Project Number 101120990

SOPRANO Open Call

Annex 1.2: Technical Description

PROJECT PARTNERS

AEGIS Humboldtstrasse 25 38106 Braunschweig, Germany	**ATB** Wiener Strasse 1 28359 Bremen, Germany
CEA NanoInnov Bat. 862 PC 174 91191 Gif-sur-Yvette Cedex, France	**Centro Ricerche Fiat** Strada Torino 50 10043 ORBASSANO, Italy
Circular Economy Foundation Rue Breydel 34-36-40 1040 Brussels, Belgium	**E-TERRY** Neuwerkstrasse 50 99084 Erfurt, Germany
F6S 77 Lower Camden Street Dublin D02 XE80, Ireland	**FORTH** N Plastira Str 100 70013 Heraklion, Greece
Harokopio University of Athens Eleftheriou Venizelou 70 176 71 Athens, Greece	**IFADO** Ardeystrasse 67 44139 Dortmund, Germany
Jade University Friedrich Paffrath Strasse 101 26389 Wilhelmshaven, Germany	**KUKA Assembly & Test** Uhthoffstrasse 1 28757 Bremen, Germany
Netcompany-Intrasoft 2B Rue Nicolas Bové 1253 Luxembourg, Luxembourg	**NTUA** Heroon Polytechniou 9, Zographou Campus 157 80 Athens, Greece
PROFACTOR Im Stadtgut D1 4407 Steyr-Gleink, Austria	**Technology Transfer Systems** Via Francesco d'Ovidio, 3 20131 Milan, Italy
The Open Group Rond Point Schuman 6, 7th Floor 1040 Brussels, Belgium	**University of Bremen** Bibliothekstrasse 1 28359 Bremen, Germany
University of York Deramore Lane York YO10 5GH, United Kingdom	

DOCUMENT CONTROL

Version	Status	Date
1.0	First version	12 March 2025
2.0	Final version for publication	13 March 2025

TABLE OF CONTENTS

1. Introduction *4*
2. SOPRANO Components Available for Third-Party Deployment and Integration into Industrial Demonstrators *4*
2.1. Visual Spatial Localization and Mapping (VLM) *4*
2.1.1. Technical summary 4
2.1.2. Planned date of release of a prototype 5
2.1.3. Requirements for the component 5
2.2. Object Perception (OBP) *6*
2.2.1. Technical summary 6
2.2.2. Planned date of release of a prototype 6
2.2.3. Requirements for the component 6
2.2.4. Suggested bundling of this component with other SOPRANO components 7
2.3. Non-Visual Localization (NVL) *7*
2.3.1. Technical summary 7
2.3.2. Planned date of release of a prototype 8
2.3.3. Requirements for the component 8
2.4. Human Monitoring (HMO) *9*
2.4.1. Technical summary 9
2.4.2. Planned date of release of a prototype 10
2.4.3. Requirements for the component 10
2.4.4. Suggested bundling of this component with other SOPRANO components 11
2.5. Mapping of High-level Plan to Robotic Capabilities (MRC) *11*
2.5.1. Technical summary 11
2.5.2. Planned date of release of a prototype 12
2.5.3. Requirements for the component 12
2.5.4. Suggested bundling of this component with other SOPRANO components 12
2.6. MH-MR Task Allocation (CTA) *13*
2.6.1. Technical summary 13
2.6.2. Planned date of release of a prototype 13
2.6.3. Requirements for the component 13
2.7. Safety Tools (DSA & RSA) *14*
2.7.1. Technical summary 14
2.7.2. Planned date of release of a prototype 15
2.7.3. Requirements for the component 15
2.7.4. Suggested bundling of this component with other SOPRANO components 17
2.8. Scalable Simulation-Based Testing (SBT) *17*
2.8.1. Technical summary 17
2.8.2. Planned date of release of a prototype 18
2.8.3. Requirements for the component 18
2.8.4. Suggested bundling of this component with other SOPRANO components 20
2.9. Human - Digital Twin (HDT) *20*
2.9.1. Technical summary 20
2.9.2. Planned date of release of a prototype 21
2.9.3. Requirements for the component 21
2.9.4. Suggested bundling of this component with other SOPRANO components 22
2.10. AI Trustworthiness (AIT) *22*
2.10.1. Technical summary 22
2.10.2. Planned date of release of a prototype 23
2.10.3. Requirements for the component 23
2.10.4. Suggested bundling of this component with other SOPRANO components 24
2.11. Context Extraction Module (CEM) *24*
2.11.1. Technical summary 24
2.11.2. Planned date of release of a prototype 25

2.11.3. Requirements for the component 25
2.11.4. Suggested bundling of this component with other SOPRANO components 26
2.12. MH-MR Architecting Tools (MAT) 26
2.12.1. Technical summary 26
2.12.2. Planned date of release of a prototype 27
2.12.3. Requirements for the component 27
2.12.4. Suggested bundling of this component with other SOPRANO components 28
2.13. MLOps Orchestrator (MLO) 28
2.13.1. Technical summary 28
2.13.2. Planned date of release of a prototype 28
2.13.3. Requirements for the component 28
2.14. AI Model Optimizer (AIO) 29
2.14.1. Technical summary 29
2.14.2. Planned date of release of a prototype 30
2.14.3. Requirements for the component 30
2.14.4. Suggested bundling of this component with other SOPRANO components 31
2.15. Robotic Capabilities Implementation (RCI) 31
2.15.1. Technical summary 31
2.15.2. Planned date of release of a prototype 32
2.15.3. Requirements for the component 32
2.15.4. Suggested bundling of this component with other SOPRANO components 32
2.16. Advanced Visualizations (AVT) 33
2.16.1. Technical summary 33
2.16.2. Planned date of release of a prototype 33
2.16.3. Requirements for the component 33
2.16.4. Suggested bundling of this component with other SOPRANO components 34
3. How to Start Developing with a SOPRANO Component? 34

List of Main Tables

Table 1: Requirements for the Visual Spatial Localization and Mapping Component 5
Table 2: Requirements for the Object Perception Component 6
Table 3: Requirements for the Non-Visual Localization Component 8
Table 4: Requirements for the Human Monitoring Component 10
Table 5: Requirements for the Mapping of high-level plan to Robotic Capabilities Component 12
Table 6: Requirements for the MH-MR Task Allocation Component 13
Table 7: Component requirements for Safety Tools Design Safety Assessment (a) 15
Table 8: Component requirements for Safety Tools Runtime Safety Assessment (b) 16
Table 9: Requirements for the Scalable Simulation-Based Testing Component 19
Table 10: Requirements for the Human Digital Twin Component 21
Table 11: Requirements for the AI Trustworthiness Component 23
Table 12: Requirements for the Context Extraction Module Component 25
Table 13: Requirements for the MH-MR Architecting Tools Component 27
Table 14: Requirements for the MLOps Orchestrator Component 28
Table 15: Requirements for the AI Model Optimizer Component 30
Table 16: Requirements for the Robotic Capabilities Implementation Component 32
Table 17: Requirements for the Advanced Visualizations Component 33

1. Introduction

The goal of the SOPRANO Open Call is to recruit SMEs/startups to exploit the SOPRANO technologies to develop new demonstrators or enhance an existing demonstrator supporting MH-MR scenarios in industrial settings.

This document provides a detailed description of the SOPRANO architecture and services available to beneficiaries under their participation in the SOPRANO Open Call Program (16 months duration).

The document shall be treated as an extension of the SOPRANO Guidelines for Applicants and should be considered when developing and submitting a proposal under the SOPRANO Open Call.

2. SOPRANO Components Available for Third-Party Deployment and Integration into Industrial Demonstrators

2.1. Visual Spatial Localization and Mapping (VLM)

2.1.1. Technical summary

A robot that navigates a dynamic and partially known environment needs to know its position with respect to the surroundings. This requires using on-board sensors to construct and maintain a map of the environment, while simultaneously keeping track of the robot's pose (i.e., position and orientation) within that environment. This task is achieved by the visual spatial localization and mapping (VLM) module that develops a visual simultaneous localization and mapping (vSLAM) system which relies solely on information extracted from images. The vSLAM pipeline for real-time location estimation is coupled with a component that provides absolute localization, i.e. expresses the camera position and orientation in a predefined coordinate system attached to the environment and being external to the camera.

VLM assumes that sufficient texture exists in the environment so that distinct, sparse visual features (aka keypoints) can be extracted from the employed images. It provides frequent 6D absolute camera pose updates. The absolute localization component relies on representations constructed off-line to capture the structure and appearance of the environment in a way that accommodates fast look-ups and robust matching for images from a running vSLAM session without relying on a pre-deployed infrastructure.

2.1.2. Planned date of release of a prototype

Mid-2025

2.1.3. Requirements for the component

VLM outputs the real-time 6D pose of the employed camera. Its requirements are specified in Table 1 below.

Table 1: Requirements for the Visual Spatial Localization and Mapping Component

Component Name	**Visual Spatial Localization and Mapping (SC01)**
Type (Software/Hardware/Both)	**Software**
Short Description	**Processes video captured by depth (RGB-D) cameras using visual SLAM algorithms in order to localize the robot and coarsely map its surroundings**
Employed at: Run Time / Design Time / Both	**Runtime**
Input requirements	
Input Data from Knowledge Base	**Yes**
Input Data from Sensors/Context	**Yes**
Format of Expected Input	**RGB-D data in raster format; accompanying calibration metadata**
Triggered by	**N/A (runs continuously)**
Interfaces	**-**
Output requirements	
Main Outputs	**6D robot pose estimate**
Output Data to Knowledge Base	**Yes**
Nature of Expected Output	**Timestamped poses (e.g., ROS message)**
Hardware & software requirements	
Software Requirements	**Ubuntu and Docker (libraries such as OpenCV, Boost, Eigen, OpenGL/GLUT, g2o, librealsense, ROS, etc. will be included in the docker image)**
Hardware Requirements	**PC (#cores >= 6, RAM>=32 GB, GPU w VRAM ≥ 6GB), RGB-D camera, network interface, large disk (>= 1Tb)**
Communications	**pub/sub platform**
Special Communication Requirement	**-**
Integration Requirements	**RGB-D camera**
Deployment Requirements	**Docker, screen, network connectivity, remote access control**

2.2. Object Perception (OBP)

2.2.1. Technical summary

The Object Perception (OBP) component aims to provide real-time detection and 6D pose estimation of objects within the robot's operating environment. Depending on the use case, object perception may range from object detection, identifying and classifying objects in 2D and estimating their position in 3D, to 6D object pose estimation, providing the precise position and orientation of object(s) to support robotic manipulation and grasping. OBP exploits input from RGB and RGB-D sensors to estimate the 6D pose of a specific set of objects, even in cases of partial occlusions, ensuring reliable performance in diverse scenarios. This capability supports various robotic manipulation tasks such as grasping, screwing/unscrewing. OBP also automates the process of training data generation for 6D pose estimation, facilitating scalability and efficient dataset augmentation. This is essential for the continuous improvement of object detection and pose estimation algorithms.

During runtime, upon initialization, the OBP component processes visual data from offboard and onboard cameras, to detect objects of interest. This involves identifying and classifying objects in 2D or 3D to ensure safety and operational efficiency. Concurrently, upon request, OBP will detect objects and estimate their position or 6D pose in real-time, along with an unbiased confidence measure for the quality of the 6D pose, before any robotic action (i.e. grasping) is attempted. Outputs from OBP, including 2D segmentation masks, 6D poses, and confidence scores, are exploited by other SOPRANO components released for deployment and integration in the framework of the Open Call, such as Human Monitoring, MH-MR Task Allocation (CTA), Robotic Capabilities Implementation, and Safety Tools.

2.2.2. Planned date of release of a prototype

Mid-2025

2.2.3. Requirements for the component

The development of OBP primarily occurs within the Linux environment, utilizing Python as the primary programming language, with additional software requirements such as libraries for the interface with the RGB-D cameras to be utilized and the GPU (the exact software requirements are pending determination). Hardware requirements include RGB-D cameras to capture RGB or RGB-D data, essential for the component’s object perception tasks. Given its reliance on deep learning methodologies, OBP requires an NVIDIA GPU (VRAM ≥ 16GB, Compute Capability ≥ 8.6) for efficient execution of complex neural network algorithms. Moreover, OBP should operate within resource-constrained embedded systems like NVIDIA Jetson, balancing between performance and power consumption. Communication protocols such as Kafka and ROS2 facilitate seamless interaction with external systems and devices. For streamlined deployment processes, Docker emerges as the preferred method, offering scalability, consistency, and portability across diverse computing environments.

Table 2: Requirements for the Object Perception Component

Component Name	Object Perception (SC02)
Type (Software/Hardware/Both)	Software
Short Description	Exploit input from camera sensors to detect in 2D and 3D objects as well as estimate the 6D pose (translation and rotation) of objects to support robot manipulation

Employed at: Run Time / Design Time / Both	Runtime
Input requirements	
Input Data from Knowledge Base	Yes*(object models, DL model)
Input Data from Sensors/Context	Yes
Format of Expected Input	RGB-D data , calibration data
Triggered by	Another module for grasping /manipulation action or runs continuously for monitoring action
Interfaces	ROS
Output requirements	
Main Outputs	6D object pose, 3D position, 2D segmentation mask, class id, object id, BBox, confidence score
Output Data to Knowledge Base	Yes
Nature of Expected Output	ROS message
Hardware & software requirements	
Software Requirements	Linux (Ubuntu and Linux 4 Tegra) environment, Python, PyTorch, C++, CUDA, ONNX Runtime, librealsense, OpenCV, Kafka, ROS, Docker (depending on the UC)
Hardware Requirements	ARM: NVIDIA Jetson AGX (# CPU cores ≥ 8, RAM ≥ 32 GB, # GPU cores ≥ 512, GPU Compute Capability ≥ 7.2, storage ≥ 1 TB) x86: # CPU cores (physical) ≥ 14, RAM ≥ 32GB, storage ≥ 1TB, NVIDIA GPU (VRAM ≥ 16GB, Compute Capability ≥ 8.6) RGB-D cameras: Intel RealSense D405, Intel RealSense D456 or D457, ZEDx, ZED2i
Communications	Kafka, ROS
Special Communication Requirement	None
Integration Requirements	Docker, RGB-D camera
Deployment Requirements	Docker, network connectivity, remote access control

2.2.4. Suggested bundling of this component with other SOPRANO components

The component can be integrated with Human Monitoring (HMO), AI Model Optimizer (AIO), Mapping of High-level Plan to Robotic Capabilities (MRC) and Robot Capabilities Implementation (RCI).

2.3. Non-Visual Localization (NVL)

2.3.1. Technical summary

Robot localization is the process of determining where a mobile robot is located with respect to its environment. The Non-Visual Localization (NVL) component thus targets the real-time monitoring of the robot's position and orientation, as well as its surrounding environment, using non-visual (input) data and sensors. Additionally, information on its dynamics (e.g. velocity) as well as a measure of the uncertainty of the location estimate can be conveyed. In rough terms, NVL can be distinguished into two broad categories, depending on the environment in which the robot operates. These are outdoor and indoor localizations. Within the framework of SOPRANO, both cases are considered.

For the first case (outdoor positioning), localization is based on the integration of GNSS and IMU sensors. These input data can provide accurate estimates of both the robot's position and its dynamics (velocities and instantaneous acceleration). The challenge here, however, is the implementation of a robust and efficient data fusion and estimation algorithm. For this purpose, an Extended Kalman Filter will be employed, integrating sensor measurements to gain optimal, real-time state estimates. This configuration allows for correctly dealing with noisy data (if such a situation arises) and additionally generates not only an estimate of the robot's location but also a measure of the uncertainty of the location estimate. To transform these estimates into something useful for the robot’s operation and share information with other operational nodes, these estimates need to be transformed to a “local” reference frame. A prerequisite for this task will be the availability of a map of the area of operation. The map shall be available during operation, and the component will thus be able to deliver real-time position estimates within the area of operation. Additionally, geometric information on Points of Interest will be available on user request (e.g. distance) which can be correlated with navigation and/or other working components.

In the case of indoor localization, the use of GNSS is, in general, not possible. To perform robust localization, a set of sensors will be used, depending on the use case. These sensors include: (a) a laser scanner offering 2D LiDAR measurements as well as a reliable solution for maximizing safety and preventing potentially unsafe situations, coupled with (b) mm-Wave Radar Sensors coupled with high resolution ultrasound sensors; these sensors use frequency to calculate distance and will be installed on each side of the robot and on the arm(s). Thus, the sensors measure the target range and its relative velocity simultaneously. Yet another option will be direction-finding and indoor positioning via the well-known Angle-of-Arrival method using wireless sensors (e.g. Bluetooth). The development of the additional subsystems that will be embedded in the indoor robotic platform will not disrupt current protocols and data transfer/processing procedures. The use of the current data transfer bus will be the top priority in the embedding procedure. Mechanical and electrical characteristics will be selected to be compatible with each platform’s characteristics in order to avoid excessive use of adapting and reconfiguring processes and devices. The whole processing sequence will run on an independent processing platform in order for only the required data to be able to be sent to the available data bus, and possible failures will not affect any existing automated operation. Results and outputs of both indoor and outdoor localization will be produced in real-time and instantaneously made available to other nodes of operation.

2.3.2. Planned date of release of a prototype

Mid-2025

2.3.3. Requirements for the component

Detailed hardware & software requirements for NVL can be found in Table 3 below.

Table 3: Requirements for the Non-Visual Localization Component

Component Name	Non-Visual Localization (SC03)
Type (Software/Hardware/Both)	Software
Short Description	Non-visual sensing devices coupled with software components

	to accurately detect the location and orientation of objects of interest
Employed at: Run Time / Design Time / Both	Runtime
Input requirements	
Input Data from Knowledge Base	Yes. Points of interest for area under consideration and map of area.
Input Data from Sensors/Context	Yes. Sensor data/measurements of the onboard (relevant) sensors (described above).
Format of Expected Input	Sensor data in custom format (depending on hardware). Points of interest in xml or serialized format (e.g. json).
Triggered by	Requested by the user or automatically delivered in real-time.
Interfaces	ROS-msg, ROS configuration, serialized messages at output, Kafka (optional/experimental)
Output requirements	
Main Outputs	Position estimates, kinematic information, spatial information
Output Data to Knowledge Base	Plain ascii or serialized data (e.g. json) containing timestamp, robot coordinates (in local system), kinematic information (e.g. velocity components and norm).
Nature of Expected Output	Kinematic and dynamics info of the robot at selected times; spatial and geometric info.
Hardware & software requirements	
Development Environment	On premise (ROS, python, C if needed).
Software Requirements	Python (version 3), C compiler preferably gcc or clang (if needed), ROS (selected nodes), Docker
Hardware Requirements	OS, ROS
Communications	TCP/IP
Special Communication Requirement	None
Integration Requirements	ROS, Docker, Python
Deployment Requirements	ROS, Docker, Screen

2.4. Human Monitoring (HMO)

2.4.1. Technical summary

The HMO component is, among others, responsible for detecting the location and body pose of multiple humans. This functionality is provided by the 3D Human Pose Estimation (3D-HPE) subcomponent. 3D-HPE provides a vision-based framework for real-time, marker-less 3D human pose estimation and, optionally, the classification of ergonomic postures performed in real, dynamic operational environments. The primary input is online visual data streams (RGB and optionally depth) acquired by one or more camera sensors, which can be stationary or mounted on a moving platform, to monitor humans in indoor or outdoor settings.

3D-HPE primarily focuses on determining the location and 3D articulated body pose of one or multiple humans. Given a standard 3D skeletal body model (body_25 or coco), it dynamically identifies the coarse body location and orientation in the form of a 2D or 3D rectangular bounding box and the 2D/3D positions and orientations of up to 25 anatomical key points of the human body, i.e. body joints (e.g., elbows, knees, shoulders, hips). Confidence scores are also provided. 3D-HPE supports the dynamic adaptation of the estimated human pose to different human body metrics and provides robustness under body occlusions and challenging environmental conditions. The output is represented in the camera coordinate system or any other global coordinate system of the environment provided. Optional functionality and related output (ergonomic posture id and score) for evaluating the estimated body postures is also supported with the aim to enhance human ergonomic awareness and safety. The component performs in real-time, provided that its hardware requirements are met.

2.4.2. Planned date of release of a prototype

Mid-2025

2.4.3. Requirements for the component

3D-HPE is developed on top of a core C++ library utilizing Python (as a wrapper) as the primary programming language with additional third-party open libraries for the interface with the RGB-D cameras and the GPU processing of deep network models. Hardware requirements include RGB or RGB-D cameras to capture time-synchronized colour and optionally depth data streams to drive the component's human pose estimation and postural assessment functionalities. The component requires a workstation desktop PC equipped with NVIDIA GPU (VRAM $\geq$ 16GB, Compute Capability $\geq$ 8.6) for efficient execution of complex neural network algorithms. For streamlined deployment, Docker is the preferred method, offering scalability, consistency, and portability across diverse computing environments.

Table 4: Requirements for the Human Monitoring Component

Component Name	3D Human Pose Estimation (3D-HPE) (SC04)
Type (Software/Hardware/Both)	Software
Short Description	Processes video captured by one or more colour and depth cameras for human pose estimation in 3D and postural assessment
Layer	Local
Employed at: Run Time / Design Time / Both	Runtime
Input requirements	
Input Data from Knowledge Base	Yes
Input Data from Sensors/Context	Yes
Format of Expected Input	Online visual sensory data (RGB-D images/image sequences, point clouds), ground truth annotation data (cvs, json, bvh), static data related to the cameras' placement and calibration parameters.
Triggered by	Manual by system operator - Auto (on request by external components)
Interfaces	POST (HTTP) messaging protocol (publish-subscribe or

	queuing), Docker
Output requirements	
Main Outputs	Body location in 2D/3D, skeletal body pose in 2D/3D, classified body posture and ergonomic score, confidence scores
Nature of Expected Output	Visualization for estimated human parameters/postures. At runtime: message types (data format needs to be defined, e.g. <Message ID, topic type, score, location, pose, workerID, timestamp>), Offline: Datasets of videos, annotations related to human monitoring tasks.
Hardware & software requirements	
Development Environment	On-premise and cloud environments
Software Requirements	Docker, Python, PyTorch, Tensorflow, Docker
Hardware Requirements	PC (minimum requirements #cores >= 12, RAM>= 24 GB, GPU-accelerated Compute capability>8.5) RGB-Depth cameras (stereo or ToF, e.g. StereoLabs ZED
Communications	Messaging protocol over local network
Special Communication Requirement	Real-time communication, messaging format able to accommodate folded data and list of structured data (e.g. 1D or 2D list of numbers), json format for messages, REST API endpoints.
Integration Requirements	Docker. Onboard online data processing and RGB-D data acquisition (camera).
Deployment Requirements	Docker, screen, network connectivity, remote access control

2.4.4. Suggested bundling of this component with other SOPRANO components

The component can be integrated with the Object Perception (OBP) component.

2.5. Mapping of High-level Plan to Robotic Capabilities (MRC)

2.5.1. Technical summary

Within the SOPRANO system, the task mapping component is designed to provide robust functionalities for the decomposition of high-level plans into executable robotic actions, ensuring adaptability and customization across robotic platforms and industrial scenarios. Platforms considered are the UR10 Platform and FELICE mobile manipulator platforms, and the KUKA platform. The component's primary services include task decomposition, capability mapping, and execution planning. At design-time, the component decomposes complex assembly plans into elementary actions, identifies the suitable robotic capabilities required for each action, and maps these actions to various robotic platforms, considering their unique kinematic and dynamic properties. The component leverages a behaviour tree-based framework as a potential approach to represent and manage the execution of tasks, ensuring modularity, flexibility, and readability of the task execution plans. Additionally, it adapts dynamically to environmental changes and platform variations, facilitating seamless task execution and enhancing the system's scalability and efficiency in multi-robot settings. Dependencies on other components include integration with perception modules for real-time environmental feedback and interaction with motion planning and robot control modules to ensure accurate and effective task execution.

2.5.2. *Planned date of release of a prototype*

Mid-2025 - early prototype

2.5.3. *Requirements for the component*

Table 5: Requirements for the Mapping of high-level plan to Robotic Capabilities Component

Component Name	Mapping of high-level plan to Robotic Capabilities (SC05)
Type (Software/Hardware/Both)	Software
Short Description	Development of algorithms & methodologies for decomposing high-level plans into elementary robotic actions accommodating variations in platform capabilities - UR10 Platform and FELICE mobile manipulator platforms; KUKA platform
Employed at: Run Time / Design Time / Both	Design Time
Input requirements	
Input Data from Knowledge Base	Yes
Input Data from Sensors/Context	Likely required (Final Prototype with configurable mapping)
Format of Expected Input	ros2 topic/ros2 actions
Triggered by	SOPRANO ID6
Interfaces	ROS 2
Output requirements	
Main Outputs	Task execution plan,
Output Data to Knowledge Base	n/a EP; New mapping possibilities FP
Nature of Expected Output	ros2 topic/ros2 actions
Hardware & software requirements	
Software Requirements	Docker, Python
Hardware Requirements	Docker
Communications	Ros, whatever useful to integrate with Knowledge-Base
Special Communication Requirement	None
Integration Requirements	Partner-Actions implemented as Ros2-Action
Deployment Requirements	Partner-Actions implemented as Ros2-Action

2.5.4. *Suggested bundling of this component with other SOPRANO components*

The component can be integrated with MH-MR Task allocation (CTA) and Object Perception (OBP).

2.6. MH-MR Task Allocation (CTA)

2.6.1. Technical summary

The CTA module is responsible for coordinating and enhancing collaboration among agents in collaborative tasks. It aims to maximize their effectiveness by optimizing mutual support and exploiting their individual skill sets to benefit the team.

This functionality is achieved through the Daisy Planner (DP) that considers the assignment of tasks to the agents participating in a collaborative task. The CTA-DP uses prior knowledge regarding the primitive actions each agent is able to execute and how these actions can be combined in a meaningful way to implement complex behaviours. The CTA-DP guides agents in sequentially and efficiently executing assigned behaviours. It also accounts for overlapping behaviours among different agents, ensuring their timely interaction within collaborative tasks. In that way it enhances the overall system performance and supports effective teamwork among agents in complex environments.

During operation it assesses the designed action assemblies against criteria related to the performance of each agent on the individual actions and other designer-specified criteria (e.g. task priorities, or energy consumption), to specify the next agent task that will make the agent maximally useful for the team. The CTA-DP enhances the coordination of the collaborating agents by taking into account the preconditions for the execution of the tasks undertaken by an agent and directing the other team members to act in a manner that satisfies these preconditions.

The CTA-DP uses background information stored in an initialization file which regards the tasks addressed in the given work environment, the preconditions for the execution of tasks and the skills of the involved robots which will be evaluated under relevant criteria (e.g. implementation time per agent).

The CTA-DP is provided as a ROS2 executable package that can be easily configured by the user/designer to support the implementation of multi-agent tasks.

2.6.2. Planned date of release of a prototype

Autumn 2025

2.6.3. Requirements for the component

The CTA-DP is a lightweight component that operates as a node within a ROS2 environment, enabling seamless communication with collaborating agents. Additionally, it is necessary to have two-way communication with all perception and action components in the given application to make timely and informed decisions. The CTA-DP outputs high level commands directed to the collaborating agents, following the task and action names specified by the user during initialization.

Table 6: Requirements for the MH-MR Task Allocation Component

Component Name	MH-MR Task Allocation (SC06)
Type (Software/Hardware/Both)	Software
Short Description	The module assigns tasks to the members of a multi-agent team in order to coordinate and enhance collaborative operation
Employed at: Run Time /	Runtime

Design Time / Both	
Input requirements	
Input Data from Knowledge Base	No. It needs static information that can be provided by the designer in a YAML structured initialization file. The information required includes: the task involved in each collaboration scenario, the prerequisites for each task, the behavioural skill of the participating agents in the form of performance scores (e.g. implementation time, success rate) and other application specific criteria specified by the user (e.g. energy consumption).
Input Data from Sensors/Context	Yes
Format of Expected Input	ros2 topic/ros2 actions
Triggered by	No particular triggering - Continuous monitoring of robot & environment state changes
Interfaces	Consumes messages reporting state changes. Determines the next robot task
Output requirements	
Main Outputs	Task allocation, short-scale decisions
Output Data to Knowledge Base	No
Nature of Expected Output	ros2 topic/ros2 actions
Hardware & software requirements	
Software Requirements	ROS2
Hardware Requirements	No
Communications	ros2 topic/ros2 actions
Special Communication Requirement	ROS2
Integration Requirements	ROS2 environment
Deployment Requirements	ROS2 environment

2.7. Safety Tools (DSA & RSA)

2.7.1. Technical summary

The Safety tools framework is intended to enable supervision of human safety in human-robot interaction workplaces. It is composed of two components: The safety assessment module that occurs at design stage (DSA) and the safety assessment that occurs at runtime (RSA). The overall framework is capable of dynamically examining the risk associated with operations in Human-Robot collaborative workplaces, and mitigating the risk based on adaptive safety strategy that ensures accident avoidance while reducing untimely interruptions of operations.

The DSA component is used at design time to conduct an offline scenario-based risk assessment. It relies on the system's specifications including system behavioural and architectural information, environmental and operational context information. Environmental information is captured as an Operational Design Domain (ODD) model to ensure that the overall scenario-space of the robot is correctly and somehow completely formalized. The system architectural information is modelled through a design model. From these inputs, one can identify the causal chains of events (failure scenarios) that may lead to accidents or undesirable behaviours in the working environment. Note that the failure scenarios identification can be completed with a review of existing accident databases and an analysis of safety standards. The damage the identified failures scenarios may cause are then evaluated using chosen criticality level metrics to better identify those on which risk reduction measures must be taken as a priority.

Risk reduction measures (preventive measures, and/or corrective measures) must be recommended according to the assessed risk to increase safety and reduce accidents. One must define a context-dependent and configurable mitigation strategy to trigger control actions aiming at avoiding accidents according to safety thresholds. Those control actions depend also on the mode of operation of the machinery, e.g. remotely controlled, semi-automatic, or fully autonomous. Note that the safety analysis defines rules to help enforce some existing control actions that the robot can perform (as part of the robot capabilities) whenever a critical situation is identified, but it is not developing novel control actions.

2.7.2. Planned date of release of a prototype

The planned release of the Safety Tools prototype is divided into two key stages to reflect the progressive development of the methodology and tools:

- Mid-2025 - a prototype of the DSA module, encompassing our methodology along with complementary classical safety methods (FMEA, FTA, SHA, and STPA). This release will include both the analytical framework and the necessary integration with ODD modelling to support comprehensive scenario-based risk assessments.
- End-2025 - the RSA module, focusing on real-time risk monitoring and adaptive safety management.

2.7.3. Requirements for the component

Detailed hardware and software requirements for DSA can be found in Table 7 below:

Table 7: Component requirements for Safety Tools Design Safety Assessment (a)

Component Name	Safety Tools Design Safety Assessment (SC07a)
Type (Software/Hardware/Both)	Software
Short Description	The module uses the system's specifications (including architectural design information), environmental & operational context to conduct scenario-based risk assessment at design time
Employed at: Run Time / Design Time / Both	Design Time
Input requirements	
Input Data from Knowledge Base	Yes
Input Data from Sensors/Context	No
Format of Expected Input	text-based, UML/XMI format
Triggered by	N/A
Interfaces	N/A
Output requirements	

Main Outputs	Customized risk analysis, model of risk scenarios, risk assessment report, risk mitigation strategies
Output Data to Knowledge Base	risk analysis, risk mitigation
Nature of Expected Output	UML/Ecore model (xml), Report
Hardware & software requirements	
Software Requirements	Eclipse Papyrus standalone plugins
Hardware Requirements	no special hardware requirement foreseen
Communications	No specific
Special Communication Requirement	No specific
Integration Requirements	No specific
Deployment Requirements	No specific

At operational time, the RSA implements the concepts of situational awareness and adaptive safety. The RSA relies on the robot's perception system to gather in real time information about the entities present in the supervised workspace, and the results from the DSA component i.e. the safety assessment results and the mitigation strategy.

Data collected from the sensing system are processed to identify environmental parameters and reliable positioning/detection of the surrounding entities in the supervised workplace, together with additional characteristics such as their speed, orientation, relative distance, etc. The DSA results are encoded as a safety dynamic model to predict potential hazardous situations in real-time. It uses an event-driven algorithm to evaluate automatically the risk level incurred for the humans in operation from the observed events compared to the formalized safety model and launch accordingly the appropriate risk reduction measures. The reduction measures may be either implemented as a visual, audible or text alert in a console. Table 8 presents detailed hardware & software requirements for RSA.

Table 8: Component requirements for Safety Tools Runtime Safety Assessment (b)

Component Name	Safety Tools Runtime Safety Assessment (SC07b)
Type (Software/Hardware/Both)	Software
Short Description	The module makes use of the DSA results, sensors data to conduct risk assessment runtime
Employed at: Run Time / Design Time / Both	Run Time
Input requirements	
Input Data from Knowledge Base	No
Input Data from Sensors/Context	Yes
Format of Expected Input	xml/ros
Triggered by	Event-driven from input components or collected periodically
Interfaces	depending of the data collection environment
Output requirements	

Main Outputs	automated safety supervision component, safety enforcement component
Output Data to Knowledge Base	to be defined (model, text-based)
Nature of Expected Output	ROS, code
Hardware & software requirements	
Software Requirements	Eclipse Papyrus standalone plugins
Hardware Requirements	robotic platform related
Communications	No specific
Special Communication Requirement	No specific
Integration Requirements	No specific
Deployment Requirements	No specific

The DSA component requires Eclipse Papyrus and model-driven engineering (MDE) tools to allow the ODD modelling, the system design modelling and the offline risk assessment. The RSA is implemented in the convenient language for its on-boarding on simulation platforms, on the robot, on an external device dedicated to safety monitoring. In general, it requires Python and ROS/ROS2.

2.7.4. Suggested bundling of this component with other SOPRANO components

The Safety Tools component is designed to operate independently, ensuring robust safety assessment and monitoring within human-robot collaborative environments. Its core functionalities do not require mandatory interaction with other SOPRANO components, making it a standalone solution for many applications.

However, optional integration with certain components enhances its capabilities and broadens its use. For example, integrating with SBT allows for the validation and testing of safety rules and mitigation strategies under diverse simulated scenarios. SBT can leverage the ODD scenarios and safety rules generated by DSA, enabling more comprehensive simulation campaigns. Additionally, RSA outputs, such as real-time safety metrics, can provide valuable data for performance tracking during simulations.

2.8. Scalable Simulation-Based Testing (SBT)

2.8.1. Technical summary

The SOPRANO SBT module aims to anticipate via simulation-based testing faults/failures that may impact the MH-MR composite system.

The SOPRANO user will initialize the process by specifying parameters for the simulation, including a testing space configuration and candidate scenario to be analysed. Following this, a testing loop will be executed, as described in the following paragraph, to find the specific faults/failure modes that the scenario exhibits, optionally incorporating feedback from other SOPRANO tools. According to the test configuration, the SBT module will inject specific manipulations such as distorted, deleted and delayed messages, and altered configuration or runtime parameters. It will then assess the scenario with respect to the given performance and safety metrics, and focus the testing upon the most significant incidents. The SBT component will provide experimental results to the user and other SOPRANO components about the set of faults and failures discovered during testing of the selected scenario, together with (optionally) a trained predictor for anticipating simulation performance given that scenario.

In order to search the large space of potential faults/failures in a given scenario in a timely manner, a distributed architecture will be used for scalable and model-based SBT upon a cluster of collaborating PCs. A centralized experiment manager maintains an evolutionary experimental loop, keeping a population of test configurations and associated results, and dispatching tests to worker nodes. Containers or archives to execute the simulation will be transmitted to the worker from a data repository and augmented for SBT testing. For each test, generated code is shared from the experiment runner to participating worker nodes. This auto-generated code includes test runners, which specify a selection of faults/failures for that test, and their activation conditions. With the augmented simulator in place, the worker nodes will then execute the test runners, which launch the simulation and inject the fault/failures as configured. Result metric values (which quantify violations of MH-MR performance/safety requirements) and simulator logs are tracked upon the relevant worker node and communicated back to the central experiment runner. In order to guide the testing campaign intelligently, information from the scenario structure, and statistics of previously executed simulations can be used to predict performance and safety violation metric values and guide the direction of a new testing campaign. The intent behind this is for the component user to discover the vulnerabilities (for example, components or scenario features which are sensitive to or intolerant of these manipulations) and allow end users to refine their design in simulation before deployment.

2.8.2. Planned date of release of a prototype

The planned release of STB is partitioned into two key stages to reflect its progressive development:

- By mid-2025 - SBT prototype with the scalable architecture will be released
- By end-2025 - the SBT component enhanced with the predictor will be released

2.8.3. Requirements for the component

The SBT component requires Linux (recommended: Ubuntu 22) or higher installed on the experiment manager PC and worker nodes. It is possible to run both the experiment manager and worker node upon the same PC in early-stage testing, but ideally a cluster would be used with multiple PCs upon the same LAN for low-latency connectivity between them. In order for SBT to test a custom or proprietary simulator, the simulator must provide an API capable of making the necessary manipulations at runtime (for example, altering or deleting messages or deactivating components/changing parameters). For ROS/ROS2/Gazebo simulations, this is implemented via the standard ROSbridge component. If source code is available for the simulator, it may be possible to add a suitable interface.

The experiment manager is provided as a Docker image and provides a GUI to allow experiment reconfiguration and execution via the testing DSL interface in Eclipse. The experiment manager system uses an internal SSH daemon/rsync to interchange generated code with worker nodes. Upon the worker nodes, Python and Maven are required, together with Docker and ROS2. The associated Java dependencies for SBT are downloaded by Maven during compilation. The Java test runners on the worker node will communicate with the simulations either over ROSbridge/jrosbridge (ROS/ROS2) or a gRPC interface for other robotic simulators (e.g., the DDD simulator from the project partner TTS). MS Windows specific simulations would have to be negotiated to verify that they can be supported.

Detailed hardware and software requirements for SBT can be found in Table 9 below:

Table 9: Requirements for the Scalable Simulation-Based Testing Component

Component Name	Scalable Simulation-Based Testing (SC08)
Type (Software/Hardware/Both)	Software
Short Description	Simulation-based testing tools for MH-MR systems for testing the functionality and robustness of systems in simulation.
Input requirements	
Input Data from Knowledge Base	Worker behaviour status from knowledge base and context, collision information
Input Data from Sensors/Context	SBT doesn't receive runtime sensor data, only simulated sensors. Context change may trigger a fault by conditions, or context may be tracked for performance metrics
Format of Expected Input	Messages according to agreed formats
Triggered by	Event-driven from input components (e.g., ROS topics)
Interfaces	gRPC/rosbridge - also JSON or typed messages (ROS)
Output requirements	
Main Outputs	Result metrics, Logs of failure configurations, Failure predictor
Output Data to Knowledge Base	Failure configurations for selected scenario/failure predictor
Nature of Expected Output	Simulation results (models) containing failure cases (Pareto front)
Hardware & software requirements	
Development Environment	Cloud or locally provisioned cluster of PCs
Software Requirements	On workers: Python, Maven, Docker, Java, ROS2 and associated JAR/pip packages. Simulator environment (e.g., Gazebo, ROS). On experiment runner; Eclipse and MDE tools
Hardware Requirements	Cluster of PCs with IP connectivity for running simulations - probably 8-core i7 or above on each worker, with GPU
Communications	TCP/IP between nodes, Kafka message broker on each worker node. SSH/SMB filesystem or other data sharing mechanism on data manager/experiment runner
Special Communication Requirement	Direct low-latency connection between each worker and TTS simulator.
Integration Requirements	Needs communications with the safety tools to extract risks for metric templates, as well as other components mentioned above at simulation time
Deployment Requirements	Cluster of PCs as stated above, ideally with the number of workers as large as the population size. AWS may not be cost-effective given the need for GPUs

2.8.4. Suggested bundling of this component with other SOPRANO components

Bundlings are compulsory since SBT can operate independently, but it may also operate alongside the following SOPRANO components:

- Safety Tools - DSA and RSA
- Human-Digital Twin (HDT)

2.9. Human - Digital Twin (HDT)

2.9.1. Technical summary

The Human Digital Twin (HDT) component provides a virtual representation of human workers, capable of simulating real human movements during manufacturing and assembling tasks as well as interactions with robots within a work environment. The HDT component enables importing a digital representation in “GLTF” format of a worker’s body, also referred to as a mesh, along with a digital skeleton. Additionally, a HDT is characterized by other parameters such as skills required to perform a task. End users can define worker skills using specific stochastic distributions which represent the ability to execute a task according to scheduled timing and within tolerance limits. The HDT prototype can be further customized by developing appropriate Java algorithms.

This HDT prototype is seamlessly integrated into our Digital Twin Service platform. This platform allows the creation of a virtual working environment where digital twins of workers and robots collaborate and interact to complete tasks. The ultimate goal of the HDT component, integrated into the Digital Twin Service platform, is to investigate various situations of human-robot collaboration within the same fenceless workspace using discrete event simulation. This component facilitates the identification of potential safety failure points and minimizes the risk of unforeseen collisions. A robot collision detection module has been developed, based on two-three different customizable safety zones. Each zone can trigger specific robot reactions when a worker or an obstacle is detected. For instance, the robot stops immediately when a worker enters the “stop zone” and resumes its motion automatically once the area is cleared. These robot behaviours can be tailored by developing specific Java algorithms and tested within the virtual environment. Additionally, the virtual robot can be equipped with virtual sensors, based on infrared or safety laser scanner technologies, to detect workers and obstacles. These sensors emulate the behaviour, data and characteristics of real physical sensors and provide raw data for the collision detection module.

The virtual robot can also be connected to the real robot controller to test its robustness in avoiding obstacles and workers. In this configuration, the virtual environment simulates the robot’s movements in space and sends data generated by the virtual sensors to the real controller. This data is processed by the safety procedures implemented in the robot controller system, and the corresponding robot commands are sent back to the virtual one. This closed-loop allows testing and debugging the robot’s safety procedures to enhance their reliability and robustness. In fact, the HDT component can interact with the scalable simulation-based testing (SBT) tools to analyse many different scenarios aiming to identify failures in the safety procedures in the robot control unit, ensuring a safer and more reliable collaborative work environment between workers and robots. 2.9.2. Functionalities developed specifically for the SOPRANO industrial use cases

A specific virtual environment has been created for representing a SOPRANO industrial use case using our Digital Twin Service platform. In particular, a virtual mobile platform equipped with an anthropomorphic robot and its sensor system has been developed, and their behaviours have been simulated. An HDT instance was, then, created to represent a worker collaborating with the virtual robot.

The available DT prototypes in the Digital Twin Service platform can be reused to create a new simulation environment. However, they require customization by adjusting input parameters and developing specific Java algorithms to accurately simulate the real behaviours of the physical assets of an MH-MR system.

2.9.2. Planned date of release of a prototype

End-2025

2.9.3. Requirements for the component

The HDT component functions exclusively within our Digital Twin Service Platform, as it enables the instantiation of digital twins and the creation of simulation models. This platform operates under Windows or Linux operating systems. Utilizing the HDT component requires the development of a new simulation model representing the real system to be analysed. Simulating MH-MR systems involves creating a virtual representation of robots, workers and workspace as well as developing safety and operational rules to ensure accurate tasks simulation. Proficiency in creating and managing simulation models along with strong skill in developing Java code is essential.

Detailed hardware and software requirements for HDT can be found in Table 10 below.

Table 10: Requirements for the Human Digital Twin Component

Component Name	Human Digital Twin (SC09)
Type (Software/Hardware/Both)	Software
Short Description	The module creates a digital representation of human workers, seamlessly integrated within the digital twin service platform that provides the virtual environments for simulations
Employed at: Run Time / Design Time / Both	Both
Input requirements	
Input Data from Knowledge Base	Yes. 3D models of robots, Robots parameters, 3D model of working environment, 3D models of workstations, 3D models of worked components
Input Data from Sensors/Context	Our solution emulates the behaviours of sensors
Format of Expected Input	JSON/XML
Triggered by	End user/Event-driven from input components (Simulation testing based tool)
Interfaces	Data stream (Shared memory), Batch files
Output requirements	
Main Outputs	Human performance metrics (task duration, saturation rate), KPIs of the system (avg productivity, resources utilization rate), safety zones violation

Output Data to Knowledge Base	Simulation results in XML format
Nature of Expected Output	JSON files
Hardware & software requirements	
Software Requirements	Digital Twin Service platform, Windows or Linux operating system
Hardware Requirements	No specific
Communications	GRPC, TCP/IP
Special Communication Requirement	Direct low-latency connection between Digital Twin Service platform and SBT tool
Integration Requirements	No specific
Deployment Requirements	No specific

2.9.4. Suggested bundling of this component with other SOPRANO components

It is strongly suggested to bundle the HDT component with the SBT tool when the main scope is to investigate safety issues in MH-MR systems to prevent unexpected collisions and mitigate the risk of workers injuries.

2.10. AI Trustworthiness (AIT)

2.10.1. Technical summary

The AI Trustworthiness component underpins the safety assurance of AI-based components powering the robots, when deployed in interaction with humans. This involves assessing the quality attributes of DL models, offering a balance between correctness and reliability. The framework has a design time and a runtime AI assurance module.

The design time module carries out quality assurance of DL components used in the robotic team or within the MH-MR interaction tasks, driven by a systematic testing methodology, leveraging an AI explainability-driven test criterion for detecting behaviour inconsistencies in the DL models. In particular, given a trained DL model (e.g., for object classification) along with the training and testing sets, the design time module conducts white-box analysis and testing to assess the fault-revealing ability of the test set and its defect-detection capabilities. The results are exploited for the generation of semantically meaningful synthetic datasets with the twofold objective of increasing the test coverage and exposing the DL model to a diverse set of inputs that could yield unexpected decisions. Identifying these issues and fixing them before the system is deployed is paramount. At this stage, technologies developed in the AI Model Optimizer SOPRANO component will be used to efficiently process, transfer, and integrate the augmented data between the different components of this module.

The runtime module carries out the reliability evaluation by conducting uncertainty estimation enabling the identification of situations where the DL model may be particularly uncertain. The final pillar of this module is a continuous monitoring and repairing methodology that relies on the data augmentation process. This data augmentation process will entail creating synthetically meaningful inputs derived from the test/training data. Additionally, this data will be deployed for model finetuning and incremental retraining to address issues related to data diversity and domain adaptation by introducing corner-case scenarios replicated using the data augmentation process.

The planned release of AIT is partitioned into two key stages to reflect the progressive development of its constituent modules: Explainability-driven testing for DL models: This module introduces a white-box testing approach that leverages explainability concepts to identify important neurons contributing to decision-making and target their adequate testing. This module will be released by mid-2025.

Uncertainty estimation of DL models: This module assesses the confidence of predictions made by DL models. By quantifying uncertainty, it ensures that models provide reliable and interpretable outputs, especially in critical industrial applications. This module will be released by mid-2025.

Improved DL models testing through data augmentation: A data augmentation module is utilized to enhance the generalization and robustness of AI models. This module generates synthetic data using generative AI (GenAI) to increase prediction accuracy and expand the prediction space, helping models better handle diverse scenarios. This module will be released by Q4 2025.

2.10.2. Planned date of release of a prototype

The planned release of AIT is partitioned into two key stages to reflect the progressive development of its constituent modules:

- By mid-2025 - an AIT prototype comprising the explainability-driven testing module and the runtime uncertainty estimation module.
- By end-2025 - an enhanced AIT component augmented with the data augmentation module.

2.10.3. Requirements for the component

The AIT framework is implemented as a stand-alone package, based on widely used ML frameworks, i.e., TensorFlow, Pytorch, Keras. The machine used for deployment needs to have Python3, pip, Anaconda and Docker installed, as well as network connectivity.

More information about the software and hardware requirements can be found in Table 11 below.

Table 11: Requirements for the AI Trustworthiness Component

Component Name	AI Trustworthiness (SC11)
Type (Software/Hardware/Both)	Software
Short Description	AI trustworthiness component performs AI explainability-driven test criterion to detect behaviour inconsistencies in the ML/DL at design time and advanced testing techniques are exploited at runtime
Employed at: Run Time / Design Time / Both	Both
Input requirements	
Input Data from Knowledge Base	Training and testing datasets for each Deep Neural Network used

Input Data from Sensors/Context	Image data, LiDAR/GPS data
Format of Expected Input	data (e.g. RGB images, pointcloud) with ground truth annotations (e.g. CSV, JSON), model and hyperparameters (e.g. pd, hdf5)
Triggered by	Design time: every model, run time: every raw prediction
Interfaces	Shared message bus (broker application)
Output requirements	
Main Outputs	Synthetically augmented datasets, repaired WP2 models, uncertainty estimation
Output Data to Knowledge Base	DT: Synthetically augmented datasets, updated DNN; RT: uncertainty estimation
Nature of Expected Output	Datasets, re-trained models for perception (model architecture and weights), percentage values indicating the uncertainty for each prediction
Hardware & software requirements	
Software Requirements	Python3, TensorFlow, Pytorch, Keras, Pip, Docker, Anaconda
Hardware Requirements	GPU-accelerated machine with RAM>=16GB, screen, network (wired/wifi)
Communications	Broker application/Shared message bus
Special Communication Requirement	No specific
Integration Requirements	Python3, Pip, Docker, network connectivity
Deployment Requirements	Python3, Pip, Docker, network connectivity

2.10.4. Suggested bundling of this component with other SOPRANO components

The AIT component is designed to operate whilst using as an input a deep neural network model, though at the same time it works independently, offering testing APIs for DL models to ensure its confidence for human use. Both runtime and design time parts of AIT do not require other SOPRANO components' interaction and should be model-agnostic.

However, integrating AIT with other components could be helpful for enhancing the use cases' capabilities. For example, given the 6D (or 3D human) pose data, AIT could enhance object detection reliability by integrating uncertainty estimation and explainability-driven testing to ensure safety in both the OBP and HMO components.

2.11. Context Extraction Module (CEM)

2.11.1. Technical summary

The Context Awareness solution extracts context under which the work is carried out, interprets the current context and provides context information as a basis to adapt the robot actions/recommendations to the context. This solution consists of three main components: Context Models, Context Monitoring and Context Extraction. Within the SOPRANO project the Context models will be used as a baseline to gather and represent knowledge about collaborative human-robot work in various settings. The Context models will be a set of concepts and their relations which describe the stages, entities, attributes and stakeholders within collaborative human-robot work.

The Context Monitoring and Extraction components allow for identifying changes in the Contexts of the environment. The identified Context is used to support the decision-making / optimization / reconfiguration. The Context Monitoring and Extraction method uses monitored “raw data” provided from systems/sensors, as well as knowledge available in different (SOPRANO) modules to derive the current context. Using the Context model, the monitored data is evaluated, and the context extracted. Based on the identified context, it can be compared to previous ones and stored. A continuous process, coordinating with the monitoring and followed by the extraction process to give current context meaning to the provided knowledge, is built around the main extraction of a Context in SOPRANO. The method covers the definition of the mechanisms to monitor context and process changes – in relation to the quantitative relationships influencing the meaning of contexts from various perspectives (user, modules, services). This includes a definition of the entities relevant for monitoring contexts and parameter changes, and investigation of the extent to which these mechanisms depend on the viewpoint, for which changes are being monitored, according to the defined Context models.

The Context Extraction Module is based on the Eclipse OpenSmartClide Context Handler, developed within the H2020 SmartCLIDE project and consists of two independent components, the Context Monitoring and the Context Extraction. Both components are developed in Java and can be seen as backend-services. The Context Extraction Module will be provided in two versions, one for smaller installation, where both components will be deployed in one docker container and a version for large installation, where two docker containers will be provided (one for the Context Monitoring and a second one for Context Extraction).

2.11.2. Planned date of release of a prototype

The core functionality of the Context Extraction Module is already available via the Eclipse OpenSmartCLIDE project (https://github.com/eclipse-opensmartclide/smartclide-context).

By mid-2025 - the SOPRANO specific additions, such as the capability to communicate with the SOPRANO MessageBroker are planned to be published, and the SOPRANO Context Model is planned to be realized.

2.11.3. Requirements for the component

Detailed hardware and software requirements for CEM can be found in Table 12 below.

Table 12: Requirements for the Context Extraction Module Component

Component Name	Context Extraction Module (SC12)
Type (Software/Hardware/Both)	Software
Short Description	The component incorporates the ontology approach for model contexts while the context extraction service evaluates, manages & converts the raw sensory data to aggregated data
Employed at: Run Time / Design Time / Both	Both
Input requirements	

Input Data from Knowledge Base	Yes (previously stored context information to be able to reason current context)
Input Data from Sensors/Context	No
Format of Expected Input	.json
Triggered by	Triggered either by another component or periodically by Context Extraction itself
Interfaces	Interface to the Middleware (e.g. Kafka) and to the robots
Output requirements	
Main Outputs	Extracted Context
Output Data to Knowledge Base	Extracted context
Nature of Expected Output	Extracted context in RDF format
Hardware & software requirements	
Development Environment	Docker container
Software Requirements	Context Extraction Module will be provided as docker container image
Hardware Requirements	no special hardware requirement foreseen
Communications	Via Middleware using publish subscribe mechanisms
Special Communication Requirement	No specific
Integration Requirements	Docker, Middleware
Deployment Requirements	Docker, Middleware

2.11.4. Suggested bundling of this component with other SOPRANO components

If any of the components SC1, SC2, SC3, SC4 are used then they could be used as inputs to the CEM component, but they are not compulsory, other monitors that observe the situations of the robots can be developed to input the wanted situational data.

2.12. MH-MR Architecting Tools (MAT)

2.12.1. Technical summary

The MH-MR Architecting Tools model the environment in which the robot operates by using the FloorPlan and Variation Domain Specific Languages (DSLs). The goal of these DSLs is to provide a user-friendly, text-based, declarative language to specify environmental models. As its name suggests, the Variation DSL enables the generation of variations of a FloorPlan model by sampling measurements and variables that describe the FloorPlan geometry (e.g., the length of a hallway or the height of a door). These models are transformed into a composable model representation, which is used by the scenery builder tools for the generation of executable artefacts, both for simulation and real-world execution.

Optionally, the MAT tools can use an Industry Foundation Class (IFC) representation of a Building Information Modelling (BIM) model as an input, which is transformed into Composable FloorPlan models and conforms to the FloorPlan meta-models. These models are composed into a graph which can be queried using SPARQL or RFD libraries. The scenery builder then generates artefacts for simulation in Gazebo (3D meshes, Gazebo models and worlds), occupancy grids and ROS launch files. It can also use features in the environment (or those extracted and transformed from the BIM model) to generate task specifications according to some predefined schema.

2.12.2. Planned date of release of a prototype

Mid-2025

2.12.3. Requirements for the component

Detailed hardware and software requirements are presented in Table 13 below.

Table 13: Requirements for the MH-MR Architecting Tools Component

Component Name	MH-MR Architecting Tools (SC13)
Type (Software/Hardware/Both)	Software
Short Description	A meta-modelling approach is employed to compose dynamic environmental features (floorplan, occupancy grids, task specifications etc.)
Employed at: Run Time / Design Time / Both	Design Time
Input requirements	
Input Data from Knowledge Base	Yes
Input Data from Sensors/Context	No
Format of Expected Input	.ifc
Triggered by	-
Interfaces	Gazebo plugins
Output requirements	
Main Outputs	Floorplan DSL models, composable models, generated execution artifacts
Output Data to Knowledge Base	FloorPlan models, Occupancy grid maps, Gazebo world and models, STL meshes
Nature of Expected Output	.floorplan, .variation, .jsonld, .stl, .sdf, .world, .yaml, .pgm, .launch, .poly
Hardware & software requirements	
Development Environment	Python, ROS
Software Requirements	Blender, ROS, Gazebo
Hardware Requirements	No specific
Communications	No specific
Special Communication Requirement	No specific
Integration Requirements	Docker
Deployment Requirements	Docker, Gazebo, ROS2

2.12.4. Suggested bundling of this component with other SOPRANO components

The component can be integrated with the Scalable Simulation-Based Testing (SBT) component.

2.13. MLOps Orchestrator (MLO)

2.13.1. Technical summary

The MLOps Orchestrator provides automation and monitoring at all steps of ML system development and deployment, including integration, testing, releasing, deployment and infrastructure management and is needed in cases of frequent retraining and redeployment of machine learning models/features. MLO in collaboration with AI Model Optimizer provides the packaging and automatic deployment of the ML pipeline in the cases that the computational resources that are available cannot meet the expected requirements. The combined solution of the two components automates the model training process and identifies suitable strategies for meeting the pipeline requirements and adapt the pipeline accordingly.

The MLO will consist of five modules, the Airflow Engine, the Worker Management that handles the infrastructure resources, the Resource Registry that contains all the relevant resources, the Workflow Editor and the Artifact Repository which serves as an integration point with AIO and contains the ML models and docker containers. The workflow editor module defines in a graphical way the components involved and the sequence of execution of an AI pipeline. The Editor can manage the data sources (inputs/outputs), analytics processors and infrastructure resources (workers). The editor exports a building and deployment plan that is first consumed by AIO which may extend it and then is used by MLO Airflow module which provides the workflow orchestration through automation, monitoring and maintenance of the ML pipelines.

2.13.2. Planned date of release of a prototype

Mid-2025 - early prototype, depending on available SOPRANO components/algorithms that will utilize MLO.

2.13.3. Requirements for the component

MLO requires as initial inputs the available infrastructure resources, the digital resources meaning the communication interface for data types to be transferred and the available processors being the analytics algorithms through docker containers. With these inputs predefined in MLO, the user can then specify the AI components to be included in the pipeline, the input parameters and settings for each AI component and the data sources (ex. KAFKA or any message bus). Through a user-friendly pipeline editor, the user connects the AI components and defines their configuration parameters. Then, the MLO outputs a fully configured AI pipeline based on the user inputs, including the selection of AI components, parameter settings, and data sources. The component provides visual representations of the constructed pipeline and manages data flow within the pipeline, ensuring efficient transfer and processing of data between different AI components.

Table 14: Requirements for the MLOps Orchestrator Component

Component Name	MLOps Orchestrator (SC14)

Type (Software/Hardware/Both)	Software
Short Description	This infrastructure will facilitate the seamless coordination and orchestration of the SOPRANO AI components across a network of devices with varying capabilities, ensuring efficient resource utilization, low-latency data processing, and robust fault tolerance
Employed at: Run Time / Design Time / Both	Design Time
Input requirements	
Input Data from Knowledge Base	Image streams, sensor data
Input Data from Sensors/Context	Deployment Plan
Format of Expected Input	YAML/JSON/etc.
Triggered by	Developer (CI/CD operation)
Interfaces	Interfaces for integration with AI Model Optimizer
Output requirements	
Main Outputs	Pipeline configuration, data flow management, visualization of the pipeline, integration with external systems, orchestration service
Output Data to Knowledge Base	N/A
Nature of Expected Output	AI/ML Pipelines
Hardware & software requirements	
Development Environment	on-premise/cloud/edge
Software Requirements	Depends on Worker Purpose. Specific Software requirements can be configured when building worker docker images (requirements.txt).
Hardware Requirements	WME - Workflow Management Engine (front & back): 4 GB RAM, Airflow Core - (Web Server, Redis, Postgres, Scheduler): 4 GB RAM, For deployment only: 2GB RAM
Communications	REST API, SSH/SCP, Web Sockets (possibly in the future)
Special Communication Requirement	No specific
Integration Requirements	git, docker
Deployment Requirements	Using Ansible: Internet + ansible in jumphost + ssh key (jumphost -> core/worker). Without Ansible: Internet, docker, git, ssh keys

2.14. AI Model Optimizer (AIO)

2.14.1. Technical summary

The AI Model Optimizer is a software system that transforms deep neural networks (DNN) in a way to optimize their inference performance given the application requirements and the available compute and network infrastructure.

There are two distinct core functionalities within the system:

- It estimates the performance of a DNN model in inference mode under various settings (standalone, distributed) on a given compute and network resource infrastructure
- It performs transformations on a DNN model (without retraining it) and generates a deployment plan. Those transformations include the creation of an equivalent distributed configuration, the pruning of the network weights/filters/layers and the quantization of the network weights. The functionality includes the generation of one or more new docker images for the execution of the model.

The two functionalities are combined so as for the AIO to generate optimal (for a given DNN, requirements and resources) transformations and deployment plans. Those deployment plans are consumed by the MLOps Orchestrator while the user interaction with the AIO for the submission of the DNN model characteristics and the application requirements are provided by a minimal user interface. The availability of the resources is provided by an agent installed in each of the computing nodes. The AIO starts with the processing of the DNN structure that is provided as a .pt or .pht file (Pytorch). The output of the AIO is a deployment plan either for an Apache Airflow or for a Kubeflow installation or the StreamK3S subsystem.

2.14.2. Planned date of release of a prototype

Mid-2025 - the initial release of the AI Model Optimizer (AIO) is expected. Since a significant portion of its functionality depends on seamless integration with the MLOps Orchestrator and other SOPRANO components, the timeline for a more refined version is yet to be determined.

2.14.3. Requirements for the component

Detailed hardware and software requirements are presented in Table 15 below.

Table 15: Requirements for the AI Model Optimizer Component

Component Name	AI Model Optimizer (SC15)
Type (Software/Hardware/Both)	Software
Short Description	The main functionality of the optimizer is to convert a computationally intensive AI model/component into an optimized model in terms of latency and inference time, that will be able to operate with reduced resources if needed.
Employed at: Run Time / Design Time / Both	Design Time
Input requirements	
Input Data from Knowledge Base	Yes
Input Data from Sensors/Context	Yes (monitoring agents for tracking the availability of the computational resources)
Format of Expected Input	YAML/JSON/etc.
Triggered by	MLOps Orchestrator
Interfaces	1) to data bus, 2) interfaces for input data, 3) interfaces for integration with MLOps Orchestrator
Output requirements	
Main Outputs	Optimized AI models for resource-constrained devices

Output Data to Knowledge Base	Deployment plan, new software components [optionally]
Nature of Expected Output	yaml/json/etc, docker images [optionally]
Hardware & software requirements	
Software Requirements	Python, Docker engine
Hardware Requirements	Commodity computer/VM
Communications	TCP/IP
Special Communication Requirement	Lan-Connection to UR10
Integration Requirements	Partner-Actions implemented as Ros2-Action
Deployment Requirements	Partner-Actions implemented as Ros2-Action

2.14.4. Suggested bundling of this component with other SOPRANO components

The AIO can fit its output to the MLO so that the submodels generated by the MLO can be deployed in the available resources. It also bundles with the OBP, since for its first version, the AIO, optimizes the inference of the OBP DNN.

2.15. Robotic Capabilities Implementation (RCI)

2.15.1. Technical summary

The task mapping component within the SOPRANO system is designed to deliver essential functionalities that enable the decomposition of high-level assembly plans into executable robotic actions, specifically tailored for the robotic platforms in SOPRANO - FELICE Robot and UR10. This component (Robot Capabilities Implementation) ensures efficient execution of low-level robotic skills such as object handling, manipulation, screwing tasks, and collaborative tasks with human operators. Implementing non-continuous robotic actions like grasping, holding, and screwing requires precise prediction of parameters such as grasp points, object properties, and environmental conditions. The proposed behaviour tree-based framework leverages the modularity and hierarchical structure of behaviour trees to represent and execute robotic tasks efficiently.

For specific tasks like object handling and manipulation, the system identifies, and grasps objects using optimized grasp points and poses, adapting to varying object geometries and properties. Screwing tasks involve accurate alignment of screw threads, dynamic torque control based on material hardness, and consistent thread engagement using real-time feedback to prevent stripping or cross-threading. Collaborative tasks synchronize robotic actions with human interventions, ensuring timely and coordinated execution in collaborative assembly environments. High-level control could involve integrating low-level control functionalities with the ROS framework to enable smooth trajectory execution and self-collision awareness. Integration with open-source implementations such as MoveIt and ros_control allows for seamless coordination between planning and execution, with dynamic scene adaptation based on object localization and sensor data. The implementation is envisioned to include grasp planning and execution, involving the configuration and optimization of grasp points and poses based on CAD models, followed by extensive testing and parameter optimization. The system dynamically adapts task execution strategies based on real-time feedback and environmental conditions, leveraging perception modules for accurate object detection and recognition. This comprehensive approach ensures efficient, adaptable, and reliable execution of low-level robotic skills, enhancing productivity in industrial assembly processes.

2.15.2. Planned date of release of a prototype

- By mid-2025 - some of the capabilities (Object handling, Object manipulation)
- By end-2025 - most capabilities (collaborative handover, unscrewing).

2.15.3. Requirements for the component

Detailed hardware and software requirements are presented in Table 16 below.

Table 16: Requirements for the Robotic Capabilities Implementation Component

Component Name	Robotic Capabilities Implementation (SC16)
Type (Software/Hardware/Both)	Software
Short Description	A comprehensive framework capable of executing diverse robotic actions will be developed. The component will also support the integration of the low-level control functionalities with the ROS framework (for FELICE Robot and UR10)
Employed at: Run Time / Design Time / Both	Runtime
Input requirements	
Input Data from Knowledge Base	Yes
Input Data from Sensors/Context	Yes
Format of Expected Input	Ros message (ROS2 actions),
Triggered by	ID5
Interfaces	ROS2
Output requirements	
Main Outputs	High-level robot control, Manipulation planning, Grasp planning and execution
Output Data to Knowledge Base	No
Nature of Expected Output	Ros2 actions (Robot Trajectories, process triggers)
Hardware & software requirements	
Software Requirements	Python, Docker
Hardware Requirements	depends on the use case
Communications	Ros2, Rest
Special Communication Requirement	Direct Ethernet-Connection (e.g. via switches) to the Robot
Integration Requirements	Partner-Actions implemented as Ros2-Action
Deployment Requirements	Partner-Actions implemented as Ros2-Action

2.15.4. Suggested bundling of this component with other SOPRANO components

The component can be integrated with the MRC, OBP and CTA components.

2.16. Advanced Visualizations (AVT)

2.16.1. Technical summary

The Advanced Visualization Toolkit is the component that bridges the gap and offers to the SOPRANO end-user all the necessary features, services and indications under a unified user interface. Its real-time and historical data analytics capabilities will be utilized to provide dedicated informative and user-friendly custom dashboards, assisted by AI modules. During the Design Time execution of the SOPRANO system, AVT will provide operational historical data, stored in the Knowledge Base, to the end-user utilizing visualization widgets that best fit each scenario accompanied by relevant notifications and indicators. Additionally, the operator will be able to log-in to the AVT using an Identity and Access Management (IAM) solution. The user management process will be handled by Keycloak which will be integrated with the AVT to not only serve the application through a secure channel but also allow the creation of different custom dashboards relevant to the operator's rights.

During Runtime, upon initialization, all the spatial information (floorplans, BIMs etc.) will be retrieved and displayed from the Knowledge Base as an after product of the BIM-files procedure. The available options will be retrieved by MH-MR Task Allocation and presented to the operator, and the selection will be reported back to allow further configuration. During the execution of the scenarios, interactions with the Safety tools and Human Digital Twin will be established to display possible safety warnings and performance metrics with particular attention to the representation of time-sensitive information and indicators. Finally, in specific scenarios, the operator will be able to interact with the options offered by the Robotic Capabilities Implementation required to proceed with the execution. Any additional functionalities that might arise during the initial integration of the components will be integrated in the AVT, in an effort to provide a single and central point of entry and control to the operator.

2.16.2. Planned date of release of a prototype

Mid-2025 - initial release of the Advanced Visualizations is expected. Since a significant portion of its functionality depends on seamless communication with other SOPRANO components, the timeline for a more refined version is yet to be determined.

2.16.3. Requirements for the component

In terms of data storage capabilities, AVT can operate with its own storage facility (mainly Elasticsearch search engine) and/or also use multiple other sources of data, be it data streams (e.g. Kafka) or batch data (e.g. via RESTful APIs, external ES, MongoDB, etc.). This flexibility is possible due to a number of tailor-made converters, which can request the data from the respective data sources and transform it to a suitable format for AVT's visualization widgets. APIs/other methods (e.g., direct access to data) are required to get the indicator values and the task related information. Streaming data is also supported (e.g. via Kafka). An Angular-based web application offers the frontend of the Advanced Visualization Toolkit. A Node.js application serves as the middleware that handles data inputs and the business logic of the toolkit. AVT can be installed as a standalone application, but a dockerized version of the toolkit is also available. End-users only need a browser to access AVT.

Table 17: Requirements for the Advanced Visualizations Component

Component Name	Advanced Visualizations (SC17)

Type (Software/Hardware/Both)	Software
Short Description	A set of data collection, processing, and presentation components and tools, which can assist users in examining and analysing digital information collected from the monitored sources. Interactive commands and authentication processes will be also developed
Employed at: Run Time / Design Time / Both	Both
Input requirements	
Input Data from Knowledge Base	Yes
Input Data from Sensors/Context	No
Format of Expected Input	JSON/XML
Triggered by	End-user / Auto (from other components)
Interfaces	1) Data streams (e.g. Kafka) 2) Batch data (e.g. via RESTful APIs, external ElasticSearch, MongoDB, flat files etc.)
Output requirements	
Main Outputs	Metrics / Safety Warning / Task Progress, Configuration Options
Output Data to Knowledge Base	Yes
Nature of Expected Output	Visualizations (graph based etc.), Json files
Hardware & software requirements	
Development Environment	cloud and on-premise
Software Requirements	Express.js, Elasticsearch, Kafka, MongoDB, Node.js, etc depending on the implementation Also available as a standalone Docker image
Hardware Requirements	• CPUs >= 2 • RAM >= 4GB (depending on the data volume and utilized visualizations) • Disk space >= 20GB
Communications	1) rest-API endpoints and 2) web sockets
Special Communication Requirement	No specific
Integration Requirements	Docker

2.16.4. Suggested bundling of this component with other SOPRANO components

Bundling Advanced Visualizations with other SOPRANO components is not justified in the current analysis.

3. How to Start Developing with a SOPRANO Component?

The SOPRANO consortium has already started the integration of the components into three industrial use cases. The third-party partners should follow the development process already implemented in these use cases. The following approach is warranted:

- Describe the use case according to SOPRANO specifications and concepts.

- Choose the SOPRANO components that you would like to use and integrate into the industrial setting of your use case.
- Start the development with the help of SOPRANO mentors and the application templates and samples that will be provided. To help the 3rd-party partners, the consortium will provide, as needed, additional technical information, documentation and samples.
- In addition, the three SOPRANO use-case partners will present what can be achieved with the SOPRANO technologies.