Ref. Ares(2026)5995354 - 12/06/2026





SOPRANO

Socially-Acceptable and Trustworthy Human-Robot Teaming for Agile Industries

Project Number 101120990

D4.2 Modular, Reconfigurable, and Flexible Tools II

Version 1.0
2 April 2026
Final

Public Distribution

ATB, HUA, PROFACTOR, and University of Bremen

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 outline	1 February 2026
0.2	Initial partner contributions	1 March 2026
0.3	Updates to outline for final prototype	6 March 2026
0.7	First draft for internal review	23 March 2026
0.9	Updated version from internal reviews	31 March 2026
1.0	Final QA version	2 April 2026

TABLE OF CONTENTS

1 Introduction **1**
1.1 Overview . . . 1
1.2 Structure of the Document . . . 1

2 Context Extraction Module **1**
2.1 Objectives and Approach . . . 1
2.2 Context Models . . . 2
2.3 Context Extraction Services . . . 3
2.3.1 Context Monitoring . . . 7
2.3.2 Context Extraction and Provision . . . 8
2.4 Results . . . 9
2.4.1 KUKA Pilot – Renovation Robot Scenario . . . 10
2.4.2 CRF Pilot – Vehicle Door Disassembly . . . 10
2.4.3 E-TERRY Pilot – Symbiotic Farming Scenario . . . 11
2.4.4 Cross-Pilot Summary . . . 12
2.5 Challenges and Future Work . . . 12

3 MH-MR Architecting Tools **13**
3.1 Objectives . . . 13
3.2 Composable Models . . . 13
3.2.1 Composable FloorPlan Models and Building Information Modelling (BIM) . . . 14
3.2.2 Composable Motion Specifications . . . 19
3.3 Architecting Tools . . . 23
3.3.1 Scenery Builder . . . 23
3.3.2 Control Architecture . . . 26
3.3.3 Experimental Setup at Uni-Bremen . . . 29
3.4 Results . . . 30
3.4.1 Composable models . . . 30
3.4.2 Architecting tools . . . 30
3.5 Challenges and Future Work . . . 31
3.5.1 Composable models . . . 31
3.5.2 Architecting Tools . . . 31

4 AI Model Optimizer & MLOps Orchestrator **31**
4.1 Objectives . . . 31
4.2 AI Model Optimizer . . . 31
4.3 MLOps Orchestrator . . . 32
4.4 Results . . . 34
4.5 Challenges and Future Work . . . 37

5 Robot Capabilities Implementation **37**
5.1 Objectives . . . 37
5.2 Task Execution Framework for Robot Capabilities . . . 37
5.3 Robotic Manipulation Approach . . . 39
5.3.1 UR10 Robot . . . 41
5.3.2 Felice Robot . . . 42
5.4 Results . . . 45
5.4.1 Task Execution Framework . . . 45
5.4.2 Robot Manipulation . . . 45
5.5 Challenges and Future Work . . . 49

6 Conclusions and Next Steps **49**

A Updates since D4.1 **50**
A.1 Context Extraction Module (T4.1) . . . 50
A.1.1 Advancements since D4.1 . . . 50
A.2 MH-MR Architecting Tools (T4.1 & T4.2) . . . 51
A.3 AI Model Optimizer (T4.4) & MLOps Orchestrator (T4.5) . . . 51
A.3.1 AI Model Optimizer . . . 51
A.3.2 MLOps Orchestrator . . . 51
A.4 Robot Capabilities Implementation (T4.6) . . . 52
A.4.1 Advancements since D4.1 . . . 52
A.4.2 Remaining Challenges and Future Work . . . 52

LIST OF FIGURES

1 Ontology view of the AutonomousAgent class . . . 3
2 Ontology view of the AutonomousAgent class in the KUKA instantiation . . . 3
3 Ontology view of the Event class for a KUKA instantiation . . . 4
4 SOPRANO ontology context model KUKA instance main entities . . . 4
5 Ontology view of the Context Extraction Module (CEM) input relevant SOPRANO components 4
6 SOPRANO context model ontology CRF instance main entities . . . 5
7 SOPRANO partial view of the ontology instantiation for farming use case (main entities) . . . 5
8 Context Extraction Service . . . 6
9 Context Extraction Service . . . 7
10 Model-to-text transformations for the FloorPlan models for v1 and v2, on the top and bottom, respectively . . . 14
11 Systematic navigation of models using Jinja templates at different levels of abstraction . . . 15
12 Concepts and their subclasses present in the Industry Foundation Classes (IFC) models . . . 18
13 *IfcShapeRepresentation* concepts that have a 3D geometric subcontext in IFC models. . . . 19
14 Motion Specification pipeline . . . 19
15 The updated composition and generation pipeline to integrate IFC models and high-level task specifications for the construction domain. . . . 23
16 Occupancy grid and 3D mesh generated from IFC model of the KUKA test bed. . . . 26
17 Experimental setup for the CRF unscrewing use case at Uni-Bremen. A dual-arm Kinova Gen3 is positioned in front of a car door carrying screws, electrical components, and cables. . . . 29
18 Perception results for the CRF use case: RGB-based screw detection on the car door (left) and 6D pose estimation of the detected screws (right). . . . 29
19 AIO Architecture . . . 33
20 Quantizer . . . 33
21 Distributed Inference . . . 35
22 MLO–AIO import . . . 35
23 MLO–AIO Processor Manifest Mapping . . . 36
24 MLO–AIO flow creator . . . 36
25 Mapping Robotic Capabilities(MRC) and Robot Capability Implementation (RCI) Architecture as deployed on Felice robot, UR10 robot and KUKA Platform . . . 38
26 Simplified flowchart of the unscrewing process, divided into the execution layers. . . . 42
27 ROS2 Hardware and controller software architecture for mobile robots and mobile manipulators 42
28 Planning scene and MoveIt Task constructor pipeline for Tool 10 pickup . . . 43
29 Planning scene and MoveIt Task constructor pipeline for Tool 20 pickup . . . 44
30 UR10 Setup installed at the CRF Disassembly Line, Melfi, Italy . . . 46
31 Felice Robot Tool Pickup . . . 47
32 Felice Robot executing all desired tasks . . . 47
33 Felice Robot executing Tool Pickup at Workstation 20 . . . 48

Acronyms

AIO AI Model Optimizer. 1, 31, 32, 34, 36, 37, 49

AVT Advanced Visualization Toolkit. 1, 46, 48

BIM Building Information Modelling. 13, 14

BT Behavior Tree. 37, 38, 49

BTRTL Behavior Tree Run-Time Layer. 37, 38, 41

CAD Computer-Aided Design. 17, 18, 45

CEM Context Extraction Module. vi, 1, 4, 35, 49

CL Control Layer. 38, 39, 41

CLI Command-Line Interface. 23

CRF Centro Researche Fiat. 48

CTA MH-MR Task Allocation. 1, 38, 45, 46, 48

DAG Directed Acyclic Graph. 34

DP Dynamic Programming. 32

DPE Data Processing Engine. 34

DSL Domain-Specific Language. 15, 16, 18, 21, 23, 28, 30

ERP Enterprise Resource Planning. 32

FOAF Friend of a Friend. 2

GIS Geographic Information System. 33

GNSS Global Navigation Satellite System. 3, 11

HDT Human Digital Twin. 1, 24, 26, 30

HMO Human Monitoring. 1–3, 8–10

IFC Industry Foundation Classes. vi, 13, 15–19, 23, 24, 26, 30, 31, 49

M2M Model-to-Model. 16, 24

M2T Model-to-Text. 15

MAT MH-MR Architecting Tools. 1, 13, 49

MES Manufacturing Execution System. 32

MLO MLOps Orchestrator. 1, 32–37, 49

MPPE Multi-Purpose Processing Engine. 33

MRC Mapping Robotic Capabilities. 1, 38, 41, 46

NVL Non-Visual Localization. 1–3, 8–10

OBP Object Perception. 1–3, 8–10, 46, 48

OPC-UA OPC Unified Architecture. 8, 10, 12, 49

OWL Web Ontology Language. 2, 12

PID Proportional-Integral-Derivative. 20, 28

RCI Robot Capabilities Implementation. 1, 38, 39, 42, 46–49

RDF Resource Description Framework. 10, 12, 29

RL Robot Layer. 38, 39

SIL Skill Implementation Layer. 37–39, 41

SLAM Simultaneous Localization and Mapping. 13

SOHO Sharework Ontology for Human-Robot Collaboration. 2, 3

TRL Technology Readiness Level. 14, 30, 49

VLM Visual Spatial Localization and Mapping. 1–3, 8–10, 24, 30

WME Workflow Management Editor. 33, 34, 49

EXECUTIVE SUMMARY

This deliverable provides a comprehensive description of the final prototypes of the components in WP4. We cover the development, methodology and tools designed to address multi-human, multi-robot (MH-MR) collaboration in dynamic industrial environments. Each component discusses what has been achieved in the early (D4.1) and final prototypes, as well as identifying challenges, limitations and future work. On Appendix A we provide reviewers a guide to the changes to this document since the early prototype (D4.1).

The primary focus of this deliverable is to establish the foundational tools and frameworks needed for flexible, reconfigurable MH-MR collaboration. The key contributions include:

- **Context Models and Context Extraction Services**
  - Development of an ontology-based context modelling framework enabling formal representation of multi-human multi-robot (MH-MR) collaboration environments across heterogeneous domains.
  - Definition and extension of SOPRANO Context Model Ontologies, building upon existing standards (e.g., SOHO, FOAF)
  - Real-time semantic context identification and reasoning, transforming low-level sensor and system data into high-level, machine-interpretable contextual knowledge.
  - Deployment of event-driven context change detection, enabling dynamic updates of operational context based on human presence, object detection, robot state, and task transitions.
  - Contribution to adaptive MH-MR coordination, supporting context-aware decision-making, safety monitoring, and dynamic task execution within the SOPRANO ecosystem.
- **Architecting Tools for MH-MR Systems**
  - Design and implementation of composable models to describe task environments and specifications, and motion specifications.
  - Semantic alignment of IFC concepts to existing metamodels of the MH-MR Architecting tools component and generation of simulation artefacts based on IFC models
  - A Motion Specification DSL to describe motion specifications in a human-readable form, modelling arm motions based on atomic motions and constraints.
  - Code generation tooling for the control loop of motion specifications from the composable models into C++ execution code.
- **Support for AI/ML Pipelines**
  - Design and implementation of an MLOps Orchestrator (MLO) that combines a Workflow Management Editor (WME), a Multi-Purpose Processing Engine (MPPE), and dedicated registries for digital resources, data processors, DAGs, and assets to manage end-to-end AI workflows.
  - Provision of a user-friendly drag-and-drop GUI and REST APIs that enable non-expert users to configure data sources, register reusable processors, visually compose workflows, and deploy containerised pipelines without writing code.
  - Integration of MLO into the SOPRANO architecture to orchestrate data and AI processing across heterogeneous MH-MR scenarios, connecting IoT systems, enterprise data sources, and robotic components under a governed, traceable execution environment.
  - Development of a component (AIO) to apply split computing in neural network models and find the optimal matchmaking plan with cluster devices in order to minimize the distributed inference latency and local resource usage.
  - The component selected for testing splitting and distributed inference is the OBP module, which is based on the DeepLab and YOLO models for tool detection.
  - To enable non-technical users to deploy AI models, an MLOps orchestrator has been developed. It supports a drag-and-drop interface and, based on the provided inputs, automatically deploys the required Docker containers
- **Robotics Capabilities Implementation**

- A modular and scalable robot capabilities execution framework (RCI) has been successfully developed, enabling the translation of high-level task plans into executable robotic actions (e.g., grasping, holding, unscrewing) across multiple platforms (UR10, Felice Robot, KUKA platform).
- A Behavior Tree-based task execution architecture has been implemented and validated, integrating perception, control, and decision-making layers to ensure flexible, robust, and adaptive operation in dynamic industrial environments (CRF use case).
- Core robotic manipulation capabilities and atomic skills have been developed and tested, demonstrating reliable performance in key operations such as tool handling (including manipulation, grasping), handover (collaborative handover of tools to human), and disassembly tasks (unscrewing), particularly in the CRF use case.
- The framework has been successfully integrated and deployed in real-world within the overall system architecture, while ongoing refinements focus on enhancing robustness, safety, extended validation and performance.

1 Introduction

1.1 Overview

This deliverable summarizes the design and development of the components developed during the SOPRANO project to support multi-human, multi-robot (MH-MR) collaboration in industrial settings. The **Modular, Reconfigurable and Flexible Tools** covered in this document are: A Context Extraction Module (CEM), the MH-MR Architecting Tools (MAT), AI Model Optimizer (AIO), MLOps Orchestrator (MLO) and the Robot Capabilities Implementation (RCI).

When relevant, we describe how components interact with the following SOPRANO components: Visual Spatial Localization and Mapping (VLM), Object Perception (OBP), Non-Visual Localization (NVL), Human Monitoring (HMO), Human Digital Twin (HDT), Advanced Visualization Toolkit (AVT), MH-MR Task Allocation (CTA) and Mapping Robotic Capabilities (MRC).

Results are presented across the three SOPRANO industrial pilots: A building renovation use case with KUKA, a vehicle door disassembly with CRF, and symbiotic farming with E-TERRY. As a shorthand, we refer to use cases by the industrial partner name (e.g., CRF use case).

1.2 Structure of the Document

This document is organized as follows:

- Sect. 2: Context Extraction Module – Context models and context extraction developed under Task 4.1
- Sect. 3: MH-MR Architecting Tools – Tools for MH-MR system architecting, including composable models and motion planners (Task 4.2 and 4.3)
- Sect. 4: AI Model Optimizer and MLOps Orchestrator – AI pipelines and interoperable platforms supporting non-expert users. (Task 4.4 and 4.5)
- Sect. 5: Robot Capabilities Implementation – Robotic capabilities implementation with examples from pilot use cases. (Task 4.6)
- Sect. 6 Conclusions and next steps, summarizing progress and outlining future directions.
- Appendix A: The appendix summarizing changes from the early prototype version (D4.1) for reviewers.

2 Context Extraction Module

2.1 Objectives and Approach

The objectives of this component are (1) the development of generic context models using an ontological approach that can be easily adapted for different scenarios (2) the monitoring of the different sources of information from the industrial pilots and (3) the extraction of the context at each moment in time using the context models and the context monitoring components. The hypothesis is that by using an ontological approach, we could model different contexts under which the (multi) human and (multi) robot collaboration is taking place and be able to extract the changed context under which the human and robots are in a wide range of settings. The models include classes and relations relevant for collaborative human-robot work in various settings. Based on these context models/ontologies, the Context Extraction Services makes dynamic, real-time extraction of context.

2.2 Context Models

The Context Models describe the dynamic aspects under which the SOPRANO agents (humans and robots) are interacting between them and with the environment. The chosen ontological approach means that we use an ontology based on the Web Ontology Language (OWL) as context model to represent the extracted information, as explicit machine interpretable knowledge. Following best practices for the reuse of existing work in ontology engineering, we have investigated several existing ontologies on the topic of robot and human collaboration (see the most relevant mentioned below) [12] and have integrated published vocabularies, taxonomies and ontologies into the SOPRANO context model ontology. In cases where existing and integrated ontologies for modelling SOPRANO dynamic context were incomplete, these ontologies were extended. The approach to context model development being followed is:

- Specification of the main concepts to be used, coming from the SOPRANO environment
- Note: the analysis of the already existing ontologies was made in parallel, overtaking said concepts from these ontologies.
- Specification of sub-classes (either reusing from the identified ontologies or from scratch)
- Specification of the relationships among concepts (either reusing from the identified ontologies or from scratch)
- Verify that all requirements of the identified project use cases are covered within the SOPRANO Context Model Ontology and make improvements as necessary
    - Iterate and add more details
- Revisit and check completeness of the ontology with the use cases specific human-robot environment
    - Iterate and add more details
- Currently in the development we focus on a first instance of generic models that depict a set of concepts and their relations which describe a small set of entities, attributes and stakeholders within collaborative human-robot work.
- The above mentioned external ontologies and vocabularies that were identified for reuse are: Friend of a Friend (FOAF) and, Sharework Ontology for Human-Robot Collaboration (SOHO) [25]. Others such as KnowRob [23] and SkiROS [17][were studied but not considered as they model a single robot, or SpaceOntology [4] where the navigation concepts are more complicated than the SOPRANO needs.
- After the analysis, SOHO was taken as basis for the SOPRANO context model ontology for having already a good set of concepts that we were looking to model (also including entities that are covered by FOAF to model human agents) and for being based in a top-level ontology, DOLCE [5]. A top-level ontology aims at describing reality from a high-level perspective to define concepts that are general enough to be valid across many domains. Their use is generally recommended and represents a good design choice in order to base new (more specific) ontological models on well-structured semantics and will facilitate later interoperability with other ontologies [3], [1] when the need to extend the context models to cover a wider set of identified contexts is identified.

The SOPRANO context model ontology done for early prototype was revisited and extended for the current development and contains the main concepts in the fields of:

- Agents identification such as humans or Robots and respective connected categories. Autonomous Robots are Agents in the environment with the capabilities they can perform (see Fig. 1). Please note that in the respective pilot instantiations, the KUKA and E-TERRY robots are sub-classes of Autonomous-Robot and both CRF robots are sub-classes of Cobot.
- The ontology also considers the events and actions that may happen in an industrial environment and actions that could be taken (see Fig. 3 for some additional events that were identified for the Construction Robot in the KUKA use case).
- To support the project components that can provide data that influences the context under which the robots are being used or are interacting with the Humans (and vice-versa), the ontology is extended to model these components. The components in question are VLM, OBP, NVL and HMO (see Fig. 4).

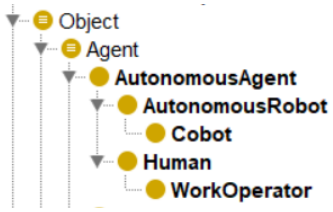


Figure 1: Ontology view of the AutonomousAgent class

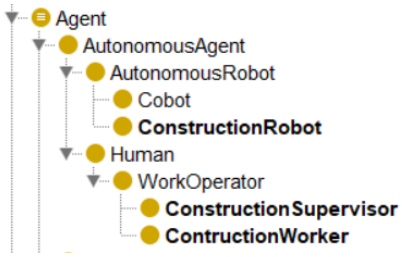


Figure 2: Ontology view of the AutonomousAgent class in the KUKA instantiation

- To support the pilot use case scenarios and needs, the ontology also models Human body position and posture (for instance to identify a new command given to the Robot).

Fig. 5 shows the overview of the main entities that are used in the current prototype of the SOPRANO ontology. In addition to the entities, relationships between these entities are also modelled to enable the reasoning over the ontology at runtime.

All pilot use case scenarios were analysed and all identified entities that were not present in SOHO were added extending the ontology for SOPRANO. For instance, since SOHO was considering mostly manufacturing settings, KUKA's construction setting is not present as well as the farming setting of the E-TERRY pilot were not contemplated and needed to be extended in each of the dedicated pilot instantiations. The KUKA instance main entities model the motion state and robot status as well as the robot arm position. It models as well the construction robot pose estimation and localization in the room that the VLM component can provide and the object detection and relative position with respect to the construction robot that the OBP component may provide (see Fig. 5).

In the CRF shop-floor environment, the Context Monitoring component observes worker presence, posture and proximity to robot (provided by the HMO component), Robot localization in assembly workspace (provided by NVL / VLM) and the robot state, such as active task, tool usage and motion state (provided by the ROS2-based interfaces). The core SOHO ontology has already an automotive ontology that only needs the addition of the SOPRANO components and connection to the robot and Humans observed events (see Fig. 6).

In the E-TERRY pilot context model we extended the ontology to include the AutonomousRobot to a FarmingRobot and modelled the events that are observed by the Context Monitoring module, namely the localization using the Global Navigation Satellite System (GNSS), objects detection (obstacles and crops), human detection in the field and the robot operating mode. These events are then related to the sensing components that detect these and the actors involved. Fig. 6, presents an overview of the main classes modelled.

2.3 Context Extraction Services

Context Extraction Service architectural overview is presented in Fig. 8. The Context Monitoring and Context Extraction subcomponents are documented in the following subsections.

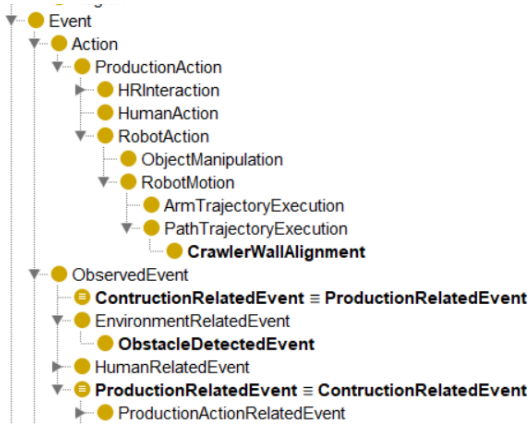


Figure 3: Ontology view of the Event class for a KUKA instantiation

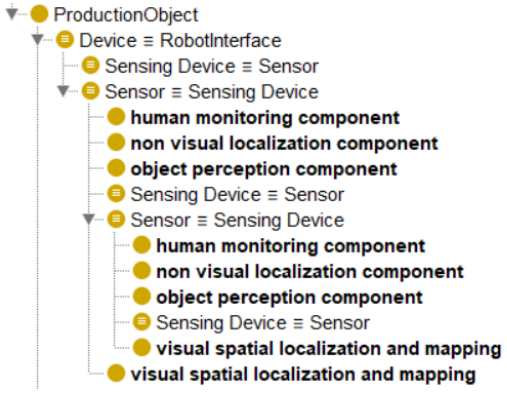


Figure 4: SOPRANO ontology context model KUKA instance main entities

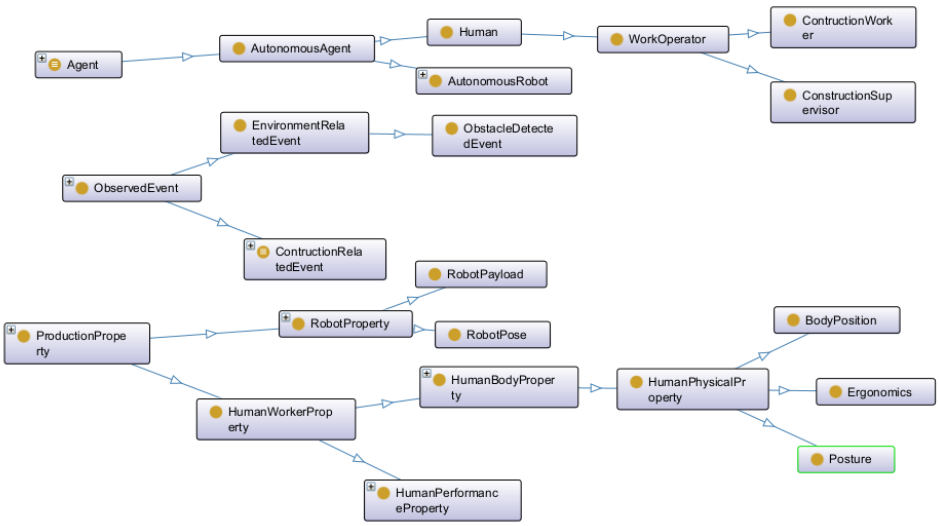


Figure 5: Ontology view of the CEM input relevant SOPRANO components

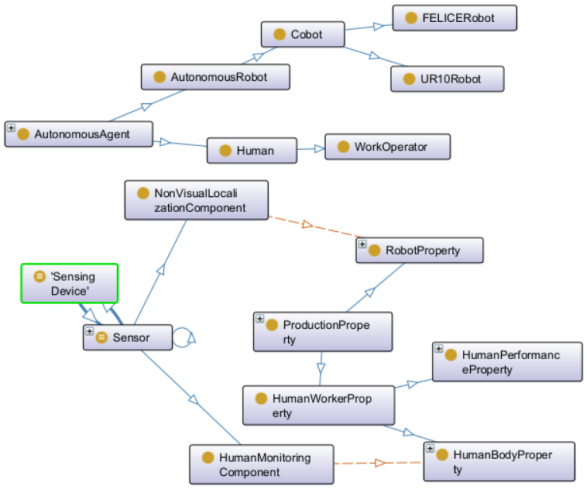


Figure 6: SOPRANO context model ontology CRF instance main entities

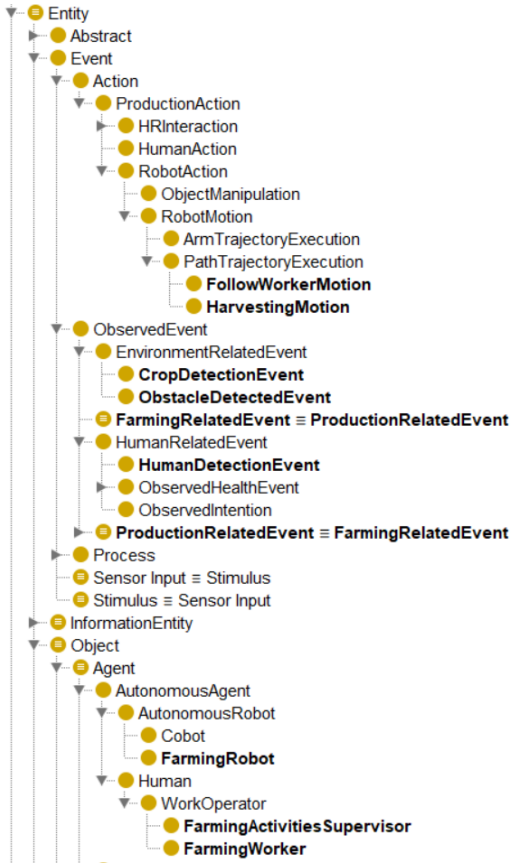


Figure 7: SOPRANO partial view of the ontology instantiation for farming use case (main entities)

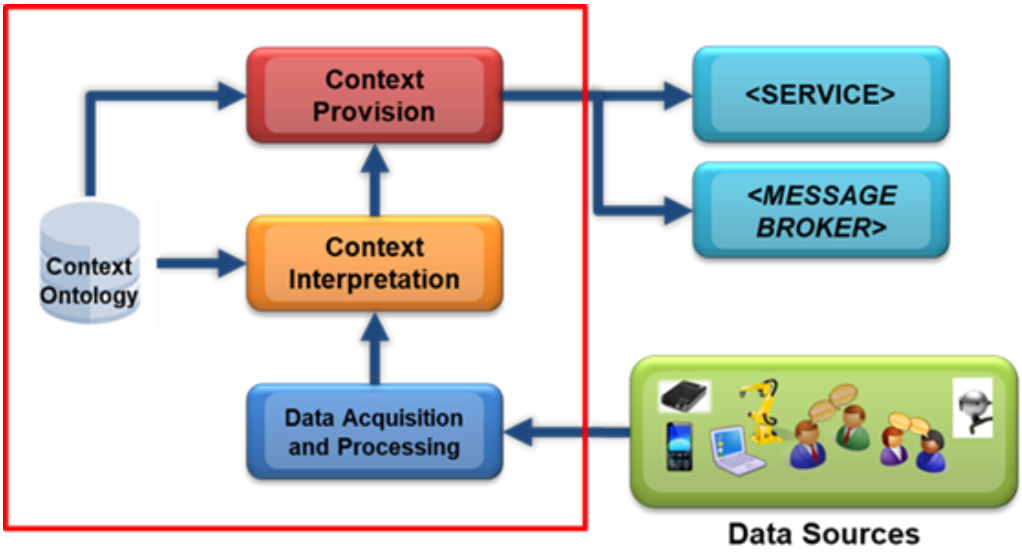


Figure 8: Context Extraction Service

2.3.1 Context Monitoring

The objective of the Context Monitor service is to receive raw data and provide aggregated context data. It is a generic solution for monitoring data sources, which is customizable for different communication protocols and data structures. It enables data pre-processing or data aggregation.

2.3.1.1 Design Approach

The main objective of the Context Monitoring component is to receive raw sensor data and provide aggregated context data. To this end, it allows monitoring of legacy systems in enterprises and products via different interfaces from the Data Access Layer. Therefore, it is able to standardize and correlate data from distinct systems (e.g., map actions from file systems and web services), which later serve as a basis for identification and extraction of context. The main component of Context Monitoring is the modular monitoring process, used for all monitoring features with an extendable and configurable standardized process (see Fig. 9).

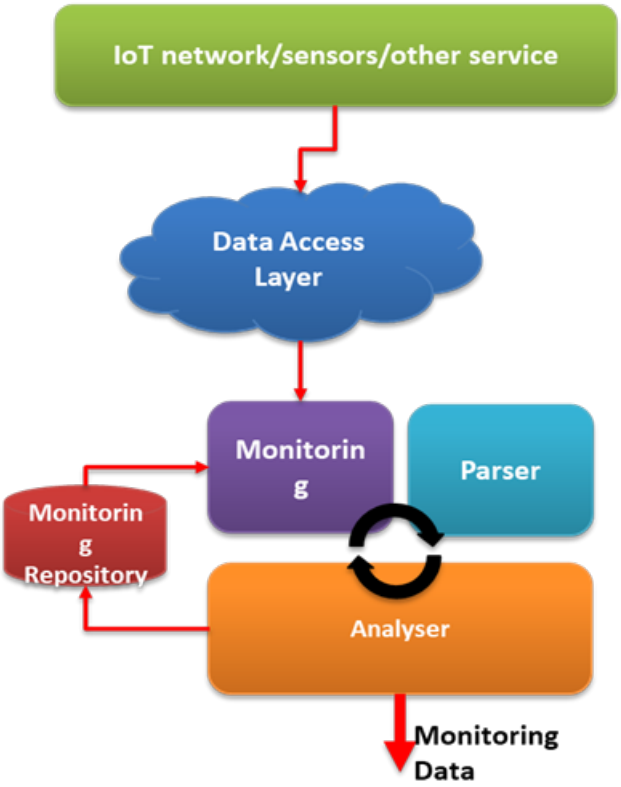


Figure 9: Context Extraction Service

The process consists of three parts:

- Monitoring system/sensor/behaviours service, which contains all features to monitor legacy systems and devices in enterprises via the Data Access Layer. The distributed monitoring services also call back to this module with their gathered information. The monitoring features can be extended and configured for different systems and do not need to comply with other modules.
- Parser module, which contains content parser for the different possible data captured by the monitoring module. The parser offers access to the diverse data possible interacted and therefore monitored with. It provides access to the analyser and may parse available environment properties.

- Analyzer / Monitoring Data Builder module, which correlates the monitored content (and maybe environment properties) and constructs the standardized monitoring data to be stored and handed over to the Context Monitoring / Determination service or any other service that needs this information.

For each data source to be monitored an index can be specified, which will hold the environment properties of the resources monitored. Depending on the actual resources to be parsed and analysed (e.g., XML data from web services), several resource specific plugins can used by the generic Context Monitoring framework. Each of these plugins allows inclusion of parsers and analysers, which are specified for additional external and legacy systems.

2.3.1.2 Implementation

The following section describes the specific implementation of the Monitors of the Context Monitoring component.

Monitors

The System Monitor has several generic monitors that were implemented in the current prototype. Each of the monitors is implemented as a software service. These monitors are the following:

- Monitoring Service for the SOPRANO message broker
- Monitoring Service for REST-based web services
- Monitoring Service for legacy file systems
- Monitoring Service for relational databases

Utilising the Monitoring Service for the SOPRANO message broker, the following monitors were implemented:

- HMO: This monitor observes the HMO component for context relevant information.
- NVL: This monitor observes the NVL component for context relevant information.
- OBP: This monitor observes the OBP component for context relevant information.
- VLM: This monitor observes the VLM component for context relevant information.
- ROS2: This monitor observes robotics information in the CRF and E-TERRY pilots that are provided via ROS2.
- OPCUA: This monitor observes the KUKA robot, that is providing its robot status via OPC Unified Architecture (OPC-UA).

2.3.2 Context Extraction and Provision

The objective of the Context Extractor service is to identify the context of development processes or specified systems and to provide it for further use within the SOPRANO solution to other modules or external systems.

2.3.2.1 Design Approach

The service uses a context model for an integrated representation of the observed environment (e.g., products, systems, machines, processes, or users).

The Context Extraction component consists of the following services:

- Context identification: It determines the current context of products, machines and production processes using monitored raw data from the Context Monitoring component and historic context information stored in a context repository, based on the context model.

- Context reasoning: It uses reasoning rules on the context provided by the context identification part, and determines more accurate context, which cannot be directly identified during the process in the context identification part.
- Context provision: It provides detected context to the other services

The Context Extraction component identifies context based on monitoring data, provided by the monitoring module, enhances it through different types of reasoning techniques (i.e., context reasoning), and provides the refined context for further exploitation to modules (or external systems).

The Context Extraction module is based on the services developed within the project SmartCLIDE. The module was extended to address the specific needs of the SOPRANO solution, as well as the particularities of the three business cases. This includes the modelling of the specific SOPRANO domain and specifically of the business cases specific context (addressed by the Context Model), the monitoring of the SOPRANO specific data sources (addressed by the Context Monitoring module) and finally the Context identification based on the new Context Model and Monitors (addressed by the Context Extraction module). The collection of input data will be conducted by the monitoring module, and the Context Extraction part will analyse structured and unstructured data to specify the current context and identify the parameters of the product/machine or process environment that could affect its performance. The identification of the respective context is carried out based on the context models, which define each time what information is relevant to the observed context. The identified context information is being stored in the context repository as annotation to the content that is used in the observed context.

2.3.2.2 Implementation

The following section describes the specific technical specifications of the Context Extraction and Provisioning component.

Context Extraction

For the SOPRANO solution, a specific version of the context extraction was implemented. As described, the Context Monitoring continuously observes the SOPRANO components HMO, NVL, OBP, VLM. On top it also observes the robots either using ROS2 or an OPCUA server.

The current implementation identifies a context change when either the OBP or the HMO identifies an object or a human operator. Further, a context change could also be initiated by robotic information, e.g. emergency stop executed.

Context Provision

The context provision creates a message for the SOPRANO message broker, whenever a context change is detected. It uses a XML structure, that is according to the SOPRANO data model

2.4 Results

The preliminary testing of the Context Extraction Service was conducted across all three SOPRANO industrial pilots:

- KUKA – Building Renovation
- CRF – Vehicle Door Disassembly
- E-TERRY – Symbiotic Farming

The objective was to evaluate whether the ontology-based context awareness framework can reliably detect, interpret, and provision context information in heterogeneous, real-world MH-MR environments.

The validation focused on:

- Multi-source monitoring
- Semantic context identification
- Context change detection
- Context provision via the SOPRANO message broker
- Cross-domain applicability of the ontology model

2.4.1 KUKA Pilot – Renovation Robot Scenario

Experimental Setup

Three data sources were monitored:

- KUKA renovation robot
- Motion state (moving / stationary)
- Arm position
- Status information via OPC-UA
- VLM
- Robot pose estimation
- Robot localization in room
- OBP
- Object detection and relative positioning

Identified Context Situations

The system successfully identified:

- Potential Collision Scenario
- Robot moving
- Object detected in front
- Trajectory indicates movement towards object → Inferred risk of collision
- Safe Movement Scenario
- Object detected behind
- Robot moving away → Safe continuation of motion

Outcome

The ontology-based reasoning correctly transformed sensor-level data into semantically meaningful contextual states. The identified context was provisioned to the SOPRANO message broker in structured Resource Description Framework (RDF) form.

2.4.2 CRF Pilot – Vehicle Door Disassembly

Experimental Setup

In the CRF shopfloor environment, the Context Monitoring component observed:

- HMO
- Worker presence
- Worker posture
- Worker proximity to robot
- NVL / VLM
- Robot localization in assembly workspace
- Robot state (ROS2-based interfaces)
- Active task

- Tool usage
- Motion state

Identified Context Situations

The Context Extraction Service successfully identified:

- Human–Robot Proximity Context
- Human detected within predefined safety zone
- Robot in motion → Context indicating safety-sensitive collaboration phase
- Tool Handling Context
- Robot in active disassembly operation
- Human entering shared workspace → Context requiring task re-evaluation or speed adaptation
- Idle / Standby Context
- No human in vicinity
- Robot in stationary mode → Safe autonomous task continuation

Outcome

The ontology model successfully represented:

- Human agents
- Robot agents
- Workspace relations
- Safety-relevant spatial configurations

The system demonstrated the capability to integrate structured perception data (pose, detection events) into unified semantic context descriptions applicable to collaborative assembly workflows.

2.4.3 E-TERRY Pilot – Symbiotic Farming Scenario

Experimental Setup

For the agricultural environment, monitoring included:

- GNSS-based localization
- Object detection (crops, obstacles)
- Human detection in field
- Robot operating mode (harvesting / unloading)

Interfaces were provided via ROS2-based communication.

Identified Context Situations

The Context Extraction Service identified:

- Field Obstacle Context
- Robot in harvesting mode
- Obstacle detected within trajectory → Context indicating required path adaptation
- Human-in-Field Context
- Worker detected within configurable safety radius
- Robot in active operation → Context indicating reduced-speed or pause condition
- Mode Transition Context
- Transition from harvesting to unloading mode → Context change affecting task allocation and navigation strategy

Outcome

The experiment demonstrated that the ontology-based context model can represent:

- Outdoor operational conditions
- Human posture and presence
- Multi-modal robot operational states
- Environmental variability

This confirms cross-domain adaptability of the context model beyond manufacturing and indoor environments.

2.4.4 Cross-Pilot Summary

Across the three pilots, the following capabilities were tested:

- Continuous monitoring of heterogeneous data sources (ROS2, OPC-UA, perception modules)
- Context change detection triggered by:
- Human appearance
- Object detection
- Robot motion changes
- Mode transitions
- Semantic integration of structured and unstructured data
- Context instantiation into OWL-based ontology
- Provision of RDF-based context descriptions to the SOPRANO message broker

The experiments confirm that the ontology-driven approach is:

- Domain-independent
- Extensible
- Compatible with heterogeneous communication protocols
- Suitable for runtime MH-MR coordination

2.5 Challenges and Future Work

Despite successful cross-pilot validation, the prototype remains at early maturity:

- Reasoning rules are currently rule-based and deterministic
- No probabilistic reasoning under uncertainty
- Limited ontology scope for complex multi-agent interactions
- No quantitative benchmarking (latency, scalability, throughput)
- Multi-robot simultaneous coordination not yet evaluated

The cross-pilot experiments demonstrate that:

- The Context Extraction architecture is technically feasible
- The ontology model is adaptable across construction, manufacturing, and agriculture
- Context provision can serve as decision-support input for task allocation, safety monitoring, and adaptive behaviour

Future development will focus on:

- Extension of ontology for richer pilot-specific semantics
- Optimization of ROS2 and OPC-UA monitors
- Integration of advanced context reasoning mechanisms
- Performance evaluation under increased system load

3 MH-MR Architecting Tools

3.1 Objectives

To realize multi-human and multi-robot application scenarios requires the composition of various models expressing knowledge, about the tasks of both robots and humans, their shared environment, and the robots and humans' capabilities in carrying out tasks within such an environment. Our composable modelling approach aims to support the specification and execution stages. The former is aimed for robot developers to express their requirements and assumptions whereas the latter is aimed for both humans and robots to bring the models into action. That is to simulate and assess scenarios and to use the models as a prior for decision-making. The models described in this section are used for both. This transition was initially demonstrated in the SESAME project, and is being extended here to other domains based on the SOPRANO use cases (i.e., construction).

To realize the KUKA use case, a model of the environment is necessary. Current approaches rely on Simultaneous Localization and Mapping (SLAM) approaches to build 2D representations of the environment for navigation. The use of Building Information Modelling (BIM) as foreseen in this use case, allows robots to exploit 3D information that has been modelled by domain experts at design time.

The composable model approach relies on linked data models, which enable us to model domain-independent information separately (e.g., environment and task knowledge), and compose them into a model. The graph-based structure and underlying semantics allow us to reason about relations between concepts (e.g., spatial information describing the environment and relevant for the waypoints in a navigation task), increasing the flexibility and reconfigurability of the environment models. This allows our tooling to support different environments but the same task, and different tasks in the same environment.

Our objectives are:

- **To identify the semantics in environmental models for the KUKA use case.** In the KUKA use case, the environment is modelled by end-users using BIM. These models are often represented using the IFC format[1].
- **To formalize and harmonize BIM models with the composable model representation.** Generating a composable model representation of the BIM models, and aligning the semantics of the IFC models with the FloorPlan DSL models developed in SESAME.
- **To develop meta-models to describe robot kinematics and dynamics.** Following the composable model approach, we define meta-models and models that describe the motion specification of a robot using JSON-LD.
- **To compose and generate environmental models.** Incorporating the composable representation of the IFC models into the current Scenery Builder pipeline.
- **To compose and verify motion specifications.** Composition of specifications of motion specifications and the appropriate controller architectures

3.2 Composable Models

This section describes Task 4.2 and focuses on the modelling, semantics and specification of the composable models. We highlight how the composable modelling approach contributes to the flexibility and reconfigurability of the MH-MR Architecting Tools (MAT) components.

[1] `https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/`

3.2.1 Composable FloorPlan Models and Building Information Modelling (BIM)

3.2.1.1 Updates to the FloorPlan Models In our approach, the environment is modelled using the FloorPlan DSL [14]. The language is defined using textX[2], and allows users to specify indoor environments in a text-based, human-readable way. The main concepts it utilizes to do so are: spaces (e.g., rooms), features (e.g., columns) and openings (e.g., entryways). Using Jinja[3] and textX's generators, the FloorPlan models were converted to their composable representation and an interpreter generated the 3D mesh and occupancy grid artefacts. Dynamic doors were modelled directly in JSON-LD in two models: an object and an instance.

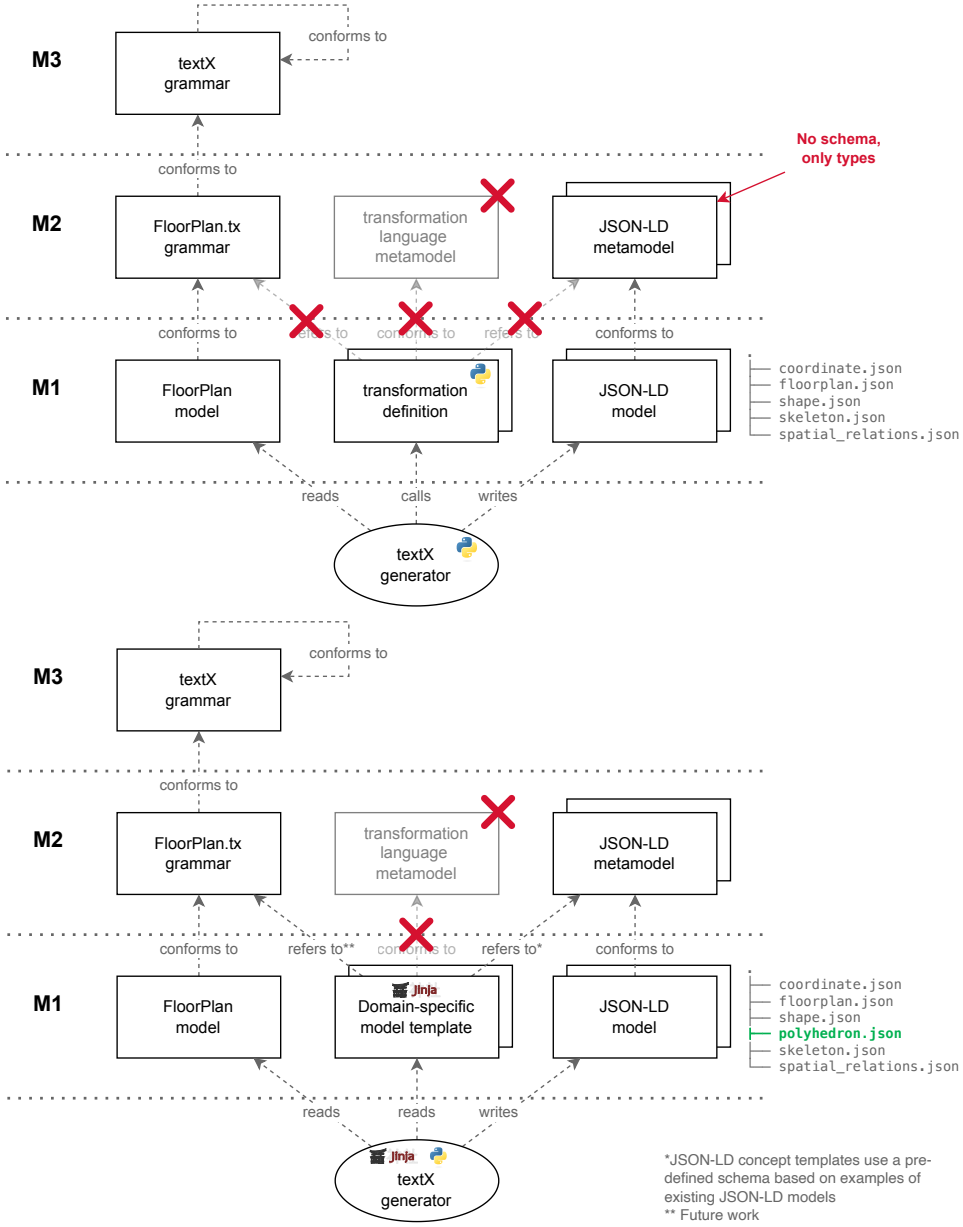


Figure 10: Model-to-text transformations for the FloorPlan models for v1 and v2, on the top and bottom, respectively

However, the tools developed for the SESAME project were proof-of-concept research prototypes. To incorporate the BIM models into our pipeline and increase their Technology Readiness Level (TRL), changes to the language and generators were necessary: First, the information about the 3D mesh was missing from its

[2] http://textx.github.io/textX/
[3] https://Jinja.palletsprojects.com/en/stable/

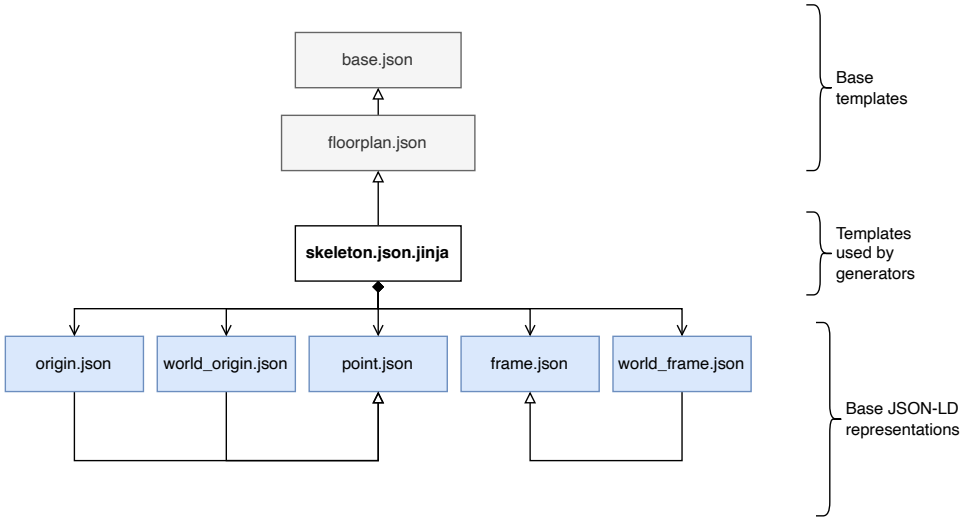


Figure 11: Systematic navigation of models using Jinja templates at different levels of abstraction

composable representation (as it was not required before) and is required to streamline the pipeline to use both FloorPlan and IFC composable models. Second, when using dynamic doors, because more detailed models were written manually, the FloorPlan models did not include them (just their entryways), although information about the geometry and pose of the doors was easier to specify in the Domain-Specific Language (DSL). Third, due to the complexity of the interpreter and generator code, some concepts were missing in some of the JSON-LD representations; the code used to implement the transformation was complex because the grammar and syntax did not align to the concepts in the JSON-LD metamodels. Finally, because JSON-LD metamodels do not impose one "true" schema, and because textX is the language used for the DSL, it was not possible to use other well-established model-to-model transformation practices (see Fig. 10).

To tackle these issues, we redesigned the language and took steps to systematize the Model-to-Text (M2T) transformations. First, we aligned the concepts to those in the composable representation as much as possible. We did so by introducing the concepts in the composable representation as syntactic sugar (i.e., users only need to optionally specify them, otherwise they are expanded with reasonable defaults). This had the added benefit of reducing the size of the specification, e.g. by not requiring the specification of rotations of 0 degrees. Second, we made the definition of concepts using the DSL consistent to increase its usability. Third, we optimized the Jinja templates used by the textX generators by using inheritance and inclusion to minimize duplication and systematically generate the composable representations. This also enabled us to systematically navigate the models, and prevent concepts from being left out. We did so by defining Jinja templates with separation of concerns, as seen in Fig. 11: JSON-LD document templates to make the structure consistent, domain-specific model templates that define which concepts will be included, and concept templates (one per concept) matching the JSON-LD representation of existing FloorPlan models. Finally, we added a generator to transform from the syntax in v1 to the new syntax in v2.

3.2.1.2 Alignment with IFC concepts with composable representation of the FloorPlan models

Although some concepts are present both in the FloorPlan DSL and the IFC models, their semantics differ to varying degrees. At the highest level, IFC models the spatial structure through *IfcSite* and *IfcBuilding*. The *IfcStorey* is what currently more closely represents the granularity of the FloorPlan concept; however the DSL currently only supports a single level. As a result, we can add concepts to describe the Building and multiple Storeys. In practice, these concepts for high-level spaces are not mandatory in IFC, but required by the FloorPlan, and thus would require IFC models to always include these concepts for the KUKA use case. Note that, for the generation of simulation artefacts (c.f. Sect. 3.2.2), next steps include the design decisions for the execution pipeline, since Gazebo worlds tend to be single-floors. Supporting multi-level environments is out of scope for this project.

The next set of concepts are *IfcSpace* and Space, which differ in their meaning, and *IfcWall* and Wall, which differ in their specification. In the FloorPlan DSL, Spaces are closed areas that have Walls; Walls are currently not specified directly, and are instead expanded during parsing to match the Space shape; this means that contiguous Spaces that share walls, which often means one space is specified relative to that shared Wall frame, both have a "separate" shared Wall which is merged at parsing. *IfcSpaces*, on the other hand, are not required (*IfcWalls* can be modelled independently), and are not necessarily bounded by walls. This also means that, even when they are representing closed areas, they are not required to have all the walls that bound them, e.g., it is possible that modellers assign shared walls of contiguous spaces only to one of them. To be able to perform the Model-to-Model (M2M) transformation, we can add an OpenSpace concept to the DSL and require that all rooms are modelled as *IfcSpaces* and related to their corresponding *IfcWalls*, including those that are shared with contiguous *IfcSpaces*. The update of the FloorPlan grammar allows us now to specify Walls within a Space, but an additional modification is required to be able to model Walls independent of a Space; in practice, the Divider concept has been used to model independent walls before, but without the same semantic meaning. A more thorough analysis of the implications of modelling Walls independently and is required to determine if it can be supported.

The DSL currently only supports static doors, which are modelled as Dividers; dynamic doors are currently only modelled in JSON-LD and have more details about their behaviour. *IfcDoors* specifications are richer. The parameters of the *IfcDoor* are specified in the *IfcDoorType* concept, including pre-defined types (*IfcDoorTypeEnum*) and how it operates (*IfcDoorOperationEnum*). In IFC, *IfcDoors* fill an *IfcOpening* using an *IfcRelFillsElement* relation. The geometry of an *IfcDoor* can be specified as its 3D profile or by using the profile of the *IfcOpening* element it fills. In the DSL, an Entryway — a special type of Opening — is modelled separately but has no semantic relation to the door that fills it; both the Divider and Entryway are specified with respect to the Wall, and their geometries are modelled independently. Similarly to doors, another type of *IfcOpening*, *IfcWindows* have much richer specifications in IFC and similar to *IfcDoors*, they fill an *IfcOpening*. *IfcWindows* model the actual object, while Windows on the FloorPlan only model the Opening. We have planned to add concepts and relations based on the IFC door representations to the DSL. Windows will benefit from some of these additions, but have a lower priority for making the Window models themselves richer as they are not as relevant for the KUKA use case.

IfcColums and Columns are generally the same, the only noticeable difference being the pre-defined types of *IfcColumns*. *IfcBeam* is a similar concept for horizontal, load-carrying elements, but this level of granularity is not currently required in the DSL. The *IfcSlab* provides the floor or ground, and while it is not modelled explicitly in the DSL, it matters for the simulation artefacts where the default ground floor from Gazebo is used. This is possible because the FloorPlan currently assumes that the floor is even and all the measurements in the z-axis are specified relative to it, assuming that its frame is at zero. Supporting uneven floors and floors of varying height, would require the addition of concepts such as *IfcStair* or *IfcRamp*, but are out of scope for the SOPRANO project.

To describe the locations and shapes of all the elements in a IFC model, the *IfcRepresentationItem* and its subclasses are used. Specifically, *IfcShapeRepresentation* describes the 3D geometry of an element, while *IfcLocalPlacement* describes its location with respect to an absolute frame or other element's frame. However, because there are many possible ways to model such relations and representations, we decided to narrow the possibilities by looking at the commonly used practices (cf. Sect. 3.2.1.3).

Finally, for the KUKA use case, in addition to the *IfcWalls* and other concepts described above, the concepts relevant to the three actions the robot should perform (drill, paint and grind) are (cf. Table 1). One key element, is the distinction between the object to be installed (outlet, duct, etc.) and the opening in the wall that the robot is tasked with milling. The milling task requires that the *IfcOpening* is explicitly modelled for two reasons. First, the geometry of the object to be installed may differ to that of the opening in the wall it requires. And, second, to perform the task, we need an explicit relation between the wall where the robot should mill and the object in the model from which we derive the goal. That must be modelled explicitly by an opening and wall through *IfcRelVoidsElement*, and by an object and opening through *IfcRelFillsElement* as objects cannot be related directly to a wall in a semantically-meaningful way with existing IFC concepts.

Concept	Description
IfcOutlet	Defines the object to be installed. It determines the location of the milling action in the use case.
IfcDuct, *IfcPipe*	Elements used to represent cable canalization to be installed and for which an opening must be milled.
IfcSwitchingDevice, *IfcJunctionBox*	Other IFC elements that could be used to represent an object to be installed for the milling task.
IfcOpening	While multiple objects can be represented in the IFC model, this element represents the resulting opening created by the milling task and it is thus the element used to describe the goal. Objects do not have relations to a *IfcWall*, so the relation mush be established by the *IfcOpening* and *IfcRelVoidsElement*, and an object relating to the opening with *IfcRelFillsElement*.
IfcSpaceBoundary, *IfcSpaceRelBoundary*	Describes the plane at which a room and an surface (e.g. of a wall or outlet) intersect. This is required to identify which face of a polyhedron is aligned to the inner wall where the robot should start a task (milling, painting or grinding)
IfcSurface, *IfcCurveBoundedPlane*	This concept describes the geometry of the *IfcSpaceBoundary* for non-wall elements (e.g., outlets or ducts). It is also relevant for the grinding and painting tasks, and could be used to define surfaces on walls that represent the goals for these tasks.

Table 1: IFC concepts relevant to the tasks of the construction domain use case

3.2.1.3 Domain analysis To better understand how end-users specify concepts relevant to the KUKA tasks (e.g., outlets, light switches, etc.) and how these concepts align to the FloorPlan models, we conducted a domain analysis of IFC models. We collected a total of 577 publicly available IFC files from GitHub (buildingsmart test files, toy examples, among others), the Duraark project (generated from point clouds obtained by a robot), and the openingdesign studio (an architectural studio with open models, including IFC). Note that openingdesign's files often are divided by domain, i.e., one file models the architecture of a building (walls) and separate files model the electrical system, HVAC and plumbing. A total of 41 files were not readable due to errors in the IFC schema or other parsing errors.

Of the remaining 536 files, we observe that about 50% use the IFC4 (or its variations) and the remaining 50% use the IFC2X3. We analyse the spatial structure elements to determine what is used in practice on the subset of files that model walls (including at least one *IfcWall*, *IfcStandardCaseWall* or *IfcCurtainWall*). The *IfcBuilding* concept is included on every file; *IfcSite* is not included in six models and six files do not specify any *IfcBuildingStorey* (they only have one file in common). In general, a single *IfcBuilding* is modelled per file, but multiple *IfcSites* may be modelled in it, which does not follow the underlying semantics of the IFC concepts, where *IfcSite* is higher in the hierarchy. Another aspect is the relation of *IfcSpaces* and *IfcWalls*. At a glance, our analysis indicates that many *IfcWalls* are often modelled independently.

Fig. 12 shows the IFC concepts used in practice. For the task-relevant concepts, only one file had outlets modelled using the *IfcOutlet* concept, three files modelled *IfcSwitchingDevices*, *IfcDuct* and *IfcDuctSegment* are present in 21 files, while *IfcPipe* and *IfcPipeSegment* are present in 23. *IfcRelSpaceBoundary* is present in 15 files, and *IfcCurveBoundedPlane* is present in 13.

What does this mean for the KUKA use case? We should not assume task-relevant information will be specified in the IFC model. Modelling this information in IFC models requires specialized training and software, and in practice it is effort intensive to manually model these details using Computer-Aided Design (CAD) and

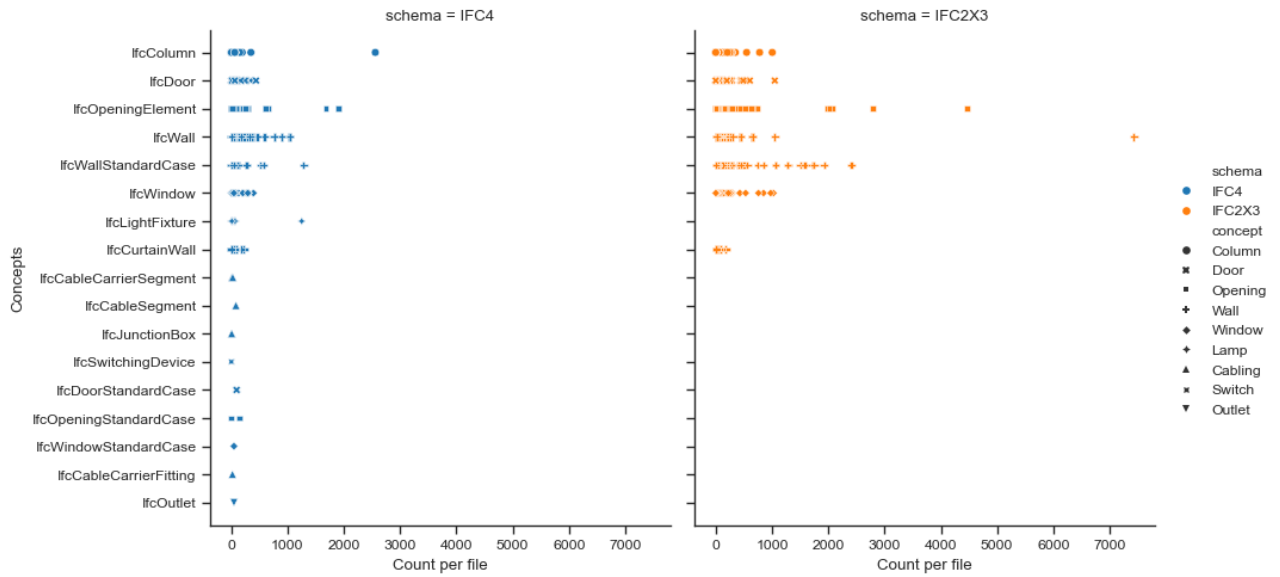


Figure 12: Concepts and their subclasses present in the IFC models

adding the necessary metadata to conform to IFC. Additionally, it would be limited to design-time, making it challenging and/or expensive to fix issues, make adaptations or specify new tasks on-site where the robot is deployed, and would require a more specialized operator or additional training in CAD software. Our proposal (c.f. Sect. 3.3.1) is to specify the high-level tasks using smaller, domain-specific composable models. This would make the specification or adaptation easier at deployment type, without requiring a CAD specialist. By using a text-based specification, the robotics company not only could potentially save the costs of expensive CAD licenses if the end-users provide the IFC files, but also use version control software, such as .git. Similarly to the FloorPlan, a DSL can be designed to facilitate the specification by end-users, and the composable representation can be generated at design or deployment-time. The composable approach allows us to exploit domain-specific knowledge (e.g. Building codes and standards) or allow the end user to specify and/or modify the task, increasing the flexibility and reconfigurability of the approach. Furthermore, the composition would allow us to automate some parts of the specification by adding quantifiers to the DSL to specify tasks, e.g. For all rooms, for any door, place an outlet to the right at 150cm (or at some other relative position).

Fig. 13 shows the types of *IfcShapeRepresentation* whose *IfcGeometricRepresentationSubContext* is *MODEL_VIEW* (the only 3D representation). We use these insights to prioritize which shape representations to support in the conversion in order to generalize the IFC generation pipeline.

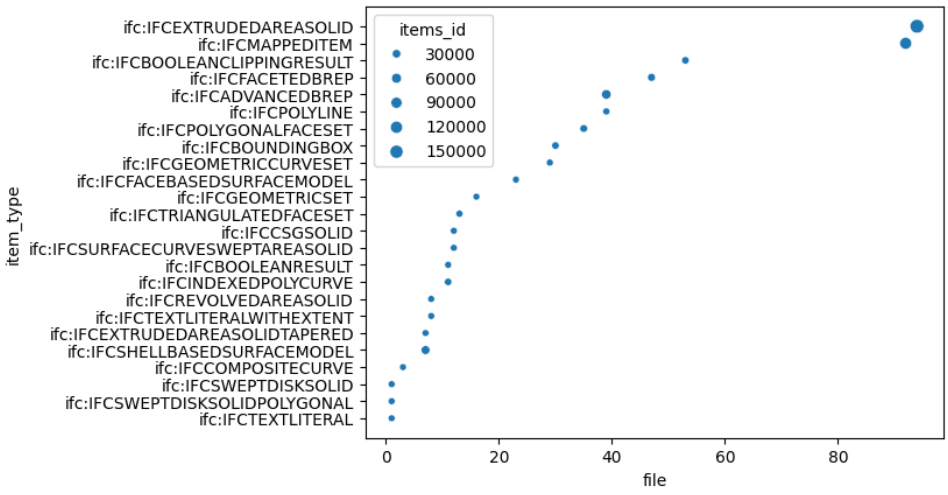


Figure 13: *IfcShapeRepresentation* concepts that have a 3D geometric subcontext in IFC models.

3.2.2 Composable Motion Specifications

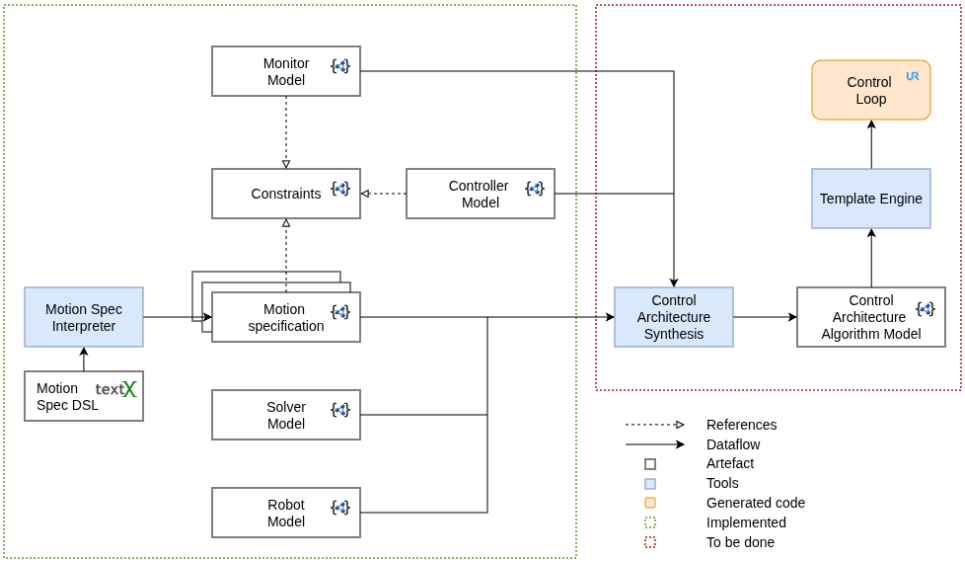


Figure 14: Motion Specification pipeline

Fig. 14 shows an overview of the planned model composition to describe robot motions. In the following, we describe the individual models and components, and what they represent.

A *Motion Specification* is a declarative description of a robot motion. It composes a set of constraints into three temporal categories: (i) pre-conditions (`when`), which must all be met before the motion begins; (ii) per-conditions (`while`), which must be maintained during execution; and (iii) post-conditions (`until`), where any single satisfied condition terminates the motion.

A `Constraint` is imposed on a *geometric coordinate quantity* — a scalar extracted from a higher-dimensional physical quantity such as a velocity twist, a wrench, or a pose. This scalar can represent a velocity component, a force or torque along a specific axis, a displacement, or an angle, depending on which physical quantity it is derived from. The constraint type then determines the admissible values of this scalar: an `EqualityConstraint` prescribes an exact reference value, a `UnilateralConstraint` imposes a one-sided threshold, and a `BilateralConstraint` keeps the quantity within a lower and upper bound.

Because each constraint acts on a single scalar, constraints are independent of one another and can be freely added or removed without affecting the rest. A six-dimensional end-effector task can therefore be specified by composing six one-dimensional constraints — one per controlled degree of freedom — each targeting a different scalar extracted from the same underlying twist or wrench. Adding, removing, or swapping a constraint changes only that one dimension without restructuring the others. This is the composability of the framework.

Monitors keep track of the pre- and post-condition constraints, triggering an event on a state transition (e.g., contact detection via a torque spike) or setting a flag while a state is maintained. The per-condition constraints are either monitored or controlled, referenced by a `ConstraintMonitor` or a `ConstraintController`. The `ConstraintController` additionally references a control signal and a controller model — for instance, a Proportional-Integral-Derivative (PID) controller that maps a velocity error to an acceleration energy fed into the Vereshchagin hybrid dynamics solver.

All of these models are represented in JSON-LD [20, 18], a data-exchange format that combines JSON with linked-data concepts from the semantic web. JSON-LD enables machine-interpretable graphs: every entity carries a unique identifier via `@id` and one or more semantic types via `@type`, and properties symbolically point to other entities by referencing their `@id`. The motion specification models described here have been published[19].

Listing 1: JSON-LD model of the find-screw guarded motion. Lines 5–9 compose six one-dimensional per-conditions constraining all end-effector DoF; lines 10–12 define the three termination conditions. The equality constraint on z angular velocity (lines 15–19) and its setpoint (lines 21–26) are shown below.

```
1 {
2     "@id": "motion-find",
3     "@type": "GuardedMotion",
4     "when": [],
5     "while": [
6         "cstr-angvel-ee-z-find",
7         "cstr-linvel-ee-x",  "cstr-linvel-ee-y",  "cstr-linvel-ee-z",
8         "cstr-angvel-ee-x",  "cstr-angvel-ee-y"
9     ],
10    "until": [
11        "cstr-torque-ee-z-shutoff", "cstr-rotation-ee-z-limit", "cstr-torque-ee-z-safety"
12    ]
13 },
14 {
15    "@id": "cstr-angvel-ee-z-find",
16    "@type": [ "Constraint", "EqualityConstraint", "AngularVelocityConstraint" ],
17    "quantity":        "angvel-ee-z",
18    "reference-value": "angvel-ee-z-find-ref"
19 },
20 {
21    "@id": "angvel-ee-z-find-ref",
22    "@type": [ "Quantity", "AngularVelocity" ],
23    "quantity-kind": "AngularVelocity",
24    "unit":  "RAD-PER-SEC",
25    "value": "3.665"
26 }
```

In JSON-LD, the motion specification is expressed as a `GuardedMotion` entity (line 3) whose `when`, `while`, and `until` arrays hold symbolic `@id` references to `Constraint` entities. Listing 1 illustrates the *find* motion, taken from the CRF use case of a find-and-unscrew task, in which the robot rotates its end-effector clockwise at $35\,\mathrm{min}^{-1}$ to engage the screwdriver tip with the screw head. The `while` list (lines 5–9) composes six one-dimensional equality constraints: the z angular velocity is controlled to the prescribed speed `cstr-angvel-ee-z-find`, while the remaining five dimensions (linear x, y, z and angular x, y) are held at zero. The `until` list (lines 10–12) terminates the motion when any one of three conditions is met: the engagement torque exceeds $4\,\mathrm{Nm}$ (shut-off torque, tip locked), the accumulated rotation exceeds $120°$ (shut-off angle, screw not found), or the torque exceeds $12\,\mathrm{Nm}$ (safety upper limit).

The representative per-condition constraint `cstr-angvel-ee-z-find` (lines 15–19) is an `EqualityConstraint` on the z angular velocity (line 17), referencing the setpoint `angvel-ee-z-find-ref` (line 18) whose value is $3.665\,\mathrm{rad/s}$ (line 25).

Listing 2: Motion Spec DSL equivalent of Listing 1. The `CONTEXT` block (lines 3–18) replaces the scattered world-model JSON-LD entities; constraint predicates reference context variables via dot-notation views.

```
1  ID:
2    motion_find
3  CONTEXT:
4    c1: World {
5          twist:  VelocityTwist,  // of=ee, wrt=world, as-seen-by=[ee]
6          wrench: Wrench,         // at=ee.origin,     as-seen-by=[ee]
7          pose:   Pose            // of=ee, wrt=world, as-seen-by=[world]
8        }
9    c2: Spec {
10         angvel_z_find: AngularVelocity = 3.665 rad/s,
11         vel_zero:      LinearVelocity  = 0.0   m/s,
12         angvel_zero:   AngularVelocity = 0.0   rad/s
13       }
14   c3: Post {
15         torque_shutoff: Torque = 4.0    Nm,
16         rotation_limit: Angle  = 2.0944 rad,
17         torque_safety:  Torque = 12.0   Nm
18       }
19 WHEN:
20 WHILE:
21   wi1: twist.angular.z  equal to Spec[angvel_z_find]
22   wi2: twist.linear.x   equal to Spec[vel_zero]
23   wi3: twist.linear.y   equal to Spec[vel_zero]
24   wi4: twist.linear.z   equal to Spec[vel_zero]
25   wi5: twist.angular.x  equal to Spec[angvel_zero]
26   wi6: twist.angular.y  equal to Spec[angvel_zero]
27 UNTIL:
28   u1: wrench.angular.z  greater than Post[torque_shutoff]
29   u2: pose.angular.z    greater than Post[rotation_limit]
30   u3: wrench.angular.z  greater than Post[torque_safety]
```

However, authoring JSON-LD requires end-user navigating multiple files and following symbolic `@id` references manually, and having domain knowledge. To address this, the same models are expressible in a DSL built with TextX[4], which collapses the scattered JSON-LD entities into a single structured block. Listing 2 shows the DSL equivalent of Listing 1. The `CONTEXT` block (lines 3–18) declares all geometric quantities that correspond to world-model entities in JSON-LD. The `WHILE` and `UNTIL` predicates (lines 20–30) reference these quantities via dot-notation views (e.g., `twist.angular.z` at line 21), replacing the explicit `View` entities and quantity models required in JSON-LD.

Additionally, for the KUKA use case, we have started modelling the motion specifications required for the milling task [27], focusing on two motions: circular and spiral milling. In both motions, the tool must maintain a constant contact force against the wall, stay within a depth tolerance band, and keep its orientation fixed by holding zero angular velocity around the tool's transverse axes. The key difference lies in how the radial dimension is handled: circular milling constrains the end-effector to a fixed radius while completing a full revolution, whereas spiral milling replaces the radius constraint with a constant outward radial velocity, continuously expanding the cutting radius until an outer limit is reached.

Listing 3 shows the JSON-LD model of both guarded motions for a concrete parameterisation. The circular motion (lines 2–12) revolves in the YZ plane at $0.5\,\mathrm{rad/s}$ (line 15) with the radius locked to $0.05\,\mathrm{m}$ via a `LinearDistanceConstraint` (line 21), a contact force of $15\,\mathrm{N}$, and a depth tolerance of $\pm 3\,\mathrm{mm}$ (lines 5–11); it terminates after a full revolution ($\theta \geq 2\pi$). The spiral motion (lines 27–37) replaces the radius constraint with a `LinearVelocityConstraint` for the outward radial velocity (line 46, $0.002\,\mathrm{m/s}$) while

[4] `https://textx.github.io/textX/`

Listing 3: JSON-LD model of the circular and spiral milling guarded motions for the KUKA use case. The circular motion (lines 2–12) constrains the orbital velocity and locks the radius; the spiral motion (lines 27–37) replaces the radius constraint with an outward radial velocity. Representative constraints for both phases are shown below each guarded motion.

```
1 {
2     "@id": "motion-circular-mill",
3     "@type": "GuardedMotion",
4     "when": [],
5     "while": [
6         "cstr-angvel-orbital-circular",
7         "cstr-dist-ee-radius",
8         "cstr-frc-ee-x-contact",
9         "cstr-pos-ee-x-depth",
10        "cstr-angvel-ee-y-zero",  "cstr-angvel-ee-z-zero"
11    ],
12    "until": [ "cstr-rotation-circular-complete" ]
13 },
14 {
15    "@id": "cstr-angvel-orbital-circular",
16    "@type": [ "Constraint", "EqualityConstraint", "AngularVelocityConstraint" ],
17    "quantity":        "angvel-orbital",
18    "reference-value": "angvel-orbital-ref-circular"
19 },
20 {
21    "@id": "cstr-dist-ee-radius",
22    "@type": [ "Constraint", "EqualityConstraint", "LinearDistanceConstraint" ],
23    "quantity":        "dist-ee-to-center",
24    "reference-value": "dist-ee-radius-ref"
25 },
26 {
27    "@id": "motion-spiral-mill",
28    "@type": "GuardedMotion",
29    "when": [],
30    "while": [
31        "cstr-angvel-orbital-spiral",
32        "cstr-radvel-ee-outward",
33        "cstr-frc-ee-x-contact",
34        "cstr-pos-ee-x-depth",
35        "cstr-angvel-ee-y-zero",  "cstr-angvel-ee-z-zero"
36    ],
37    "until": [ "cstr-dist-ee-spiral-limit" ]
38 },
39 {
40    "@id": "cstr-angvel-orbital-spiral",
41    "@type": [ "Constraint", "EqualityConstraint", "AngularVelocityConstraint" ],
42    "quantity":        "angvel-orbital",
43    "reference-value": "angvel-orbital-ref-spiral"
44 },
45 {
46    "@id": "cstr-radvel-ee-outward",
47    "@type": [ "Constraint", "EqualityConstraint", "LinearVelocityConstraint" ],
48    "quantity":        "radvel-ee-to-center",
49    "reference-value": "radvel-ee-outward-ref"
50 },
51 {
52    "@id": "cstr-frc-ee-x-contact",
53    "@type": [ "Constraint", "EqualityConstraint", "ForceConstraint" ],
54    "quantity":        "frc-ee-x-sensed",
55    "reference-value": "frc-ee-x-contact-ref"
56 }
```

revolving at $0.8\,\mathrm{rad/s}$ (line 40); the same contact force constraint is shared (line 52), and the motion terminates once the radius reaches $0.08\,\mathrm{m}$.

3.3 Architecting Tools

This section describes Task 4.3 and focuses on how the models described in the section above become executable. In the composable model approach, an architecture refers to the composition of (a subset) of models, and their use to generate the artefacts required for the execution of a task or tasks.

3.3.1 Scenery Builder

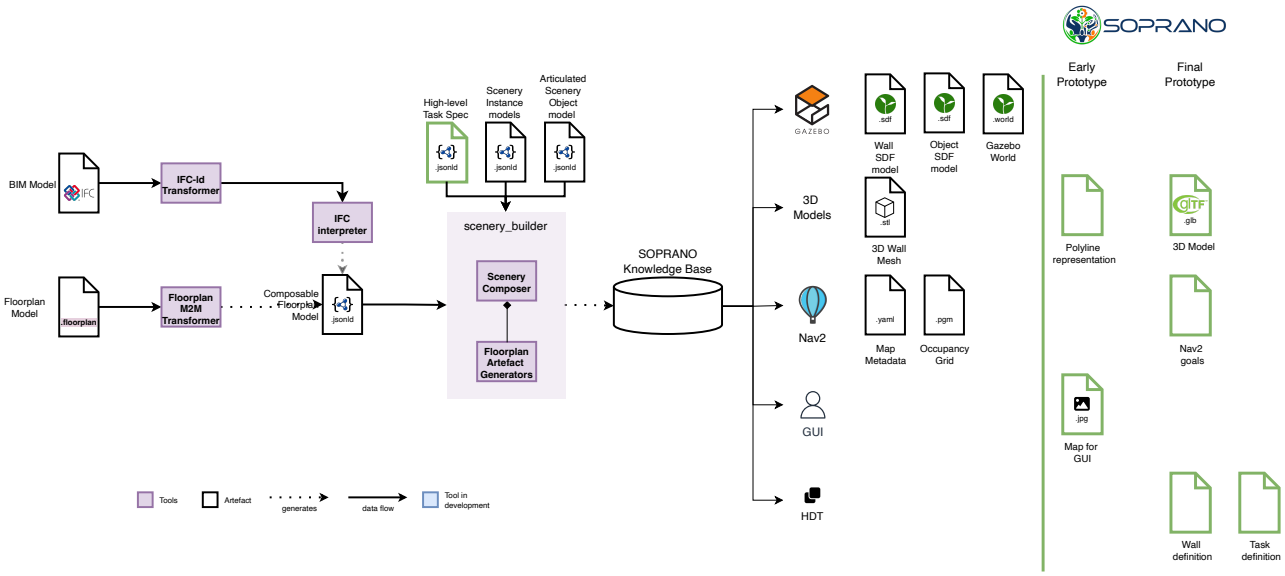


Figure 15: The updated composition and generation pipeline to integrate IFC models and high-level task specifications for the construction domain.

The scenery builder pipeline [13] leveraged the FloorPlan models specified with the textX DSL to generate executable artefacts. The models specified with the DSL were transformed into the composable representation and used to generate the occupancy grid map and 3D meshes. The scenery builder tools would take the composable representation, and additional object models and instances (e.g. to model doors) and compose the environment model, and use this specification to generate the missing artefacts required for execution: the Gazebo object and world models, and the navigation goals.

To incorporate the IFC models, the pipeline required significant changes. First, the collection of tools was refactored into a single repository[5], where the Jinja templates used for the model transformations were consolidated, and the Command-Line Interface (CLI) interface was made consistent. Second, to fully take advantage of the composable approach, the scenery builder tools needed to be updated to generate the 3D meshes, and occupancy grid maps from the composable representation instead of the FloorPlan models. Finally, the Gazebo plugins[6] that provided support for dynamic behaviours for the doors in simulation were refactored into a single repository, and updated for ROS2 Jazzy. The updated pipeline is shown in Fig. 15.

To facilitate the integration with other SOPRANO components, we added artefact generators using the FloorPlan models as a base to produce SOPRANO-specific artefacts. We added support for gltf 2.0 meshes that are generated using Blender. Additional image formats for the occupancy grid were added to the generation

[5] https://github.com/secorolab/scenery_builder
[6] https://github.com/secorolab/floorplan-gazebo-plugins

step so they could be included in the SOPRANO GUI. Next, we included support for two new models via queries to the composable representations of the floorplan and custom Jinja templates: A polyline representation that describes the 3D shape of the internal walls of a floorplan and conforms to the requirements of the VLM component; for the HDT component, we generate two JSON files: a wall definition that describes the walls as rectangular prisms and any openings in them (e.g., doorways). The navigation goals and the HDT task definition were only added once the IFC transformation was completed and are discussed later.

The next step in our implementation was to convert the IFC models to a JSON-LD format. Fig. 15 shows the **IFC-LD transformer** on the left side. Our implementation is inspired on an existing Part21 IFC parser and JSON-LD converter[7], but modified substantially to reduce model size, simplified to remove the web-based components, and fully exploiting the capabilities of JSON-LD by using it's compact document form, resulting in a linked-document model that is a 1:1 match of the IFC model. A small section of a transformed model is shown in Listing 4. This design decision was required for our future work, in case effects from the execution of the KUKA tasks needed to be updated in the model, and thus need to be transformed back into IFC syntax. The current prototype of the IFC-LD transformer is publicly available[8].

Listing 4: Subset of an IFC-LD model showing the object placement of an *IfcOutlet*

```
{
  "@id": "ifc-model:7785",
  "@type": "IFCLOCALPLACEMENT",
  "placementrelto": "ifc-model:7308",
  "relativeplacement": "ifc-model:7784"
},
{
  "@id": "ifc-model:7786",
  "@type": "IFCCARTESIANPOINT",
  "coordinates": [0.0, 0.0, -1.0]
},
{
  "@id": "ifc-model:7787",
  "@type": "IFCDIRECTION",
  "directionratios": [0.0, -0.0, 1.0]
},
{
  "@id": "ifc-model:7788",
  "@type": "IFCDIRECTION",
  "directionratios": [1.0, 0.0, 0.0]
},
{
  "@id": "ifc-model:7789",
  "@type": "IFCAXIS2PLACEMENT3D",
  "axis": "ifc-model:7787",
  "location": "ifc-model:7786",
  "refdirection": "ifc-model:7788"
},
{
  "@id": "ifc-model:7790",
  "@type": "IFCLOCALPLACEMENT",
  "placementrelto": "ifc-model:7785",
  "relativeplacement": "ifc-model:7789"
},
{
  "@id": "ifc-model:2978",
  "@type": "IFCOUTLET",
  "globalid": "3ICdJ_HFzBd948pehGKrRL",
  "name": "Outlet.007-Space.02-bottom",
  "objectplacement": "ifc-model:7790",
  "representation": "ifc-model:2990"
},
```

We integrated the **IFC Interpreter** on the scenery builder. Its main function is to perform a M2M transformation between the IFC-LD model and a composable model that conforms to the FPM concepts. The transfor-

[7] https://github.com/devonsparks/ifcld-service
[8] https://github.com/secorolab/ifcld

Listing 5: SPARQL query to obtain IfcOutlet, IfcDuct, IfcOpening and IfcWall based on their relations

```
SELECT DISTINCT ?wall ?opening ?object ?voids ?fills
WHERE {
    ?wall rdf:type ifc:IFCWALL  .
    ?opening rdf:type ifc:IFCOPENINGELEMENT  .
    ?object rdf:type ?object_type  .

    ?voids rdf:type ifc:IFCRELVOIDSELEMENT .
    ?voids ifc:relatingbuildingelement ?wall .
    ?voids ifc:relatedopeningelement ?opening .

    ?fills rdf:type ifc:IFCRELFILLSELEMENT .
    ?fills ifc:relatedbuildingelement ?object .
    ?fills ifc:relatingopeningelement ?opening .
}
```

Listing 6: A composable FPM model for the outlet in Listing 4

```
{
    "@id": "pose-coord-outlet-2978-milling",
    "@type": ["PoseReference", "PoseCoordinate", "DirectionCosineXYZ", "VectorXYZ"],
    "unit": ["UNITLESS", "MiliM"],
    "direction-cosine-x": [1.0, 0.0, 0.0],
    "direction-cosine-z": [0.0, -1.0, 0.0],
    "coordinates": [285.0, 2975.0, 500.0],
    "of-pose": "pose-outlet-2978-milling"
},
{
    "@id": "outlet-2978-milling-frame",
    "@type": "Frame",
    "origin": "outlet-2978-milling-origin"
},
{
    "@id": "outlet-2978-milling-origin",
    "@type": ["3D", "Euclidean", "Point"]
},
{
    "@id": "pose-coord-outlet-2978-milling",
    "@type": "PoseCoordinate",
    "as-seen-by": "placement-8052-frame"
},
{
    "@id": "pose-outlet-2978-milling",
    "@type": "Pose",
    "of": "outlet-2978-milling-frame",
    "with-respect-to": "placement-8052-frame",
    "quantity-kind": ["Angle", "Length"]
},
{
    "@id": "opening-5357",
    "@type": "Opening",
    "vectors": ["outlet-2978-milling-normal-to-boundary"],
    "voids": ["wall-1818"]
},
```

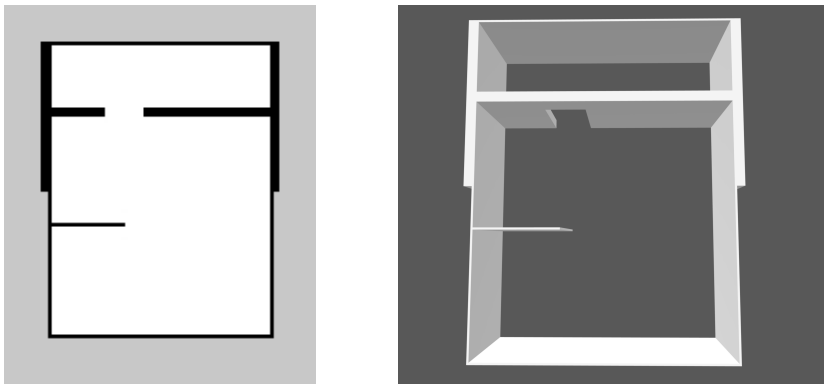


Figure 16: Occupancy grid and 3D mesh generated from IFC model of the KUKA test bed.

mation is based on SPARQL queries, such as the one shown in Listing 5, using the IFC semantics to obtain the relevant concepts for the transformation. An example of an interpreted model is shown in Listing 6. That is, the model itself is not a FloorPlan composable model, but instead uses the same concepts in the metamodel while it not needing to meet the same modelling assumptions and conventions (e.g., location of wall frames). This means, we can use the same pipeline as for the FloorPlan models to generate simulation artefacts, such as the occupancy grid and 3D mesh shown in Fig. 16.

Next, we added the final two models required for the SOPRANO use cases: The task definition used by the HDT component for the milling task that describes the geometry of the openings and their locations with respect to a world frame; and the navigation goals for the milling task, which add a translation to the outlet and duct poses so the robot can be correctly positioned for the milling action. The navigation goals also convert the representation to a Nav2-like format (i.e., from transformation matrices to position and orientation with quaternions as in the Nav2 messages) so developers can integrate it into their navigation stack more easily. Just like before, these models were created by querying existing elements in the composable representations and used custom Jinja templates to create the artefacts.

3.3.2 Control Architecture

Contact-rich manipulation tasks, such as screwing or working in close proximity to humans, require the robot to regulate interaction forces rather than simply track trajectories. A purely position-controlled robot would resist unexpected contact, risking damage to the workpiece or injury to a human co-worker. Compliant control addresses this by treating forces and velocities as first-class constraints, allowing the robot to yield appropriately when the environment pushes back. Depending on the task, this can be realised through impedance control, admittance control, or dynamics solvers such as the recursive Newton–Euler[8] or the Popov–Vereshchagin[15, 26] acceleration-constrained hybrid dynamics algorithm — each resolving a mix of Cartesian constraints and external forces into joint-space torque commands that satisfy both the task requirements and the robot's dynamics. Realising such compliant behaviour requires the robot to support joint torque control at the hardware level; the current toolchain therefore targets torque-controlled robots exclusively.

The control architecture describes how a motion specification is realised as executable control commands using such a solver. It connects the declarative constraint models to the controllers and solvers by defining the computations each controller performs, how their outputs are structured and passed as solver inputs, and the order in which all computations are carried out. The control architecture models thus fully determine the runtime behaviour of the robot for a given motion specification.

The control architecture implements the per-condition constraints from the motion specification (see Sect. 3.2.2). Listing 7 shows the full pipeline for the *find* phase. Each per-condition constraint is either *monitored* or *controlled*. For controlled constraints, a `ConstraintEvaluator`

Listing 7: JSON-LD control architecture for the *find* motion: error evaluator (lines 2–6), PID controller (lines 8–15), acceleration constraint and specification (lines 17–30), constraint handler (lines 32–44), motion drivers and prioritization (lines 46–57), and solver (lines 59–67).

```
1  {
2      "@id": "eval-angvel-ee-z-find",
3      "@type": [ "ConstraintEvaluator", "ErrorEvaluator" ],
4      "constraint": "cstr-angvel-ee-z-find",
5      "error":      "angvel-ee-z-err-find"
6  },
7  {
8      "@id": "ctrl-angvel-ee-z-find",
9      "@type": [ "Controller", "ProportionalIntegralDerivative" ],
10     "error-signal":      "angvel-ee-z-err-find",
11     "control-signal":    "eacc-ang-z-find",
12     "proportional-gain": "8.0",
13     "integral-gain":     "0.5",
14     "derivative-gain":   "3.0"
15 },
16 {
17     "@id": "acc-cstr-ee-ang-z-find",
18     "@type": [ "AccelerationConstraint", "AxisAligned" ],
19     "subspace":            "angular-acceleration",
20     "axis":                "z",
21     "acceleration-energy": "eacc-ang-z-find"
22 },
23 {
24     "@id": "spec-acc-ee-find",
25     "@type": "AccelerationConstraintSpecification",
26     "constraints": [ "acc-cstr-ee-ang-z-find",
27                      "acc-cstr-ee-ang-x",  "acc-cstr-ee-ang-y",
28                      "acc-cstr-ee-lin-x",  "acc-cstr-ee-lin-y",  "acc-cstr-ee-lin-z" ],
29     "attached-to": "link-ee"
30 },
31 {
32     "@id": "cstr-handler-find",
33     "@type": "ConstraintHandler",
34     "motion":      "motion-find",
35     "evaluators":  [ "eval-angvel-ee-z-find",
36                      "eval-linvel-ee-x",  "eval-linvel-ee-y",  "eval-linvel-ee-z",
37                      "eval-angvel-ee-x",  "eval-angvel-ee-y" ],
38     "monitors":    [ "mon-torque-ee-z-shutoff",
39                      "mon-rotation-ee-z-limit",
40                      "mon-torque-ee-z-safety" ],
41     "controllers": [ "ctrl-angvel-ee-z-find",
42                      "ctrl-linvel-ee-x",  "ctrl-linvel-ee-y",  "ctrl-linvel-ee-z",
43                      "ctrl-angvel-ee-x",  "ctrl-angvel-ee-y" ]
44 },
45 {
46     "@id": "drv-arm-find",
47     "@type": "MotionDrivers",
48     "acceleration-constraint": [ "spec-acc-ee-find" ],
49     "cartesian-force":          [],
50     "joint-force":              [],
51     "prioritization-hierarchy": [ "prio-find" ]
52 },
53 {
54     "@id": "prio-find",
55     "@type": "PrioritizationLevel",
56     "motion-drivers": [ "spec-acc-ee-find" ]
57 },
58 {
59     "@id": "slv-arm",
60     "@type": "SolverWithInputAndOutput",
61     "motion-drivers":   [ "drv-arm-find" ],
62     "output":           [ "pose-ee-world", "twist-ee-ee", "wrench-ee-ee" ],
63     "kinematic-chain":  "chain-arm",
64     "root":             "frame-world",
65     "gravity":          "earth-uniform-gravitational-field",
66     "solver":           "AccelerationConstrainedHybridDynamicsAlgorithm"
67 }
```

(lines 2–6) computes the error $e(t) = v_{set} - v(t)$, which feeds into a PID `Controller` (lines 8–15) whose output is an `AccelerationEnergy` (line 11). Each energy is associated with an `AccelerationConstraint` (lines 17–22) specifying the subspace and axis, and grouped per link into an `AccelerationConstraintSpecification` (lines 24–30). A `ConstraintHandler` (lines 32–44) assembles all evaluators, controllers, and monitors, and validates against the `GuardedMotion` (line 34) to ensure completeness. The `AccelerationConstraintSpecification` entities, together with any Cartesian or joint forces, form the `MotionDrivers` (lines 46–52), which are ordered by a `PrioritizationLevel` (lines 54–57). Finally, a `SolverWithInputAndOutput` (lines 59–67) maps all drivers to joint torques on the kinematic chain via the Popov–Vereshchagin acceleration-constrained hybrid dynamics algorithm.

Listing 8: Motion Spec DSL equivalent of Listing 7. The `CONTEXT` block (lines 3–10) declares the kinematic chain, root frame, and gravity; the `CONTROLLERS` block (lines 12–22) specifies one PID entry per per-condition constraint; the `SOLVER` block (lines 23–28) references these context variables and specifies the algorithm.

```
1 ID:
2   ctrl_find
3 CONTEXT:
4   c1: World {
5         chain: KinematicChain,  // chain-arm
6         root:  Frame            // frame-world
7       }
8   c2: Spec {
9         gravity: Gravity = [0.0, 0.0, -9.81] m/s2
10      }
11 MOTION: motion-find
12 CONTROLLERS:
13   ctrl-angvel-ee-z-find: PID {
14     constraint: cstr-angvel-ee-z-find,
15     Kp: 8.0,  Ki: 0.5,  Kd: 3.0
16   }
17   ctrl-linvel-ee-x: PID { constraint: cstr-linvel-ee-x, Kp: 5.0, Ki: 1.0, Kd: 3.0 }
18   ctrl-linvel-ee-y: PID { constraint: cstr-linvel-ee-y, Kp: 5.0, Ki: 1.0, Kd: 3.0 }
19   ctrl-linvel-ee-z: PID { constraint: cstr-linvel-ee-z, Kp: 5.0, Ki: 1.0, Kd: 3.0 }
20   ctrl-angvel-ee-x: PID { constraint: cstr-angvel-ee-x, Kp: 5.0, Ki: 1.0, Kd: 3.0 }
21   ctrl-angvel-ee-y: PID { constraint: cstr-angvel-ee-y, Kp: 5.0, Ki: 1.0, Kd: 3.0 }
22
23 SOLVER:
24   algorithm: Vereshchagin,
25   chain:     World[chain],
26   root:      World[root],
27   gravity:   Spec[gravity]
28
```

Listing 8 shows the DSL equivalent of Listing 7, following the same conventions as the motion specification DSL (Listing 2). The `CONTEXT` block (lines 3–10) declares the world entities and specification values needed by the solver: the kinematic chain and root frame as `World` quantities, and the gravity vector as a `Spec` value with concrete components in $\mathrm{m/s^2}$. The `MOTION` keyword binds this controller specification to `motion-find`, linking it to the per-condition constraints declared in Listing 2. The `CONTROLLERS` block (lines 12–22) declares one PID controller per per-condition constraint — identified by its constraint name — with proportional, integral, and derivative gains. The evaluator, error signal, `AccelerationConstraint`, and `AccelerationConstraintSpecification` are all implied by the constraint reference and generated automatically. The `SOLVER` block (lines 23–28) specifies the algorithm and references the chain, root, and gravity via the `World` and `Spec` context lookups, from which the `MotionDrivers`, `PrioritizationLevel`, and `SolverWithInputAndOutput` entities of Listing 7 are generated.

The control architecture generation tool consumes all models described above: the motion specification, controller models, solver specification, and a given robot model, and produces robot-executable code. The solver specification comprises the motion drivers (Cartesian forces and accelerations, joint forces), the solver type (ei-

ther the recursive Newton–Euler or the Popov–Vereshchagin acceleration-constrained hybrid dynamics solver), and the gravity specification.

For a given motion specification, the generation tool loads all models into an RDF graph, queries the required information via RDFLib and SPARQL, and generates an intermediate *control architecture algorithm*. This captures all data structures, processing functions, and the execution schedule, i.e. the order in which functions must be called, for every constraint, monitor, controller, and solver in the motion phase. The intermediate algorithm is then passed to a template engine that generates the final robot-executable code. Further details are available in the tutorial[9].

3.3.3 Experimental Setup at Uni-Bremen

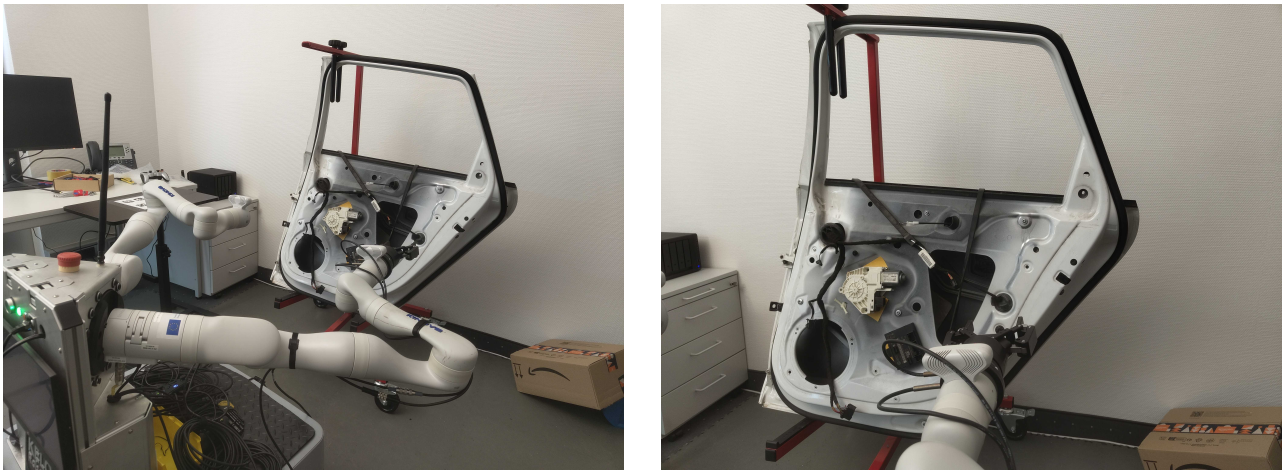


Figure 17: Experimental setup for the CRF unscrewing use case at Uni-Bremen. A dual-arm Kinova Gen3 is positioned in front of a car door carrying screws, electrical components, and cables.

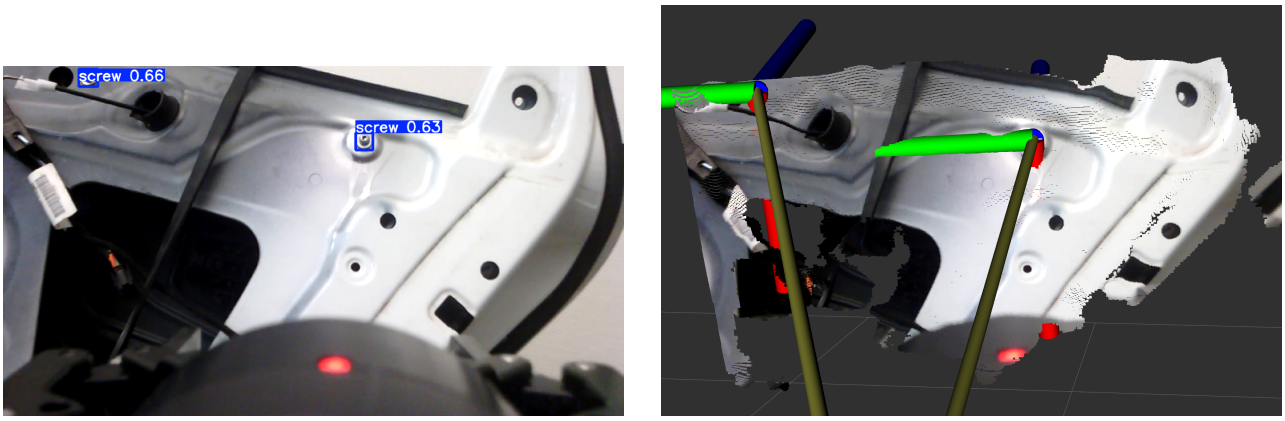


Figure 18: Perception results for the CRF use case: RGB-based screw detection on the car door (left) and 6D pose estimation of the detected screws (right).

The CRF use case targets the disassembly of a car door, which carries screws, electrical components, and cables. The task requires the robot to detect the pose of each screw and execute the find-and-unscrew motion using the motion specifications and control architecture described above. Fig. 17 shows the experimental setup at Uni-Bremen, where a dual-arm Kinova Gen3 manipulator[10] is positioned in front of a car door mounted on a

[9] https://secorolab.github.io/motion-spec-ral/tutorial.html
[10] https://www.kinovarobotics.com/product/gen3-robots

mobile-base. Fig. 18 shows results from the perception pipeline, which combines RGB-based screw detection with 6D pose estimation to provide the robot with the target locations and orientations needed to initialise each unscrewing motion.

3.4 Results

An overview of the improvements to our approach is listed below.

3.4.1 Composable models

- **Alignment of IFC and FloorPlan DSL concepts**. We conducted a domain analysis of publicly available IFC models using the IFC format to identify used concepts and modelling practices.
- **Composable motion specification models**. We developed and validated JSON-LD based models covering the full pipeline from declarative constraint specifications through constraint handlers and controller models to solver specifications.
- **Specification of the high-level tasks for the KUKA use case.** Models to describe the navigation goals where the robot will perform an action with respect to features of the environment and are part of the IFC-LD to composable model interpretation. Those same models can be specified manually and composed into the graph instead.
- **Motion Specification DSL**. We implemented a Motion Specification DSL in TextX that allows motion specifications to be expressed in a concise, human-readable form, collapsing the scattered JSON-LD entities into a single structured block.
- **CRF find-and-unscrew motion specification models**. We specified and validated the full set of motion specification models for the CRF find-and-unscrew use case in JSON-LD[11].

3.4.2 Architecting tools

- **Updates to the composition and artefact generation of environmental models**. To add IFC models to the pipeline, the generation of all execution artefacts is being updated to use the composable representation as a basis. The tooling also required updates to increase its TRL level, particularly improving its usability and maintainability to facilitate its adoption.
- **Generation of SOPRANO-specific artefacts from FloorPlan Models**. We added models to support the VLM and HDT components in SOPRANO from FloorPlan models. This demonstrates the ease of use of FloorPlan models and the scenery builder on development workflows. Once the IFC conversion was added, minimal changes were required to generate all the required artefacts from IFC models instead.
- **Transformation of IFC models into a JSON-LD representation**. Based on the draft IFC-LD specification and available tool, we transform existing examples of IFC models into a composable representation using the IFC-LD Transformer.
- **Implementation of the IFC-LD Interpreter**. This component in the scenery builder converted the IFC-LD representation to FPM concepts so the same artefact generation pipeline and queries could be used to generate artefacts from IFC.
- **Control architecture generation tooling**. We added tooling for the code generation of the control loop for a given motion specification. The generation exploits the graph structure of the composed models to extract the relevant information and generates the execution code using string templates in `C++`.

[11] `https://github.com/secorolab/motion-spec-ral/tree/crf/models/21-crf`

3.5 Challenges and Future Work

We summarize the next steps for T4.2 and T4.3 in the subsections below.

3.5.1 Composable models

- Adding relevant concepts from IFC representations to the FloorPlan DSL to enrich the floorplan elements that can be specified with it.
- Design of the composable representation for the Variation DSL concepts.
- Consolidation of the controller and Motion Specification DSL models. These models were developed in parallel, and thus require some minimal alignment to make their specification consistent.
- Specification of complete models for CRF use-case using the Motion Spec DSL and generating JSON-LD models from them.

3.5.2 Architecting Tools

- Extension to the IFC transformation pipeline to cover multi-storey buildings
- Executing defined motion specifications for the CRF use-case on the UR10e/UR20e robot at Profactor and on Kinova Gen 3 robot in Uni-Bremen.
- Integration of the motion specifications into the task execution for the CRF use case.
- Finalizing the integration of the architecting tool with other SOPRANO components for the final prototype evaluation, including new model transformations or adjustment to the existing artefact generation pipeline.

4 AI Model Optimizer & MLOps Orchestrator

4.1 Objectives

The objective of the AI Model Optimizer (AIO) component is to facilitate efficient distributed inference by automatically partitioning deep neural network models and assigning the resulting submodels to appropriate devices in a heterogeneous environment in order to minimize inference time. Additionally, the component generates a deployment plan for the distributed inference pipeline, supporting both x86 and ARM device architectures.

4.2 AI Model Optimizer

The AI Model Optimizer (AIO) is a component that performs split computing on designated AI models and finds the desired hosts in order to execute the generated distributed inference pipeline with minimum inference time for each submodel. The AI Model Optimizer (AIO) component collaborates with the MLOps orchestrator to enable distributed model deployment.

For the splitting, the main methodology is a model-distributed horizontal approach that identifies critical connection points in the DNN computation graph as potential splitting points, using Tarjan's algorithm [22] for bridge detection. A bridge is defined as an edge whose removal will result in increasing the number of connected components in the graph; in other words, it partitions the graph. To reduce the inference time of each submodel, the AI Model Optimizer employs a profiling mechanism based on an XGBoost predictor. This model estimates inference time using both model-specific features and cluster resource metrics. It is trained primarily on computation-intensive CNN architectures for computer vision. The training dataset was constructed from a range of state-of-the-art deep learning models in ONNX format [24] covering key vision tasks, including image classification, object detection, and 6D pose estimation.

The profiling mechanism mainly assigned submodels to devices based on their predicted inference latency. To take into account the network latency between the desired hosts, the same sub-component employs a dynamic programming methodology. For each submodel–device pair, it constructs a feature vector and predicts the execution time, forming a matrix $t_{i,k}$, where i denotes the submodel and k denotes the device. The profiler builds network matrices: latency $L_{k,j}$ and bandwidth $B_{k,j}$ between devices. Additionally, it estimates output data sizes S_i, which determine the communication cost between consecutive submodels.

Mathematically, the objective is to assign each submodel i to a unique device a_i (without reusing devices) in a way that minimizes the total latency. The overall cost can be expressed by Eq. (1).

$$\text{Total Latency} = \sum_{i=1}^{n} t_{i,a_i} \; + \; \sum_{i=2}^{n} \left(L_{a_{i-1},a_i} + \frac{S_{i-1}}{B_{a_{i-1},a_i}} \right) \tag{1}$$

To solve this optimization problem, the Profiler employs a Dynamic Programming (DP) algorithm that searches for the globally optimal assignment. The approach needs to know the submodels, their predicted inference times and the available devices. The algorithm keeps track of all possible combinations of used devices (via bitmasking) and the last device selected, ensuring that the no-reuse constraint is satisfied while minimizing total latency.

The architecture of AIO is illustrated in Fig. 19. The intended pipeline begins with a developer pushing an ONNX model to the model registry. The desired input of AIO is a list of devices and their resource usage, a list of their bandwidths and latencies and a set of minimal model characteristics. Developers should trigger the Model Splitter to partition the model. The resulting submodels are stored in the registry and passed to both the model characteristics extractor and the profiler, which predict the appropriate hosts for executing each submodel. Finally, the framework matches submodels to hosts and generates a Docker Compose file that serves as the deployment plan. The framework can complete the deployment plan by selecting an appropriate combination of Docker images that run on either x86 or ARM devices, depending on the architecture of the chosen optimal devices.

All of the aforementioned components are containerized with Docker. For the purposes of the project, the outputs of AIO can be used by the MLOps orchestrator in order to assist the non-technical user to perform the deployment. The AIO could potentially be used as a stand-alone system without the usage of the MLOps orchestrator.

As part of the same task, a stand-alone component called the quantizer (Fig. 20) was developed for the Open Call projects. The quantizer accepts an ONNX model along with a representative input dataset and compresses the model using static quantization. Specifically, it optimizes convolutional layers and weights by converting a floating-point (FP32) model into a more efficient 8-bit integer (INT8) model. The provided dataset is used for calibration and fine-tuning during the quantization process. The tool was evaluated using the yolov4.onnx model, and the dataset selected for fine-tuning was COCO 2017.

4.3 MLOps Orchestrator

The MLOps Orchestrator (MLO) provides a modular execution and orchestration environment for data workflows that is built for reusability, explainability and governance. Its architecture is organised into four conceptual layers: the Monitored IoT System Layer, the Data Orchestration & Applications Layer, the Core Services Layer and the User Interfaces Layer.

The Monitored IoT System Layer forms the data-producing edge of the architecture and includes all physical and virtual systems that act as sources of data within a deployment. It typically comprises sensors and machines (e.g. industrial devices and controllers), robotic systems such as ROS-based agents, digital twins and simulated processes, human inputs (e.g. manual stations, quality checks, overrides) and external data services or repositories (e.g. Enterprise Resource Planning (ERP), Manufacturing Execution System (MES),

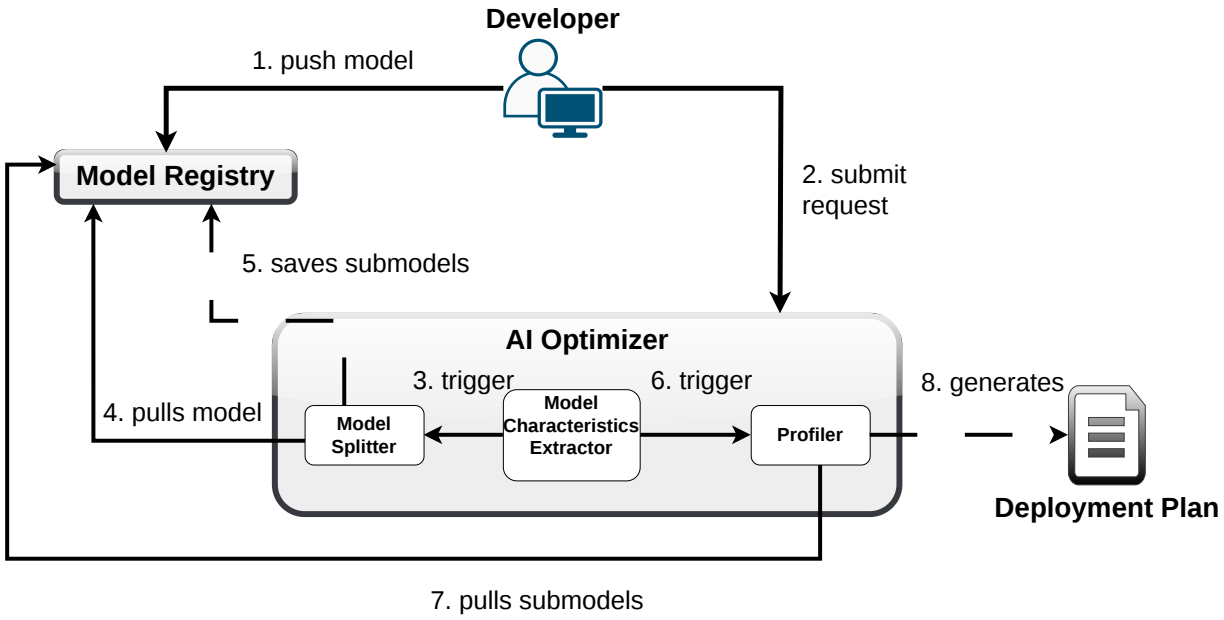


Figure 19: AIO Architecture

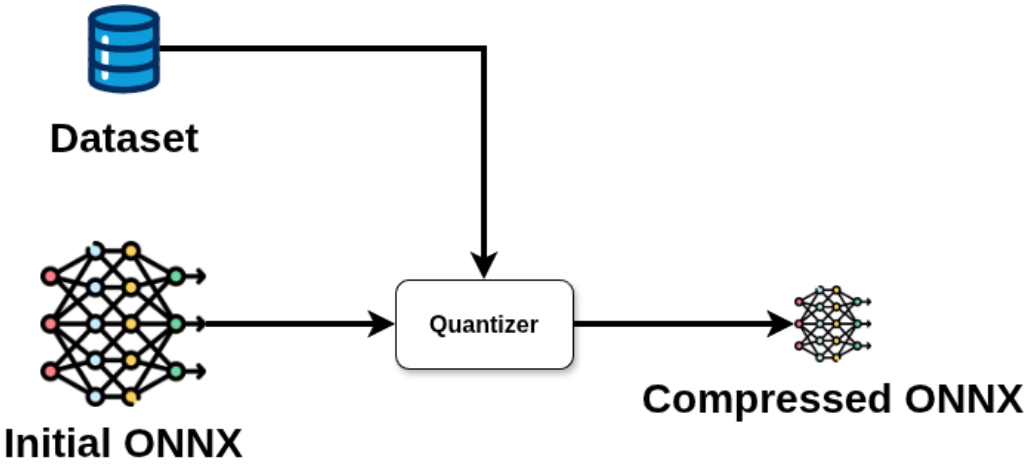


Figure 20: Quantizer

weather APIs, cloud-based Geographic Information System (GIS) platforms). Communication between this layer and MLOps Orchestrator (MLO) is protocol-agnostic and extensible: data can be pushed into the system via MQTT, REST APIs or WebSockets, or pulled periodically from repositories, with interactions managed through a Data Bus that acts as a communication backbone using a publish/subscribe model.

The Data Orchestration & Applications Layer is the core execution environment where data is collected, transformed, annotated and processed. At its heart is the Multi-Purpose Processing Engine (MPPE), which drives the scheduling, coordination and runtime management of workflows in cooperation with Apache Airflow. Workflows, defined through the Workflow Management Editor (WME), specify how digital resources (e.g. sensors, APIs, files) and processors (e.g. analytics, transformations, model inference) are connected and executed across distributed infrastructure. This layer is responsible for data collection, routing, transformation and semantic annotation, using out-of-the-box processors such as Logstash as well as user-defined processors for protocol translation, cleaning, enrichment, unit conversion or domain-specific logic.

The Core Services Layer supports the platform with shared services and design patterns. It includes the API and service gateway, identity and access management (based on Keycloak), policy enforcement components, and

central registries for digital resources, processors, workflows, Directed Acyclic Graph (DAG) and assets. These services implement a layered, microservices-oriented architecture with publish/subscribe communication and plug-and-produce capabilities, so that new processors or resources can be dynamically added and reused.

The User Interfaces Layer exposes MLO functionality through web-based tools. The central element is the Workflow Management Editor (WME), which offers a graphical interface that allows users to model digital resources, register processors and design workflows without writing code. WME includes a Warehouse view (for managing workflows, digital resources, workers, data kinds and processor definitions), a Lab view with tabs for workflow properties and visual flow creation, and an Airflow view for inspecting and managing DAG execution. Access and authentication are handled through integration with Keycloak, and role-based access control is applied to protect operations.

The underlying data and registry models are structured into three logical groups: resource management, data processing engine management and observations. The resource management group introduces entities such as Data Kind, Data Interface Type and Digital Resource, which are used by the Digital Resource Registry to describe and access data sources. The data processing engine group defines Processor Definition, Processor Manifest and Processor Orchestrator entities, which describe processor types, concrete instances and processor pipelines and are managed in the Data Processing Engine (DPE) and Directed Acyclic Graph DAG registries. The observations group defines the Observation entity as the canonical dataset produced by a digital resource, referencing the corresponding Digital Resource and Data Kind and capturing the main attributes of each measurement.

MLO supports onboarding new processors as containerised components, using two main methods. In Method A (Local Deployment – Modules Folder), Docker images and docker-compose configurations for a processor are placed in a designated modules folder; the platform then runs these services and they can be registered as processors. In Method B (Remote Deployment – Docker Hub), processors are provided as images in a remote container registry and referenced directly by MLO. In both cases, Processor Definitions and Manifests are created to describe metadata, input and output data kinds, parameters and resource requirements, after which the processors become available as reusable building blocks within the Workflow Management Editor.

4.4 Results

The AIO methodology was assessed using the OBP 6D component of the CRF use case. The monolithic version of OBP performs on-demand 6D pose estimation from an RGB image and 3D CAD models of the objects of interest in two stages: (1) DNN inference using DeepLab and (2) post- processing with RANSAC to extract the 6D pose.

The DeepLab model was supplied to AIO for partitioning, producing two submodels that are stored in the model registry and retrieved by two generic containers during deployment. The containers can communicate via ROS2. The distributed implementation of the DeepLab model is intended to serve as an alternative to the original version, allowing developers to select the appropriate version for deployment within SOPRANO. In essence, it replaces the original version of the model, interfaces with the ROS node server to access the camera stream, and forwards the inference results to the post-processing stage (Fig. 21). The distributed inference and post-processing stages are implemented as ROS2 services orchestrated by a ROS2 action server which can be triggered using the same interface as the monolithic OBP component.

Table 2: Inference performance on Jetson AGX Orin

Model	Avg. CPU (%)	Avg. RAM (%)	Inference Time (s)
DeepLab	10.8	10.9	5.6
DeepLab Part 1	7.3	5.7	5.2
DeepLab Part 2	1.4	4.8	0.3

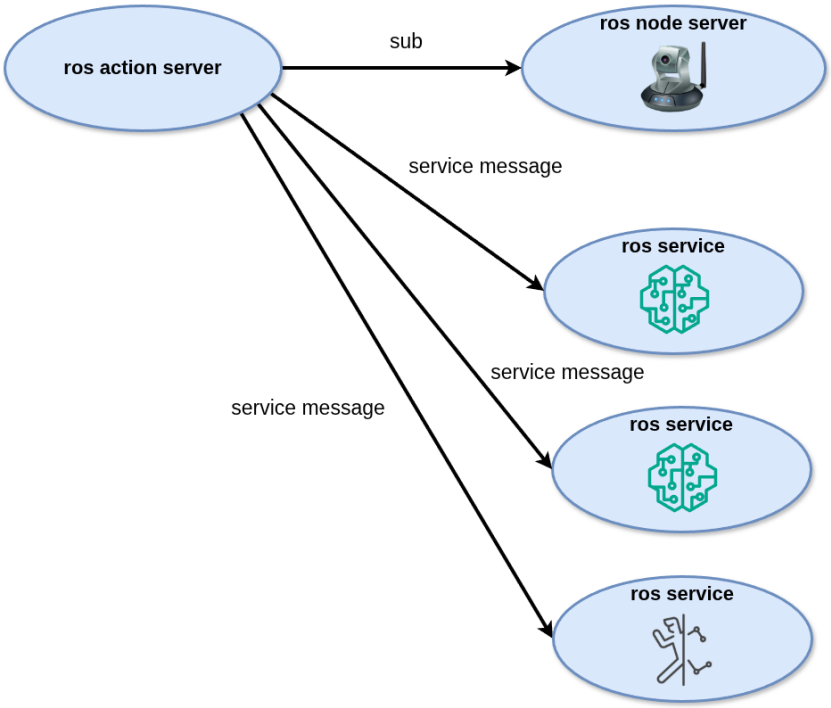


Figure 21: Distributed Inference

By executing the generated distributed inference pipeline 100 times on a Jetson AGX Orin (on CPU) running on 30W, the results presented in Table 2 demonstrate the potential of the splitting methodology to produce model partitions that reduce resource utilization. The profiling methodology was evaluated separately on a simulated hardware cluster. When tested with 30, 60, and 90 devices, it achieved near-stable response times of 13.95 s, 12.89 s, and 13.30 s, respectively, demonstrating its applicability to real-world device clusters.

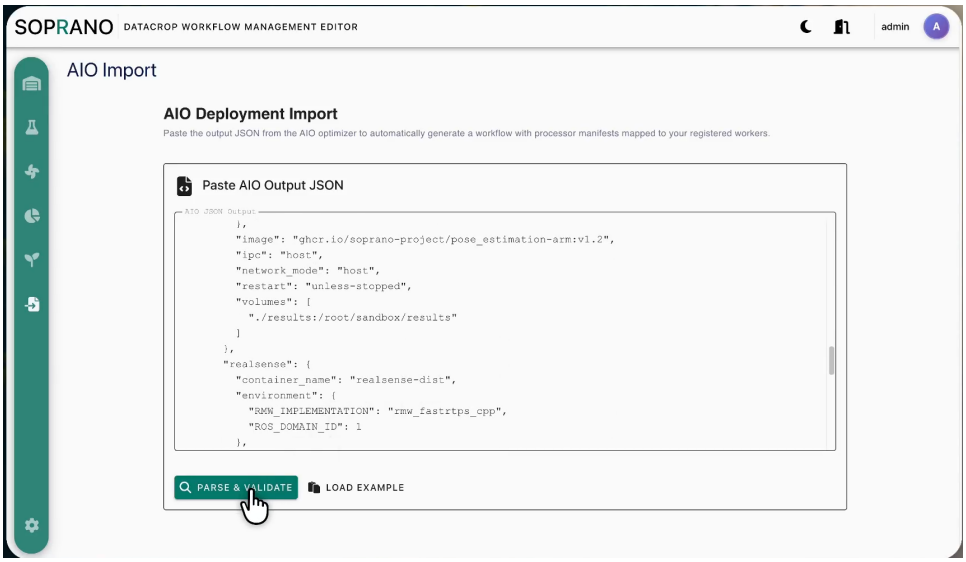


Figure 22: MLO–AIO import

The MLO is deployed and operational in the KUKA and E-TERRY pilots, where it is integrated with the Context Extraction Module (CEM) in the already deployed data pipelines. Using the Workflow Management Editor, pilot partners configure workflows that register context-enriched observations produced by CEM as digital resources and route them via Logstash-based preprocessors and Elastic/Kibana processors to monitor-

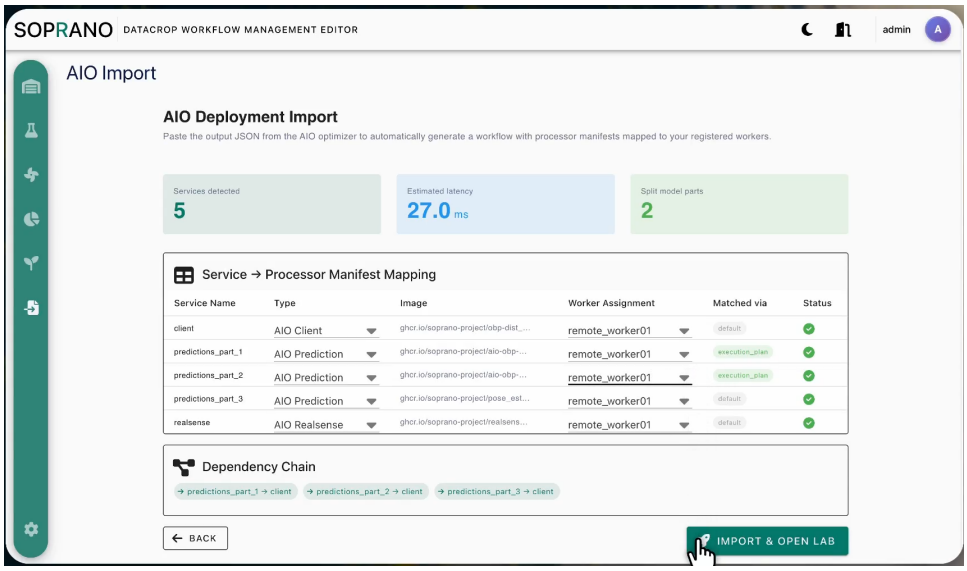


Figure 23: MLO–AIO Processor Manifest Mapping

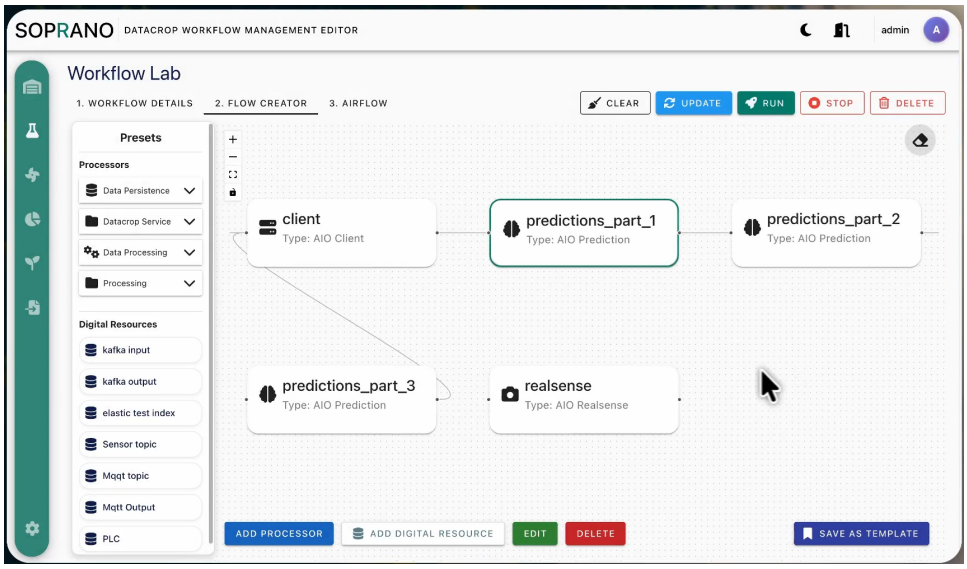


Figure 24: MLO–AIO flow creator

ing and analytics dashboards, validating the usability of the graphical interface for non-expert users and the correctness of the underlying registries, scheduling, and worker-pool orchestration. .

In the CRF pilot, the integration between the AI Model Optimizer (AIO) and the MLOps Orchestrator is being prepared around the OBP 6D component: AIO partitions the DeepLab model into sub-models and stores the resulting containers in the shared artifact registry, and MLO references these artifacts in workflows that orchestrate the ROS2-based distributed inference and post-processing stages alongside other SOPRANO components.

Processor onboarding supports both local deployment (modules folder with `docker-compose.yml`) and remote images (Docker Hub), enabling reuse of custom or third-party algorithms as WME building blocks. The platform provides complete Docker Compose deployment documentation and is validated on local hardware across the SOPRANO use cases.

4.5 Challenges and Future Work

As future work related to AIO, we plan to conduct experiments using the YOLO ONNX model from the E-TERRY use case and finalize its full integration with the MLOps Orchestrator. Additionally, a final enhancement of AIO will involve incorporating a dynamic programming approach to compute the optimal total inference time using GPUs instead of CPUs for distributed inference components. The main challenge of this part is to find tools that can measure GPU usage across different device architectures.

Future work related to MLO includes implementing OPC-UA communication support within MLO-orchestrated pipelines to further facilitate integration with SOPRANO industrial use case requirements.

5 Robot Capabilities Implementation

5.1 Objectives

The aim of this task is to create a strong framework for incorporating robotic abilities that facilitate the performance of different non-continuous actions, including grasping, holding, and disposing. This framework aims to translate high-level task plans (e.g., those from MH-MR task allocation in Task T2.6) into concrete robotic actions, guaranteeing smooth and accurate performance in dynamic industrial environments. An essential aspect of the framework will be its capability to adjust parameters that are not directly specified by high-level plans but are vital for task performance, including grasp locations. Furthermore, it will facilitate modular and reusable action frameworks, enabling adaptability and expandability across various tasks and robotic systems(UR10, Felice Robot [12] and KUKA Robot). In the CRF use case, both the task execution framework and the manipulation capabilities are implemented, while in the KUKA use case the primary emphasis is placed on the development and validation of the task execution framework. The main emphasis for development in this task, guaranteeing that the framework tackles practical issues and is customized to meet the particular needs of the car door disassembly procedure.

5.2 Task Execution Framework for Robot Capabilities

A task execution framework in this context refers to a structured control architecture, based on Behavior Trees (BTs) [7, 21, 16], that systematically translates high-level task plans (e.g., from MH-MR task allocation) into executable robotic actions such as grasping, holding, and unscrewing. The framework organizes these actions into modular and reusable components, enabling flexible sequencing, monitoring, and adaptation during execution in dynamic industrial environments. A key capability of the framework is the refinement and parameterization of task-relevant variables that are not explicitly defined at the planning level—such as grasp locations—ensuring robust and accurate task performance. By integrating perception, control, and decision-making within a unified Behavior Tree-based execution model [2], the framework supports scalability and interoperability across different robotic platforms (e.g., UR10, Felice, and KUKA systems). Furthermore, it is designed with a strong emphasis on addressing practical deployment challenges and aligning with the specific requirements of industrial use cases (CRF use case, KUKA use case), while maintaining extensibility for future tasks and system configurations. The state-of-the-art review for the Robot Capabilities Implementation was already presented in D4.1 (Modular, Reconfigurable, and Flexible Tools I, Section 5) and is not repeated here. This section focuses on the iterative developments, integration progress, and validation results achieved since the early prototype.

The Task execution framework has been designed considering different layers of execution as shown in Fig. 25. These different layers of execution are namely, Behavior Tree Run-Time Layer (BTRTL), Skill Implementation

[12] The felice robot was develoed within the FELICE EU project https://www.felice-project.eu/, by Accrea, a project partner in the felice project https://accrea.com/

Layer (SIL), Control Layer (CL) and Robot Layer (RL).
The BTRTL is responsible for breaking down commands from MH-MR Task allocation (T2.6) into parameters for execution and triggering a specific task based behavior tree. The BTRTL is a run time component i.e. it just executes the behavior trees designed at run-time using MRC component.

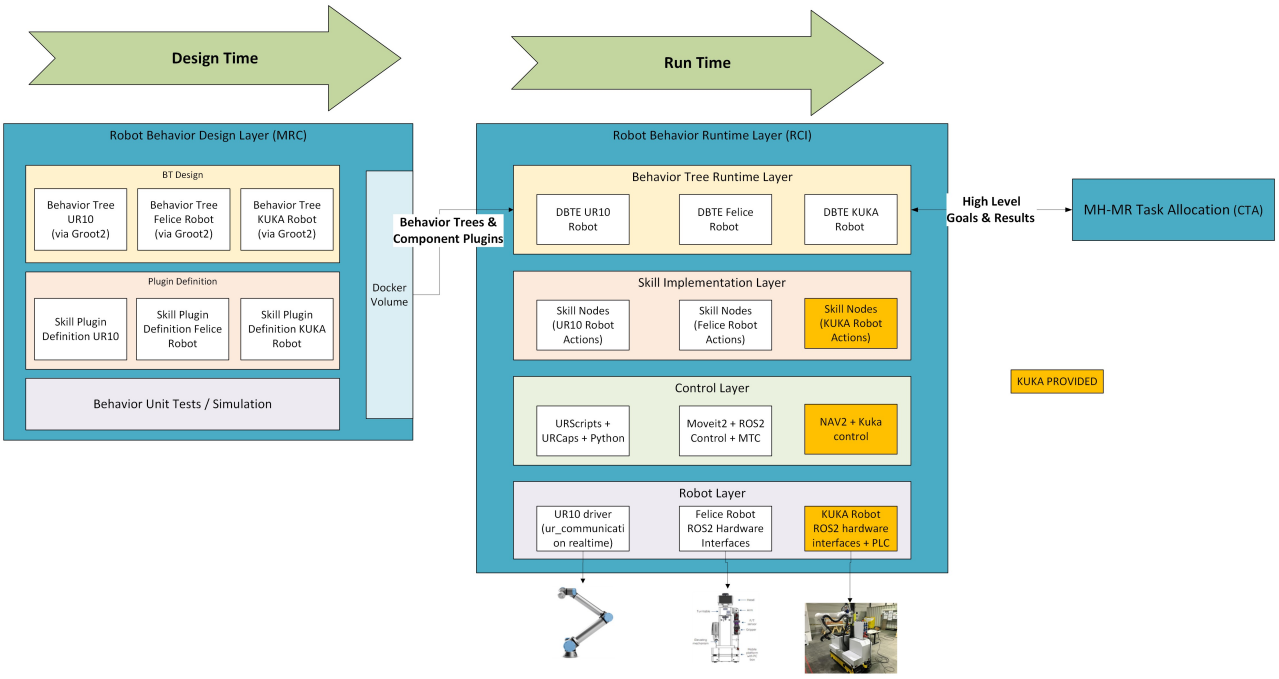


Figure 25: Mapping Robotic Capabilities(MRC) and Robot Capability Implementation (RCI) Architecture as deployed on Felice robot, UR10 robot and KUKA Platform

At the same time SIL is responsible for wrapping around the basic functionality of the robot like navigation, Cartesian space manipulation, joint space manipulation or gripper functionality in a ROS2 action server or service server [11]. This definition of ROS2 action/service server is important as the behavior trees use these to create action/service clients (BT nodes) called as skill plug-ins to communicate with low level functionality. The CL is responsible for planning, re-planning and trajectory generation for the robot and the gripper. Different robots have or may have different set of components in this layer. In our case with UR10 robot it is URScript, URCaps and python doing the job; for Felice robot it is the ROS2 control, MoveIt2 [6], MoveIt Task Contructor [9] and NAV2 [10] doing the same while for KUKA robot it is NAV2 and KUKA native control for manipulation.
Finally, the RL part takes care of how does the robot actually generates low level commands to execute trajectories generated by CL. For the UR10 robot it is UR10 driver which allows execution of UR scripts, joint or Cartesian motion; Similarly, for the Felice Robot the ROS2 hardware interface provides ROS2 position controllers which executes the trajectories and the KUKA robot layer has mix of ROS2 Hardware interfaces (ROS2 control) and PLC based control.
Fig. 25 shows the combined architecture of MRC-Robot Capabilities Implementation (RCI)-CTA. The MRC is a design time component designing the robot behavior i.e. arranging atomic skills of the robot like (arm movement, gripper motion, etc.) into a behavior tree with its respective skill plug-ins. CTA interacts with the RCI using the BTRTL and then robot executes the requested task using the behavior trees, atomic skills and various layers of RCI. More details on MRC are given deliverable D2.4.

5.3 Robotic Manipulation Approach

For the Manipulation and Grasping approach we would only consider UR10 robot and the Felice Robot [13] as their SIL, CL and RL are implemented by the RCI component. The Robot Capabilities are a combination of atomic skills, where atomic skills are the lowest possible level of robot action (see Tables 3 and 4 for atomic skills while Tables 5 and 6 for robot capabilities). Lets look at the status of the development of the atomic skills and the robot capabilities as described in deliverable D4.1.

Atomic Skill Type	Description	Status
Move Arm	Move the Felice Robot Arm system to a Cartesian coordinate or joint configuration	Developed and Tested
Move Head	Move the Felice Robot Head system to a Cartesian coordinate or joint configuration	Developed and Tested
Open	Open the Felice Robot Gripper system	Developed and Tested
Close	Close the Felice Robot Gripper system	Developed and Tested
Half	Open the Felice Robot Gripper system half	Developed and Tested
Move Platform	Move the platform to a location (x,y,theta)	Developed and Tested
Grasp Tool/Object	Plan motion towards tool/object and execute the combined motion of the arm and gripper system of the Felice Robot	Developed for all tool types, Tested for small screwdriver, Tests for long screwdriver pending
Put back Tool/Object	Plan motion towards holder and execute the combined motion of the arm and gripper system of the Felice robot	Developed for all tool types, Tested for small screwdriver, Tests for long screwdriver pending
Localize/Scan	Ability of the camera mounted on Felice Robot to trigger Object Detection and return 6D poses of tool or object.	Developed and Tested by NTUA

Table 3: Atomic skills developed for the Felice Robot

[13] https://www.felice-project.eu/, the felice robot was developed by Accrea, a project partner in the felice project https://accrea.com/

Atomic Skill Type	Description	Status
Move UR Arm	Move the UR Arm to a Cartesian coordinate or joint configuration	Developed and Tested
Unscrew	A specific skill of the screwing gripper mounted on the UR robot (given exact coordinates)	Developed and Tested
Localize/Scan	Ability of the camera mounted on UR Robot to trigger Object Detection and return 6D poses of screws.	Developed and Tested by NTUA

Table 4: Atomic skills developed for the UR10 Robot

Robot Capabilities Type	Description	Status
Pickup	This skill combines atomic skills of; localize object, move arm to object and grasp object	Developed & Tested for small screwdriver pending for long screwdriver
Place	This skill combines atomic skills of; localize tool or object holder, move arm to holder and put back tool/object	Developed & Tested for small screwdriver pending for long screwdriver
Navigate	This skill uses the atomic skill of move platform with recovery strategies	Developed & Tested
Handover	This skill combines atomic skills of; move arm to human position	Developed & Tested
Release	This skill combines atomic skills of; open gripper, close the gripper then move arm to parking position	Developed & Tested
Takeback	This skill combines atomic skills of; move arm to human position, open the gripper	Developed & Tested
Takebackclose	This skill combines atomic skills of; close the gripper then move arm to parking position	Developed & Tested
Cancel	This gives an operator the ability to cancel any running action	Developed & Tested

Table 5: Robot Capabilities developed for Felice Robot and the UR 10 Robot

Robot Capabilities Type	Description	Status
Cancel	This gives an operator the ability to cancel any running action	Developed & Tested
Disassembly	This skill combines atomic skills of; move UR arm to scanning, localize screws, move UR arm to screws then unscrew with the gripper	Developed & Tested

Table 6: Robot Capabilities developed for UR 10 Robot

Now, let us look at the technical approach for developing Robot Capabilities and atomic skills for each robot.

5.3.1 UR10 Robot

The UR-side implementation of the robot skills only uses ROS2 to provide interfaces for use with MRC and the BTRTL. Internally, it uses a custom, Python- and URScript-based software stack that makes the component highly flexible. This enables experienced robot programmers or automation engineers to implement new features using only UR's on-board methods (therefore changing only the CL). Fig. 26 shows an abstract flowchart of the unscrewing process.

The SIL mostly serves as a middleware that connects the subsystems, but also implements error handling and most of the safety features (omitted in Fig. 26 to simplify the illustration). Coordinate frame transformations are managed by the SIL, including the hand-eye calibration required for the end-effector-mounted vision system. In addition to the atomic robot skills, such utility methods are also made available in the form of ROS2 actions.

Path planning is done at the CL using the built-in kinematic solver and URScript programs that exploit the additional degree of freedom we obtain by rotating the tool around the rotation axis of the screwdriver. This can be necessary to extend the set of reachable poses, avoiding kinematic singularities and axis limits. The Deprag screwdriver also runs a short programmed sequence that ensures the correct torque is applied.

For development and debugging purposes, the SIL publishes OpenTelemetry traces and logs to a built-in visualization tool (.NET Aspire Dashboard). The component also includes a specialized user interface for standalone operation.

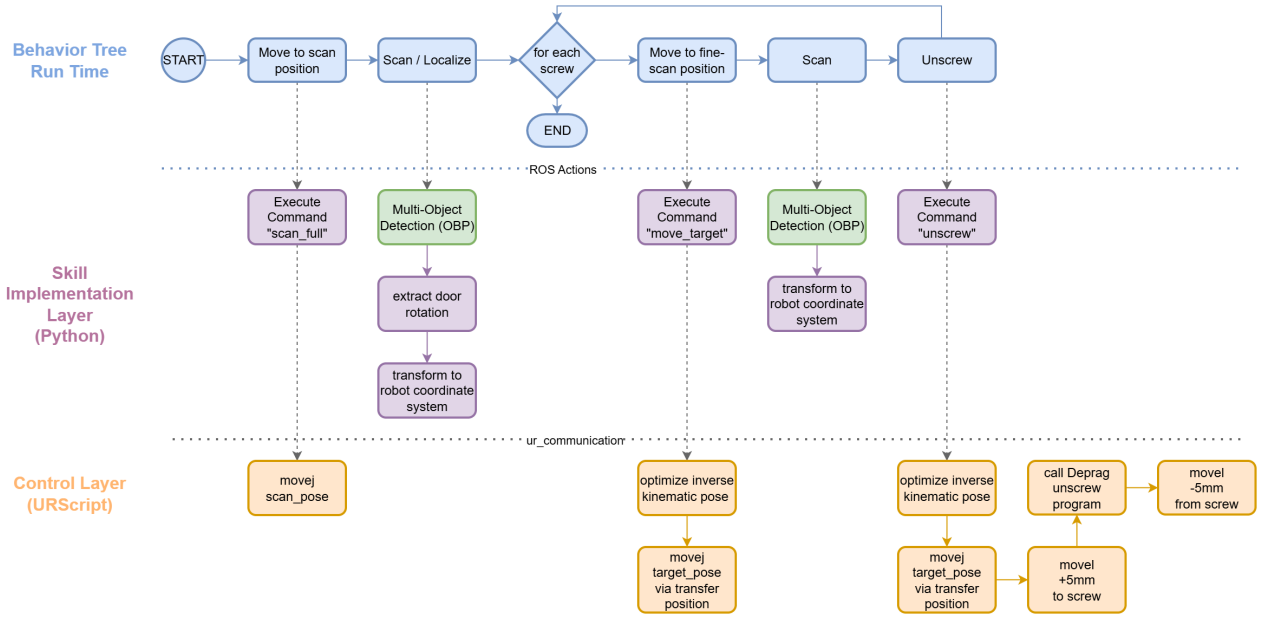


Figure 26: Simplified flowchart of the unscrewing process, divided into the execution layers.

5.3.2 Felice Robot

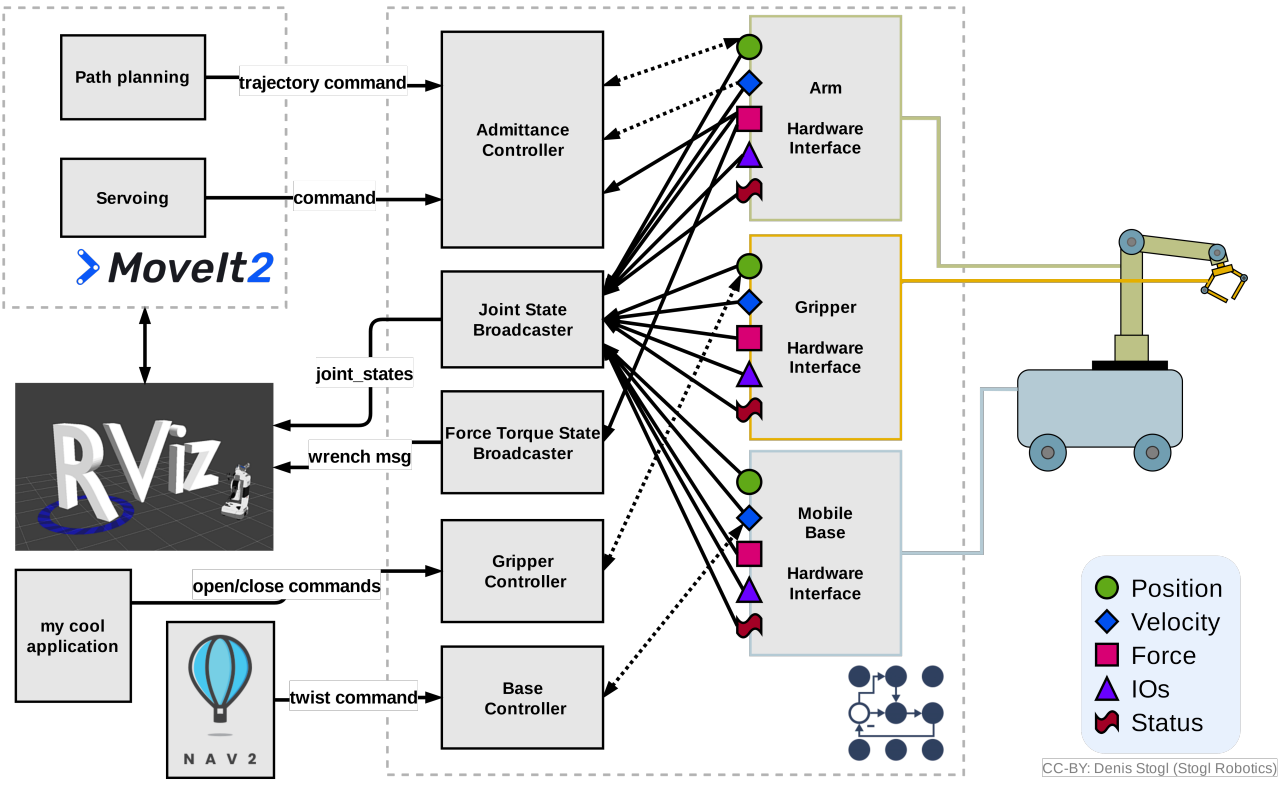
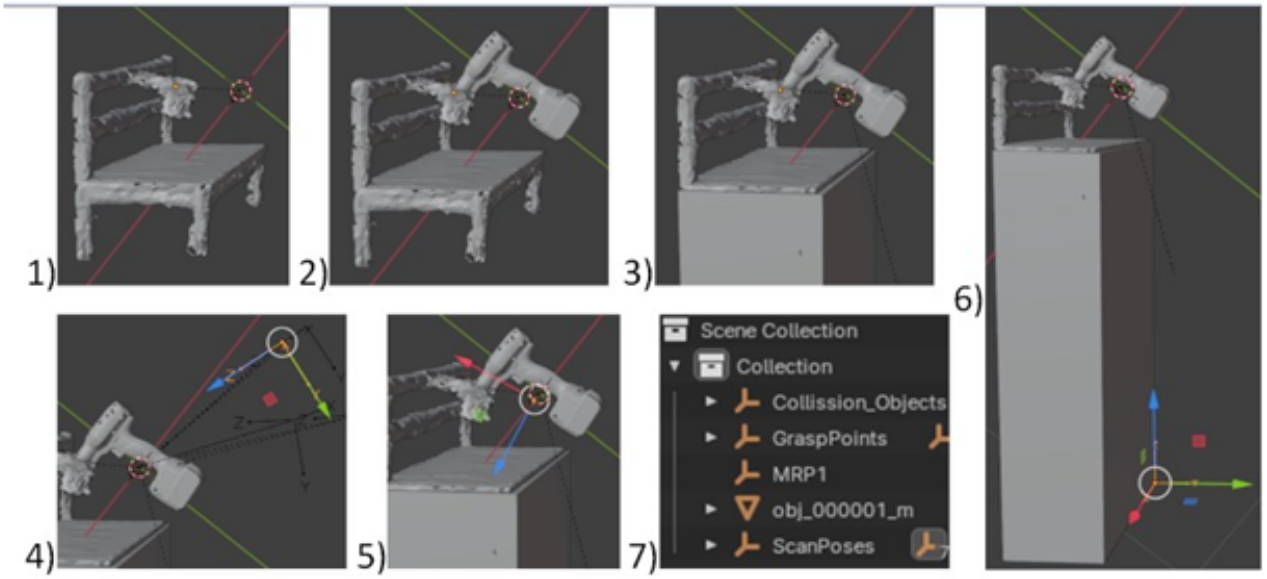


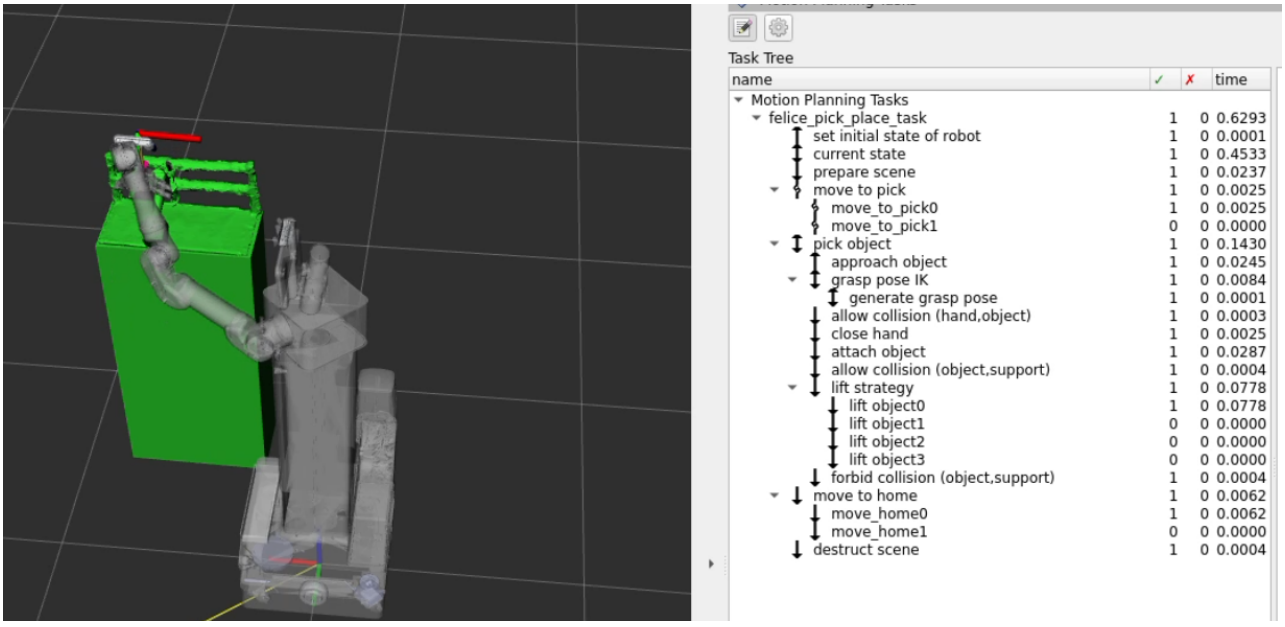
Figure 27: ROS2 Hardware and controller software architecture for mobile robots and mobile manipulators

The Felice Robot uses a close knit ROS2 architecture for manipulation and grasping by using ROS2 controllers, hardware interfaces, MoveIt2 and MoveIt Task Constructor as shown in the Fig. 27. The navigation of the robot is handled by robot manufacturer while the RCI is responsible for manipulation and grasping. The approach

starts with grouping chain of joints to control for manipulation. The first group is the felice_arm, the second is felice_gripper and the third is felice_head; then deciding kinematic solvers for each groups. Once this is done MoveIt2 automatically starts a ROS2 action server which communicates with low level ROS2 controllers to execute trajectories. Grouping the chains and MoveIt2 setup helps developing Move Arm, Move Head, Open, Close and Half atomic skills. MoveIt Task Constructor with blender based planning scene injection allows development of Grasp and Putback atomic skill developments.

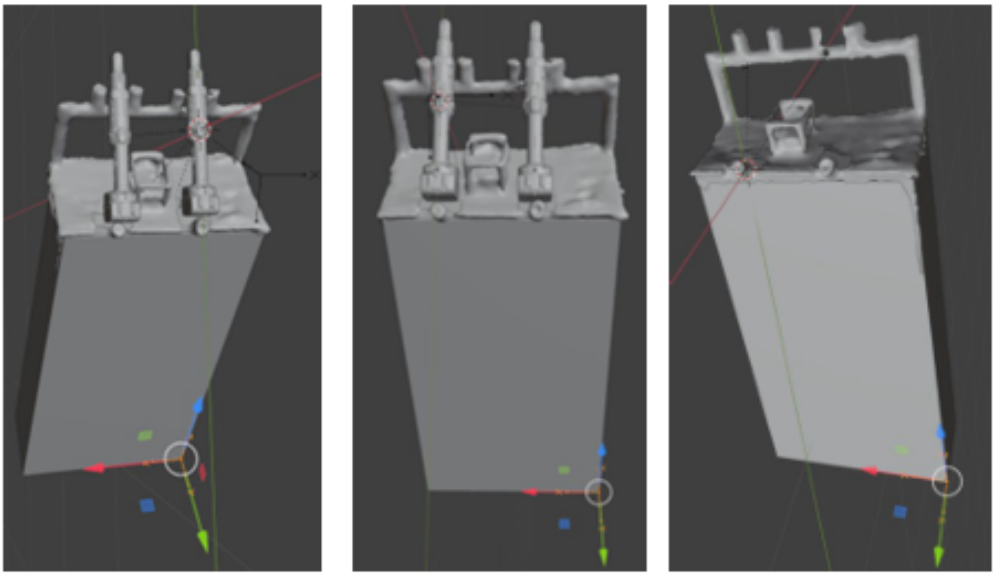


(a) Pick/Place object scene configuration in Blender. The sub-images 1) to 6) show the configuration steps mentioned above. Finally, the complete configuration results in an outline as depicted in 7)

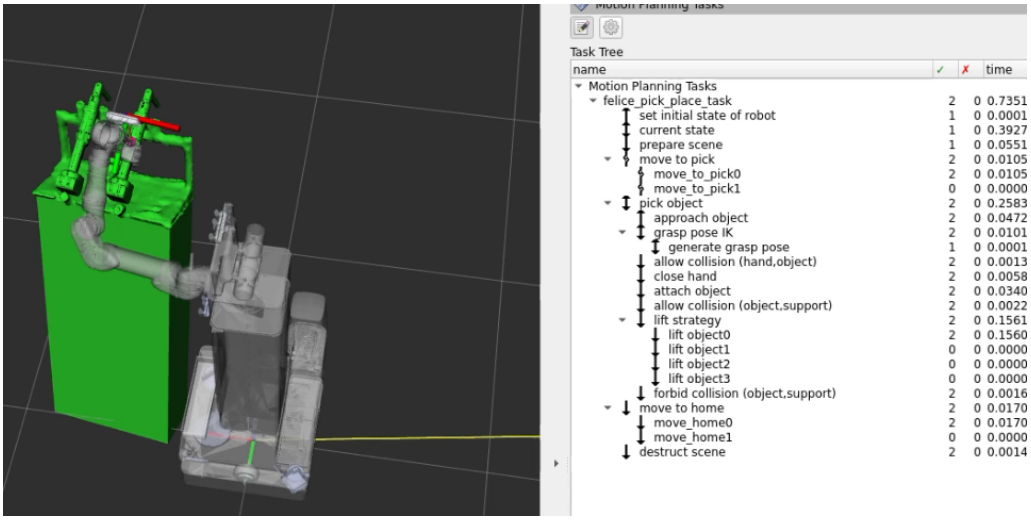


(b) Robot while Planning Tool Pickup for tool10

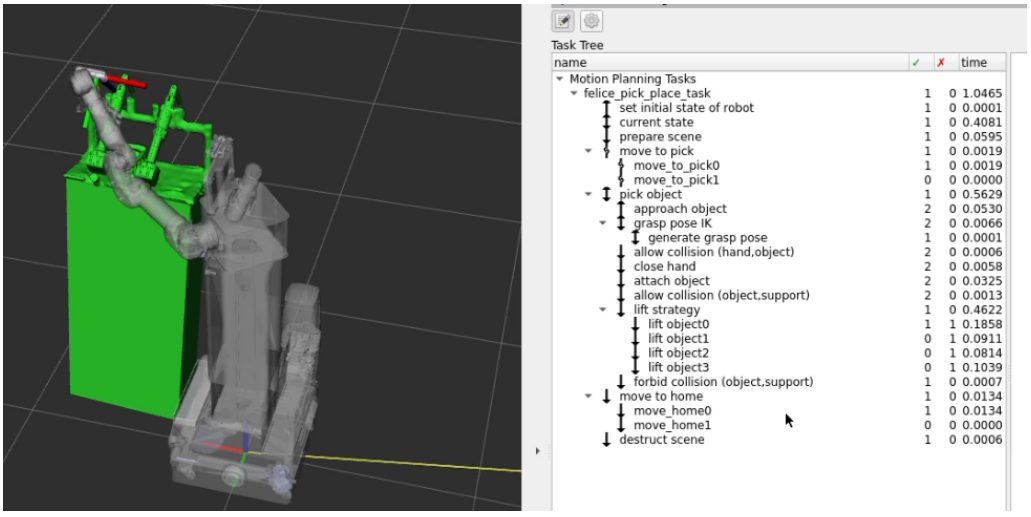
Figure 28: Planning scene and MoveIt Task constructor pipeline for Tool 10 pickup



(a) leftmost figure is of the planning scene wrt right tool,middle figure is of the planning scene wrt left tool and rightmost is of the planning scene for placing the long screw driver



(b) Robot while Planning Tool Pickup for tool201



(c) Robot while Planning Tool Pickup for tool210

Figure 29: Planning scene and MoveIt Task constructor pipeline for Tool 20 pickup

Blender is a 3D modelling tool which has been used to generate a 3D planning scene which could be inserted in MoveIt Task Constructor to plan grasping or putback trajectories. The planning scene generation is performed by creating blender files and the process of doing it as below (Fig. 28a):

- Add tool stand CAD models: Adding CAD model of the tool stand
- Add the tool/object: Adding CAD model of the tool/object
- Define collision objects: the big cube below the tool stand models act as collision object such that the robot does not plan a trajectory under the tool stand
- Define scanning poses: Defining scanning poses wrt tool stand model
- Define grasping/putback poses: Defining grasping pose wrt tool and putback pose wrt tool holder.
- Define a reference frame for the collision scene: it is a frame at the bottom right corner of the cube

So, lets see how is the planning scene is used. The pickup/putback process involves following steps; 1) Felice_arm moves to defined scanning position in blender file, the movement is collision free as the collision object also loads while movement, 2) Felice_arm plans pickup/place using MoveIt Task Constructor where the planning scene provides collision scene and 3D scene where to pick/place the tool considering detection results. Fig. 28 shows the planning scene generation using blender Fig. 28a which is then loaded to MoveIt task constructor as glTF file and the trajectory is planned as shown in Fig. 28b.

Similarly for Toolstation 20, Fig. 29 shows the planning scene generation and tool pickup for both the tools. For Tool201 (Right), Tool210 (Left) planning scene is as shown in Fig. 29a while the MoveIt task constructor planning trajectory and executing tool pickup is as shown in Figures 29b and 29c.

5.4 Results

We break down the result section into two subsections. Task Execution Framework and Manipulation approach.

5.4.1 Task Execution Framework

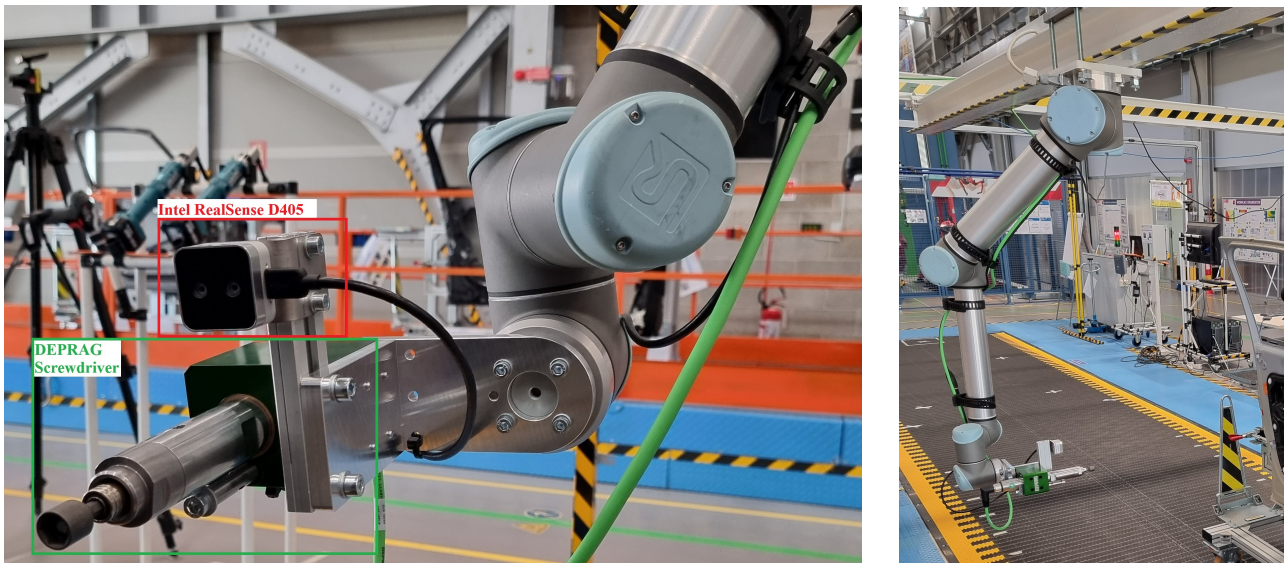
5.4.1.1 UR10 Robot The UR10 robot task execution engine, as discussed in Sect. 5.2 has been successfully integrated. The main challenge with this component is ensuring that errors are handled safely and correctly, because the root cause of any error needs to be communicated to the caller of the request. Immediate cancellation has to be possible at any time without leaving the system in an erroneous state. Therefore, the focus of the integration tests was correction communication, especially in case of errors and cancellations.

5.4.1.2 Felice Robot The Felice robot task execution engine as discussed in Sect. 5.2 has been successfully integrated with CTA. Similar to the UR10 robot execution engine, all the features have been developed and tested for Felice Robot as well.

5.4.1.3 KUKA Robot The KUKA robot task execution engine as discussed in Sect. 5.2 has been developed awaiting integration with CTA and later robot validation. Similar to the UR10 robot and Felice Robot execution engine, all the features have been developed, deployed on the robot and awaiting testing for KUKA Robot. The behavior trees for KUKA robot include capabilities; Navigation and drilling.

5.4.2 Robot Manipulation

5.4.2.1 UR10 Robot The planned hardware setup, as documented in D4.1, has been finalized and implemented at CRF. Fig. 30b shows the UR10 robot setup installed at the CRF disassembly line. A close-up image of the end-effector, including the DEPRAG screwdriver and the Intel RealSense D405 camera can be seen in Fig. 30a.



(a) End-effector with DEPRAG screwdriver and Intel RealSense D405 camera

(b) UR10 mounted overhead

Figure 30: UR10 Setup installed at the CRF Disassembly Line, Melfi, Italy

The unscrewing process has been successfully implemented and tested. It consists of a multi-step workflow. First, the screws are detected from a position where the whole door is visible. Since the camera has a rather low resolution and this would not be precise enough, every screw is again detected from a closer position (at a distance of 20cm from the screw). This way the screws can be unscrewed with the required sub-millimetre accuracy.

The integrations tests already carried out, involving the RCI, MRC, OBP, CTA and AVT components, show that the robot is successfully integrated into the Soprano framework.

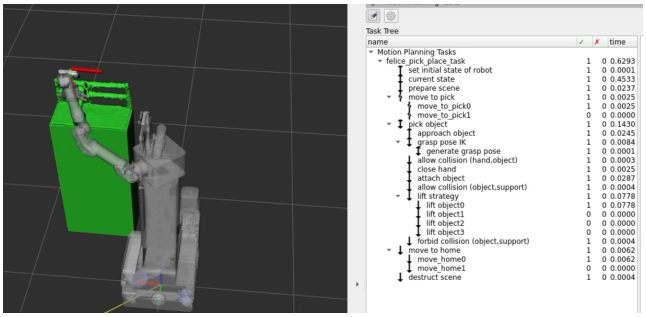
5.4.2.2 Felice Robot All the earlier planned steps as mentioned in the deliverable D4.1 (Modular, Reconfigurable and Flexible Tools I) Section 5.5 have been achieved and detailed below:

- Developing hardware interfaces for Arm, Head and the Gripper systems; Concluded by the robot manufacturer
- Allowing Force Torque Sensor to be included in robot description and allow efficient data transfer from Force Torque Sensor; Concluded by the robot manufacturer
- Efforts for integration between Felice Robot Hardware Interface, controllers and planners; Concluded by RCI
- Test MoveIt2 with real robot; Concluded by RCI
- Develop MoveIt Task Constructor Pipeline such that efficient tool or object grasping could be made possible; Concluded by RCI
- Design and develop a Behavior Tree based task-execution framework able to handle failures and perform given task; Concluded by RCI
- Parameter reconfiguration is achieved by using ROS2 params and lifecycle nodes

The Felice Robot has successfully shown tool grasping, tool placing and tool handover at work station 10.



(a) Robot while Planning Tool Pickup



(b) MoveIt Task Constructor planned tool pickup



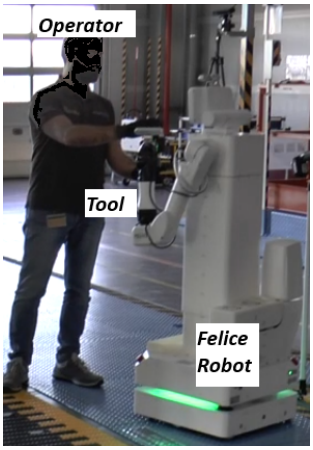
(c) Finally, Robot picks up the tool

Figure 31: Felice Robot Tool Pickup

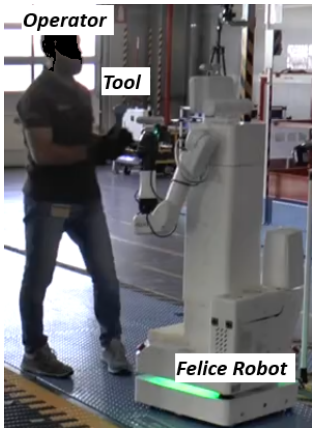
Fig. 31 shows all the layers of RCI in action. The tool pickup is first divided into scanning the tool, then planning tool pickup using MoveIt Task Constructor and finally trajectory execution using position controllers.



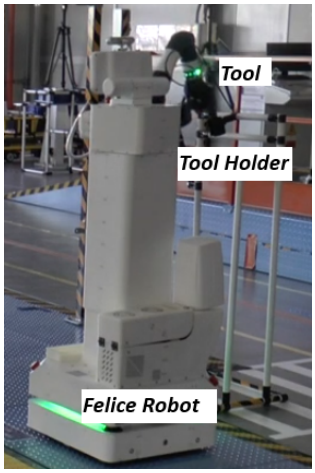
(a) Felice Robot Picking up the tool



(b) Felice Robot Handing over the tool



(c) Felice Robot Taking back the tool



(d) Felice Robot Placing the tool

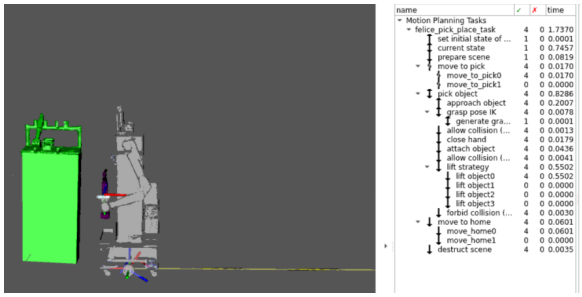
Figure 32: Felice Robot executing all desired tasks

Similarly Fig. 32 shows the Felice Robot workflow and execution of Robot Capabilities like Tool pickup, tool handover, tool release, tool takeback, tool takebackclose and finally tool placing back to the tool stand by the Felice Robot.

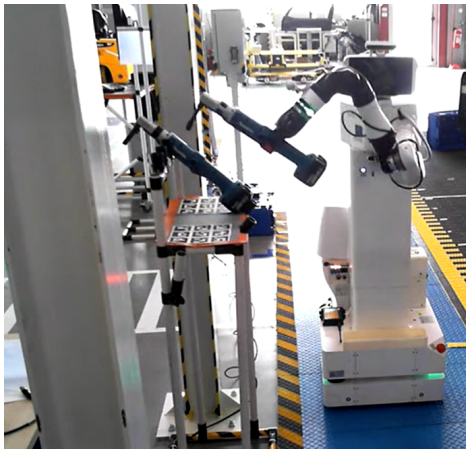
The Robot capabilities of Tool pickup, place, handover, takeback, release and tackbackclose has been tested at Centro Researche Fiat (CRF). Figure 33 shows how the successful long screwdriver pickup looks like with planning how to pick it. From left the first one shows left screwdriver pickup, second one showing its corresponding planning strategy, third one showing right tool pickup and finally its corresponding planning strategy.



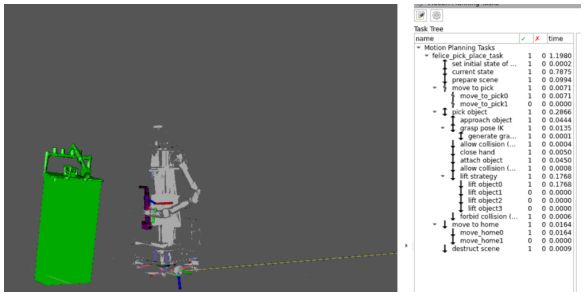
(a) Felice Robot Picking up the long screwdriver (Left)



(b) Felice Robot Pickup planning Left



(c) Felice Robot Picking up the long screwdriver (Right)



(d) Felice Robot Pickup planning Right

Figure 33: Felice Robot executing Tool Pickup at Workstation 20

RCI has been also well integrated with CTA and OBP, while the complete pipeline of CTA, RCI, OBP , Nav2 and AVT has been successfully tested. The complete evaluation of the Felice Robot at workstation 10 has been also carried out for assigned agent action accuracy where RCI is at the centre of evaluation and is reported in deliverable D5.3 (Use Case Evaluation I). The robot capabilities have been developed, deployed, tested and integration verified for workstation 20 with long screwdrivers. The complete evaluation of RCI at workstation 20 with long screwdrivers is in the scope of the upcoming deliverable D5.6(Use Case Evaluation II) and will be reported accordingly.

5.5 Challenges and Future Work

The main remaining challenge for the UR10 unscrewing process is that the door is not very sturdily mounted and can be pushed away by the robot if the applied pressure during unscrewing is too high. Further optimizations and thorough testing of the robot program will be necessary to achieve better reliability.

In order to improve the worker's safety in the collaborative setting, an additional emergency button and system status indicators need to be installed in the immediate vicinity of the robot. An additional laser-based safety sensor will be installed. This creates a safety zone, which causes the robot to slow down if a human comes too close while in operation.

The Felice Robot still needs integrations tests at workstation 20 to pickup, place and handover long screwdrivers. These tests will be carried out shortly. Then optimization in terms of robot speed may take place, as current timing mean in terms of execution of tasks as reported in deliverable D5.3 increases cycle time.

6 Conclusions and Next Steps

This deliverable demonstrates the results achieved in the development of Modular, Reconfigurable and Flexible Tools for MH-MR collaboration. The Context Extraction Module (CEM) component leverages ontology-based approaches for real-time data extraction and monitoring. Experiments demonstrate that the ontology model can be used as decision-support by other SOPRANO components across varied use cases. The MH-MR Architecting Tools (MAT) uses composable models to describe environments and motion specifications for arms. These models are used to generate execution code (e.g., the control loop for an arm) and execution artefacts (e.g., occupancy grids, task specifications, etc.). The current prototype supports the transformation of IFC models into simulation artefacts, and the generation of controllers for compliant arm motions. AI Model Optimizer (AIO) and MLOps Orchestrator (MLO) implement AI pipelines tailored for non-expert users focusing on reusability explainability and governance. Experiments show the AIO's potential of the model partitions reducing resource utilization. The MLO has been validated in two use cases for user-friendliness through the Workflow Management Editor interface. Finally, the Robot Capabilities Implementation (RCI) component integrates robotic task execution frameworks using behaviour trees and motion planning tools like ROS2 and MoveIt2. The BTs for navigation and drilling capabilities were developed, and the current prototype has gone integration testing on two use cases. In addition to finalizing the hardware setup, the robot manipulation approach for unscrewing has been tested for the CRF use case, and integration tests with multiple SOPRANO components.

For our future work, we have identified challenges and next steps for the prototypes presented on this document. This includes extensions to the ontology-based context models with pilot-specific semantics and integration of advanced reasoning mechanisms for CEM, extensions to the IFC transformation pipeline for multi-storey buildings and additional tasks, the full motion specification for the CRF use case motions for MH-MR Architecting Tools (MAT), additional experiments with YOLO ONNX and improvements to the computation of the optimal inference time for AIO, adding OPC-UA support in the MLO pipelines, and increasing the safety of the worker and the reliability of the RCI. Finally, additional work is needed on the integration of the SOPRANO components for the three use cases to increase the TRL of our solutions. This includes optimization of the implemented solutions, thorough integration testing and extensive validation in the real-world pilots. We plan to address the above mentioned limitations and challenges, and those identified during the Use Case evaluation in D4.3.

A Updates since D4.1

A.1 Context Extraction Module (T4.1)

This section summarises the main advancements of the **Context Extraction Module (CEM)** with respect to the early prototype presented in D4.1, highlighting the progress towards a fully operational, pilot-integrated context awareness solution within the SOPRANO framework.

A.1.1 Advancements since D4.1

The updates in Sect. 2 compared to D4.1 are summarized below:

- **Extension of the ontology-based context models** to fully support the three SOPRANO pilot domains (construction, manufacturing, and agriculture), including the introduction of new domain-specific entities, relationships, and event types. These extensions enable the representation of complex MH-MR interactions, such as human proximity, robot operational states, and environment-specific conditions.
- **Refinement and instantiation of context models per pilot**, ensuring that each use case (KUKA, CRF, eTERRY) is supported by a tailored ontology capturing relevant actors (humans, robots), resources, and contextual events (e.g., GNSS localization, tool usage, object detection).
- **Implementation of a modular and extensible Context Monitoring framework**, capable of acquiring and aggregating heterogeneous data from multiple sources, including:
  - SOPRANO perception modules (HMO, OBP, VLM, NVL)
  - Robotic middleware (ROS2)
  - Industrial communication protocols (OPC-UA)
  - External and legacy data sources (e.g., REST services, databases, file systems)
- **Development of a complete Context Extraction pipeline**, integrating:
  - Context identification based on real-time monitoring data and ontology models
  - Context reasoning mechanisms to infer higher-level contextual situations
  - Context provision services to disseminate refined context information to other SOPRANO components
- **Introduction of real-time context change detection mechanisms**, enabling the system to dynamically identify significant events such as:
  - Human presence or proximity to robots
  - Object detection in robot trajectories
  - Changes in robot motion or operational modes
  - Safety-critical conditions (e.g., emergency stops)
- **Integration with the SOPRANO message broker**, allowing context information to be published in a structured, interoperable format (RDF-based), supporting seamless communication with downstream modules (e.g., task execution, safety analysis).
- **Enhanced support for heterogeneous and distributed environments**, ensuring compatibility with both indoor industrial settings (CRF, KUKA) and outdoor environments (eTERRY), including the incorporation of GNSS-based localization and environment-aware sensing.
- **Cross-pilot validation of the Context Extraction Module**, demonstrating:
  - Reliable transformation of low-level sensor data into semantically enriched context
  - Consistent context interpretation across different domains and data sources
  - Applicability to real-world MH-MR collaboration scenarios
- **Improved architectural maturity**, transitioning from a conceptual and early prototype (D4.1) to an integrated and deployable solution, aligned with the SOPRANO system architecture and supporting runtime context-aware decision-making.

A.2 MH-MR Architecting Tools (T4.1 & T4.2)

- Table 1 was updated to include relevant concepts for the milling, painting and grinding tasks based on the final concept alignment between IFC and the FloorPlan concepts.
- Sect. 3.2.1.3 was updated with the final analysis and concept alignment.
- Sect. 3.2.2 was updated with the complete JSON-Ld models for the CRF find-and-unscrew motion task, and their corresponding DSL models.
- Sect. 3.3.1 added details on the generation of SOPRANO-specific artefacts from FloorPlan models.
- Sect. 3.3.1 added details about the implementation of the transformation of IFC models into executable artefacts. This includes a polyline representation for the VLM component, wall and task descriptions for HDT and the generation of navigation goals for the KUKA use case.
- We also note the implementation of the IFC transformation in Sect. 3.3.1. We describe the implementation of two components: the IFC-LD Transformer and the IFC-LD Interpreter, and show examples of their use to generate artefacts from IFC models.
- Sect. 3.3.2 was updated with the control models required to implement the motion specifications and generate the code from them.
- Added Sect. 3.3.3 showcasing the scene setup at the Uni-Bremen with the robot and a car door required to test the CRF use case.

A.3 AI Model Optimizer (T4.4) & MLOps Orchestrator (T4.5)

A.3.1 AI Model Optimizer

The changes introduced in Sect. 4.2 compared to D4.1 are summarized below:

- The Kubeflow experiments reported in D4.1 were not implemented, due to the use of Docker instead of Kubernetes. The purpose of the task is not intended for model training.
- The deliverable now provides a complete description of the AIO workflow and functionality.
- The AIO subcomponents have been containerized and support model splitting and profiling to identify suitable candidates for submodel generation.
- The AIO pipeline operates in a plug-and-play manner, enabling straightforward integration.
- A deployment plan for distributed inference of the OBP component on heterogeneous edge devices is now generated automatically.

A.3.2 MLOps Orchestrator

The changes introduced in Sect. 4.3 compared to D4.1 are summarized below:

- Evolved the MLOps Orchestrator from a conceptual design in D4.1 to a production-ready implementation, with all core components now operational: Multi-Purpose Processing Engine (MPPE), Workflow Management Editor (WME), persistent registries, Keycloak-based IAM integration and a complete Docker-based deployment stack.
- Finalised and documented the four-layer functional architecture: Monitored IoT System Layer (sensors, ROS robots, digital twins, human inputs and external services via MQTT/REST/WebSockets over a pub/sub Data Bus), Data Orchestration & Applications Layer (MPPE + Airflow for real-time and asynchronous workflow execution), Core Services Layer (API Gateway, IAM, policy enforcement and operational registries), and User Interfaces Layer (web-based WME for workflow composition).
- Extended the description of the Workflow Management Editor (WME) from a high-level sketch to a fully specified, code-free, drag-and-drop interface that lets domain experts register processors, define digital resources and compose workflows visually through coordinated Warehouse/Lab/Airflow views.

- Introduced and documented a concrete processor onboarding workflow for MLO, with two supported deployment methods (local deployment via a modules folder and `docker-compose.yml`, and remote deployment via Docker Hub images) and a complete registration flow based on Processor Definitions/Manifests to turn third-party algorithms into reusable building blocks.
- Updated the deployment description to cover the use of MLO in the KUKA and E-TERRY pilots (robot telemetry, context orchestration and heterogeneous agricultural robot data flows), the prepared deployment for the CRF use case, and the operational integration with the Context Extraction Module (CEM).
- Added details on the complete deployment packaging of MLO, including a full Docker Compose setup validated on the reference infrastructure (e.g. Ubuntu 22.04, 16-core CPU, 32 GB RAM) and the corresponding installation and system-requirements documentation.
- Clarified the separation of concerns with AIO: compared with D4.1, the MLO section in D4.2 no longer repeats the Kubeflow-based training pipelines, Dataset Registry and CI/CD training automation concepts, which are now only covered in the AIO text, while MLO focuses on runtime orchestration and pilot deployments.

A.4 Robot Capabilities Implementation (T4.6)

A.4.1 Advancements since D4.1

- Transition from an initial prototype of the task execution framework (D4.1) to a fully implemented and validated Robot Capabilities Implementation (RCI) framework, supporting real-world deployment in industrial use cases Sect. 5.2.
- Development and deployment of a Behavior Tree-based Task Execution Engine (DBTE/RCI runtime layer), enabling robust execution of modular robot capabilities such as Pick, Place, Handover, Takeback, Navigation, and Disassembly across multiple robotic platforms (UR10, Felice, KUKA) Sect. 5.3.
- Extension from simulation-focused validation (D4.1) to real hardware integration and testing, including successful deployment on UR10 and Felice robots in the CRF disassembly use case Sect. 5.4.1.
- Implementation and validation of a layered execution architecture (BTRTL, SIL, CL, RL), improving modularity, fault handling, and scalability of task execution compared to the initial prototype sec:5.4.1 Task Execution Framework.
- Integration of advanced robot manipulation pipelines using MoveIt2 and MoveIt Task Constructor, enabling structured planning for complex tasks such as tool pickup, placement, and disassembly Sect. 5.4.1.
- Enhanced system-level integration with SOPRANO components, including CTA (task allocation), OBP (perception), and Nav2, enabling coordinated multi-component execution within the overall MH-MR framework Sect. 5.4.2.
- Demonstration of end-to-end task execution workflows, including perception-driven grasping, tool handover, and unscrewing operations, validating the framework in realistic industrial scenarios Sect. 5.4.2.
- Deployment of a lightweight RCI runtime for the KUKA use case, supporting navigation and drilling capabilities, extending the applicability of the framework across different pilot domains Sect. 5.4.1.
- The state-of-art review about the task execution frameworks and the manipulation and grasping approaches are done in D4.1 (Modular, Reconfigurable and Flexible Tools I) and are not repeated in this deliverable. This deliverable focuses on the iterative developments, integration progress, and validation results achieved since D4.1.

A.4.2 Remaining Challenges and Future Work

- Extended validation for manipulation and handover tasks (e.g., long screwdriver handling at workstation 20).
- Further optimisation of execution performance in collaboration with the CTA module, particularly reducing cycle times observed during integrated evaluations (see D5.3).

- Enhancement of robustness in physical interaction tasks (e.g., tool handover and takeback).
- Strengthening safety integration through reactive behaviours with the help of RSA/DSA and CTA modules in collaborative human-robot scenarios.

References

[1] Nadja Adamovic, Ernst Dieter Janotka, Hedi Karray, Arkopaul Sarkar, Dimitris Kiritsis, Gerhard Goldbeck, Michela Magas, John Breslin, Lan Yang, Anna Fensel, Umutcan Simsek, Silvana Muscella, Cristina Mancarella, Florina Piroi, and Emanuele Ghedini. *OntoCommons Roadmap V1*. April 2022.

[2] Sharath Chandra Akkaladevi, Matthias Propst, Kapil Deshpande, Michael Hofmann, and Andreas Pichler. Towards a behavior tree based robotic skill execution framework for human robot collaboration in industrial assembly. In *2024 10th International Conference on Automation, Robotics and Applications (ICARA)*, pages 18–22. IEEE, 2024.

[3] Robert Arp, Barry Smith, and Andrew D. Spear. *Building ontologies with Basic Formal Ontology*. Massachusetts Institute of Technology, Cambridge, Massachusetts, 2015.

[4] Lamia Belouaer, Maroua Bouzid, and Abdel-illah Mouaddib. *Spatial Knowledge In Planning Language*. October 2011.

[5] Stefano Borgo, Roberta Ferrario, Aldo Gangemi, Nicola Guarino, Claudio Masolo, Daniele Porello, Emilio M. Sanfilippo, and Laure Vieu. Dolce: A descriptive ontology for linguistic and cognitive engineering. *Applied Ontology*, 17(1):45–69, January 2022.

[6] Sachin Chitta. Moveit!: an introduction. In *Robot Operating System (ROS) The Complete Reference (Volume 1)*, pages 3–27. Springer, 2016.

[7] Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[8] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer, 2014.

[9] Michael Görner, Robert Haschke, Helge Ritter, and Jianwei Zhang. Moveit! task constructor for task-level motion planning. In *2019 International conference on robotics and automation (ICRA)*, pages 190–196. IEEE, 2019.

[10] Steve Macenski, Francisco Martín, Ruffin White, and Jonatan Ginés Clavero. The marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2718–2725. IEEE, 2020.

[11] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[12] Sumaira Manzoor, Yuri Goncalves Rocha, Sung-Hyeon Joo, Sang-Hyeon Bae, Eun-Jin Kim, Kyeong-Jin Joo, and Tae-Yong Kuc. Ontology-based knowledge representation in robotic systems: A survey oriented toward applications. *Applied Sciences*, 11(1010):4324, January 2021.

[13] Argentina Ortega, Samuel Parra, Sven Schneider, and Nico Hochgeschwender. Composable and executable scenarios for simulation-based testing of mobile robots. *Frontiers in Robotics and AI*, 11, 2024.

[14] Samuel Parra, Argentina Ortega, Sven Schneider, and Nico Hochgeschwender. A thousand worlds: Scenery specification and generation for simulation-based testing of mobile robot navigation stacks. In *IROS*, pages 5537–5544, 2023.

[15] E. P. Popov, A. F. Vereshchagin, and S. L. Zenkevich. *Manipuljacionnyje roboty: Dinamika i algoritmy*. Nauka, Moscow, 1978.

[16] Alireza Rastegarpanah, Hector Cruz Gonzalez, and Rustam Stolkin. Semi-autonomous behaviour tree-based framework for sorting electric vehicle batteries components. *Robotics*, 10(2):82, 2021.

[17] Francesco Rovida, Matthew Crosby, Dirk Holz, Athanasios Polydoros, Bjarne Großmann, Ronald Petrick, and Volker Krueger. *SkiROS—A skill-based robot control platform on top of ROS*, page 121–160. May 2017. journalAbbreviation: Studies in Computational Intelligence.

[18] S. Schneider, N. Hochgeschwender, and H. Bruyninckx. Domain-specific languages for kinematic chains and their solver algorithms: Lessons learned for composable models. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 9104–9110, London, UK, May 2023.

[19] S. Schneider, V. Kalagaturu, H. Bruyninckx, and N. Hochgeschwender. From composable models to correct-by-construction software for contact-rich robotic mobile-manipulation tasks. *IEEE Robotics and Automation Letters*, 10(10):9894–9901, October 2025.

[20] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, Pierre-Antoine Champin, and Niklas Lindström. Json-ld 1.1: A json-based serialization for linked data. W3C Standard, 2020.

[21] Jonathan Styrud, Matteo Iovino, Mikael Norrlöf, Mårten Björkman, and Christian Smith. Combining planning and learning of behavior trees for robotic assembly. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 11511–11517. IEEE, 2022.

[22] R.Endre Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 2(6):160–161, 1974.

[23] Moritz Tenorth and Michael Beetz. Representations for robot knowledge in the knowrob framework. *Artificial Intelligence*, 247:151–169, 06 2017.

[24] The Linux Foundation. Onnx.

[25] Alessandro Umbrico, Andrea Orlandini, and Amedeo Cesta. An ontology for human-robot collaboration. *Procedia CIRP*, 93:1097–1102, January 2020.

[26] A. F. Vereshchagin. Modelling and control of motion of manipulational robots. *Soviet Journal of Computer and Systems Sciences*, 27(5):29–38, 1989.

[27] Wikipedia contributors. Milling (machining) — CNC pocket milling. `https://en.wikipedia.org/wiki/Milling_(machining)#CNC_pocket_milling`, 2026. Accessed: 2026-03-23.