SCOTT:

Secure COnnected Trustable Things

Safety-Critical Sandbox Final Evaluation

Document Type	Deliverable
Document Number	D10.7
Primary Author(s)	Andrii Berezovskyi / KTH
Document Version / Status	1.0 | Final
Distribution Level	PU (public)

Project Acronym	SCOTT
Project Title	Secure COnnected Trustable Things
Project Website	www.scottproject.eu
Project Coordinator	Michael Karner | VIF | michael.karner@v2c2.at
JU Grant Agreement Number	737422
Date of latest version of Annex I against which the assessment will be made	2020-05-29

SCOTT has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737422. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, Norway.

CONTRIBUTORS

Name	Organization	Name	Organization
Alberto Hata	EAB	Amadeu Nascimento Jr.	EAB
Klaus Raizer	EAB	Swarup Kumar Mohalik	EAB
Yifei Jin	EAB	Aneta Vulgarakis Feljan	EAB
Rafia Inam	EAB	Andrii Berezovskyi	KTH
Ajay Kattepur	EAB	Magnus Isaksson	RTE
Ricardo Souza	EAB		

FORMAL REVIEWERS

Name	Organization	Date
Rocío Gómez	INDRA	2020-09-01
Francisco Parrilla	INDRA	2020-09-01
Peter Priller	AVL	2020-09-07

DOCUMENT HISTORY

Revision	Date	Author / Organization	Description
0.1	2020-05-12	Andrii Berezovskyi	Initial version
0.2	2020-07-30	Andrii Berezovskyi	Version for internal review
0.3	2020-08-05	Andrii Berezovskyi	Version for formal review
1.0	2020-09-11	Andrii Berezovskyi	Final version

TABLE OF CONTENTS

CONTRIBUTORS 2

TABLE OF CONTENTS 3

LIST OF FIGURES 5

LIST OF TABLES 8

1 EXECUTIVE SUMMARY 9

2 OBJECTIVES 11

2.1 Deliverable Objectives 11
2.2 SCOTT Objectives 11

3 DESCRIPTION OF WORK 12

3.1 Work Overview 12
3.2 Use Case Architecture 13
3.3 Use cases 14

3.3.1 Use cases overview 14
3.3.2 Digital Twins 16

3.3.2.1 Overview 16
3.3.2.2 Digital Twin service 16
3.3.2.3 Other services 17
3.3.2.4 Publish-Subscribe system and State Event Filtering service 19
3.3.2.5 Domain modelling 20
3.3.2.6 Robot and robot software 28

3.3.3 Task Level Planning 36

3.3.3.1 Planner Service 36
3.3.3.2 Estimator 40

3.3.4 Risk Assessment for Safe Operations 41

3.3.4.1 Risk Identification 41
3.3.4.2 Risk Analysis and Evaluation 43
3.3.4.3 Scene Understanding 44
3.3.4.4 Scene Graph for Risk Mitigation 47
3.3.4.5 Fuzzy Logic System-based Risk Mitigation 48
3.3.4.6 Reinforcement Learning-based Risk Mitigation 49
3.3.4.7 Distributed Solution of Risk Management Nodes 54

3.3.5 Goal State Generation out of Natural Language Mission Specifications 55
3.3.6 Explanations for Synthesized Plans 56
3.3.7 Formal Verification of Strategic Plans 60
3.3.8 Monitoring and Visualization of Warehouse KPIs 72

3.3.9 Design and V&V of Planning Domain Model 79

3.4 Additional use case from RTE 77

3.4.1 Overview 77
3.4.2 System Architecture 79
3.4.3 Specifications 82
3.4.4 Implementation 83
3.4.5 Potential Application Scenarios 83
3.4.6 Relation to requirements and building blocks 83
3.4.7 Conclusions 84

3.5 Evaluation and future outlook 84

3.5.1 Work Package demo evaluation 84
3.5.2 Overall Work Package evaluation 85

4 DISSEMINATION, EXPLOITATION AND STANDARDISATION 87

5 LINK TO TECHNOLOGY LINES 88

5.1 TL Distributed Cloud Integration (WP24) 88
5.2 TL Reference Architecture (WP26) 88

6 INTEROPERABILITY 89

7 CONCLUSIONS 90

REFERENCES 91

A. ABBREVIATIONS AND DEFINITIONS 93

LIST OF FIGURES

Figure 1. Illustration of the work package scenario. 12

Figure 2. Warehouse Sandbox architecture levels as mapped onto the SCOTT HLA architecture. 13

Figure 3. Warehouse Sandbox Architecture description. Green components are part of the final demonstrator scenario. The grey elements were already covered in the previous deliverables. 14

Figure 4. Robot twin internal service structure. 17

Figure 5. Model of the Digital Twin and a Warehouse Controller services. 18

Figure 6. Internal model of the Warehouse Controller service. 18

Figure 7. Linked Data exchange approaches. 19

Figure 8. Fragment of the system architecture with a State Event Filtering service introduced. 20

Figure 9. PDDL-based planning ontology. 22

Figure 10. Imported subset of the planning ontology in Lyo designer. 23

Figure 11. Pictorial representation of the mission ontology. 24

Figure 12. A sample mission and the relation of its instances to the concepts of the underlying ontology. 24

Figure 13. SCOTT robot domain modelled in Eclipse Lyo as a domain specification. 27

Figure 14. ROS-based architecture for the simulated robots as well as their integration with the rest of the sandbox through digital twins. 29

Figure 15. ROS-based architecture for the physical robot scenario. 30

Figure 16. V-REP scene of the warehouse sandbox of the WP10. 31

Figure 17. High-level product picking task. 32

Figure 18. High-level product dropping task. 33

Figure 19. ROS Navigation overview. The move_base is the main ROS package that integrates the path planner and costmap packages. It also presents other dependency packages (blue and gray boxes). 34

Figure 20. Robot performing navigation. Robot is moving towards the green arrow. 35

Figure 21. Gateway architecture. 36

Figure 22. Components and interfaces of the planner service. 37

Figure 23. Plan synthesis process. 37

Figure 24. Output PDDL domain and problem from /pddlCreationFactory. 38

Figure 25. Output plan from /PlanCreationFactory. 39

Figure 26. Output annotated plan from /ValidatedPlanCreationFactory. 39

Figure 27. Samples of the robot's camera images used to evaluate the scene understanding methods. 45

Figure 28. Segmentation and scene graph outputs obtained from Mask R-CNN and intersection-based scene graph construction. 46

Figure 29. Object detection and scene graph outputs obtained from MSDN. 47

Figure 30. Caption generated from MSDN and the corresponding log probabilities. 47
Figure 31. Cartesian representation of scene graph when received by risk mitigation process. 48
Figure 32. MF of different linguistic variables used in the risk mitigation. 49
Figure 33. Scenarios used to evaluate the RL-based and FLS-based risk mitigation methods. 51
Figure 34. Local and distributed computation setups. 54
Figure 35. The duration of local and edge computational time in seconds. 55
Figure 36. NLP component interoperate from natural language to RDF triples. 56
Figure 37. XAIP tool user interface. 57
Figure 38. Explanation generated for why was robot1 used to pickup object 1 in the plan. 59
Figure 39. Explanation generated for why did robot visit waypoint WP1 rather than waypoint WP2. 59
Figure 40. Temporal plan verification and strategy synthesis. 61
Figure 41. Example use case of multiple robots leading to deadlock due to uncertainty during execution. 62
Figure 42. Global plan and local plans for individual robots. 63
Figure 43. Deadlocks due to Temporal Plan Deviations. 64
Figure 44. Timed Game Automata for Multi-Robot Coordination with Temporal Drifts. 65
Figure 45. Plan Strategy with Uncontrollable Transitions Temporal Drifts of ±10 units, Optimal Controllable Zones. 67
Figure 46. Number of Control Rules vs. Control Zones. 68
Figure 47. (a) Optimal / Worst Case Strategy Times for Increasing Control Rules, (b) Per Plan State Temporal Drifts against Control Rules. 69
Figure 48. Plan Completion Timelines with Deviations. 70
Figure 49. ROSGraph Representing Nodes and Messages Passed to the actionlib Server /Client. 71
Figure 50. An overview of warehouse with different kinds of active robots (in this case they are Autonomous Robot 1 and Conveyor Belt 1). 72
Figure 51. The front page of the dashboard where different levels of the system are listed. 72
Figure 52. The modal with secondary information about the truck including the total activity hours and the metric related to the sustainability. 73
Figure 53. Map view of the warehouse where different robots, their end destination and battery level are illustrated. 74
Figure 54. Details of Robot 1 are presented in a pop-up window. 75
Figure 55. Stacking different robots in the warehouse with each other to compare the performance of their combination for the year 2019. 76
Figure 56. A chord diagram was used to visualize the interoperability between all the available robots present in Warehouse 1 for the year of 2019. 76
Figure 57. The state tree and initial state after importing the domain and problem file. 80

Figure 58. Selecting an enabled action and deleted/added predicates shown on left. The tree highlights in red the current state after the action is executed. 81
Figure 59. Fish tank used in the demonstrator. 77
Figure 60. GUI showing the historical and current temperature in a graph format. 78
Figure 61. GUI showing the camera image taken each minute. 79
Figure 62. Architecture of the demonstrator. 81

LIST OF TABLES

Table 1. Summary of active use cases. 15
Table 2. Summary of non-active use cases. 16
Table 3. Description of hazards for collaborative operations 43
Table 4. Examples of rules to calculate the risk level given the obstacle properties. 44
Table 5. The evaluation result of scenario with known static obstacles in safety aspect. 52
Table 6. The evaluation result of scenario with unknown static obstacles in safety aspect. 52
Table 7. The evaluation result of scenario with humans in safety aspect. 53
Table 8. The evaluation result of scenario with humans in safety aspect. 53
Table 9. The evaluation result of scenario with humans and unknown static obstacles in safety aspect. 54
Table 10. Demo requirements score self-assessment for WP10. 84
Table 11. Demo and BB requirements score self-assessment for WP10. 86
Table 12. Demo and BB requirements score self-assessment for WP10. 86

1 EXECUTIVE SUMMARY

Deliverable D10.7 is the final deliverable for the work package 10 (WP10). It provides a summary of the work package itself as well as the work done in the SCOTT project. The report is concluded with a self-evaluation of the work package status.

The WP10 centres around the use case of an automated warehouse, where work carried out by humans and machines is planned, evaluated for safety and coordinated in order to fulfil orders and carry out different tasks in connection with that in a timely and trustworthy fashion.

The work progress was previously reported in the reports D10.2, D10.4, and D10.6. In the first year, the following progress was made:

- The architecture, which was initially designed and developed during year 1. The architecture is built on principles of linked data.
- The information model is described in Section 3.3. The information model is based on a three-level hierarchy of ontologies. Resources defined in ontologies that represent various domains are used across multiple components within the architecture and links between those resources are uniformly maintained.
- Planning Components were developed to enable generating automated plans across different devices according to the model definitions.
- Initial prototypes of Digital Twins were developed to establish Linked Data-based interfaces and pave the way to state synchronisation.
- Robot Software and V-REP simulation scenes were developed to establish a warehouse sandbox.

In the second year, the following progress was made in the following components:

- Robot ROS nodes: online safety, robot navigation, motion planning and scene understanding methods.
- Digital Twins: improvements the pub-sub protocols to improve Linked Data systems applicability in the CPS domain.
- GUI/Dashboard (monitoring and control) was introduced.
- NLP processor was introduced.
- V&V tool for planning domain was introduced.

In the final third year, the following progress was made:

- Verification and Validation (V&V) tool for planning domain: In V&V tool for planning domain a temporal plan verification was developed to avoid potential deadlocks and also investigated the explanation of generated plans.
- Robot ROS nodes: advances in the scene understanding and the risk management using AI.
- Digital Twins: State Event Filtering is introduced to improve the communication between twins, and the progress in the thing gateway is presented.

The work planning was done through the definition of requirements as part of the overall SCOTT process under WP6 and the definition of the use case scenarios detailed in the deliverables D10.1, D10.3, and D10.5. This deliverable summarises the progress reported in all previous reports,

grouped by the use cases addressed by the work. The progress on the applicable requirements is used to carry out the evaluation of the use case.

Keywords: warehouse, automation, CPS, robotics, ROS, Linked Data, AI planning, formal methods, Digital Twins, NLP.

2 OBJECTIVES

2.1 Deliverable Objectives

This deliverable provides an overview of the work package, a summary of the intermediary progress reports D10.2 [1], D10.4 [2], and D10.6 [3] as well as an evaluation of the final state of the work in the work package.

2.2 SCOTT Objectives

The work in WP10 has contributed to the overall SCOTT objectives:

- **Focus on wireless systems.**
  The elements present in the warehouse, such as the mobile robot, shelf and conveyor belt depend also on the wireless communication to enable the automation of the tasks.
- **Focus on European leadership and market opportunities.**
  The results of the work done in the WP10 are used in the projects to deploy robotic solutions at the Ericsson HW supply factories that are interested in automation. This opportunity can make the sandbox closer to real use cases and reusable between the factories.
- **Focus on smart sensors and actuators.**
  D10.7 presents work on the actuation in the *Digital Twins* use case and on the sensors in the *Risk Assessment for Safe Operations* use case.
- **Focus on Security, Safety, Privacy and Trustability.**
  Contributions in safety and trustability can be seen in many use cases presented in this report, mainly through the use of predictable formal methods to carry out planning and various checks and assessments, as well as explanations of synthesized plans.
- **Focus on including psychological and socio-contextual enablers for trust formation.**
  Trust related aspects are covered in the contributions to WP28 and are covered as part of the *Risk Assessment for Safe Operations* scenario.
- **Focus on eco-system with well-defined re-usable Technical Building Blocks.**
  The WP10 actively contributed to WP24 and participated in the other connected Building Blocks (see Section 5). In addition, the WP used open web-based technologies and common industry standards in order to maximise system-level interoperability.
- **Focus on solutions to be used in multiple industrial domains.**
  Problems of automating a set of industrial systems combined using equipment from various vendors in a trustworthy way are not limited to logistics. Additionally, several components such as planning services and Digital Twin related services can be reused in other domains owing to the use of open protocols and industry standards. Results of this WP are considered in the ongoing work on automation at Ericsson manufacturing facilities.
- **Focus on higher Technology Readiness Levels (TRLs).**
  The components of the WP10 sandbox environment are targeting TRL levels 5-7.

3 DESCRIPTION OF WORK

3.1 Work Overview

The work package focuses on research into warehouse automation, in particular on methods of interoperable, safe and trustworthy coordination of work in complex and heterogeneous environments.

The detailed end-to-end scenario can be found in the Section 3.3 of the D10.1, while the use case scenarios presented in deliverables D10.1, D10.3, and D10.5 go into detail of the use case specifics. At the high level, the work package considers a warehouse where machines and people should be coordinated in order to enable safe and timely movement of goods and fulfilment of orders.

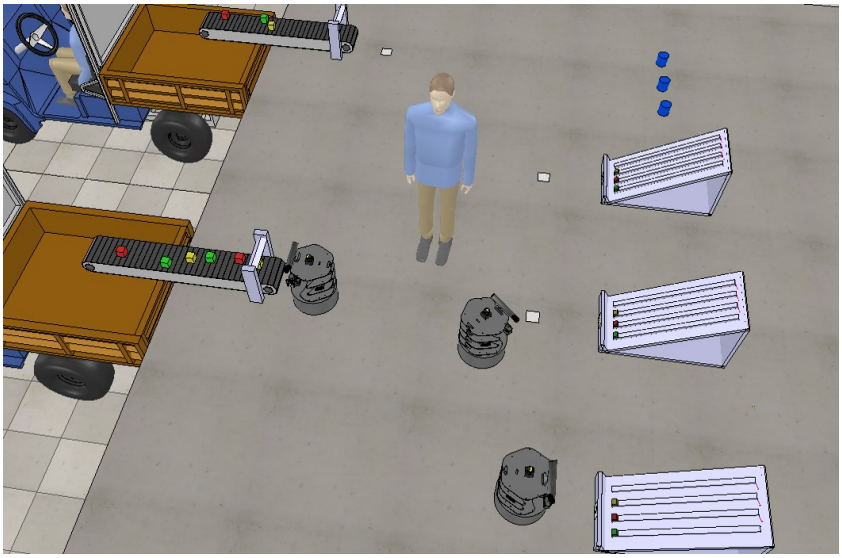


Figure 1. Illustration of the work package scenario.

Some of the technical objectives of this work include:

1. Utilization of open-source IoT components for monitoring and recording information from different sensors/actors in the “scene”.
2. Definition and prototyping of formal models that describe cross-functional characteristics between the different devices and the overall goal which would be the task that these devices aim at performing in combination.
3. Identification of safety critical cases/malfunctions that may cause abnormal behaviour.
4. Development and prototyping of pre-emptive mechanisms that can effectively (and within certain margins) eliminate or minimize effects of abnormal behaviour.
5. Development and evaluation of Visual Analytics techniques in order to assess the interoperability of CPS systems and CPS development environments.

Ericsson and KTH have actively been working on implementing the Final version of the *Warehouse Sandbox* to cover the requirements listed in the Section 2 of this deliverable. The main developments of the third year was in the plan V&V to check for deadlocks through game theory approaches; Progresses in the scene understanding and risk mitigation components of the safety

framework for the human-robot collaboration; and the advances in the digital twin communication by the introduction of state event filtering services.

RTE has developed the *Fish Tank* demonstrator presented in the Section 3.4. The demonstrator uses a fish tank as an example of any kind of habitable environment.

3.2 Use Case Architecture

The Warehouse Sandbox architecture (presented in Figure 2) was developed to also fulfil specific demo requirements:

- RQ 640 Monitoring and Control (Demo)
- RQ 641 Digital Twins (Demo)
- RQ 643 Pubsub (Demo)
- RQ 648 Robot & Robot Software (Demo)
- RQ 650 Testing Framework for PDDL Domains (Demo)
- RQ 681 Safe Operations (Demo)
- RQ 682 Collaborative Operations (Demo)
- RQ 689 NLP Component (Demo)

At the high level, the architecture spans multiple layers, as mapped onto the SCOTT HLA (High Level Architecture). The mapping can be seen in the Figure 2. Details of each layer as well as the description of its components can be found in the D10.5 Final Sandbox [4].

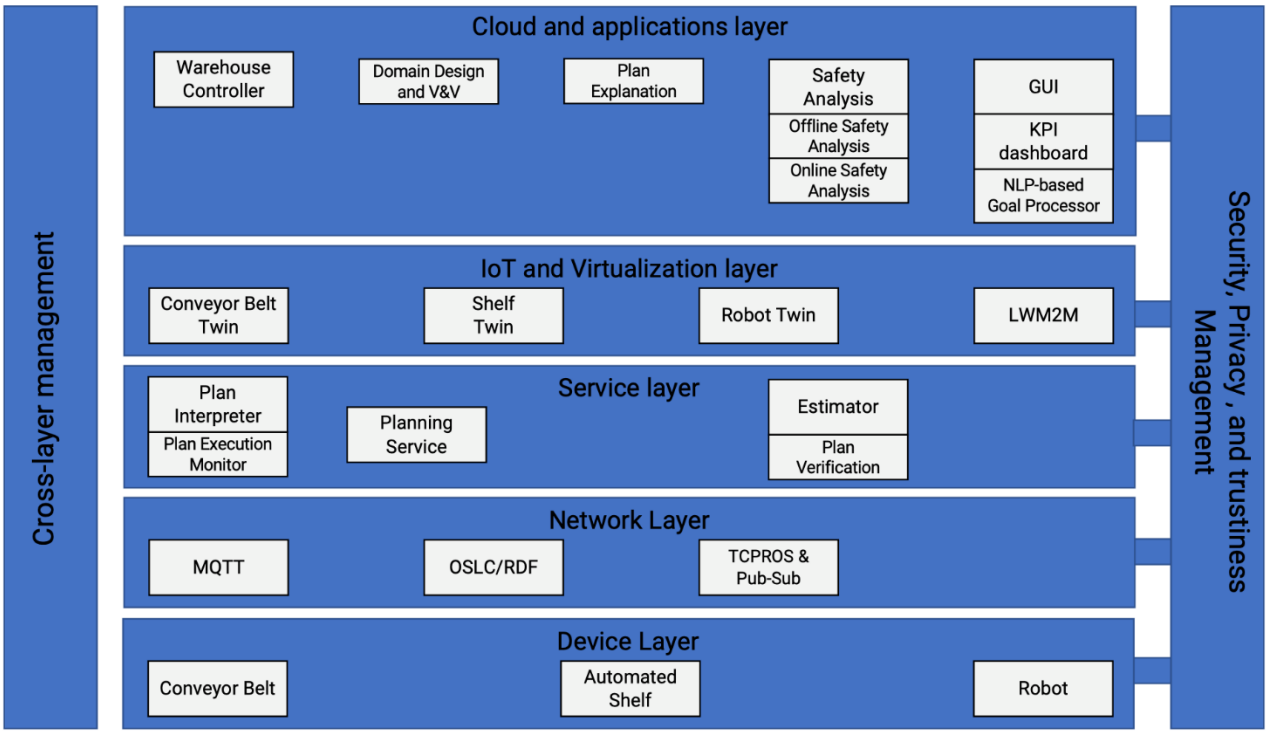


Figure 2. Warehouse Sandbox architecture levels as mapped onto the SCOTT HLA architecture.

The architecture illustrated in Figure 3 is the final architecture which is based on the architecture presented in D10.5 Final Sandbox Specification [4]. With respect to the SCOTT Bubbles, there is

no L0 bubble but L1 bubble and L2 cloud are clearly shown in the figure with the L1 Gateway being the single point of entry into the L1 Bubble.

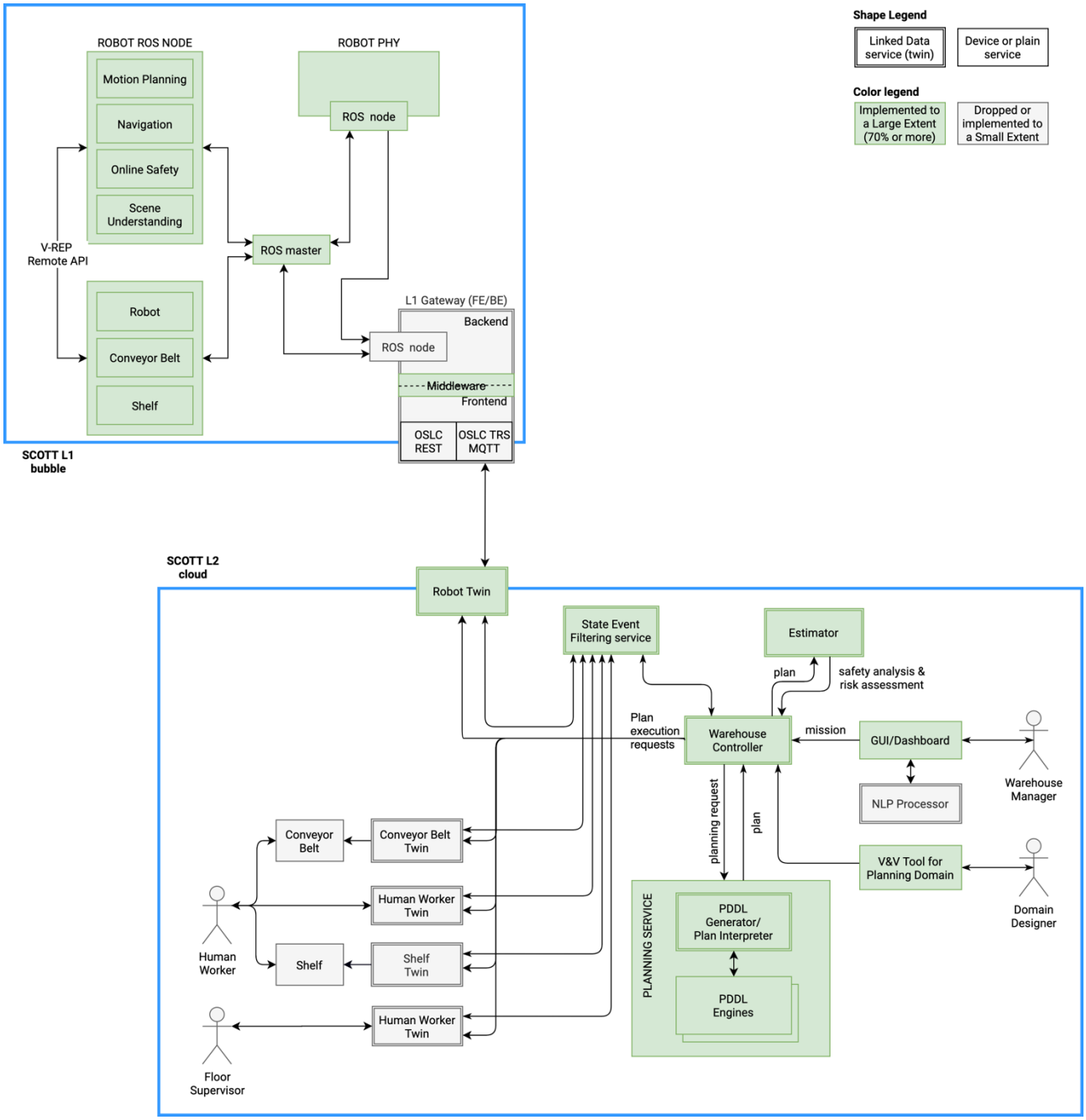


Figure 3. Warehouse Sandbox Architecture description. Green components are part of the final demonstrator scenario. The grey elements were already covered in the previous deliverables.

3.3 Use cases

3.3.1 Use cases overview

Use cases were defined in D10.1 [5] and refined in D10.3 [6] and D10.5 [4]. Below you will find a quick overview of the use-cases as introduced in each deliverable and short remarks where use cases were merged, reformulated or removed.

The finalized set of use cases as presented in the final version of the sandbox specification (D10.5):

Scenario	Comment	Key Requirements
Digital Twins	Introduced in D10.1.	REQ-641
Risk Assessment for Safe Operations	Introduced in D10.1.	REQ-640
Goal State Generation out of Natural Language Mission Specification	Was part of the Missions and Automated Planning scenario in D10.3. Expands and supersedes Task Level Planning and Safety Analysis of Task Plans scenario in D10.1.	REQ-689
Explanations for the synthesized plan	Introduced as part of the Missions and Automated Planning scenario in D10.3	REQ-646
Formal verification of Strategic Plan	Introduced as part of the Missions and Automated Planning scenario in D10.3	REQ-646
Design and V&V of planning domain model	Introduced in D10.3	REQ-650
Monitoring and Visualization of Warehouse KPIs	Introduced in D10.1	REQ-640

Table 1. Summary of active use cases.

Below you will find a summary of the use cases that were superseded, removed or changed name:

Scenario	Comment
Collaborative Operations	The scenario was introduced in D10.3, while the REQ-682 was added in Iteration 2. While this specific scenario was removed, the Collaborative Operations are considered in the scenarios *Formal Verification of Strategic Plan*, as well as *Digital Twins*, and *Monitoring and Visualization of Warehouse KPIs*, and *Risk Assessment for Safe Operations* (under *Distributed Solution of Risk Management Nodes*).

Scenario	Comment
Task Level Planning and Safety Analysis of Task Plans	The scenario was introduced in D10.1 and was superseded by the scenarios under *Missions and Automated Planning* in D10.3 as well as the safety part was expanded into scenarios *Risk Assessment for Safe Operations* and *Formal Verification of Strategic Plan*. This report will recap the *Task Level Planning* scenario status separately for the sake of completeness.
Optimized PDDL Problem Construction	Introduced in D10.3; removed because the requirement REQ-645 was dropped.
Reactive planning	Introduced in D10.3; removed because the requirement REQ-645 was dropped.
New ML model for object detection	Introduced in D10.1; removed because the requirement REQ-642 was dropped. The object detection, however, is performed as part of the *Risk Assessment for Safe Operations* scenario.

Table 2. Summary of non-active use cases.

3.3.2 Digital Twins

3.3.2.1 Overview

There are multiple definitions of a Digital Twin today. In WP10, we take a communication perspective on a Twin, where the first and foremost task of a twin is to keep its state in sync with the physical Asset and to provide an interface to it for the rest of the system components.

On one hand, a Digital Twin it is responsible for communication with the Asset it represents using asset-specific technologies. On the other hand, it provides a uniform way of integrating heterogeneous set of assets into a coherent system allowing for cross-asset applications and information exchange. So, in short, Digital Twin can be seen as a back-to-back information and control asset adapter. Additionally, a Digital Twin may maintain a model of its Asset and act on behalf of the Asset towards the rest of the system (e.g., a temperature sensor digital twin may report the last known temperature reading even if the physical sensor is in a power saving mode and not reachable at the time of the request). Due to their nature, Digital Twins are easy to replicate. This allows us to create virtual contexts and populate them with digital twins connected to simulators. Such setups can be used to analyse functionality of the rest of the system with high degree of accuracy and interface authenticity without involving physical assets.

3.3.2.2 Digital Twin service

On the application layer, Twins implement OSLC (Open Services for Lifecycle Collaboration) interfaces and exchange notifications about change events to the tracked resources over MQTT, as detailed in the following section.

The purpose of using OSLC as a basis for the Digital Twin interface is to provide a uniform REST-based[1] interface to a variety of heterogeneous Assets from different vendors, which forms a basis of *system interoperability*. Publish-Subscribe pattern is used to allow a Twin to keep an up-to-date copy of the subset of a corresponding Asset's state. Broader Publish-Subscribe pattern rather than a single point-to-point persistent connection also allows *other* Twins to be informed of the state changes in other Assets (Twins). This forms a basis of a *distributed autonomy*.

The Robot Twin service structure is shown in Figure 4. The service allows basic operations on a Plan resource that originate from a Warehouse Controller Service and full operations on the RegistrationMessage resource, which is used to keep track of Twin registrations.

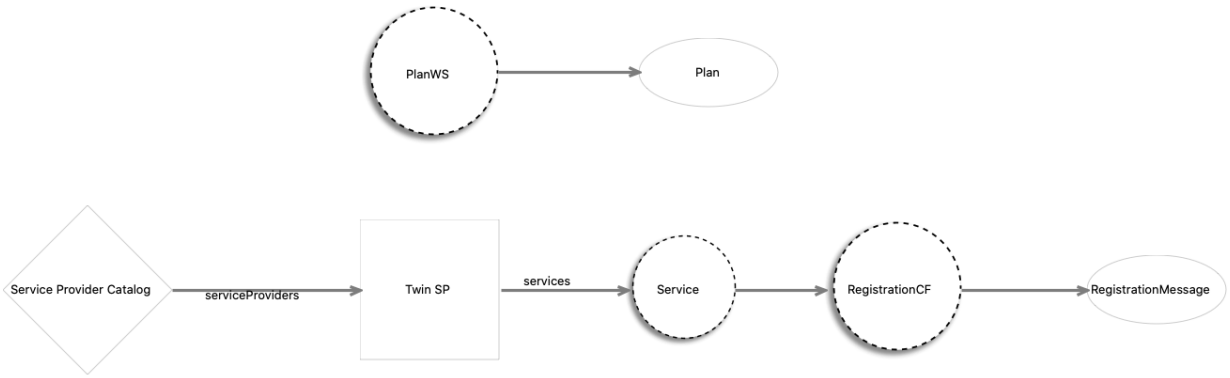


Figure 4. Robot twin internal service structure.

In year 1, an initial prototype of a twin was developed to perform LWM2M (Lightweight M2M, where M2M stands for machine-to-machine) communication with the ROS (Robot Operating System) Nodes for the corresponding things (robots and shelves), but it was considered to be too much overhead and the native ROS communication mechanisms were used in a subsequent prototype. ROS Messages needed for such communication were defined manually in year 1 (see Section 3.3.2.5.4 for an example). Each twin has a corresponding ROS Node in order to communicate with the ROS Node of the corresponding *Thing*. The description of the robot-related software can be found in Section 3.3.2.6.

3.3.2.3 Other services

All services were modelled, and code generated using Lyo Designer[2], an OSLC service modelling tool as shown in a high-level overview in Figure 5. In cases where implementation had to be done manually, empty service placeholders were modelled to allow client code to be generated.

[1] REpresentational State Transfer.

[2] https://github.com/eclipse/lyo.designer

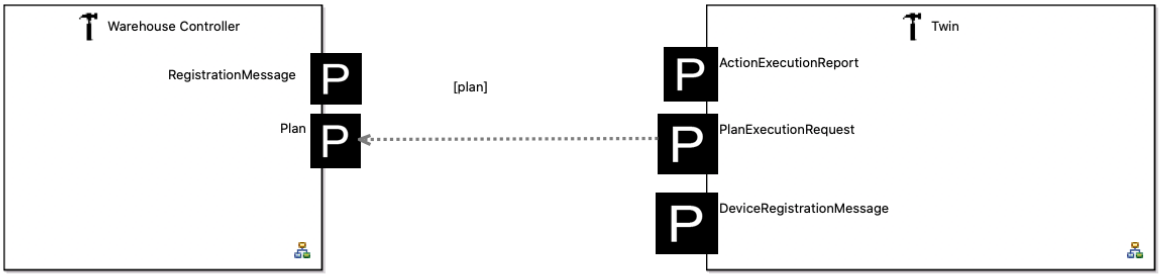


Figure 5. Model of the Digital Twin and a Warehouse Controller services.

The most important service that the Digital Twins communicate with is the Warehouse Controller service that registers Twins, initiates planning and distributes tasks. The internal structure of the service is shown in Figure 6. The Warehouse Controller operates on the following resources:

- DeviceRegistrationMessage in order to keep track of the Twin registrations, synchronised with the Twin service itself.
- PlanExecutionRequest in order to orchestrate plan execution between multiple entities through their Twins.
- ActionExecutionReport in order to track plan execution status across the warehouse.

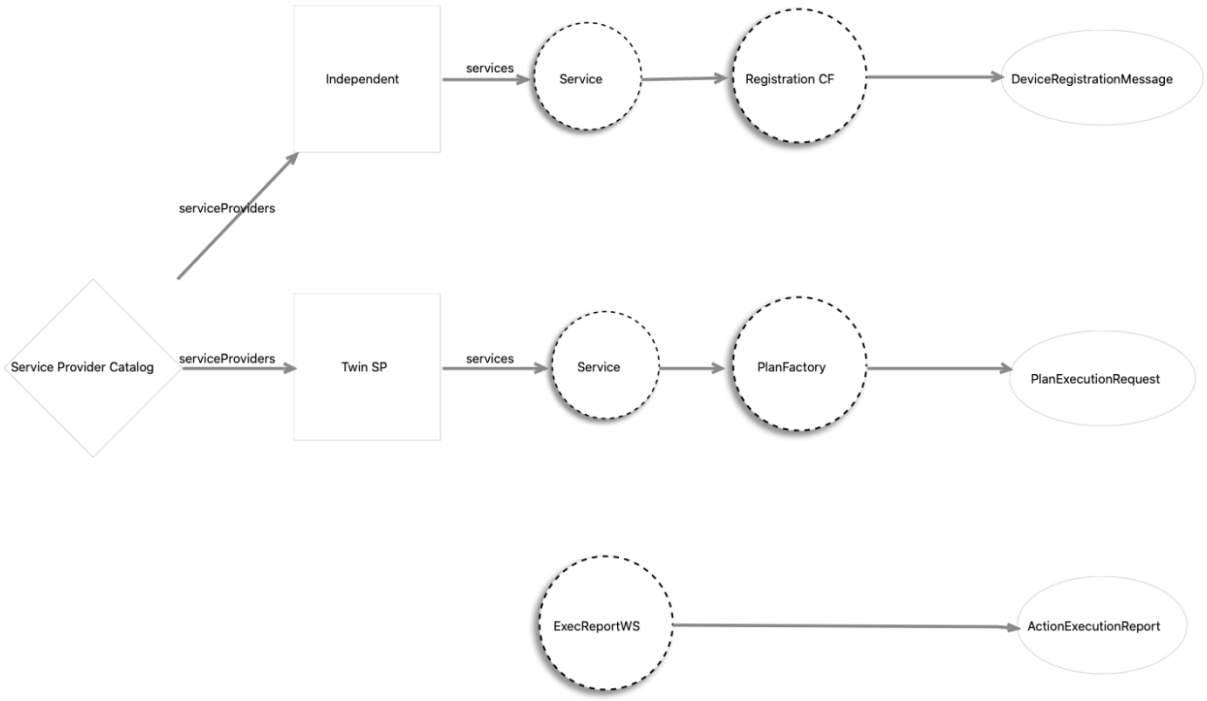


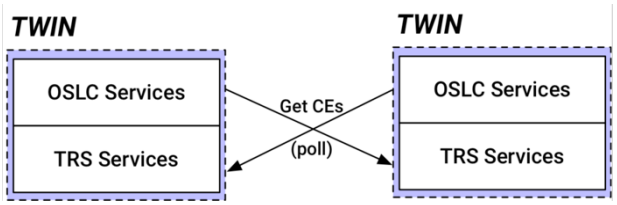
Figure 6. Internal model of the Warehouse Controller service.

3.3.2.4 Publish-Subscribe system and State Event Filtering service

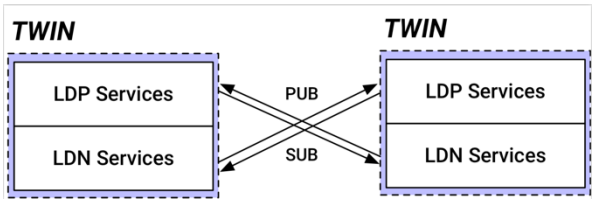
The **Publish-Subscribe communication** among the Digital Twins and other components is based on the OSLC TRS protocol. This protocol was designed for generic use between software tools. We have improved it concerning the specific requirements of the CPS since D10.2. These improvements were presented at the IEEE ICIT 2019 conference [7].

The **State Event Filtering service** was introduced in order to improve communication between Digital Twins. In order to take into account not only the state of the *Thing* mirrored by its Digital Twin but also the state of the other *Things*, the Digital Twin shall be able to obtain changes to the state from other Digital Twins as well as to distribute the information about the state changes of its Thing.

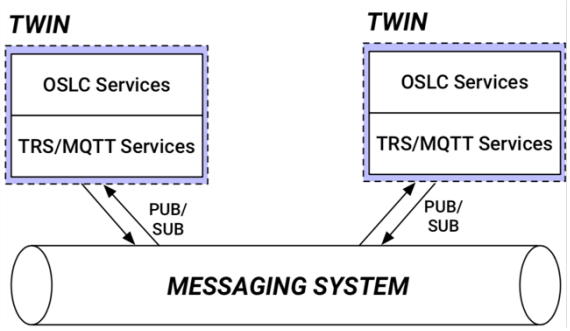
Such communication between the Digital Twins can quickly become expensive due to a number of subscribers subscribing to the updates (message fan-out) as well to load caused by excessive polling, depending on whether push or pull approach is used to distribute the updates. Push-based, polling-based systems as well as a mixed push-based system with a log are shown in Figure 7.



(a) Polling-based approach.



(b) Push-based approach.



(c) Mixed approach with a log-based polling system being converted into a push-based one.

Figure 7. Linked Data exchange approaches.

Finally, any improvements shall consider the slow rate of change of the industrial communication protocols and caution with which modifications to the standardised protocols are accepted. Systems addressing problems without requiring protocol modifications are preferred to the ones requiring them.

The system developed as part of the work package takes into account both push-based (Linked Data Notifications, LDN) and pull-based (Tracked Resource Set, TRS) protocols used in Linked Data. Communication systems based on Linked Data are specifically considered here because we made a decision to equip Digital Twins and other components in the system with application programming interfaces (APIs) based on Linked Data to improve interoperability between components in the warehouse. State Event Filtering (SEF) service can be seen within the system architecture in Figure 8.

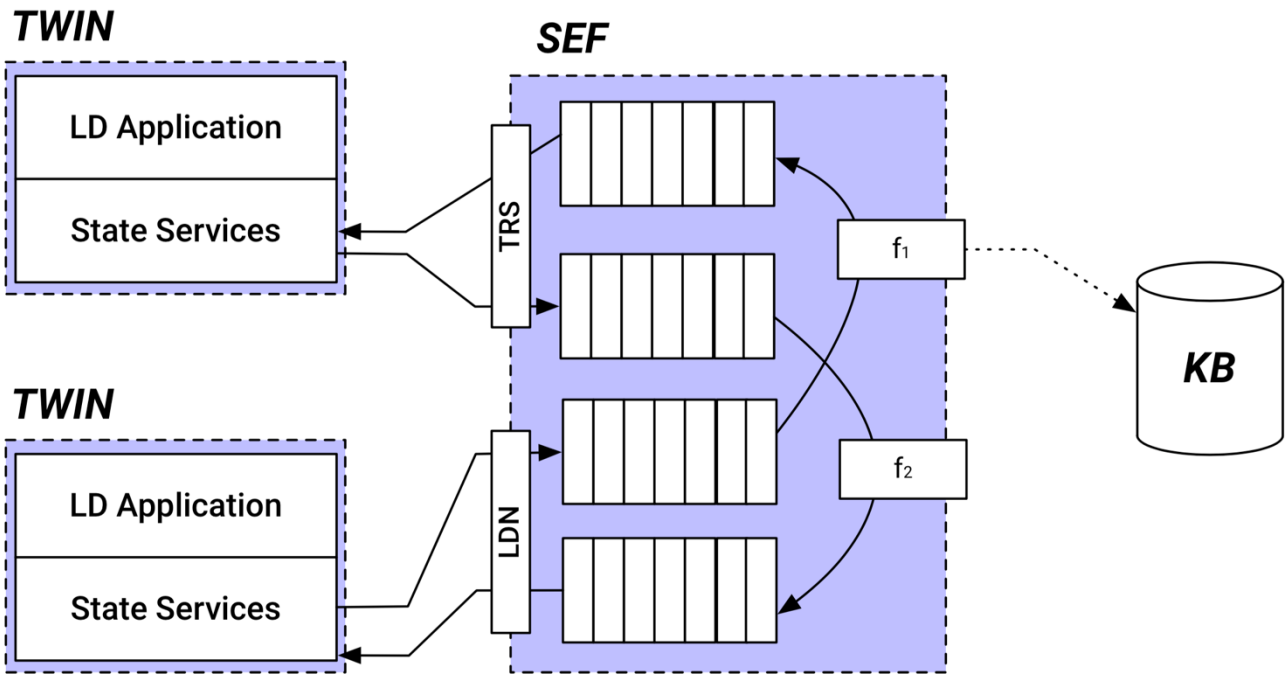


Figure 8. Fragment of the system architecture with a State Event Filtering service introduced.

The SEF service works simultaneously as a publisher and a consumer of the changes and directly connects to other publishers and consumers that require its services. Subscribers register their subscriptions and *filtering preferences.* As the published changes come in (via either LDN push notifications, fetching the TRS Change Log or by receiving newly appended TRS Change Events through a message broker), the SEF service filters them out according to the previously declared preferences. The preferences are declared as filtering functions of different types:

1. Primitive functions.
2. Functions with limited window.
3. Functions that can perform lookup.

The range of options for declaring filtering preferences allows to maintain the balance between the expressivity of the filtering operations against the overall performance of the system.

This work has been presented at IEEE International Conference on Industrial Informatics, INDIN'19 [8].

3.3.2.5 Domain modelling

Within the Warehouse Sandbox, different components designed independently would need to exchange information and provide an ability to perform Actions. In order to allow the system to plan and control system activity across various domains, a three-level information model is designed based on RDF-based web ontologies. Use of such ontologies ensures their portability and reuse.

At the first level, an ontology for a generic planning domain is defined (see Section 3.3.1). In our use-case, it is modelled after the Planning Domain Definition Language (PDDL) and its concepts. Using the concepts from this ontology, all kinds of plans would be formulated for the rest of the system.

To plan real Actions, they must be previously defined in an application-specific domain. For example, the action responsible for triggering the robotic arm movement shall be defined in the ontology, which is specific to the planning within robotic domain (second level).

Finally, this domain-specific planning ontology needs to be based on the ontology of the underlying domain, such as one that would define the classes and properties to describe robots and related concepts. In this case, an established ontology may be reused if it does not require drastic changes to the information model of the sandbox. Such ontologies constitute a third level of the information model (robotic domain is described in Section 3.3.3).

3.3.2.5.1 Planning Domain

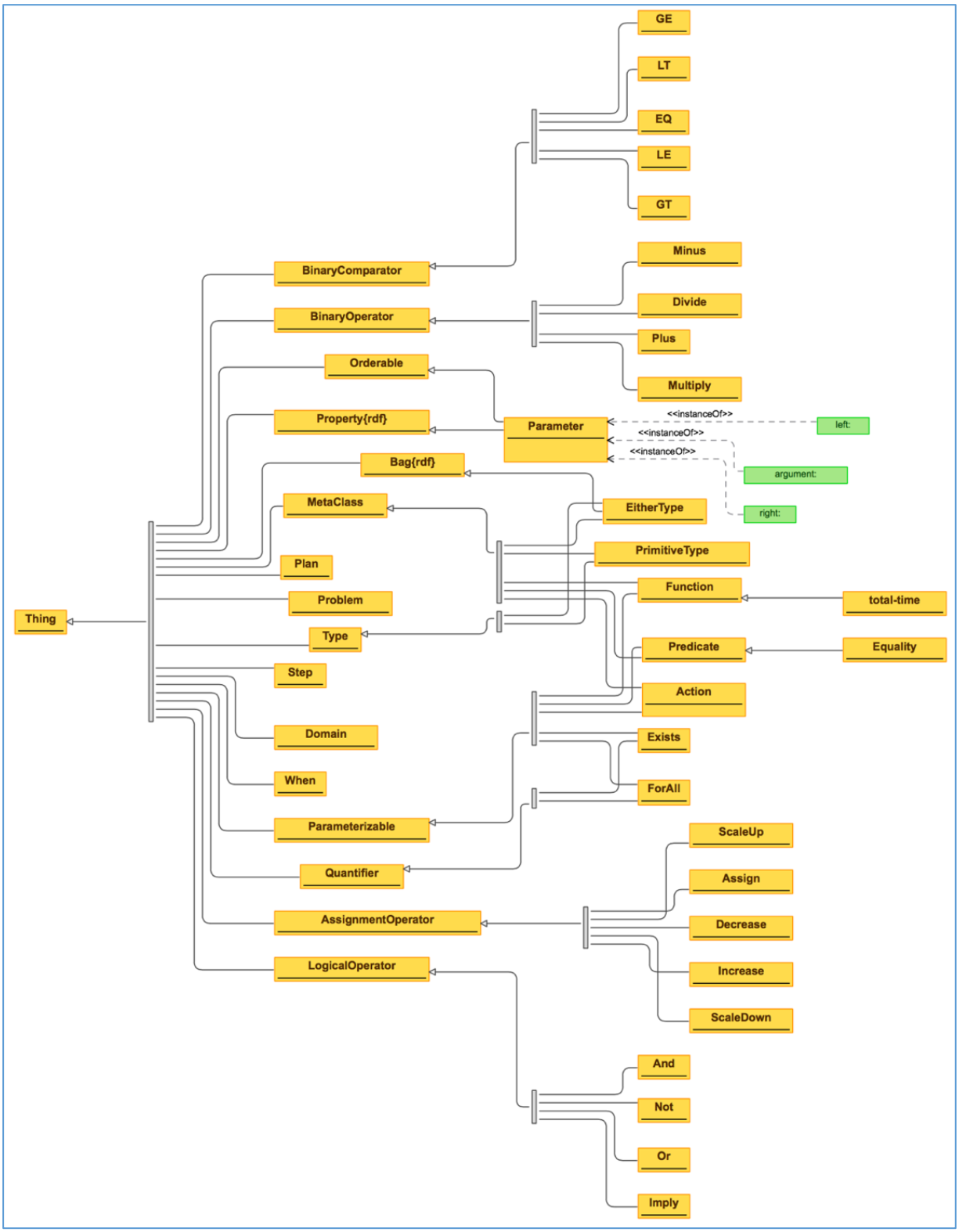


Figure 9. PDDL-based planning ontology.

The ontology for the PDDL domains is shown in Section 3.3.2.5.1. It was modelled closely after the concepts described by the Backus Naur form (BNF) in PDDL 2.1 [9].

The model was imported into Lyo Designer[3] (Figure 10) was then used to define the interfaces of the services. In addition to that, it was used to generate Plain Old Java Object (POJO) class definitions. This allows POJO instances to be manipulated programmatically in an ordinary fashion, while allowing the Lyo SDK to marshall and unmarshall these instances to and from one of the RDF model representations, such as Turtle, RDF/XML, JSON-LD etc. This, in turn, lowers the barrier for developing Linked Data-based integrations that interface legacy systems.

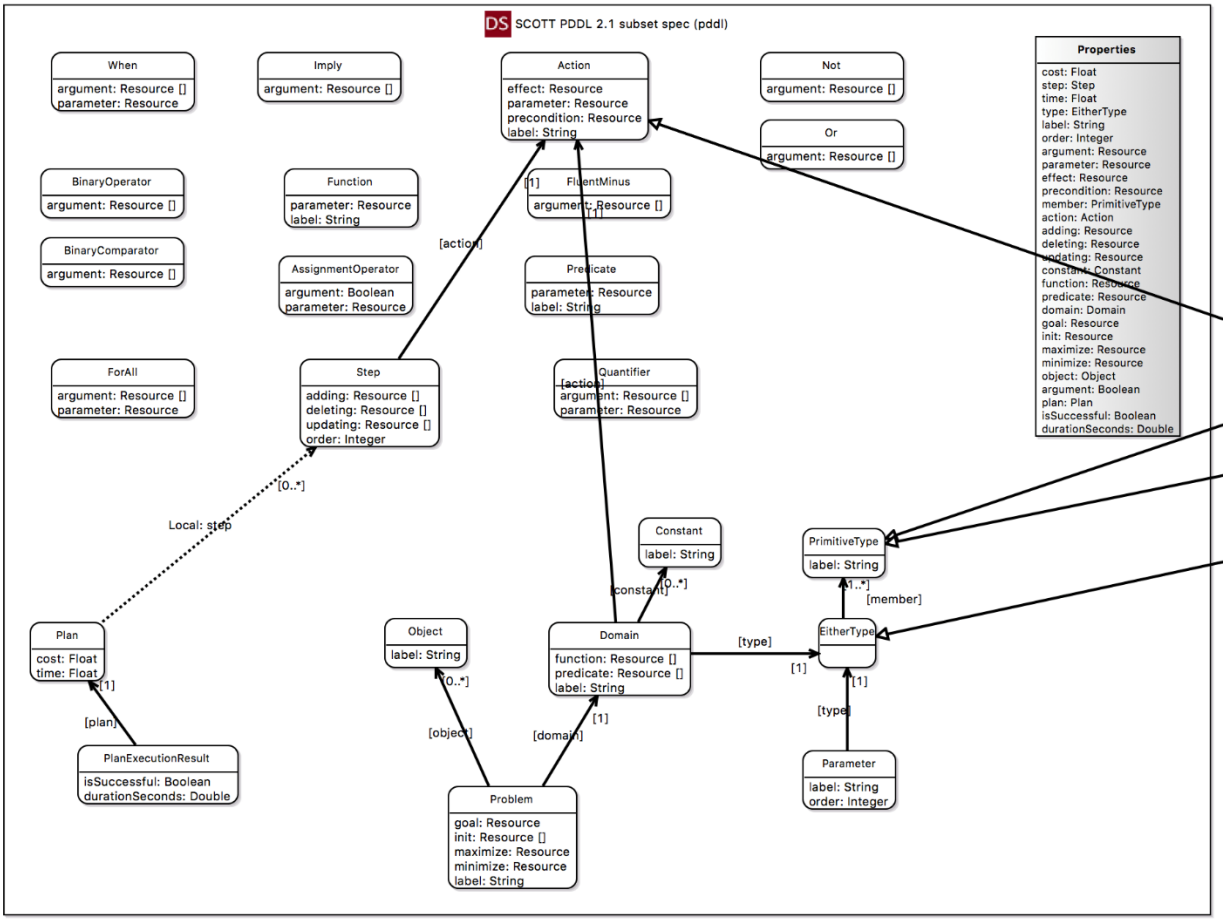


Figure 10. Imported subset of the planning ontology in Lyo designer.

The lines stretching outside the figure indicate the relations on the classes of the Planning ontology from other ontologies.

3.3.2.5.2 Mission Domain

One of the exploration threads in the project was about the Mission definition. It was necessary to provide the Warehouse Manager with a more compact and expressive Mission specification process than explicit, list-based missions. For example, the manager would be able to specify the following missions for the robot:

1. Pick a specific object O1 and deliver at conveyor belt 1.

[3] https://www.eclipse.org/lyo/

2. Pick any N objects from the warehouse and move to a destination.
3. Pick any N objects of type T and move to a destination.
4. Get N objects of type T from a Shelf S and move to a destination.

This ontology allows to define such Missions and conceptually resides at the second level of our information model. The pictorial design of the ontology is given in the Figure 11.

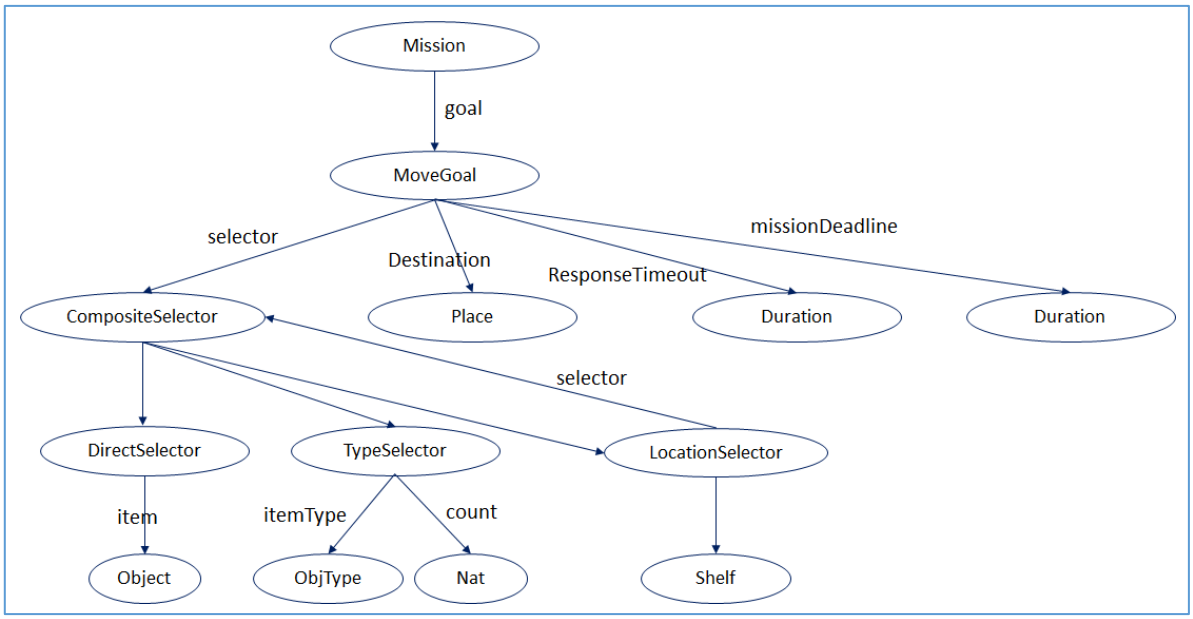


Figure 11. Pictorial representation of the mission ontology.

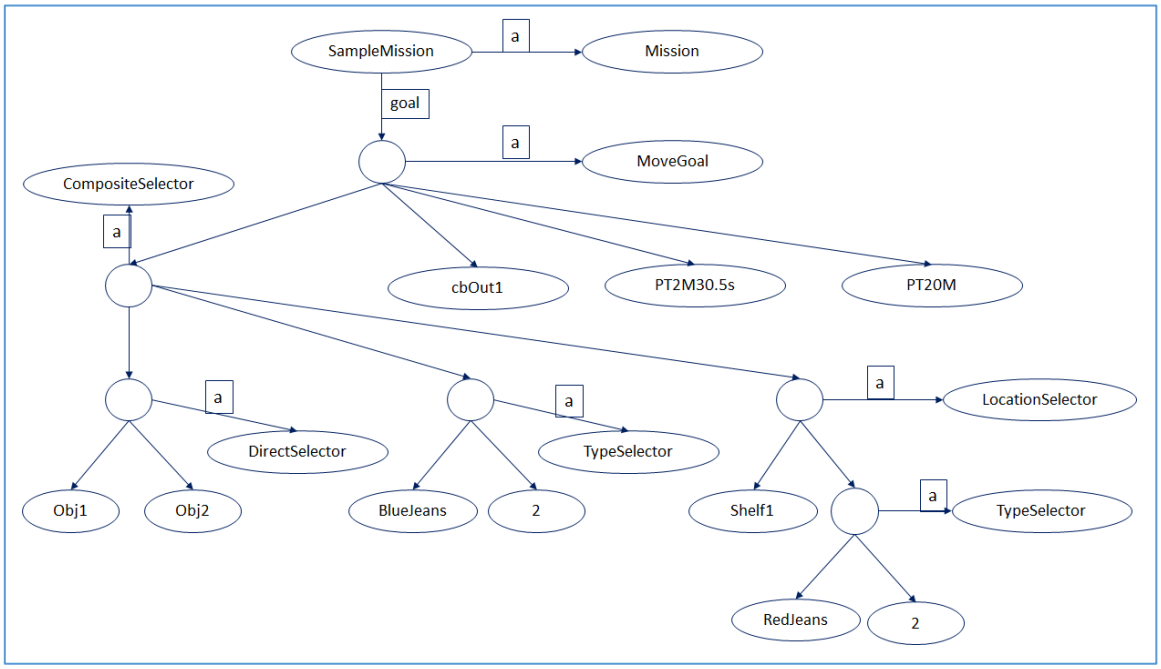


Figure 12. A sample mission and the relation of its instances to the concepts of the underlying ontology.

Below are a few examples of how the ontology could be used to specify the Missions. These examples are written in Turtle, a compact and readable RDF format.

The visualisation of the first mission shown below is presented in Figure 12.

```
:sampleMission a :Mission;
  :goal [ a :MoveGoal;
                :selector [ a :CompositeSelector ;
                        :selector [ a :DirectSelector ;
                                        :item <obj1>,  <obj2>
                         ] ,
                        :selector [ a :TypeSelector ;
                                :itemType <BlueJeans> ;
                                :count 2
                         ] ,
                        :selector [ a :LocationSelector ;
                                        :location <Shelf1> ;
                                        :selector [ a :TypeSelector ;
                                                        :itemType <BlueJeans>
                                                        :count 2
                                         ]
                        ]
                ]
                :destination <cbout1>  ] ;
                :responseTimeout "PT2M30.5S"^^xsd:duration ; # 2min 30.5s to produce a plan
                :missionDeadline "PT20M"^^xsd:duration . # 20 min to execute the plan
```

We give some more examples of selecting the objects in the following which shows the rich expressiveness of the ontology:

“Select N objects from shelf A and M objects from shelf B”

```
[ a         <CompositeSelector> ;
        m:selectors  [ a        <CountSelector> ;
                m:count     "N"^^m:integer ;
                m:selector  [ a         <LocationSelector> ;
                        m:location  []
                ]
        ] ;
        m:selectors  [ a        <CountSelector> ;
                m:count     "M"^^m:integer ;
                m:selector  [ a         <LocationSelector> ;
                        m:location  []
                ]
        ]
] .
```

“Select object ‘obj1’”

```
[ _:sel a :DirectSelector;
        :item _:obj1
].
```

“Select 2 Bluejeans objects (no matter where they are now)”

```
[_:sel a CountSelector ;
        count "2"^xsd:integer ;
        selector [ a TypeSelector ;
                itemType _:Bluejeans
        ]
] .
```

“Select 2 Bluejeans objects from shelf1”

```
[_:sel a CountSelector ;
        :count "2"^xsd:integer ;
        :selector [ a TypeSelector ;
                itemType _:Bluejeans .
                selector [ a LocationSelector ;
                        :location _:shelf1
                ]
         ]
] .
```

3.3.2.5.3 Robotic Domain

The robotic domain was modelled to satisfy requirements needed to carry out planning and communication with the Twin. The domain is shown in Figure 13.

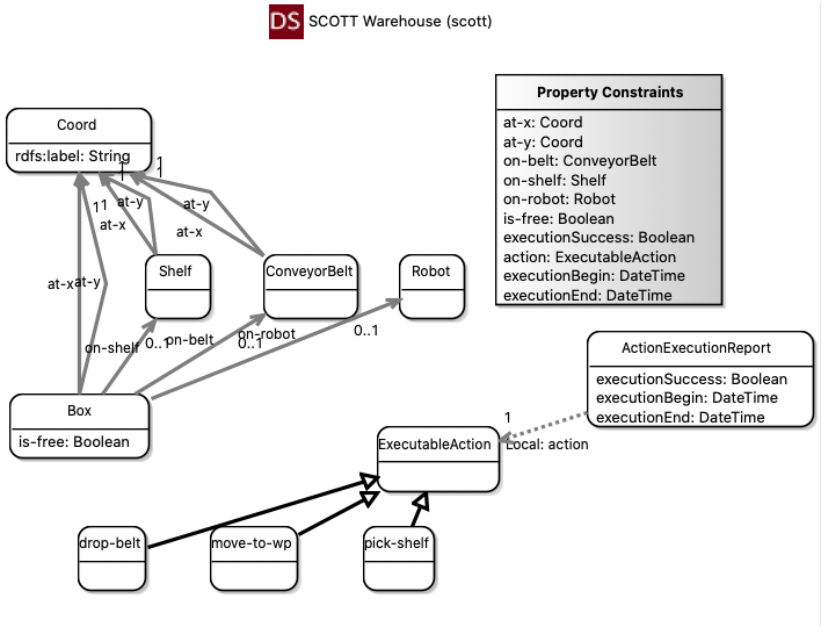


Figure 13. SCOTT robot domain modelled in Eclipse Lyo as a domain specification.

3.3.2.5.4 ROS Messages

Data shared between ROS Node is defined by ROS Message structures, which basically describes the data fields and their serialisation. ROS services in addition describe both request and response messages together. There are standard ROS message types and ROS service types (e.g. pose, velocity, geometry), but it is possible to create new ones. This can be used for mapping the resources defined in the domains presented in this section onto the corresponding ROS messages. An example below shows the ROS Service definition, which contains the description of both the request message as well as the response message. In this case, the service is a simple robot status retrieval service:

```
# Request
time stamp
string robot
---
# Response
time stamp
string task
string object
Pose waypoint
```

3.3.2.6 Robot and robot software

3.3.2.6.1 Overall architecture

The ROS framework[4] is used (ROS 1 specifically), among other things, to let the robots communicate with their respective digital twins. In our scenario the robot is a Turtlebot 2i[5], which is equipped with a robotic arm, two 3D cameras and a LIDAR (Scanse Sweep) sensor. The proposed ROS architecture was designed to support multiple robots, and to work with both real and simulated robots seamlessly. Moreover, for the simulated scenario the communication can be extended to other elements (e.g. shelf, conveyor belt and truck).

The main advantage of ROS is that, for most cases, it dispenses the necessity of developing low level algorithms by reusing code available in its repository. Additionally, ROS provides ready to use libraries (i.e. ROS packages) to be deployed on the robots. Another advantage of ROS a standard interface (e.g. ROS topics and ROS services) to perform communication between processes (i.e. ROS nodes).

Terminologies derived from ROS such as ROS topic, ROS message, ROS node, ROS master are out of scope of this report and its description can be found in [10].

In the warehouse scenario, there are two possible architectures using ROS, one for simulated environment and another for real robots:

- **Simulated Robot Scenario:** The robots and the warehouse are simulated within V-REP[6]. and all the necessary information is gathered from the simulator. In this scenario, a single *ROS Node* is used for the whole simulation which centralizes the communication between the digital twins and the simulated robots. All the physical components of the warehouse are modelled and simulated by V-REP, such as robots, shelves and conveyor belts. In this case, ROS is responsible for processing the simulated data generated by V-REP (camera, lidar) and for transmitting the appropriate controls to the robots. As such, all data generated by V-REP is converted into ROS Messages. The *ROS_INTERFACE* is responsible for providing V-REP the support for the ROS messages. To control the robotic arms, specifically, the *ROS API* is used which use ports to send the messages. In Figure 14 the software architecture of the simulated scenario is illustrated which has a single ROS Master as a single machine is used.
- **Physical Robot Scenario:** In this scenario, it is used real Turtlebot 2i robots equipped with their own onboard computer running ROS. Each robot has its own ROS Master and ROS Node. Therefore, the OSLC/MQTT layer is used to perform communication between robots, through their respective twins. The digital twins access the robot data from the *ROS Nodes* of each robot. As already mentioned, all the robot algorithms are developed through ROS libraries and, therefore, they should run the same algorithms of the simulated setup. In Figure 15 the software architecture of the real robot containing individual ROS Master for each robot is illustrated.

[4] ROS (Robot Operating System): https://www.ros.org/

[5] Turtlebot2i Mobile Robot Platform (Trossen Robotics): https://www.trossenrobotics.com/interbotix-turtlebot-2i-mobile-ros-platform.aspx

[6] V-REP simulator: https://www.coppeliarobotics.com

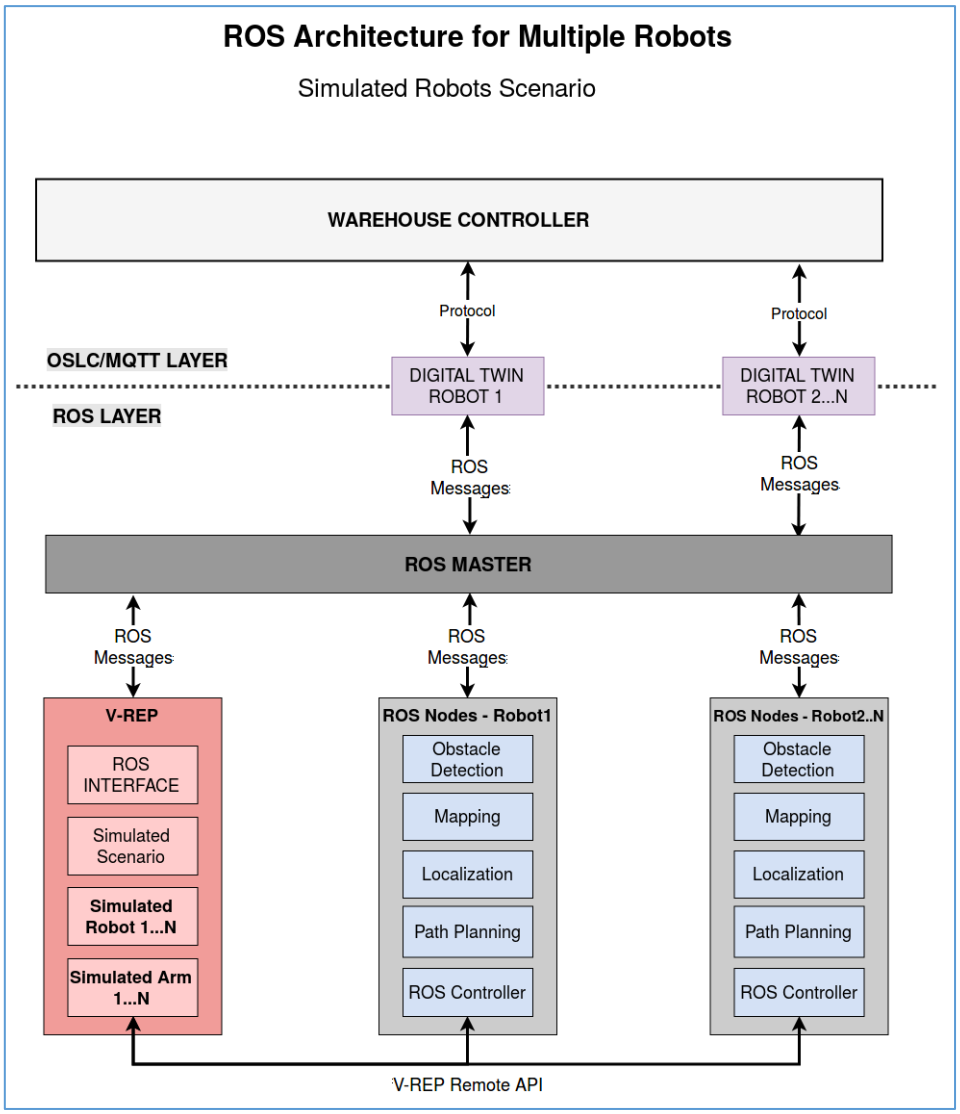


Figure 14. ROS-based architecture for the simulated robots as well as their integration with the rest of the sandbox through digital twins.

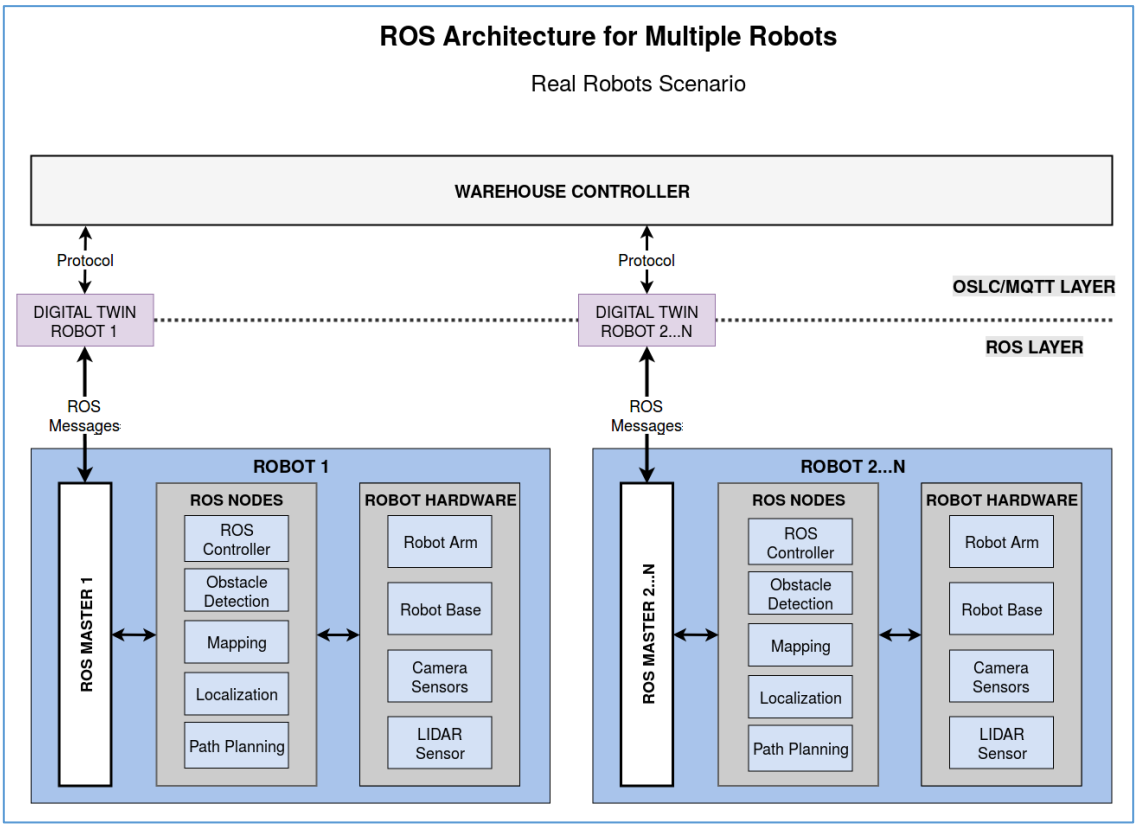


Figure 15. ROS-based architecture for the physical robot scenario.

All the robot algorithms are written through ROS libraries and executed as ROS Nodes. Some of the robot ROS Nodes are: obstacle detection, mapping, navigation and robot arm controller.

3.3.2.6.2 V-REP simulator

Before running experiments with real robots, it is less costly and safer to validate the code in a simulated environment. Here, V-REP simulator was chosen to perform all the validation experiments before deploying the developed methods in the real robot. A snapshot of V-REP simulation environment can be seen in Figure 16.

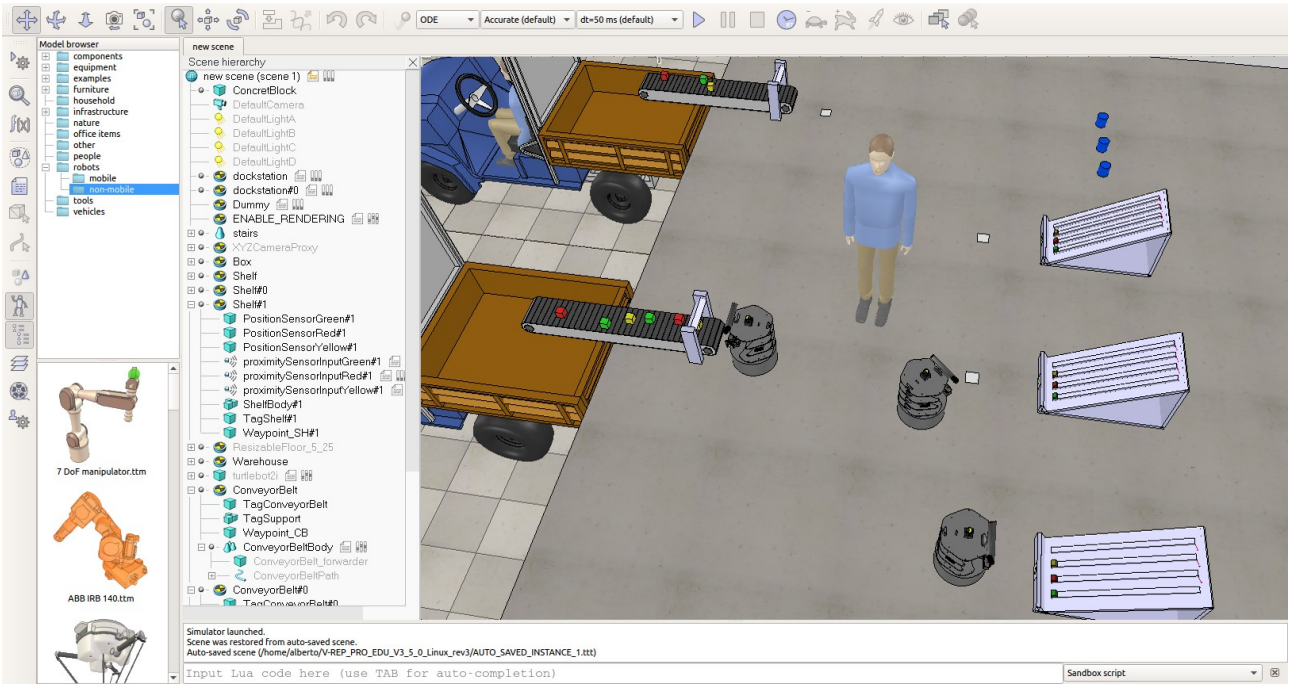


Figure 16. V-REP scene of the warehouse sandbox of the WP10.

The scene is simulated in a physically realistic manner. This means that robots, shelves, conveyor belts and products have their dynamic behaviour dictated by a physics simulation library embedded in the simulator. This dynamic behaviour of components in the scene can be modified at will in order to make the simulation more efficient and faster. E.g. it is possible to disable the physics of products when they are not being manipulated, which saves computational power since their physical behaviour do not have to be calculated at those times.

Robots are controlled from outside the simulator, by means of algorithms implemented in ROS Nodes. For more details about it please see the previous section on robot and scene ROS Nodes.

The main reason for choosing V-REP is the presence of many ready-to-use models, possibility to draw new models and demands relatively less computational power (compared to Gazebo[7] based on user experience).

3.3.2.6.3 ROS Nodes

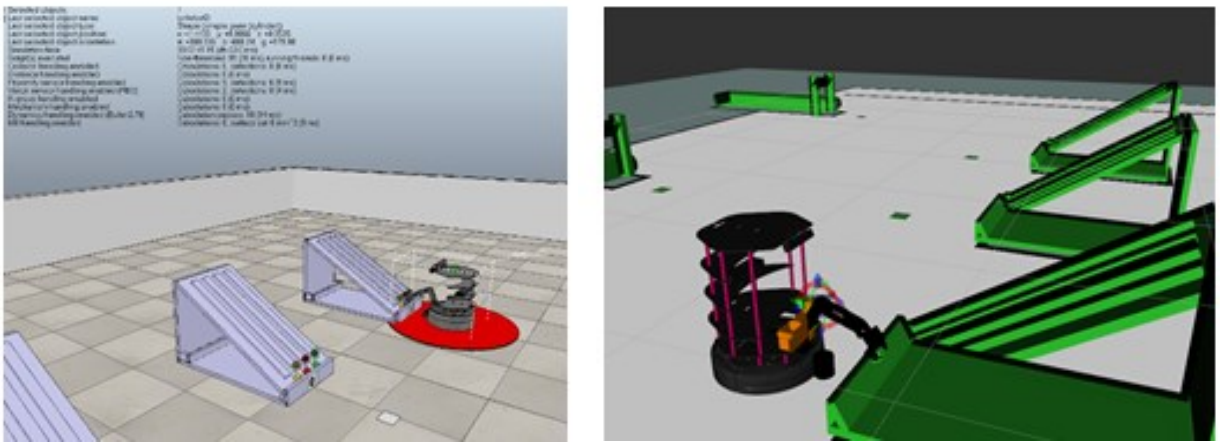
ROS nodes are the processes that run on the mobile robot's computer or in the simulated environment. These nodes enable the robot to accomplish tasks specified in the incoming plans. Some of the nodes developed in the WP10 are scene understanding, risk management (online safety), robot navigation and robot arm manipulation (motion planning). These nodes are described as follows:

Robot Arm Manipulation (Motion Planning)

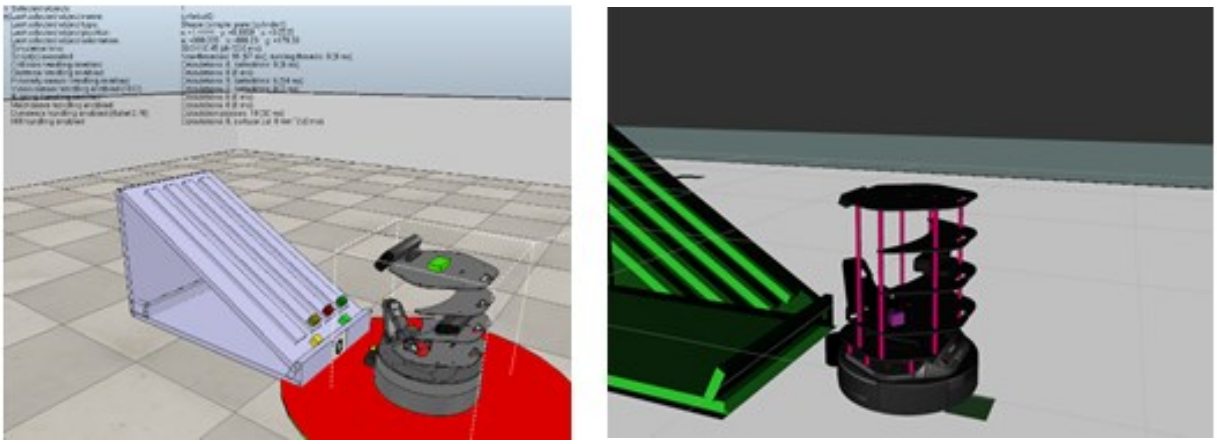
In robotics, manipulation usually refers to the process of moving and rearranging objects in the environment [11], a task usually executed by a robotic arm. A manipulation task may take into

[7] Gazebo simulator: http://gazebosim.org/

account factors like physical characteristics of contact between the robot and environment, robot and the object to be manipulated and even between two collaborative robots.



a) Robot grasping product from the shelf. On the left, the V-REP view and on the right, the planning scene state.



b) Robot storing product on its tray. On the left, the V-REP view and on the right, the planning scene state.

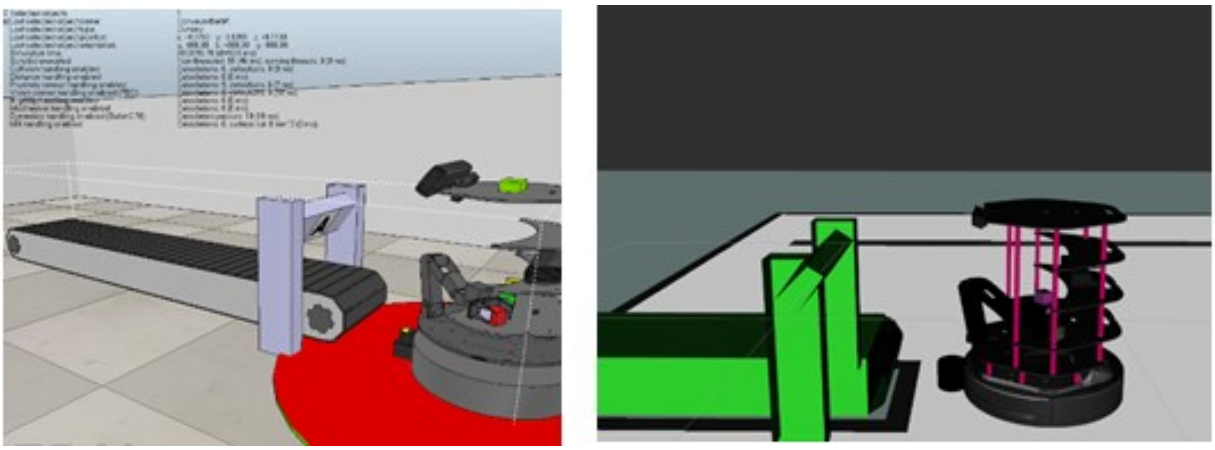
Figure 17. High-level product picking task.

In the present scenario, the simplified manipulation task consists of picking a cubic product (green, yellow and red) from a shelf (Figure 17) and dropping it on a conveyor belt (Figure 18). To complete the picking, the robot must grasp a product from a shelf (Figure 17 a) and place it (Figure 17 b) on its tray. For the dropping, the robot must pick the product from its tray (Figure 18 a) and drop it on a conveyor belt (Figure 18 b). To complete the tasks, the robot arm must be able to execute two fundamental jobs: pick and place. In general, to execute them the robot needs to:

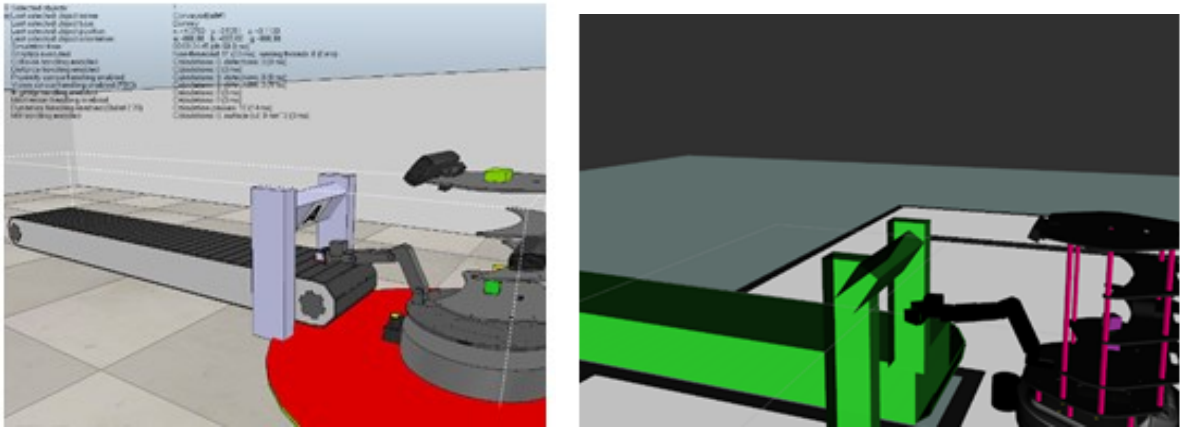
- Plan the end-effector pose to grasp the target object;
- Transform the goal pose to a joint state pose (inverse kinematics);
- Calculate a path to a desired pose;
- Check for collision-free paths;
- Check for joints constraints;
- Follow the generated path within bounds;

- Calculate the contact force needed to pick the product;
- Control the end-effector.

To simplify the manipulation task for the presented scenario, contact forces between the arm and the manipulated objects are unbounded, and the grasp poses are pre-defined.



a) Robot grasping product from its tray. On the left the simulation and on the right the planning scene state.



b) Robot dropping product on the conveyor belt. On the left the simulation and on the right the planning scene state.

Figure 18. High-level product dropping task.

Robot Navigation

The robot navigation node is essential to enable the mobile robot to move around the environment (e.g. deliver the product from the shelf to the conveyor belt). The first step of robot navigation is to calculate the path from a given origin to a destination [12]. To make this work, the robot relies on different ROS nodes, such as localization (amcl package), odometry, 2D map of the environment and sensor data. The output of the navigation is the robot control (i.e. robot's wheel speeds). Figure 19 presents an overview of the ROS move_base package from the navigation stack.

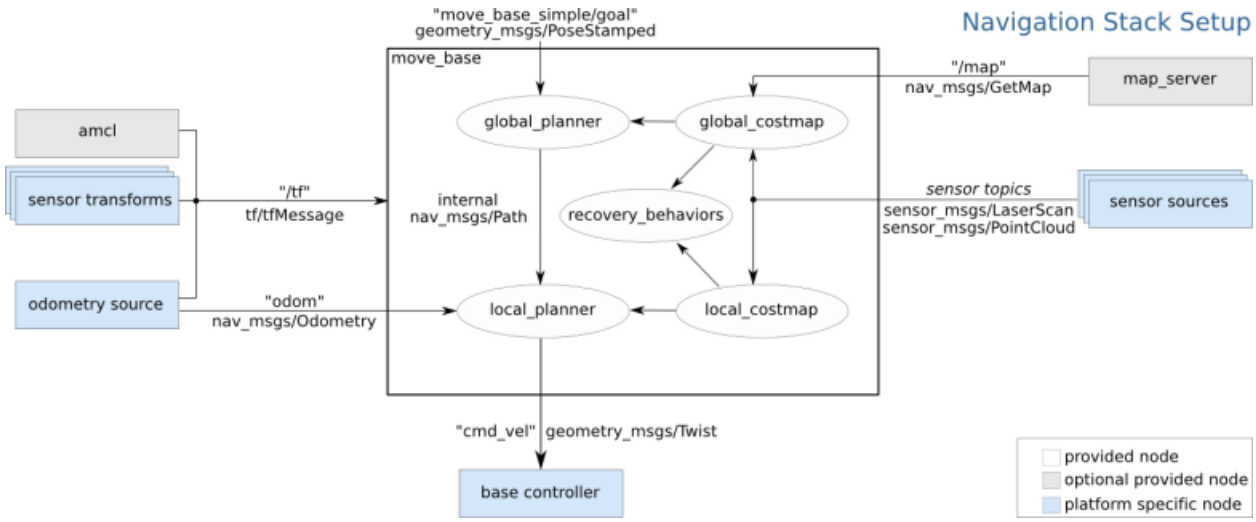


Figure 19. ROS Navigation overview[8]. The move_base is the main ROS package that integrates the path planner and costmap packages. It also presents other dependency packages (blue and gray boxes).

The main components of the navigation stack are:

- **Costmap:** Generates a map that inflates the obstacles detected in the environment. There are two costmaps: a local one that is constantly updated by using the sensor data to detect closer objects and dynamic objects; and a global one that is static and generated from the 2D map of the environment.
- **Path Planner:** Generates the path that the robot must follow to reach the target. This path takes as input the robot properties (e.g. holonomic/non-holonomic, maximum/minimum speed and acceleration) and the costmap. As in the costmap, there are two path planners, a local one that uses the local costmap to avoid dynamic obstacles and the global one that uses the global costmap as input.

In Figure 20, the robot navigation execution is shown. The robot is moving towards the goal pose depicted by the green arrow. Red and blue lines correspond to the local and global paths, respectively, generated by the navigation planner. The black and grey areas are the 2D grid map used to generate global costmap (blue areas). Local costmap represented by the squared area around the robot, is generated by the sensor data (red points - lidar data in this scenario).

[8] Figure extracted from http://wiki.ros.org/move_base

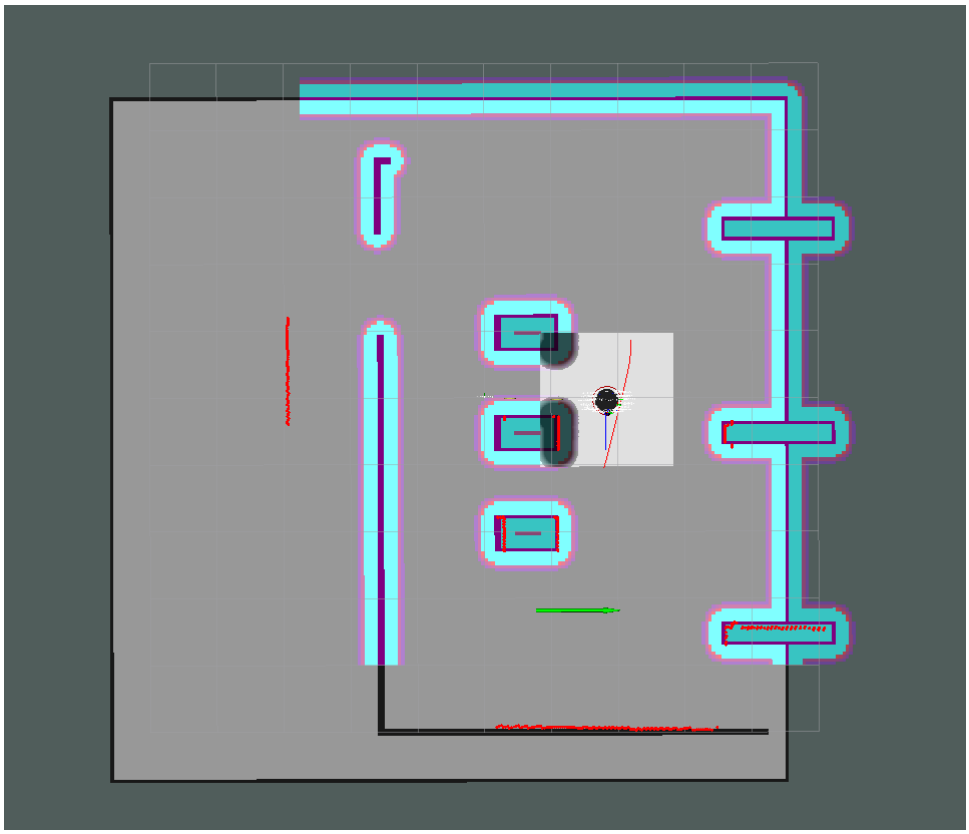


Figure 20. Robot performing navigation. Robot is moving towards the green arrow.

Scene Understanding

This node is described in the Risk Assessment for Safe Operations scenario.

3.3.2.6.3.1 Gateway service

The development of the Thing Gateway between things that do not have a Linked Data interface and Digital Twins is ongoing. The architecture of the Gateway Backend and Frontend has been defined (as shown in Figure 21), specifically designed to take advantage of the following aspects:

1. The Frontend can be written in a majority of popular programming languages (the only requirements are support for the ‘msgpack’ format and a ZeroMQ binding; there are over 30 languages supported). This enables the developers to use native SDKs provided by Thing vendors, also including the use of legacy technologies.
2. The Backend can be written in the language suitable for its need, taking advantage of the best Linked Data libraries, such as the Java ecosystem.
3. The Backend and the Frontend are coupled in a way that is aligned with the cloud-native deployment architectures, enabling both of the components to be placed in a same Kubernetes Pod.
4. Dynamic library loading nature of the Backend enables it to be written once with Frontend developers supplying class libraries defining their ontologies and shapes.

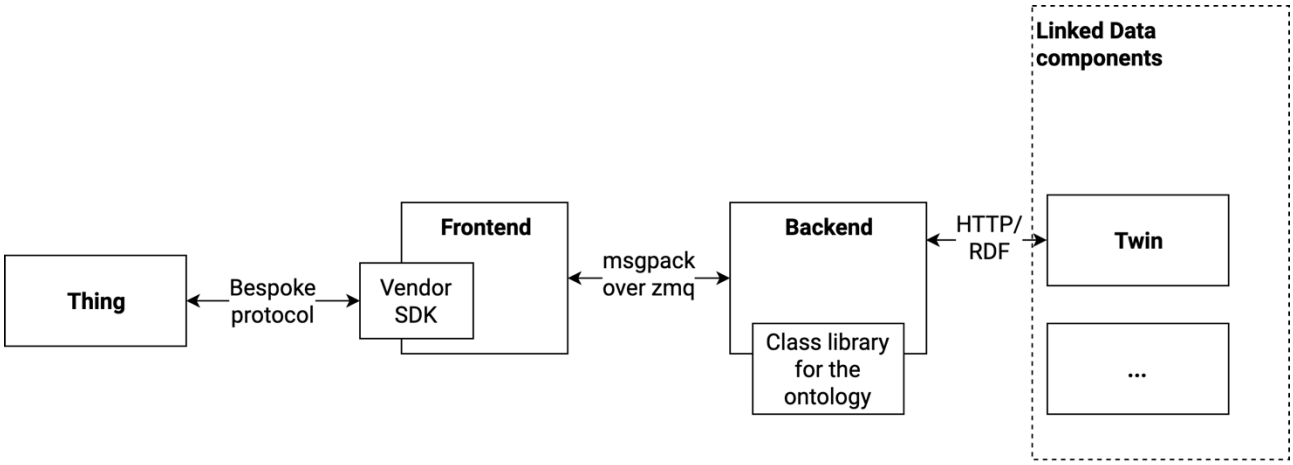


Figure 21. Gateway architecture.

3.3.3 Task Level Planning

Planning is carried out through a Linked Data-based service that provides an HTTP/REST API to a number of planners compatible with the Planning Domain Definition Language (PDDL) specification as well as provides an additional functionality of estimating and upper-bound of the plan execution. The planner service and the estimator are described in the subsections below.

3.3.3.1 Planner Service

Overview

The Planner service provides general purpose planning functionality to other components in the project. It is implemented as a number of OSLC adapters (based on OSLC Prolog library) and provides four endpoints (see Figure 22). Under the hood, planning service uses Metric-FF planner[9] and VAL validator[10].

Metric-FF is a plan synthesis software which derives plans from domain and problem files specified in a standard language called PDDL (planning domain definition language) using various heuristic search strategies.

VAL is a plan validation tool which is used to validate whether a user specified plan indeed achieves the Goal state from the specified Initial state. For a plan derived from a planner, it is expected that the validation process would be trivially satisfied (unless the planner has bugs), however, VAL also outputs detailed reports of the preconditions and effects.

The /pddl endpoint serves OSLC documents created according to PDDL ontology. The /pddlCreationFactory endpoint provides auxiliary function of converting OSLC planning problem and domain documents to the PDDL syntax supported by the internally used Metric-FF and VAL tools. The /planCreationFactory endpoint allows to create plans according to given planning domain and problem definitions. The /validatedPlanCreationFactory is an extended version of the /planCreationFactory is the sense that the plans generated by it are augmented with the information about state changes after every step of plan execution. The endpoints are visually shown in Figure 22.

[9] https://fai.cs.uni-saarland.de/hoffmann/metric-ff.html

[10] https://github.com/KCL-Planning/VAL

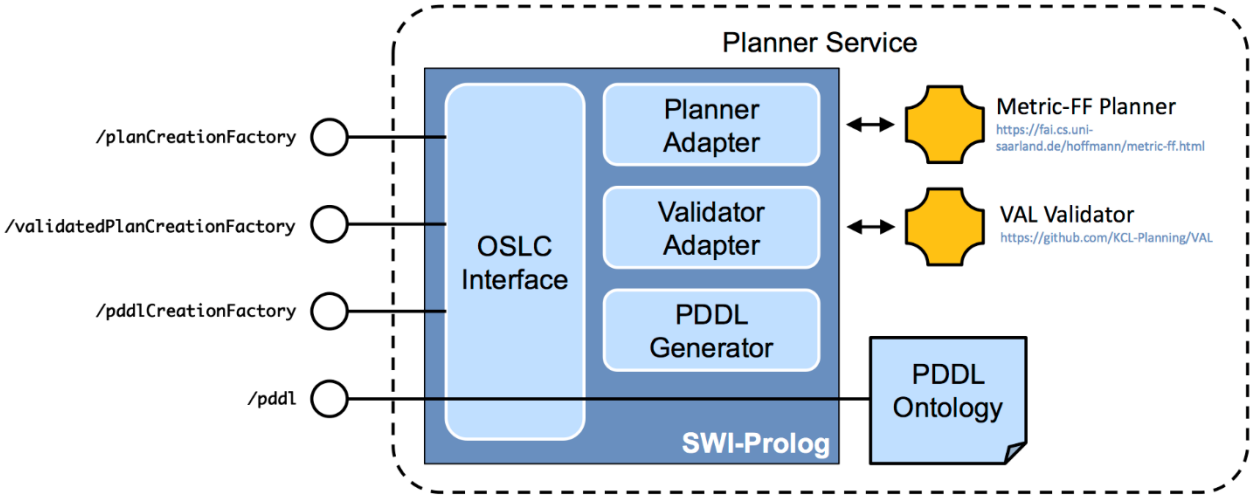


Figure 22. Components and interfaces of the planner service.

Planning process

The process and data flow within the planning service are shown in Figure 23. Plan creation factories accept planning domain and planning document in one of the RDF formats compliant with to the service's PDDL ontology. The PDDL generator converts input documents to the PDDL textual format (domain.pddl and problem.pddl). Planner adapter invokes Metric-FF planner, provides converted document as input, and interprets planner output by converting it to plan document according to the PDDL ontology. In case of creating a validated plan, the validator adapter is subsequently used to invoke VAL validator. The adapter generates input compatible with the syntax expected by VAL and augments the plan RDF document according to the output.

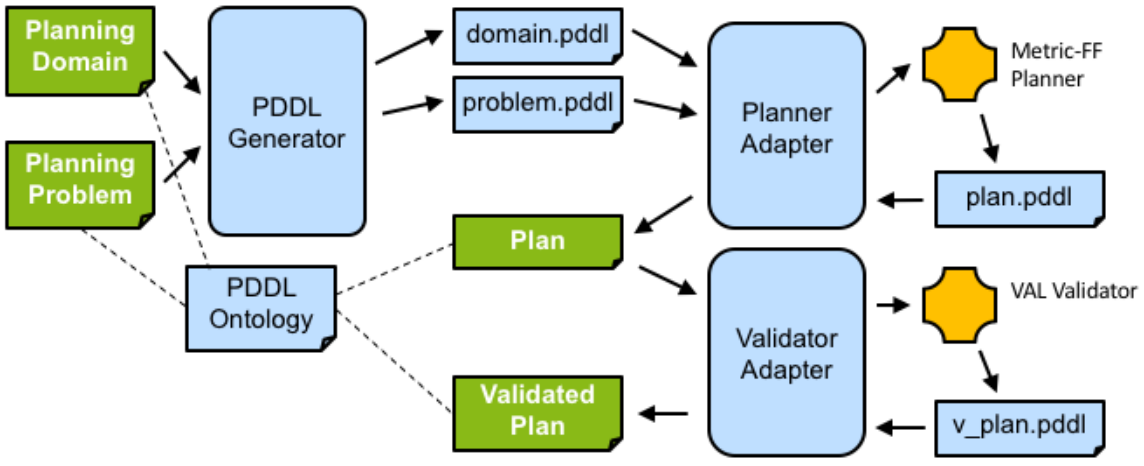


Figure 23. Plan synthesis process.

An example of PDDL generation can be seen in Figure 24. The PDDL domain and problem were obtained through the REST interface endpoint /pddlCreationFactory of the Planner service.

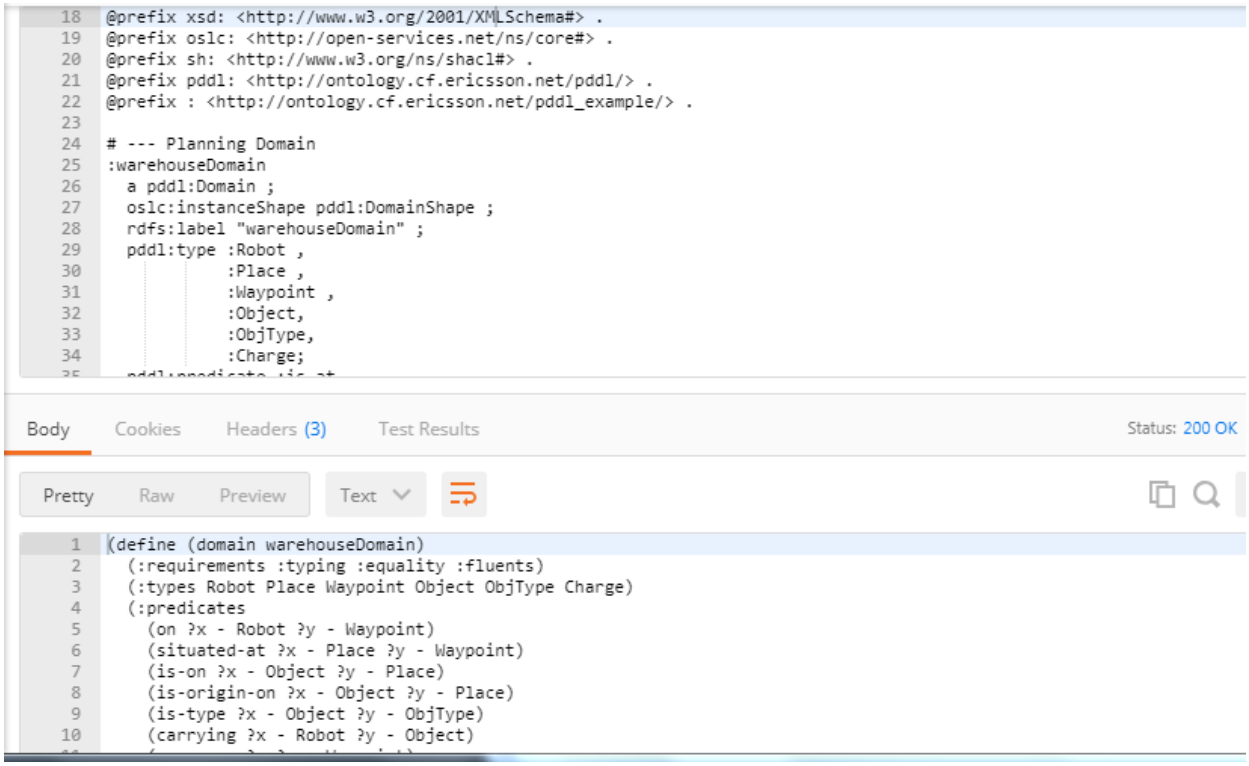


Figure 24. Output PDDL domain and problem from /pddlCreationFactory.

The plan synthesized by the Planner Service is represented according to a planning ontology described in the Section 3.3.2.5.1 (See Figure 25). This is annotated by additional predicates that are expected to be added or deleted as an Effect of the Actions in each step in a *validated plan* (shown in Figure 26).

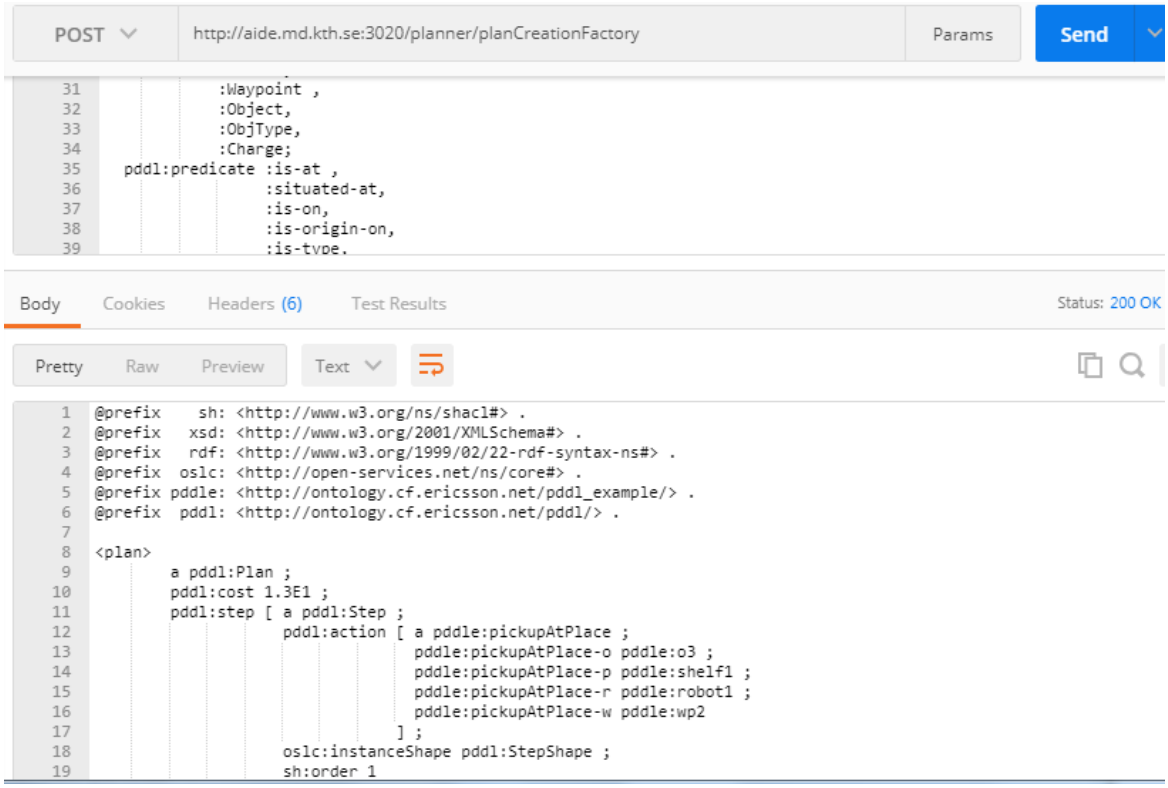


Figure 25. Output plan from /PlanCreationFactory.

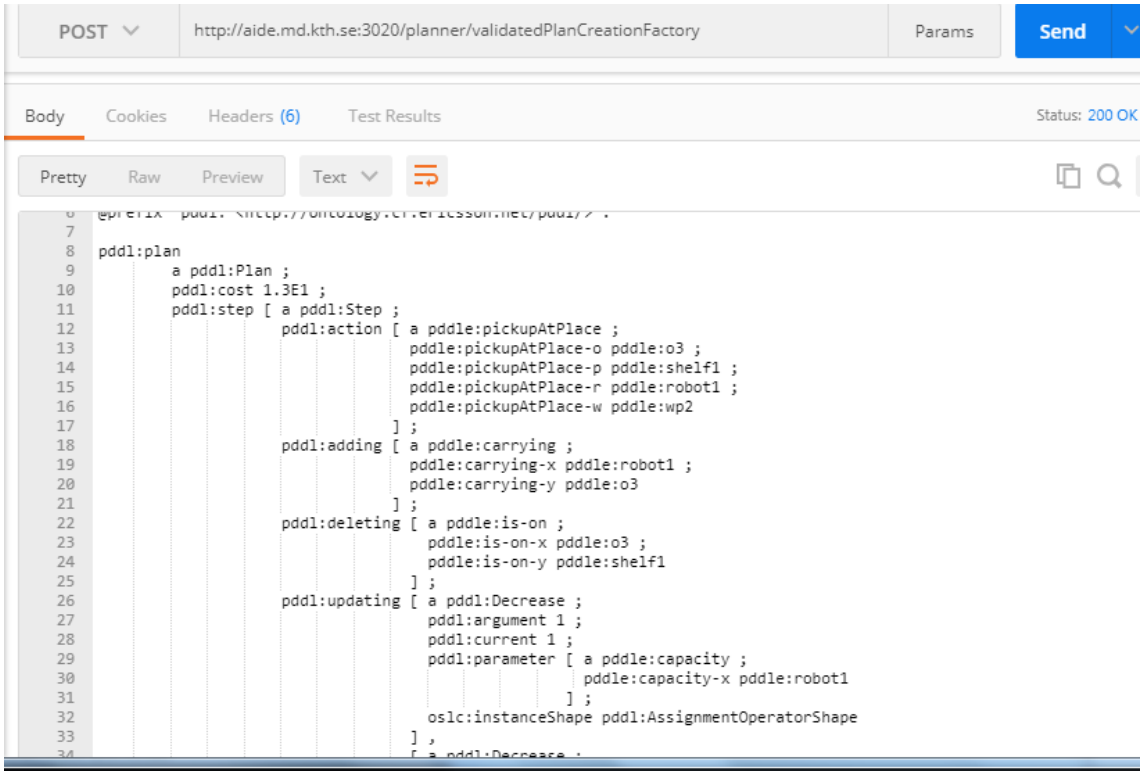


Figure 26. Output annotated plan from /ValidatedPlanCreationFactory.

Textual version of a sample plan is presented below. The Actions in the Plan have the agent as a parameter (See the value "robot1", "robot2" in the action arguments).

moveToWayPoint robot1 w0 w1
moveToWayPoint robot2 w10 w1
pickupAtPlace robot1 obj1 shelf1 w1
moveToWayPoint robot1 w1 w4
pickupAtPlace robot2 obj2 shelf1 w1
moveToWayPoint robot2 w1 w4
dropAtPlace robot2 obj2 cbelt w4
dropAtPlace robot1 obj1 cbelt w4

The global Plan obtained from the Planner service is sent to the Robot twins. The Twins filter the respective Actions based on the agent parameter and send to the corresponding ROS Nodes for execution. For example, Robot twins 1 and 2 have the following filtered Plans derived from the global Plan above.

robot1:

moveToWayPoint w1
pickupAtPlace obj1 shelf1
moveToWayPoint w4
dropAtPlace obj1 cbelt

robot2:

moveToWayPoint w1
pickupAtPlace obj2 shelf1
moveToWayPoint w4
dropAtPlace obj2 cbelt

3.3.3.2 Estimator

The Estimator component takes a plan and an upper-bound on execution time as an input. It outputs two items: (1) whether the plan can indeed be carried out within the specified bound and (2) a safety value considering the paths and navigations of the robots and adopted safety rules e.g. robots should not pass each other within X cm if their relative speed is more than Y m/s. In D10.2, computation of feasibility and estimated completion time is provided.

In the current scope, the Estimator maintains a static model of the warehouse to evaluate the plan cost and returns a pseudorandom value for safety in order to test the response of the WHC. In the following phase(s), the static model will be replaced by the VREP simulator itself, which will simulate the entire plan and determine feasibility and safety value.

To recall, WHC gets the Mission and the execution time bound from the warehouse manager. The URI of the corresponding plan obtained by WHC from the planner service is then passed to the Estimator, along with the execution time bound through a REST API. The Estimator extracts the plan, computes the feasibility and returns the response to WHC as a JSON object (shown below).

{

"id":"xyz",

"feasibility": True,

"reason": 0,

"completion-time":281.6204

}

The response includes the information on whether the plan is feasible within the execution time bound, the estimated completion-time of the plan, a reason if it cannot be completed in time.

3.3.4 Risk Assessment for Safe Operations

The online safety mechanism minimizes the risks while the mobile robot is navigating in the warehouse. Risk management framework performs the central role by ensuring that the robot movements won't cause any injuries to humans.

Three phases of the risk management were addressed: risk identification, risk analysis/evaluation and risk mitigation. The main role of risk mitigation is to determine the robot movements that maximize the safety of the robot itself and the surrounding elements, in special humans. The risk identification is performed manually by verifying all possible risks involved in the interaction between human and robot. The risk analysis and evaluation phase rely on the information of the robot's surrounding environment to make possible to understand the space configuration around the robot. Scene understanding is employed in the risk analysis step. The risk mitigation takes the risk level output (*very high*, *high*, *medium*, *low* or *very low*), calculated in the risk evaluation step and interferes in the robot navigation to maximize the safety [13]. The implementation of both steps was based on fuzzy-logic and reinforcement learning. Details of these steps are described in the next sections.

3.3.4.1 Risk Identification

The first phase of the risk assessment is the hazard and risk identification. We conduct it manually by identifying and then describing all possible existing threats in our human-robot collaboration (HRC) scenario. Additionally, the possible consequences and damages to the human and to other objects are also catalogued.

There are several methods to perform this phase such as, Preliminary Hazard Analysis (PHA), HAZard OPerability analysis (HAZOP), Fault Tree Analysis (FTA), and Failure Modes and Effects Analysis (FMEA). PHA is a simple but inductive method in which hazards for a specific scenario are identified from hazard checklists of a standard (e.g. [14] Annex B: Examples of hazards, hazardous situations and hazardous events). However, robotic standards [15], [16], [17] do not include hazards for HRC scenario. We apply HAZOP method, which is a structured and systematic examination approach to identify hazards and is suitable for our use case. HAZOP first models the scenarios by use case diagrams, sequence diagrams and state-machine diagrams. Then, attributes and guidewords are used to generate deviations. The hazards list can be obtained after merging redundant deviations and removing meaningless deviations. A detailed list of hazards for our HRC use case along with the identification of its type, consequences and effected human body area is presented in Table 3 Description of hazards for collaborative operations

below.

Task	Problem	Hazard			Body Area
		Description	Type	Consequence	
Pickup operation	Product is not properly placed, and robot fails	HN1: Robot cannot pick up the product because either the product is not present on the shelf or is not placed at a proper place	Temporal	Time loss	None
Pickup operation and the worker is placing/replacing the products	Human is very close to the robot.	HN2: Physical human injury as transient contact between gripper and hand. Followed by clamping and dragging along the hand while continuing the planned pick-and-place task	Mechanical	Human injury: Gripping	Back of Workers hand
The robot navigates its arm to pick up a product	Human is very close to the robot.	HN3: Physical human injury. The robot's moving arm can hit the worker's body	Mechanical	Human injury: impact	Upper part of the body
Manipulator drops the product and a worker is nearby	Product is not held properly, and the robot drops the product close to the human who can get hurt	HN4: Physical human injury on worker's foot or leg due to the fallen product	Mechanical	Human injury: crushing	Foot / leg
Robot navigation and a worker/visitor is moving closely	Human is very close to the moving robot.	HN5: Physical human injury on worker's body due to the moving robot	Mechanical	Human injury: impact	Lower part of the body
Place operation	Product cannot be placed properly	HN7: No place for the product because conveyor belt is not moving	Temporal	Time loss	None

Task	Problem	Hazard			Body Area
		Description	Type	Consequence	
Place operation	Product cannot be placed properly	HN8: Robot is not able to place the grip from product properly	Mechanical	Financial loss	None
Change in the Robot's behavior due to new/updated software	Robot does not behave as anticipated by worker	HN9: Physical injury, stress to collaborative worker with unexpected behavior	Communication	Human physical/ mental injury	Any body area
Multiple robots are moving close to each other	Proximity sensor failed or software error	HN10: Damage due to robot collision	Mechanical	Financial loss	None
Pickup and/or place operations	Improper force limitation or force control failure	HN11: Property damage on fragile products due to robot	Mechanical	Financial loss	None
The robot is performing a task	Software error	HN12: Failure to switch modes when a reaction is needed.	Software	Financial loss	None
The robot is performing a task	Software or hardware error	HN13: False emergency stop	Software/hardware	Financial loss	None
The robot is performing a task	Software or hardware error	HN14: Robot shutdown during a task	Software/hardware	Time loss	None
The robot is performing a task	Software error	HN15: False alarm or indicator light	Software	Time loss. Physical/ mental injury	Any body area

Table 3 Description of hazards for collaborative operations

3.3.4.2 Risk Analysis and Evaluation

This phase comprehends in determining the nature of risks, the level of risk, including risk estimation [14]. For that, we identify key entities, attributes of the entities, and the relationships among the attributes and then perform risk estimation. The key entities in our case are the shared

workspace which consists of some static objects (e.g. shelves, products, conveyor belts and dock stations) and some dynamic objects (other robots and human workers). The risk level is essential to determine the optimal movements of the robot that maximize the overall safety, performed during the risk mitigation step. The high dynamicity of the warehouse scenario makes necessary the recalculation of the risk level before each new robot move.

The value of the risk level depends mainly on the relative position of the objects. This information is provided by the perception module of the robot through a scene graph (details in Section 3.3.4.3). Using the scene graph as input, a fuzzy system is used to calculate the risk level from a set an If-Then rules. The following attributes of the scene graph are used as the input of the fuzzy system: obstacle type (i.e. dynamic or static), obstacle distance, obstacle speed, obstacle direction and obstacle orientation. The risk level output of the fuzzy system can take one of the following values: very low, low, medium, high and very high. Table 4 presents a sample of rules modelled for the fuzzy system. In total 883 rules were generated.

Input					Output
Type	**Distance**	**Direction**	**Speed**	**Orientation**	**Risk**
Static	Near	Front	—	—	Very High
Static	Far	Rear	—	—	Very Low
Dynamic	Near	Right	Medium	Rear	Medium
Dynamic	Medium	Front	Slow	Front	High
Dynamic	Medium	Left	Slow	Front	Low
Human	Near	Front	Slow	Front	Very High
Human	Medium	Left	Slow	Left	Slow
Human	Far	Front	Fast	Right	Medium

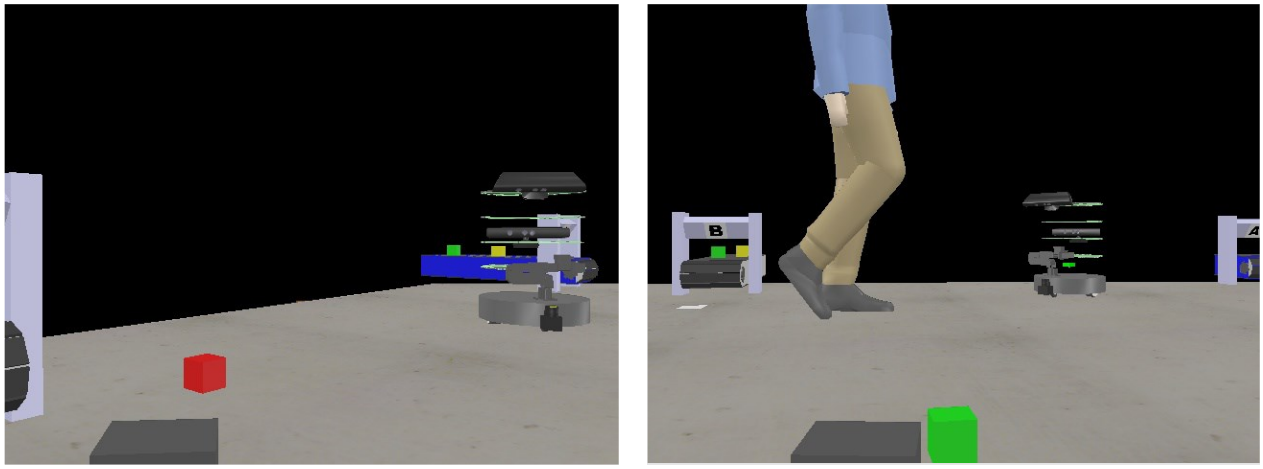
Table 4. Examples of rules to calculate the risk level given the obstacle properties.

3.3.4.3 Scene Understanding

The risk management relies on environment information to determine the risk levels of those objects that surround the robot and to determine the safest movements for the robot. Figure 27 presents robot's camera images in situations where the detection of objects is fundamental to determine the safest robot movements. Scene understanding provides such mechanism by detecting objects from camera images and extracting semantic information from that. The scene graph structure, which is the main output of the scene understanding node, is used to represent the semantic and contextual information of the objects.

Two approaches were investigated to generate scene graph from robot's sensor: (a) combination of Mask R-CNN instance segmentation and intersection-based scene graph construction; and (b) usage of Multi-level Scene Description Neural Network (MSDN) method. In both approaches, data

gathered from robot's camera is used as input (i.e. RGB and depth images). Details of both methods are described in the following subsections.



(a) A robot is occluding the conveyor belt and a part of another conveyor belt was captured on the left side of the image.

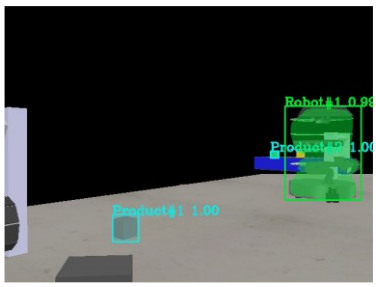
(b) A human is walking near two conveyor belts and a robot. The conveyor belt on the right side of the image was partially captured.

Figure 27. Samples of the robot's camera images used to evaluate the scene understanding methods.

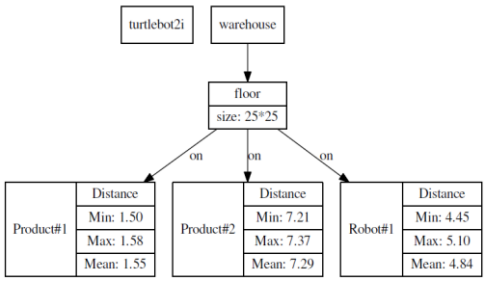
Scene Graph Construction from Mask R-CNN Output

Mask R-CNN is originally used to perform instance segmentation (object detection combined with image segmentation) to determine the objects present in the environment, their size and relative position in the image [18]. Thanks to the different information that Mask R-CNN can provide, it was integrated with the scene graph generation method. In other words, the instance segmentation results are converted into a scene graph representation.

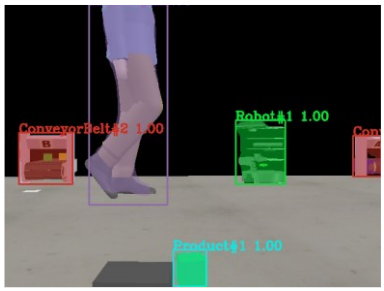
The algorithm to generate the scene graph is based on the paper "*On support relations and semantic scene graphs*" [19]. The main idea behind this method is to connect in the graph those objects that have physical connection. However, instead of calculating superpixel map for each image over generated bounding boxes, we took the proposed segmentation masks computed along with object detection from Mask R-CNN. Moreover, we use prior contextual knowledge for each class object in the scene which helps to eliminate the support relationships that are unlikely to occur. Our algorithm takes leverage from RGB-D camera of the robot and segmentation masks of Mask R-CNN by calculating precise distance of the object. The scene graph generation algorithm becomes relatively easy to compute given the object detection and segmentation output from Mask R-CNN and prior contextual knowledge. In Figure 28 is illustrated the instance segmentation and scene graph generation results of this solution that uses the images from Figure 27 as input.



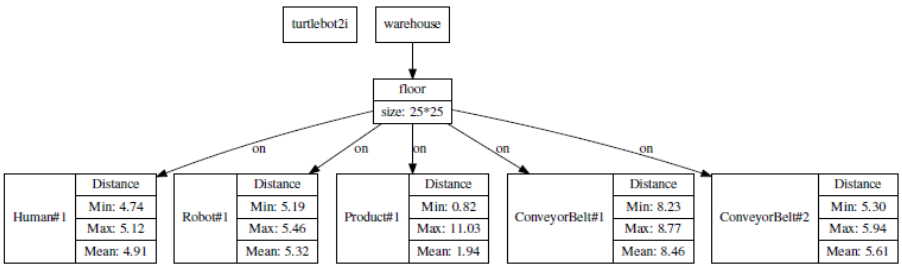
(a) Mask R-CNN output from the first scenario.



(b) Scene graph output from the first scenario.



(c) Mask R-CNN output from the second scenario.

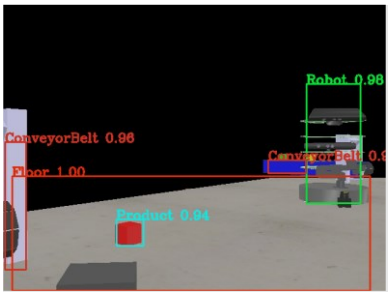


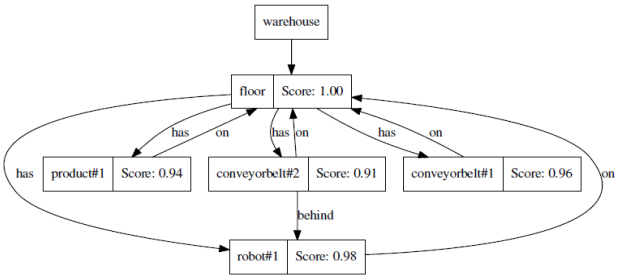
(d) Scene graph output from the second scenario.

Figure 28. Segmentation and scene graph outputs obtained from Mask R-CNN and intersection-based scene graph construction.

Scene Graph Construction from MSDN

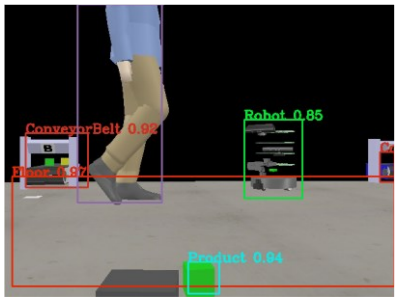
In this approach, the whole scene understanding is performed directly by a single network that combines convolutional neural network (CNN) and long short term memory (LSTM) network. This architecture, named MSDN, has network layers responsible for object detection, scene graph generation, sentence generation, among others. Compared to the previous approach, MSDN has the advantage of having an end-to-end solution for scene understanding and generates sentences (captions) that describes the scene. Differently from the previous approach, the relationships of the graph are not restricted to "*on*" and can have different values such as "*has*", "*behind*" and "*next to*". However, it is not possible to easily configure the way the scene graph is constructed, and the object detection method may have an accuracy lower than Mask R-CNN as it uses Faster R-CNN instead. Figure 29 illustrates the generated object detection and scene graph from MSDN by using images depicted in Figure 27 as input. Regarding to the sentence generation, MSDN outputs a set of phrases with corresponding log probabilities by using the LSTM network. Figure 30 presents the captions produced by MSDN.



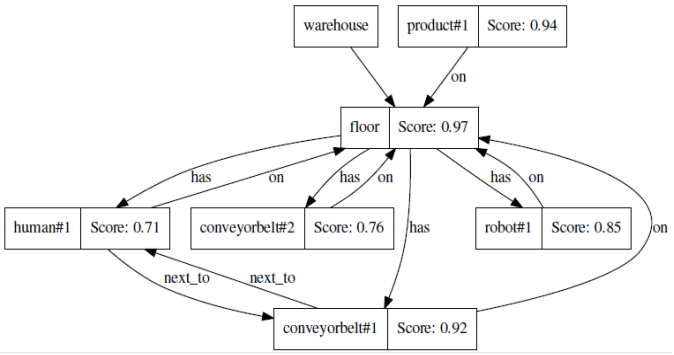


(a) Object detection output from the first scenario.

(b) Scene graph output from the first scenario.

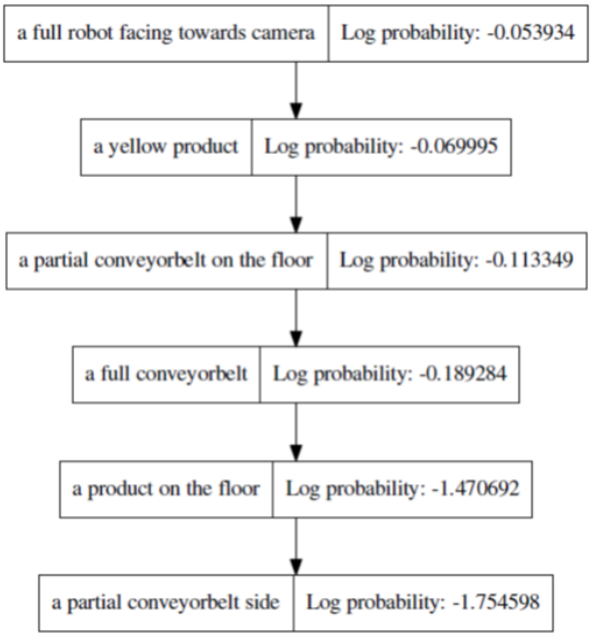


(c) Object detection output from the second scenario.

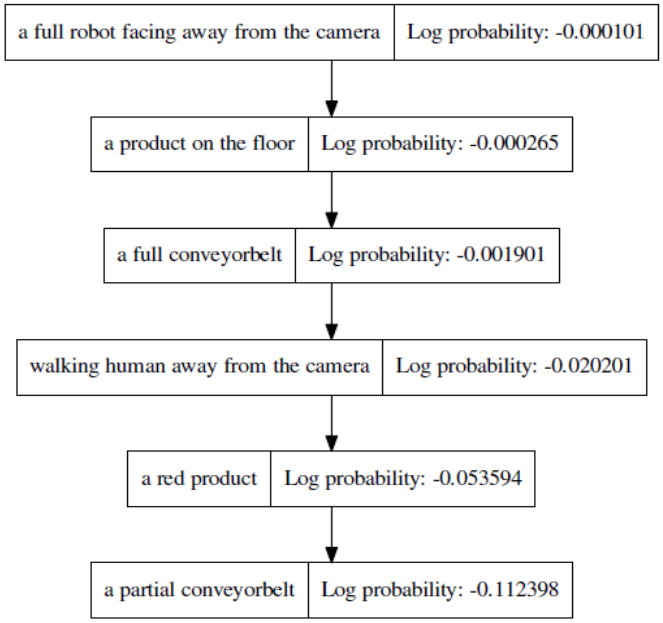


(d) Scene graph output from the second scenario.

Figure 29. Object detection and scene graph outputs obtained from MSDN.



(a) Caption output from the first scenario.



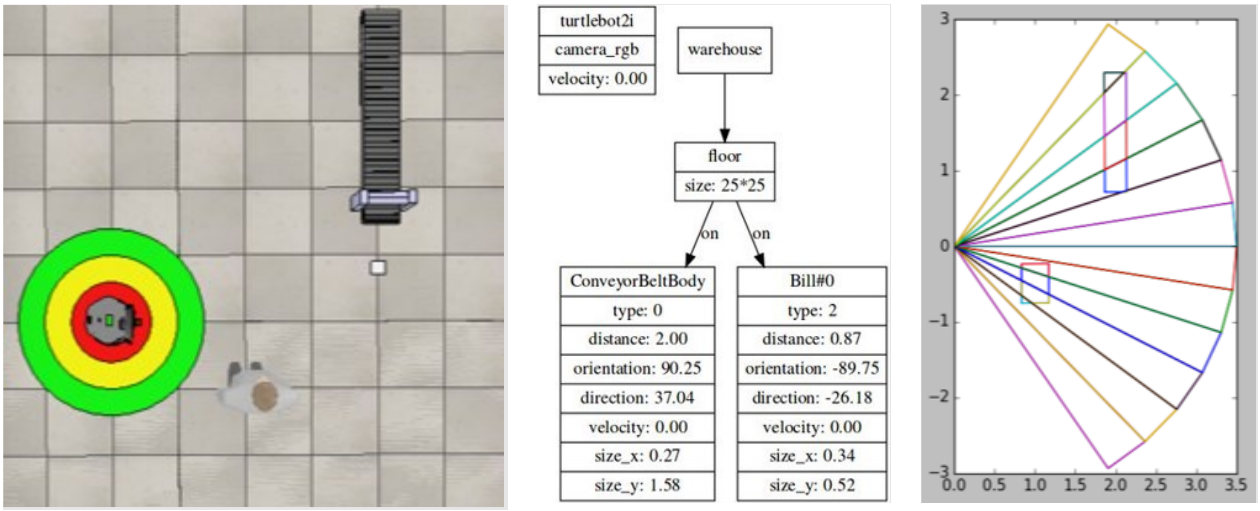
(b) Caption output from the second scenario.

Figure 30. Caption generated from MSDN and the corresponding log probabilities.

3.3.4.4 Scene Graph for Risk Mitigation

The risk mitigation node requires different information from environment, such as object type, size, and distance, which are already available in the scene graph structure. To make possible the usage of scene graph data in the risk mitigation node, the scene graph must be parsed to identify each obstacle detected during the scene understanding process. The geometric information of the scene graph is also virtually converted into a cartesian representation to obtain the relative position of the objects. From that, it is used as input of the risk mitigation method. Figure 31 illustrates the cartesian representation of the scene graph. This representation divides the space into zones (circular sectors). It is highlighted that the scene graph is not discarded at this point and it is used

afterwards to provide other relevant information, such as the object type and the relationship between objects.



(a) Top view of the scenario.

(b) Scene graph generated by the elements detected from robot's camera.

(c) Conversion of generated scene graph into cartesian representation.

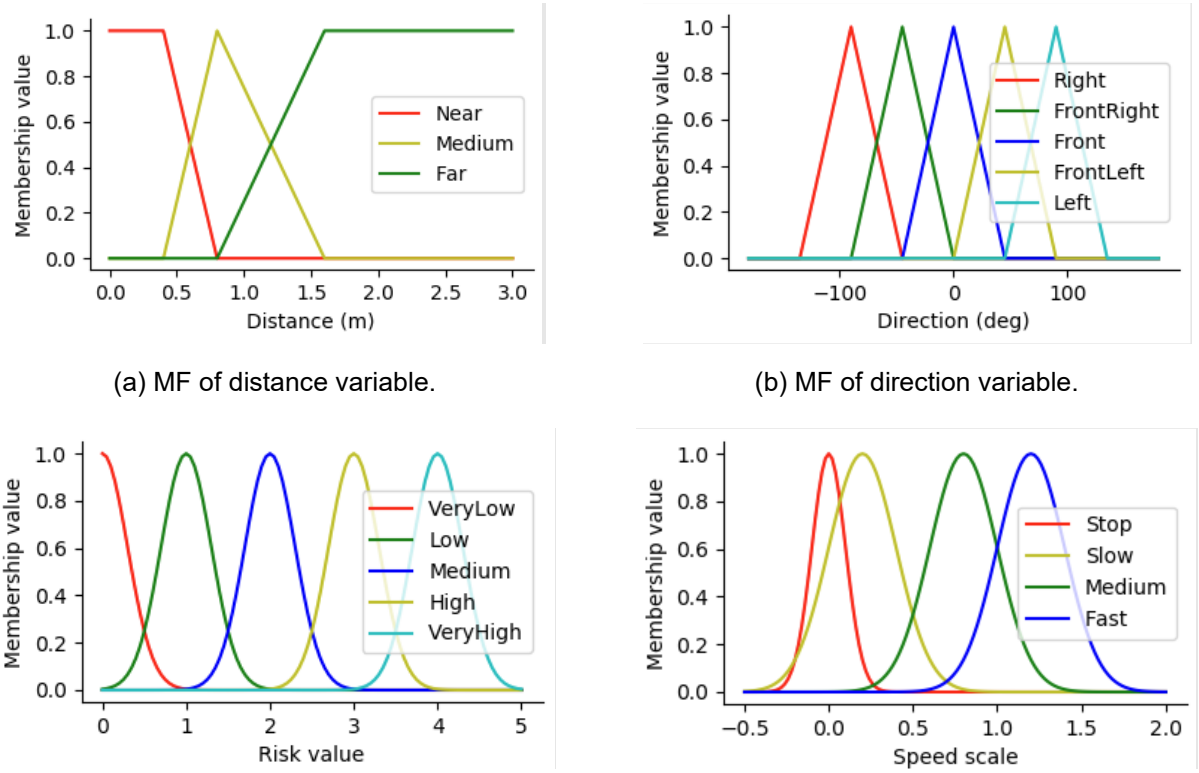
Figure 31. Cartesian representation of scene graph when received by risk mitigation process.

3.3.4.5 Fuzzy Logic System-based Risk Mitigation

In the Fuzzy Logic System (FLS) based risk mitigation, it is considered the scene graph of the closest obstacle to the robot. Attributes of this scene graph are extracted and associated to linguistic variables, which are object risk value, direction and distance. From that, rules are evaluated using the membership functions (MFs) defined for each linguistic variable. The shape of the MFs are presented in Figure 32. The output is the speed scaling of the robot, which can have the following linguistic values: "*stop*", "*slow*", "*medium*" and "*fast*".

An example of a rule is "*If obstacle's distance is near and its direction is on the front then the left wheel scale is stop and the right wheel scale is stop*". When this condition is satisfied (some obstacle is close and is moving towards to the robot) robot wheels should stop. An advantage of the FLS is its interpretability as rules can be written in natural language. All rules[11] were manually created following the safety recommendation of ISO/TS 15066 [17].

[11] The complete list of modelled rules can be found in SCOTT GitHub: https://github.com/EricssonResearch/scott-eu/blob/master/simulationros/src/turtlebot2i/turtlebot2i_safety/src/mitigation_rules.py



(a) MF of distance variable.

(b) MF of direction variable.

(c) MF of risk value variable.

(d) MF of speed scale variable.

Figure 32. MF of different linguistic variables used in the risk mitigation.

3.3.4.6 Reinforcement Learning-based Risk Mitigation

One of the disadvantages of the FLS-based risk mitigation is the necessity of manually defining the rules, which is prone to errors. Differently, the application of Reinforcement Learning (RL) enables learning the robot actions that maximize the reward function. In our case, the reward function is modelled to minimize the chances of collisions between a robot and an object. This is performed through successive interactions of the mobile robot (i.e. agent) with the automated warehouse (i.e. environment).

To determine the appropriate robot movements (i.e. actions), it is necessary to obtain the current state of the robot. This is obtained by parsing the scene graph as described in the Section 3.3.4.5 (Fuzzy Logic System-based Risk Mitigation). The robot state is formed by the following information:

- Obstacle distance in each zone (12 values).
- Distance to the nearest obstacle.
- Direction of the nearest obstacle.
- Linear speed of the robot.
- Angular speed of the robot.
- Maximum value of the risk value.
- Radius of warning zone.
- Radius of safe zone.

The reward function has a direct influence on the robot's behaviour. The robot should take actions that maximizes the overall reward. Consequently, this function was modelled to keep the robot

away from the nearby obstacles and at the same time make it move towards the target waypoint. The reward function is summarized as follows:

- **Yaw reward:**
  - Avoid obstacle: +1
  - Towards obstacle: - % of yaw
- **Clearance reward:** 1/nearest obstacle's distance
- **Reward:**
  - Collision happens: -5000
  - Obstacle inside critical zone: (clearance reward X yaw reward): – 50
  - Obstacle inside warning zone: (clearance reward X yaw reward): - 10
  - Obstacle inside safe zone and robot moved 0.017 m (clearance reward X yaw reward): +1
  - Obstacle outside safe zone and robot moved 0.017m: +1
  - Obstacle outside warning zone and robot moved less than 0.017m: -1

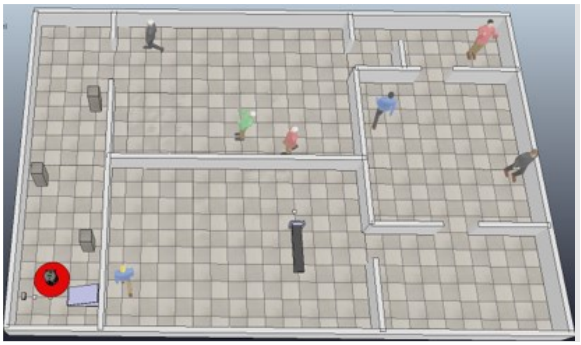
Actions define how the robot will move based on its current state. In our formulation, actions are speed scales applied in the left and right wheels of the turtlebot2i robot. The speed scales can assume one of the following values: [0.0, 0.4, 0.8, 1.2]. Right and left wheels can have different speed scales, which leads to 16 possible combinations of speed scales.

Q-Learning (DQN) was employed to learn which actions the robot should take (i.e. policy) depending on the current state. DQN is based on Q-Learning that builds a Q-Table that defines the actions that the agent should take for a given state in a tabular format [20]. To learn the Q-Table, DQN uses deep neural network artefacts [20].

For risk mitigation, three DQN architectures were developed: (a) fully connected multi-layer perceptron, (b) one-dimensional convolutional neural network and (c) hybrid architecture that combines (a) and (b). In Figure 33 is illustrated the four scenarios used to evaluate the DQN models. In those scenarios that human and obstacles are present, they are placed in a random position when the simulation starts. In every scenario, the robot will follow this path: start – goal – start – goal to evaluate the AI algorithms as seen in Figure 33a.

(a) Warehouse with known obstacles.

(b) Warehouse with unknown static obstacles.



(c) Warehouse with humans.

(d) Warehouse with humans and unknown static obstacles.

Figure 33. Scenarios used to evaluate the RL-based and FLS-based risk mitigation methods.

The experiment results of the risk mitigation, both for FLS and RL approaches are presented in Tables 5 to 9. Each table shows the results obtained for a single scenario. The first column of the table presents the metric, second column presents results for the default operation without risk mitigation (this is calculated to make a comparison with different algorithms). Third, fourth, fifth and sixth columns present results of using FLS, FCN, CNN and Hybrid models respectively.

In this scenario, all operations that implement risk mitigation modules has higher value of the time-percentage for the robot in keeping the obstacle outside the critical and warning zones as compared to the default operation without risk mitigation. The time-percentage of critical zone metric is less in each module. The FCN model has the highest mean risk value (i.e. 0.542) because the robot operates within a close distance to the obstacles (FCN has the lowest value of mean distance to obstacles). However, the FCN model has a lower value of mean of (risk X speed) than the normal operation without risk mitigation. It indicates that the robot lowered down its speed when it encountered a high-risk obstacle.

Parameter	w/o RM	FLS	FCN	CNN	Hybrid
Safe Zone (%)	92.214	92.895	93.498	94.927	92.373
Warning Zone (%)	7.645	7.069	6.431	5.048	6.506
Critical Zone (%)	0.141	0.036	0.071	0.026	1.121
Mean Risk Value	0.469	0.402	0.542	0.434	0.449
Max Risk Value	2.544	2.476	2.803	2.347	3.144
Mean of (risk X speed)	0.210	0.202	0.203	0.146	0.211
Max of (risk X speed)	1.086	1.340	1.093	0.898	1.531
Mean dist. to obstacle (m)	2.525	2.539	2.517	25.311	2.654
Mean min. dist. to obstacle (m)	0.531	0.547	0.516	0.328	0.349

Parameter	w/o RM	FLS	FCN	CNN	Hybrid
Min. dist. to obstacle (m)	0.233	0.234	0.226	0.255	0.200
Collision	0	0	0	0	0

Table 5. The evaluation result of scenario with known static obstacles in safety aspect.

Parameter	w/o RM	FLS	FCN	CNN	Hybrid
Safe Zone (%)	69.615	77.242	73.639	77.869	71.405
Warning Zone (%)	22.858	22.103	22.518	20.801	24.350
Critical Zone (%)	7.527	0.655	3.843	1.330	4.246
Mean Risk Value	1.132	1.022	1.236	1.116	1.113
Max Risk Value	3.071	2.773	2.932	2.988	3.368
Mean of (risk X speed)	0.498	0.484	0.479	0.369	0.548
Max of (risk X speed)	1.603	1.567	1.316	1.165	1.985
Mean dist. to obstacle (m)	2.174	2.141	2.180	2.161	2.166
Mean min. dist. to obstacle (m)	0.307	0.366	0.348	0.314	0.256
Min. dist. to obstacle (m)	0.200	0.231	0.224	0.254	0.200
Collision	2	0	0	0	1

Table 6. The evaluation result of scenario with unknown static obstacles in safety aspect.

Parameter	w/o RM	FLS	FCN	CNN	Hybrid
Safe Zone (%)	69.615	77.242	73.639	77.869	71.405
Warning Zone (%)	22.858	22.103	22.518	20.801	24.350
Critical Zone (%)	7.527	0.655	3.843	1.330	4.246
Mean Risk Value	1.132	1.022	1.236	1.116	1.113
Max Risk Value	3.071	2.773	2.932	2.988	3.368
Mean of (risk X speed)	0.498	0.484	0.479	0.369	0.548
Max of (risk X speed)	1.603	1.567	1.316	1.165	1.985

Parameter	w/o RM	FLS	FCN	CNN	Hybrid
Mean dist. to obstacle (m)	2.174	2.141	2.180	2.161	2.166
Mean min. dist. to obstacle (m)	0.307	0.366	0.348	0.314	0.256
Min. dist. to obstacle (m)	0.200	0.231	0.224	0.254	0.200
Collision	2	0	0	0	1

Table 7. The evaluation result of scenario with humans in safety aspect.

Parameter	w/o RM	FLS	FCN	CNN	Hybrid
Safe Zone (%)	83.212	87.297	85.351	86.088	85.722
Warning Zone (%)	14.999	11.349	14.549	13.108	11.280
Critical Zone (%)	1.787	1.354	0.101	0.804	2.998
Mean Risk Value	0.980	0.816	0.950	0.901	0.893
Max Risk Value	3.267	3.682	3.606	3.680	3.670
Mean of (risk X speed)	0.421	0.330	0.317	0.282	0.352
Max of (risk X speed)	1.687	1.479	1.306	1.272	1.945
Mean dist. to obstacle (m)	2.501	2.449	2.484	2.498	2.583
Mean min. dist. to obstacle (m)	0.437	0.316	0.329	0.230	0.218
Min. dist. to obstacle (m)	0.225	0.200	0.200	0.200	0.200
Collision	0	1	0	2	0

Table 8. The evaluation result of scenario with humans in safety aspect.

Parameter	w/o RM	FLS	FCN	CNN	Hybrid
Safe Zone (%)	63.672	66.288	67.801	72.835	65.753
Warning Zone (%)	27.224	28.802	28.145	24.784	25.252
Critical Zone (%)	9.104	4.911	4.055	2.381	8.995
Mean Risk Value	1.327	1.380	1.535	1.395	1.260
Max Risk Value	3.670	3.666	3.669	3.653	3.658

Parameter	w/o RM	FLS	FCN	CNN	Hybrid
Mean of (risk X speed)	0.569	0.566	0.462	0.438	0.526
Max of (risk X speed)	3.549	1.800	1.219	1.356	2.006
Mean dist. to obstacle (m)	2.139	2.100	2.145	2.187	2.132
Mean min. dist. to obstacle (m)	0.203	0.286	0.212	0.288	0.214
Min. dist. to obstacle (m)	0.200	0.200	0.200	0.200	0.200
Collision	5	4	0	0	1

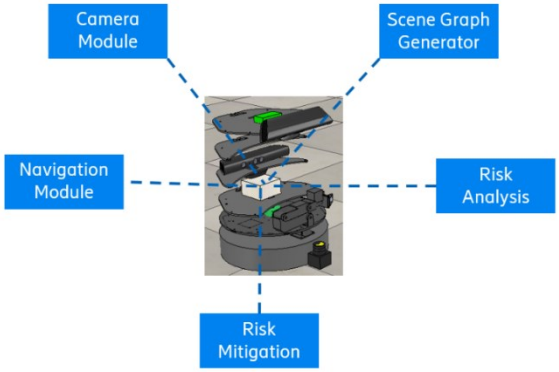
Table 9. The evaluation result of scenario with humans and unknown static obstacles in safety aspect.

3.3.4.7 Distributed Solution of Risk Management Nodes

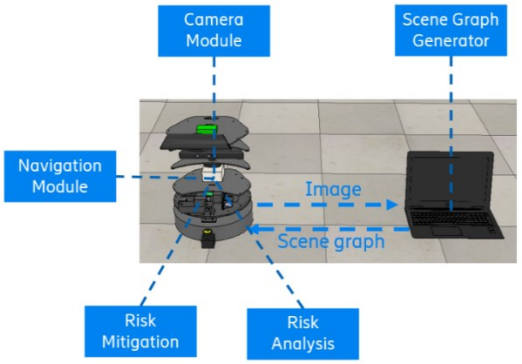
Running experiments in the simulated environment is more convenient as it helps making the development and evaluation of the method faster and safer. However, this setup requires running all ROS nodes in a computer with high specifications.

After testing the methods in the simulated environment, it is now possible to run on the robots. However, the limited computing capacity of the robot's computer makes difficult to fulfil real-time requirements. (Here, real-time means that the response time must be enough to achieve a safe state). This makes necessary the investigation of solutions to distribute some of the robot's processes over the network. Two computation configurations were tested: centralized computing and edge computing.

In case of centralized (or local) computing, all nodes run through the robot's processor. On the contrary, in the distributed computing, the scene understanding node runs on the edge and the other nodes run in the robot's processor. The choice of running scene understanding in a different machine was due to its high processing cost. Scene understanding also relies on computer vision methods that run faster in GPU devices, which are not present in the turtlebot2i robots.



(a) Centralized computation setup. All modules are computed using the robot's machine.



(b) Distributed computation setup. An external computer is used as edge computing device.

Figure 34. Local and distributed computation setups.

We observed that the process to generate a scene graph information required around 15 seconds in a centralized computation setup. This performance is not suitable for risk mitigation because a collision may have already happened during the process of generating the scene graph information. However, it only took 0.3 s in average to generate the scene graph on the edge device. In this architecture, the duration of risk analysis module was reduced by 45%. The risk mitigation with RL decreased the processing time by 36% in average while the FLS module has 14% less running time than the centralized approach. Figure 35 shows the complete comparison between local and edge computation architecture.

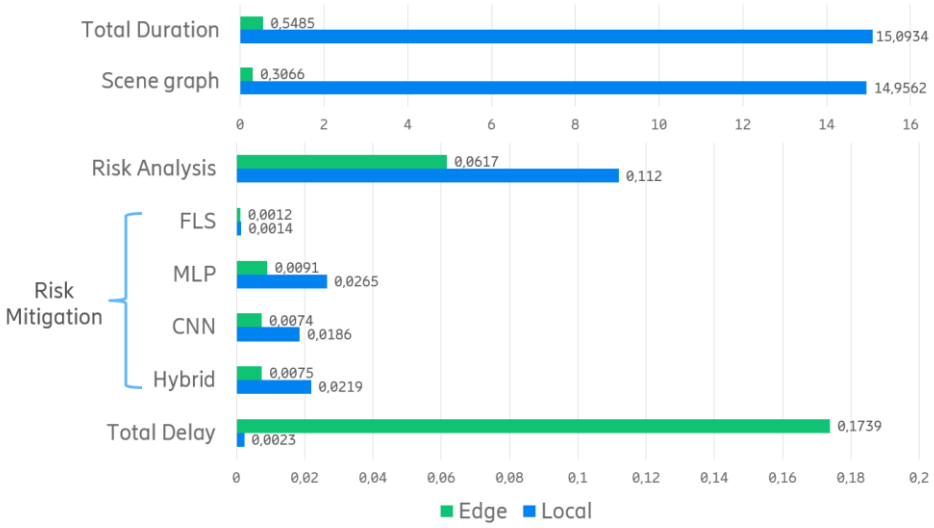


Figure 35. The duration of local and edge computational time in seconds.

3.3.5 Goal State Generation out of Natural Language Mission Specifications

In order to enable the warehouse manager to understand natural language, an NLP component is needed. The state-of-art learning-based NLP engine requires huge amount of conversation pairs to train the recurrent model to associate questions and respective answers. Since, in our case, the prior conversation pair dataset is absent, we built a semantic model to fit most-common plain English that may occur in warehouse scenario. Based on Attempto Controlled English (ACE), the corpora include query for the state of robot, description of the status of object, commands to the robot to perform an action and statements of the command result.

The model could extract the keywords in natural-language-based mission and interpret into separate sub-goals through semantic rules. As shown in Figure 36, first, the sentence is classified according to syntax as: declarative, imperative, general question (asking a statement is true or false) or a wh-problem (what, where or when). Once a sentence is classified, more minor component of the sentence is processed at the level of word. The NLP component extracts the part-of-speech (PoS), usage of words, grammar and tense to find the core essence in the sentence, which describes the goal. The essence of sentence is presented as First Order Logic (FOL), which includes the action towards a goal, properties of the object, or the executor of the action. As a result, goals are parsed into machine-readable RDF triples for further processing.

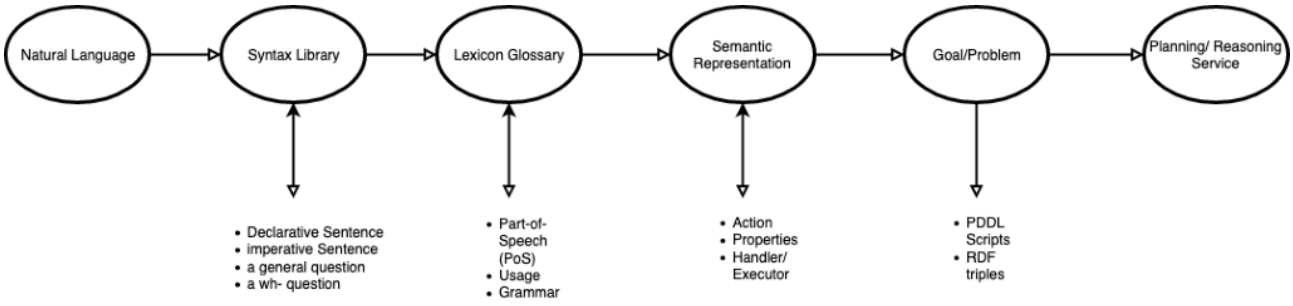


Figure 36. NLP component interoperate from natural language to RDF triples.

Some of the example statement and processing result is given:

1. a box is on a shelf. (State of object)
    - FOL: some(A,and(box(A),some(B,and(shelf(B),location(on(A,B),B)))))
2. a robot carries a box. (Command result)
    - FOL: some(A,and(robot(A),some(B,and(box(B),carry(A,B)))))
3. the box is red. (Properties of object)
    - FOL: some(A,and(box(A),color(red(A),A)))
4. carry a box to the shelf. (Command)
    - FOL: some(A,and(shelf(A),location(to(some(B,and(box(B),carry(C,B))),A),A)))
5. A box is the thing in the shelf (Properties and state of object)
    - FOL: some(A,and(shelf(A),location(on(some(B,and(box(B),some(C,and(box(C),eq(B,C))))),A),A)))

In the previous example, we regard "on the shelf" as a property "location" of the object "box", so the FOL interpretation of the sentence is equivalent as the phase "box on the shelf", where "on" is performing adjective preposition. Since we attempt to formulate language in semantic perspective, our solution is less costly than conventional corpora-based solution and more accurate than performing unsupervised word embedding on raw-text.

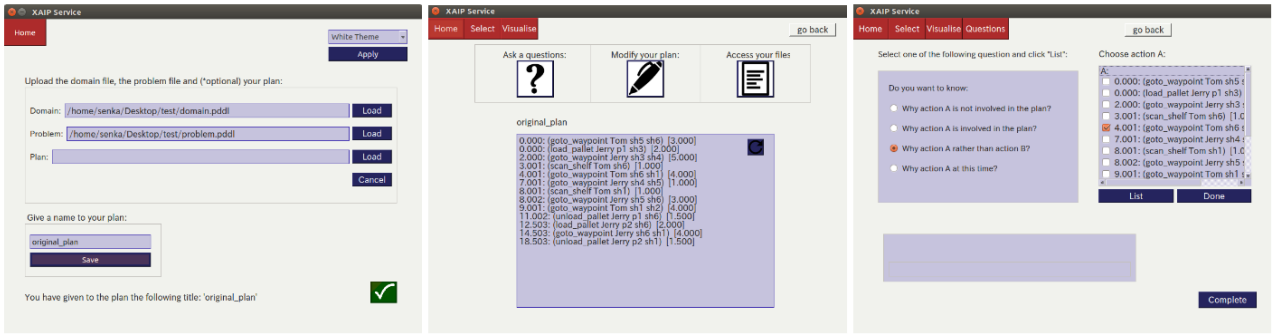
3.3.6 Explanations for Synthesized Plans

Strategic plan synthesis involves complex state space search algorithms. In order to build trust in such complex algorithms, there should be provision in the system to provide reasonable explanations for the output of plan synthesis. If a plan cannot be synthesized, the system should be able to explain the key reason that prevented plan synthesis. When a plan is generated, the system should be able to explain contrastive questions such as why a particular action was selected at a point where there were other choices, what points to the optimality etc. We are developing solutions based on techniques used in [21] and [22]. The explanations help the warehouse manager to trust the planner service, in the scenarios when the service returns with the result that a plan cannot be synthesized for a given mission, and also when a seemingly complex plan has been proposed.

One approach is to provide explanations through answers to counterfactuals. For example, when a plan is derived by the planner service, a user tries to understand the plan by posing “what…if…” questions which are one of the following:

1. *Why is action A used in the plan, rather than not being used? This is answered by disallowing the action in the plan, trying to find an alternative plan if possible and showing the cost of the alternate plan.*
2. *Why is action A NOT used in the plan? This is answered by forcing the action to be used somewhere in the plan. One way of doing this is to have a predicate which marks the completion of the action and specifying the predicate as a goal.*
3. *Why is action A used rather than action B? One provides an explanation by producing, if possible, plans that include action B and exclude action A, and then comparing the new plans with the given plan.*
4. *Why is action A used before/after action B, rather than after/before? This is answered by imposing constraints that force B before/after A and comparing the resulting plans with the given plan.*
5. *Why is action A used outside of time window W, rather than being used only within W? This is explained by imposing constraints so that action A is used only within the time window W.*

To summarize, for all these questions the explanations are provided through plans satisfying the user's query and proving that either there are no contrastive plans exist or they are costlier than the one produced by the planner. In [22], these explanations are facilitated through a tool that provides XAIP (explainable AI planning) as a service (See Figure 37). Given a domain, problem and a plan (which could be generated or provided externally by the user), the user can select explanation questions as the above. Then the system produced explanations through alternate plans, when such synthesis is possible and a comparison tuple (*existing, removed, added, diffcost*) which describes the steps that continue in the new plan, the steps that were removed, new steps added and the difference in cost.



(a) Loading model (b) Plan visualisation (c) Question selection

Figure 37. XAIP tool user interface.

The application of the tool in the warehouse logistics use case is described in the following. Since the tool accepts only durative plans, the warehouse domain was modified using dummy durations of 1 second for the move, pickup and drop actions. The prototype tool needs improvements to handle features like negative preconditions, therefore the domain was further simplified by dropping the charging related actions and predicates. In the following the illustrations are based on the goal shown below.

```
; deliver object 1 and 2 to conveyor belt 2. // note the reference to specific objects
   (:goal
      (and
        (is-on o1 cbOut2)
        (is-on o2 cbOut2)

      )
   )
```

The durative plan generated from the domain and this goal is as below.

Durative Plan:

1. 0.000: (movetowaypoint robot1 wp2 wp3) [1.000]
2. 0.000: (movetowaypoint robot2 wp3 wp1) [1.000]
3. 0.001: (pickupatplace robot1 o1 shelf1 wp2) [1.000]
4. 1.001: (movetowaypoint robot2 wp1 wp2) [1.000]
5. 1.002: (dropatplace robot1 o1 cbout2 wp3) [1.000]
6. 2.002: (pickupatplace robot2 o2 shelf1 wp2) [1.000]
7. 2.003: (movetowaypoint robot2 wp2 wp3) [1.000]
8. 3.004: (dropatplace robot2 o2 cbout2 wp3) [1.000]

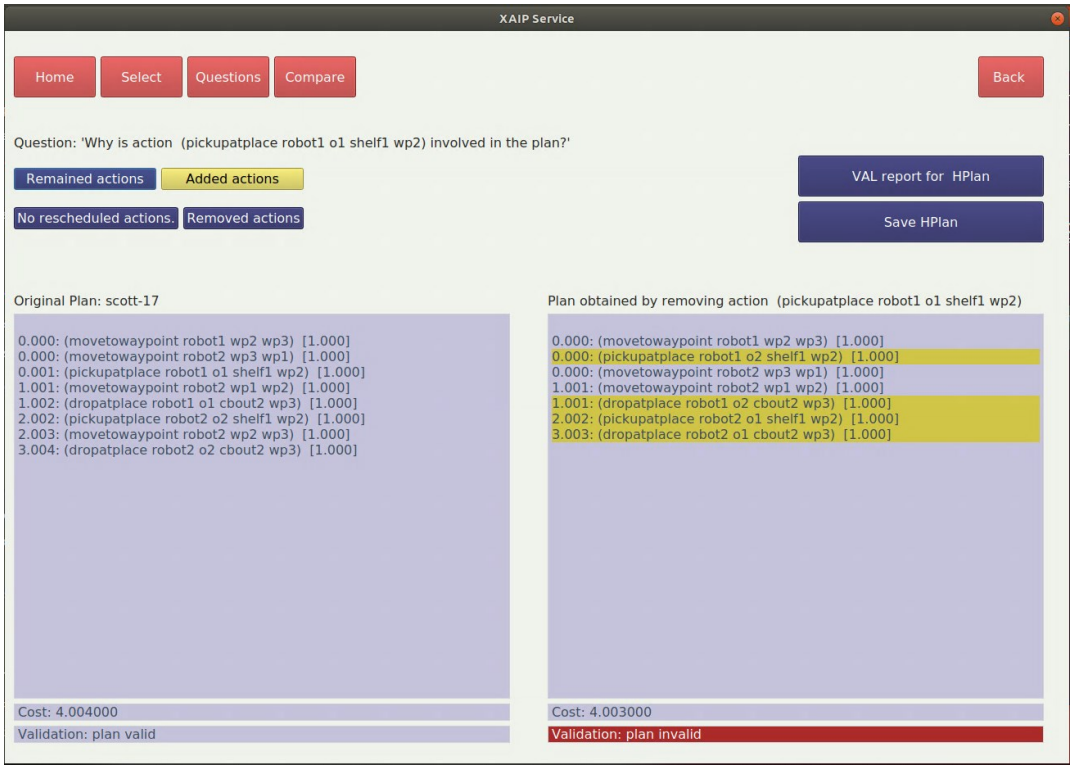
Explanations for the following questions are answered through the XAIP-as-a-service tool.

1. *Why was robot1 used to pickup object 1 in the plan?*

   Template : “Why action A is involved in the plan?”

   3 : (pickupatplace **robot1** o1 shelf1 wp2) □ Action A

The tool removes the action A and tries to find a plan without it. It is found that the action is indispensable to find the goal (Figure 38).



Plan validation fails.

Figure 38. Explanation generated for why was robot1 used to pickup object 1 in the plan.

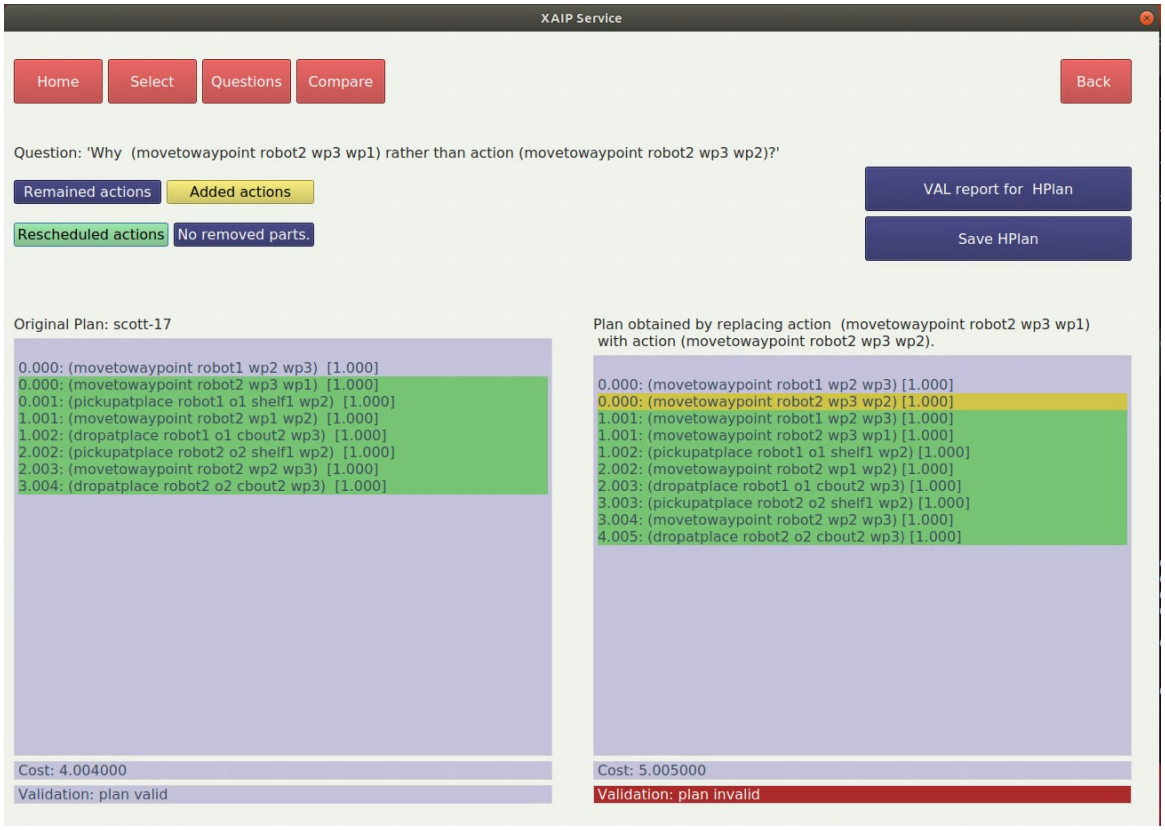
2. *Why did robot visit waypoint WP1 rather than waypoint WP2? Why not shortcut Steps 4 and 5.*

Template : “Why action A is rather used than action B?”

4: (movetowaypoint robot2 wp3 wp1)
5: (movetowaypoint robot2 wp1 wp2)

(movetowaypoint robot2 wp3 **wp1**) □ Action A
(movetowaypoint robot2 wp3 **wp2**) □ Action B

The tool replaces the action (moveToWayPoint robot2 wp3 wp1) with action (moveToWayPoint robot2 wp3 wp2) in the original plan and tries to find if there is a valid plan possible. In this case, it is found that an alternative plan is not possible, which gives the explanation that the shortcut action is really not possible (Figure 39).



Plan validation fails.

Figure 39. Explanation generated for why did robot visit waypoint WP1 rather than waypoint WP2.

Another question for which explanation can be provided using the methodology proposed in the XAIP-as-a-service is the following. The instrumentations necessary are yet to be implemented in the tool though.

3. What if I used only Robot1 (or robot2) to achieve the goal?

One way is to re-model by having a predicate e.g. (is-available robot1) for each robot and control the usage of robots through these predicates by modifying the preconditions of actions to add (is-available ?robot) predicates. Therefore, if we specify (not(is-available robot1)) in the initial state of the problem file, the robot will be barred from usage in any action.

Original plan:

; States evaluated: 84
; Cost: 4.004
; Time 0.02
0.000: (movetowaypoint robot1 wp2 wp3) [1.000]
0.000: (movetowaypoint robot2 wp3 wp1) [1.000]
0.001: (pickupatplace robot1 o1 shelf1 wp2) [1.000]
1.001: (movetowaypoint robot2 wp1 wp2) [1.000]
1.002: (dropatplace robot1 o1 cbout2 wp3) [1.000]
2.002: (pickupatplace robot2 o2 shelf1 wp2) [1.000]
2.003: (movetowaypoint robot2 wp2 wp3) [1.000]
3.004: (dropatplace robot2 o2 cbout2 wp3) [1.000]

Plan obtained by disallowing Robot1 from being used:

; States evaluated: 187
; Cost: 6.005
; Time 0.06
0.000: (movetowaypoint robot2 wp3 wp1) [1.000]
1.001: (movetowaypoint robot2 wp1 wp2) [1.000]
1.002: (movetowaypoint robot2 wp1 wp3) [1.000]
2.002: (pickupatplace robot2 o1 shelf1 wp2) [1.000]
3.003: (dropatplace robot2 o1 cbout2 wp3) [1.000]
4.004: (pickupatplace robot2 o2 shelf1 wp2) [1.000]
5.005: (dropatplace robot2 o2 cbout2 wp3) [1.000]

Original plan has better cost.

Thus, the explanation of why robot1 is used is that it leads to a plan with better cost.

3.3.7 Formal Verification of Strategic Plans

In the warehouse with multiple robots, temporal planning is required to take advantage of the concurrency in the system. Temporal planners have been proposed with “durative actions” included for concurrent plan generation. However, the durations are either mean values or within fixed intervals, which can lead to deviations in planned timelines during execution in real-life. These deviations can lead to violation of safety properties. Therefore, the generated plans for the robots must be verified before execution. We suggested to use timed automata-based formal verification in D10.5.

We translated the generated timed plans to timed-game models in UPPAAL-TIGA [23]. The models have a parameter that can be set for allowable jitter in the durations of the actions of the robots. With specific values for the jitter parameter, we can then verify the possibility of deadlocks in the system during execution. An additional benefit of this approach is that UPPAAL-TIGA can provide a winning-strategy (if it exists) which can be seen as controlled plan execution that can avoid the deadlocks. A plan interpreter can use the winning strategy which will ensure that the robots can avoid the deadlock situation with minor adjustments in the speed of their operations.

The overall sub-architecture of this solution is given in Figure 40. This is sub-architecture extends the plan interpretation and estimation (safety analysis and risk assessment) blocks in Figure 3. Prior to dispatching the temporal plan to the robotic nodes, the plan is verified to guarantee safety properties in the dispatched actions. The deployment controller may also choose to add controlled delays between task steps, to ensure the deployed actions do not violate the required properties.

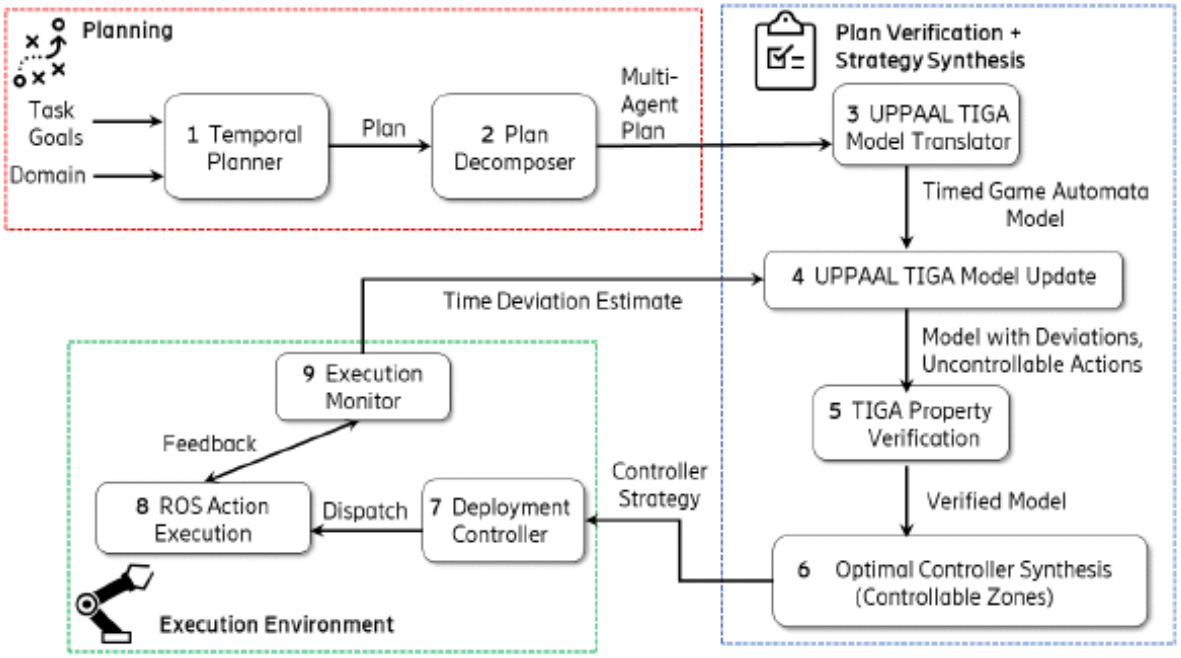


Figure 40. Temporal plan verification and strategy synthesis.

Figure 40 comprises of the following main modules, which overlaps with the warehouse sandbox architecture:

- **Planning Module:** The temporal planner (1) takes the inputs from the domain request and the problem goal request to generate an (optimal) timed plan. Additional goal details may be incorporated in the planning environment (such as minimizing robot battery usage). This output plan is then decomposed by the plan decomposer (2) to multi-agent plans, that may be run on individual robots. Note that this entails concurrent actions with timing constraints incorporated
- **Plan Verification and Strategy Synthesis Module:** The generated multi-agent plans are input to the UPPAAL TIGA model translator (3), that generates timed game automata models with states, transitions and timing guards correctly mapped to the individual plans. The automata models are then appended with realistic timing deviations (jitter) in the UPPAAL TIGA model update (4). The updated model is then verified with property verification (5), with alternate strategies proposed in case the properties are not satisfied. The specific controller strategy chosen to be deployed is then synthesized (6) to a format specified for deployment. This model also contains information about the zones that may be controlled, with additional delays or constraints incorporated.
- **Deployment Execution Module:** The generated plans and control strategies are sent to the dispatch controller (7). The ROS nodes and action lib modules perform the execution (8) on the robots in the warehouse. The execution monitor (9) logs the execution output – in

case the recorded temporal deviations exceed the planned strategies, a rerun of steps (4) to (8) may be required.

Justification for the approach

The core planner service in the solution architecture is based on PDDL-based planning, with FF, LPG and OPTIC planning tools to synthesize strategic plans. The synthesized plan is verified through timed-game automata models. One question is: why not model the planning domain in UPPAAL-TIGA directly and derive a winning strategy that can be used to direct the plan interpreter?

The reasons are:

1. UPPAAL modelling of the state space including very large number of ground actions will not be efficient. On the other hand, since the generated plans have ground actions and simple structure, verifying their correctness and strategy synthesis is going to be much easier.
2. Strategic planners like FF and OPTIC are mature and scale very well to large problem instances. Therefore, we take advantage of the strengths of both the planners and the verifiers.

Details of the solution via multi-robot deadlock avoidance example

We consider the following multi-robot coordination example (Figure 41), that might typically be seen in warehouse logistics. Robot 1 (initial position A1) and Robot 2 (initial position B1) must pick objects from their initial positions and drop them off midway. Robot 1 and Robot 2 then make their way to final goal positions of E1 and E2, respectively. The dropped objects are collected by Robot 3 (initial position A5) and Robot 4 (initial position C4) to drop them off at goal locations of Object 1 (goal position E5) and Object 2 (goal position E6). The values provided represent the time units to complete movement tasks (this can be expanded with distances, robot speed). No two robots may be at the same grid position simultaneously. This represents a task typically observed in warehouse logistics, wherein human and robotic participants coordinate to complete the task.

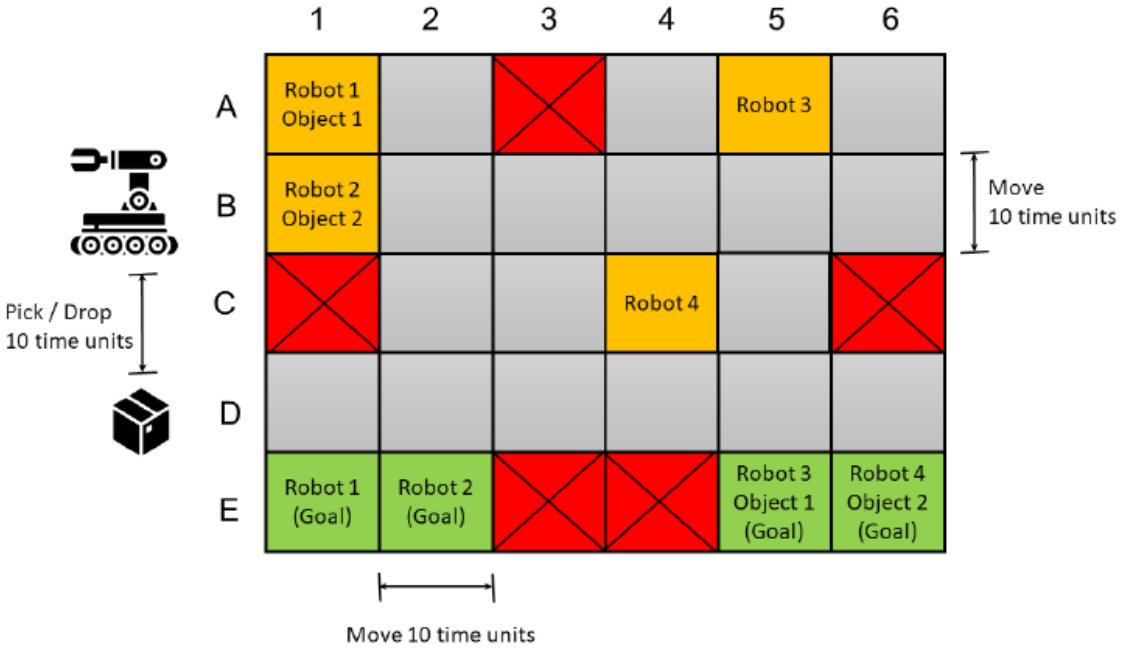


Figure 41. Example use case of multiple robots leading to deadlock due to uncertainty during execution.

Given the above example, the PDDL 2.1 problem file would include the following goals and objective metric:

((**:init** (at_location robot1 A1) (at_location robot2 B1) (at_location robot3 A5) (at_location robot4 C4) (at_location_ob object1 A1) (at_location_ob object2 B1))

(**:goal** (and (at_location_ob object1 E5) (at_location_ob object2 E6) (at_location robot3 E5) (at_location robot4 E6) (at_location robot1 E1) (at_location robot2 E2)))

(**:metric** minimize ((* 1.0 (total-time)))

An example output using temporal planners such as LPG or OPTIC is provided in Figure 42below. Note that parallel movement of all robots are allowed, with durations of atomic move, pick and drop actions set at 10 time units. The plan is decomposed to generate individual temporal plans for Robot 2 and Robot 3, that run concurrently.

```
Plan for ROBOT2:
 0.0000: (PICK ROBOT2 OBJECT2 B1) [D:10.00; C:0.10]
 110.0000: (MOVE ROBOT2 B1 B2) [D:10.00; C:0.10]
 120.0000: (MOVE ROBOT2 B2 B3) [D:10.00; C:0.10]
 130.0000: (MOVE ROBOT2 B3 B4) [D:10.00; C:0.10]
 140.0000: (DROP ROBOT2 OBJECT2 B4) [D:10.00; C:0.10]
 150.0000: (MOVE ROBOT2 B4 C4) [D:10.00; C:0.10]
 160.0000: (MOVE ROBOT2 C4 C3) [D:10.00; C:0.10]
 170.0000: (MOVE ROBOT2 C3 D3) [D:10.00; C:0.10]
 180.0000: (MOVE ROBOT2 D3 D2) [D:10.00; C:0.10]
 190.0000: (MOVE ROBOT2 D2 E2) [D:10.00; C:0.10]

Plan for ROBOT3:
 0.0000: (MOVE ROBOT3 A5 B5) [D:10.00; C:0.10]
 160.0000: (MOVE ROBOT3 B5 B4) [D:10.00; C:0.10]
 170.0000: (PICK ROBOT3 OBJECT2 B4) [D:10.00; C:0.10]
 180.0000: (MOVE ROBOT3 B4 B5) [D:10.00; C:0.10]
 190.0000: (MOVE ROBOT3 B5 C5) [D:10.00; C:0.10]
 200.0000: (MOVE ROBOT3 C5 D5) [D:10.00; C:0.10]
 210.0000: (MOVE ROBOT3 D5 D6) [D:10.00; C:0.10]
 220.0000: (MOVE ROBOT3 D6 E6) [D:10.00; C:0.10]
 230.0000: (DROP ROBOT3 OBJECT2 E6) [D:10.00; C:0.10]
 240.0000: (MOVE ROBOT3 E6 E5) [D:10.00; C:0.10]


METRIC_VALUE = 260.00
Solution number: 3
Total time:      9.20
Search time:     9.19
Actions:         61
Duration:       260.000
Total Num Flips: 5166
```

Figure 42. Global plan and local plans for individual robots.

Deadlock Occurrence Example

Once the individual agents' plans are generated, the warehouse controller typically dispatches them assuming the temporal specifications guarantee avoidance of unsafe situations. However, there are repeated instances where the executions deviating slightly from the intended plan timeline will lead to unintended outputs. An example is given in Figure 43where a **±10 time unit**

deviation in each action results in multiple deadlocks in the multi-robot example. We take an example of the interaction between Robot 2 and Robot 3. Robot 2 is scheduled to reach B4 at 140 time units but gets delayed. At 150, both Robot 2 and Robot 3 attempt to get to B4, that results in a deadlock / unsafe condition. The entire plan has to be aborted in this case that can cause significant delays and hazards. While traditional systems make use of onboard sensors to generate interrupts, this system has to adhere to global temporal plans without reaching unsafe states.

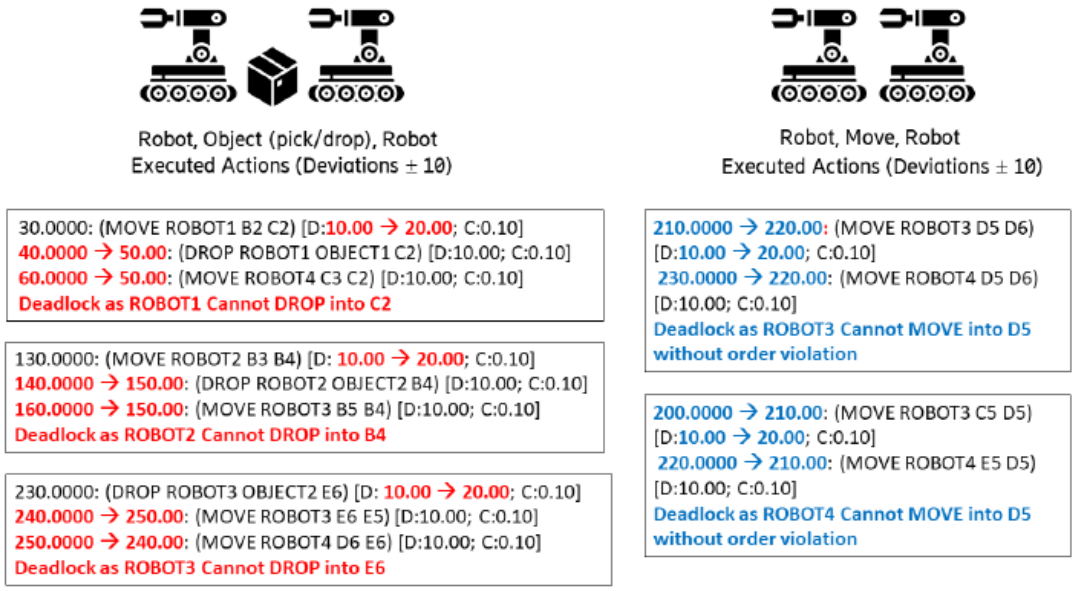


Figure 43. Deadlocks due to Temporal Plan Deviations.

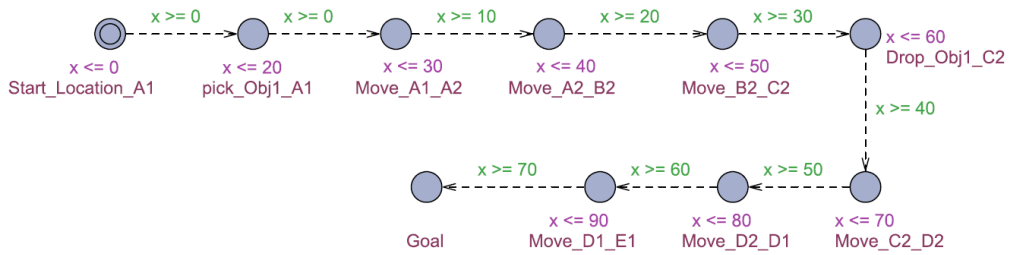
Timed Game Automata Model Transformation

In order to study the individual robotic plans, we make use of the timed game automata models presented below. To ensure one-to-one mapping between plans and transitions, temporal plan timing constraints and plan actions are provided with exact specifications. Given an individual robot plan with a set of sequential temporal tasks T_1 (D_1) … T_k (D_k), where T_i refers to the action label and D_i the corresponding action duration, the following transformation rules are used to transform individual robot plans to timed automata representation:

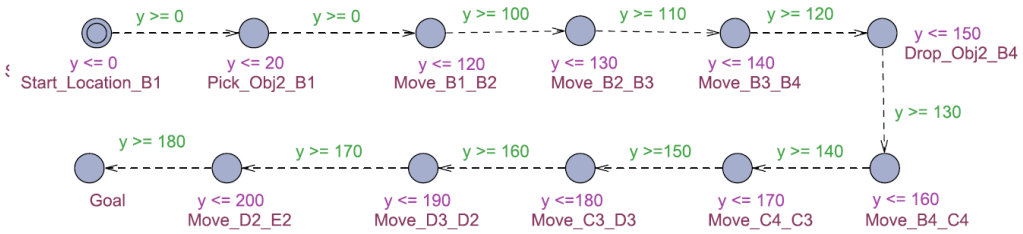
- Each robot is provided an individual clock c (initialized to 0), an initial urgent location S_0 and a final goal location S_g.
- Each planned task step T_1…, T_k is reduced to an intermediary location state between the start and goal state. These states are labelled S_1…, S_k appropriate to the task action labels.
- The sequence of temporal planned steps map to edge transitions E_1…E_{k+1} leading from the initial to the goal states. Formally, the edge E_i is a tuple (S_{i-1}, c_g, S_i), with the transition from location S_{i-1} to S_i provided with the guard c_g for $i\epsilon(1 \dots k)$.
- Clock constraints are added based on the durations of the task D_1…D_k. Each state Si contains and invariant $c_g \leq D_i$ while each transition out of the state (E_i) has the guard $c_g \geq D_i$ for $i\epsilon(1 \dots k)$. The goal location has no output transition or clock constraints. These clock constraints ensure that the transitions are enabled at the exact times specified in the temporal plan.

Note that no channels are used for coordination between synchronous actions – the timing constraints included in clocks x and y are to be followed to ensure temporal ordering of actions.

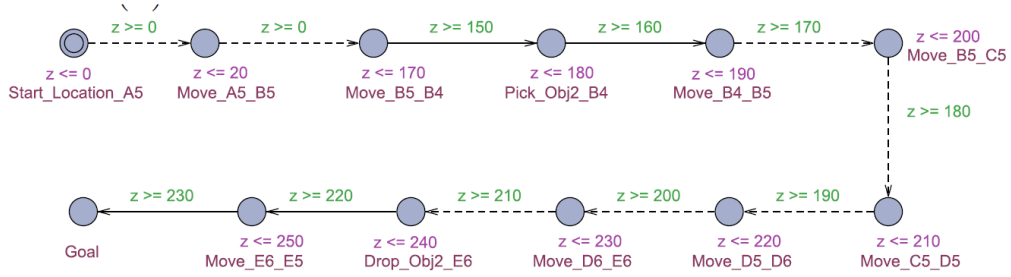
In order to study the individual robotic plans, we make use of the models presented in Figure 44. Note that we make use of controllable actions and uncontrollable actions. The controller has president in **controllable actions** as against the environment having precedence in **uncontrollable actions**. The controller continuously observes the system (moves and delays). The controller can then take the following actions: (i) wait (delay actions) (ii) controllable move (iii) preventing delay with a move. In our example in Figure 44 Robot 1 and Robot 2 are chosen to have only uncontrollable transitions; Robot 3 and Robot 4 have a few controllable transitions. The objective is to still satisfy safety properties despite deviations in the uncontrollable transitions in certain areas of deployments.



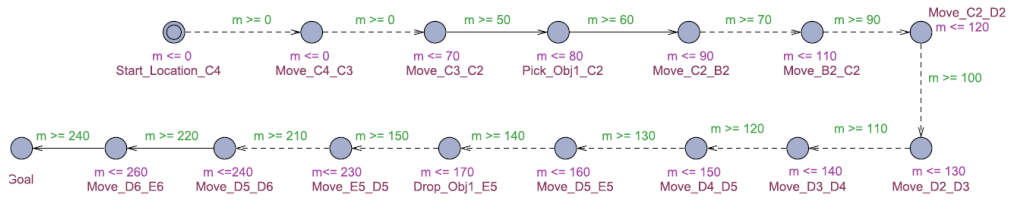
(a) Robot 1 with Uncontrollable Transitions.



(b) Robot 2 with Uncontrollable Transitions.



(c) Robot 3 with Partially Controllable Transitions.



(d) Robot 4 with Partially Controllable Transitions.

Figure 44. Timed Game Automata for Multi-Robot Coordination with Temporal Drifts.

Verification and Controller Synthesis

The model checker associated with UPPAAL TIGA can check for traditionally reachability, safety and liveness properties. In addition, UPPAAL TIGA specifies generating a control strategy that verifies the following properties:

- **Pure Reachability**: “must reach win”.
- **Strict Reachability with Avoidance (Until):** “must reach win and must avoid lose”.
- **Weak Reachability with Avoidance (WeakUntil):** “should reach win and must avoid lose”.
- **Pure Safety:** “must avoid lose”.
- **Time Optimality:** “must reach win within less than u-g time units and must avoid lose”.

Controller Strategy Synthesis

In order to study the effects of the execution deviating from the temporal plan, the generated clock constraints in the timed game automata are modified to include overlaps in clock guards. We perturb the state and action transitions originally planned by up to 10 time units. This produces automata in Figure 45 with overlapping timelines, that were not intended at plan time. Note that this automata only have a few controllable regions in Robot 3 and Robot 4 We would like to control these transitions such that properties are maintained during execution. The robots and humans should avoid collisions and deadlocks, despite the temporal deviations.

The objective is to generate a strategy for the dispatcher such that the following property Strict Reachability with Avoidance (Until) is satisfied (note that other properties such as strict reachability and weak reachability may also be verified similarly):

control: A[not(((Robot2.Drop_Obj2_B4) and (Robot3.Pick_Obj2_B4)) or (Robot1.Drop_Obj1_C2 and Robot4.Pick_Obj1_C2) or ((Robot4.Move_D6_E6) and (Robot3.Drop_Obj2_E6))) **U** (Robot1.Goal and Robot2.Goal and Robot3.Goal and Robot4.Goal)]

The above control properties prevents three occurrences of deadlocks/collisions until the goal is reached. Uppaal-Tiga generates the following winning strategy. Given the relative states of the robots and observed clock values rules are generated to takes transitions for Robot 3 and Robot 4. These rules, when followed, will ensure ``winning" against uncontrollable environmental conditions -- maintaining the temporal ordering of robot events.

Strategy to win:

State: (Robot1.Goal Robot2.Move_D3_D2 Robot3.Pick_Obj2_B4 Robot4.Move_E5_D5)

When you are in (Robot2.y==170 && Robot2.y==Robot3.z && Robot3.z==Robot4.m && Robot4.m==170), **take transition** Robot3.Pick_Obj2_B4->Robot3.Move_B4_B5 { z >= 170, tau, 1 }

....

State: (Robot1.Goal Robot2.Move_C4_C3 Robot3.Move_B5_B4 Robot4.Drop_Obj1_E5)

When you are in (Robot2.y==160 && Robot2.y==Robot3.z && Robot3.z==Robot4.m && Robot4.m==160), **take transition** Robot3.Move_B5_B4->Robot3.Pick_Obj2_B4 { z >= 160, tau, 1}

In order to visualize the outputs of the original timed plan and the generated strategies, we make use of timed message sequence charts (MSC) in Figure 45. The vertical lines in the timed MSC model represent instances of entities that participate in the message passing. A message is denoted by an arrow from the sending instance to the receiving instance. The messages are labeled, with the task actions that are to be performed. The messages are ordered according to the timelines specified in the temporal plan. Time stamps are provided for individual actions, with timers t1, t2, t3 and t4 initialized. Dotted time measurement vertical lines are used to measure events at different temporal levels. A slanted transition in implies the range of time during which the transition may occur.

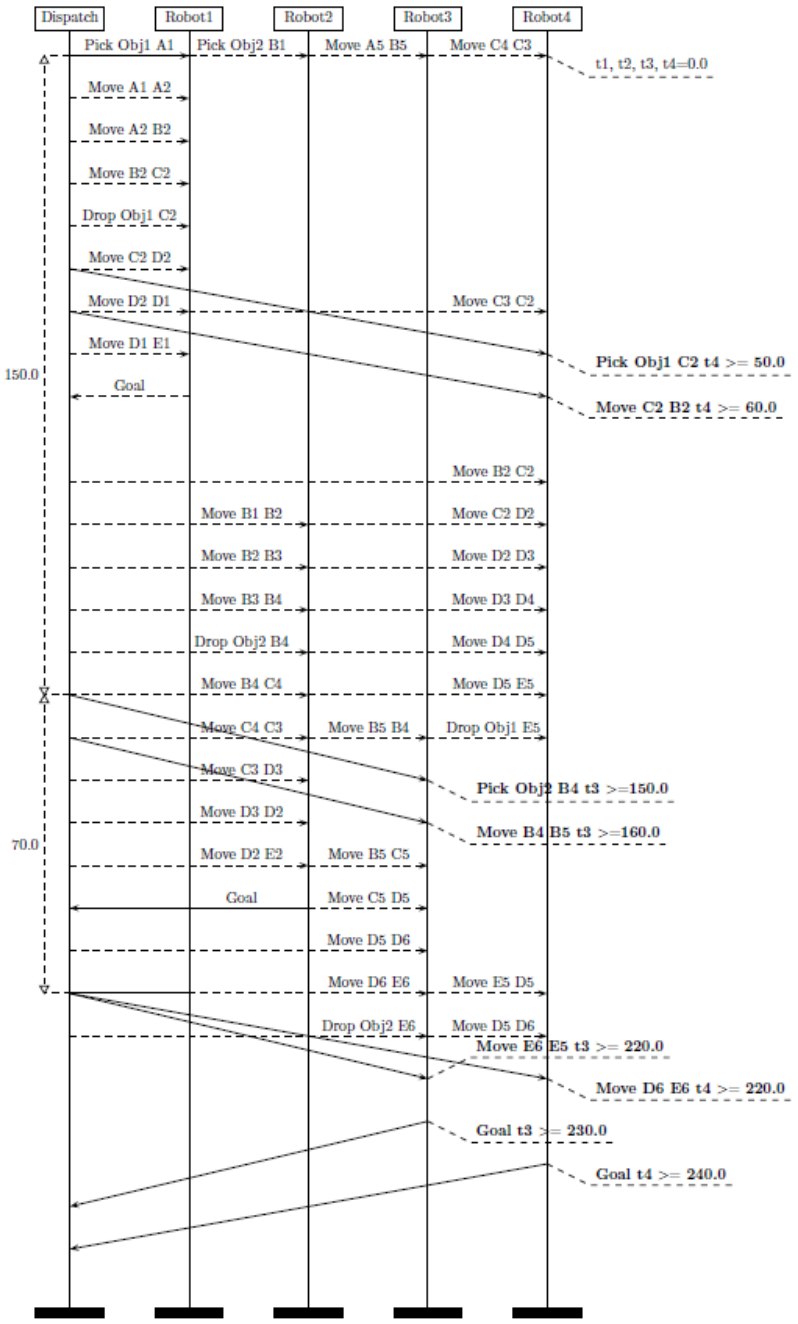


Figure 45. Plan Strategy with Uncontrollable Transitions Temporal Drifts of ±10 units, Optimal Controllable Zones.

There are multiple rules associated with the same transition, that needs to be monitored. **With 37 rules**, the system is able to control the robots despite temporal drifts of up to □10 time units in each planned step. It is important to judiciously select the controllable transitions, as analyzed next.

Optimal Controller Zones

Another important aspect to consider is the complexity of the control strategy that is generated. As these are rules on relative states and clock values of multiple agents, increasing the number of rules would imply additional complexity to the controller. As the proposed system is envisioned to reduce the overhead of heavy uncertainty planning or runtime re-configurations, it is important to analyze the tradeoffs between choice of controllable robots and the number of rules generated by Uppaal-Tiga. We make use of the following technique to compare two strategies ST_1 (time_1, rules_1) and ST_2 (time_2, rules_2):

Optimal Strategy (ST_1, ST_2) =
if (time_1 <= time_2) then ST_1
elseif (rules_1 <= rules_2) then ST_1
else ST_2

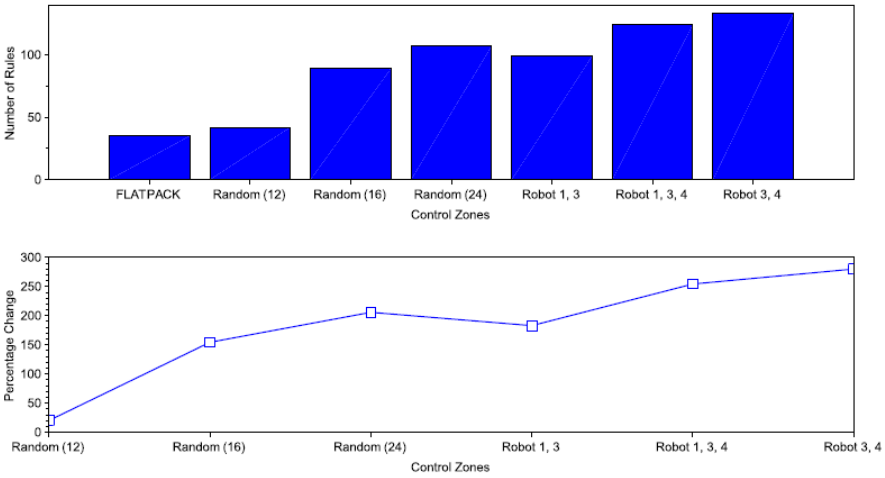


Figure 46. Number of Control Rules vs. Control Zones.

We begin with the analysis of Figure 46, which demonstrates the number of control rules for varying number of controllable transitions in the timed automata from Figure 44. The minimum number of rules are generated using our Algorithm 1 followed by an increase if random transitions are added. We see a **150%** increase in the number of rules if Robots 1 and 3 are made controllable. In order to reduce the overhead on monitoring and enforcing a high number of control rules, a systematic approach to generate the controllable transitions effectively, so as to minimize this overhead.

Algorithm 1 presents the iterative approach for generating the minimal number of controllable transitions. Starting from the initial states Φ_p^S on which the property has to be derived, the algorithm iteration increases the controllable zones until the winning strategy in UPPAAL-TIGA is found. If no winning strategy is found even when $\phi_i^{\lambda_u} \leftarrow \emptyset$ (all transitions are controllable), either the jitter model or the temporal plans have to be reconfigured. This has been implemented over the robotic case study to provide minimal controllable zones.

Algorithm 1: Generating Optimal Control Zones.

Input: Timed game automata $\mathcal{G}$ with control property $\mathcal{P} = \mathbf{A}(\phi^S, \phi^\lambda)\mathbf{U}\phi_g$ to be verified until goal condition
Output: $\phi^{\lambda c}$ the set of observable controllable actions; $\phi^{\lambda u}$ the set of observable uncontrollable actions
Initialize controllable actions $\phi^{\lambda c} \leftarrow \emptyset$ in $\mathcal{G}$;
Initialize uncontrollable actions $\phi^{\lambda u} \leftarrow \phi^\lambda$ in $\mathcal{G}$;
for *each state* ϕ^S *in property* $\mathbf{A}(\phi^S, \phi^\lambda)\mathbf{U}\phi_g$ **do**
    Change every incoming and outgoing transition of ϕ^S to controllable $\phi^{\lambda c}$;
    Evaluate *winning strategy* in UPPAAL-TIGA over the timed game automata $\mathcal{G}$;
    if winning strategy *in* UPPAAL-TIGA *maps correctly to* $\mathcal{P} = \mathbf{A}(\phi^S, \phi^\lambda)\mathbf{U}\phi_g$ **then**
        Return the optimal control zones $\phi^{\lambda c}$;
        stop ;
    else
        while *winning strategy in* UPPAAL-TIGA *is not found* **do**
            Change every incoming and outgoing transition to current controllable zone $\phi^{\lambda c}$;
            Evaluate $\mathcal{P} = \mathbf{A}(\phi^S, \phi^\lambda)\mathbf{U}\phi_g$ over current controllable zones;

Algorithm 1 presents the iterative approach for generating the minimal number of controllable transitions. Starting from the initial states on which the property has to be derived the algorithm iteration increases the controllable zones size gradually until the winning strategy is found.

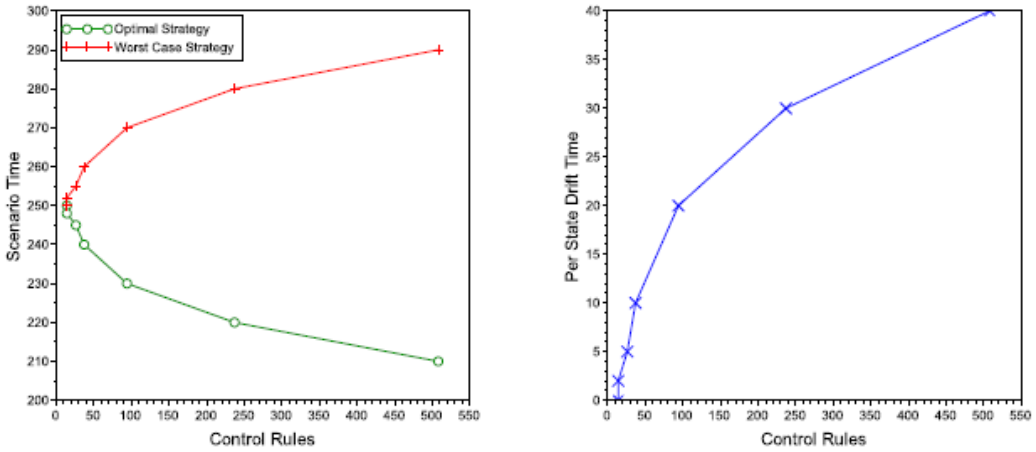


Figure 47. (a) Optimal / Worst Case Strategy Times for Increasing Control Rules, (b) Per Plan State Temporal Drifts against Control Rules.

With an increase in the amount of temporal drifts per state, we notice an increase in the number of control rules in Figure 47(b). For instance, with drifts of 10 time units, the optimal control zones provide 37 rules; if this is increased to 20 time units, the number of rules increases to 98. This also causes a deviation in the best case and worst-case execution times as shown in Figure 47(a). With 100 control rules (corresponding drift of 20), the scenario time ranges lie between 230-270. Similarly, for 510 control rules (corresponding drift of 40), the scenario time ranges lie between 210-290. These tradeoffs must be taken into account before categorizing the drifts that can be handled via execution side strategy synthesis vs. temporal re-planning from current states.

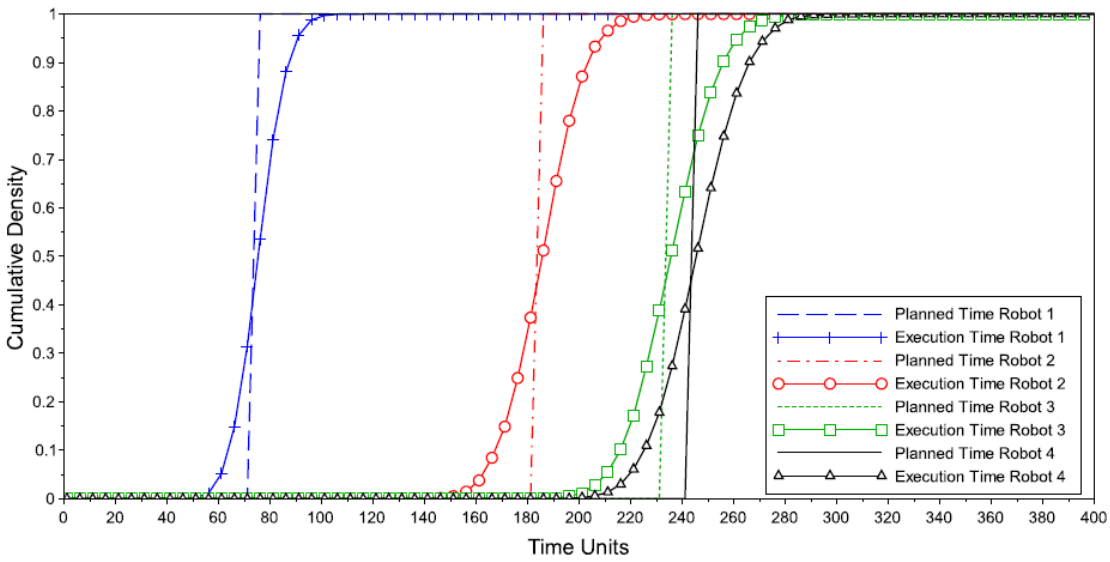


Figure 48. Plan Completion Timelines with Deviations.

The execution times of the scenario are presented in Figure 48. The planned times are compared with the best-case scenario strategies executed. The strategies are then simulated with a Poisson arrival process with mean times as those specified in the strategy. We notice that there are a range of values that the robots can complete executions, that must be taken into account when deriving strategies. For instance, even though Robot 1, 3, 4 complete the end-to-end plan within 200 time units, Robot 3 can take 220 time units to complete. Strategies may be generated up to a certain threshold, after which coarse re-planning or re-evaluation of the planning domain model may be needed.

ROS ActionLib Integration

The last stage of the temporal plan verification and control system in Figure 40 is the integration with physical robots running Robot Operating System (ROS). The temporal plans and optimized dispatched strategies are integrated using the ROS **actionlib** framework[12]. The actionlib package allows creation of client-server applications to executed long-running goal tasks with feedback and pre-emption. The ActionClient periodically sends a goal task to the server with information about success criteria (state, time, location). The ActionServer attempts to perform the task (with robotic capabilities), with a final result sent in completion of the goal task. Incremental feedback might be provided to the ActionClient, that updates on the progress of the task.

[12] https://github.com/ros/actionlib

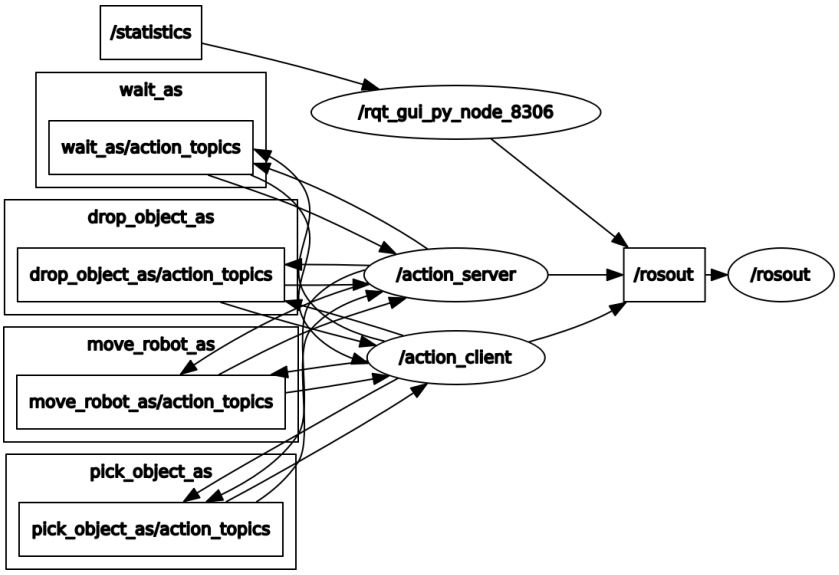


Figure 49. ROSGraph Representing Nodes and Messages Passed to the actionlib Server /Client.

As we would like to integrate optimal dispatch strategies that maintain safety properties despite temporal deviations, we make use of the ROS nodes and messages in ROS actionlib as provided in Figure 49. Along with the move, prick and drop tasks seen in the grid scenario of Figure 41, we add the additional **wait** task, that can be enforced by the controller. An output of the received feedback from the ActionServer when completing a plan is shown below.

Feedback received: robot_moved:

"robot moved location - 1 timestamp: 12:45:53 -- time bounds exceeded"

Feedback received: robot_moved:

"robot moved location - 2 timestamp: 12:46:16 -- completed within time bounds"

Feedback received: waiting_robot:

"Robot Waiting Time Units 7 timestamp: 12:47:12"

Feedback received: robot_pick_object:

"robot picked object 1 timestamp: 12:47:34 -- completed within time bounds"

Feedback received: robot_moved:

"robot moved location - 3 timestamp: 12:48:33 -- time bounds exceeded"

Feedback received: waiting_robot:

"Robot Waiting Time Units 8 timestamp: 12:50:42"

Feedback received: robot_drop_object:

"robot dropped object 1 timestamp: 12:51:07 -- completed within time bounds"

As noticed, there are cases where the planned time bounds are exceeded, triggering an appropriate waiting time at a later stage according to the strategy. This allows for successful execution of planned goal tasks in a temporally sound, verifiably safe and flexible manner.

3.3.8 Monitoring and Visualization of Warehouse KPIs

The dashboard supports different stakeholders, such as warehouse manager, floor supervisor, human/skilled worker, to monitor different KPIs related to the warehouse. In this scenario, several components of the warehouse work autonomously to fulfil the inventory replenishment, storage and delivery requests. These components of the warehouse can be listed as: Automated Storage and Retrieval Systems (AS/RSs); robotics arms; and autonomous robots. There are several additional components such as cameras, conveyor belts and humans that collaborate with the robots to accomplish tasks related to the business objectives. Figure 50 illustrates these components.

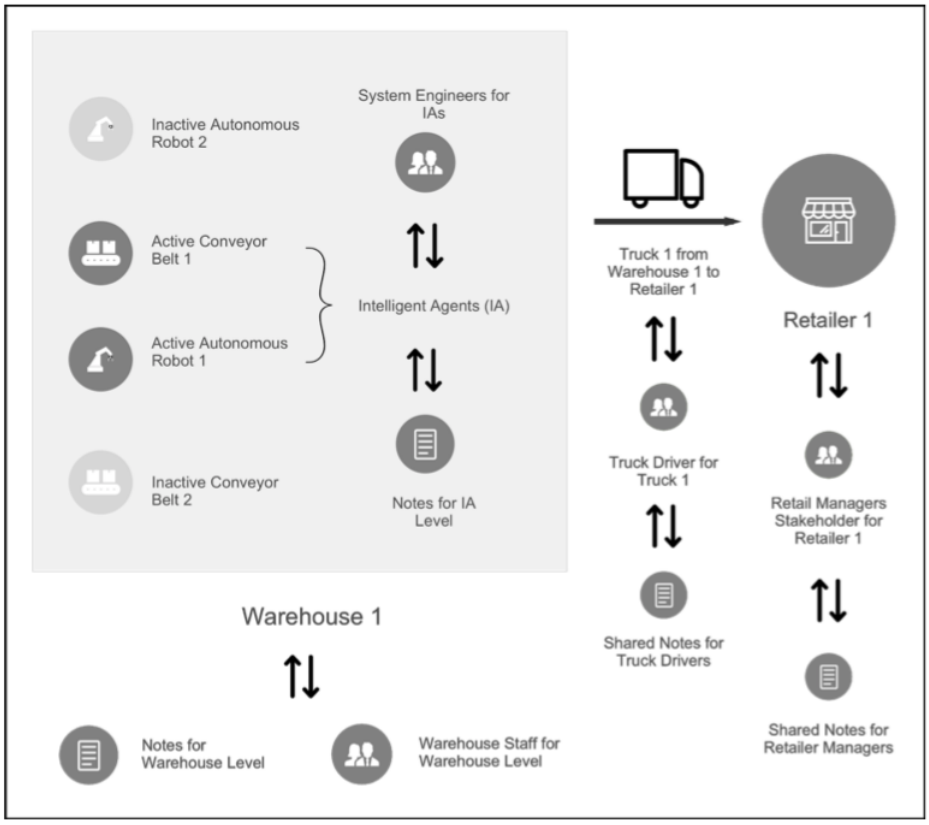


Figure 50. An overview of warehouse with different kinds of active robots (in this case they are Autonomous Robot 1 and Conveyor Belt 1).

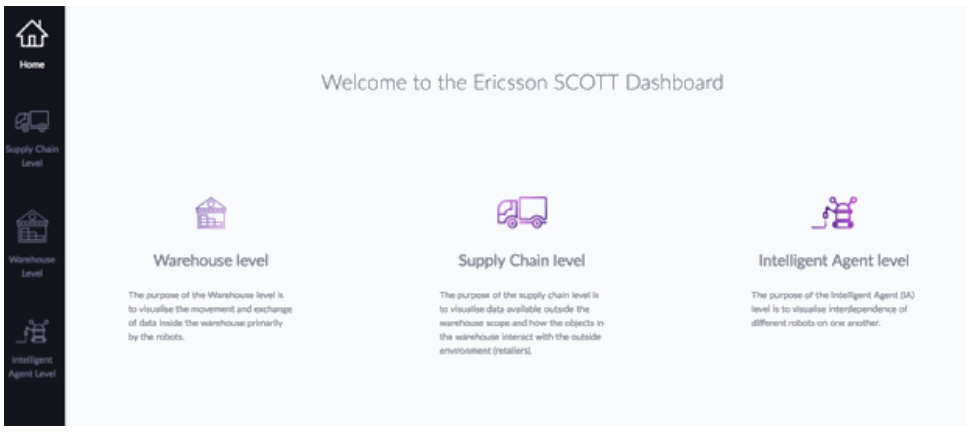


Figure 51. The front page of the dashboard where different levels of the system are listed.

The dashboard aims to visualize information not only related with the warehouse but also retailers and trucks (Figure 51). Therefore, the first page of the dashboard shows three different levels related to the warehouse. For instance, Figure 52Figure 52 shows detailed information about the Truck 1 such as, name, Truck ID, Destination from, Destination to, Status of the journey in percentage, details analysis about the sustainability metric in percentage and comparison of the activity time in relation to the average. In next paragraphs, we will only focus on the warehouse level and explain the visualizations related to components/entities which operate inside the warehouse such as robots, robotic arms, and conveyor belts and so on.

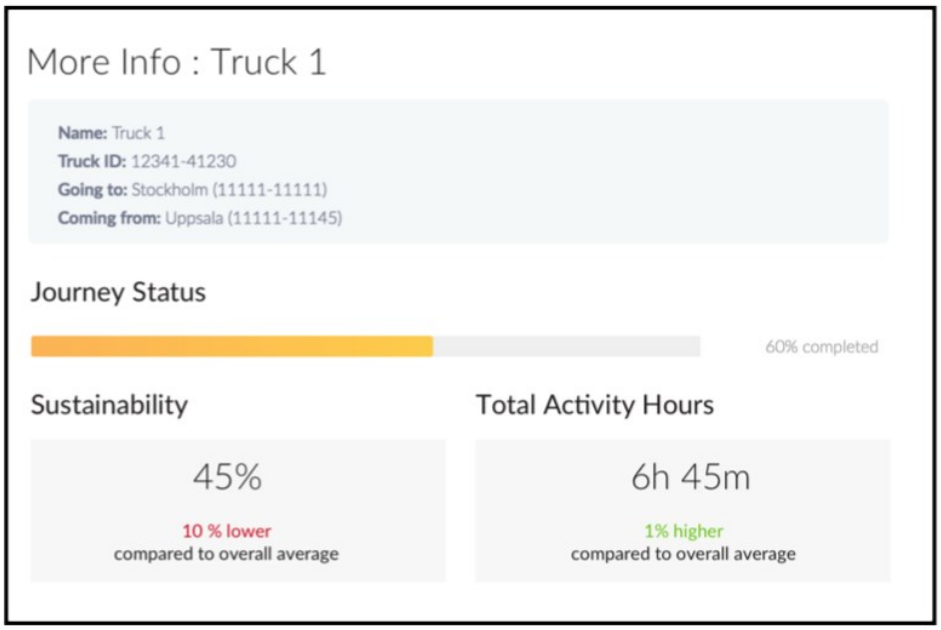


Figure 52. The modal with secondary information about the truck including the total activity hours and the metric related to the sustainability.

One of the important KPIs of the warehouse is about the battery level of robots. Figure 53 shows the map of the warehouse. The dotted lines are representing different way points. The circle in green tones are representing robots where the size and opacity of the circle is proportional with the battery level. Moreover, the direction of the robot is illustrated by arrows and destination is represented either by object or charging station graphics/icons.

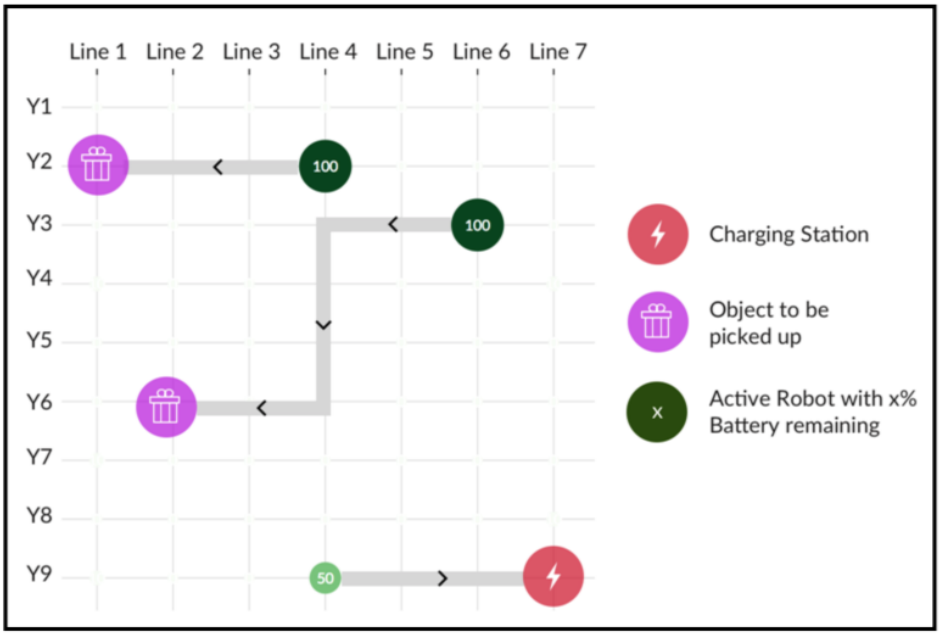


Figure 53. Map view of the warehouse where different robots, their end destination and battery level are illustrated.

The dashboard user can acquire more information related to each robot by clicking on the green circles. Figure 54 shows this detailed view of the Robot 1. The top left corner of this pop-up window shows the privacy level of the robot. The name, robot ID, destination information is shown on the right top corner. Description about two metrics are presented in the middle section of the window and at the bottom the metrics are shown. The danger zone metric, which is related to the safety KPI, shows the distance of the robot to the nearest object and it changes the colour from green to yellow to red according to the distance.

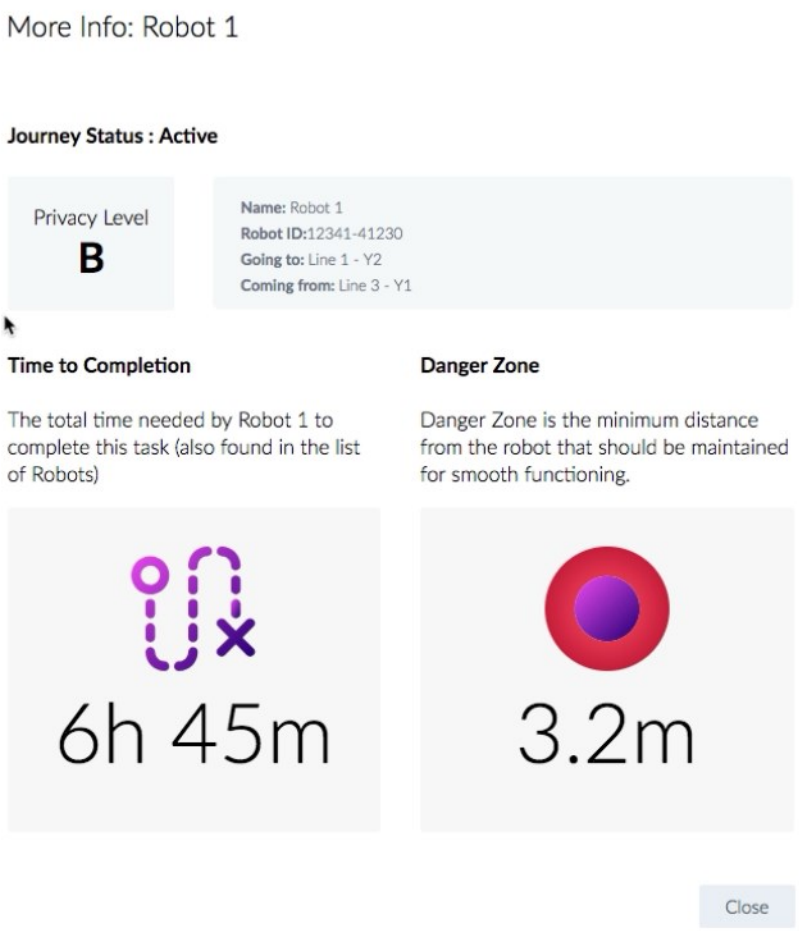


Figure 54. Details of Robot 1 are presented in a pop-up window.

The performance of different components is also displayed through a bar chart. Figure 55 shows the visualization related with this KPI. User is able to add or subtract more components to this list by searching these components. This allows the user to compare different robots and to understand how the performance of a robot can be optimized. It is also possible to search information by time to see overall performance of different components of the warehouse in a specific period of time.

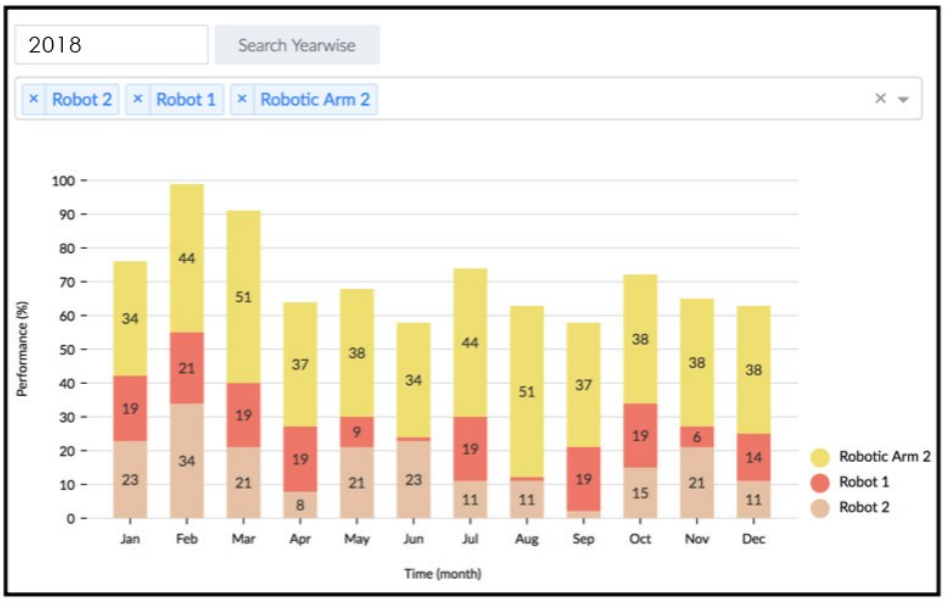


Figure 55. Stacking different robots in the warehouse with each other to compare the performance of their combination for the year 2019.

The interoperability KPI is mainly visualized by extracting the interaction between different components of the warehouse. Figure 56 shows a chord diagram, which represents the interactions between different robots, robotic arms, conveyor belts and charging stations. User can add or remove more components to see the interoperability between different components or make a year wise comparison.

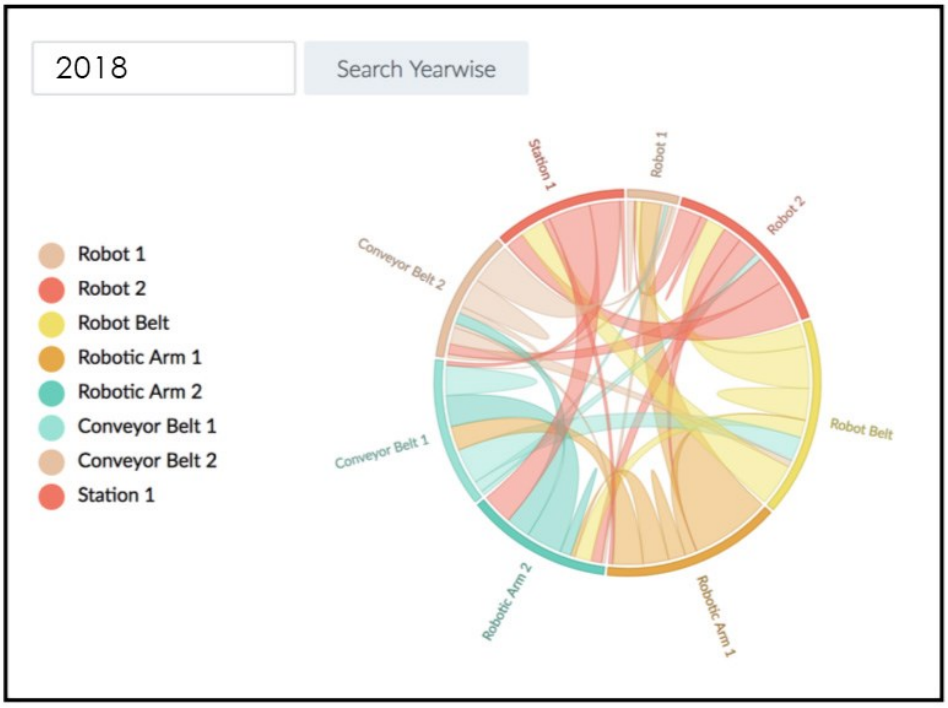


Figure 56. A chord diagram was used to visualize the interoperability between all the available robots present in Warehouse 1 for the year of 2019.

The data and visual analytics approach and the implementation of the dashboard focused on minimalistic data model for monitoring purpose. During the implementation of the data model, team members had the opportunity to talk about what kind of data is required for each purpose, and different ideas were discussed before taking any decisions. At the end of this iterative process, a minimal set of required data was selected to be used.

3.4 Additional use case from RTE

3.4.1 Overview

RTE has developed a connected IoT system for monitoring and controlling houses and facilities such as warehouses. Human inhabitants can interact securely with the system through the developed user interface, to obtain sensor data and affect the environment using connected actuators. Non-human inhabitants such as robots can also interact with the system through APIs in order to monitor and control the state of the building. The information and control afforded by this system can be used by robots when executing mission plans as described in the "Monitoring and Control" use case scenario.

The demonstrator that has been selected is a fish tank or aquarium illustrated in Figure 57. The purpose is just to symbolize this as a building with inhabitants.

Figure 57. Fish tank used in the demonstrator.

The automation that has been implemented in the demonstrator consists of the following sensors and actuator:

- Temperature in water (“indoor climate”).
- Temperature in air (“outdoor climate”).
- Relay for switching on and off the lightning (“automation”).
- Camera for pictures of the fish tank (“surveillance”).

There is a front-end or Graphical User Interface (GUI) as depicted in the Figure 58 and Figure 59.

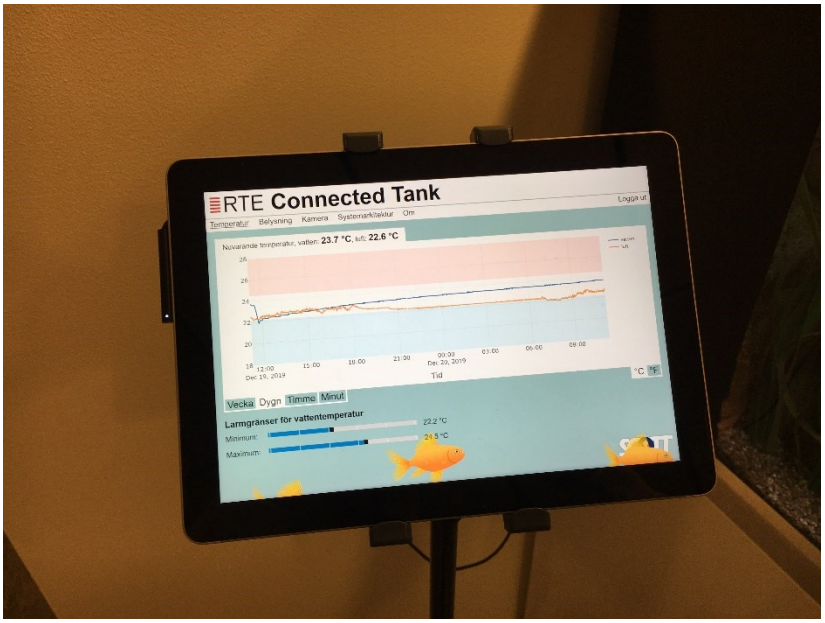


Figure 58. GUI showing the historical and current temperature in a graph format.

The graphs show the temperature values now and historically (Figure 58). There are two alarm thresholds connected to the water (“indoor”) temperature, a max and a min value. If temperature goes outside these values, an alarm indication will be visualized on the GUI. The picture from the camera is pushed to the GUI every minute (Figure 59).

Figure 59. GUI showing the camera image taken each minute.

3.4.2 Design and V&V of Planning Domain Model

Overview

The definition of the planning domain for warehouse logistics is a critical part of the use case because the goal-oriented control of the robotics is driven by the plans derived from the domain definition and the current state of the warehouse. A planning domain contains the types of objects, various predicates and functions, and the model of actions that the robots are capable of carrying out. Before using the domain definition for plan synthesis, it is checked for Consistency, Completeness and Correctness [24].

Consistency

Consistency of PDDL model is an issue which is internal to the specification. The model as specified should be consistent i.e. the initial state should be consistent and at no point the execution of an action should lead to an inconsistent state. Consistency check can be done by simulating a plan and detecting the presence of a predicate and its negation in the knowledge base.

Correctness

Correctness of PDDL domain models is an issue which needs an external oracle (e.g. human tester, specification) which checks if the run-time behaviour of the model is as expected. To a large extent, this can be verified by simulating the PDDL model. Given a state before an action, the simulator can give one-step execution and the resulting state. The tester can inspect this to ensure that the intended effects have indeed taken place.

Completeness

Completeness of PDDL model is a measure of the behaviours that are possible in the real world but not in the PDDL model. A planner will not be able to generate a plan from an incomplete model even if plans to achieve the required goals exist in the real world. It is a different thing that the planners may themselves be incomplete. However, it is always better to capture as much of the domain behaviour as possible in the model.

In order to support this, the user can specify a known set of sequences and verify that these are valid runs in the PDDL model i.e. they can be simulated.

Solution

In order to discover the design issues described above and debug the domain models and problems, we provide a lightweight tool implemented using the C# language (.NET platform). The tool processes domain and problem files (actually, only the initial state) and presents a visual simulation tree to the user. Starting from an initial state (Figure 60), the tool presents all actions possible from a state and lets the user select an action for simulation (Figure 61). It is possible to undo actions and go up the simulation tree to select other branches. One can go back to the initial state through a *reset* button.

The predicates in each state is listed in a side bar. When an action is selected, the deleted and added predicated are highlighted which helps the tester check the expected progress. It is easy to check deadlock states during simulation; the predicates listed in the state can be checked with the preconditions of the expected actions at this state. As noted above, the consistency, correctness and completeness checks can be done through this visual simulation tool.

In order to help the user better, the tool helps (1) gradual building of the state graph starting from an initial state, and; (2) directly designating a state as current state and starting to explore the state graph below it.

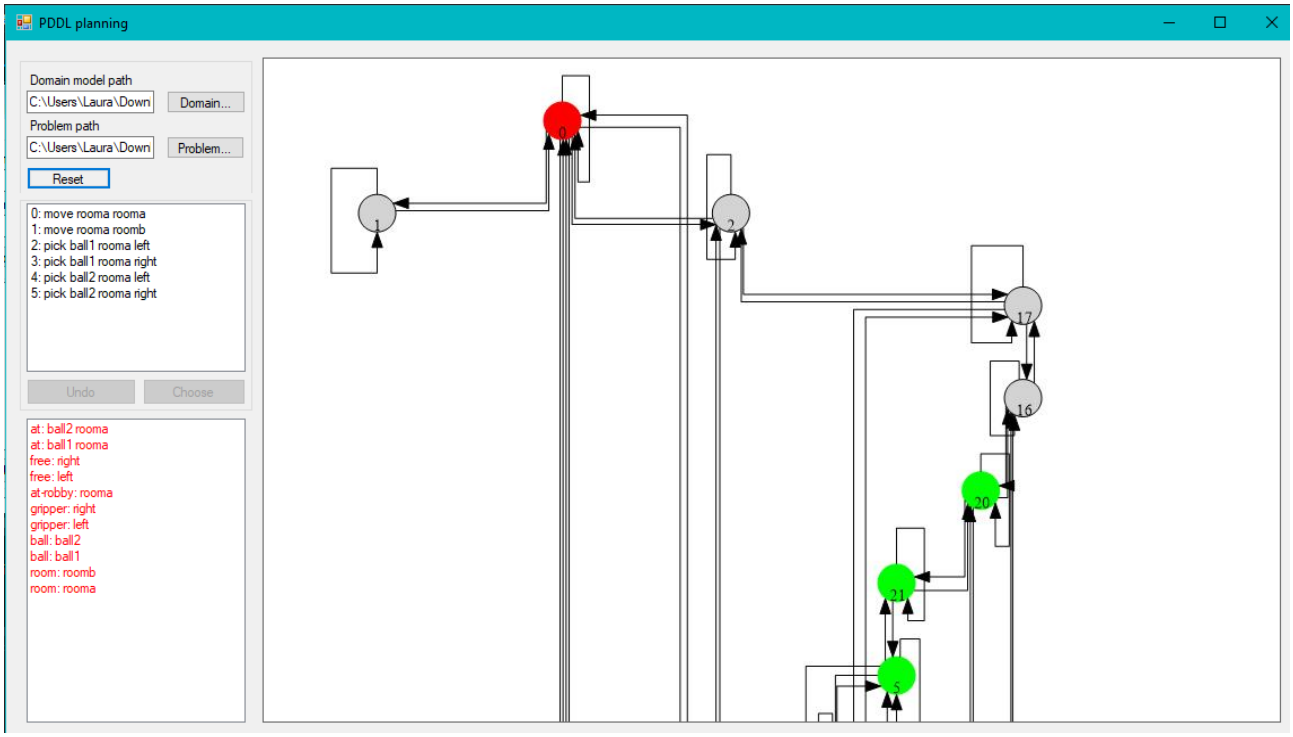


Figure 60. The state tree and initial state after importing the domain and problem file.

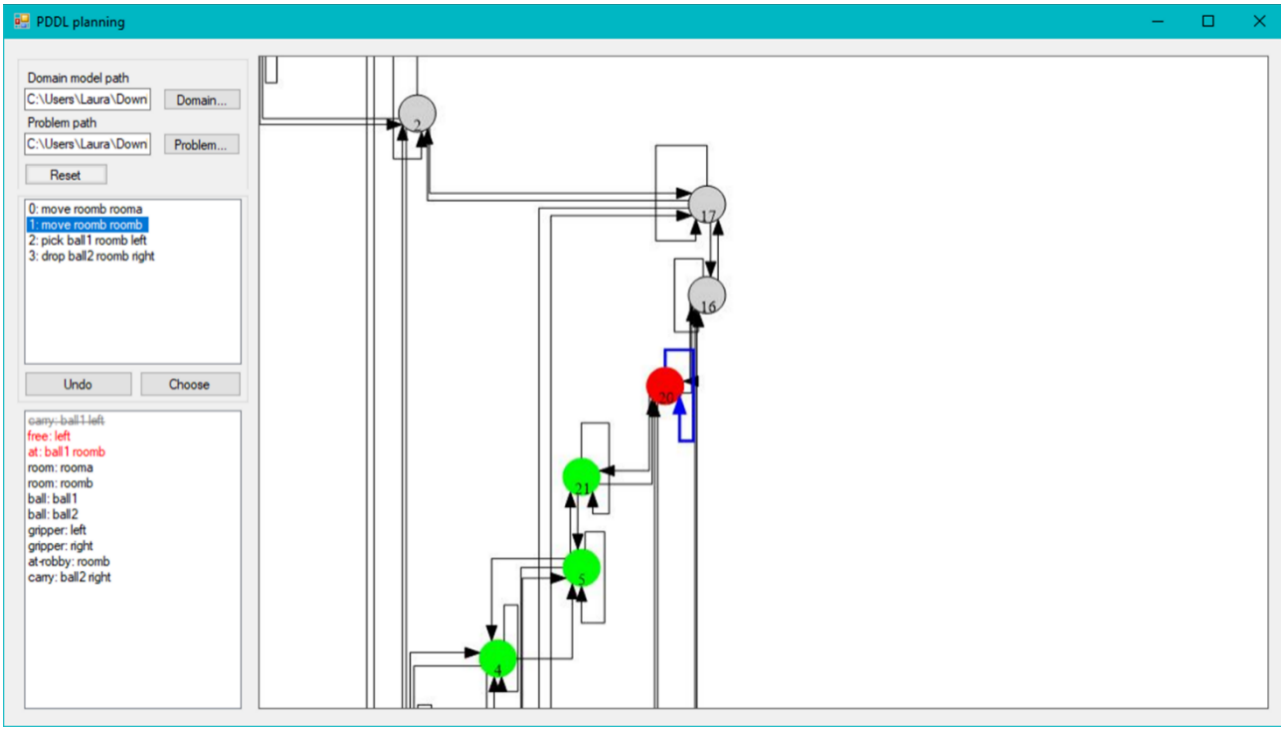


Figure 61. Selecting an enabled action and deleted/added predicates shown on left. The tree highlights in red the current state after the action is executed.

3.4.3 System Architecture

A general system architecture of the implemented Proof of Concept is shown in Figure 62.

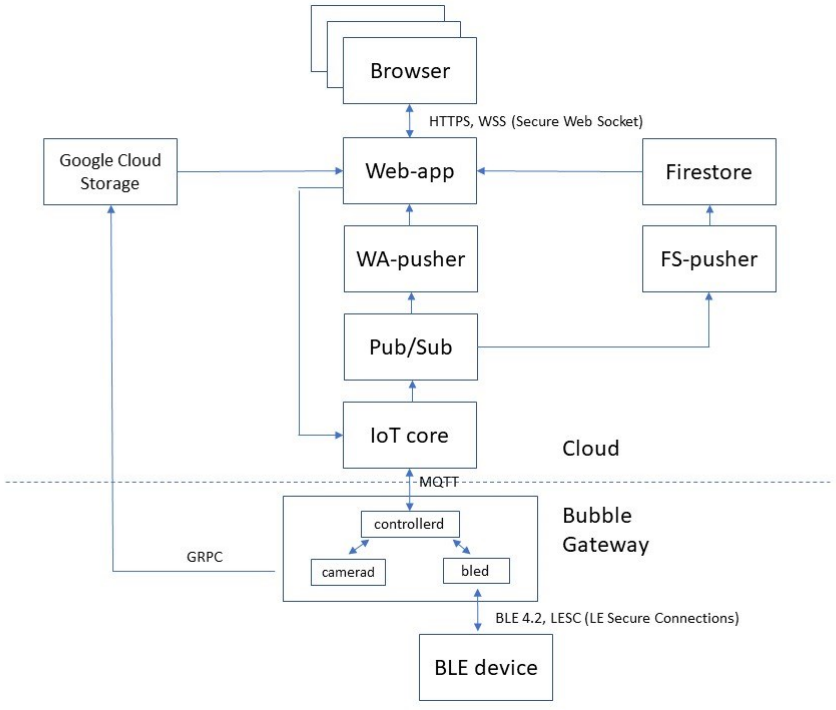


Figure 62. Architecture of the demonstrator.

The system architecture is formed around the Google Cloud solution for IoT. It is assumed that the Google components are known to the reader. A short explanation of the RTE developed components starting from the top:

- Browser: provides a GUI for the IoT application. The GUI presents sensor data – both current values and historical data. The GUI also gives the possibility to set a schedule or change the state of actuators. Finally, the GUI shows the latest picture taken by the camera sensor. The GUI has been implemented as a web browser and is accessible from anywhere. User control is by basic authentication.
- Web-app: Acts as a backend and handles all requests from the Browser regarding sensor data and actuator commands.
- Google Cloud Storage: This is used to store the pictures taken by the camera sensor.
- Bubble Gateway (a.k.a. Gateway): Consists of three software parts on top of a custom-built Linux distribution.
    - Controlled implemented as a daemon handles the MQTT towards the Cloud and camera and BLE processes.
    - BLEd handles the BLE (Bluetooth Low Energy) wireless link towards the sensor/actuator devices
    - Camerad handles the camera sensor data communication with Google Cloud Storage via GRPC protocol.

3.4.4 Specifications

Our general specifications that we have used in this project are very similar to the commercial projects we do for industry clients.

General requirements:

- Use IPv4 based internet access from Gateway to Cloud.
- Internet access from Gateway is wired.
- Gateway must be able to work even if internet access is temporarily broken, e.g. for 1-2 days.
- Security between Gateway and Cloud should be certificate based.
- No need for UPS for the Gateway.
- Gateway and devices should be based on COTS hardware components.

Specific SCOTT requirements:

- The connection between Gateway and sensor/actuator devices must be wireless.
- The wireless connection should be of type low energy since sensor/actuator devices are expected to be battery powered.
- The wireless connection must be encrypted to prevent from a Man-In-the-Middle attack.
- The Gateway must be able to handle any number of devices.
- Camera sensor can be connected by wire to the Gateway.

3.4.5 Implementation

Primary software for the various system components:

- Browser: Javascript.
- Web-app: Python.
- Gateway: Python and C, and a custom Linux (Yocto) build.

Development of software for Browser, Web-app, Gateway and BLE devices has been made using the Visual Studio Code IDE. Browser and Web-app are both running on Google Kubernetes Engine.

The Gateway consists of:

- Nitrogen6x board for hosting controllerd, bled, and camerad software.
- Alternative: RPi 3B+ board (we have both installations).

3.4.6 Potential Application Scenarios

The use cases and scenarios that has been seen are most type of Industrial IoT solutions where the industry wishes to connect some of their existing equipment to the cloud. There is no limit in type of industries where this is of interest.

There are two types of use cases:

- Client has equipment they want to connect to the cloud. Main reason is usually to see when it is time for maintenance, i.e. preventive maintenance. But also, to get other types of information from various sensors to make a value-added product proposition.
- Client has equipment that runs a PLC type of local controller but want to connect the equipment to the cloud to get a possibility for remotely manage and control the equipment.

3.4.7 Relation to requirements and building blocks

BB24.C Application Layer Protocols

The RTE demonstrator uses the application protocols MQTT for gateway/device management and telematics, and GRPC (over HTTP with TLS) for uploading of images.

Pub/Sub is used for communication and data storage in the backend.

BB26.B Cloud Computing Services Platform

The RTE demonstrator uses the protocol MQTT for connecting the external devices to the Cloud. The Cloud is implemented as a Google Cloud using IoT Core, Pub/Sub, Storage, and Firestore.

Requirements addressed in the RTE demonstrator:

- DemoRQ 640 Monitoring and control
- DemoRQ 643 Pubsub exchange

3.4.8 Conclusions

The RTE demonstrator has successfully accomplished its target of implementing Monitoring (sensors) together with Control (actuators) and demonstrated its use in a web browser-based user interface. We have also demonstrated the two Scott objectives Wireless Systems and Focus on Security, Safety, Privacy and Trustability. More specifically, BLE been used for wireless communication together with encryption to connect sensors and actuators to the Bubble Gateway. For the Bubble Gateway, we have used a well-suited application protocol (MQTT) to connect the devices (via the gateway) to the Cloud. For security we have also adopted Keycloak for enhancing the security between the user interface and the Cloud. For future outlook, it could be interesting to also add elements of edge based machine learning, serving the robots in the warehouse and possibly also using other wireless standards.

3.5 Evaluation and future outlook

3.5.1 Work Package demo evaluation

The evaluation of the demo sandbox is formally done by assessing the completeness of the Demo requirements. Satisfaction of the demo requirements is evaluated below according to the standard SCOTT requirement assessment scoring for the Use Case work packages in SP2 [[25], §3.1.2]. Note that the adjusted scale [[25], Table 2] for the non-waterfall model is used, as WP10 follows agile approach.

Requirement	Self-assessment score
REQ-640 (DemoRQ) Monitoring and control	100%[13]
REQ-641 (DemoRQ) Digital Twins	70%
REQ-643 (DemoRQ) Pubsub exchange	100%
REQ-648 (DemoRQ) Robot and robot software	70%
REQ-650 (DemoRQ) Testing framework for PDDL domains	100%
REQ-681 (DemoRQ) Safe operations	100%
REQ-682 (DemoRQ) Collaborative Operations	70%
REQ-689 (DemoRQ) NLP Component	70%**[14]

Table 10. Demo requirements score self-assessment for WP10.

[13] REQ-640 includes the monitoring for the internal purposes of control but also for the visualisation and monitoring by the human operators. Only the monitoring for the internal purposes through the use of Digital Twins and the Pubsub are part of this deliverable.

[14] REQ-689 was added for year 2 and described in D10.3 [4].

For the reviewers' convenience, below is the summary of the demo requirements that were dropped:

- REQ-639 (DemoRQ) Operation Optimisation. Dropped to focus on intralogistics.
- REQ-647 (DemoRQ) Warehouse instrumentation. Merged with another requirement.

The average completion status of the demo requirements is 85%.

Additionally, the readiness of the components in the demo can be seen on the architecture diagram as shown in Figure 3. The components that are not fully complete (shown in grey) map to the requirements that were fulfilled to a major extent (70% or more according to the SCOTT classification) or map to the requirements and scenarios that were dropped during the course of the project.

3.5.2 Overall Work Package evaluation

Formally, the evaluation of the overall WP progress is done by assessing the completeness of the requirements in the demo scope as well as the relevant requirements from the connected BBs (see Section 5) according to the SP1 leadership guidelines. In the overall WP evaluation, we take into account both the applicable requirements from the associated Building Blocks as well as the WP demo requirements.

Requirement	Self-assessment score
REQ-640 (DemoRQ) Monitoring and control	100%[15]
REQ-641 (DemoRQ) Digital Twins	70%
REQ-643 (DemoRQ) Pubsub exchange	100%
REQ-648 (DemoRQ) Robot and robot software	70%
REQ-650 (DemoRQ) Testing framework for PDDL domains	100%
REQ-681 (DemoRQ) Safe operations	100%
REQ-682 (DemoRQ) Collaborative Operations	70%
REQ-689 (DemoRQ) NLP Component	70%**[16]
REQ-644 (BB24.C) Common Knowledge Representation	100%
REQ-651 (BB24.C) Linked Data Platform	100%
REQ-646 (BB24.D) Plan verification for safety	100%

[15] REQ-640 includes the monitoring for the internal purposes of control but also for the visualisation and monitoring by the human operators. Only the monitoring for the internal purposes through the use of Digital Twins and the Publish-Subscribe are part of this deliverable.

[16] REQ-689 was added for year 2 and described in D10.3 [4].

Requirement	Self-assessment score
REQ-262 (BB24.E) CDN for content delivery	70%
REQ-649 (BB24.E) Software upgrade in controllers	90%
REQ-260 (BB26.B) Support IPv6	80%
REQ-261 (BB26.B) Auto Scaling System	80%
REQ-263 (BB26.B) Connection Restrictions	100%
REQ-264 (BB26.B) Auto Deployment	70%

Table 11. Demo and BB requirements score self-assessment for WP10.

Across the requirements targeting the WP10 and the related requirements, the average completion status is 86%.

For the reviewers' convenience, below is the summary of the BB requirements that were dropped:

- REQ-642 (BB24.D) Knowledge transfer. Considered in the concept and dropped (10%). Knowledge transfer in general is ensured through the use of ontologies (see Section 3.3.2.5 of this deliverable and REQ-644), among other approaches.
- REQ-645 (BB24.D) Strategic and reactive planning. Implemented partly, reported in D10.2. Dropped to focus on verification and communication aspects of the sandbox.

Additionally, the informal rating of the scenarios in one of the categories (Small Extent, Large Extent, Complete) is presented in the Table 12 in order to allow easy comparison how requirement status maps onto the use case scenario completeness.

Scenario	Self-assessment score
Digital Twins	Large Extent
Task Level Planning	Complete
Risk Assessment for Safe Operations	Complete
Goal State Generation out of NLP Specifications	Large Extent
Explanations of Synthesized Plans	Large Extent
Formal Verification of Strategic Plans	Complete
Monitoring and Visualization of Warehouse KPIs	Complete
Design and V&V of Planning Domain Model	Complete

Table 12. Demo and BB requirements score self-assessment for WP10.

4 DISSEMINATION, EXPLOITATION AND STANDARDISATION

All the source code used to develop the final sandbox are maintained in the following repository:

https://github.com/EricssonResearch/scott-eu

The efforts for Year 3 have already been reported in D10.5 and D10.6. The following publications have been done since the deliverable D10.6:

1. A. Kattepur and S. K. Mohalik, "FLATPACK: Flexible Temporal Planning with Verification and Controller Synthesis", 8th Planning and Robotics Workshop (PlanRob), 30th International Conference on Automated Planning and Scheduling (ICAPS), 2020. (submitted)
2. A. Buksz, A. Mujumdar, M. Orlic, S. Mohalik, M. Daoutis, B. Ramamurthy, D. Magazzeni, M. Cashmore, A. Vulgarakis Feljan, “Intent-driven Strategic Tactical Planning for Autonomous Site Inspection using Cooperative Drones”, IEEE/RSJ Intelligent Conference on Robots and Systems (IROS), October, 2020.

The efforts for Year 2 were reported in D10.2, D10.3, and D10.4 and at the F2F meetings in Gdansk and Madrid, while efforts for Year 1 were reported in D10.2 and at the F2F meetings in Porto and Tromsø.

5 LINK TO TECHNOLOGY LINES

5.1 TL Distributed Cloud Integration (WP24)

WP10 has links to the following Building Blocks:

- BB24.C Application Layer Protocols (mainly work on protocols used by the Digital Twins and robot software)
- BB24.D Big Data Analytics (various components of the system, mainly Monitoring and Visualisation work as well as Digital Twin state processing)
- BB24.E Cloud computing services for novel connected mobility applications (indirect link based on requirements)

5.2 TL Reference Architecture (WP26)

WP10 has links to the following Building Blocks:

- BB26.B Cloud Computing Services Platform (indirect link based on requirements)

6 INTEROPERABILITY

There were no changes since D10.2 [1] to the overall approach to interoperability in WP10. For the reviewers' convenience, the contents are reproduced in full below.

Interoperability is ensured at multiple levels. First, at the project level through the participation in Building Blocks and the alignment of the requirements. The interoperability is further ensured through the alignment of the WP10 architecture with respect to the SCOTT high-level architecture (HLA). Interoperability at the interface level is ensured through the use of open yet standardised protocols, which were analysed in BB24.C.

For the interoperability at the application level, however, the whole architecture of the Warehouse Sandbox developed in WP10 was built with the interoperability in mind. In particular, this included:

1. The development of a semantic Information Model.
2. The development of a Linked Data architecture.

The information model is particularly important for the interoperability because it allows the work from the WP10 to be reused in the future with only slight changes to the lower-level domain to reflect the semantics of the environment where it is applied. Additionally, the same property allows the same architecture to be extended in the future with more kinds of equipment, systems etc. This reflects the real-world scenarios more realistically because business needs constantly require change.

The Linked Data architecture allows the heterogeneous components connected through various protocols to expose their data and the operations via a uniform and small web interface consisting of standardised technologies like HTTP, RDF, OSLC, as well as MQTT and ROS. This allows to lower the barrier of integration for the new systems now and in the future. Finally, a common pattern to apply Linked Data in the enterprise systems is through the use of *adaptors,* where the target system remains unchanged, while a small Linked Data microservice is built on top of it to serve as an adaptor. This further reduces cost of increasing the interoperability by eliminating the need for introducing a breaking change during the integration phase.

7 CONCLUSIONS

This deliverable represents a final report and evaluation of the WP10. The evaluation was done according to the requirements defined at the work package level as well as a wider SCOTT level for the related Building Blocks. Additionally, the evaluation covered scenarios and the sandbox architecture to give a better understanding how the requirements map to the work done. The requirements progress both on the demo and the overall level exceeds the SCOTT-defined level of implementation to a Major Extent of 70% and is 85% and 86% correspondingly. The majority of the use case scenarios are fully completed, and the rest are finished to a large extent. Finally, in the conclusions the connection to the overall WP objectives from the SCOTT Description of Work is made.

Utilization of open-source IoT components for monitoring and recording information from different sensors/actors in the "scene".

All of the components and protocols used to interface with the robots, process information, communicate with the Digital Twins and the rest of the system, including monitoring and visualisation are open. Furthermore, all of the results of the WP are open (both the SCOTT deliverables and the reference implementation hosted on Github and licensed under a permissive open-source license). The only component that is not open-source is a V-REP simulation system used to model the "scene", which was done in order to speed up and simplify prototyping and does not affect the developed components and their interfaces.

Definition and prototyping of formal models that describe cross-functional characteristics between the different devices and the overall goal which would be the task that these devices aim at performing in combination.

Early on, the focus of the WP was shifted to increase effort on the use of formal methods, with a few requirements and use case scenarios being added in the Year 2 of the project, resulting in the demonstration of the components developed for plan explanation, plan verification as well as the validation & verification of the planning domain to assist the domain experts designing trustable systems.

Identification of safety critical cases/malfunctions that may cause abnormal behaviour.

Development and prototyping of pre-emptive mechanisms that can effectively (and within certain margins) eliminate or minimize effects of abnormal behaviour.

Throughout the duration of the project, the work was actively carried out on the *Risk Assessment for Safe Operations* scenario while collaborating with the WP28 to ensure this leads to a more trustable system development. Additionally, newly added scenarios on plan explanation and verification as well as the tools for the domain designers help prevent problems both at the design time and runtime.

Development and evaluation of Visual Analytics techniques in order to assess the interoperability of CPS systems and CPS development environments.

The work has been successfully carried out as part of the *Monitoring and Visualisation of Warehouse KPIs* use case (Section 3.3.8).

8 REFERENCES

[1] SCOTT Deliverable D10.2 "Sandbox (alpha): Initial implementation of the Sandbox and its evaluation and deployment", v1.0, 2018-08-24.

[2] SCOTT Deliverable D10.4 “Sandbox (Beta) - A Sandbox for Intelligent Warehouse Logistics”, v1.0, 2019-04-18.

[3] SCOTT Deliverable D10.6 “Safety Critical Sandbox (Release Candidate)”, v1.0, 2020-04-22.

[4] SCOTT Deliverable D10.5 “Safety Critical Sandbox Specification (Final) - A Sandbox for Intelligent Warehouse Logistics”, v1.0, 2019-07-16.

[5] SCOTT Deliverable D10.1 “Sandbox Specification - A Sandbox for Intelligent Warehouse Logistics,” v1.0, 2017-11-28.

[6] SCOTT Deliverable D10.3 “Sandbox Specification (Beta) - A Sandbox for Intelligent Warehouse Logistics”, v1.0, 2018-08-30.

[7] Berezovskyi, J. El-khoury e E. Fersman, “Linked Data Architecture for Plan Execution in Distributed CPS,” IEEE, Melbourne, Australia, 2019.

[8] Berezovskyi, R. Inam, J. El-khoury, M. Törngren e E. Fersman, “Efficient State Update Exchange in a CPS Environment for Linked Data-based Digital Twins,” em International Conference on Industrial Informatics, INDIN’19, Aalto, 2019.

[9] M. Fox e D. Long, “PDDL 2.1: An Extension to PDDL for Expressing Temporal Planning Domains,” Journal of Artificial Intelligence Research , 2003.

[10] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler e A. Y. Ng, ROS: an open-source Robot Operating System, ICRA Workshop on Open Source Software, 2009.

[11] Siciliano e O. Khatib, “Springer Handbook of Robotics,” Springer-Verlag Berlin Heidelberg, 2008.

[12] O. Brock e O. Khatib, “High-speed navigation using the global dynamic window approach,” IEEE, 1999.

[13] R. Inam, K. Raizer, A. Hata, R. Souza, E. Fersman, E. Cao e S. Wang, “Risk Assessment for Human-Robot Collaboration in an automated warehouse scenario,” IEEE, Turin, Italy, 2018.

[14] ISO, “ISO 12100:2010 Safety of machinery – General principles for design – Risk assessment and risk reduction,” International Organization for Standardization, Geneva, Switzerland, November 2010.

[15] ISO, “ISO 10218-1 (2011): Robots and robotic devices - Safety requirements for industrial robots - Part 1: Robots,” International Organization for Standardization, Geneva, Switzerland, July 2011.

[16] ISO, “ISO 10218-2 (2011): Robots and robotic devices - Safety requirements for industrial robots - Part 2: Robot systems and integration,” International Organization for Standardization, Geneva, Switzerland, July 2011.

[17] ISO, “ISO/TS 15066:2016 Robots and robotic devices – Collaborative Robots,” International Organization for Standardization, Geneva, Switzerland, February 2016.

[18] K. He, G. Gkioxari, P. Dollár e R. Girshick, “Mask R-CNN,” arXiv, 2017.

[19] M. Ying Yang, W. Liao, H. Ackermann e B. Rosenhahn, “On support relations and semantic scene graphs,” Elsevier, 2017.

[20] S. Richard e A. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.

[21] S. Sreedharan, S. Srivastava, D. Smith e S. & Kambhampati, “Why can't you do that Hal? Explaining unsolvability of planning tasks,” In S. Kraus (Ed.), Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019 (pp. 1422-1430), 2019.

[22] M. Cashmore, A. Collins, B. Krarup, S. Krivic, D. Magazzeni e D. E. Smith, “Towards Explainable AI Planning as a Service,” CoRR abs/1908.05059 (2019)..

[23] F. Cassez, A. David, E. Fleury, K. Larsen e D. Lime, “Efficient On-the-Fly Algorithms for the Analysis of Timed Games,” In: Abadi M., de Alfaro L. (eds) CONCUR 2005 – Concurrency Theory. CONCUR 2005. Lecture Notes in Computer Science, vol 3653. Springer, Berlin, Heidelberg.

[24] S. Charan Syba, M. Babu Jayaraman, S. Kumar Mohalik e B. Ramamurthy, “Framework and tool support to design high quality domain models for intelligent systems,” IEEE, Bangalore, India, 2017.

[25] SCOTT Deliverable D1.3 “First Evaluation of Technical Progress & Assessment of Project Objectives”, v1.0, 2018-04-20.

[26] SCOTT Deliverable D27.1 “Project Handbook”, v1.0, 2017-08-10.

[27] “MoveIt! Frequently Asked Questions,” [Online]. Available: https://moveit.ros.org/documentation/faqs/.

[28] “ROS Control Wiki,” [Online]. Available: https://github.com/ros-controls/ros_control/wiki.

[29] “MoveIt! Concepts,” [Online]. Available: https://moveit.ros.org/documentation/concepts/.

[30] Deliverable D10.2 "Sandbox (alpha): Initial implementation of the Sandbox and its evaluation and deployment", v1.0, 2018-08-24.

A. ABBREVIATIONS AND DEFINITIONS

Term	Definition
AIOTI	Alliance for the Internet of Things Innovation
API	Application Programming Interface
BB	SCOTT Building Block
BNF	Backus Naur Form
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPS	Cyber-Physical Systems
CRUD	Create, Read, Update, Delete (operations)
DevOps	compound of "Development" and "Operations"
FLS	Fuzzy Logic System
HLA	High-level architecture
HRC	Human-Robot Collaboration
HTTP	Hypertext Transfer Protocol
KPI	Key Performance Indicator
LDN	Linked Data Notifications
LSTM	Long Short Term Memory
mAP	Mean Average Precision
MQTT	Message Queuing Telemetry Transport
NLP	Natural Language Processing
OSLC	Open Services for Lifecycle Collaboration
PDDL	Planning Domain Definition Language
POJO	Plain Old Java Object
R-CNN	Region-based Convolutional Neural Network
RDF	Resource Description Framework
REST	Representational State Transfer
RL	Reinforcement Learning
ROS	Robot Operating System
SDK	Software Development Kit
SEF	State Event Filtering
TBB	Technical Building Block
TRS	OSLC Tracked Resource Set
V&V	Verification and Validation
V-REP	Virtual Robot Experimentation Platform
WHC	Warehouse Controller