Ref. Ares(2026)5995217 - 12/06/2026





Project Number 101120990

D3.3 Technologies for Safety Analysis and Testing of MH-MR Systems II

Version 1.0
30 March 2026
Final

Public Distribution

TTS, University of York, CEA, KUKA

Project Partners: **AEGIS, ATB, CEA, Centro Ricerche Fiat, Circular Economy Foundation, E-TERRY, F6S, FORTH, Harokopio University of Athens, IFADO, Jade University, KUKA Assembly & Test, Netcompany-Intrasoft, NTUA, PROFACTOR, Technology Transfer Systems, The Open Group, University of Bremen, University of York**

Every effort has been made to ensure that all statements and information contained herein are accurate, however the SOPRANO Project Partners accept no liability for any error or omission in the same.

© 2026 Copyright in this document remains vested in the SOPRANO Project Partners.

PROJECT PARTNER CONTACT INFORMATION

AEGIS Nikos Kostalas Humboldtstrasse 25 38106 Braunschweig, Germany E-mail: n.kostalas@aegisresearch.eu	**ATB** Sebastian Scholze Wiener Strasse 1 28359 Bremen, Germany E-mail: scholze@atb-bremen.de
CEA Nikolaos Matragkas NanoInnov Bat. 862 PC 174 91191 Gif-sur-Yvette Cedex, France E-mail: nikolaos.matragkas@cea.fr	**Centro Ricerche Fiat** Felice Tauro Strada Torino 50 10043 ORBASSANO, Italy E-mail: felice.tauro@crf.it
Circular Economy Foundation Ioannis Basdekis Rue Breydel 34-36-40 1040 Brussels, Belgium E-mail: ioannis.basdekis@circular.foundation	**E-TERRY** Fabian Rösler Neuwerkstrasse 50 99084 Erfurt, Germany E-mail: fabian.roesler@e-terry.de
F6S Gergana Dimitrova 77 Lower Camden Street Dublin D02 XE80, Ireland E-mail: gergana@f6s.com	**FORTH** Manolis Lourakis N Plastira Str 100 70013 Heraklion, Greece E-mail: lourakis@ics.forth.gr
Harokopio University of Athens Iraklis Varlamis Eleftheriou Venizelou 70 176 71 Athens, Greece E-mail: varlamis@hua.gr	**IFADO** Felix Dreger Ardeystrasse 67 44139 Dortmund, Germany E-mail: dreger@ifado.de
Jade University George Athanassiou Friedrich Paffrath Strasse 101 26389 Wilhelmshaven, Germany E-mail: georgios.athanassiou@jade-hs.de	**KUKA Assembly & Test** Georgios Karantinakis Uhthoffstrasse 1 28757 Bremen, Germany E-mail: georgios.karantinakis@kuka.com
Netcompany-Intrasoft John Soldatos 2B Rue Nicolas Bové 1253 Luxembourg, Luxembourg E-mail: john.soldatos@netcompany-intrasoft.com	**NTUA** Maria Pateraki Heroon Polytechniou 9, Zographou Campus 157 80 Athens, Greece E-mail: mpateraki@mail.ntua.gr
PROFACTOR Sharath Akkaladevi Im Stadtgut D1 4407 Steyr-Gleink, Austria E-mail: sharath.akkaladevi@profactor.at	**Technology Transfer Systems** Antonio Avai Via Francesco d'Ovidio, 3 20131 Milan, Italy E-mail: avai@ttsnetwork.com
The Open Group Scott Hansen Rond Point Schuman 6, 7th Floor 1040 Brussels, Belgium E-mail: s.hansen@opengroup.org	**University of Bremen** Nico Hochgeschwender Bibliothekstrasse 1 28359 Bremen, Germany E-mail: ni_ho@uni-bremen.de
University of York Simos Gerasimou Deramore Lane York YO10 5GH, United Kingdom E-mail: simos.gerasimou@york.ac.uk	

DOCUMENT CONTROL

Version	Status	Date
0.1	Initial draft with outline and first content	11 January 2026
0.2	First input from partners	27 February 2026
0.5	Refined version	01 March 2026
0.8	Final draft for partner review	15 March 2026
0.9	Document finalization	28 March 2026
1.0	Final version for EC submission	30 March 2026

TABLE OF CONTENTS

1. Introduction 1
1.1. Overview 1
1.2. Structure of the document 2
2. The safety analysis and testing solution 3
2.1. The conceptual architecture 3
2.2. SOPRANO Safety Approach within Robotics Safety Engineering 7
3. SOPRANO SAT Solution 9
3.1. Design Time and Runtime Safety Analysis tool 9
3.1.1. Design-Time safety analysis (DSA) 10
3.1.2. Extension workflow from DSA to runtime monitoring 13
3.1.3. Runtime Safety monitoring layer 14
3.1.4. Integration within the SOPRANO safety and testing ecosystem 21
3.2. The scalable and extensible simulation based testing tool 22
3.2.1. Simulation-Based Testing Overview 22
3.2.2. SBT Configuration Methodology 22
3.2.3. SBT High-Level Architecture 24
3.2.4. Offline SBT Predictors 26
3.2.5. Offline Prediction Approach 30
3.3. Human Digital Twin for MH-MR system (HDT) 37
3.3.1. Introduction 37
3.3.2. Data management and communication mechanism 39
3.3.3. Integration between HDT and SBT 46
3.3.4. Integration between HDT and MAT 48
3.3.5. Integration with the mobile platform 49
3.4. Application on symbiotic farm use case 53
3.4.1. Overview of symbiotic farm use case 53
3.4.2. Safety Analysis Tools Demonstration 54
3.4.3. Simulation-Based Testing Tools 66
4. Conclusion 74
A) Progress Since D3.1 76
A1) Design Time and Runtime Safety Analysis tool 76
A2) The scalable and extensible simulation based testing tool 77
A3) Human Digital Twin for MH-MR system (HDT) 78

TABLE OF FIGURES

Figure 1: Conceptual architecture of SAT solution 4
Figure 2: Sophia Tool methodology 11
Figure 3: SOPHIA ODD editor 12
Figure 4: SOPHIA ODD textual DSL 12
Figure 5: SOPRANO HAZOP table 13
Figure 6: From DSA to RSA workflow 13
Figure 7: Continuous probabilistic risk modelling used in RSA 17
Figure 8: RSA pipeline 19
Figure 9: Design-time and runtime safety components 22
Figure 10: SOPRANO SBT methodology 23
Figure 11: The high-level architecture of the SOPRANO SBT components 25
Figure 12: Overall methodology of offline SBT prediction 27
Figure 13: UML representation of the SBT Testing Model DSL 29
Figure 14: Transformation of a test case definition into time series representation 31
Figure 15: UML package for configuration of offline predictors 34

Figure 16: Experimental loop communications to and from the predictor daemon 36
Figure 17: Example of the use of the Shared Memory and the datasources in SOPRANO, here with a OPC-UA datasource that publishes one value via OPC-UA 41
Figure 18: The KUKA platform Digital Twin and the HDT in the simulation environment. The Kuka platform is in the safety pose with the robotic arm going up, to not hurting anything while crawling around, and the worker is the rest pose, no animations are playing. 45
Figure 19: UML sequence diagram of the interaction between the SBT and some components of the DTS platform 47
Figure 20: Communication between HDT and SBT 48
Figure 21: A BIM file as it appears in Blender 49
Figure 22: Details of the same structure in which is possible to see in orange planned sockets and ducts 49
Figure 23: OPC-UA nodes on the KUKA OPC-UA server for communication with the DTS platform 50
Figure 24: Data exchange within the DTS platform and the KUKA components 52
Figure 25: An example of the E-TERRY use case for evaluation when transporting produce boxes from the field to a central location 54
Figure 26: E-Terry mission workflow 55
Figure 27: The E-Terry OpenODD model in Papyrus4ODD. 56
Figure 28: Sophia DSA plugin 57
Figure 29: Extract of hazard analysis results for the E-Terry use case 59
Figure 30: Example of a generated scenario 63
Figure 31: Degraded safety state without stop request 64
Figure 32: Emergency scenario leading to emergency state and stop request 65
Figure 33: Path completion metric 68
Figure 34: Static distance human metric 68
Figure 35: Dynamic distance human metric 68
Figure 36: Comparison of different test selection selectors with different hyper-parameters 69
Figure 37: Recorded number of RSA degradation events at various Gazebo update rate configurations in the E-TERRY simulation 71
Figure 38: The robot stops to avoid the human resulting in transient discrepancies between the navigation and safety velocities 74

EXECUTIVE SUMMARY

This deliverable presents the **Safety Analysis and Testing (SAT)** solution developed in the SOPRANO project to support the safe design, validation, and operation of **multi-human, multi-robot (MH-MR)** systems. As collaborative robotic applications become more dynamic and less structured, ensuring safety in shared workspaces requires methods that can address both design-time uncertainties and runtime risks. The SAT solution responds to this need by combining complementary technologies for simulation, testing, and safety analysis within a unified framework.

The proposed solution integrates three main technological pillars. The first is **digital twin and simulation technology**, which enables the creation of virtual representations of robots, workers, sensors, and workspace elements in order to analyse system behaviour under realistic and hypothetical operating conditions. The second is a **scalable and extensible simulation-based testing platform**, which supports automated testing campaigns and uses exploratory and evolutionary techniques to identify faults, edge cases, and unsafe interactions in MH-MR systems. The third is a **design-time and runtime safety analysis capability**, which supports hazard identification, risk assessment, and continuous monitoring of safety-relevant conditions during operation.

Together, these technologies provide an integrated approach to improving the robustness and dependability of collaborative robotic systems. The digital twin environment enables safe experimentation and virtual commissioning before deployment. The simulation-based testing platform expands coverage by automatically generating and executing test scenarios, including rare and critical cases that are difficult to identify manually. The safety analysis tool complements these capabilities by assessing risks systematically and by monitoring system and sensor data to detect unsafe conditions and trigger appropriate responses. The overall result is a framework that helps reduce the likelihood of accidents, supports safer human-robot collaboration, and strengthens confidence in system performance across the lifecycle.

The document describes the conceptual architecture of the SAT solution and the main components that implement it within SOPRANO. It covers the digital twin service platform and the associated human, robot, and sensor digital twin technologies; the simulation-based testing platform and its architecture, configuration methodology, manipulation mechanisms, and search-based testing capabilities; and the safety analysis tool for both design-time and runtime use. Through this combination, the deliverable demonstrates how SOPRANO advances the state of the art in safety assurance for MH-MR systems operating in shared, flexible, and human-centred industrial environments.

1. INTRODUCTION

1.1. OVERVIEW

The increasing deployment of **multi-human, multi-robot (MH-MR)** systems in industrial environments creates significant opportunities for productivity, flexibility, and resilience, but it also introduces complex safety challenges. In shared and often dynamic workspaces, robots, workers, sensors, and control systems must interact safely under changing operational conditions. Ensuring safe behaviour in such environments requires more than static safeguards: it calls for an integrated approach that supports hazard identification, design-time validation, and runtime supervision throughout the system lifecycle.

This deliverable presents the SOPRANO technologies for **Safety Analysis and Testing (SAT)** of MH-MR systems. The objective of the SAT solution is to support the proactive identification, assessment, and mitigation of safety risks both before deployment and during operation. To achieve this, the proposed approach combines three complementary technological pillars.

First, **digital twin and simulation technologies** enable the creation of virtual representations of MH-MR systems, including robots, workers, sensors, and workspace elements. These virtual environments support the exploration of realistic and hypothetical scenarios, allowing engineers to analyse behaviours, validate assumptions, and detect safety issues without exposing workers or equipment to risk.

Second, **simulation-based testing** supported by **exploratory and evolutionary algorithms** enables the systematic generation and execution of testing campaigns. This makes it possible to explore large design spaces efficiently, uncover edge cases and rare events, manipulate system inputs or communications, and automatically generate tests that challenge the robustness of robot controllers and safety mechanisms.

Third, **design-time and runtime safety analysis** provides a structured framework for identifying hazards, assessing risks, defining safety constraints, and monitoring compliance during operation. At design time, safety analysis supports the modelling of operational conditions, hazard identification, and derivation of mitigation strategies. At runtime, monitoring mechanisms analyse system states and sensor data in real time in order to detect emerging unsafe situations and trigger appropriate warnings or corrective actions.

The value of the SAT solution lies in the integration of these capabilities into a coherent framework. Digital twins and simulation support safe experimentation and virtual commissioning. Simulation-based testing strengthens robustness by systematically stressing the system under diverse scenarios. Safety analysis ensures traceability from risk identification to runtime monitoring and operational response. Together, these technologies contribute to improving the dependability, adaptability, and overall safety of collaborative robotic systems. This document describes the conceptual architecture of the SOPRANO SAT solution and the main technologies developed within the project. It explains how digital twin services, simulation-based testing, runtime and design-time safety analysis, and shared data infrastructures interact to support both validation and operational safety supervision in MH-MR environments.

The results presented in the following sections are related to the following tasks (from the SOPRANO
DoW):

- Task 3.1 Design-Time Model-Based Safety and Robustness
- Task 3.2 Scalable Intelligent Simulation-based Testing
- Task 3.3: Human-Digital Twin
- Task 3.6: Runtime System Dependability

1.2. STRUCTURE OF THE DOCUMENT

The remainder of this deliverable is organised as follows. Chapter 2 presents the overall safety analysis and testing solution, starting with its conceptual architecture and the main components involved. It then describes in detail the design-time and runtime safety analysis tool, the scalable and extensible simulation-based testing platform, and the digital twin technologies developed for MH-MR systems, including the digital twin service platform and the human, robot, and sensor digital twin prototypes. The final chapter summarises the main outcomes of the work and highlights the contribution of the SAT solution to safer and more robust collaborative robotic systems within the SOPRANO project.

2. THE SAFETY ANALYSIS AND TESTING SOLUTION

This chapter presents the SAT solution developed in SOPRANO for multi-human, multi-robot (MH-MR) systems. Its purpose is to provide an integrated framework for analysing, validating, and monitoring safety in collaborative environments where humans, robots, sensors, and control systems interact under dynamic operating conditions. The SAT solution is designed to support both design-time activities, such as modelling, simulation, and hazard identification, and runtime activities, such as system monitoring and detection of unsafe situations.

The chapter first introduces the conceptual architecture of the SAT solution, highlighting its main components, their interactions, and the principal input and output data exchanged across the framework. The architecture is centred on four main elements: the Digital Twin Service platform, which provides virtual representations of the MH-MR system and simulation services; the Simulation-Based Testing platform, which manages and executes automated testing campaigns; the Safety Analysis tool, which supports design-time risk assessment and runtime safety monitoring; and the Shared Memory component, which enables data exchange among tools and with robot controller systems. Together, these components form a coherent technical solution for safety assurance across the system lifecycle.

A key input to the SAT solution is the scenario definition, which captures the information required to model and explore an MH-MR system in simulation, including workspace characteristics, contextual elements, workers, robots, sensors, and workplans. On this basis, the framework supports both stand-alone simulation and hybrid configurations connected to real robot controllers. The resulting outputs include simulation campaign reports, experimental test results, and runtime safety information that can be used to assess workplan feasibility, collision risks, and overall system robustness.

The following sections describe each component of the SAT solution in greater detail. They present the objectives, architecture, and main technical features of the design-time and runtime safety analysis tool, the scalable and extensible simulation-based testing platform, and the digital twin technologies developed in SOPRANO, including the service platform and the human, robot, and sensor digital twin prototypes.

2.1. THE CONCEPTUAL ARCHITECTURE

The conceptual architecture of the SAT solution, which is illustrated in **Figure 1**, defines the main components of the framework, the relationships among them, and the principal data exchanged to support safety analysis and validation in MH-MR systems.

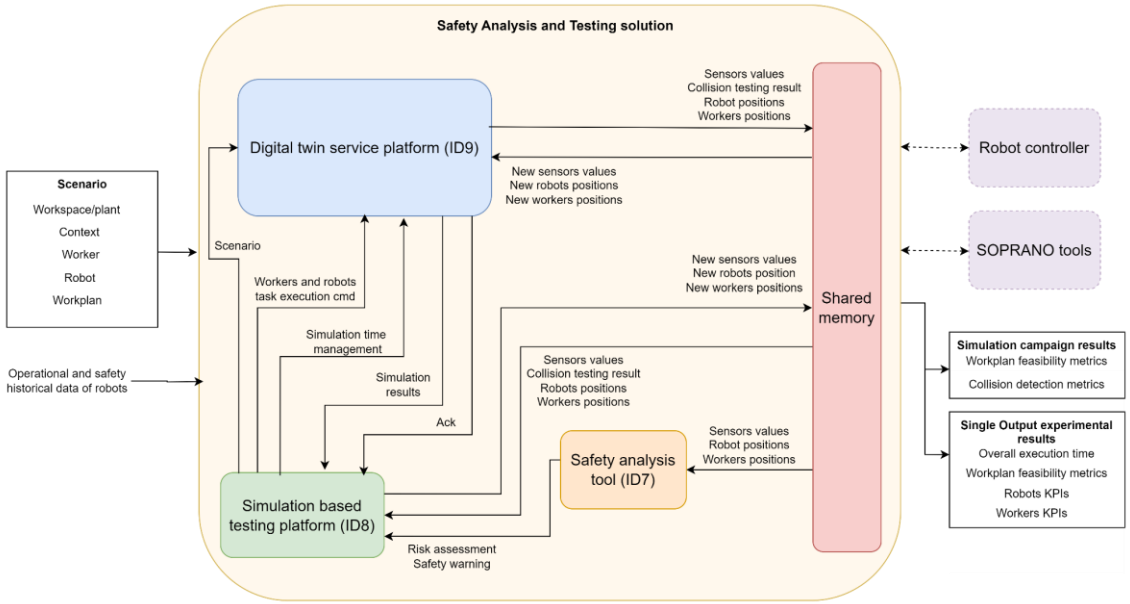


Figure 1: Conceptual architecture of SAT solution

The architecture has been designed to support both design-time and runtime activities, enabling the analysis of safety-related behaviour before deployment and the monitoring of operational conditions during system execution. As shown in the corresponding architecture figure, the SAT solution integrates four main components: the Digital Twin Service platform, the Simulation-Based Testing platform, the Safety Analysis tool, and the Shared Memory component. Together, these elements provide a coherent framework for modelling, testing, monitoring, and assessing collaborative robotic systems.

The **Digital Twin Service platform** provides the virtual representation of the physical MH-MR system. It supports the creation and management of digital twin instances for robots, workers, sensors, and workspace elements, enabling simulation of the system behaviour under analysis. In addition to simulation services, the platform supports monitoring by exchanging data with the other SAT components through the Shared Memory. This allows the digital twin environment to be used not only for virtual experimentation and validation, but also as part of a broader safety assurance workflow in which simulated and operational data can be analysed consistently.

The **Simulation-Based Testing platform** manages testing campaigns based on simulation scenarios and interacts with the Digital Twin Service platform to execute supervised experiments. Its purpose is to assess the robustness of robot controllers and MH-MR coordination strategies under a wide range of operating conditions, including rare events and critical edge cases. By automating the generation and execution of tests, the platform enables systematic exploration of the design space and provides evidence on workplan feasibility, collision risks, and the effect of perturbations or manipulated inputs on system behaviour.

The **Safety Analysis tool** supports both design-time and runtime assessment of safety risks in shared and fenceless work environments. At design time, it analyses the Operational Design Domain (ODD) of the system in order to identify hazards, assess risks, and evaluate safety-related properties of the system under study. At runtime, it processes data originating from the real robot controller systems in order to monitor the

current state of the MH-MR system and detect potentially unsafe situations. When required, the tool can generate warnings or alerts that are communicated to the testing platform and to the robot controller infrastructure, thereby supporting timely mitigation actions.

The **Shared Memory** component acts as the data exchange backbone of the SAT solution. It manages interaction among the different tools, provides access to sensor data and the positions or states of workers and robots, and enables communication with robot controller systems. Its role is not limited to data storage: it also supports bidirectional exchange with robot controllers, publication of gathered data to the other SAT components, controlled alteration of selected memory sections or registers for testing purposes, interaction with other SOPRANO tools, and data logging for subsequent analysis. In practice, Shared Memory implements a topic- or register-based data-sharing mechanism that allows information produced by one component to be consumed by multiple others in a consistent and technology-agnostic way.

The primary input to the SAT solution is the **scenario**, understood as the structured collection of data required to model and explore an MH-MR system through simulation. This scenario includes information on the workspace or plant, contextual elements such as walls, columns, and obstacles, worker models, robot models, sensor specifications, and workplans describing the sequence of tasks to be executed by the different actors in the system. In addition to scenario data, the framework may also use historical information on robot operational and safety behaviour, which can support the assessment of worker-related risks and the interpretation of system performance over time.

Based on these inputs, the SAT solution produces two main categories of outputs. The first consists of **simulation campaign results**, namely reports generated from testing campaigns that assess workplan feasibility, detect collisions, and evaluate the robustness of the MH-MR system under different configurations and disturbances. The second consists of **single experimental results**, which provide the outcome of individual simulation tests, including key performance indicators for robots and workers, as well as metrics related to feasibility and safety. These outputs support engineers in understanding system behaviour, identifying hazards, and refining both system design and safety measures.

An important feature of the architecture is its support for both **stand-alone simulation** and **hybrid configurations**. In the stand-alone mode, the MH-MR system is represented entirely through simulation. In the hybrid mode, the framework connects to real robot controller systems through the Shared Memory, enabling bidirectional data exchange during virtual commissioning and runtime monitoring. In this configuration, the simulated environment reproduces robot movements and sensor behaviour, while the actual control logic remains implemented in the physical robot controller. This makes it possible to combine the realism of real control software with the flexibility and safety of simulated experimentation.

The **SAT solution** can connect to real robot controller systems through **Shared Memory**, enabling bidirectional data exchange during virtual commissioning and continuous robot status monitoring at runtime. It also supports data exchange with other SOPRANO tools through Shared Memory, although this may require the development of dedicated interfaces for data integration. By integrating these technologies, the SAT solution

advances safety analysis by enabling the robustness of MH-MR systems to be tested against a wide range of design-time scenarios. Testing campaigns can be carried out either with a full stand-alone simulation model of the MH-MR system or with a hybrid model managed by an external robot controller. In the hybrid configuration, the model simulates robot movements and sensor behaviour, while the control logic implemented in the physical robot controller is used to supervise operations and make decisions.

The solution also enables runtime safety monitoring. The current status of the MH-MR system is continuously stored in Shared Memory and analysed by the Safety Analysis tool through runtime monitoring mechanisms. These mechanisms assess safety conditions and can trigger warnings or safety signals when hazardous situations are detected.

Examples of safety testing that can be performed with this solution include:

- **Functional safety testing**: This type of testing verifies that the system's safety mechanisms operate correctly under both normal and abnormal conditions. It ensures that robots and automation systems can prevent hazardous situations through appropriate design and built-in safeguards. For example, it can assess whether robots stop immediately when a human enters a danger zone, or how the system behaves when sensors fail or when robot-assessed localisation is inaccurate.
- **Proximity and collision avoidance testing**: This type of testing verifies that robots can operate safely around humans without causing collisions or entering unsafe proximity zones. It focuses on evaluating how effectively robots use sensors, such as position sensors, LiDAR, and radar, to detect humans and avoid dangerous interactions. For example, it can assess whether an autonomous robot detects workers along its path and slows down or stops to avoid collisions.
- **Environmental safety testing**: This type of testing ensures that robots and humans can operate safely under varying environmental conditions. It evaluates how factors such as obstacles affect the safety of human-robot interaction. For example, it can assess the ability of autonomous robots to navigate safely in indoor environments while avoiding workers and obstacles, even when workers suddenly move, block paths, or create new obstacles without warning.
- **Scalability and stress testing**: This type of testing ensures that the system's safety mechanisms remain effective as the number of humans and robots in the environment increases. It assesses how well the system performs under heavier loads or in more complex scenarios, such as environments involving many simultaneous human-robot interactions.

The key stakeholders of this solution are:

- **System designers and engineers**: ensuring that the MH-MR system architecture includes all necessary safety devices and mechanisms to prevent collisions.
- **Robotics manufacturers**: ensuring that robots operate reliably and safely in dynamic environments.
- **System integrators**: validating that MH-MR systems operate safely once integrated.

- **Robotic software developers**: developing robust software that mitigates risks arising from unpredictable or dynamic robot behaviour, thereby ensuring safe and smooth human-robot interaction.
- **Safety engineers**: defining safety requirements, assessing hazards and risks, and ensuring that the MH-MR system complies with applicable safety principles, methods, and safeguards.

2.2. SOPRANO SAFETY APPROACH WITHIN ROBOTICS SAFETY ENGINEERING

The **SAT** solution proposed in SOPRANO is not intended to replace traditional machinery safety measures. On the contrary, it is designed to complement them in collaborative MH-MR systems where robots share space with humans, cooperate with them, and rely on sensing, software, and sometimes AI-based functions. In such systems, traditional protective devices such as light curtains or safety scanners remain essential. They continue to provide the primary and most direct means of reducing risk at runtime. A safety scanner, for example, can detect a person entering a protected area and trigger a stop before contact occurs. A safety bumper can detect contact and limit the consequences of an unexpected collision. These functions are fundamental, and they remain part of the safety baseline in collaborative robotics. Standards such as ISO 12100 [1], ISO 13850 [2], ISO 13855 [3], ISO 13857 [4], ISO 10218-1 [5], ISO 10218-2 [6], and ISO/TS 15066 [7] remain fully relevant in this context.

However, in the collaborative applications considered in SOPRANO, the safety question is not limited to whether a final protective device can stop the system when a dangerous situation is already imminent. It also concerns whether the robot behaves **correctly**, **predictably**, and **robustly** that such situations are avoided as far as reasonably possible during normal operation. This is where the additional contribution of SOPRANO becomes important.

From the perspective of **ISO 12100**, the starting point remains the classical process of hazard identification, risk estimation, risk evaluation, and risk reduction throughout the machine lifecycle. The standard also establishes the hierarchy of risk reduction: inherently safe design measures first, technical protective measures second, and information for use third. The SOPRANO approach fits naturally within this logic. It supports better hazard identification, more systematic validation of design choices, and improved supervision of operational conditions. It should therefore be seen as an extension of established safety engineering practice, not as an alternative to it.

Traditional safeguards usually act as **protective barriers** at **runtime**, whereas SOPRANO mainly improves the **engineering** and **assurance** process around those barriers. In other words, a safety scanner or a safety-rated monitored stop directly reduces risk when a hazardous condition occurs. By contrast, the SOPRANO SAT solution helps engineers understand when and why the system may approach such hazardous conditions, whether the chosen safeguards are sufficient for the actual operating context, and whether the robot behaviour remains acceptable under realistic and degraded conditions. The value of SOPRANO is therefore not to duplicate the function of a safety scanner, but to reduce the likelihood that the system repeatedly reaches the point at which the scanner must intervene.

This distinction is especially important for systems that rely on AI-based perception or decision support. In a conventional safeguarded robot cell, many hazards can be controlled

through geometric separation, interlocking logic, and deterministic safety functions. In a collaborative application, however, the robot may need to detect a human, estimate position or intent, interpret the environment, select an appropriate motion, and react correctly to changing task conditions. In a tool handover scenario, for example, the robot must not only stop if a person enters a protected zone. It must also approach correctly, present the tool in a suitable pose, recognise whether the worker has grasped it, and react appropriately if the worker moves unexpectedly or another person comes close.

In such a case, a traditional safeguard may still prevent an accident, but that does not by itself demonstrate that the collaborative behaviour is well designed. If the robot regularly approaches the worker in an awkward or unsafe manner and the safety scanner must frequently trigger a stop, the immediate protective function may still work, but the overall application is clearly not satisfactory. It may become inefficient, difficult to use, or prone to repeated near-miss situations. Similarly, if a bumper prevents harmful consequences after contact, this is valuable as a last line of defence, but it is generally preferable that the collaborative system avoids reaching contact conditions during normal operation. SOPRANO addresses these upstream questions by helping engineers analyse, test, and monitor the behaviour before the final protective layer is activated.

This is why the SOPRANO approach is particularly relevant for MH-MR systems in which humans and robots intentionally share a workspace. In such systems, strict permanent separation is often incompatible with the intended function of the application. The relevant question is therefore not whether traditional safety measures remain necessary; they do. The question is how to design and justify the overall safety of an application in which the robot is expected to work near humans without constantly relying on emergency reactions. This is precisely the problem addressed by standards such as **ISO 10218** and **ISO/TS 15066**, and it is also the context in which SOPRANO contributes.

The complementarity can be described as follows. Traditional safety devices provide direct risk reduction at runtime. SOPRANO provides additional safety assurance by strengthening the design and validation process and by supporting runtime supervision of residual and emerging risks. The design-time safety analysis helps identify hazardous scenarios, including those related to perception uncertainty, task sequencing, environmental variability, and complex human-robot interactions. The digital twin environment makes it possible to explore operating conditions and abnormal situations that would be difficult, costly, or unsafe to assess only on the physical system. The simulation-based testing platform enables systematic exploration of edge cases and rare combinations of events that may not be discovered during conventional commissioning. The runtime safety analysis adds an additional supervision layer capable of detecting degradations, inconsistencies, or repeated approaches to safety limits during operation.

This contribution should not be interpreted as meaning that runtime monitoring automatically becomes a safety function in the strict standards sense. In many cases, a sound and conservative architecture will keep the primary risk reduction path in conventional, well-understood safety functions and devices, while using advanced monitoring and analysis as a complementary layer. For example, a safety scanner or monitored stop may remain the primary means of ensuring separation, while SOPRANO runtime monitoring detects degraded localisation confidence, abnormal interaction patterns, or repeated near-miss situations and requests degraded mode, reduced speed, or

operator intervention. This layered architecture is justified because it preserves a deterministic safety baseline while still benefiting from richer context-aware reasoning.

Another important advantage of SOPRANO is lifecycle continuity. Traditional safety engineering already requires safety to be addressed during design, integration, commissioning, operation, and maintenance. SOPRANO strengthens this lifecycle view by linking design-time analysis, virtual commissioning, systematic testing, and runtime monitoring through a common framework. This is particularly useful in modern robotic systems that evolve over time, for example because sensors are changed, AI models are updated, tasks are extended, or the robot is deployed in a modified workspace. In such cases, digital twins, reusable scenarios, and repeatable testing campaigns provide a practical means to reassess safety in a structured manner instead of relying only on ad hoc physical testing.

To conclude, the SOPRANO SAT solution should be understood as an extension of traditional robotics safety engineering for collaborative MH-MR systems. It remains fully grounded in the established **machinery safety framework of ISO 12100** and the applicable robot and control-system standards. Conventional protective measures still provide the primary means of runtime risk reduction. SOPRANO complements them by improving hazard identification, extending validation through digital twins and simulation-based testing, and supporting runtime supervision of residual risks. This complementarity is particularly important for robots with AI components operating in shared spaces with humans, where safe operation depends not only on stopping when danger is imminent, but also on ensuring that perception, decision-making, and interaction behaviour remain robust enough that such dangerous situations are avoided as far as reasonably possible.

3. SOPRANO SAT SOLUTION

This section describes the main technological building blocks of the SOPRANO SAT solution, namely the safety analysis tools, the digital twin technologies, and the simulation-based testing platform. Although each of these components addresses a specific aspect of safety assurance, their full value emerges from their combined use within a common framework. Safety analysis supports the identification and assessment of risks, digital twins enable the virtual representation and observation of system behaviour, and simulation-based testing supports the systematic evaluation of the system under a wide range of operating conditions and disturbances. Together, these technologies provide the basis for a comprehensive approach to the design-time and runtime safety assurance of MH-MR systems.

3.1. DESIGN TIME AND RUNTIME SAFETY ANALYSIS TOOL

Ensuring safety in MH-MR environments is a major challenge due to the dynamic nature of interactions between robots, workers and the surrounding environment. In such contexts, safety cannot rely only on predefined protection mechanisms. It requires a continuous assessment of risks throughout the system lifecycle, from early design phases to real operational conditions.

Within the SOPRANO project, the safety analysis approach developed by CEA is implemented through the SOPHIA model-based safety engineering platform, which provides integrated support for risk modelling, analysis and monitoring activities. The

approach aims to support a lifecycle perspective by combining design-time risk analysis with runtime safety supervision capabilities.

At design time, the objective is to identify potential hazards, analyse unsafe situations and define mitigation strategies based on risk assessment methods. These activities rely on dedicated modelling extensions for Operational Design Domain (ODD) specification, hazard identification, risk table generation and safety rule derivation. This early analysis helps engineers understand how environmental conditions, perception uncertainties and system behaviours may affect safety.

However, due to uncertainty in real environments, safety issues may only appear during system operation. For this reason, the proposed framework extends traditional safety analysis by introducing runtime monitoring mechanisms. These mechanisms enable the continuous evaluation of safety conditions based on real data and system states. They support the detection of emerging risk situations and the activation of appropriate safety responses during mission execution.

The overall framework therefore combines deterministic and probabilistic reasoning concepts, including structured hazard analysis and emerging approaches based on Bayesian reasoning. In addition, classical reliability analysis perspectives such as FMEA remain relevant in collaborative robotic systems where safety functions depend strongly on perception and software components.

This combined design-time and runtime safety analysis framework contributes to improving the robustness and trustworthiness of collaborative robotic systems. It also supports the validation of safety strategies through simulation-based testing and digital twin technologies, which allow engineers to explore critical scenarios before real deployment.

The following subsections first recall the design-time safety analysis (DSA) workflow and the associated SOPHIA tool support. They then present the main extensions developed during the project, especially regarding runtime safety monitoring, integration with robotic middleware and advanced risk reasoning mechanisms.

3.1.1. Design-Time safety analysis (DSA)

Design-time safety analysis workflow: At design time, the safety analysis process focuses on anticipating hazardous situations before the deployment of collaborative robotic systems. The methodology follows a structured engineering workflow that supports the progressive understanding of operational conditions, risk sources and potential safety impacts.

A key activity consists in formalising the ODD through a structured description of mission phases, environmental constraints and interaction conditions. This structured representation enables engineers to reason about the situations in which safety-critical events may occur and to identify relevant boundaries for safe operation.

Based on this operational characterisation, risk analysis activities are conducted to identify unsafe scenarios and to assess their potential consequences on human-robot interaction. The analysis considers functional insufficiencies, perception uncertainties, environmental disturbances and possible misuse situations that may influence system behaviour. The results are captured in structured risk analysis artefacts that support the classification and prioritisation of safety concerns.

The design-time analysis also contributes to the preparation of validation activities. Identified risk situations are translated into representative test scenarios that can be explored through simulation-based testing environments. This allows engineers to evaluate the effectiveness of mitigation strategies and to refine safety assumptions before system deployment.

Tool support for design time safety analysis: Design-time safety activities are supported by an integrated model-based engineering environment implemented within the SOPHIA safety framework. It extends the Eclipse Papyrus modelling platform by providing a coherent set of safety-oriented viewpoints, modelling artefacts and analysis plugins that support engineers throughout the safety engineering workflow.

SOPHIA also promotes a structured safety engineering methodology that guides the overall modelling and assessment process. As illustrated in **Figure 2**, this methodology follows a progressive workflow starting from system architecture modelling, followed by the explicit representation of dependability aspects such as hazards, risks and dysfunctional behaviours. Safety analyses are then conducted using model-based techniques, enabling the systematic evaluation of failure propagation mechanisms and safety properties. Finally, the results of these analyses are consolidated within the system model, supporting traceability, decision-making and iterative refinement of design solutions. This methodological approach contributes to integrating safety considerations early in the system engineering lifecycle and facilitates consistency between design assumptions, analysis outcomes and validation activities.

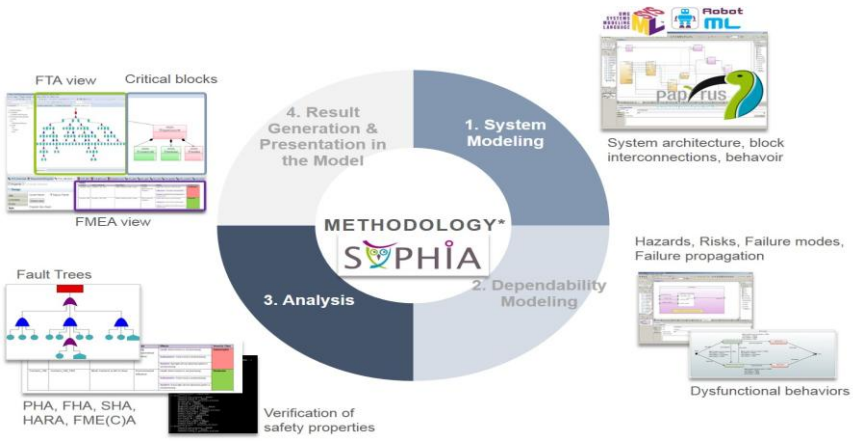


Figure 2: Sophia Tool methodology

Within this environment, the modelling of operational conditions is performed through the Papyrus4ODD capability, which has been adapted and integrated into the SOPHIA platform. This component allows engineers to formally represent contextual assumptions

related to environmental conditions, dynamic entities and mission configurations. Through dedicated graphical constructs, operating features and associated properties can be specified in a structured manner, enabling the systematic identification of combinations of conditions that may influence system behaviour. An example of such graphical modelling is presented in **Figure 3**.

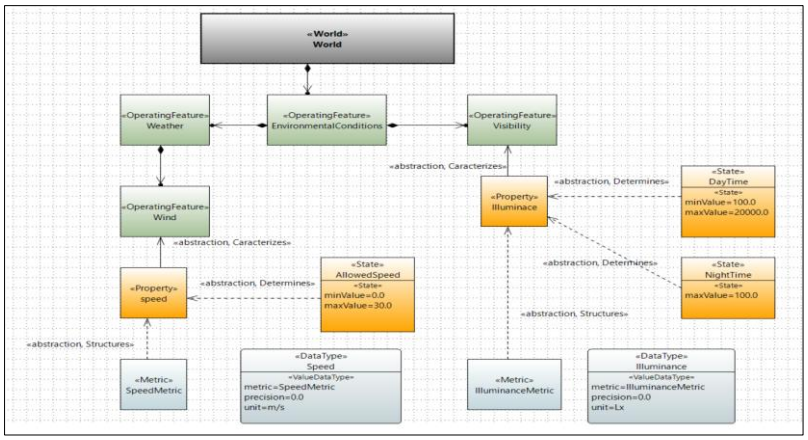


Figure 3: SOPHIA ODD editor

In addition to graphical modelling, SOPHIA provides a complementary textual specification mechanism for describing operating conditions. This dual representation approach improves modelling flexibility and allows engineers to select the most suitable representation depending on the analysis phase. The textual description also facilitates the formalisation of operational constraints and supports automated consistency checks (example is shown in **Figure 4**).

ODD_Example.odd

```
1  ADD Weather TO EnvironmentalConditions AS OperatingFeature
2  ADD Visibility TO EnvironmentalConditions AS OperatingFeature
3  ADD Illuminance TO Visibility AS Property
4  MESURE  IlluminanceMetric Illuminance Type Integer UNITS IlluminanceUnitKind.Lx
5  DETERMINE State DayTime WHEN [ 100 < Illuminance < 20000]
6  DETERMINE State NightTime WHEN [ Illuminance < 100]
7  ADD Wind TO Weather AS OperatingFeature
8  ADD Speed TO Wind AS Property
9  MESURE SpeedMetric Speed Type Real UNITS SpeedUnitKind.ms
10 DETERMINE State AllowedSpeed WHEN [ 0 < Speed < 30]
11 ACCEPT AllowedSpeed WHEN [DayTime]
```

Figure 4: SOPHIA ODD textual DSL

Beyond the specification of operational contexts, SOPHIA integrates a set of safety analysis plugins that support the structured identification and assessment of risks. Classical model-based safety analysis techniques such as Failure Mode and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) are available within the platform and can be applied directly to system architecture models. These analysis capabilities enable engineers to investigate component-level failures, fault propagation mechanisms and critical combinations of events affecting overall system safety. These existing capabilities have been complemented by the development of a dedicated DSA plugin, which extends traditional HAZOP-inspired approaches to better address the specific challenges of MH-MR systems. This plugin allows engineers to capture functional insufficiencies related to perception processes, decision-making strategies and control behaviours. Structured

analysis tables support the systematic derivation of hazardous situations by combining operational assumptions with system functional behaviour. An example of such an analysis table is illustrated in **Figure 5**.

Operating Conditions	In ODD?	Function	Acquisition	Interpretation/ processing	Decision	Keyword	Deviated function	Hazardous scenario	Consequence	Risk Classification
Weather: Clear **Time of day:** Day **Location:** Straight road, Minor road **Road condition:** Dry road, with pedestrian crossover **Vehicle Operation:** drive at low speed **Other road participants:** pedestrian is walking accross (from left)	yes	Perception	Not applicable	Object detection	Not applicable	Faulty	Unable to detect object due to faulty algorithm.	The perception of the environment is faulty while the vehicle is operating on a minor road at low speed, and a pedestrian crosses the lane in front of it from the left side of the road at a close distance.	Collision with pedestrian	Minor

Figure 5: SOPRANO HAZOP table

The integration of these modelling and analysis capabilities within the SOPHIA environment enables a high level of traceability between system architecture elements, operational assumptions and safety analysis outcomes. Risks identified during design time can be explicitly linked to architectural components, functional chains or contextual conditions. This structured traceability supports the generation of consolidated safety artefacts, including hazardous scenario lists, safety constraints and candidate monitoring rules.

Furthermore, the use of a unified modelling environment facilitates iterative refinement of safety analyses. Engineers can progressively update operational assumptions, modify architectural elements and reassess safety impacts using the same modelling workspace. This continuity contributes to improving the consistency of safety engineering activities and reduces the risk of information fragmentation across different tools.

3.1.2. Extension workflow from DSA to runtime monitoring

The extension of the safety analysis approach towards runtime monitoring is supported by a structured workflow that links the design-time artefacts produced in SOPHIA to a dedicated Runtime Safety Analysis (RSA) module implemented in Python. This workflow ensures traceability between hazard identification activities performed during system design and the safety monitoring functions executed during system operation, as illustrated in **Figure 6**.

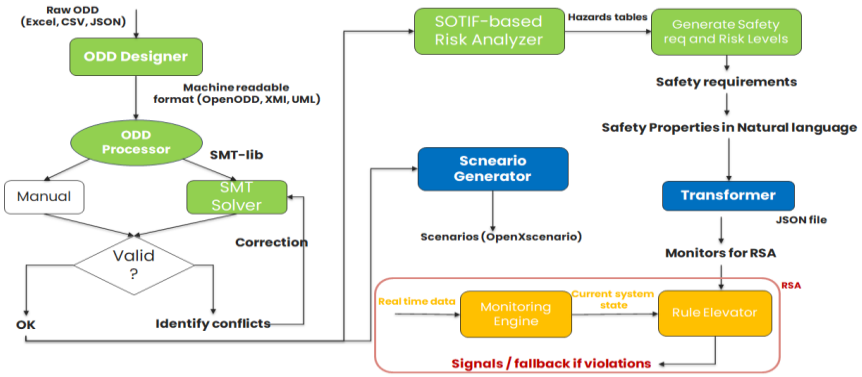


Figure 6: From DSA to RSA workflow

At design time, the workflow starts with the formal specification and validation of the ODD using modelling capabilities available in SOPHIA. Operational conditions are represented through model-based artefacts that can be expressed in machine-readable forms such as OpenODD or UML. Constraint verification mechanisms based on SMT solving techniques are used to detect inconsistencies or conflicts between operating assumptions, thus improving the consistency of the safety analysis process.

On the basis of the validated operational assumptions, the safety analysis capabilities of SOPHIA support the identification of hazardous situations, functional deviations and mitigation strategies. The results are structured into analysis tables capturing unsafe scenarios, associated causes, fallback measures and candidate safety rules.

A key objective of the proposed workflow is to transform these design-time results into runtime-monitorable artefacts. In the current implementation, this transformation is not yet fully automatic. Safety rules and monitoring conditions are derived from the analysis tables through a semi-automatic engineering process, where safety analysts identify the relevant runtime variables, define observable indicators and prepare the rule structures that will be used by the RSA module.

These runtime-oriented rules are then encoded into structured configuration files and provided as inputs to the separate Python-based monitoring framework. This approach already enables a first continuity between design-time safety analysis and runtime supervision. In future developments, this transformation step is intended to be further automated so that SOPHIA can directly generate the rules or monitors to be deployed in the RSA module. In parallel, the workflow also supports the derivation of operational scenarios from the ODD specification. These scenarios can be used in simulation-based testing activities and to evaluate the effectiveness of candidate monitoring strategies before deployment in real environments.

During operation, the runtime monitoring engine processes real-time data streams originating from perception modules, localisation components and robot controllers. The monitoring logic checks the current system state against the rule set derived from the design-time analysis. When violations are detected, the RSA module can generate safety events or fallback requests that are interpreted by higher-level system components.

3.1.3. Runtime Safety monitoring layer

Layered runtime monitoring architecture: The runtime safety monitoring framework is structured as a layered architecture designed to support real-time supervision of collaborative robotic systems while maintaining traceability with the safety analyses performed during system design. The monitoring rules executed at runtime originate from the safety tables and mitigation strategies generated within the SOPHIA engineering environment. This ensures continuity between design-time risk assessment and operational safety supervision. The internal organisation of the runtime monitoring module is summarised in **Table 1**.

Table 1: RSA internal architecture

Layer	Description
Input Layer	ROS Topics (e.g., /camera/info, /vsm/pose_confidence, /lidar/scan, /cta/state), sensor streams, internal system status (e.g. CPU load, battery)
Preprocessing Layer	Timestamp validation, confidence filtering, freshness check, normalization
Rules Elevator	Parses .rsa.rules.json rules and checks real-time conditions
Reasoning Layer	Bayesian Network (BN) for probabilistic reasoning + optional Monte Carlo sampling to manage uncertainty under variation
Fusion State Machine	Combines rule evaluations into a global status (OK, Degraded, Emergency) via a generic runtime State Machine
Output Publisher	Publishes safety events on ROS topics: /rsa/emergency_stop, /rsa/degraded_mode, /rsa/warning, etc.

The input layer continuously collects runtime information generated by the robotic platform and its perception and control subsystems. Typical inputs include human detection outputs, localisation confidence indicators, navigation states or task execution feedback. By maintaining an updated representation of the operational context, the monitoring framework is able to evaluate safety conditions in a dynamic and data-driven manner.

Prior to safety evaluation, incoming data are processed by a dedicated pre-processing layer. This stage improves the robustness of monitoring decisions by verifying message timeliness, filtering perception outputs according to confidence indicators and normalising heterogeneous data formats. Such mechanisms are essential in collaborative environments, where sensing uncertainty, communication delays or signal losses may otherwise lead to incorrect safety assessments.

The core deterministic supervision mechanism is implemented in the rule evaluation layer. Monitoring rules are specified in structured JSON configuration files that describe observable safety constraints such as acceptable human-robot proximity conditions, localisation reliability thresholds or task execution consistency requirements. In the initial prototype, these rules were manually extracted from SOPHIA safety analysis tables and directly implemented in the runtime software as hard-coded logic. The current implementation introduces a more flexible configuration-driven approach in which rule specifications are automatically parsed and transformed into executable monitoring conditions. This evolution improves scalability and facilitates updates when new safety analyses or mitigation strategies become available. Although the derivation of monitoring rules from design-time artefacts still involves engineering judgement, the runtime interpretation of rule configurations is now systematic and modular.

To increase robustness under uncertainty, the architecture integrates probabilistic reasoning mechanisms inspired by Bayesian Network models. This reasoning layer aims to estimate the likelihood of hazardous situations by combining uncertain observations

such as degraded perception confidence or task execution anomalies. This capability is currently under progressive integration within the RSA module and complements the rule-based supervision by introducing continuous risk estimation mechanisms.

A first example concerns the modelling of human–robot proximity. Instead of relying on a strict safety threshold, the risk associated with the distance d is expressed using a logistic function:

$$P_{risk}(d) = \frac{1}{1 + e^{-k(d_{safe} - d)}}$$

where d_{safe} represents a reference safety distance and k controls the steepness of the transition. This formulation enables a smooth transition between safe and unsafe conditions, avoiding abrupt changes when operating near safety limits. This behaviour is consistent with safety principles defined in collaborative robotics standards such as ISO 15066, where safety distances are defined but real-world operation requires accounting for uncertainty and gradual risk variation.

A second aspect concerns the modelling of perception reliability and data validity. The probability of entering a degraded safety state is estimated by combining confidence and data freshness indicators:

$$P_{degraded} = max(0, C_{min} - c) \cdot \alpha + \beta_{stale}$$

where c represents the perception confidence, C_{min} a minimum acceptable confidence level, and β_{stale} a penalty term reflecting outdated or missing data. This formulation captures two complementary sources of uncertainty: unreliable perception outputs and stale information. Such considerations are aligned with the principles of ISO/PAS 21448, where safety must explicitly consider performance limitations and uncertainties of perception systems.

These probabilistic estimations are normalised within the range [0,1] to ensure compatibility with probabilistic reasoning mechanisms and future integration with Bayesian Network models. In the current implementation, these models are not used as standalone decision systems but are combined with deterministic rules within the RSA fusion layer.

In practice, continuous risk indicators are aggregated with rule-based evaluations to support safety state transitions. For example, a high proximity risk combined with low perception confidence increases the likelihood of transitioning to an emergency state, while moderate values may lead to a degraded mode. This hybrid reasoning allows the monitoring system to anticipate risk situations and to provide smoother and more robust safety supervision under uncertainty.

This probabilistic modelling layer also prepares the integration with more advanced reasoning mechanisms. In particular, the computed risk indicators can be interpreted as input variables for future Bayesian Network models, enabling probabilistic inference across multiple sources of uncertainty. In parallel, these indicators can be combined with

structural risk propagation models derived from Fault Tree Analysis (FTA), allowing the monitoring system to reason not only on current observations but also on the evolution of underlying failure conditions.

Furthermore, this approach is particularly relevant in the context of simulation. Continuous risk metrics can be exploited as quantitative indicators to evaluate system behaviour across large sets of automatically generated scenarios. Rather than relying solely on binary safety outcomes, the probabilistic outputs of the RSA enable a more nuanced assessment of safety performance, facilitating the identification of borderline situations and supporting the calibration of monitoring thresholds. The overall behaviour of this probabilistic modelling approach is illustrated in **Figure 7**, where continuous variables such as distance, perception confidence and data freshness are transformed into risk probabilities used in the runtime decision process.

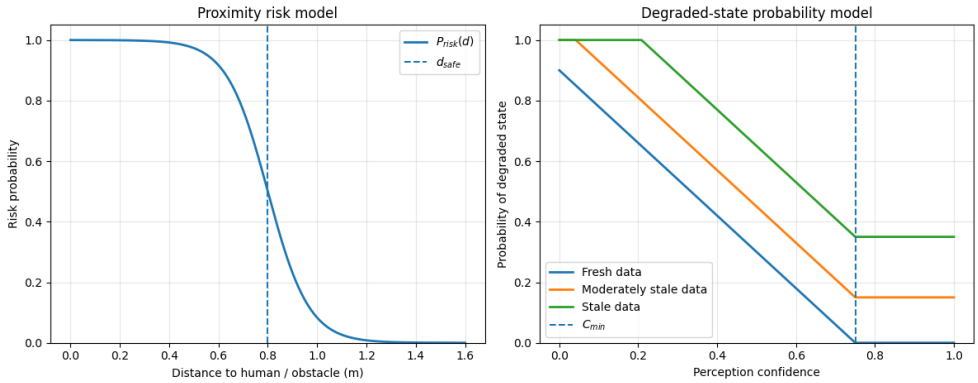


Figure 7: Continuous probabilistic risk modelling used in RSA

Building on this probabilistic modelling approach, future extensions will explore the exploitation of structural risk propagation models derived from the FTA functionalities already available in SOPHIA. At design time, fault trees allow engineers to analyse combinations of component failures leading to hazardous events. The objective is to extend this reasoning towards runtime monitoring by coupling structural fault propagation models with probabilistic inference. Such an approach would enable the monitoring framework not only to detect safety rule violations but also to anticipate the emergence of critical situations based on evolving system degradation patterns.

The outputs of deterministic rule evaluation and probabilistic reasoning are aggregated in a fusion layer relying on a runtime state machine. This component determines the global safety status of the robotic system by classifying the current situation into operational modes such as nominal operation, degraded conditions requiring mitigation or emergency situations requiring immediate protective action. State-based supervision contributes to consistent safety decision-making and helps prevent oscillatory behaviour in highly dynamic contexts.

Finally, the safety signalling layer communicates monitoring results to the rest of the robotic system. Rather than directly controlling robot behaviour, the runtime module publishes safety states, events and mitigation requests through dedicated ROS communication channels. These signals can then be interpreted by navigation, perception

or task allocation components, enabling coordinated safety responses while preserving architectural modularity.

Monitoring logic and safety states: The runtime safety monitoring framework relies on a structured evaluation logic designed to continuously assess the operational safety of the robotic system during mission execution. This logic is centred on the verification of safety properties derived from the design-time safety analysis activities performed within the SOPHIA engineering environment and subsequently translated into executable runtime monitoring rules. Each rule represents an observable safety constraint that must remain satisfied in order to ensure safe human-robot interaction.

At runtime, heterogeneous data streams originating from perception modules, localisation systems, navigation components, task execution managers and internal system status indicators are continuously analysed with respect to these predefined safety constraints. The monitoring process follows an event-driven logic and operates in real time, enabling the detection of risk situations emerging from dynamic environmental changes, sensor uncertainty or unexpected system behaviours. Beyond instantaneous threshold checking, the monitoring logic also considers temporal aspects such as persistence of unsafe conditions, repeated violations or degradation trends observed over successive evaluation cycles.

A key element of the monitoring strategy is the aggregation of individual rule evaluations into a consolidated safety assessment. Rather than treating each monitored condition independently, the framework combines multiple safety indicators in order to determine the global operational safety status of the system. This aggregation mechanism is implemented through a generic runtime state machine that supports consistent interpretation of monitoring outcomes and facilitates coordinated safety responses.

Three main safety states are considered within the current implementation:

- **Nominal state:** all monitored safety properties are satisfied and the robotic system is authorised to continue its operation under normal mission conditions.
- **Degraded state:** one or more safety indicators suggest a potential reduction in operational safety, for instance due to decreased perception confidence, localisation uncertainty or temporary violation of recommended safety margins. In this state, the monitoring framework can generate precautionary safety events or mitigation requests that may lead to adaptive behaviours such as speed reduction or trajectory adjustment.
- **Emergency state:** critical safety rule violations are detected and the continuation of normal operation may lead to hazardous events. The runtime module therefore issues high-priority safety signals, such as emergency stop requests or safety gate activations, which can be interpreted by higher-level control or task allocation components.

Transitions between safety states are governed by configurable thresholds and decision policies that are defined on the basis of risk assessment results, experimental validation campaigns and integration discussions with system partners. In addition, temporal stabilisation mechanisms such as persistence conditions or hysteresis thresholds can be

applied in order to avoid oscillatory behaviour between safety modes in highly dynamic environments.

RSA monitoring logic implementation: The runtime monitoring logic has been implemented as an independent software module developed in Python. The objective of this implementation is to provide a reusable and configurable safety supervision component that can be deployed across different SOPRANO use cases without redesigning the monitoring core for each robotic platform. Although the module is currently external to the SOPHIA engineering environment, its inputs are directly derived from the safety artefacts produced during design-time analysis, in particular the risk tables and mitigation strategies.

In the initial implementation phase, the monitoring rules were manually extracted from the design-time safety tables and hardcoded within the runtime supervision logic. In the current version, this approach has been significantly improved. Safety rules are now defined in structured configuration files using a JSON format and are automatically interpreted at runtime by the monitoring engine. A dedicated parsing mechanism transforms the rule expressions into executable evaluation structures, enabling the monitoring module to process different rule sets without requiring modifications of the source code. This evolution represents an important step towards the future objective of fully automated rule generation directly from the SOPHIA DSA workflow.

The monitoring engine operates according to a continuous evaluation loop, as illustrated in **Figure 8.** At each cycle, runtime data originating from perception components, localisation modules, navigation systems and task execution managers are collected and analysed with respect to the configured safety rules. If all safety conditions remain satisfied, the system maintains its nominal operational state. When a violation is detected, the monitoring logic updates the global safety state and produces corresponding runtime outputs such as safety events or mitigation requests. Depending on the severity of the detected situation, these outputs may lead to degraded operational modes or emergency reactions interpreted by higher-level system components.

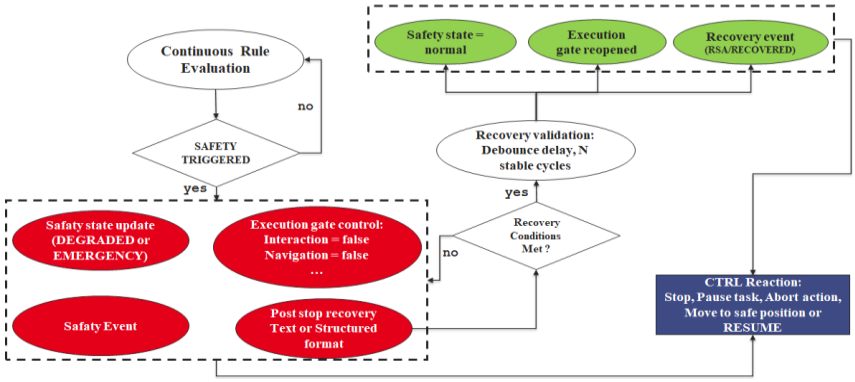


Figure 8: RSA pipeline

A key implementation feature is the consolidation of individual rule evaluation outcomes into a global safety status. Rather than exposing low-level rule violations, the monitoring

module publishes explicit safety states that provide a simplified and consistent interface to the rest of the robotic architecture. This abstraction facilitates integration with navigation or task coordination modules and supports coordinated system reactions. In the current implementation, the monitoring rules are represented in JSON format which allows each rule to be defined in a structured and readable way. A simplified example is given below:

```
{"rule_id": "human_distance_violation",
"input_topic": "/hmo/distance",
"condition": "value < min_safe_distance",
"thresholds": {
"min_safe_distance": 0.8 },
"on_violation": {
"safety_state": "EMERGENCY",
"publish_event": "/rsa/emergency_stop",
"recommended_action": "stop_robot_motion" }}
```

This configuration approach enables flexible adaptation of the monitoring behaviour to different operational scenarios by modifying rule parameters rather than software logic.

Another important aspect concerns recovery management. When a previously detected safety violation disappears, the system does not immediately return to nominal operation. Instead, the monitoring engine verifies that safe conditions remain stable over a predefined validation period or over a minimum number of evaluation cycles. Only after this stabilisation phase does the module publish recovery events and reopen the corresponding execution gates. Such mechanisms reduce oscillatory transitions between safety states and improve the robustness of runtime supervision in dynamic human-robot collaboration contexts.

Runtime signalling and system interaction: The RSA module interacts with the robotic system through the publication of safety-related information using ROS2 communication topics. This interaction follows a publisher/subscriber model in which the monitoring component continuously disseminates safety awareness data while functional modules subscribe to these updates and adapt their behaviour accordingly. This design ensures that safety supervision remains decoupled from low-level robot control while still enabling coordinated system reactions.

During operation, the monitoring engine periodically evaluates the current system state and publishes several categories of runtime safety outputs. These include:

- the global safety state of the system (e.g. nominal, degraded, emergency),
- safety events associated with detected risk situations (for example human proximity alerts or perception reliability degradation),
- mitigation requests, expressed as recommended safety actions to be interpreted by higher-level decision modules,
- execution gate indicators, representing the authorisation level for task continuation or motion execution.

These signals are transmitted through dedicated ROS topics such as safety state updates, emergency stop requests or warning notifications. Navigation, task allocation and

perception modules can subscribe to these topics and implement appropriate responses, including motion limitation, temporary task suspension, trajectory replanning or safe stop procedures (as shown in Figure 8).

A key aspect of this communication mechanism is that the RSA module does not directly command robot actuators. Instead, it provides a structured safety supervision layer that informs decision-making components about the current safety context. This architectural choice preserves modularity and allows safety supervision strategies to be adapted without modifying the internal control logic of the robotic platform.

As operational conditions evolve, the monitoring engine updates safety states, events and gating variables in real time, maintaining a consistent representation of the system safety situation. Recovery signalling follows a similar principle. When previously violated safety conditions are no longer observed and remain stable over a predefined validation period, the RSA module publishes recovery events and reopens the corresponding execution gates. This mechanism supports smooth transitions back to nominal operation while preventing premature resumption of potentially unsafe activities.

3.1.4. Integration within the SOPRANO safety and testing ecosystem

The runtime safety monitoring module is integrated as a supervisory component within the global safety and testing architecture developed in the SOPRANO project. Its role is to ensure operational safety awareness while remaining consistent with the safety engineering artefacts produced during system modelling and validation activities. The safety analysis tool contributes to a broader safety and testing solution that involves simulation platforms, digital-twin services and shared data infrastructures. Within this ecosystem, the RSA module operates at operational time by exploiting runtime information originating from perception modules, robot controllers and monitoring infrastructures. This positioning enables the safety supervision logic to remain aligned with the design-time safety assessment performed within the engineering environment.

The integration workflow follows a lifecycle perspective. During system specification and design phases, safety artefacts such as hazard analyses, mitigation strategies and candidate safety rules are produced and refined. These artefacts can then be evaluated within simulation-based testing environments, allowing engineers to observe system behaviour in representative operational scenarios. Such validation activities contribute to calibrating monitoring thresholds and improving confidence in runtime safety supervision mechanisms.

At operational time, the monitoring module provides continuous safety awareness that can be exploited by decision-making or control components. As depicted in **Figure 9**, runtime safety assessment complements design-time safety analysis by enabling dynamic rule validation and risk situation awareness during system execution. The safety information generated at runtime supports adaptive control enforcement or fail-safe mechanisms implemented within the robotic system architecture. More generally, this integration contributes to ensuring continuity between modelling, testing and operation phases. This ecosystem-level integration is essential for addressing the dynamic and uncertain nature of collaborative robotic environments targeted in the project.

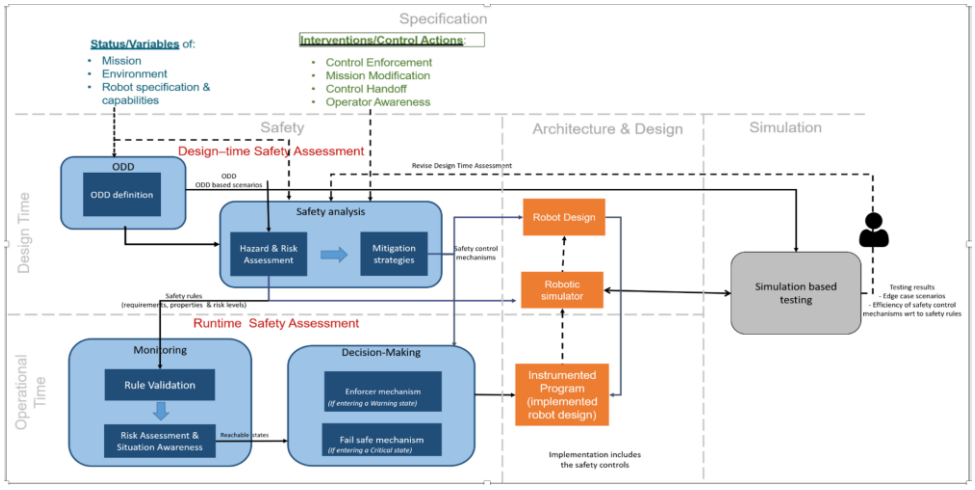


Figure 9: Design-time and runtime safety components

3.2. THE SCALABLE AND EXTENSIBLE SIMULATION BASED TESTING TOOL

3.2.1. Simulation-Based Testing Overview

The simulation-based testing (SBT) platform combines model-driven engineering and exploratory algorithms to search the design space of potential tests during the design time of the SOPRANO workflow. Model-driven engineering and code generation are used in conjunction with distributed communications to run testing campaigns, utilising a number of parallel workers distributed across multiple PCs. Workers launch MH-MR simulations and interface with them, deliberately altering or manipulating simulation state in order to assess the impact of the test upon specific performance metrics (which measure for example, inter-robot distances, safety zone violations etc).

3.2.2. SBT Configuration Methodology

SOPRANO SBT enables the generation of testing campaigns that reveal violations of system safety requirements, in conjunction with domain experts to specify and define the performance metrics and nature of testing campaigns. The SBT component can operate independently in order to test specific performance characteristics, and can also interface with the RSA component, both to obtain its output via performance metrics and to inject simulation changes to trigger its behaviour.

The methodology presented in **Figure 10** should be followed in order to perform an SBT experiment. The methodology steps are labelled according to whether they involve human action or whether they are automated.

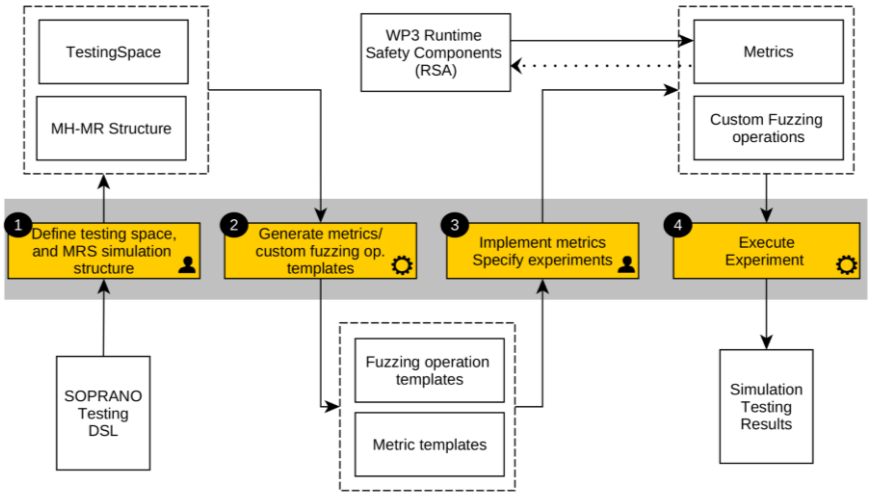


Figure 10: SOPRANO SBT methodology

Step 1: The users specify the testing space specification and MH-MR structure model. It specifies the names and format of the performance metrics and constrains the available testing to be performed, specifying the allowable manipulation types, affected variables and the range of their parameters. The testing DSL is also coupled to an MH-MR structure model which reflects the simulation communications, components and data types, as they are implemented in simulation, and defines the potential connection points to other SOPRANO components such as the RSA.

Step 2: The code generation routines in the experiment manager employ the populated models and automatically generate performance metric templates. The system testers must implement such metrics to quantify whether a mission or safety requirement is met, and if not, the extent and impact of the violation. Since the manipulations selected and the performance metrics are mission and simulator-specific, users are responsible for populating these templates with executable code to define precisely how the simulation events are used to quantify a performance metric.

Step 3: Users implement the previously defined metric templates, to create the required custom scenario-specific metrics. If the performance characteristics incorporate safety requirements which should be verified under SBT campaigns, the output of these can be sampled directly from the RSA subcomponent of the SAT solution. In order to strengthen this integration, during system testing of the RSA with the SBT tools we support a concept known as *metric feedback*, in which the RSA component can also receive its inputs from computed performance metrics within the SBT tool.

During Step 3, in addition to defining the performance metrics, users specify via the testing DSL particular testing campaigns, corresponding to the specific experiments which they intend to execute. These constitute a selection of a subset of the manipulation's operations defined in the testing DSL, and a subset of the defined metrics used to quantify requirement violations. Experiment-specific parameters are also specified for the campaign. For example, when using genetic algorithms to drive the evolution process, the population size and termination criteria such as the number of tests to generate can be included. The specification of the testing campaign may involve testing of not just the MH-MR simulation itself, but also the verification of correct behaviour by the RSA or another SOPRANO component.

Step 4: The SOPRANO SBT platform then uses its experiment manager to perform a given experiment. The experiment manager loads the testing model configuration for the selected campaign, and dynamically generates and launches tests according to the given strategy specified for that campaign. In this process, prediction components can be used in order to accelerate simulation campaigns, by anticipating the results of tests in advance based upon the historical behaviour of previous experiments and skipping predicted low value tests that are not likely to be productive of significant values of the performance metrics. The experiment manager interfaces with remote workers in order to invoke the test and simulator. If the use case uses the Digital Twin as described in Section SEC-DIGITAL-TWIN then this will be invoked in order to simulate that case. During execution the performance metrics are monitored, and feedback from their values is used later in evolutionary optimisation to optimise for more interesting test cases. During the campaign, the experiment manager ensures that performance metric results and prediction status reports are logged back to the model as they become available.

3.2.3. SBT High-Level Architecture

The high-level architecture of the simulation-based testing framework is depicted in **Figure 11**. A primary component of the SOPRANO simulation-based testing framework is a DSL, that supports the specification of the testing space, defining the available manipulations operations to be included in later steps, as well as storing the configuration of the prediction components added in the Final Prototype (FP) and storing prediction results.

The experiment manager which coordinates the evolutionary algorithm evaluates each test by first dynamically generating a specialised test runner which acts as a middleware. Our distributed architecture interfaces with multiple software components (workers) that handle the execution, monitoring and termination of test runners. In order to accelerate simulation execution via increased parallelism, each worker can execute independent simulation tests upon a dedicated PC (this hardware is referred to as the “worker node”). The metric/custom operation definitions and dynamically generated test runners/configuration files are transferred to the workers by synchronization of the generated code.

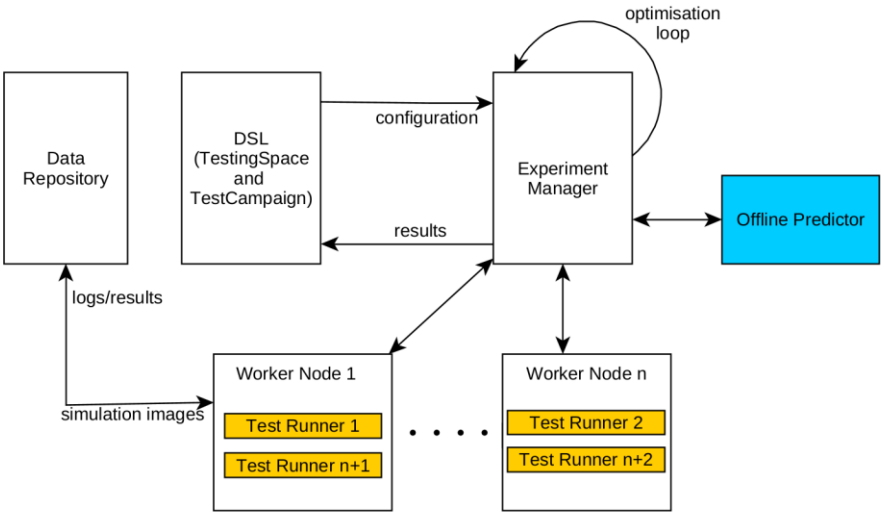


Figure 11: The high-level architecture of the SOPRANO SBT components

Upon the assigned worker, the test runners are invoked and communicate with the low-level simulator via a simulator-specific interface. The test runners implement a 'man-in-the-middle' (MITM) communication setup, where messages from publishing components are redirected via the test runner before reaching their subscribing components. Thus, the test runner may manipulate messages passing between robot components to impact the normal functioning of the MRS simulation. The test runner may act as a null relay, transmitting the messages directly back to recipients unmodified, or may execute some of the available manipulations, based on the configuration of the current test runner and whether each manipulation is currently active.

Via this MITM mechanism, the SBT platform permits several common operations including value modifications (altering message content according to the operation definition and its parameters in the DSL), timing modifications (e.g., delaying a message), or partial or complete dropping of messages (according to a supplied probability). In order to increase flexibility and provide extension points to particular MRS scenarios, additional custom operations can be provided tailored to the particular MRS scenario under study; for example, modifying video transmissions in order to disrupt image processing algorithms. It is also possible to specify message-independent manipulations of the simulation; for example, altering simulation parameters at runtime, or modifying configuration files before simulation execution. These manipulations are described in further detail in deliverable D3.1 on the SBT tools.

Throughout simulation execution, the test runner continuously receives messages and computes metric updates in Step 3 to quantify the impact of the fuzzing test. The final metric results are communicated by the worker back to the experiment manager, which both records them to the DSL and uses them to guide the exploration of the testing space. At the end of simulation execution, the worker terminates the test runner and any simulation containers, and becomes available for the next execution. As workers become available, more work will be assigned to them.

Once the experiment manager has received results for the full generation of an evolutionary experiment from the workers, its evolutionary algorithm (EA) can update its population in response to the evaluated results. This process uses genetic operations such as mutation

and crossover to create new tests, discarding the worst performing campaigns from the population. This iterative process of the EA continues until an experiment-specific termination criterion is satisfied, i.e., either the maximum number of permitted generations is reached, or a specific coverage criterion is achieved indicating an adequate exploration of the testing space. Once the evolution terminates, the testing platform produces the Pareto optimal set of tests that achieve the most impactful values for the performance metrics, which allows these test configurations to be determined. The experiment manager logs these final and intermediate results to the model.

For the Final Prototype of the SOPRANO SBT tools, we extend this approach via predictors (as shown in the "Offline Prediction" component of **Figure 11**) which can filter the population of tests before execution. For example, an offline predictor can be run alongside the experiment manager, attempting to anticipate correlations between the static structure of a test and the performance metrics it will generate. This would allow the acceleration of the overall evolutionary campaign by skipping redundant or low-value tests, while retaining the highest-value tests that contribute most to the evolutionary campaign and its Pareto front. In upcoming sections, we describe the architecture of these prediction components, and later present an experimental evaluation of their performance on the E-TERRY use case.

3.2.4. Offline SBT Predictors

3.2.4.1. Introduction

The final prototype version of the SBT tools considers the addition of performance predictors. These predictors add additional techniques and software modules predicting the outcomes of candidate tests by identifying simulation characteristics most likely to reveal system failures. The major motivation of this arises from the large and potentially non-deterministic search space encountered in testing. Even with simulation parallelisation as described in the early SBT prototype deliverable D3.1, the search space remains intractably large. Therefore, it is important for effective performance to develop strategies that can anticipate the outcome of a proposed test by learning which characteristics of a simulation test are most likely to lead to failures when executed.

The prediction approach presents several advantages. Firstly, it can provide the ability to accelerate simulations upfront by identifying a greater number of interesting tests and discarding irrelevant or low-value test cases before execution begins. Secondly, it may increase the number of high-value test cases that can be examined within a given time frame, as a result of focusing the testing campaigns more intensely. Finally, explainability may be increased through the ability to analyse the prediction models obtained and discover which tests yield significant variations in the system behaviour.

Offline Prediction Methodology: The offline prediction strategy utilises machine learning techniques in order to estimate the impact of test cases corresponding to transient or permanent faults (represented by manipulated, delayed or distorted messages) on a robotic system before their execution. The offline component estimates the impact of test cases for robotics systems, incorporating transient or permanent faults and message fuzzification, before they are executed by devising a machine-learning based predictor. This predictor leverages a historical dataset of previously executed test cases for training and subsequently guides test selection during experimental execution. Then at runtime, the less informative

test cases are filtered out, thus reducing the overall execution time of the testing campaign and utilising the testing budget for the evaluation of high-value tests. In this section, we present the methodology for our prediction training process, specifying the example machine learning techniques that can be used in the construction of predictors, and explain how predictors can be used in practice to enhance the testing process.

Firstly, the offline predictor is trained on a set of previously executed test cases, and prediction components attempt to learn the correlations in the testing set between the static test definitions and the performance metrics obtained (giving the nature and intensity of faults quantified). Then, during the experimental loop, speculative tests can be assessed by using the prediction component to anticipate the performance metrics that will occur if a speculative test is executed. After this, a decision will be made as if the speculative test will contribute to the overall progress of the evolutionary testing campaign through the structure of the predictors, and if not, that test will be skipped. The intent is that this will reduce the overall time spent in testing, or contribute to a greater quality of test campaign results in a given time.

The overall methodology of offline estimation of the quality of a test case is shown in **Figure 12**, and comprises the following steps.

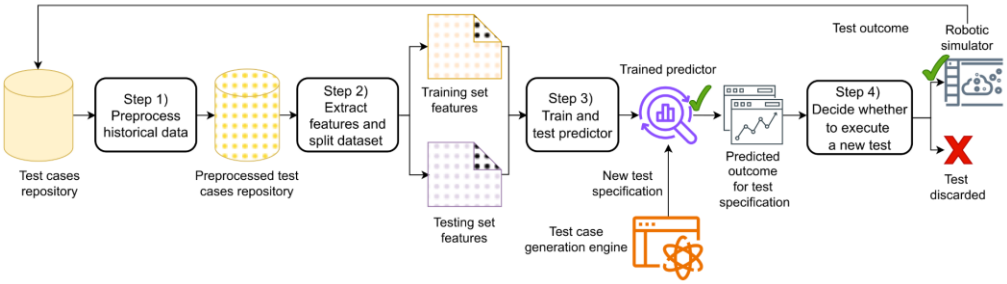


Figure 12: Overall methodology of offline SBT prediction

Step 1 - Historical Data Preprocessing: The offline prediction process leverages training data from previously executed testing campaigns, linking fault configurations to observed outcomes. Given the test specifications, our offline prediction process executes a preprocessing step transforming these specifications into a time series data grid, where each column corresponds to an operation-variable pair and rows denote quantised time steps, with grid cell values encoding operation intensity.

Step 2 - Feature Extraction and Dataset Split: Given the high-dimensional encoding of the timeline representation produced by the preprocessing step, feature extraction techniques are used to extract feature representations using convolutional kernels and windowed timed regions. Following the conventional dataset split process when developing machine learning models, the extracted feature representations are split into two disjoint subsets: (i) a training dataset that acts as the repository of historical test cases used to train the offline predictor; and (ii) a testing dataset used to evaluate the accuracy of the predictor before being put into operation.

Step 3 - Predictor Training and Testing: The quality of the predictor is assessed through its performance on the testing dataset, comparing its predictions based upon test structure to the ground truth recorded performance metrics during the execution of the test set. If its performance is insufficient, returning to Step 2 and adjusting the hyperparameters of the

predictor or using an alternative algorithm for feature extraction is recommended. If the quality is not sufficient with any possible choice, it may be necessary to return to Step 1 and gather additional data.

Step 4 - New Test Prediction and Execution Selection: The offline predictor can now be deployed within the SBT loop to predict the outcome of new tests produced by the test generation strategy. Given the predicted outcomes of a test, specific components known as selectors determine if the test could yield meaningful results and thus should be executed or otherwise discarded. When a test is skipped, it is recorded within the DSL as having been skipped, and execution continues with later tests, which provides an overall time saving for the execution campaign. When a test is executed, its outcome is saved in the repository - both for the generation of results, and for retraining the predictor.

We first describe the fundamentals of the testing specification for fault injection underpinning offline prediction, with particular emphasis on how faults are dynamically varied across test cases during the execution of an optimisation loop for test case generation, and how such experiments can be configured by our domain specific language, including setting the nature and parameters of the predictors to be used within this. We then present the methodology for training the offline prediction model, outline representative AI techniques suitable for the predictor construction, and demonstrate how predictors can be instrumented to enhance SBT.

3.2.4.2. Testing Space And DSL

The SOPRANO deliverable D3.1 specified a DSL which defines the structure of the SOPRANO SBT testing space of possible manipulations. The DSL also acts as a repository of previously executed test results for an experimental campaign, correlating the test definition with the performance metrics. In this section, we will recap from D3.1 how the DSL defines the experimental campaign, and describe the points at which it is extended to support prediction components.

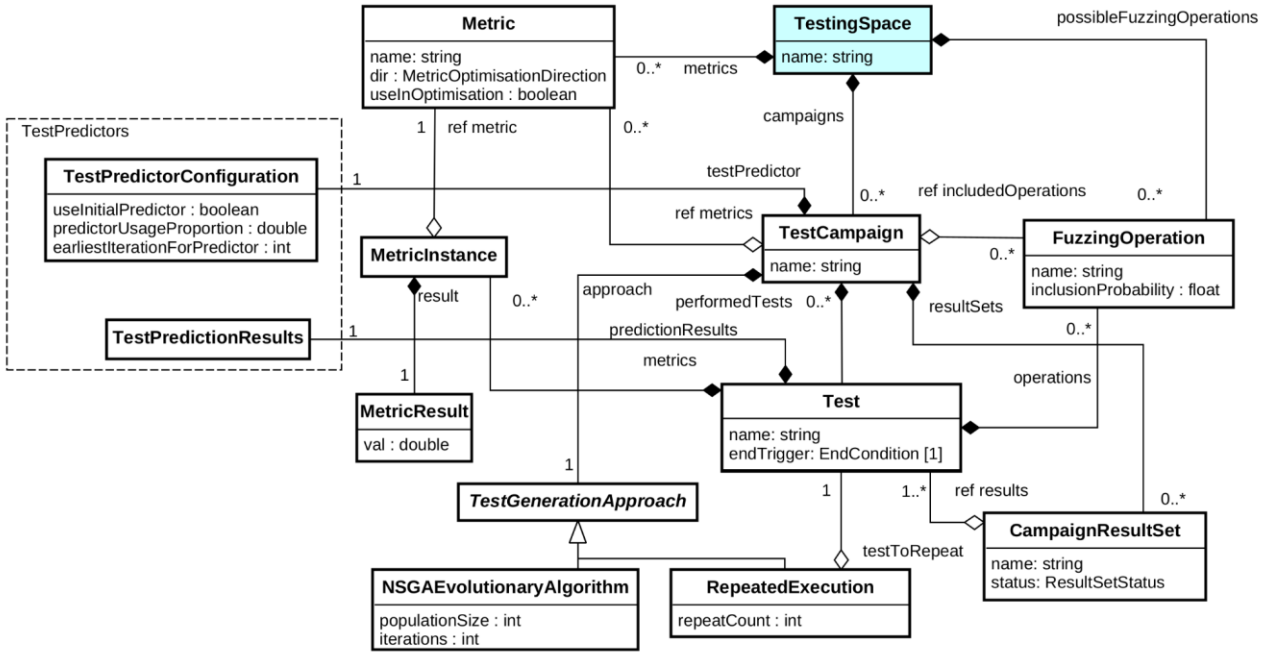


Figure 13: UML representation of the SBT Testing Model DSL

A UML diagram showing a fragment of the main classes of the DSL relevant to testing is presented in **Figure 13**. *TestingSpace* is the root element of the DSL, and consists of a set of permissible manipulation operations. The particular operations that can be injected into the robot system in a particular test are always a subset of the operations in the *TestingSpace*, potentially specialised with more specific parameters or activation conditions. The *TestingSpace* also allows the user to configure multi-objective optimisation, by referencing performance metrics which are used to quantify the robotic system performance. Each of these metrics is represented by the *Metrics* class, which names the metric and specifies whether and in what direction (maximisation or minimisation) it will be used in testing.

The *TestCampaign* class configures an SBT experiment and holds the results obtained during its execution. *TestCampaigns* reference a choice of particular metrics to use when evaluating that campaign, with the *includedOperations* reference list allows the selection of particular operations in order to constrain and focus an experiment. *TestGenerationApproach* allows the user to specify the parameters for an experiment by selecting one of its subclasses. For example, including *NSGAEvolutionaryAlgorithm* selects an evolutionary experiment with the NSGA-II algorithm [8] and lets the user define population parameters. The *performedTests* attribute is populated with the particular tests generated during the execution of experiments. This class contains a reference to a *TestPredictorConfiguration* class which allows the selection of specific parameters relating to the prediction configuration for that experiment.

The *Test* class represents one test configuration that can be applied to the MRS, corresponding to a particular selection of operations with their own customised parameters and activations. The containment of *MetricInstances* records performance metrics resulting from execution of that particular Test. During execution of the *Test*, the metric instance and associated *MetricResult* numeric values are recorded and stored for that test. *CampaignResultSets* contain references to particular sets of *Tests* that serve as intermediate

or final experiment results; for example, the Pareto fronts at various points in evolution as the evolutionary algorithm progresses. The Test also stores *TestPredictionResults*, which store the decisions and choices that the predictor made about a specific test.

The *FuzzingOperation* abstract base class represents one specific manipulation operation - an entity which represents a specific strategy for making runtime modifications or disruptions to the MRS. A hierarchy of subclasses of *FuzzingOperation* are used to represent the specific semantics of this manipulation operation, for example, the *PacketLossNetworkOperation* class represents probabilistic loss of a proportion of packets. *variableToAffect* is the attribute used when setting up variable subscriptions and defining the variables to which fuzzing operations can be applied. It is notable that the *variableToAffect* reference is stored in a subclass, which enforces that some operations can only apply to specific variables.

3.2.5. Offline Prediction Approach

3.2.5.1. Historical Data Processing and Conversion

Offline prediction begins with data derived from the history of previously executed testing campaigns, which provides the foundation for the predictor to learn associations between specific testing specifications (fault configurations) and their outcomes. This information is stored in one or more model experimental files which conform to the testing DSL, where each test case is stored alongside its corresponding outcomes from completed campaigns.

Since the test specifications considered within the offline prediction approach involve dynamic manipulation operations, with operations active only during fixed intervals, the data is represented in an intermediate form as a set of timelines. In this representation, each column corresponds to a specific operation applied to a system variable; rows denote quantised time steps conforming to the quantisation granularity specified by the test engineer. The total number of possible columns in the timeline grid is the product of the number of system variables and the range of possible operations for fault injection. The values within each grid cell specify the activation intensity of the corresponding operation at a given time step.

For operations supporting variable intensities, values are normalised relative to the operation's parameters. For example, an intensity of 1.0 corresponds to the maximum allowable delay, while 0.0 indicates the absence of any delay. Similarly, for message value alterations, intensities are scaled proportionally across the defined range, such that 0.0 represents the minimum and 1.0 the maximum. By contrast, operations with binary activation (e.g., those that are either enabled or disabled) are assigned a fixed intensity of 1.0 when active. If a particular operation-variable pair is not exercised in a given test case, the corresponding column remains zero for the entire period.

As an example, **Figure 14** shows the transformation of an example test case specification Test_001 - into the timeline representation using a 0.5s time quantisation. In this case, two operations on a single robot (a modification of the human X position, and a deletion of a robot laser scanner point cloud messages), are converted into the time series representation.

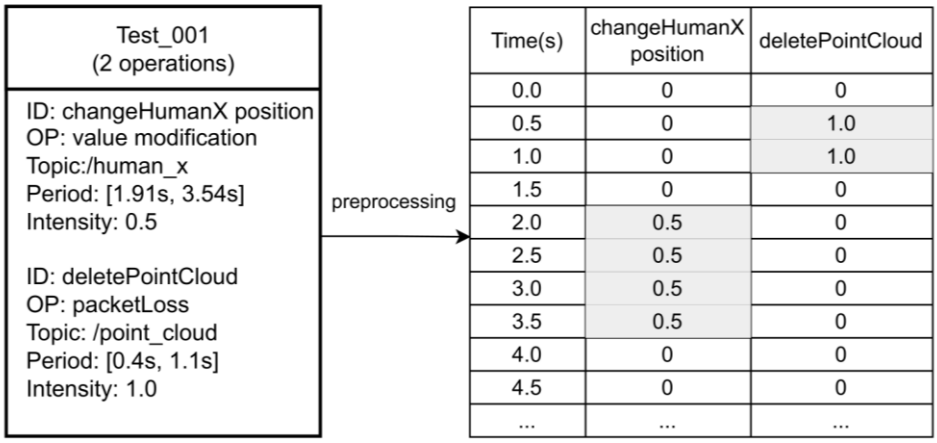


Figure 14: Transformation of a test case definition into time series representation

3.2.5.2. Feature Extraction

Following preprocessing for time series conversion, offline prediction employs feature extraction techniques to aggregate and transform the data into a representation suitable for the predictor. Several alternative strategies for feature extraction are presented below. The choices of feature extraction strategy supported for offline prediction are diverse and a wide variety of potential choices are possible.

An interesting method for feature extraction is ROCKET [9], which generates an extensive collection of randomised convolutional kernels (often to the order of thousands) and applies them to the time series data. Although many kernels may be redundant, the resulting high-dimensional feature space enables efficient feature generation and facilitates downstream learning of those that are relevant to the data. For fault intensity traces that are constant over long periods, the kernels primarily capture the sparse activation points where changes occur. Presently, we evaluate an alternative version proposed by the same authors, named MiniROCKET [10], since it is significantly faster than ROCKET, but also achieves effectively the same or greater accuracy.

A complementary method employs the TSFresh time series processing library [11] to extract statistical features from windowed intervals of the time series, such as the maximum operation intensity within each window. This method provides a fixed and interpretable representation while retaining essential temporal characteristics, since the time series conversion generates roughly constant intensity values over the course of a particular fault which can be easily captured during the maximisation operation. A further example is TSForest [12] which combines feature extraction and learning as covered in the next section.

3.2.5.3. Offline Prediction Learning and Insight Generation

The next offline prediction stage involves selecting an appropriate learning technique capable of identifying associations between the extracted features and the outcomes of testing specifications in the training set. In offline prediction, several alternative learning approaches are supported, each with distinct advantages and trade-offs: gradient boosting yields high predictive accuracy via ensemble learning, ridge regression ensures efficiency

and robustness in high-dimensional settings, and TSForest delivers an integrated treatment of temporal structure.

It is also possible to support insight generation about the particular features from certain generation methods. For example, with gradient boosting regression trees, methods such as impurity-based feature importance, or permutation-based feature importance allow the importance of specific features to be extracted based upon the faults sampled in the training dataset. This enables, for example, determination of the particular operations which are useful in the generation of a particular type of fault, or highly correlated with its presence or absence. When coupled with mechanisms such as widowed time regions, that can potentially support insight generation by identification of particular time regions in which the MH-MR system is potentially vulnerable to specific faults.

Gradient Boosting Regression Trees: In this approach, decision trees are trained on the extracted feature values, with each tree predicting output metric values from the input features. Gradient boosting regression constructs an ensemble of such trees, where each successive tree is trained on the residual errors of the preceding trees, thereby improving overall predictive accuracy by iteratively refining weak learners [13].

Ridge Regression: Ridge regression applies a linear regression model to the extracted features while incorporating an L_2 penalty term. This regularisation term mitigates overfitting by constraining the influence of individual coefficients, thus improving generalisation when dealing with high-dimensional feature spaces.

TSForest: TSForest integrates feature extraction and regression within a single model by recursively constructing decision trees, using entropy and distance measures to determine node splits [12]. Since the available implementation operates only on univariate time series, multi-variate inputs (arising from multiple operation instances) are first concatenated into a single sequence. This transformation is performed automatically before applying TSForest.

3.2.5.4. Offline Prediction Test Selection

Once the learning process completes and the predictor's performance has been validated on testing data, the trained offline predictor becomes available as a discrete component. Given a new test specification, the predictor estimates the performance outcomes that the speculative test is expected to produce. To accelerate SBT via the offline predictions, it is necessary to determine whether the proposed test should be executed or skipped. We introduce a component known here as a *selector*, which receives the speculative metrics, tracks the current experimental population state (for example, the Pareto front and related statistics) and emits an execute/skip binary decision. If the selector deems a test sufficiently valuable, it is executed; otherwise, it is discarded and a new candidate test is generated. Several selector instantiations are available to guide this decision.

Fixed threshold (FT) Selector: The fixed-threshold selector provides a simple baseline that considers thresholds for each requirement and a count parameter (C) specifying how many metrics must meet their thresholds. For each test, predicted metric values are compared to their thresholds, using a greater-than or less-than criterion depending on whether the metric is maximised or minimised. The FT-based selector accepts the test for execution if at least C thresholds are satisfied. This selector is straightforward to

implement and can exploit domain knowledge in selecting thresholds, but it lacks adaptability compared to more advanced methods.

Simulated Annealing (SA) Selector: The SA-based selector adopts adaptive thresholds guided by simulated annealing. A temperature parameter, initially high and decreasing over time, controls the tolerance for executing potentially unproductive tests. Early in the campaign, the selector explores more broadly, to take advantage of initial uncertainty and provide a diverse initial population; as temperature decreases, it becomes increasingly selective. This selector adopts the conventional SA-based decision-making, i.e., if the predicted outcome of a new test is better than the actual best outcome among all executed tests, then the test is executed; otherwise, it is probabilistically accepted as per the Metropolis–Hastings acceptance criterion, considering the delta between the predicted outcome and the current best outcome, and a cooling schedule. In the equation below, P_{exec} gives the execution probability for the test. $SA_{PRED}(t,r)$ and $BEST(r)$ are the predicted and best outcomes for performance metric r, t is the test specification, T_0 the initial temperature and T the reciprocal (Caughy) cooling schedule.

$$P_{exec} = exp\left(-\frac{SA_{PRED}(t,r) - BEST(r)}{T}\right) \text{ where } T \leftarrow \frac{T_0}{epoch+1}$$

Diversity-Driven (DD) Selector: The Diversity-Driven (DD) selector emphasises diversity preservation among the executed tests, preventing prediction from focusing exclusively on tests already predicted as highly useful, encouraging also broader exploration of the test space. The objective is to expand coverage in the objective space and improve approximation of the Pareto front generated during testing. To achieve this, we leverage well-established performance indicators from multi-objective optimisation, specifically Hypervolume (HV) and Generational Distance (GD), which are commonly used to assess the quality of Pareto fronts and their approximations [14]. The Pareto front approximation is updated after each test execution, incorporating feedback from observed outcomes. This selector works as follows. If the predicted metrics of a new test are expected to increase the HV, the test is always executed. When improvements are less certain, execution is determined probabilistically based on the relative GD contribution of the candidate point compared to existing points on the front. This consideration ensures that tests likely to enhance diversity in the objective space are prioritised, while those with lower predicted HV still have a non-zero probability of being selected, depending on their proximity to the front.

The selectivity of this mechanism is governed by a coefficient LP which controls the probability of executing tests that do not yield a predicted HV improvement. Higher values of this parameter encourage greater exploration by accepting such tests more frequently, thus accommodating uncertainty in predictor estimates. **Algorithm 1** provides further details about the various steps underpinning the diversity-driven selector.

Algorithm 1 Diversity-Driven ASSIST Selector

Require: $LP \in [0,1], t$ ▷ lower probability coefficient, test specification
Require: R, PF, PF_{HV} ▷ requirements set, Pareto front approximation and hypervolume
1: $V \leftarrow \{\text{ASSIST}_{\text{PREDICT}(t,r)} \forall r \in R\}$
2: $\widetilde{PF} \leftarrow \text{EXTRACTPF}(PF \bigcup \{V\})$ ▷ Hypothetical PF with predictor outcomes
3: $\widetilde{PF}_{HV} \leftarrow \text{COMPUTEHV}(\widetilde{PF})$ ▷ Hypervolume calculation
4: **if** $\widetilde{PF}_{HV} > PF_{HV}$ **then**
5: **return** True ▷ Always execute if HV improves
6: **else**
7: $V_{GD} \leftarrow \text{COMPUTEGD}(PF, V)$ ▷ Computer GD for V
8: $PF_{GD} \leftarrow \{\text{COMPUTEGD}(PF \setminus v, v), \forall v \in PF\}$ ▷ Compute GD for all elements in PF
9: $p_{\text{accept}} \leftarrow LP \times (1 - \frac{V_{GD}}{max(PF_{GD} \bigcup \{V_{GD}\})})$ ▷ Acceptance probability relative to the GD distance
10: **return** $\text{RAND.UNIFORM}() < p_{\text{accept}}$
11: **end if**

Algorithm 1: Diversity-Driven Selector for Offline Predictor

3.2.5.5. Testing and Prediction DSL Modifications

For the final prototype, a new package *TestPredictors* was added to our DSL in order to configure the prediction strategies to be used in a particular experiment, and to handle the storage of prediction results. This section describes the DSL for offline predictor configuration and how it interfaces with the previously described DSL changes.

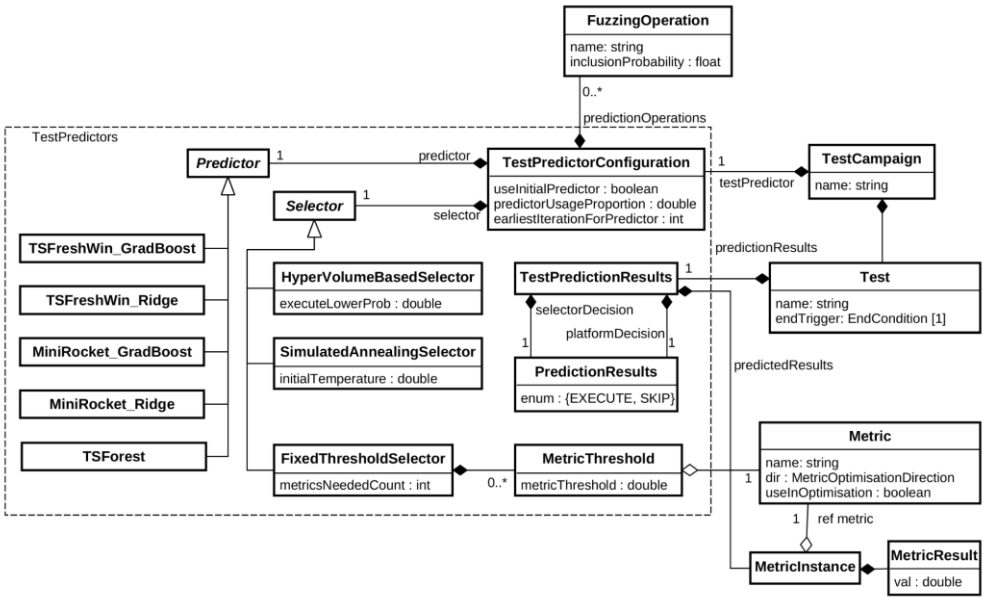


Figure 15: UML package for configuration of offline predictors

Relevant parts of the prediction DSL (in a new package TestPredictors) and its connections to existing DSL classes are illustrated in **Figure 15**. The *Predictor* abstract base class is extended by several implementations, corresponding to particular combinations of feature extraction and learning strategies from a given experiment. The *Selector* abstract base class defines the component for test selection, and three extensions of it give the relevant. For example, with the *FixedThresholdSelector*, instances of *MetricThreshold* permit the choice of metrics that should be used in threshold selection, and the threshold values. With the *SimulatedAnnealingSelector*, the initial temperature T_0 can be defined, and with the *HyperVolumeBasedSelector*, *executeLowerProb* gives the value of LP for **Algorithm 1**. The *TestPredictorConfiguration* is referenced from the *TestCampaign*, and allows the

choice of a particular predictor and selector, together with parameters when being used in an experiment. For example, the use of an initial predictor (which would always be true except when using dynamic retraining), the proportion of elements in the population which should use the predictor, and the first generation to use the predictor. This allows ensuring a good balance between exploration and exploitation, and maintains diversity in the initial population.

TestPredictionResults are referenced from a *Test*, and store the decisions that the predictor made about a specific test; i.e. whether it should be executed or skipped. There are two fields in *TestPredictionResults*; *selectorDecision* and *platformDecision*. The selector decision field stores the decision received from the selector component directly. But the platform decision may potentially override it, for instance insisting on execution even when the selector itself decides to skip a test. This supports, for example, ensuring that execution is selected for all the early tests in a population, to ensure sufficient diversity upon selectors that do not natively support diversity. Also, the predicted metrics are stored under *TestPredictionResults*, giving the predicted metric values using *MetricInstances*. This both supports debugging (by allowing actual predictions to be compared against empirical results, in cases in which the test is executed) and supports potential dynamic retraining of the predictor.

3.2.5.6. Offline Predictor Implementation and Communication

The offline predictor is implemented as a Python component using the sktime library [15]. It is typically executed upon the experiment manager PC, but runs as a separate Docker container in order to permit the use of distinct libraries and package versions. It communicates with the experiment manager via the same Pyro interface as described for worker communications in deliverable D3.1, to determine whether a particular speculative test should be executed or not. The high-level architecture of the system involving the online predictor was as specified earlier. In this section we describe the communications from the experiment manager to and from the predictor, showing relevant fragments of the worker node communications.

The communications from the experiment manager to and from the predictor daemon operate as illustrated in **Figure 16** with the following numbered steps:

1) In response to the user starting the experiment upon the Eclipse console of the experiment runner, the experiment manager sets up the evolutionary algorithm and its population. For all generations of the population, time series conversion of the test specification is performed, and CSV files representing the generated time series are placed under the shared code directory to be communicated to the predictor daemon, together with the generated code for these tests for the worker daemon.

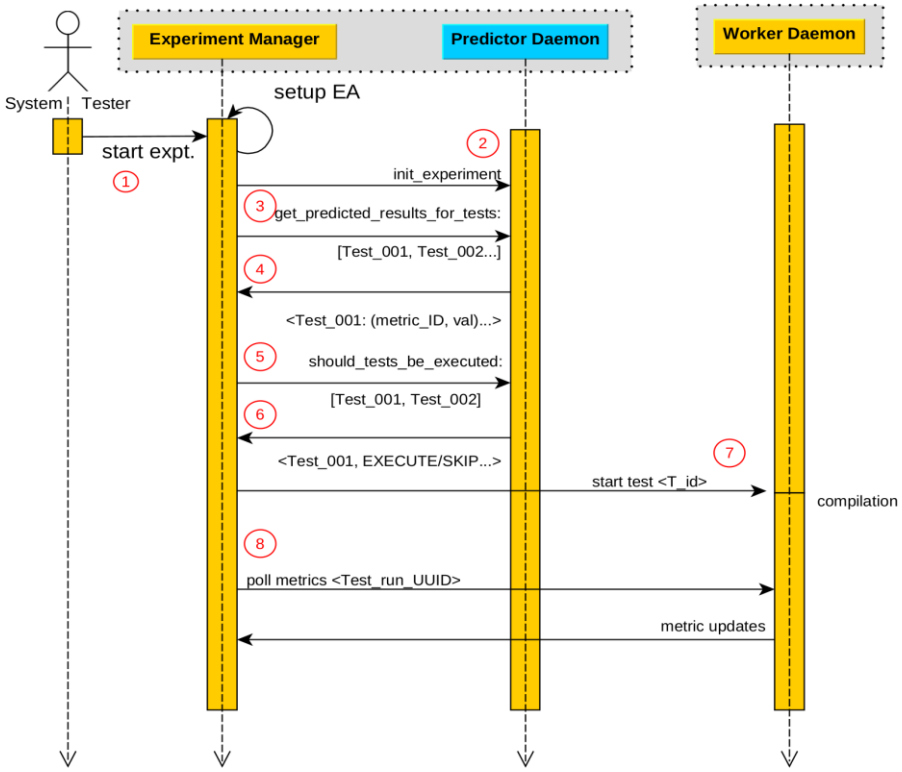


Figure 16: Experimental loop communications to and from the predictor daemon

2) The experiment manager transmits a pre-initialisation message to the registered worker predictor for the experiment. This includes an encoding of the predictor configuration data (from TestPredictorConfiguration) as provided in the DSL, specifying the choice of predictor and selectors. The predictor daemon loads the necessary pre-trained predictor file based upon a picked Python image object.

3) The predictor sends a Pyro message *get_predicted_results_for_tests*, specifying the relevant test IDs for the upcoming generation. The predictor daemon synchronises over rsync the CSV test definitions, and invokes the prediction component to determine the metrics (internally using the feature extraction and inference from the trained component).

4) The predictor returns a map of predicted metric values for each test. These metrics, and the platform decision are stored in the DSL, both for debugging and to allow the predicted values to be compared with actual values.

5) The predictor sends a Pyro message *should_tests_by_executed*, specifying the relevant test IDs for the upcoming generation. The predictor daemon takes the metrics as determined in Step 4, and passes them to the relevant selector component to generate an EXECUTE/SKIP decision based upon the predictions.

6) The predictor returns to the experiment manager a map of EXECUTE/SKIP decisions for each test. These predictor decisions are stored in the DSL.

7) If the platform has chosen to execute the speculative test, then the worker daemon is triggered to begin the test execution as normal.

8) The worker daemon executes the simulation and normal and returns the metric values during the simulation. These metrics can be stored in the DSL as the expected results of the test.

Further Work: This section has covered the offline prediction methodology and its application to the SBT component, presenting the overall methodology, feature extraction and learning strategies, selectors, DSL modifications and the communication flows.

Potential future work and extensions will involve:

- Integrating prediction strategies with online feedback from live simulations, enabling dynamic decisions to terminate partially completed tests during execution, if their ongoing development is anticipated to not lead to productive outcomes. This enhancement would enhance the adaptability to non-deterministic robotic testing scenarios and further increase the efficiency of larger-scale testing campaigns, which can be useful during integration testing with more sophisticated SOPRANO use cases.
- Dynamic retraining of predictors during execution, in particular modifying the evolutionary strategy in response to prediction decisions. This allows for correction in cases in which the evolutionary behaviour during an experiment does not conform to well-explored regions of the training set. Dynamic retraining would allow updating the predictor during an evolutionary experiment with incremental retraining, taking into account the ongoing test structure and performance metrics during the current experiment. This could be performed either periodically or when triggered by performance statistics of prediction quality. Another alternative would be to guide evolution (through selection) towards regions of the search space in which predictors are uncertain, in order to support more efficient exploration of the search space and expose challenging edge cases.
- Use of alternative feature time series analysis and feature engineering strategies, for example, currently our time series conversion is performed using a static time series conversion of all variable-operation combinations. A more advanced time series conversion and feature-generation strategies could capture the relative timing interactions between faults, thus improving sensitivity to correlated fault behaviours and increasing scalability to more sophisticated use cases.
- Integrated deployment of the SOPRANO SBT component into the wider integrated system for the supported use cases (E-TERRY/KUKA), supporting the analysis of safety scenarios and associated faults/vulnerabilities in external SOPRANO components. In addition, using feedback from these components in computation of system performance metrics.

3.3. HUMAN DIGITAL TWIN FOR MH-MR SYSTEM (HDT)

3.3.1. Introduction

In this chapter, the new developments and improvements of the Human Digital Twin (HDT) are presented with respect to what was reported in Deliverable D3.1.

As a brief recap, the main objective of this task is to develop a digital counterpart of an MH-MR system, where humans and robots share the same working environment and collaborate to achieve a common goal. The HDT enables the identification of potential hazards and failure points during the design and virtual commissioning phases, thereby reducing the risk of unexpected issues during real operations.

Additionally, it supports the verification of the reachability of all waypoints where the robot must perform its tasks, as well as the assessment of the cycle time required to complete a job. The HDT developed in this task is applied and validated on the building renovation use case where the KUKA mobile platform is involved.

During the first period of SOPRANO project, a prototype of **Human Digital Twin** was developed with the primary objective of simulating human walking behaviour within the working area, in order to analyse potential collisions between humans and robots. No advanced human behaviours were initially modelled, as the primary objective of this HDT is to assess the safety level of an MH-MR system by evaluating the risks associated with human presence in a shared workspace. The underlying principle is that the robot must implement the necessary safety procedures to avoid collisions with workers, rather than requiring workers to adapt their behaviour to the robot. This approach is aligned with the human-centric paradigm promoted by Industry 5.0.

A **Digital Twin of the KUKA mobile platform** was developed using the Digital Twin Service platform, by TTS. A virtual representation of both the crawler and the KUKA robotic arms was implemented, enabling the simulation of

- the crawler's movements across the floor
- the robot joints during task execution.

Additionally, the Digital Twin prototypes of two different types of sensors were developed:

- **The infrared positioning sensors**: these sensors are used by the KUKA mobile platform to measure distances from surrounding objects, primarily the walls of the building that must be renovated. For this reason, they are also referred as positioning sensors.
- **The safety laser scanners**: these sensors scan the environment in two dimensions using infrared laser beams, enabling safety and proximity detection. They allow the definition of multiple safety:
  - "**Protective field**", when an obstacle is detected within this field, the safety laser scanner immediately sends a signal to the crawler to halt.
  - "**Warning field**", two warning fields can be defined to trigger different responses from the crawler. The first warning field sends a signal to slow down the speed, while the second field can trigger a warning notification.

The Digital Twin prototypes of the sensors include a simplified 3D model, along with their logic required to simulate measurement and detection objects. Once instantiated, the end user can configure parameters such as the measurement, detection angles, and safety zones dimensions.

When connected to the virtual robot, sensor signals are transmitted to the robot controller, enabling it to react according to the implemented control logic. These reactions may include issuing warnings or immediately stopping the robot's movement.

In summary, during the first SOPRANO project period, Digital Twin prototypes for the building renovation use case were developed together with the basic control logic required to simulate the movements of the crawler, the KUKA robot and the workers within the shared workspace.

In the second phase of the project, the main objectives are:

- The **integration of HDT with the SBT tool** to perform simulation-based testing campaigns aimed at identifying vulnerabilities in the MH-MR system, including corner cases.
- The **integration of HDT with MAT tool** to import the environment, tasks and working points.
- The **integration of the HDT with the mobile platform**, enabling validation of the safety logic implemented in the robotic system prior to deployment in a real environment, as well as verification of the reachability of working positions on the wall.

These integrations require enhancements to specific features of the Digital Twin Service platform, in order to ensure efficient data and signals exchange and to improve the simulation capabilities of both human and robot Digital Twins. The following section describes these improvements in detail.

3.3.2. Data management and communication mechanism

Within the Digital Twin Service (DTS) platform developed by TTS, data management and communication mechanism is implemented through a core component named **Shared Memory**, which can be mapped to the Data Management and Integration layer defined in ISO 23247. This component is responsible for managing structured data repositories (registries) and enabling controlled data exchange between Digital Twin elements. Each registry represents a logical data container associated with specific information flows of the Digital Twin, such as state variables, sensor data, or simulation outputs. In ISO 23247 terms, these registries support the representation and synchronization of the digital representation of the physical entity.

Data exchange between registries is governed by a configuration mechanism called the **datamap**, a JSON-based configuration file that defines how data flows across the system. This mechanism implements the information integration and mapping functions described in ISO 23247, enabling interoperability between heterogeneous components without requiring direct coupling.

The interaction with the Shared Memory is performed through modular components called **datasources**, which can be interpreted as communication and integration interfaces within the Communication Network layer of ISO 23247. Each datasource is responsible for interfacing with a specific external system or protocol by reading and writing data to its associated registries. While a datasource can create and manage its own registries, it operates within a controlled scope and cannot directly access registries owned by other datasources, ensuring modularity and data isolation.

A dedicated **simulation datasource** is used to inject into the Shared Memory all relevant variables generated by the simulation environment, including robot states, sensor

measurements, and environmental conditions. This component contributes to maintaining the dynamic behaviour of the Digital Twin by continuously updating its digital representation.

Within the SOPRANO project, the combination of Shared Memory and datasources enables the integration of the Digital Twin with both simulated and real-world systems, supporting bidirectional data exchange between the Digital Twin and external components representing the physical entity or other digital services. This is achieved through protocol-specific datasources, including:

- A **gRPC datasource**, enabling the communication with the SBT component
- An **OPC UA datasource**, supporting interoperability with industrial systems such as with the KUKA OPC UA server
- A **ROS2 datasource**, enabling interaction with robotic middleware such as the Nav2 stack.

These datasources typically operate as follows: during the initialization phase of the simulation, they instantiate the required registries within the Shared Memory. The Shared Memory then establishes the data flows between registries according to the datamap configuration. During runtime, datasources periodically check their output registries to determine whether updated values need to be published. At the same time, they monitor incoming data through their respective communication protocols to update the Shared Memory registries accordingly. To improve efficiency and align with event-driven communication paradigms described in ISO 23247, datasources may also implement subscription-based mechanisms, whereby updates are propagated only when new data becomes available, reducing unnecessary polling and communication overhead.

As an example, in the interaction with the SBT component, data generated within the Digital Twin are first published into the Shared Memory through the simulation datasource. Through the datamap configuration, these data are then routed to the registries associated with the gRPC datasource, which exposes them via gRPC endpoints to the SBT component. Conversely, data produced by the SBT component are received through the same datasource, routed via the Shared Memory, and made available to the simulation environment as shown in **Figure 17**.

This architecture ensures compliance with key ISO 23247 principles, including **modularity,** interoperability, scalability and decoupled communication. The Shared Memory acts as the central data integration backbone, while datasources implement standardized communication interfaces, enabling seamless extension of the Digital Twin with additional systems and protocols. New datasources can be easily integrated to support additional communication protocols, while the Shared Memory ensures efficient data exchange without requiring direct coupling between system components.

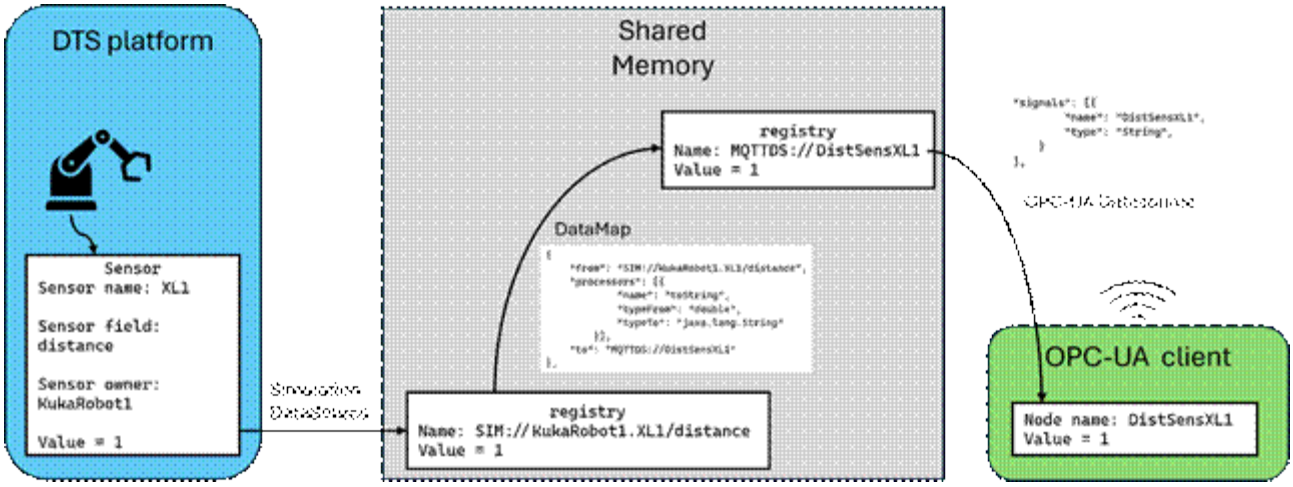


Figure 17: Example of the use of the Shared Memory and the datasources in SOPRANO, here with a OPC-UA datasource that publishes one value via OPC-UA

I/O Files for HDT simulation

This section describes the structure of the input and output files used to run a simulation of MH-MR systems. The following input files are required:

1. Workplan input file
2. Environment input file

At the end of each simulation run, three output files are automatically generated:

1. The results output file
2. The task trace output file
3. The collision trace output file.

The following sections describe their structure and semantics.

Workplan input file

The Workplan input file is written in JSON format, and consists of three main sections:

1. **Tasks**: defines all the information required to perform a complete job.
2. **Constraints**: specifies tasks dependencies and execution logic.
3. **ActionPlan**: assigns tasks to actors and defines execution order.

This file is automatically created by the MAT module.

Task

Every task is defined by an **id**, **type**, **taskData** structure describing task specific information and an **executionData** array describing how different actors perform the task.

Name	Type	Notes
“id”	String	Unique identifier of the task

Name	Type	Notes
“type”	String	Type of the task
“taskData”	Custom	Task specific parameters
“executionData”	Custom	Actor-specific execution parameters

The common fields are illustrated in the table below.

The main task types are:

- Move
- Milling
- Plastering and installing

The **Move task** is used to move an actor within the environment on a floor. The mobile platform follows a predefined path and reacts to safety zones detected by laser scanners. The key parameters are:

- Path: sequence of 3D coordinates
- Speed (human)
- speedLevel (robot: Normal of Fast)

The **Milling task** represents the execution of wall-processing such as sockets or grooves. When performed by a robot, the crawler attempts to reach the target destination in front of the wall, and the robot moves its arms to achieve the desired point on the wall and starts milling. If unreachable, the task is marked as failed and the failure is reported in the “Task trace output file”.

For human execution, the worker reaches the target destination and starts working. The task duration is defined explicity in the “executionData” field.

The main taskData element are:

- **Socket**: contains information about the position, number and size of the socket that will be milled on the wall.
- **Groove**: contains information about the start, end and depth of a groove

Socket describes a line of socket. It could be a vertical line or an horizontal one. Sockets of the same group have the same diameter, the same distance between each other, and the same depth. The position refers to the center of the highest socket (in a vertical line) or the leftmost one (in an horizontal line).

The **plastering and installing task**s represent manual operation performed by workers on walls.

The main parameters of these tasks are:

- Position: the working point
- executionTime: time required to complete the task

Constraints

- The constraints define dependencies between tasks.

- The most common type “follows”, which specifies that a task can only start after other tasks are completed

Plan

The plan assigns tasks to actors and defines the execution order.

Each actor is associated with:

- A starting position
- A sequence of tasks

Actors execute their tasks independently unless constrained by dependencies.

The Environment input file

The Environment input file, formatted in JSON, describes the rooms and walls of the simulated environment of an MH-MR system. It includes:

- Storeys
- Rooms
- Walls

Walls are defined to be sensible to collisions.

The result output file

This JSON file contains performance statistics of the system, including:

- Actor saturation rate
- System-level metrics

Saturation is expressed as the percentage of the time spent in each state:

- Work: the actor is performing a task
- Move: the actor is moving on the floor
- Blocked: the actor cannot continue its activity because something blocks it
- Idle: the actor is inactive.

The task trace output file

The file, written in JSON format, provides detailed information about task execution, mainly whether the task has been executed successfully or not. It includes:

- Actor performing the task
- Execution time
- Task result (Success/Failed)
- Failure notes

The collision trace output file

The file, written in JSON format, records all collisions events during each simulation run. It includes the following information:

- Timestamp
- Involved actors
- Collision severity can be low, medium, critical, if the two actors have physical contact.

Simulation Improvements

This section presents the improvements introduced to enhance the simulation of the building renovating use case.

To ensure that the data generated by the simulation is consistent with the real system, an accurate and high-fidelity representation of all involved actors is required. In particular, the execution of tasks, such as the milling of sockets, must replicate the behaviour of the real robotic platform as close as possible. This is achieved through detailed behaviour modelling, ensuring that the Digital Twin reproduces the same operational logic, motion constraints, and task sequences as the physical entity. This also allows to get an accurate estimation of the real processing time and verify whether all the target points are reachable by the robotic arms.

The milling process is modelled as a sequence of well-defined steps:

- Reaching the approach position
- Performing the milling operation
- Returning to the safety position

The first step consists of positioning the robotic arm in a collision-free configuration. The end effector follows a trajectory that includes intermediate poses specifically designed to avoid collisions with the surrounding environment, mainly the wall. Trough these intermediate poses, the end-effector reaches the approach position, where the milling bit is correctly aligned in front of the target milling location.

Once in position, the robot performs the milling operation following the same strategy as the real system: the milling bit enters the wall perpendicularly and progressively enlarges the hole following a spiral motion. The Digital Twin can simulate the execution of multiple socket holes arranged in rows or columns, (i.e., socket groups), in accordance with Kuka specifications. If required, the process is completed by milling the corresponding vertical or horizontal duct.

After completing the milling task, the virtual robot transitions back to a predefined safety position (see **Figure 18**) to ensure safe movement within the environment. This transition is again performed trough intermediate collision-free poses, guaranteeing that no unintended contact occurs during motion.

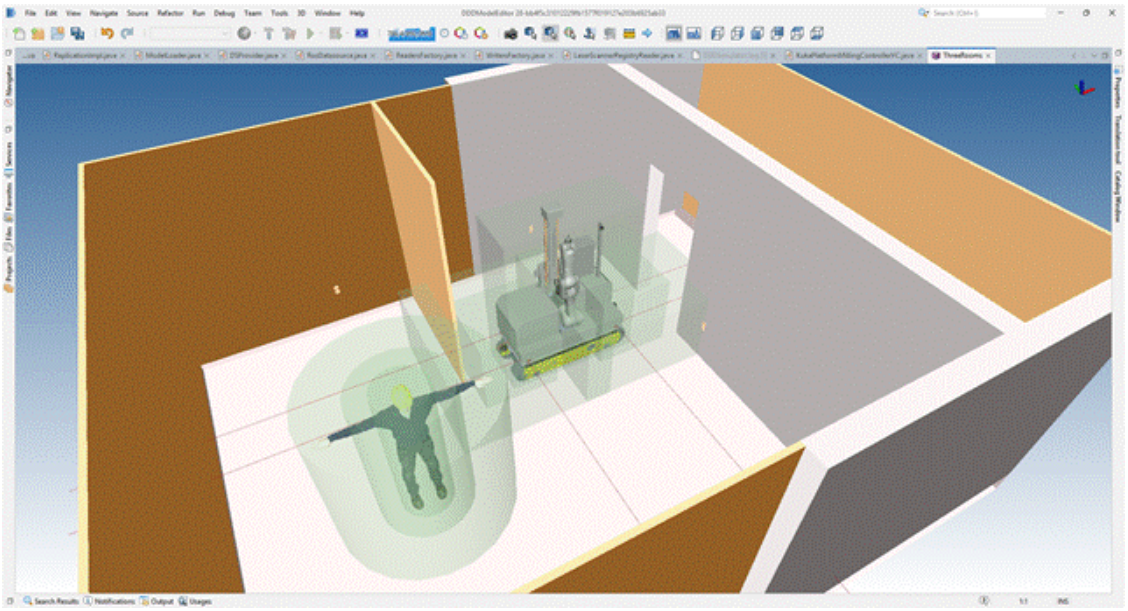


Figure 18: The KUKA platform Digital Twin and the HDT in the simulation environment. The Kuka platform is in the safety pose with the robotic arm going up, to not hurting anything while crawling around, and the worker is the rest pose, no animations are playing.

In addition to improve accuracy and realism in simulating milling process, significant improvements have been introduced in the simulation of the mobility of the robotic platform. Unlike the initial prototype, where the robot was constrained to linear movements (forward and backward), the updated system allows full navigation capabilities within the environment. The virtual mobile robot can now move freely on the floor towards specified navigation targets, which are defined in the input task file. The motion behaviour of the Digital Twin—including parameters such as speed and movement dynamics—are calibrated to reflect the real platform, ensuring behavioural consistency between the Digital Twin and the physical entity.

These enhancements enable the robot to operate on any wall within the environment. While the initial prototype supported operations only on a predefined wall, the current virtual system is capable of determining the correct position and orientation for any milling point based on the information provided in the task input file. This capability reflects an advanced level of context-aware behaviour modelling within the Digital Twin.

A similar modelling approach has been adopted for human workers represented as Human Digital Twin. The execution of human tasks is defined within the task description file, which specifies the sequence of operations to be performed. The Human Digital Twin interprets and executes these tasks enabling the simulation of human activities in MH-MR systems.

The Human Digital Twin is capable of operating on any wall and performing both robot-equivalent and complementary tasks, such as socket installation and plastering. These actions are synchronized with appropriate animations, providing a clear and interpretable visualization of human behaviour within the Digital Twin environment. This ensures that both robotic and human activities are consistently represented, supporting the analysis and validation of tasks performed in shared environments.

3.3.3. Integration between HDT and SBT

The SBT component, developed within the SOPRANO project, is designed to construct and execute test campaigns aimed at identifying weaknesses and improving the reliability of MH-MR systems. To achieve this goal, the SBT operates in conjunction with the HDT component that provides a high-fidelity representation of the system under analysis.

To support the SBT in exploring edge cases and failure scenarios, a set of simulation parameters is exposed and can be modified dynamically, even at runtime. This enables the SBT to manipulate the Digital Representation of the system and evaluate system behaviour under controlled perturbations. In the building renovating use case, the following robotic parameters are exposed:

- The 6 distance laser sensors
- The 2 safety laser scanners
- The 2 LiDAR sensors
- The joint values of the robot's 7 axes

By controlling these parameters, the SBT can simulate various fault conditions by changing the appropriate parameter. For example, fixing the value of a robot axis can emulate a mechanical failure, while injecting abnormal sensor readings can reproduce sensor malfunctions. This capability supports what-if analysis and fault injection, which are key aspects of Digital Twin-based validation in ISO 23247.

To ensure reliable interaction between the SBT and the HDT component, it is necessary to guarantee that the data exchanged through the interfaces remain stable during analysis. For this purpose, a set of control APIs has been introduced to manage the execution flow of the simulation. These APIs allow the SBT to start, pause, and resume the simulation. When the simulation is paused, the system reaches a consistent state in which data values are stable, enabling the SBT to perform reasoning and determine the next actions without interference from ongoing state changes. This mechanism aligns with the concept of synchronized interaction between Digital Twin components and services.

Communication between the SBT and the HDT component is implemented through the gRPC protocol, which provides efficient, low-latency, and language-independent communication. Initially, due to architectural constraints, two separate components were used:

- a **gRPC controller**, responsible for sending control commands to the simulation engine
- a **gRPC datasource**, responsible for exchanging simulation data.

Following an upgrade of the simulation engine, which improved both performance and architectural flexibility, these functionalities were consolidated into a single component: an enhanced gRPC datasource capable of handling both data exchange and simulation control commands.

In the current architecture, data generated during simulation execution flows from the simulation engine to the SBT through the following pipeline (see **Figure 19**):

SBT ->gRPC server -> gRPC datasource -> Simulation engine -> simulated robot controller

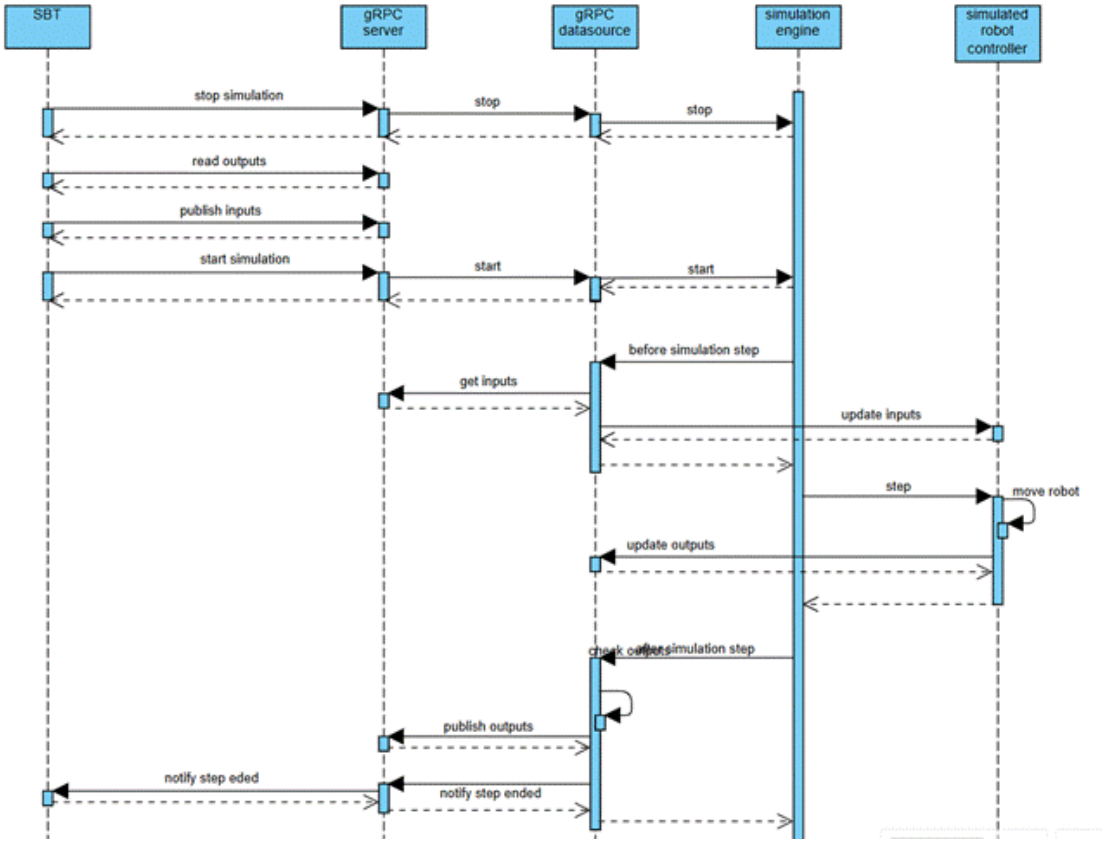


Figure 19: UML sequence diagram of the interaction between the SBT and some components of the DTS platform

Conversely, control commands generated by the SBT (e.g., start, pause, parameter updates) are transmitted through the gRPC datasource directly to the simulation engine.

To ensure data consistency and stability, the gRPC datasource is configured to expose two distinct topics for each shared variable as seen in **Figure 20**:

- A primary topic, which publishes the current value of the variable
- A shadow (override) topic, which allows the SBT to inject alternative values

When a value is provided through the override topic, it temporarily replaces the original simulation value. This mechanism enables controlled perturbation of the Digital Twin state while maintaining a stable and traceable data flow, ensuring robust testing conditions.

The integration between the HDT and the SBT enables advanced validation capabilities, including fault injection, scenario exploration, and system robustness assessment**.**

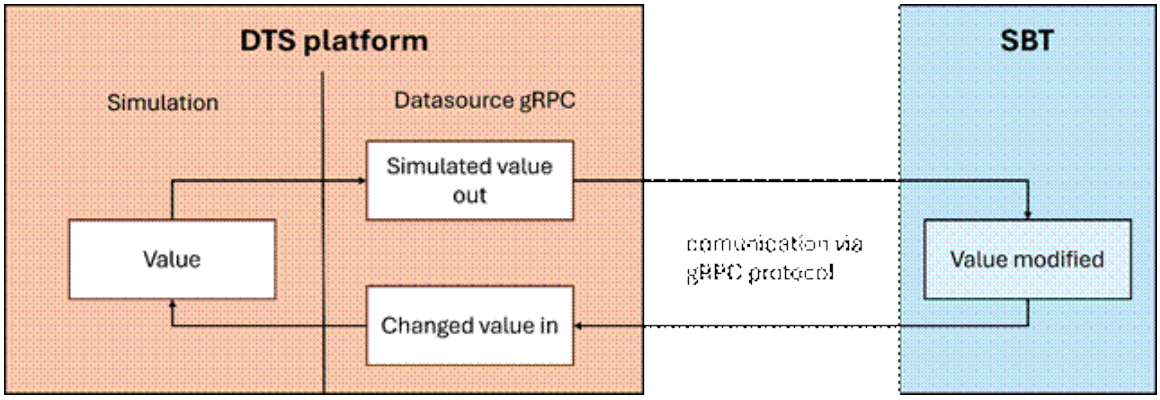


Figure 20: Communication between HDT and SBT

3.3.4. Integration between HDT and MAT

The MAT component is responsible for the analysis and the processing of the Building Information Modeling (BIM) file, which provides a structured and information-rich description of the operational environment and task targets. Within the SOPRANO framework, the BIM file represents a key input for configuring the Digital Twin and defining the context in which the system operates.

In particular, the BIM file contains:

- The building structure, describing the environment in which the robotic platform as well as workers will work (**Figure 21**)
- The milling target, including their type (e.g., sockets or ducts) and geometric properties (**Figure 22**)

This representation enables the simulation to accurately reflect the real-world context in which the physical entity (i.e., the robot and human operators) performs its tasks.

To enable interoperability with the HDT, the MAT component processes the BIM file and generates three distinct artifacts that can be directly interpreted by the simulation environment:

- A glTF file, used for 3D visualization of the environment
- A building configuration file, encoding the structural and spatial information required for simulation
- A task description file, defining the set of operations to be executed, including milling targets and task parameters.

These artifacts correspond to different aspects of the Digital Twin:

- The glTF file supports the visualization of the digital depresentation
- The building configuration file encodes the environmental model used for simulation and navigation
- The task description file defines the behavioural inputs driving the simulation

The generated files are provided as input to the HDT component, enabling automatic configuration of the simulation environment. This approach allows the same simulation

framework (including the Human Digital Twin and robotic models) to be reused across different buildings and operational scenarios with minimal user effort.

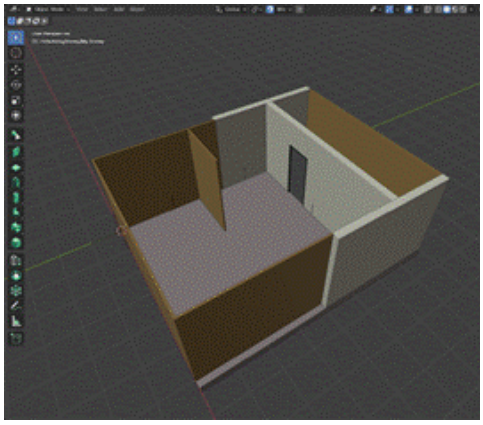


Figure 21: A BIM file as it appears in Blender

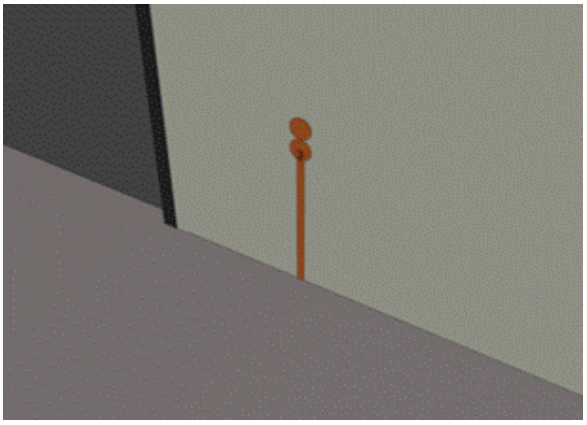


Figure 22: Details of the same structure in which is possible to see in orange planned sockets and ducts

3.3.5. Integration with the mobile platform

The integration between the mobile robotic platform (KUKA crawler) and the Digital Twin Service (DTS) platform is achieved through a communication mechanism conceptually similar to the one adopted for the integration with the SBT component, but implemented using an **OPC UA-based interface.**

In the real system, the KUKA PLC exposes data through an OPC UA server, which acts as the main communication interface with external components. This server provides access to control commands and status information, including those generated by the navigation stack to drive the crawler within the environment. These commands enable movement in a 2D space and allow dynamic path adaptation in the presence of obstacles.

The Digital Twin of the robot accesses the OPC UA server through a dedicated **OPC UA datasource**, which retrieves control signals and updates the Digital Twin accordingly. As a result, the simulated platform follows the same trajectory planned by the navigation stack, ensuring behavioural consistency between the physical entity and its digital representation.

The main control signals involved in crawler navigation include:

- Stop
- Forward
- Backward
- Right
- Left

When the crawler reaches a navigation target, the PLC triggers the execution of the milling operation. This is done by publishing a start milling command along with the coordinates of the target location via the OPC UA server (**Figure 23**). These coordinates are expressed

in the wall reference frame and, as in the real system, are transformed into the robot reference frame using data from distance sensors. These sensors are also used to compensate for positioning errors caused by navigation inaccuracies, ensuring precise alignment before milling.

Additional signals are exchanged to support synchronization between the KUKA PLC and the virtual mobile platform:

Between the KUKA PLC and the virtual mobile platform:

- Start milling
- Waiting for milling result
- Milling target x
- Milling target y

Between virtual mobile platform and PLC

- Milling started
- Milling finished

The simulated robot executes the milling process upon receiving the corresponding command and communicates its execution status back to the PLC through the OPC UA interface. This bidirectional communication ensures a closed-loop interaction between the physical entity and the digital twin.

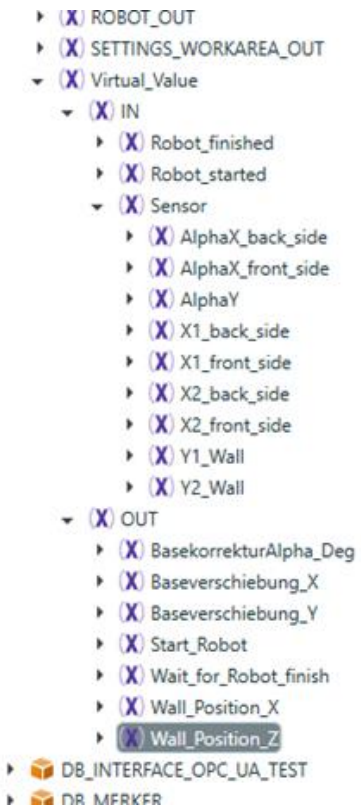


Figure 23: OPC-UA nodes on the KUKA OPC-UA server for communication with the DTS platform

To enable accurate navigation, the system requires continuous feedback on the platform's position and surrounding environment. In the real system, this function is performed by LiDAR sensors, which provide environmental perception and obstacle detection.

To replicate this behaviour, two **virtual LiDAR sensors** have been integrated into the Digital Twin of the mobile platform. These sensors replicate the characteristics of the real devices (Hokuyo UAM-05LP-T301 safety scanners), ensuring that the navigation stack receives data consistent with real-world conditions. The sensors are mounted on opposite corners of the platform and, with a field of view of 270°, provide full coverage of the surroundings.

The navigation stack communicates with the LiDAR sensors through **ROS2 topics**. To emulate this behaviour, a dedicated **ROS2 datasource** has been developed, capable of publishing simulated LiDAR data using the standard message type:

- sensor_msg/msg/LaserScan

This message includes both measurement data and sensor parameters, such as:

- The complete angle of the scan (270° for the Hokuyo scanners)
- The angular increment of each measurement that compose one scan (0.125°)
- The time difference between the start of a measurement and the start of the next measurement (1.04 μs)
- The time difference between the start of a scan and the start of the next scan (30 ms)
- The range of measurement (20 m)
- The measurement values

The timing parameters are particularly important, as they allow the navigation stack (Nav2) to estimate platform motion between consecutive scans (odometry estimation).

The integration workflow operates as follows (**Figure 24**):

1. The virtual LiDAR sensors generate environmental data within the Digital Twin
2. The ROS2 datasource publishes this data to the Nav2 stack
3. The Nav2 stack processes the data, computes navigation goals, and generates motion commands
4. These commands are published to the KUKA OPC UA server
5. The OPC UA datasource retrieves the commands and applies them to the simulated crawler
6. When a target is reached, the PLC coordinates the execution of the milling task through OPC UA communication

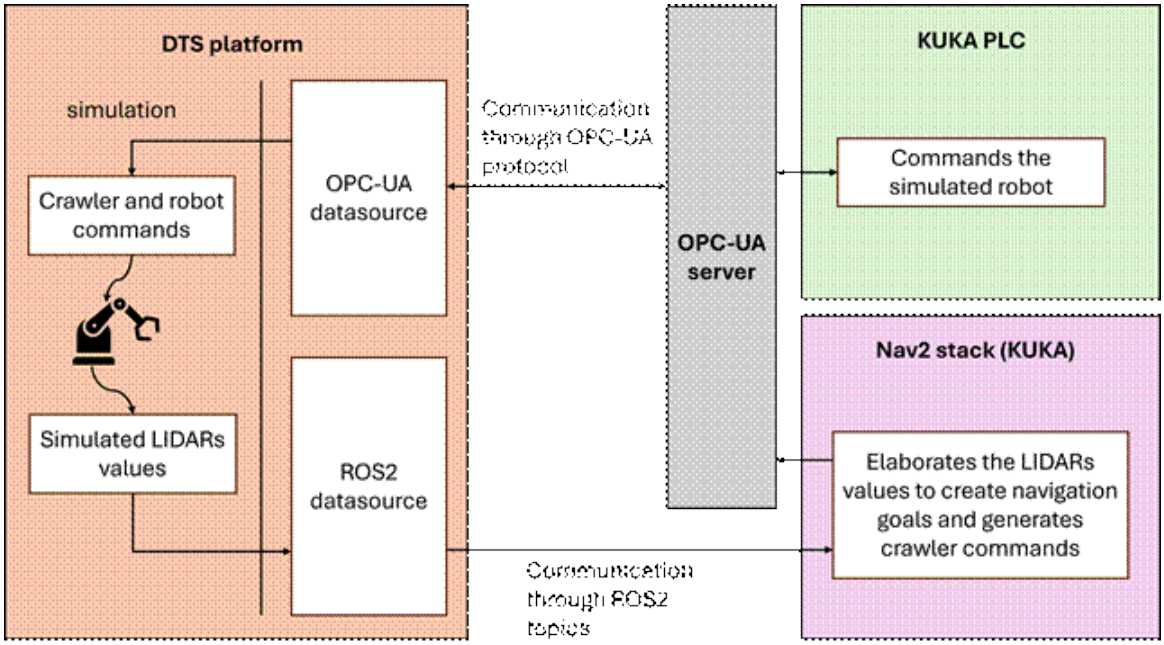


Figure 24: Data exchange within the DTS platform and the KUKA components

This architecture ensures tight integration between perception, navigation, and task execution, supporting a fully synchronized interaction between the Digital Twin and the real mobile platform. It enables real-time data exchange, behavioural consistency, and closed-loop control, providing a reliable framework for simulation, safety validation, and system optimization.

Future work

Future work will focus on finalizing the integration between the HDT, the MAT component and the SBT component, and on performing an end-to-end validation of the data flow across these modules. This validation will assess the correct synchronization between the Digital Twin, the Digital Twin Services (e.g., SBT), and the data derived from BIM processing (MAT), ensuring a coherent and reliable digital representation of the system and its environment.

Additionally, the integration between HDT and the real KUKA mobile platform will be tested and validate through the following experimental scenarios:

- The **target navigation test**, as explained in the Deliverable D1.2 named “Evaluation plan”, scenario 2 of the residential building renovating use case. The virtual robot, driven by the robotic middleware and the KUKA PLC is required to autonomously reach a predefined target position within the residential building environment while avoiding collisions and respecting the defined tolerance thresholds.
- The **obstacle detection test**: as explained in the Deliverable D1.2 named “Evaluation plan”, scenario 3 of the residential building renovating use case. The virtual robot, controlled by the robotic middleware and the KUKA PLC, must avoid both static

obstacles and workers, leveraging the safety mechanisms implemented in the real system.

- The **task execution test**: the virtual robot verifies the reachability of all targets points and executes all the planned milling operation accordingly. This test enables the evaluation of the reachability of milling points, the processing and cycle time estimation and the correctness of the task execution.

These validation activities will support the assessment of the closed-loop interaction between the Digital Twin and the physical entity, ensuring alignment between simulated and real-world behaviour. Furthermore, they will contribute to verifying the robustness, reliability, and accuracy of the overall system, supporting its readiness for real-world deployment.

3.4. APPLICATION ON SYMBIOTIC FARM USE CASE

3.4.1. Overview of symbiotic farm use case

The E-TERRY case study consists of one or more mobile robotic platforms which can assist with a variety of tasks within an agricultural robotics scenario. The E-TERRY use case seeks to enhance the efficiency, reliability, and safety of multi-human multi-robot (MH-MR) farming systems. SOPRANO will help achieve smoother coordination between humans and robots, ensuring seamless handovers during harvesting and transport.

Key desired improvements include more robust navigation in dynamic outdoor environments, better handling of uncertain terrain and weather conditions, and stronger safety and trust mechanisms through model-based dependability and runtime monitoring. SOPRANO should also reduce configuration time by providing intuitive interfaces and adaptive task allocation, allowing the robot to quickly switch between functions such as transport, weeding, and monitoring. Overall, these improvements will make autonomous farming systems more efficient, flexible, and trustworthy for real agricultural operations.

E-TERRY, functioning as an autonomous robot, offers versatile control modes including teleoperation (remote control). This adaptability allows for seamless transitions between autonomous and manual control modes, catering to diverse operational needs. Serving as a versatile carrier system, E-TERRY fulfils multiple tasks, including but not limited to spraying, monitoring, weeding, and transporting. The continuous data logging feature provides real-time monitoring, enabling operators to track the robot's status and performance in the field. The E-TERRY robot may be controlled by direct physical, visual, and interface-based communication (e.g., stop buttons, GUI, joystick, LEDs, HMI), and ultimately by gesture-based interaction. It also supports spatial coordination in which humans and the robot share the same workspace but maintain safe zones. This is enabled by a LIDAR and control unit which triggers the robot to slow down and ultimately stop in order to maintain separation distances from human workers.

The E-TERRY robot can operate in several different modes, corresponding to distinct scenarios as discussed in SOPRANO D5.3. The main mode of movement for the E-TERRY as a transport solution is the Harvesting Mode. Here the robot moves as a stop-and-go system, following behind the field workers. Harvesting mode is activated until the loading area under the E-TERRY becomes full. When the loading platform is full, the unloading

mode is activated. The E-TERRY drives out of the field in a row and drives autonomously to the loading point where all the full boxes are collected. Here the E-TERRY stops until all the boxes have been unloaded. After the field worker at the loading point gives the E-TERRY the signal to move on, the E-TERRY moves back to the same row as before to the field worker's position.

In this section, we consider the application of the WP3 technologies, specifically the DSA and RSA tools and the SBT component to the E-TERRY use case as described during simulation. We use a specific simulation scenario implementation presented in **Figure 25**. This scenario shows the E-TERRY robot together with two human workers, and three rows of crops. In this demonstration evaluation for the deliverable, we investigate the capability of the robot to safely travel out into the field and recover boxes of produce from two human workers at fixed locations (highlighted in red). Further details of the E-TERRY use case are provided in SOPRANO deliverables D1.2 and D5.3.

In this scenario, the agricultural robot follows a lawnmower pattern, moving eastward along the first row of crops, turning at the row's end, shifting slightly south, and then proceeding westward along the second row. The first worker (circled in blue) is mobile, walking eastward along the first row. The robot follows behind this worker, transporting collected fruits to the base station. As the worker may periodically stop to harvest crops, the robot must adapt to such spontaneous interruptions by slowing down or stopping to avoid creating safety hazards. The robot is equipped with a laser scanner for perceiving surrounding objects and a collision detection module that processes these sensor inputs to detect and react to obstructions, including workers.

Also, two further workers (circled in red) are stationed near the lowest row of crops performing weeding tasks. Their positions may deviate northward from the expected locations (as indicated by the dashed red lines), potentially intersecting with the robot's trajectory. In this situation, the robot must reliably detect worker positions and ensure that no unsafe close approach occurs.

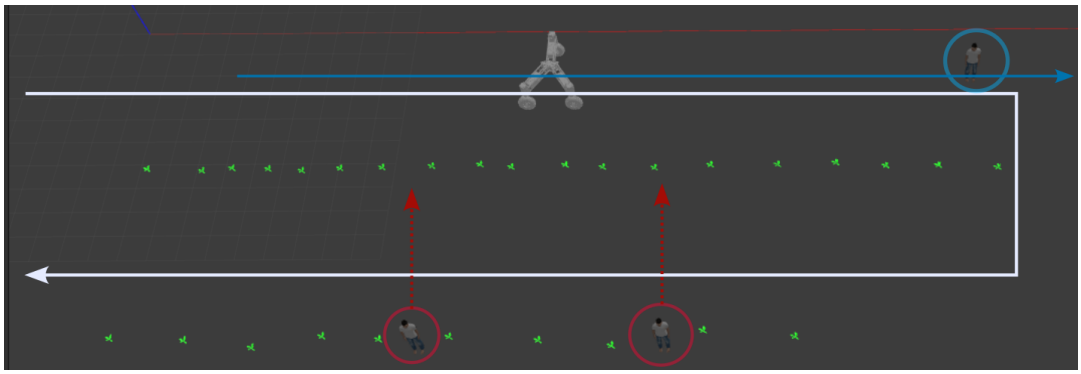


Figure 25: An example of the E-TERRY use case for evaluation when transporting produce boxes from the field to a central location

3.4.2. Safety Analysis Tools Demonstration

3.4.2.1. Use case overview and safety objectives

The proposed DSA/RSA methodology has been applied to the E-TERRY use case, which considers the deployment of an autonomous robotic platform supporting harvesting

activities in agricultural environments. This scenario represents a typical MH-MR operational context characterised by dynamic interactions between mobile robots, field workers and changing environmental conditions.

The analysed mission consists of a sequence of operational modes defining the robot behaviour throughout task execution. The overall mission workflow is illustrated in **Figure 26**.

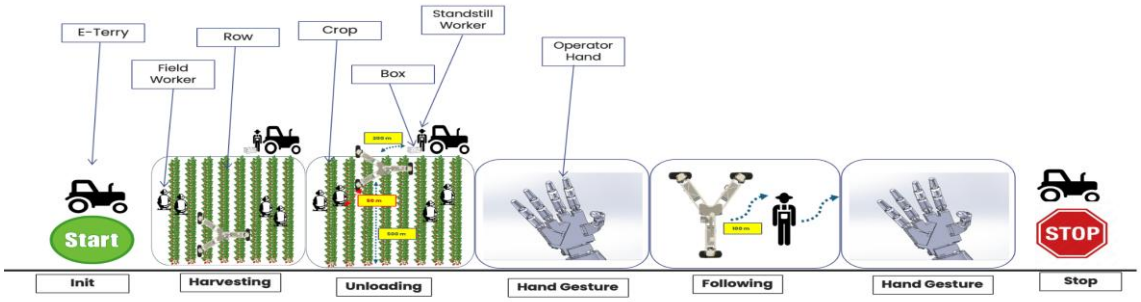


Figure 26: E-Terry mission workflow

The main operational phases are summarised:

- **Initialization mode:** The robot is activated and transitions from an idle state to operational readiness. This phase includes system checks and preparation of localisation, perception and control modules.
- **Harvesting mode:** This mode represents the primary operational phase (approximately 70 % of mission time). The robot navigates along crop rows while assisting workers during harvesting tasks. Safe operation requires:
  - controlled robot speed
  - accurate localisation
  - continuous perception of workers and environmental obstacles
- **Unloading mode:** The robot moves towards designated unloading zones where harvested products are transferred to storage containers. Compared to harvesting mode, the robot may operate at higher speed, which increases safety constraints related to motion supervision and environment awareness.
- **Hand gesture recognition mode:** In this interaction-oriented phase, the robot remains stationary and interprets commands provided by human operators. This mode supports manual task coordination and flexible human-robot collaboration.
- **Following mode:** The robot autonomously follows a designated worker while maintaining a predefined safety distance. Key safety challenges include:
  - correct identification and tracking of the target worker
  - preservation of safe separation distance
  - compliance with operational area boundaries
- **Stop mode:** This mode may correspond either to mission completion or to safety-related halt conditions triggered by supervision mechanisms.

The diversity of operational modes makes this use case particularly relevant for evaluating the proposed safety analysis framework. In particular, the scenario illustrates:

- The need to analyse safety risks associated with human-robot interaction in outdoor environments.
- The importance of defining operational assumptions during system design
- The role of runtime monitoring mechanisms in detecting and mitigating safety-critical situations.

This use case therefore provides a representative application context for demonstrating how DSA results can be transformed into runtime safety supervision capabilities.

3.4.2.2. Design-time safety analysis results

ODD Specification

The safety analysis relies on the explicit specification of the ODD, which defines the conditions under which the robot is expected to operate safely in the agricultural environment. An initial version of the ODD was established in collaboration with the use case partners. The specification focused on identifying key operational factors related to:

- environmental conditions of outdoor farming environments,
- structural characteristics of the working area,
- dynamic entities, including field workers and agricultural equipment.

The resulting operational domain was formalised within the SOPHIA safety engineering framework using the Papyrus4ODD modelling environment. In this context, Papyrus4ODD is integrated as a dedicated modelling plugin within SOPHIA, enabling engineers to specify operating conditions in a structured and model-based manner. Each element of the ODD is explicitly represented and can be directly exploited for subsequent hazard identification and risk analysis steps.

To ensure the consistency of the ODD specification, validation mechanisms are applied to detect conflicting or inconsistent operating conditions. These checks contribute to improving the safety analysis by ensuring that risk assessment is performed on a coherent and well-defined ODD. An overview of the ODD model developed for the E-Terry use case is illustrated in **Figure 27**, which presents the textual specification of the operational conditions. In addition, the hierarchical organisation of the ODD elements is also illustrated, highlighting the decomposition of environmental and dynamic factors considered during the analysis.

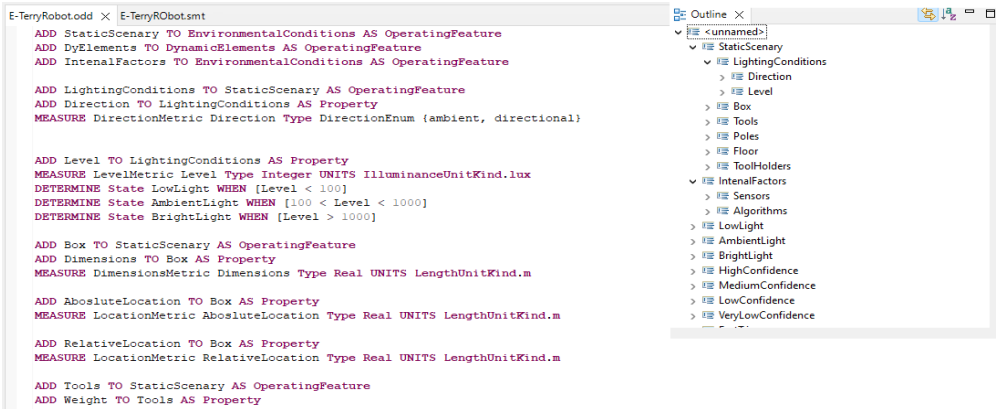


Figure 27: The E-Terry OpenODD model in Papyrus4ODD.

Hazard identification and risk analysis

Based on the defined ODD, a structured safety analysis was carried out in order to identify hazardous situations that may affect the robot during agricultural operations. This analysis was conducted within the SOPHIA framework using the dedicated DSA plugin, which supports a systematic and model-based exploration of safety-critical situations.

From an implementation perspective, this analysis is supported by a dedicated DSA plugin integrated within SOPHIA (**Figure 28**). This plugin provides interactive tabular views that allow engineers to construct and analyse safety scenarios in a structured manner. The tables are automatically organised based on the combination of ODD elements and system functional chains, ensuring consistency between operational conditions, system behaviour and identified risks.

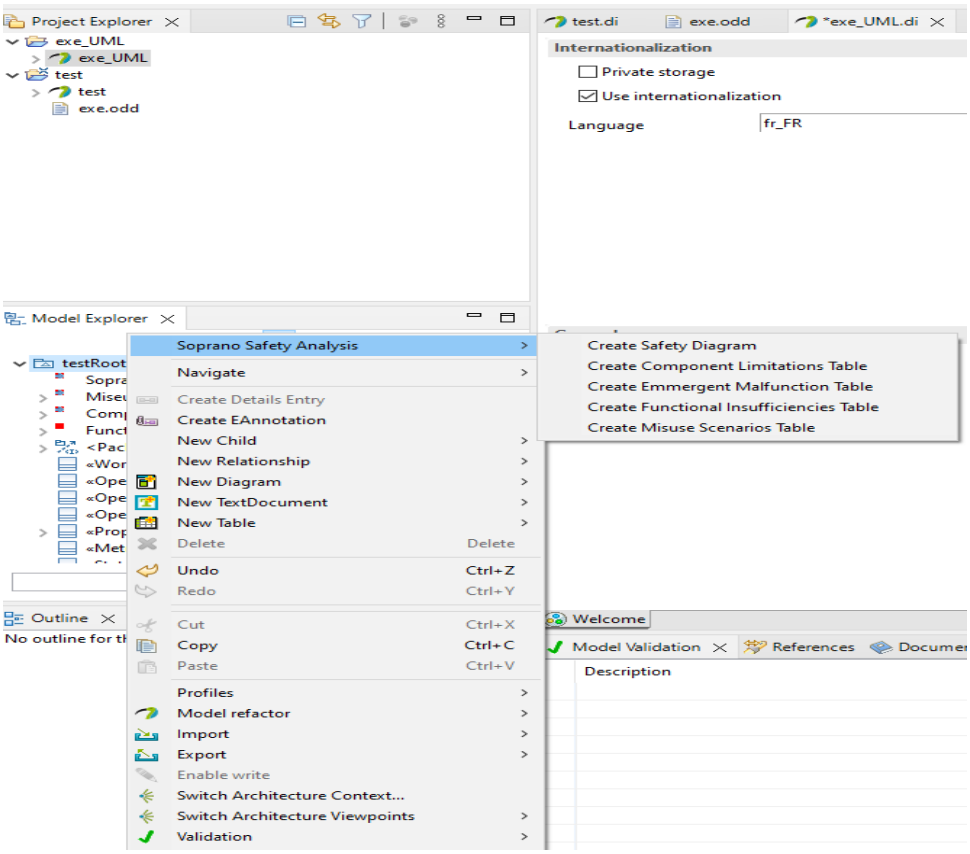


Figure 28: Sophia DSA plugin

In addition to tabular representations, SOPHIA leverages SysML/UML annotations to maintain traceability between system architecture elements and safety analysis artefacts. Engineers can associate causes, deviations, consequences and mitigation strategies directly with model elements through dedicated stereotypes and libraries. These libraries include reusable safety artefacts such as failure modes, hazard types and mitigation patterns, facilitating consistency and reuse across different analyses. This integrated approach enables both graphical and tabular views of the safety analysis, allowing engineers to navigate between system architecture models and safety artefacts within a unified modelling environment.

The analysis covered several mission modes, including worker following, harvesting, unloading and hand-gesture interaction, under different environmental and operational conditions. For each situation, the study established explicit traceability between the considered operating condition, the impacted system function, the acquisition and processing chain involved, the identified deviation, the resulting hazardous scenario, the expected consequence and the associated risk classification. In addition, the analysis was extended with mitigation strategies, fallback measures and candidate safety rules intended to support both design-time risk reduction and future runtime supervision.

An extract of the resulting analysis table is presented in **Figure 29**. This extract illustrates how a single operational factor may affect multiple system functions and lead to different levels of risk depending on the mission context. It also shows how the safety analysis goes beyond hazard identification by structuring mitigation measures and explicitly capturing safety constraints. The analysed cases show that environmental degradations influence perception, localisation, decision and interaction functions in different ways. The proposed analysis therefore provides a structured basis for identifying not only hazardous situations, but also the design assumptions and fallback strategies required to maintain safe behaviour.

An important outcome of this work is the derivation of safety rules expressed at system level. These rules capture the safety constraints that should remain satisfied during operation, for example in relation to human detection confidence, landmark reliability, object visibility or sensor validity under degraded environmental conditions. They constitute a key bridge between the design-time analysis and the runtime monitoring mechanisms presented in the following subsection.

Derivation of runtime safety properties

Based on the identified hazards, a subset of safety-critical scenarios was selected for further refinement towards runtime monitoring. This refinement aimed at identifying measurable indicators that can be used to supervise robot behaviour during operation.

In particular, navigation risks observed during harvesting activities were associated with trajectory deviation indicators and localisation performance metrics. These parameters were selected because they can be directly obtained from the robot perception and control subsystems through ROS2 interfaces, enabling their use in runtime supervision.

Within the SOPHIA environment, this process is facilitated by the structured organisation of safety analysis tables, which allow engineers to identify relevant runtime variables and monitoring indicators directly from the analysed scenarios. The resulting monitoring logic is therefore not defined in isolation but grounded in the formalised safety analysis artefacts.

The derivation of executable monitoring rules follows a semi-automatic process within the SOPHIA framework. While the safety analysis workflow supports the structured identification of hazardous situations and mitigation strategies, the definition of runtime monitoring conditions still requires engineering judgement. This includes the selection of observable variables, the definition of operational thresholds and the specification of mitigation responses compatible with the robot control architecture.

Operating Conditions	In ODD	Function	Acquisition (HW)	Interpretation / Processing	Decision	Keyword	Deviated Function
1 < ILLUMINANCE LEVEL < 200	YES	Worker Following	RGB / Stereo Camera	Human Pose Estimation (HMO)	Distance Control	Shadow blur	Partial human detection
1 < ILLUMINANCE LEVEL < 200	YES	Worker Following	Stereo Camera	Landmark Matching (VLM)	Positioning	Drifted landmark	Recalage inexact
LIGHTING / SOURCE = OUT OF F	YES	Worker Following	RGB / Stereo camera	Human Pose Estimation (HMO)	Distance Control	Backlighting	Human silhouette lost
LIGHTING / SOURCE = OUT OF F	YES	Worker Following	Stereo camera	Landmark Matching (VLM)	Positioning	Landmark glare	Inaccurate recalage
100 < VISIBILITY < 500m	YES	Worker Following	Stereo / RGB camera	Human Pose Estimation (HMO)	Distance Control	Low contrast	Partial detection
100 < VISIBILITY < 500m	YES	Hand Gesture Control	RGB camera	Gesture Detection (HMO)	Gesture Interpretation	Visual blur	Gesture missed
Temp < -25°C	YES	Worker Following	Camera (ZED X)	Human Detection (HMO)	Distance Control	Cold noise	Degraded image quality
Temp < -25°C	YES	Hand Gesture Contr	Camera (ZED X)	Gesture Interpretation (H	Gesture Decision	Frame loss	Unstable video feed

Hazardous Scenario	Consequence	Risk Classification	Mitigations		
			Design	Assumptions	Fallback
Worker partially occluded, intermittent tracking	Unsafe following, robot stops abruptly	MEDIUM	Use temporal smoothing to maintain tracking confidence despite transient occlusions. Adjust	Assume the worker remains in well-lit zones o	If human detection becomes intern
Robot misaligns with path due to bad anchor	Unsafe navigation near human	HIGH	Improve the robustness of landmark matching under low-light conditions by integrating confid	Assume the environment contains enough dist	If landmark confidence drops belov
Human detected only partially --> intermittent tracking	Unsafe following / loss of target	HIGH	Integrate backlight-aware detection strategies (ex: silhouette enhancement or infrared fallbac	Assume the human operator remains in adequ	If pose confidence drops below def
Landmark direction poorly lit --> poor matching	Robot deviates from worker's path	MEDIUM	Implement landmark validation and glare detection mechanisms. Use alternative visual cues o	Assume landmarks used for positioning are vis	If landmark confidence or positioni
Worker partially visible --> inconsistent following	Unsafe path adjustment	MEDIUM	Integrate contrast enhancement and temporal smoothing in human tracking pipelines to ensur	Assume the visibility remains above 100 m and	If human pose tracking becomes int
Dust reduces visibility of hand movement	Command not interpreted	MEDIUM	Apply noise filtering to improve gesture recognition under reduced visibility. Include confiden	Assume that hand gestures are performed with	If gesture confidence is persistentl
Frozen lens / blur --> worker no	Unsafe following or wrong	HIGH	Implement thermal diagnostics for sensor operability and trigger warnings or degraded mode	Assume the camera and its housing are qualifie	If human is not detected and camer
Latency or dropped frames pre	Misinterpretation / wrong	HIGH	Use temporal filtering or gesture buffering to tolerate minor frame loss. Design gesture decisic	Assume camera performance remains within v	If gesture input becomes unreliabl

Standards	Safety rules
ISO 13482 ISO 15066	The robot shall maintain continuous and reliable human tracking under ambient illumination between 1 and 200 lux, by dynamically adapting perception parameters
ISO 13482 ISO 10218	The robot shall continuously assess the confidence of landmark-based positioning, and degrade gracefully under low-light conditions to avoid unsafe trajectory deviation near humans. If landmark confidence is insufficient or drift becomes excessive, the robot shall decelerate and switch to a predefined
IEC TS 62998 ISO 13482	The robot shall reduce speed or pause if the human silhouette becomes unreliable due to lighting sources located behind the subject.
ISO 10218 ISO 13482	The system shall monitor landmark visibility and avoid recalage based on landmarks affected by glare or poor illumination. If landmark positioning fails due to glare, the robot shall decelerate and initiate a safe navigation fallback until reliable positioning is restored.
ISO 13482 ISO 10218	The system shall enhance contrast and stabilize human pose detection under degraded visibility between 100 m and 500 m. If tracking confidence drops below reliable threshold, the robot shall decelerate and alert the operator until tracking is recovered.
IEC TS 62998-1 ISO 13482	The system shall monitor visibility and adapt gesture interpretation thresholds accordingly between 100 m and 500 m. If gesture recognition fails due to low visibility, the system shall switch to an alternative interaction mode or prompt the user.
ISO 10218-1 ISO 13482	The robot shall continuously monitor camera health and image quality during low-temperature operation and switch to alternative detection if degradation is detected. If human detection fails due to camera freezing or cold blur, the robot shall reduce speed or halt until safe tracking is
ISO 13482 ISO 21448	The robot shall monitor video feed stability and disable gesture-based control if latency or frame drops exceed defined reliability thresholds. If gestures cannot be reliably interpreted due to unstable video, the robot shall switch to a verified fallback control

Figure 29: Extract of hazard analysis results for the E-Terry use case

An example of a runtime monitoring rule derived from the analysis of trajectory deviation is presented below. This rule illustrates how safety constraints identified during design-time analysis can be translated into structured monitoring configurations that are later deployed within the runtime safety supervision framework.

```
{
  "rule_id": "cte_deviation_monitoring",
  "input_topic": "/localisation/cross_track_error",
  "thresholds": {
    "safe": 0.04,
    "warning": 0.07
  },
  "logic": {
    "SAFE": "cte <= safe",
```

```
    "WARNING": "cte > safe and cte <= warning",
    "CRITICAL": "cte > warning"
  },
  "actions": {
    "WARNING": "activate_safe_controller",
    "CRITICAL": "override_motion_and_stop"
  }
}
```

In parallel with rule preparation, the same design-time artefacts can also support the generation of simulation scenarios used for validation activities.

Complementary analysis: FMECA on LiDAR-based safety function

In addition to the scenario-based safety analysis presented above, a complementary Failure Modes, Effects and Criticality Analysis (FMECA) was carried out on the LiDAR-based safety function. This complementary analysis was performed within the SOPHIA framework, which already provides dedicated support for classical safety analyses (FTA for example).

The motivation for this additional analysis is twofold. First, in the E-TERRY use case, several safety-relevant decisions depend directly on perception and obstacle detection capabilities. Second, unlike traditional industrial architectures relying on a dedicated safety PLC, the considered robotic platform mainly depends on sensing, software processing and runtime supervision to maintain safe behaviour. In such a context, analysing hazardous scenarios only from an operational and functional perspective is not always sufficient. It is also necessary to examine possible component-level failure modes that may degrade or invalidate the safety function itself.

For this reason, a focused FMECA was applied to the LiDAR chain used for obstacle perception and safety-related motion adaptation. The objective is not to replace the DSA, but to complement it by identifying plausible sensor-level faults, their local and system-level effects, and their possible impact on the robot behaviour. This analysis is particularly useful for linking classical dependability concerns with the more operational DSA/RSA workflow. In practice, it helps identify which LiDAR degradations should be monitored at runtime and which failure situations may justify additional safety rules, degraded modes or fallback strategies.

Furthermore, the FMECA is not performed independently from the DSA results. Within SOPHIA, explicit traceability links can be established between scenario-based hazards identified in the DSA tables and component-level failure modes analysed in the FMECA. This allows engineers to relate functional insufficiencies observed at the system level to underlying component degradations, and to ensure consistency between operational risk analysis and classical dependability assessments. An extract of the resulting FMECA is presented in **Table 2**. It illustrates typical failure modes affecting the LiDAR-based safety function.

Table 2: FMECA of LiDAR component

Failure Mode ID	Description	System Effect	Severity	Criticality	Risk class	Mitigations	Residual risk class
FM-01 Loss of signal	Complete absence of LiDAR output (no scan data)	Robot cannot detect obstacles → potential unsafe motion or inability to react	4	4	High	Signal monitoring, watchdog, fallback stop, redundancy (if available)	Medium
FM-02 Signal degradation	Reduced signal quality due to dust, rain or particles	Partial obstacle detection → reduced perception reliability	3	4	High	Confidence estimation, degraded mode, reduced speed, maintenance	Medium
FM-03 Measurement noise	Noisy or unstable distance measurements	Incorrect obstacle position estimation → unstable behaviour	3	3	Medium	Filtering, temporal smoothing, consistency checks	Low
FM-04 Delay / latency	Late delivery of LiDAR data	Outdated perception → delayed reaction to obstacle	4	3	High	Timestamp validation, freshness threshold, degraded mode	Medium
FM-05 Calibration drift	Progressive error in distance estimation	Incorrect safety distance → unsafe proximity or conservative behaviour	4	3	Medium	Calibration procedures, plausibility checks, periodic validation	Low

This complementary FMECA shows that several LiDAR failure modes may affect the robot's safety-related behaviour, either by masking real hazards or by producing unnecessary emergency reactions. It therefore provides a useful complement to the scenario-based DSA results presented earlier. In particular, it helps identify which component degradations should be explicitly monitored at runtime and which failure situations can be linked to degraded or emergency safety states.

3.4.2.3. Runtime monitoring deployment

The RSA module is executed in parallel with the robot control architecture and does not interfere with navigation or task execution components. Instead, it acts as a supervisory layer that evaluates safety conditions and publishes safety signals. It is deployed alongside the simulation environment based on Gazebo and RViz. Both the simulation and the monitoring module share the same ROS2 communication layer, allowing the RSA to subscribe to data streams produced by the system.

The monitoring logic is configured through external JSON files containing the safety rules derived from the DSA. These rules are loaded at runtime and automatically translated into executable conditions, allowing flexible adaptation of the monitoring without modifying the source code.

The RSA module subscribes to relevant ROS2 topics corresponding to perception, localisation and control information, and continuously evaluates these inputs against the defined safety constraints. The outputs of the monitoring process are published through dedicated ROS2 topics, including the global safety status, emergency stop requests and safety events.

This deployment approach ensures a clear separation between design-time safety modelling and runtime execution, while maintaining traceability between both stages. It also enables reproducible experimental setups, particularly when using containerised environments such as Docker.

3.4.2.4. Experimental scenarios and observations

To evaluate the behaviour of the RSA module, a set of representative experimental scenarios was defined and executed within the simulation environment. These scenarios aim to assess how the RSA module reacts under different operational conditions, ranging from nominal operation to critical safety situations.

The experiments were conducted using the Gazebo simulation environment combined with RViz visualisation, provided through the SBT framework, where the robot navigation and interaction with its environment can be observed. The RSA module was executed in parallel and monitored runtime data through ROS2 communication, enabling real-time evaluation of safety conditions.

The selected scenarios are designed to illustrate the progressive behaviour of the monitoring system, including nominal operation, degraded conditions, critical situations requiring emergency stop, and robustness to missing or inconsistent data.

To support this experimental evaluation, the E-TERRY use case relied on several capabilities implemented within SOPHIA. The ODD was specified using the integrated Papyrus4ODD plugin, the hazard analysis was performed with the dedicated DSA plugin, and representative simulation scenarios were generated from the ODD model and exported in OpenSCENARIO format (**Figure 30**). The runtime monitoring rules evaluated by RSA were then prepared from these design-time artefacts and encoded into JSON configuration files, ensuring continuity between modelling, safety analysis and runtime supervision.

```
<Storyboard>
  <Init>
    <Actions>
      <Private entityRef="robot">
        <PrivateAction>
          <LongitudinalAction>
            <SpeedAction>
              <SpeedActionDynamics dynamicsShape="step" value="0"/>
              <SpeedActionTarget>
                <AbsoluteTargetSpeed value="1.0"/>
              </SpeedActionTarget>
            </SpeedAction>
          </LongitudinalAction>
        </PrivateAction>
      </Private>
    </Actions>
  </Init>
  <Story name="LowLightAbort">
    <Act name="LowLight">
...
...
  </Story>
</Storyboard>
```

Figure 30: Example of a generated scenario

The experimental evaluation focuses on a representative safety scenario identified during the design-time analysis, combining human proximity and degraded perception reliability. This situation reflects a critical condition where the robot operates close to a human while the confidence of detection is insufficient to ensure safe interaction. A corresponding monitoring rule was derived by combining distance and perception accuracy indicators, allowing the RSA module to detect hazardous situations based on the conjunction of these conditions rather than isolated thresholds.

Baseline scenario (nominal behaviour)

In the baseline scenario, the robot operates under nominal conditions with no safety constraint violations. The navigation system follows planned trajectories, and all perception and control signals remain within their expected operational ranges.

The RSA module continuously monitors runtime inputs but does not detect any violation of safety rules. As a result, the system remains in a nominal safety state, and no warning or emergency events are generated. This scenario serves as a reference to validate that the monitoring framework does not introduce unnecessary interventions when the system operates correctly.

Degraded scenario

A representative degraded situation was reproduced by reducing the perception reliability while maintaining a safe distance between the robot and the human. This configuration corresponds to a non-critical but safety-relevant condition identified during the design-time analysis.

During execution, the perception module provides continuous estimates of human detection quality and relative distance. These indicators are monitored by the RSA module. When the detection confidence drops below the validated threshold while the human remains outside the critical proximity zone, the monitoring logic identifies a degradation of perception without triggering a safety violation. In this situation, the RSA module

transitions from a nominal to a degraded safety state and generates a warning event. This behaviour reflects a conservative supervision strategy, where uncertainty is explicitly taken into account without unnecessarily stopping the robot.

The system response can be observed through the runtime outputs, where the global safety state switches to a degraded mode. At the same time, the robot continues its motion with adapted behaviour, typically characterised by reduced speed or increased caution in navigation.

The experimental observation is illustrated in **Figure 31**, which combines the simulation view and the monitoring outputs. The figure shows that, despite degraded perception conditions, the system maintains operation while explicitly signalling the reduced safety margin through the RSA outputs.

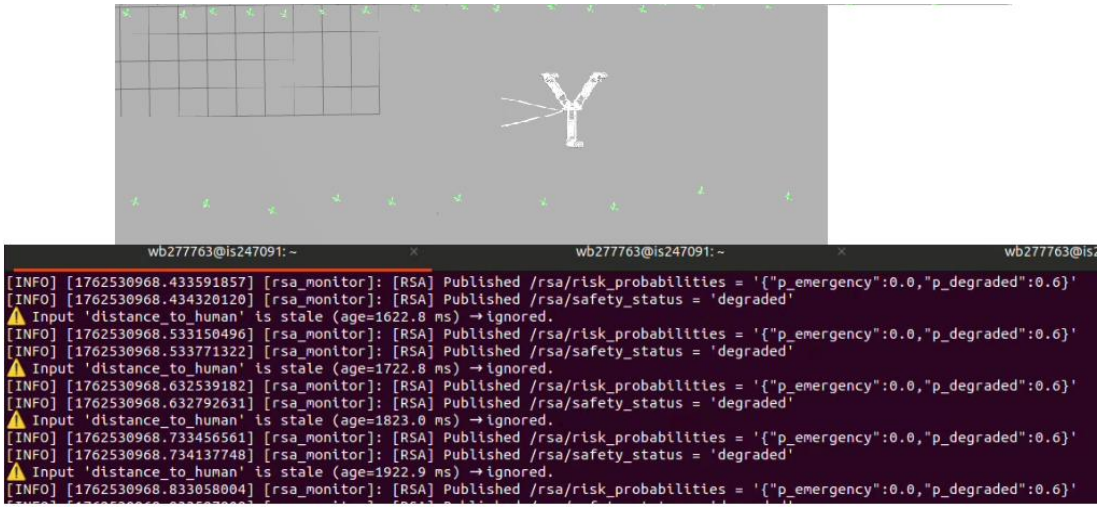


Figure 31: Degraded safety state without stop request

Emergency scenario

A critical situation was reproduced by bringing the robot into close proximity with a human under degraded perception conditions. This configuration corresponds to the unsafe combination identified during the design-time analysis. In contrast with the degraded case, the monitoring logic now detects a violation of the safety constraints due to the conjunction of insufficient perception reliability and reduced separation distance. This leads to an immediate transition to the emergency safety state. The RSA module reacts by issuing a stop request, which results in the interruption of the robot motion in the simulation. This behaviour illustrates the fail-safe response enforced when safety conditions are no longer satisfied.

The experimental observation is illustrated in **Figure 32**, where the robot is positioned in close proximity to a human. The RSA outputs confirm the activation of the emergency state and the publication of the stop request.

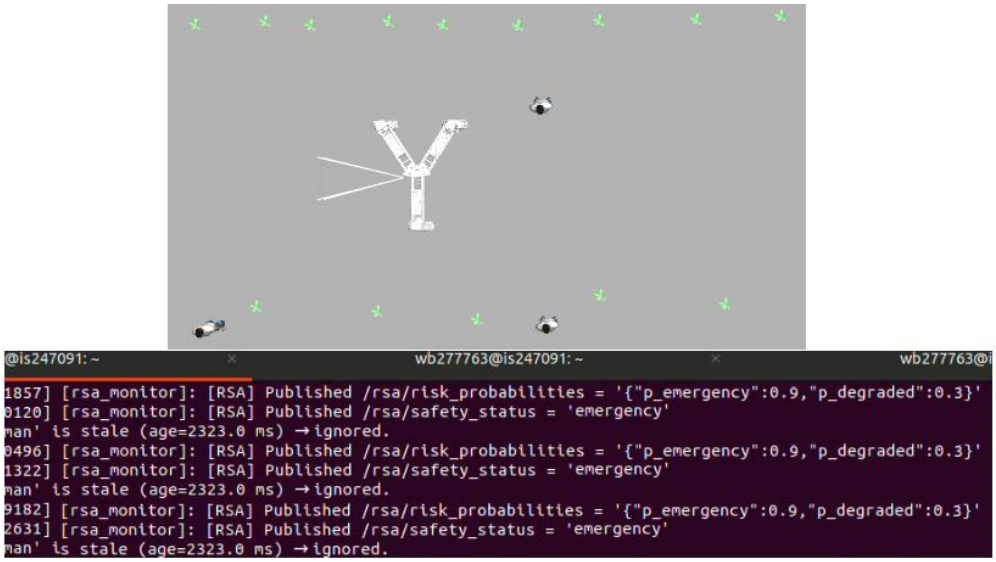


Figure 32: Emergency scenario leading to emergency state and stop request

Robustness evaluation

In addition to nominal, degraded and emergency conditions, the robustness of the monitoring framework was evaluated under incomplete or inconsistent input data. In such situations, certain runtime indicators may become unavailable, delayed or remain constant over time, preventing a reliable safety assessment.

Under these conditions, the RSA module does not assume nominal operation. Instead, it transitions to a degraded safety state, reflecting a conservative behaviour when the system cannot guarantee the validity of its inputs. This mechanism ensures that safety decisions are not based on unreliable data and highlights the importance of monitoring data freshness and consistency.

Beyond these controlled scenarios, the RSA module was also evaluated within the SBT loop. In this context, the emergency stop signal published by the RSA module is used as a quantitative metric to assess safety performance under a large number of automatically generated scenarios. Experiments performed within this framework showed that emergency situations can be effectively detected when critical conditions are met. In particular, combinations of human proximity and degraded perception were successfully identified by the monitoring logic, leading to the activation of emergency stop signals.

These results confirm that the RSA module is able to detect hazardous situations emerging from the interaction of multiple factors, rather than relying on isolated thresholds. They also demonstrate the relevance of integrating runtime monitoring within automated testing loops to explore a wider range of operational conditions.

However, the experiments also highlight several practical challenges. The behaviour of the monitoring system strongly depends on the calibration of thresholds and on the quality of the input data. In some cases, perception variability or limited dataset representativeness may lead to conservative reactions or delayed recovery.

At this stage, further work is required to improve threshold tuning, stabilise the transition behaviour between safety states and validate the monitoring logic under a broader set of environmental conditions. Future developments will also focus on strengthening the

integration between DSA and RSA, as well as on improving uncertainty management through probabilistic reasoning mechanisms.

3.4.3. Simulation-Based Testing Tools

Overview and Case Study Description

In this section, we consider the application of our SOPRANO SBT offline prediction techniques to the E-TERRY example. Further work during the project evaluation phase will consider its application to other use cases, testing the digital-twin interface and used for the implementation of the KUKA use case. Furthermore, future work during the project integration phase will also consider the interconnection of SORPANO components from other work packages during integration. The present E-TERRY system, which is subject to both operational faults and challenging environmental conditions, serves as a case study for investigating the offline prediction of robotic performance characteristics.

Beyond environmental variability, the robot may also experience internal faults and failures during mission execution. For example, if the collision detection monitor responds slowly due to processor overload, the robot may fail to react in time, resulting in unsafe proximity to workers. Similarly, the onboard laser scanner may suffer temporary degradation or interference, caused by communication failures or internal component faults, leading to missed detections of nearby humans. Such conditions highlight the necessity of robust predictive strategies to ensure safe and effective mission performance in dynamic, real-world agricultural environments.

The intent of this case study for the industrial partner (E-TERRY) is to achieve its functional requirements (recovering the boxes of produce in the minimal time possible), while also ensuring non-functional requirements related to safety. In particular, it is important to guarantee the safety requirement that robots must not too closely approach human workers during the loading process. Further it should be ensured that the robot completes its collection process and returns to the endpoint at the end of the mission. These requirements for safe operation of this scenario are specified in **Table 3**.

Table 3: E-TERRY safety requirements

R1	Robot must avoid collision hazards to human workers in the field as a result of close approach to them below a distance threshold (3m) R1 is quantified by two metrics: **Dynamic human distance** - tracks the minimal distance to the dynamic human **Static human distance** - tracks the minimal distance to either of the static humans
R2	Robot must complete its mission and return the boxes to the endpoint in the minimal time possible R2 is quantified by one metric: **Path completion metric** - tracks the proportion of the intended path that the robot completed

The manipulation operations introduced to the scenario are listed below:

- **Disruption of mobile human velocity:** supports unexpected human dynamic behaviour, e.g. stopping in front of the E-TERRY robot to deliver fruit or perform additional tasks in the field. This operation is of fixed intensity (either present or absent) but may have dynamic starting and ending times.
- **Disruption of static human positions:** supports unexpected human behaviour, e.g. standing in front of E-TERRY robot path instead of between crops. This may be of variable intensity, causing a position offset between 0 and 4m. Since this is a static variable modification, its effect is constant throughout the operation of the scenario.
- **Deletion/delay of point cloud messages:** models potential internal failure of E-TERRY component, such as transient point cloud generation/processing failure or poor performance leading to delays
- **Distortion of LIDAR point clouds**: models processing internal error in LIDAR such as misalignment of the LIDAR, noise in the point clouds, or disruption to the point cloud from beam loss in a specific angle range

Offline Predictor Analysis Predictor

According to the offline prediction methodology in Step 1 of **Figure 12**, several initial evolutionary experiments were performed and their results aggregated in order to generate both the initial training set and the validation test set for the predictors. For the E-TERRY use case, 1946 tests were generated. 15% of these tests are held back to validate the selectors, termed the selector validation set. On the remaining data, we perform a K-fold validation (K=5) with an 80:20 train-test split.

Then, we assess possible feature extraction strategies in Step 2, also considering their various hyperparameters. We use either TSFresh Windowed or MiniRocket, combined with Gradient Boosting or Ridge Regression for predictor training (Step 3). TSForest is an integrated approach, combining feature extraction with prediction. Together this results in a total of five combinations in total for feature extraction and learning.

The following hyperparameter ranges are evaluated for the various offline predictor feature extraction and learning algorithms. For TSFresh Windowed, window sizes of [10.0, 5.0, 2.0, 1.0, 0.5], while for MiniRocket, kernels counts of [2000,1000,500,100] are considered. For Gradient Boosting Regression trees, the number of estimators evaluated are [300,150,50,20], and for Ridge Regression, the maximum alpha values are set to [20,10,5]. Finally for TSForest, we investigate estimator counts of [2000,1000,500,100] and window sizes of [10.0, 5.0, 2.0, 1.0, 0.5].

Offline Predictor Results

Next, we present the E-TERRY scenario with three metrics with high-performing predictors obtained when scanning the defined range of hyperparameters for each prediction method. The **dynamic human distance metric**, modelled with TSFresh Windowed and Gradient Boosting (windowsize = 0.5s, n_estimators = 50), achieved an r2 score of 0.897, MSE of 1.14, and RMSE of 1.068 (**Figure 35**), performing well for common low and high-distance clusters but occasionally over-predicting intermediate distance cases. This may arise in cases in which the human stops but temporarily, leading

to an intermediate distance which is harder to predict, unlike in cases where the robot avoids the human entirely.

The **static human distance metric** using TSForest (n_estimators = 300, windowsize = 10.0s), achieved an r2 score of 0.894, MSE of 0.106, and RMSE of 0.326 and is presented in (**Figure 34**), with reliable performance across both low and high-distance clusters. Again, scenarios corresponding to intermediate distances had a somewhat higher variance.

Finally, for the **path completion metric**, analysed with TSFresh Windowed and Gradient Boosting (windowsize = 0.5, n_estimators = 150), achieved an r2 score of 0.839, MSE of 0.0035, and RMSE of 0.0596 (**Figure 33**). Clusters in the actual values (forming horizontal lines upon the plot) reflected robot stopping points, though lower values—associated with waiting for the dynamic human during scenarios in which it is stopped —were sometimes over-predicted, in scenarios in which the prediction failed to anticipate the dynamic human stopping. It is expected that enhancements covered in the further work (particularly online prediction feedback) could further improve prediction performance in these challenging cases.

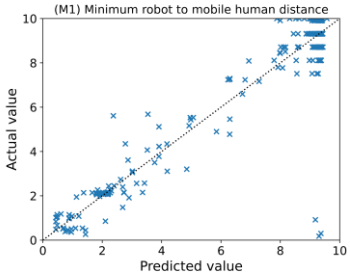


Figure 35: Dynamic distance human metric

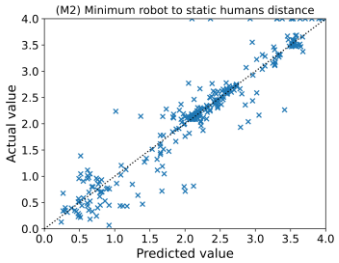


Figure 34: Static distance human metric

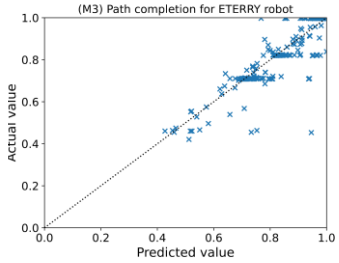


Figure 33: Path completion metric

SBT Performance prediction examples

Selectors:

To evaluate the test selectors defined in Section 3.2.5, we analyse the resulting Pareto fronts. The baseline is the execution of all tests in a node's testing set. The best selectors are those that yield the strongest front performance (i.e., highest overlap with the baseline front) while selecting the fewest tests, thus maximising time savings.

As outlined in Section 3.2.5.4, each selector has hyperparameters controlling its trade-off between performance and selectivity. For fixed-threshold nodes, metric counts of 1–3 are used. In the E-TERRY use case, thresholds are 3.0m for dynamic human distance, 2.0m for robot-static human distance, and 0.6m for path completion. For simulated annealing selectors (SA), the initial temperature T_0 is in the range [10, 1500], while for diversity-driven selectors, the coefficient LP lies in the range [0,1].

Figure 36 compares selectors with different parameters. The horizontal axis shows the proportion of validation tests selected by a selector, and the vertical axis shows the proportion of reference-front tests it found during performance testing. The optimal point for a selector is the cross indicated in the top left, which corresponds to perfect

performance; selecting tests producing a front identical to the reference front, while only executing the tests that are upon that front. The fixed-threshold node (C=2) performs best at low levels, showing that a naive approach of fixed threshold selection upon at least two metrics can work relatively well when low numbers of tests are desired for selection, although it only provides a greatly restricted front. The Diversity-Driven selector offers the widest spread over possible ranges of tests performed (0.3 — 0.65), and achieves a relatively good front, selecting at least 60% of the tests upon the front. SA-based selectors achieve the highest proportions of reference-front tests overall, but only when selecting larger test sets (above 80%).

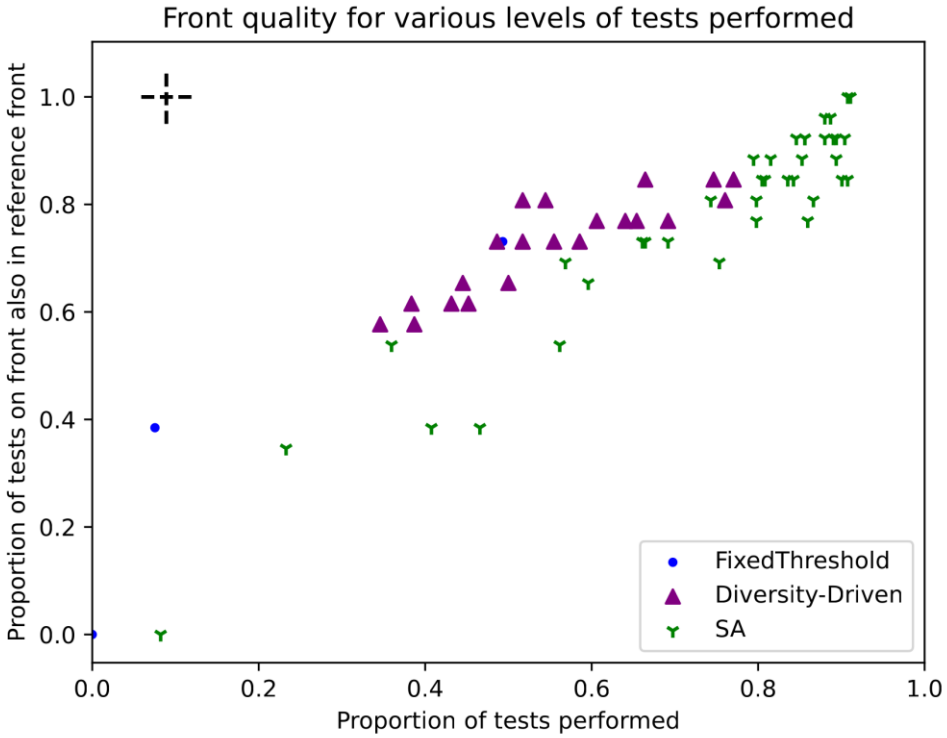


Figure 36: Comparison of different test selection selectors with different hyper-parameters

Testing the RSA component via the SBT platform

In order to demonstrate integration and verify proper behaviour, the RSA module and the SBT platform were interfaced together, in order to provide automated testing of the scenario involving the collision or very close approach of the E-TERRY robot with the mobile human. In this scenario, the RSA evaluates the following rule. An emergency is signalled on topic ***/rsa/emergency_status*** when this evaluates to true, and false is returned when it is false:

```
(/hmo/distance < 1.0m) AND (/hmo/worker_confidence < 0.6)
```

Until the HMO component is integrated, the ground truth distance from the robot to the human is measured from the SBT platform, and used as a mock input to the RSA. This value is communicated via metric feedback to the simulator as topic ***/hmo/distance***. The topic ***/hmo/worker_confidence*** can also be manipulated via the SBT platform via the

MITM mechanism, allowing it to vary the worker confidence level for every test. The SBT platform senses the output from the ***/rsa/emergency_status*** topic, and translates it into a metric. returning 1.0 for true and 0.0 for false.

RSA Component and Simulation Stability

The E-TERRY simulation is reliant upon the Gazebo simulator for physics and modelling of the underlying components, and the Gazebo implementation has a specific update rate at which its plugins refresh the simulation model_state information to ROS, such as the dynamic human position.

The RSA requires the input information for its safety rule computations to be delivered with a maximum latency, or they will be considered stale, and the inputs will be removed. For example, the ***/hmo/distance*** values need to be delivered with a maximum latency of 2000ms, and the ***/hmo/worker_confidence*** values need to be generated with a maximum latency of 3000ms. If either of these is considered stale, then the RSA will enter a degraded state, as reported upon the ROS topic ***/rsa/safety_status***.

In order to determine how the RSA degradation behaviour varies with the Gazebo update rate, the SBT platform was used to vary the configured update rate, using the static variable fuzzing mechanism to manipulate the appropriate configuration file. In turn a metric upon the SBT platform records the number of RSA messages indicating a degraded safety state are present in the system.

In **Figure 37**, it can be seen that Gazebo update rates less than 1Hz frequently generate RSA degradation states intermittently or permanently during the simulation, while they never occur at 2Hz or greater. In addition, testing at the initial update rate of 1Hz revealed that sometimes the robot may visually impact the human worker in the case of a LIDAR error (as seen upon the GUI) but the collision is sometimes not noticed by the RSA as a close approach less than 1.0m, because the metrics are irregularly sampled.

As a result, all the upcoming simulations in the following sections use a Gazebo update rate of 10Hz, and the metrics are sampled frequently, in order to ensure a suitably fresh rate of human metrics and no RSA degradation events. No impact upon simulation performance was noticed as a result of this configuration change. In addition, an updated RSA version (v2) was released in which it is possible to configure the timing to use simulated clock timing as well as wall-clock time. This is to ensure the variable freshness time used by the RSA can be consistent with the simulation settings, in cases in which the simulation runs at a different rate than the wall-clock time.

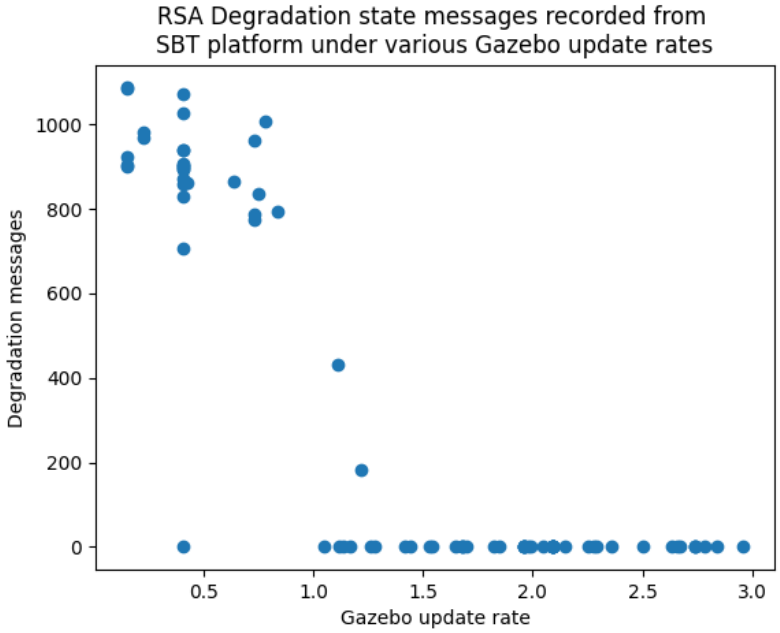


Figure 37: Recorded number of RSA degradation events at various Gazebo update rate configurations in the E-TERRY simulation

Verification of Absence of False Positives in RSA Integration Testing

Following the reconfiguration of the Gazebo update rate to 10Hz, an experiment was performed generating 80 integration tests. In each test, the SBT platform is invoked alongside the RSA. In each test, deletion of laser scanner messages, worker stopping, and manipulation of the worker confidence could be injected as manipulation operations.

To verify for the absence of false positives, every test which activated the RSA emergency status was examined, as shown in **Table 4**. In total there were 5 tests with RSA activation, and in all of them, the minimal human distance was less than 1.0, and the worker confidence was less than 0.6. This confirms there were no false positives in which the RSA activated spuriously in conditions in which it should not have activated.

Table 4: Integration tests in which an RSA activation occurred

TestID	pathCompletion	rsaEmergencyStop	distanceToHuman1 (m)	Test Structure
Test_006	0.999	1.0	0.248	deletePointCloud:{4.3->14.0s} changeHumanX:{7.4->19.3s} WorkerConfidence[0.26]
Test_019	0.997	1.0	0.312	deletePointCloud:{6.6->17.5s} changeHumanX:{4.6->14.3s} WorkerConfidence[0.18]
Test_026	0.997	1.0	0.333	deletePointCloud:{7.3->17.7s} changeHumanX:{4.1->19.6s} WorkerConfidence[0.15]

Test_064	0.971	1.0	0.137	deletePointCloud:{4.3->14.0s} changeHumanX:{9.5->18.4s} WorkerConfidence[0.26]
Test_080	0.997	1.0	0.681	deletePointCloud:{7.1->16.0s} changeHumanX:{4.5->11.4s} WorkerConfidence[0.50]

Verification of Absence of False Negatives in RSA Integration Testing

To verify the absence of false negatives, every test with a human distance less than 1.0m was examined (**Table 5)**. There were 8 tests, of which only 5 injected worker confidence less than 0.6, corresponding to the same tests as found previously in **Table 4**. The three tests in which the RSA did not activate on the close approach had a worker confidence greater than 0.6 (indicated by red in the table). This confirms there were no false negatives in which the RSA should have been activated but failed to do so, and that the RSA logic is operating correctly under the defined rule.

Table 5: Integration tests in which a close approach to the human occurred

TestID	pathCompletion	rsaEmergencyStop	distanceToHuman1 (m)	Test Structure
Test_006	0.999	1.0	0.248	deletePointCloud:{4.3->14.0s} changeHumanX:{7.4->19.3s} WorkerConfidence[0.26]
Test_019	0.997	1.0	0.312	deletePointCloud:{6.6->17.5s} changeHumanX:{4.6->14.3s} WorkerConfidence[0.18]
Test_026	0.997	1.0	0.333	deletePointCloud:{7.3->17.7s} changeHumanX:{4.1->19.6s} WorkerConfidence[0.15]
Test_036	0.997	0.0	0.336	deletePointCloud:{7.1->16.0s} changeHumanX:{5.9->16.0s} WorkerConfidence[0.82]
Test_054	0.997	0.0	0.398	deletePointCloud:{7.1->16.0s} changeHumanX:{5.9->16.0s} WorkerConfidence[0.95]
Test_060	0.997	0.0	0.768	deletePointCloud:{7.7->19.8s} changeHumanX:{5.7->14.7s} WorkerConfidence[0.64]
Test_064	0.971	1.0	0.137	deletePointCloud:{4.3->14.0s} changeHumanX:{9.5->18.4s} WorkerConfidence[0.26]
Test_080	0.997	1.0	0.681	deletePointCloud:{7.1->16.0s} changeHumanX:{4.5->11.4s} WorkerConfidence[0.50]

Testing RSA Response to Dynamic Speed Changes from Navigation Stack

In order to demonstrate integration and verify proper behaviour in dynamic collision avoidance driven fully from simulator variables, the RSA module and the SBT platform were interfaced together, in order to provide automated testing of a scenario involving simulation dynamic variables which change frequently.

In this scenario, the RSA monitors speed inputs derived from navigation stack (nav2) and collision_monitor output variables, intending to register scenarios in which there is a discrepancy between navigation speed and safe speed. These will occur in which the robot is making a safety slowdown or safety stop due to the LIDAR registering a potential obstruction ahead. A ROS node is added to consume **/cmd_vel_nav** (the commanded velocity from the nav2 stack) and emit the resulting speed as the RSA input variable **nav_speed_cmd**. Equivalently, **/cmd_vel_safe** (the safe velocity from the collision monitoring component) is also processed to produce the RSA input variable **safe_speed_cmd.** During normal operation both of these variables are identical, but they temporarily diverge during safety scenarios when the collision monitor commands a lower safe speed.

A *safety slowdown* is registered when the following RSA rule evaluates to true. This rule demonstrates the new ability of a new RSA release to compare the ratio of input variables, as well as thresholding them against fixed values:

```
nav_speed_cmd > 0.05 and
safe_speed_cmd < 0.5 * nav_speed_cmd and
safe_speed_cmd                        >                        0.01
```

A *safety stop* is registered when the following RSA rule evaluates to true, and is also reported on the RSA topic ***/rsa/emergency_status*** to be consumed by the SBT platform metrics:

```
nav_speed_cmd > 0.05 and safe_speed_cmd <= 0.01
```

In order to test the scenario behaviour, an SBT scenario was set up in which the dynamic human worker stops in front of the robot in the middle of the first row. In this scenario the LIDAR operates normally, so the collision monitor detects the obstruction ahead, first slowing down and then stopping, as expected. Meanwhile the RSA monitors the navigation speed variables and evaluates the rules as described above. This scenario in operation is illustrated in **Figure 38**.

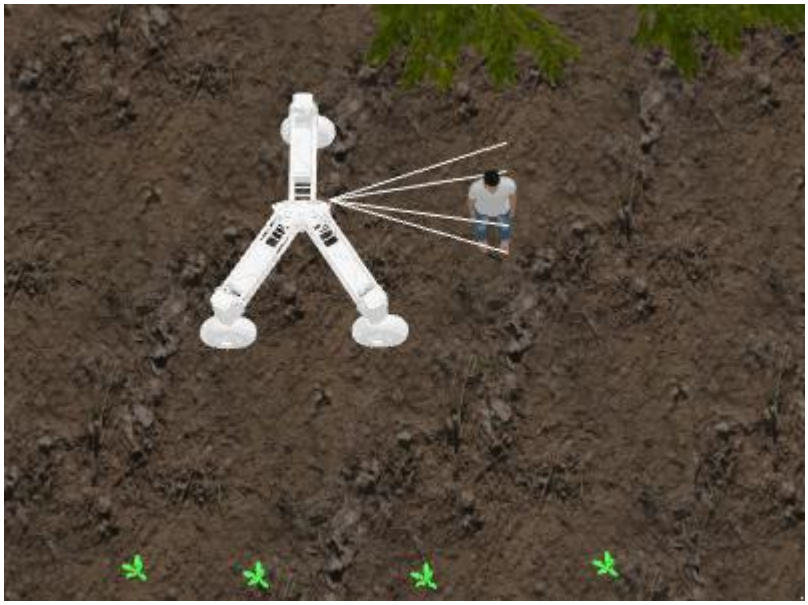


Figure 38: The robot stops to avoid the human resulting in transient discrepancies between the navigation and safety velocities

Testing of the scenario revealed the following:

- The previous version of the RSA (v1) detected the safety slowdown and stopping events and the correct evaluation of the rules, but the rule engine rejected them because the rules were rapidly changing. This is because the discrepancy between the input speed variables is very short-lived. The debugging log emitted "Rule degraded_slowdown_nav flapping frequently (4x within 0.5s)"
- As a result of this testing, a new RSA module version was produced and integrated (v2), which now uses a configurable sliding time window (5 seconds by default), and registers transient rule activations and uses them to activate the resulting rule. This eliminated any repeated flapping warnings in subsequent testing.
- When RSA v2 was tested incorporating this sliding window and improvements to prevent rule flapping, the RSA successfully registered slowdown events on close approach or adjacency to potential obstacles (passing a tree adjacent to the robot's path).
- When RSA v2 approached excessively close to the human, the safety stop rule was triggered, producing the anticipated true value on ***/rsa/emergency_status.***
- After the human has been stationary for a significant time, the navigation stack ceases publishing the commanded velocity. In this case, the RSA enters a degraded state due to the absence of updated input variables. This is expected behaviour due to its necessary inputs being stale.

4. CONCLUSION

The deliverable summarizes the progress made during the final prototype (FP) phase of the SOPRANO project for the development of three core technologies underpinning safety analysis and testing of MH-MR systems.

To this end, design-time and runtime safety assessment (DSA, RSA) which combines ahead-of-time analysis of safety risks within the integrated system, and during operation, by real-time monitoring of sensor data to detect emergency situations. The risk analysis

and assessment technology addresses not only traditional issues related to hardware and software components, but also identifies faults arising from the use of AI algorithms to manage robots. In this document we have presented further developments on the design-time safety analysis (DSA). In addition, we presented the runtime safety analysis (RSA) component and described how it can recognise and react to dynamic safety violations or hazardous situations resulting from input received from both the MH-MR system's own internal components and from external SOPRANO components.

Next, simulation-based testing (SBT) underpins the process of designing and executing testing campaigns that aid the discovery of faults in MH-MR systems, with exploratory and evolutionary algorithms, enabling the hardening of MH-MR systems in advance of deployment. In the FP we have considered the acceleration of the SBT testing process by the addition of offline performance predictors which anticipate the impact of a proposed simulation test in advance, together with selection strategies to focus the limited testing time budget on the most significant tests. We also tested the RSA response with integration under various simulation scenarios under the ETERRY pilot, in order to validate correct operation under automated and manual simulated tests. We will continue this further upon further pilots in the final project integration phase.

Finally, HDTs enable the creation of virtual replicas of the physical environment and the analysis of "what-if" scenarios without exposing workers to actual dangers. In this document we consider the integration of the HBT with the SBT tools in the form of the gRPC interface, and associated file formats for defining MH-MR tasks for the HDT and receiving its results. We also specify techniques and interfaces for the integration with external SOPRANO components and the KUKA mobile platform, which will be further validated within the integration phase of the project.

A) PROGRESS SINCE D3.1

A1) DESIGN TIME AND RUNTIME SAFETY ANALYSIS TOOL

While in D3.1 our work on safety focused on the specification of the overall methodology, the development of the tools around the ODD specification, the development of the initial architecture for the Eclipse implementation of the design-time safety analysis tools, in this deliverable the safety analysis work has been extended towards a more complete lifecycle-oriented framework combining design-time safety analysis, runtime safety monitoring and integration with the broader SOPRANO testing ecosystem.

The main extensions described in D3.3 are:

- A consolidated **design-time safety analysis workflow**, covering ODD specification, hazard identification, unsafe scenario analysis and mitigation strategy definition for MH-MR environments, supported by SOPHIA as an Eclipse Papyrus-based model engineering platform and complemented by model-based safety analysis capabilities such as FMEA, FTA and HAZOP-inspired analysis.
- Extended SOPHIA tool support for operational and safety modelling, including the **integration of Papyrus4ODD** for graphical and textual specification of operational assumptions, environmental conditions and mission configurations, as well as a **dedicated DSA plugin** for analysing functional insufficiencies related to perception, decision-making and control behaviours in collaborative robotic systems.
- A **structured transition workflow** from design-time safety analysis to runtime monitoring, ensuring traceability between hazards, unsafe scenarios, mitigation strategies and executable monitoring artefacts, with a semi-automatic transformation process from SOPHIA safety tables to runtime rules based on observable variables, monitoring conditions and configuration structures.
- A **Python-based Runtime Safety Analysis module** for operational supervision, able to process runtime data streams from robotic middleware and organised as a layered architecture covering input acquisition, preprocessing, configuration-driven rule evaluation, probabilistic reasoning, safety-state fusion and publication of safety-related outputs
- A **flexible runtime rule mechanism** based on structured JSON configuration files, replacing hard-coded monitoring logic and enabling reusable, scalable and adaptable rule sets across use cases, while preparing the future objective of automatic rule generation from SOPHIA design-time artefacts
- A **runtime safety-state management approach** defining nominal, degraded and emergency modes, together with configurable transition logic, persistence checks and recovery mechanisms to support robust supervision and avoid unstable state oscillations in dynamic human-robot collaboration contexts
- The introduction of **probabilistic risk indicators** complementing deterministic rule evaluation, especially for uncertainty-related safety aspects such as human-robot proximity, perception confidence and data freshness, and preparing future integration of Bayesian reasoning and runtime exploitation of structural risk propagation models derived from FTA capabilities

• **ROS2-based runtime signalling mechanisms** through which the RSA module publishes safety states, warning events, mitigation requests and execution gate indicators, allowing navigation, perception, control and task allocation components to adapt their behaviour while preserving modularity between safety supervision and low-level robot control
• **Integration of the RSA module within the wider SOPRANO safety and testing ecosystem**, linking design-time safety artefacts, simulation-based validation, digital twin services and runtime monitoring, and enabling simulation scenarios to evaluate degradation, emergency activation, recovery behaviour, changing distance conditions, perception confidence variations and navigation-related safety constraints.

Overall, the work described in D3.3 deliverable provides a significant extension of the safety analysis tool from a design-time engineering support environment towards an integrated safety supervision framework. It establishes continuity between modelling, risk analysis, simulation-based validation and runtime monitoring. Future work will focus on further automating the transformation from SOPHIA design-time artefacts to executable RSA rules, strengthening the integration of Bayesian and fault-propagation reasoning mechanisms, and consolidating the use of runtime safety indicators within the SOPRANO simulation and testing workflow.

A2) THE SCALABLE AND EXTENSIBLE SIMULATION BASED TESTING TOOL

In D3.3, we have considered the acceleration of the SBT testing process by the addition of offline performance predictors which anticipate the impact of a proposed test in advance, together with selection strategies to focus the limited testing time budget on the most significant tests.

Accordingly, the following changes since D3.1 have been described in D3.3:

- A description of **the predictor training methodology** and how it is integrated into the wider SBT methodology
- **Techniques for data preprocessing** to produce time-series datasets from a training set of experimental models
- **Predictor training and testing strategies** incorporating feature extraction and learning techniques incorporating leading models from the time-series extrinsic regression literature (such as MiniRocket, TSForest and gradient boosting)
- **Strategies for test selection** incorporating custom selection algorithms which take metric predictions for relevant metrics and generate a custom set of information
- **Modifications to the SBT DSL**, supporting the configuration of predictors to be used during a live experiment
- A description of **SBT inter-component communications** incorporating a new predictor daemon, and how it is accessed for test execution decisions during the experimental flow with the experiment manager and worker daemon
- An **example preliminary evaluation** on the ETERRY use case demonstrating example predictors and test selection strategies

In D3.3, we also demonstrated SBT i**ntegration with the runtime safety (RSA)** component, validating parameters and correct operation of both components under automated and manual simulated tests. This testing incorporated:

- the demonstration of **bidirectional communications between the RSA and SBT** component; with metric monitoring from the RSA via ROS topics, and metric feedback to transmit information from the SBT platform (distance values) back to the RSA
- automatic investigation via SBT of **metric update rates** from the simulation and their impact on reported RSA degradation rates
- **verification of correct RSA performance** and the absence of false positives and negatives in emergency condition activation
- use of the SBT platform to test the **dynamic RSA response to rules** incorporating discrepancies in navigation and safe speeds, resulting in enhancements to the RSA to handle rapidly changing simulation variables.

Overall, the work described in D3.3 for the SBT component provides a viable foundation for the integration of SBT into an integrated SOPRANO system. Future work will include enhancing the dynamism of the predictors using dynamic retraining during experiments, and the use of online prediction for early termination of experiments.

A3) HUMAN DIGITAL TWIN FOR MH-MR SYSTEM (HDT)

Since D3.1, the Human Digital Twin (HDT) has been significantly enhanced in terms of architecture, simulation fidelity, and system integration. A major advancement concerns the introduction of a structured data management and communication mechanism based on the Shared Memory component and modular datasources, enabling decoupled and interoperable data exchange across the Digital Twin Service platform. This architecture supports integration with heterogeneous systems and protocols (e.g., gRPC, OPC UA, ROS2), ensuring efficient communication with other SOPRANO components and alignment with ISO 23247 principles.

In parallel, the simulation capabilities have been improved, including more realistic modelling of the robotic platform (e.g., full navigation, accurate milling process) and the extension of the Human Digital Twin to support task execution and interaction within the environment. These improvements enable a more accurate representation of both human and robot behaviours, supporting reliable safety analysis and performance evaluation.

Furthermore, the HDT has evolved towards a fully integrated component within the SOPRANO ecosystem through its connection with the Simulation-Based Testing (SBT), MAT, and real robotic platforms.

The integration with SBT enables dynamic manipulation of simulation parameters, fault injection, and exploration of edge-case scenarios, while ensuring controlled and synchronized simulation execution.

The integration with MAT allows automatic generation of simulation environments and task definitions from BIM data, enhancing scalability across use cases. Additionally, the connection with the KUKA mobile platform via OPC UA and the

introduction of virtual LiDAR sensors with ROS2 communication enable hybrid simulation and virtual commissioning scenarios, ensuring consistency between simulated and real-world behaviour.

Overall, these advancements transform the HDT into a robust and interoperable Digital Twin solution supporting advanced validation, testing, and safety assessment of MH-MR systems.

REFERENCES

[1] International Organization for Standardization (2010) ISO 12100:2010 Safety of machinery- General principles for design—Risk assessment and risk reduction. Geneva: ISO.

[2] International Organization for Standardization (2015) ISO 13850:2015 Safety of machinery - Emergency stop function — Principles for design. Geneva: ISO.

[3] International Organization for Standardization (2024) ISO 13855:2024 Safety of machinery - Positioning of safeguards with respect to the approach of the human body. Geneva: ISO.

[4] International Organization for Standardization (2019) ISO 13857:2019 Safety of machinery - Safety distances to prevent hazard zones being reached by upper and lower limbs. Geneva: ISO.

[5] International Organization for Standardization (2025) ISO 10218-1:2025 Robotics - Safety requirements - Part 1: Industrial robots. Geneva: ISO.

[6] International Organization for Standardization (2025) ISO 10218-2:2025 Robotics - Safety requirements - Part 2: Industrial robot applications and robot cells. Geneva: ISO.

[7] International Organization for Standardization (2016) ISO/TS 15066:2016 Robots and robotic devices - Collaborative robots. Geneva: ISO.

[8] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation 6, 2 (2002), 182–197.

[9] Angus Dempster, François Petitjean, and Geoffrey I. Webb. 2020. ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. Data Mining and Knowledge Discovery 34, 5 (July 2020), 1454–1495. https://doi.org/10.1007/s10618-020-00701-z

[10] Dempster, Angus and Schmidt, Daniel F. and Webb, Geoffrey I. 2021. MiniRocket: A Very Fast (Almost) Deterministic Transform for Time Series Classification. Proceedings of the 27th ACM SIGsous anesthésieKDD Conference on Knowledge Discovery and Data Mining. (Aug 2021), 248-257. http://dx.doi.org/10.1145/3447548.3467231

[11] TSFresh Authors. 2026. TSFresh Library Github. https://github.com/blue-yonder/tsfresh.

[12] Houtao Deng, George Runger, Eugene Tuv, and Martyanov Vladimir. 2013. A time series forest for classification and feature extraction. Information Sciences 239 (2013), 142–153. https://doi.org/10.1016/j.ins.2013.02.030

[13] Jerome H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. The Annals of Statistics 29, 5 (2001), 1189 – 1232. https://doi.org/10.1214/aos/1013203451

[14] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. 2007. Evolutionary algorithms for solving multi-objective problems. Vol. 5. Springer.

[15] Markus Löning, Anthony Bagnall, Sajaysurya Ganesh, Viktor Kazakov, Jason Lines, Franz Király (2019): “sktime: A Unified Interface for Machine Learning with Time Series”