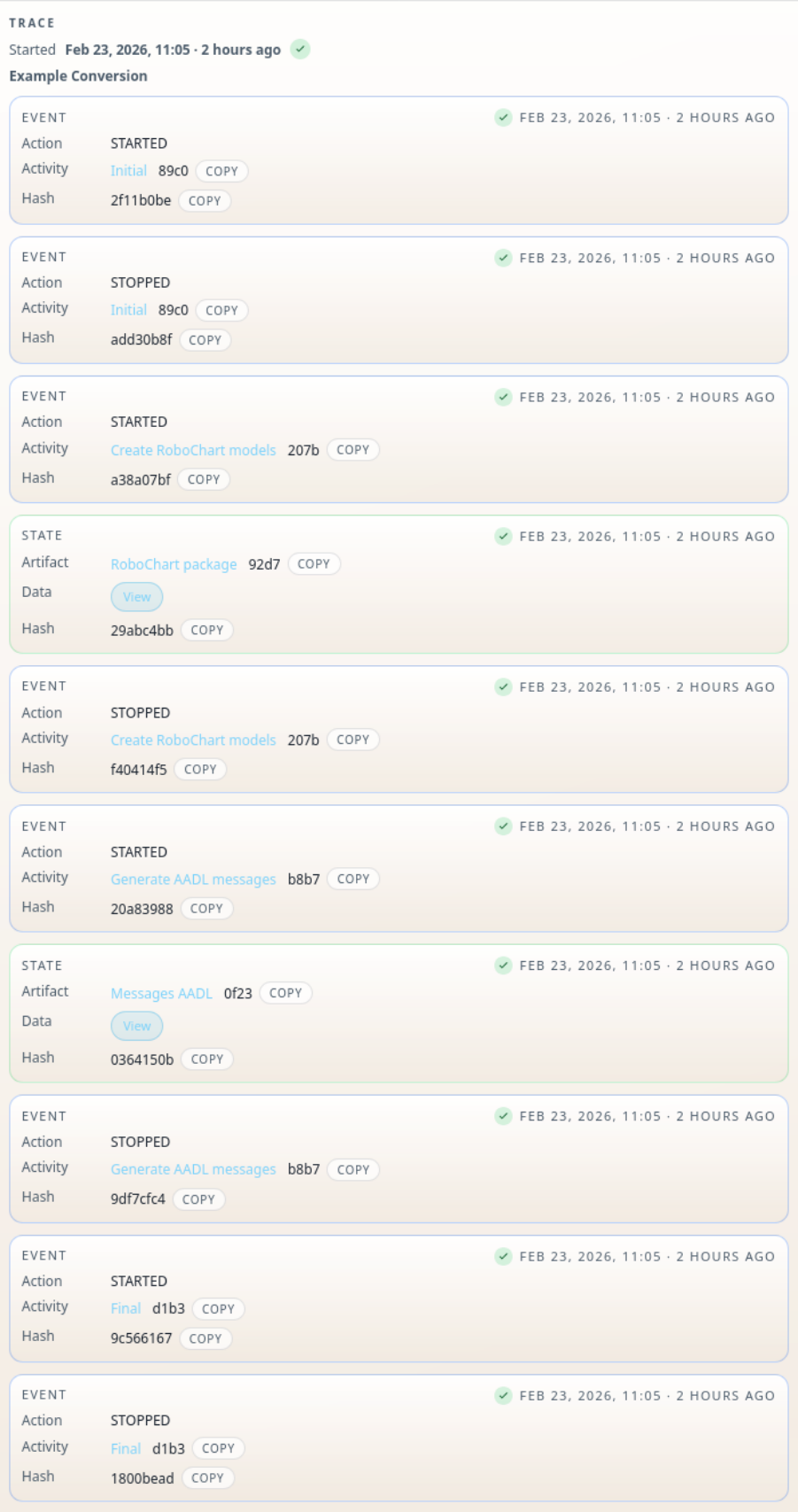



Figure 12: Single enactment trace overview with details.

Funded by
the European Union

based on execution results. For example, a *Generate AADL messages activity* consumes its corresponding RoboChart artifact, produces a result artifact containing the generated AADL, and directs the control flow according to the generation result. Artifact services manage data retrieval and storage, ensuring that required artifacts are automatically fetched when an activity is enacted. Thus, input artifacts are transparently provided at execution time.

During enactment, RoboSAPIENS-specific services are triggered as part of automated activities. Through this service mechanism, domain-specific engineering actions can be seamlessly woven into the workflow execution. While the workflow tool itself is designed as a generic enactment engine, the integration of project-specific services, such as those provided by the RPIO toolkit, allows it to be tailored to the needs of a particular project.

### 4.2.4 Workflow certification

Another key feature of the workflow tool is its capability to validate workflow execution and, by extension, provide evidence that supports certification (W-E-2). The tool ensures that no steps are skipped and that the defined process is followed without tampering by maintaining an append-only Trace of the enactment. Each trace event links to its predecessor and records artifact provenance, forming a verifiable chain over all executed activities and generated artifacts.

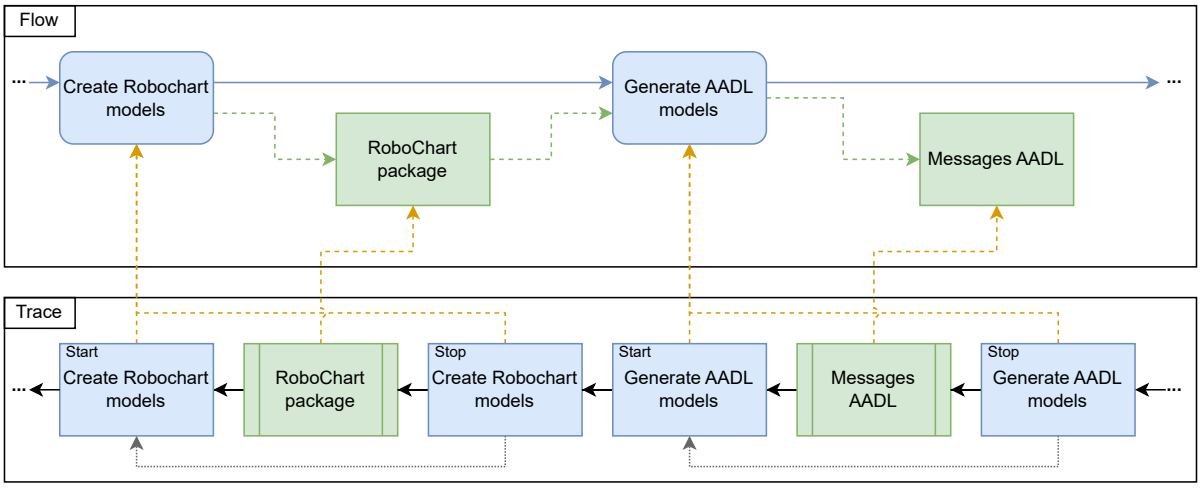



Figure 13: Single enactment trace linked to its flow.

Figure 13 is a mapping representation of a partial trace shown in Figure 12 and its flow from Figure 8. Each of the events (blue) in the trace can be linked back to an Activity (blue, rounded) in the flow. This concept also applies to the trace states (green) and their corresponding artifact (green). Each step, event or state, includes all its data when calculating its hash. For an event this is the timestamp, whether it started or finished an activity, the hash of its linked activity, and the hash of the prior step. The state also adds the hash of its payload, the data stored, into its hash calculation. This allows checking whether the retrieved data has not changed between verifications.

Funded by
the European Union

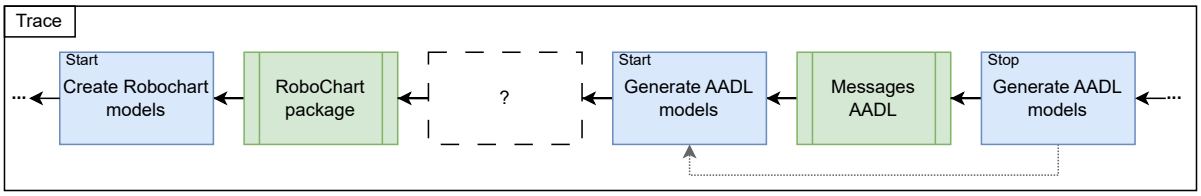



Figure 14: Single enactment trace with a missing step.

Another benefit of having a link to the flow is the ability to recognize missing steps. Figure 14 illustrates such an example where the creation of the RoboChart models was never completed and somehow the workflow was continued. This results in an invalid trace and therefore, an invalid enactment. This technique also allows checking for the missing state of an artifact or general order of operations violations.

Validation can be performed by verifying the hashes in this chain and retracing their provenance against the defined workflow. Because of the nature of the order of operations, certain activities require successful completion before continuation. For example, the validation of the code extension must be completed successfully before deployment is allowed, preventing unverified implementations from progressing to the target platform. External tooling can check the output trace during execution to verify which activities were successfully completed. All the data is exposed through an API, this allows for example the trustworthiness checker to check if certain optional branches were taken while doing its checks.

In addition, a report generation feature is being actively developed and integrated to provide a concise yet comprehensive overview of a workflow enactment. This report enables quick assessment of the overall execution status and serves as a consolidated evidence document. It includes detailed statistics and artifacts such as the workflow structure, the execution trace, and the associated certification results, thereby supporting transparency, auditability, and compliance requirements.

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. As already introduced in D5.2, 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 | Previous phase (D5.2) | Current phase (D5.3) |
|---|---|---|---|
| TurtleBot 4 simulator | UAntwerp | Finalization | Finalization |
| TurtleBot 4 Lidar occlusion | AU | Evaluation | Finalization |
| Ship motion prediction | NTNU | Implementation | Evaluation |
| Active Perception | AUTH | Implementation | Implementation |
| Multi robot human detection | PAL | Exploration | Implementation |
| Robot arm screw assistant | DTI | Exploration | Implementation |
| Dynamic risk model | IFF | Exploration | Exploration |

Use cases with updates reported in D5.3

Table 1: Current status of the RPIO toolkit use-cases

As illustrated in Table 1, the use cases have evolved since the previous deliverable (D5.2), reflecting both technical refinements and progress in development maturity. The highlighted rows indicate the use cases that have undergone changes and are discussed further in this deliverable; use cases that remained in the same phase are omitted from further discussion in this report.

The NTNU ship motion prediction case study was updated to align with a newer version of the RPIO toolkit, resulting in an increased number of validated features compared to the previous report. The TurtleBot4 case study underwent a minor update, primarily addressing debugging and stabilization issues, without altering its overall scope. In parallel, the DTI robot arm screw assistant and the PAL multi-robot human detection case studies have progressed beyond initial exploration. In both cases, the use cases have now been more precisely defined, and concrete development activities have commenced. The following sections focus exclusively

Funded by
the European Union

on these use cases, demonstrating how the reported changes are realized within each individual case study.

## 5.1 TurtleBot 4 Lidar occlusion

*Case Owner: AU* *Current Phase: Evaluation*

### 5.1.1 Description & status

The TurtleBot4 is an advanced open-source robotics platform designed for education and research. It features a Raspberry Pi 4 running Robot Operating System 2 (ROS2), an OpenCV AI Kit - Depth (OAK-D) spatial AI stereo camera, and a 2D Light Detection and Ranging (Lidar), enabling robust sensor-based applications. The platform is built on the iRobot® Create 3 mobile base, which provides essential sensing and actuation capabilities, including an Inertial Measurement Unit (IMU), optical floor tracking, wheel encoders, and infrared sensors, supporting accurate localization and navigation.

This use case serves as an academic validation case, where initial architectural changes and toolchain updates are typically introduced and verified. The primary goal of the adaptation process remains the handling of anomalies caused by intentional occlusion of the laser scanner by external objects. To address this, the navigation stack is adapted to detect Lidar occlusion events and to modify the robot's behavior by periodically stopping and rotating in place, thereby updating the navigation plan to recover from blocked sensor views.

Between versions 0.4.0 and 0.4.3 of the RPIO toolset, two minor but important architectural updates were validated within this case study. First, the Knowledge Manager was extended to support reading and writing knowledge using defined object representations (A-KM-8), improving consistency and interoperability across components and enabling more reliable information exchange within the distributed architecture. Second, the Knowledge Manager was updated to include unique identifiers for stored knowledge entries (A-AM-5), ensuring traceability of knowledge items within the knowledge base and aligning stored data with their corresponding feature identifiers.

The conceptual architecture and component development for this use case closely mirror those of the internal validation scenario.

## 5.2 Ship motion prediction

*Case Owner: NTNU* *Current Phase: Evaluation*

### 5.2.1 Description and status

In the maritime domain, accurate model prediction is fundamental for safe and effective decision support. The NTNU case study focuses on adapting ship motion prediction at runtime to account for disturbances and uncertainties, with the objective of maintaining reliable and precise predictions under changing environmental conditions such as wind, waves, and currents.

Funded by
the European Union

Within this use case, predictions are presented through a graphical decision-support interface and are generated using three complementary models: a kinematic model, a dynamic model, and a data-driven Long Short-Term Memory (LSTM) model. The kinematic and dynamic models are physics-based and continuously provide predictions during operation. In contrast, the data-driven LSTM model requires retraining when significant environmental anomalies are detected. During retraining, the kinematic and dynamic models act as a safe fallback, ensuring that reliable predictions remain available while sufficient data are collected and the learning process completes. This combination enables robustness against both model drift and delayed adaptation.

Compared to the previous deliverable, this case study was updated to use version 0.4.3 of the RPIO toolkit, upgrading from version 0.3.0. This update was primarily motivated by the integration of the Trustworthiness Checker and the adoption of newly introduced RPIO features, including dashboard support and reading and writing knowledge using object-based representations. As a result, the NTNU case study served as a validation platform for a broad set of newly introduced architectural features.

At the Adaptation Manager level, the case validated support for orchestrating events and messages across both intra-component and inter-component communication paths (A-AM-2). In addition, messages and events were extended with timestamps (A-AM-3) and unique identifiers (A-AM-4), enabling accurate temporal reasoning, traceability, and improved debugging in distributed runtime settings.

Several communication protocols were validated through this use case. The Communication Manager was extended to support Redis Pub/Sub (A-CM-3) for ultra-low-latency event distribution, RabbitMQ (A-CM-4) for reliable and structured message queuing, WebSocket communication (A-CM-5) for real-time bidirectional data exchange with client interfaces, and direct Transmission Control Protocol/Internet Protocol (TCP/IP) communication (A-CM-6) as a foundational transport mechanism.

At the Knowledge Manager level, the case study validated support for Memcached (A-KM-3) as a lightweight, high-performance in-memory cache, and Apache Kafka (A-KM-4) for durable, replayable event streaming. In addition, standardized message-based read and write operations were used to ensure consistent knowledge exchange across components (A-KM-5). Runtime logging and tracking were enhanced through Redis-based logging support (A-LT-3), enabling efficient, low-latency trace capture without introducing significant system overhead.

Finally, this case study validated the first integration of the Trustworthiness Checker within the RPIO ecosystem. The checker was integrated using both MQTT-based communication (A-TC-1) and a Redis-based fast communication path (A-TC-2), enabling real-time trust assessment of runtime data with minimal latency. These integrations allowed the NTNU case study to act as a primary validation environment for trust-related architectural capabilities.

All experiments were conducted using a high-fidelity bridge simulator manufactured by Kongsberg. This simulator, commonly used for training nautical students and professional captains, provides a realistic operational environment with con-

Funded by
the European Union

figurable wind, waves, currents, fog, and rain, enabling controlled yet realistic validation of adaptive maritime prediction systems.

## 5.3 Multi robot human detection

*Case Owner: PAL* *Current Phase: Implementation*

### 5.3.1 Description & status

In this use case, we investigate the design of a self-adaptive, distributed robotic system using PAL' omni-directional robots. The system is envisioned to operate under a distributed architecture, where multiple autonomous agents interact through a shared global knowledge base. Such a deployment requires system components to run across different hardware platforms and network nodes, making robust and efficient communication a core requirement. In dynamic human-centered environments, coordination between robots must be timely and reliable, while the shared knowledge base must remain consistent across agents. In addition, the architecture must scale efficiently as the number of robots increases.

During discussions around this use case, the focus shifted toward enabling early validation of distributed adaptation mechanisms. In particular, the need emerged for a lightweight simulation environment that allows experimentation with multiple distributed robots, making it possible to perform simulation-based legitimation of adaptation plans and to evaluate the distributed deployment of the Trustworthiness Checker using the legitimation component.

To address this need, a lightweight multi-robot simulator was developed based on the IR-Sim library. This simulator enables evaluation of the MAPLE-K loop, with a particular emphasis on the Legitimate phase, in a distributed yet computationally efficient setting. The simulation accurately mimics the kinematic behavior of the robots and provides realistic Lidar data and field-of-view modeling. These features allow meaningful evaluation of generated adaptation plans before deployment on physical robots, while keeping the simulation overhead low. Figure 15 illustrates a simple instance of the simulation environment, demonstrating multiple robots operating in a shared space. The simulator and example scenarios are publicly available as an open-source artifact.[1]

## 5.4 Robot arm screw assistant

*Case Owner: DTI* *Current Phase: Exploration*

### 5.4.1 Description & status

This case study focuses on assisting a robotic arm with screwing and unscrewing tasks in refurbished laptops. The initial scenario considers situations in which the robot fails to accurately locate the position of a screw. In such cases, the self-adaptive system is expected to trigger recovery behaviors—such as rotational movements or exploratory search strategies—to rediscover the screw location and resume the task.

[1] https://github.com/saharnn96/RoboSAPIENS-swarm-robot-simulation

Funded by
the European Union

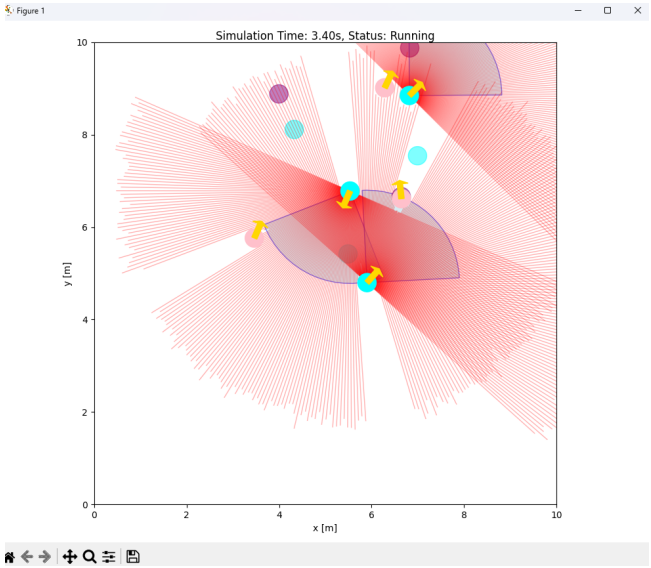



Figure 15: Distributed robots lightweight python simulation

Early discussions with use-case stakeholders identified a broad range of challenges affecting task reliability. These include visual detection difficulties caused by lighting variations or dust on the camera and other faults that may impact precision. These issues motivated the adoption of a self-adaptive architecture based on the MAPLE-K loop to handle operational uncertainty more systematically.

### 5.4.2 Conceptual architecture

Although the MAPLE-K loop was initially only partially specified for this use case, its structure has now been refined through discussions among multiple project partners. As a result, the use case has been reshaped substantially, and implementation using the RAP architecture and the RPIO toolset has started.

At the **Monitor** stage, the system observes camera images and robot joint states.

In the **Analysis** component, uncertainty quantification techniques provided by Simula Research Laboratory are incorporated to assess confidence in visual perception and execution-related signals. This enables the system to reason explicitly about uncertainty before selecting an adaptation strategy.

In the **Plan** phase, the system adapts its perception strategy by either selecting an alternative image detection model or activating an active perception algorithm to improve screw localization. These adaptations aim to reduce perceptual uncertainty and increase the likelihood of successful task execution.

The **Legitimate** component evaluates newly generated plans using historical images captured under the updated lighting conditions. This step checks whether the proposed adaptation is justified and acceptable before execution, preventing unnecessary or ineffective changes.

In addition, this use case introduces a trust-related criterion evaluated by the Trustworthiness Checker. The checker monitors the frequency of adaptive plan changes over time. If the system repeatedly switches models or strategies at a high rate,

Funded by
the European Union

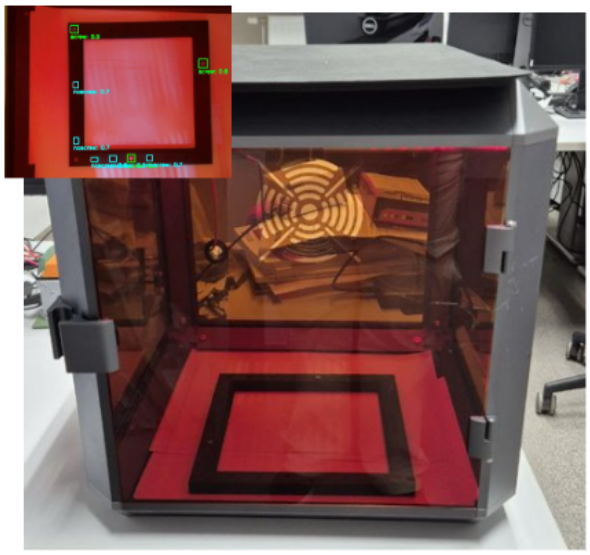


Figure 16: Remotely controlled lighting demonstrator used to study the effect of illumination changes on screw detection performance.

this behavior is interpreted as a potential indication that the current adaptive strategy is no longer trustworthy. Such situations can trigger restrictions on further adaptations or escalation to higher-level mitigation mechanisms.

The **Execute** component applies the validated adaptation in real time, while the **Knowledge** component stores information about detected anomalies, adaptation decisions, and historical outcomes to support future reasoning.

To support evaluation of this architecture, a small demonstrator was developed at the University of Antwerp. This setup provides a concrete and reproducible environment for validating the proposed adaptive architecture. Figure 16 illustrates the demonstrator used in these experiments.

Overall, this case study demonstrates how self-adaptive reasoning, combined with uncertainty awareness and trust monitoring, can improve robustness in semi-structured and uncertain industrial manipulation tasks.

# 6 Conclusion

This deliverable has presented the current status and evolution of the RoboSAPIENS tooling developed within WP5. First, it recapped the RAP software architecture and the role of the RPIO tooling, highlighting how these support the systematic development and deployment of trustworthy self-adaptive robotic systems. Next, we introduced the RoboSAPIENS workflow, which is derived from the overall RoboSAPIENS big picture and structures the engineering process across the different stages of system design, development, and deployment. Building on this workflow, we introduced the workflow tool that enables the enactment of the defined workflow. This tool supports the execution and coordination of the different engineering activities while also providing mechanisms for traceability and certification, ensur-

Funded by
the European Union

ing that engineering activities and artefacts can be tracked and validated throughout the development lifecycle. Finally, the deliverable provided an update on the project use cases, which serve as validation scenarios for the RoboSAPIENS tooling. These use cases demonstrate and evaluate the applicability of the RAP, the RPIO tool, and the workflow tool in realistic settings.

Funded by
the European Union

# References

[ANM25] Bert Van Acker, Sahar Nasimi Nezhad, and Paul De Meulenaere. Distributed architecture for the RoboSAPIENS platform. Technical report, RoboSAPIENS Deliverable, D5.2, September 2025.

[Pet66] Carl Adam Petri. Communication with automata. 1966.

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

This appendix lists only the features that have been updated or modified since Deliverable D5.2. It focuses exclusively on changes and additions, and does not repeat previously documented features.

A-KM-8 : Read Knowledge as object

| Level | Group | D5.1 status | D5.2 status | D5.3 status | RPIO version |
|---|---|---|---|---|---|
| Architecture | Knowledge Manager | nan | Todo | Done | 0.4.3 |
| **Description:** | The system supports reading and writing knowledge using defined objects. This approach ensures consistency and interoperability across components, enabling reliable exchange and synchronization of information within the distributed architecture. | | | | |
| **Linked challenges:** | Generic | | | | |
| **Validated by:** | TurtleBot 4 LiDAR occlusion, NTNU | | | | |
| **Linked requirements:** | nan | | | | |

A-AM-5 : Debug/Add unique-id for Knowledge

| Level | Group | D5.1 status | D5.2 status | D5.3 status | RPIO version |
|---|---|---|---|---|---|
| Architecture | Adaptation Manager | nan | Todo | Done | 0.4.3 |
| **Description:** | Changes for adding unique-id for the knowledge that is being stored to ensue traceability in the knowledge-base | | | | |
| **Linked challenges:** | Generic | | | | |
| **Validated by:** | TurtleBot 4 LiDAR occlusion, NTNU | | | | |
| **Linked requirements:** | nan | | | | |

# C Workflow framework features

W-M-1 : Capture RoboSAPIENS Workflow structure

| Level | Group | D5.3 status | D5.4 status | Version |
|---|---|---|---|---|
| Workflow | Modeling | To Be Validated | - | Workflow tool 1.0.0 |
| **Description:** | The model must be able to capture the complete RoboSAPIENS workflow from design to deployment. | | | |
| **Linked challenges:** | Generic | | | |
| **Validated by:** | / | | | |
| **Linked requirements:** | ID_RAP_12 | | | |

W-V-1 : Visualize RoboSAPIENS Workflow

Funded by
the European Union

| Level | Group | D5.3 status | D5.4 status | Version |
|---|---|---|---|---|
| Workflow | View | To Be Validated | - | Workflow tool 1.0.0 |
| **Description:** | The workflow tool must enable an interface to visualize the RoboSAPIENS Workflow, presenting the user with both the manual and automated activities. | | | |
| **Linked challenges:** | Generic | | | |
| **Validated by:** | / | | | |
| **Linked requirements:** | ID_RAP_12 | | | |

## W-E-1 : Execute RPIO features as services

| Level | Group | D5.3 status | D5.4 status | Version |
|---|---|---|---|---|
| Workflow | Enactment | To Be Validated | - | Workflow tool 1.0.0 |
| **Description:** | The workflow tool must be able to trigger the RPIO functions, to be used as services in the workflow. | | | |
| **Linked challenges:** | Generic | | | |
| **Validated by:** | / | | | |
| **Linked requirements:** | ID_RAP_13 | | | |

## W-E-2 : Certify enactment

| Level | Group | D5.3 status | D5.4 status | Version |
|---|---|---|---|---|
| Workflow | Enactment | To Be Validated | - | Workflow tool 1.0.0 |
| **Description:** | The tool must be able to certify whether the prescribed workflow is followed correctly without skipping steps. | | | |
| **Linked challenges:** | Generic | | | |
| **Validated by:** | / | | | |
| **Linked requirements:** | ID_RAP_14 | | | |

Funded by
the European Union

# D 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 Monitor, Analyze, Plan, Legitimate, and Knowledge (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 the DT 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 description of 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 description of 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 description of 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 description of the inter- and intra-device interfaces of the entire system | ID500 |
| ID_RAP_08 | Communication protocol support | The RAP shall support multiple standardized communication protocols for (bi-directional) inter- and/or intra-device communication with different network topologies. | - |

Table 2: 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 robot's 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 its deployment, to deployment-specific code, ready to deploy on one or more compute units. | ID501;ID503 |
| ID_RAP_12 | RoboSAPIENS Workflow | The RoboSAPIENS Workflow shall provide a clear, step-by-step workflow aligned with the RoboSAPIENS big picture, enabling the systematic construction of trust-worthy self-adaptive robotics applications. | ID505;ID506 |
| ID_RAP_13 | Workflow enactment | The workflow tool shall facilitate the execution and enactment of the RoboSAPIENS Workflow, containing both manual and automated activities. Automated activities shall be able to trigger RPIO services, such as M2M transformations, while manual activities shall enable engineers to provide required activity inputs, including annotated AADL models. | ID505;ID507 |
| ID_RAP_14 | Workflow traceability | The workflow tool shall provide traceability across the executed workflow by recording performed steps, associated inputs, and produced artifacts. This traceability information shall enable verification of correct workflow execution and provide a mechanism to ensure that the workflow is executed as intended, without tampering. | ID505;ID507 |

Table 3: High-level requirements(continued)

Funded by
the European Union