SCOTT:
Secure COnnected Trustable Things

Sandbox (Alpha)

Document Type	Deliverable (Demo)
Document Number	D10.2
Primary Author(s)	Andrii Berezovskyi | KTH Alberto Hata | EAB Swarup Mohalik | EAB Leonid Mokrushin | EAB
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	2018-07-11

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
Andrii Berezovskyi	KTH	Leonid Mokrushin	EAB
Alberto Hata	EAB	Ricardo Souza	EAB
Rafia Inam	EAB	Aneta Vulgarakis Feljan	EAB
Swarup Mohalik	EAB	Konstantinos Vandikas	EAB

FORMAL REVIEWERS

Name	Organization	Date
Jose Manuel Sanches	Integrasys SA	2018-08-16
Łukasz Szczygielski	GUT	2018-07-10

DOCUMENT HISTORY

Revision	Date	Author / Organization	Description
0.1	2018-06-13	Leonid Mokrushin / EAB	Planner Service, Planning Domain, and CI/CD DoW
		Konstantinos Vandikas / EAB	Object Detection DoW
		Swarup Mohalik, Aneta Vulgarakis Feljan / EAB	DoW f/ Estimator, demo scenario, Mission domain.
		Alberto Hata, Rafia Inam, Ricardo Souza / EAB	DoW f/ ROS messages, ROS node, V-REP simulation,
0.2	2018-06-13	Andrii Berezovskyi / KTH	DoW for the UC architecture, twin components and the information model. Sent for an internal review.
0.3	2018-06-20	Andrii Berezovskyi / KTH	Sent for a formal review.
1.0	2018-08-24	Andrii Berezovskyi / KTH	Resolve issues pointed out by both reviewers in the following sections: Objectives, DoW, TL link.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 EXECUTIVE SUMMARY

The D10.2 deliverable describes the first Alpha development iteration of the Warehouse Sandbox, following its corresponding Alpha Specification detailed in the deliverable D10.1 “Sandbox Specification - A Sandbox for Intelligent Warehouse Logistics” [1]. Changes to the Specification after the Alpha iteration will be detailed in the upcoming deliverable D10.3 for the Beta version of the Sandbox Specification [3].

This deliverable reports the development of the Alpha version of the Warehouse Sandbox demonstrator, which is illustrated in the Section 3.2 below. The architecture, which was initially designed in D10.1 [1] and developed during year 1, is presented in detail in Section 3.3 and its components are described in the subsections of the Section 3.5. The architecture is built on principles of linked data: 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. The information model is described in Section 3.4.

Out of 9 demo requirements, 4 were considered for the Alpha iteration of the Warehouse Sandbox demo prototype. As of March 2018, the WP10 has a progress score of 15%. Averaging the self-assessed scores for the demo requirements, this deliverable provides the score of 26% for WP10 progress.

Keywords: Warehouse logistics, AI planning, Digital Twins, ROS, Robotics, SCOTT Y1, Linked Data, OSLC

2 OBJECTIVES

2.1 Deliverable Objectives

The objective of this deliverable is to provide a description of the system developed as part of the Work Package (WP) 10 for warehouse logistics and evaluate its correspondence to the demo requirements laid out in Year 1. In particular, the following DemoRQ requirements are addressed:

- DemoRQ 640 Monitoring and control
- DemoRQ 641 Digital Twins
- DemoRQ 643 Pubsub exchange
- DemoRQ 647 Warehouse instrumentation
- DemoRQ 648 Robot and robot software

The following DemoRQ requirements are *not addressed* in the current deliverable:

- DemoRQ 639 Operation Optimization
- DemoRQ 650 Testing framework for PDDL domains
- DemoRQ 681 Safe operations
- DemoRQ 682 Collaborative Operations

2.2 Work Package Objectives

From the SCOTT project description [25], the main objectives for WP10 include:

1. Utilization of open-source IoT platform 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.

The deliverable addresses these objectives in the following way:

1. All code developed as part of the work package is open-source, and depends on the open-sources platforms and libraries (covered further in Sections 3.4, 3.5, 4).
2. The information model to perform planning over heterogeneous devices, presented in Section 3.3, is an important step to modelling device capabilities and scheduling real plans across them.
3. Safety-critical aspects are not the main focus of this deliverable, though work has been done on safety-related trust issues (see the Section 2.3 below regarding the work on the trust framework, specifically to cover safety-related concerns).

4. Work towards preventing abnormal behaviour is covered in Section 3.4.5 through a development of a Plan Estimator, which prevents unsound Plans from being scheduled. Runtime anomaly detection is not part of this deliverable.
5. Visual Analytics work is not part of this deliverable.

2.3 Overall SCOTT Objectives

In regard to the overall SCOTT objectives [24], this deliverable supports them in the following ways.

Focus on wireless systems.
The “things” (the T in the Internet of Things) in WP10 rely on wireless transmission. The safety features needed to ensure trustability do not depend on the real-time networking.

Focus on European leadership and market opportunities.
In order to explore market opportunities and European leadership, the presented warehouse system is continuously discussed with one of the Ericsson HW supply factories that is also interested in automating its assembly and warehouse system. Such an approach allows for simultaneous SCOTT objective achievement and realistic system development.

Focus on smart sensors and actuators.
The work package makes use of Digital Twins for all cyber-physical systems used.

Focus on Security, Safety, Privacy and Trustability.
The objective is addressed in the scenarios defined in the use-case. As part of the work on the Year 1 demo (alpha sandbox), these scenarios were defined in D10.1 [1] and further work was done within the scope of the D28.3 [15] on trust-related aspects.

Focus on including psychological and socio-contextual enablers for trust formation.

The use-case participates in WP28 to discover and define how to address specific trust-related issues within Warehouse Sandbox.

Focus on eco-system with well-defined re-usable Technical Building Blocks.
The alpha sandbox follows the Technical Building Blocks (TBB) mapping defined in the D10.1 [1].

Focus on solutions to be used in multiple industrial domains.
Interoperability issues are at the core of the work done in this WP. Furthermore, many details of the architecture, such as the use of adaptors to connect to the existing (legacy) systems and the use of multi-level ontology development are fully transferrable to other domains. Finally, general-purpose programming technologies are used, which are widely adopted throughout many industries.

Focus on higher Technology Readiness Levels (TRLs).
The components of the WP10 sandbox environment are targeting TRL levels 5-7.

3 DESCRIPTION OF WORK

In Section 3, we describe the technical details of the Alpha version of Warehouse Sandbox built in the first iteration. First, we begin by preseting a demo scenario for the Sandbox in Section 3.1. Then, a general architecture of the Sandbox is presented in Section 3.2 and the details of its components are presented in the Section 3.4. In Section 3.3, the information model of the Sandbox is described. Finally, the Section 3.5 covers the Development & Operations (DevOps) aspects of the Sandbox.

3.1 Work Overview

As of July 2018, the following work is being done:

- Ericsson and KTH actively work on implementing the technical foundation of the Warehouse Sandbox to cover the requirements listed in the Section 2 of this deliverable.
- HH has shifted PM towards other WPs and is not contributing to the development of the Warehouse Sandbox demo.
- RTE has shifted PM towards the building blocks connected to WP10 and is not contributing directly to the Warehouse Sandbox demo.

3.2 Sandbox Alpha Demo Scenario

The scenario for the demo is intended to provide an overview for the sandbox (alpha version), in general, and various components that comprise it, in particular.

The setting is an area of a warehouse modelled as a V-REP [19] scene. The warehouse is virtually split up into smaller sections, which are identified by the waypoints, e.g. ‘w0’, ‘w1’, ‘w4’ etc. Within the warehouse area, there are 3 *shelves*, identified as ‘shelf1’, ‘shelf2’ and ‘shelf3’; there are 3 *conveyor belts*, indentified as ‘cbelt1’, ‘cbelt2’, ‘cbelt3’; finally, there are two *robots* ‘robot1’ and ‘robot2’. The goods are stored on the shelves and are labelled as ‘obj1’, ‘obj2’ and so on.

In order to perform some action within the warehouse, the Warehouse Manager visits the Warehouse Controller (WHC) graphical user interface (GUI) and inputs a mission. The mission is specified as a list of object labels and their destinations, e.g. [['obj1', 'cbelt1'], ['obj2', 'cbelt1'], ['obj3', 'cbelt2']]. In D10.3, a new Natural Language Processing (NLP) component will be provided to allow warehouse manager to define a mission using a controlled natural language.

The Actions within the warehouse are not defined directly, but synthesized by a Planner through the definition of the Problems. A Problem definition consists of an Initial state and a Goal state. The Goal state is derived from the mission specification. In order to construct the Initial state, the WHC gathers the state of all the warehouse (position of shelves, conveyor belts) and robots (position, busy or available) in the system by making requests to their digital twins, which act as an interface to the actual robots for the rest of the components in the system and vice versa. Once the up-to-date information about the rest of the system is assembled, the WHC formulates a planning request consisting of the Problem definition and a Domain definition (the Actions of robots). Based on the difference between the Initial state of the warehouse and the Goal state, the Planner computes a plan which is an ordered sequence of Actions. The warehouse state would reach the Goal state once the Actions given in the plan are executed.

Before distributing this plan for the execution, the WHC passes it through an Estimator, where the plan is analysed for safety. A successfully analysed plan is then distributed among the digital twins,

where the plan is reduced to a subset of Actions that is only relevant for the particular robot (device) that this twin manages.

The twins use a connectivity mechanism specific to the device at hand to either pass a custom plan containing all of the relevant Actions (in case of a highly capable device) or control the execution on an Action-by-Action basis (for low-end embedded nodes without onboard planning capability). In this demo, the twins connect to the robots via a ROS Service mechanism and call the service for triggering a new plan execution, which is processed by the ROS Nodes of the robots.

The execution of the plan results in the robots picking up the requested objects, traversing the warehouse through paths generated by low-level ROS planners at the nodes according the high-level plan and dropping the objects on the output conveyor belts. This is how the robots fulfil the specified mission.

After the execution of every Action, robot twins report the device status back to the WHC, where it can display the progress to the Warehouse Manager and to trigger a new planning request, if necessary.

In the following, we list the scope of the alpha demo.

1. A mission is planned and executed completely before taking up the subsequent mission. In the later phases, this will be relaxed and planning and execution phases will be intermixed to optimize time and resources.
2. We assume that the missions are only of the type "pick-up at shelf, then drop on conveyor belt". This implies that there is no dependency of a pick-up action on an earlier drop action. This assumption prevents some deadlock scenarios that can arise out of real life timing delays. In the later phases, we will handle general missions and handle the deadlock scenarios.
3. Waypoints are assumed to be extended spaces where more than one robot can reside. In later models of the warehouse, we will explicitly model occupancy at waypoints which will model the warehouse more closely (there may be less space at a waypoint e.g. limited number of robots can occupy a charging station).
4. Robots will follow the local strategic plans, as provided by warehouse controller. When robots detect a conflict, the resolution is done by stopping (or, slowing), given the task priority and status. For example, when a robot arrives to an already occupied waypoint it slows down or stops, and waits for its turn. In the current scope, the priority is fixed a priori. Later the priority will be made to depend upon the task deadline, and unforseen circumstances will be dealt locally by the robot's decision-making process.
5. In case of a hazard, complete planning is done starting with the current state for all the robots. Optimal replanning methods will be implemented in the later phases.

In the next sections, we describe the system components involved in the scenario presented above.

3.3 Use Case Architecture

Related building blocks	BB26.B_JIG (Cloud computing service platform) BB24.C_INTEGRA (Application layer protocols and cloud architectures)

Related requirements	RQ 651	Linked Data Platform (BB24.C)
	RQ 641	Digital Twins (Demo)
	RQ 648	Robot & Robot Software (Demo)
Implementation scope of the TBBs in the component	The building block BB26.B_JIG is focused on the core set of software components needed for the development of SCOTT services in the cloud. The Linked Data Platform architecture developed as part of the WP10 is designed to rely on the common software components in order to align with the output of BB26.B_JIG. The Linked Data Platform architecture itself and its application level protocols is being developed for the BB24.C_INTEGRA. In turn, the output from the BB24.C_INTEGRA is being incorporated in the archecture by influencing the choice of the application level protocols, in particular.	

Table 1. TBB and requirement mapping for the UC architecture.

The Warehouse Sandbox architecture was developed in order to satisfy the requirements its implementation would have to fulfill, but also to satisfy certain specific requirements itself, such as:

- RQ 651 (BB24.C) – Linked Data Platform
- RQ 641 (Demo) – Digital Twins
- RQ 648 (Demo) – Robot & Robot Software

At the high level, the architecture spans multiple layers, as mapped onto the AIOTI architecture. The mapping can be seen in the following figure (Fig. 1).

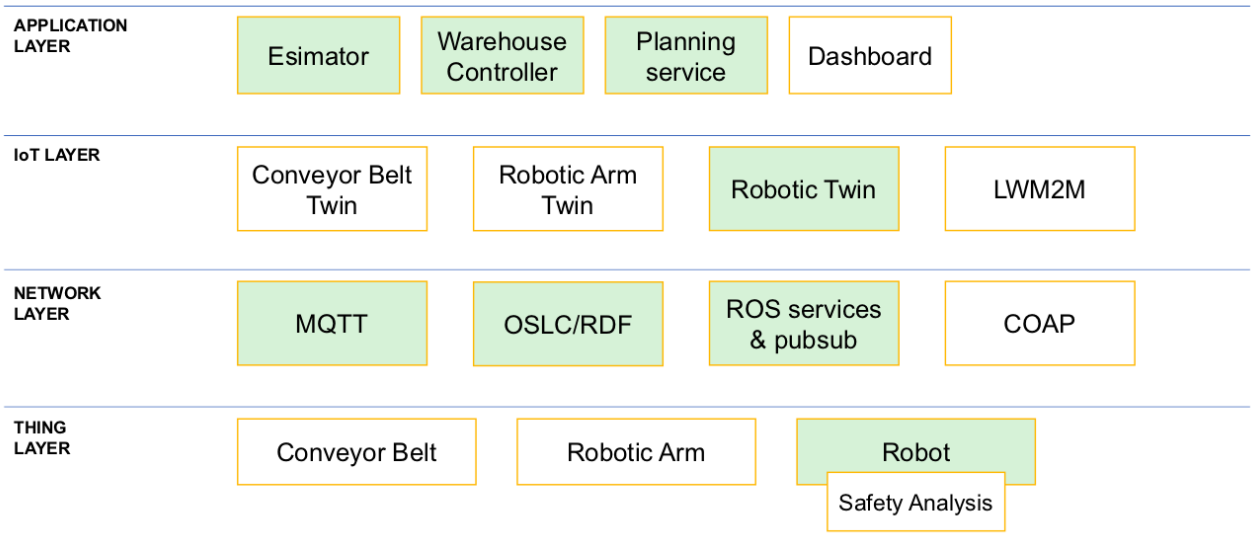

Figure 1. Warehouse sandbox architecture levels as mapped onto the AIOTI architecture.

Green colour denotes the architectural elements that are part of the D10.2 demo scope.

At the Thing Layer, the Warehouse Sandbox features robots, robotic arms, and conveyor belts, which were described in the demo scenario in Section 3.1. *Things* within the architecture have a physical embodiment but also simulated counterparts. In the first iteration of the Sandbox, as described in this deliverable, only the simulated counterparts are taken into account. The simulation of *robots* in V-REP software will be detailed in the corresponding subsection of the Section 3.4 Sandbox Components.

At the Networking Layer, two different sets of choices have to coexist: a stack of technologies chosen for the sandbox architecture and the stack of technologies used by the cyber-physical (CPS) components. The CPS components are in many cases predefined by the equipment vendors in advance and are not managed in this use-case. The sandbox stack of networking technologies includes Hypertext Transfer Protocol (HTTP), Message Queuing Telemetry Transport (MQTT), Resource Description Framework (RDF), and Open Services for Lifecycle Collaboration (OSLC). The CPS stack of networking technologies includes Robot Operating System (ROS) Services, MQTT, ROS Messages.

The IoT Layer in the Sandbox architecture is represented by the use of Digital Twins (relates to the requirement 641 mentioned at the beginning of the section). The Twins allow the Linked Data Platform to be applied throughout the architecture without affecting the design and the external interfaces of the CPS components (Things). Their design would be detailed in Section 3.4.2 Digital Twins.

At the Application Layer, the services for controlling the warehouse operations are located. This year, the focus has been on specifying the detailed architecture, defining an information model (ontologies) and prototyping the planning aspects of the Application Layer.

The detailed architecture is depicted in Figure 2, which is adopted from the architecture presented in the D10.1 Sandbox Specification [1]. The updated figure uses light green background to denote the elements of the architecture which are described as part of this deliverable. For these elements, an initial development was performed during the Alpha iteration to establish the feasibility of the architectural choices for the Sanbox Specification and provide the foundation for the rest of the use case development.

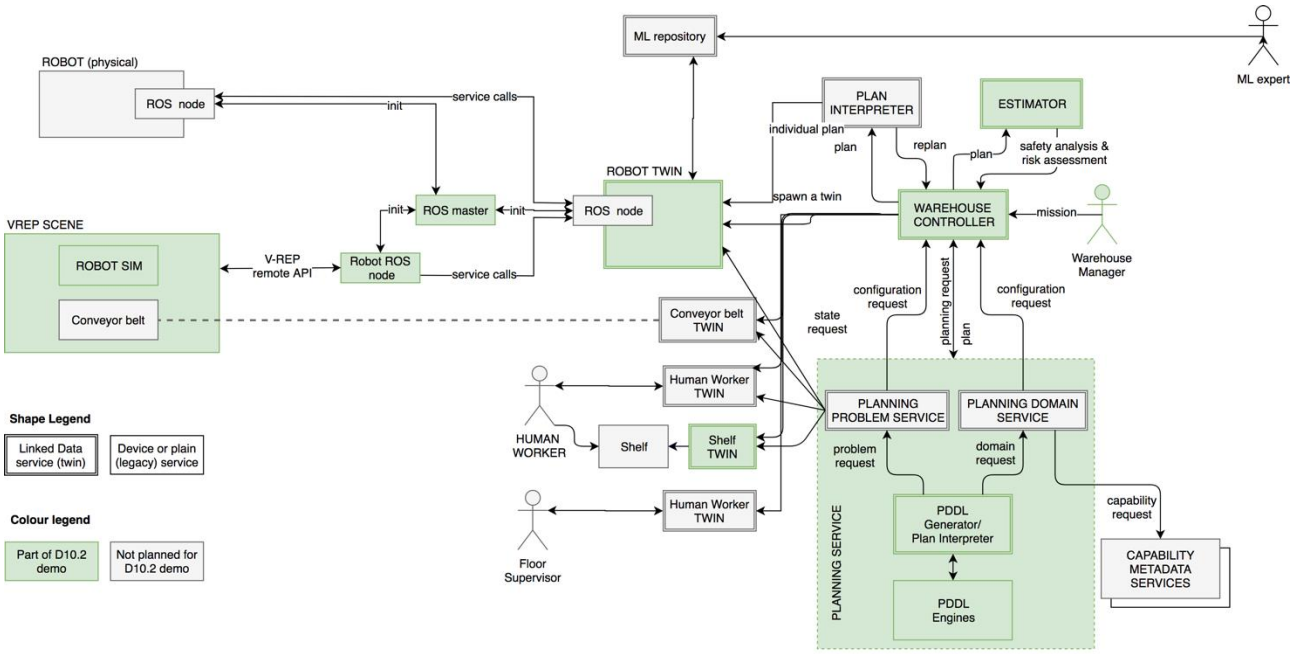

Figure 2. Warehouse sandbox architecture description.

Green elements are part of the demo scenario in the D10.2 deliverable and the elements marked in grey are going to be implemented in the next iterations.

All components in the sandbox architecture, except for the legacy or embedded ones, are implemented as RESTful web services that follow the Open Services for Lifecycle Collaboration (OSLC) specifications [16]. OSLC is a set of specifications that allow the Linked Data to be used efficiently in the enterprise application contexts. Use of Linked Data within the use-case allows to ensure that the data exchanged by all of the components in its architecture has a uniform and machine-readable semantic definition of the information model in form of RDF ontologies. Linked Data also allows to uniquely and uniformly identify all resources in the systems through URLs (URIs, more precisely). Finally, by agreeing to use the HTTP as a primary protocol for the URIs allows to achieve uniform *access* of each OSLC resource in the system.

In order to make RDF ontologies usable in the enterprise context, OSLC restricts the use of RDF(S) predicates that require either computationally intensive or potentially error prone (due to a modelling mistake) inferencing and add the notion of *shapes.* OSLC shapes together with limited RDF(S) ontologies for a single domain (technically, a common RDF namespace) constitute an OSLC Domain. OSLC specifications predefine a set of common domains, but enterprise users are encouraged to develop their own domain-specific OSLC domains. In this work package, we work with ontologies covering the following domains:

- a planning domain
- a robotic domain
- a warehouse (intralogistics) domain
- a monitoring domain

Section 3.3 details the work on the information model.

Application services and device twins are being developed using a model-based approach. The Eclipse Lyo project [17] provides a framework for the development of the OSLC compliant servers and clients, as well as a graphical engineering environment for model-based engineering of

systems built on top of OSLC. The service implementations were partially generated automatically using the Lyo Generator from the descriptions modelled using Lyo Designer. The generated code represents RESTful microservices implemented in Java based on the JAX-RS specification.

As part of work on the Sandbox Alpha, a dedicated library oslc_prolog has been developed [4] to support the development of OSLC compliant services in Prolog programming language. The library is implemented using SWI-Prolog's internal semantic web library [5] for RDF resource manipulation, and Cliopatria application server [6] for OSLC service hosting and ontology manipulation/visualization purposes. The native language ability to manipulate RDF resources, store them in the efficient internal triple store, and perform various types of inferences make Prolog a perfect environment to develop OSLC services. The oslc_prolog library provides convenient methods for resource Create, Read, Update, Delete (CRUD) operations, automatically exposes all created OSLC resources via HTTP API, and automatically performs compliance checks w.r.t. defined resource shapes. In Section 3.4.5, we describe the Planning Service realized as a set of adapters implemented in Prolog with the help of the oslc_prolog library.

To provide lower-level communication between Robots and their respective Digital Twins the Robot Operating Systems (ROS) is employed. ROS has six main components:

- ROS Node – an executable that runs a program written with ROS libraries.
- ROS Topic – a communication interface that a ROS Node uses to publish/subscribe information (message) to/from other ROS Nodes. This communication is performed in a many-to-many approach.
- ROS Service – a communication interface that a ROS Node uses to request/response information (message) to/from other ROS Nodes. This communication is performed in an one-to-one approach.
- ROS Message Types – data formats used to exchange information between ROS Nodes when used publish/subscribe (topic) communication.
- ROS Services Types – data formats used to exchange information between ROS Nodes when used request/response (service) communication.
- ROS Master – the mechanism used for resorce discovery between nodes. Once the metadata of two nodes are obtained, a peer-to-peer communication is performed.

ROS supports both publish-subscribe and request-response communication paradigm, where the former is the default model of choice. In the publish-subscribe model, nodes subscribe to desired information topics while other nodes send data under specific topic names as in any typical pub-sub application. For the request-response paradigm, ROS has the concept of a service, which is defined by a pair of respective messages and by a well-known name, that serves as the resource URL.

The V-REP simulator is used to reproduce the warehouse scenario, the robot behavior and other elements that exist in this environment (e.g. trucks, workers)

3.4 Information Model Development

Related building blocks	BB24.C_INTEGRA (Application layer protocols and cloud architectures)
Related requirements	REQ 644 Common Knowledge Representation

Implementation scope of the TBBs in the component	Robots, their digital twins, and other components need a way to represent the knowledge as well as communicate it to the other components in a flexible way. The semantic information model defined in this section serves the described purpose.

Table 2. TBB and requirement mapping for the warehouse controller component.

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) language and its concepts. Using the concepts from this ontology, all kinds of plans would be formulated for the rest of the system.

To allow real Actions to be planned, though, these Actions must be 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 withing 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).

3.4.1 Planning Domain

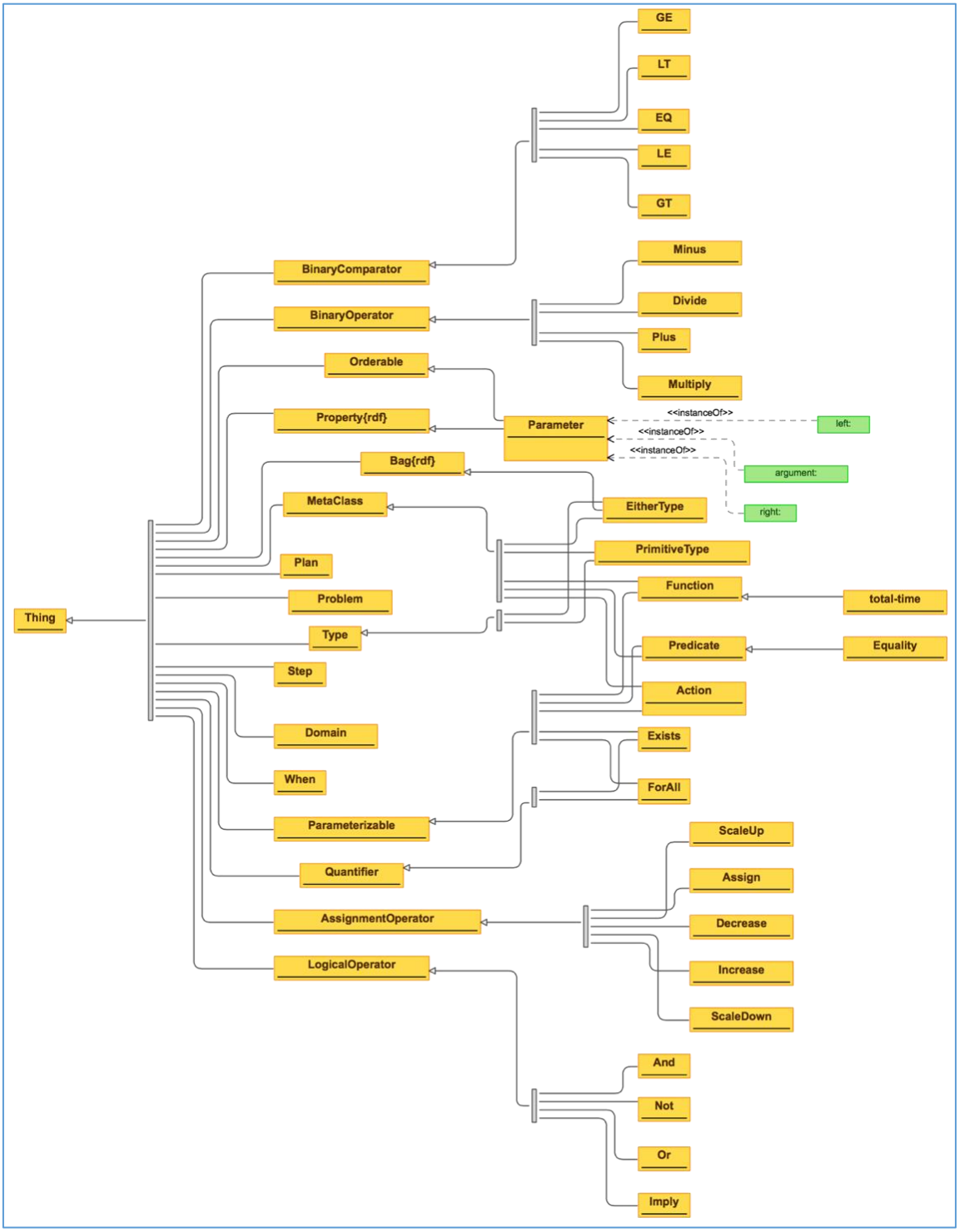

Figure 3. PDDL-based planning ontology.

The visualisation was rendered using OWLGrEd.

The ontology for the PDDL domains is shown in Figure 3. It was modelled closely after the concepts described by the Backus Naur form (BNF) in PDDL 2.1 [7].

In order to be able to use it in the model-based environment, however, it had to be imported. The first version of an automated importer for generic RDF(S) ontologies for Eclipse Lyo Designer was developed as part of the deliverable [8].

The imported model was then used to define the interfaces of the services (detailed in the Section 0 below). In addition to that, it was used to generate Plain Old Java Object (POJO) class definitions. This allows POJO instances to be manipulated programmaticaly 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.

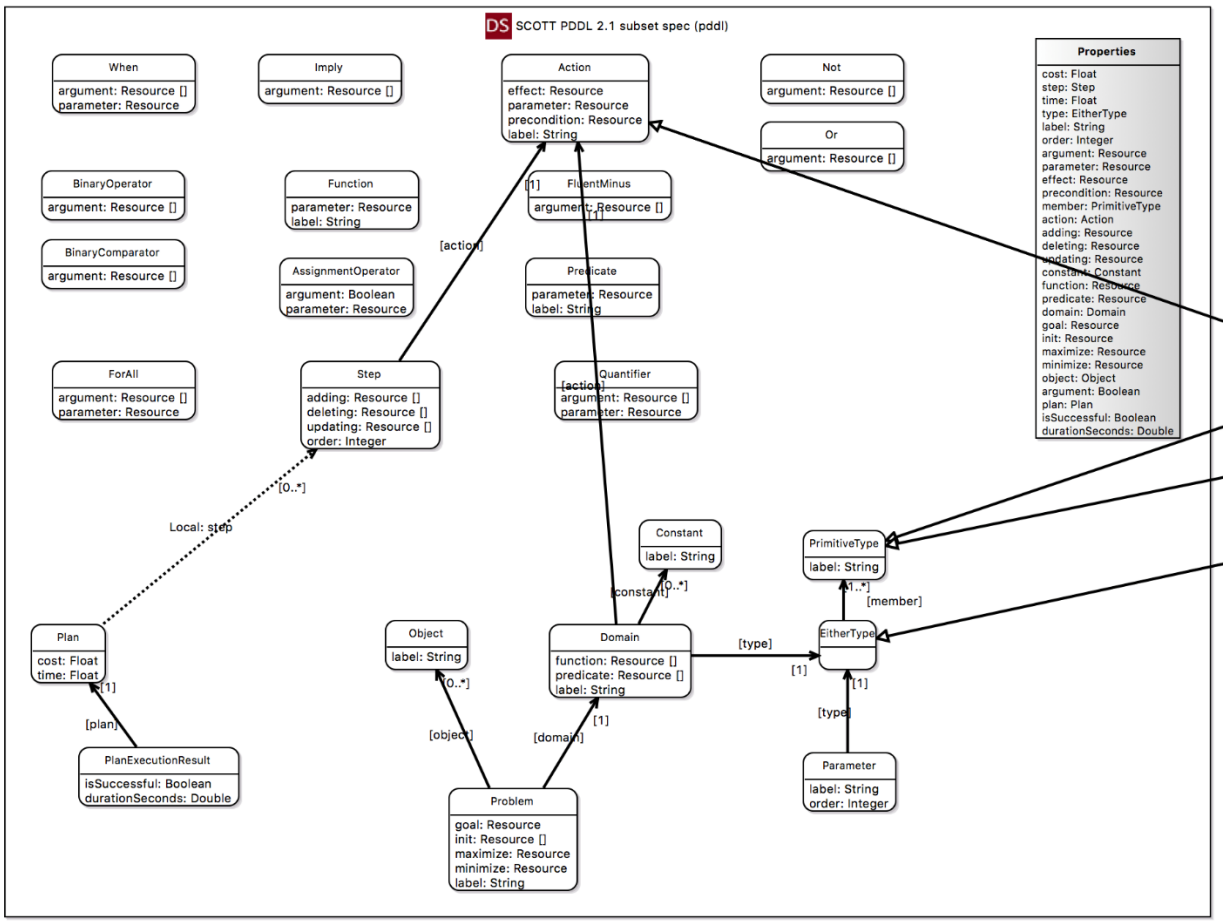

Figure 4. Imported subset of the planning ontology in Lyo designer.

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

3.4.2 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

1. Pick a specific object O1 and deliver at conveyor belt 1.
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 5.

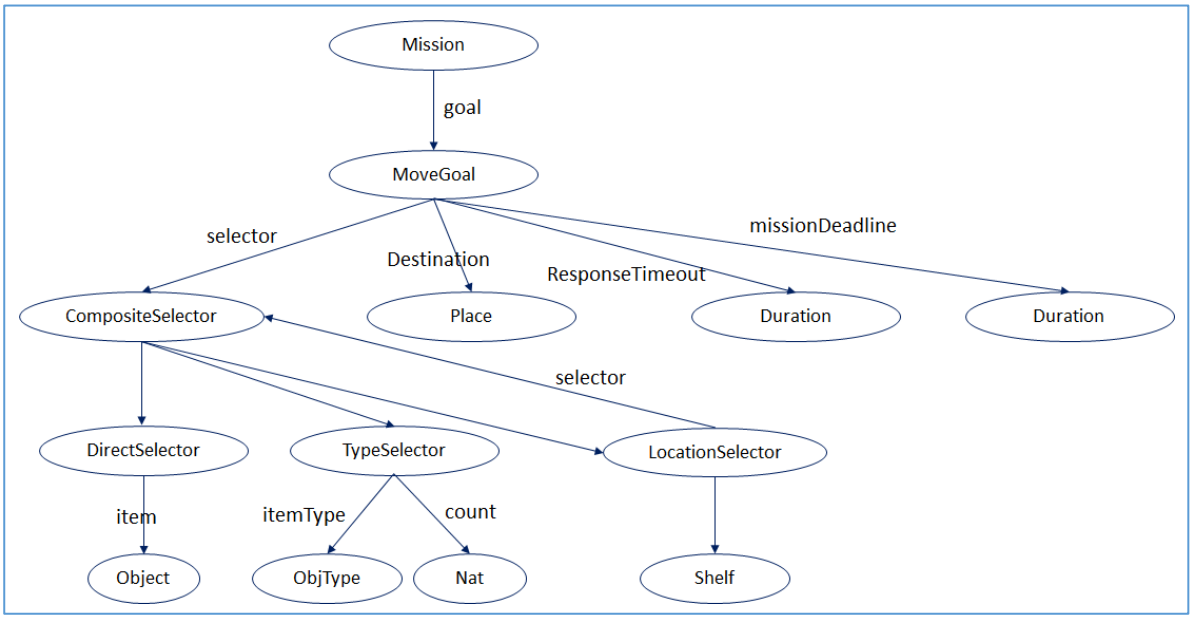

Figure 5. Pictorial representation of the mission ontology.

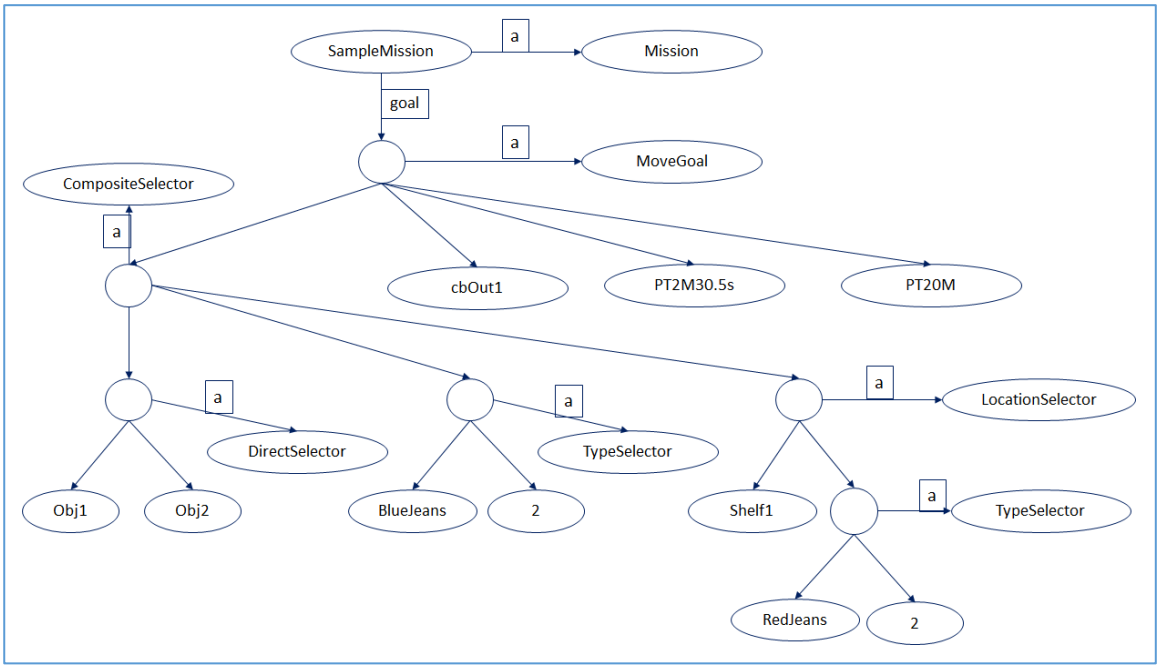

Figure 6. A sample mission and the relation of its instances to the concepts of the underlying ontology.

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.

```
: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
                ]
        ]
] .
```

3.4.3 Robotic Domain

During the first iteration of the sandbox, a simple domain of Blocks World was used. Blocks world is considered to be one of the most widespread planning domains in artificial intelligence research [9].

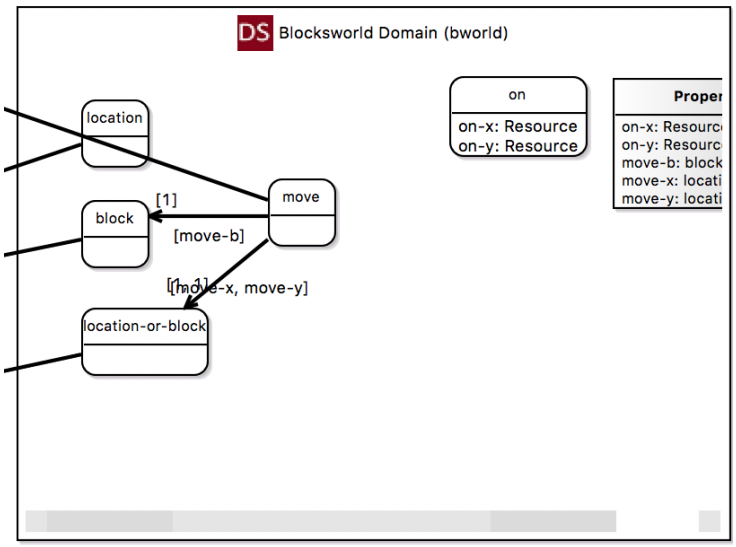

Figure 7. Block world domain modelled in eclipse Lyo as a aomain specification.

The next iteration of the sandbox will use a real-world ontology applicable for industrial robotics, such as IEEE 1872-2015 [2].

3.4.4 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
```

3.5 Sandbox Components

3.5.1 Planning-related Components

Related building blocks	BB24.D_ITI	
Related requirements	REQ 645	Strategic and reactive planning (BB24.D_ITI)
	REQ 646	Plan verification for safety (BB24.D_ITI)
Implementation scope of the TBBs in the component	In the building block BB24.D_ITI, WP10 works on the components that perform planning by analysing the data collected from the components of the warehouse and ensure their safety (hence, trustworthiness) through the subsequent plan verification.	

Table 3. TBB and requirement mapping for the planning-related components.

3.5.1.1 Warehouse Controller

As described in the demo scenario, the Warehouse Controller performs the coordinating function across the components in the use-case sandbox.

First, it allows a Warehouse Manager to input the Mission. In D10.1 [1], Ronald is the corresponding trustor from the operational scenario who assumes the role of a Warehouse Manager. In the Alpha version of the Warehouse Sandbox specification [1], the Mission input interface was not developed and required detailed input which is too technical for the Warehouse Manager. D10.3 [3] will detail a new component in the architecture that would allow the Warehouse Manager to formulate high-level plans (missions) with the Warehouse Controller using controlled natural language.

Second, it gathers the state of all *Things* through their Twins. The communication between the Twins and the Warehouse Controller is accomplished through the exchange of messages that carry OSLC Tracked Resource Set (TRS) change information over the MQTT topics. TRS is an OSLC specification that allows resource changes to be tracked through the exposure of the *change log* by the OSLC TRS Server, where such changes occur. Other systems, interested in these changes employ a TRS Client to receive said changes. Originally, the TRS specification requires TRS Clients to repeatedly poll the HTTP endpoint of the TRS Server for a list of most recent changes. For the warehouse sandbox implementation, we have extended the TRS specification to support the change event delivery over MQTT. Under a new environment, the TRS Clients that maintain a subscription to a given MQTT topic would receive each Change Event which is published on the said topic by a TRS Server. This has allowed us to eliminate excessive polling from the part of the system where operational data is expected to be exchanged at a high velocity.

Third, the Warehouse Controller formulates the request to the Planner service based on the input from the Warehouse Manager and the state gathered from the *Things.* Such request takes a form of a Planning Problem combined with the Planning Domain. The Problem Domain represents the capabilities of the *Things* and in year 1 is statically defined.

Finally, the Warehouse Controller passes the Plans obtained from the Planner service to the Twins for subsequent execution.

The Warehouse Controller service was designed using an model-based approach. The Eclipse Lyo Designer was used to model the service based on the information models described in the Section 3.4 above. The internal structure of the Warehouse Controller service is shown in Figure 8. Internal structure of the warehouse controller service. The structure does not have much complexity because some operations have been carried out over MQTT and are not reflected in the model.

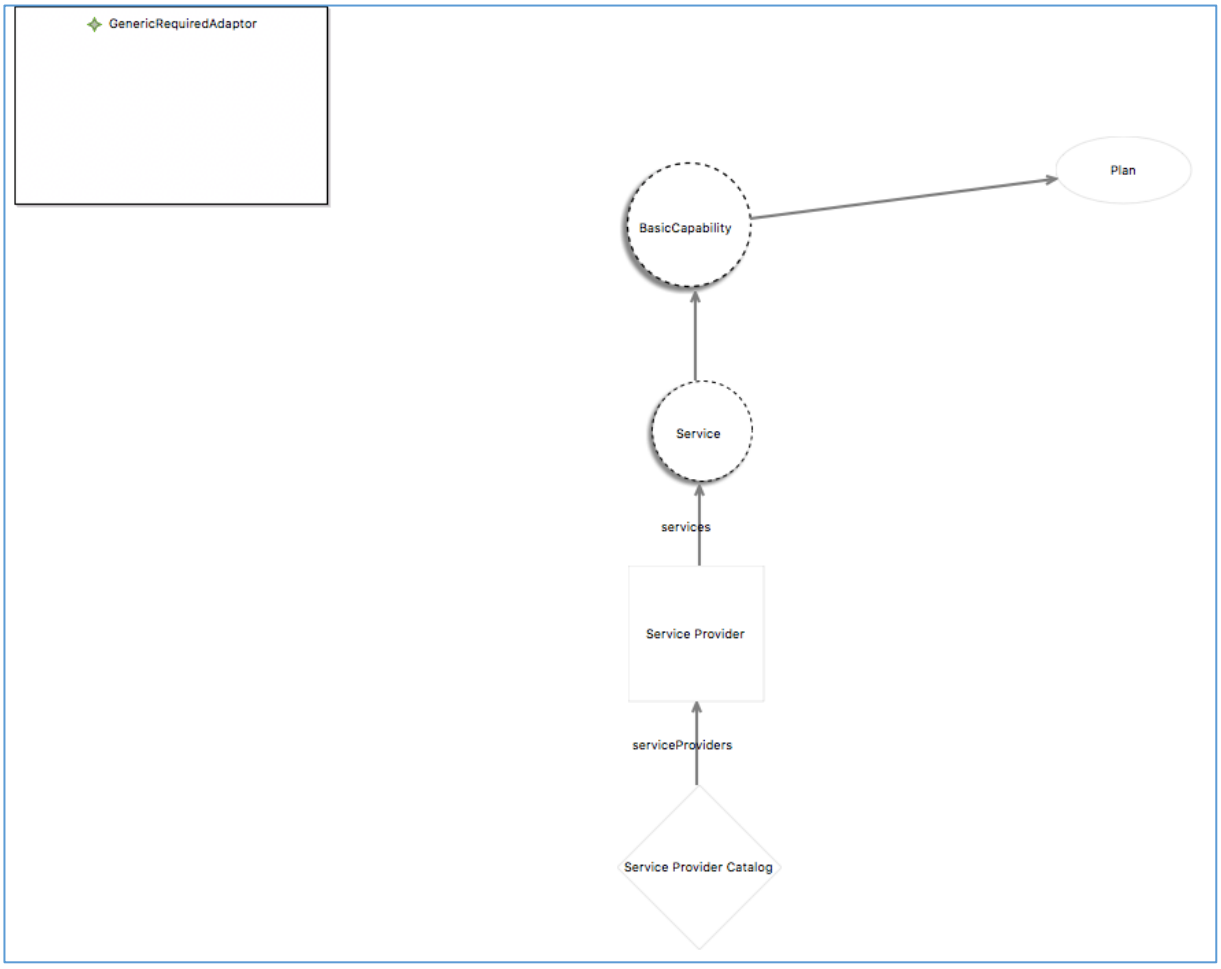

Figure 8. Internal structure of the warehouse controller service.

3.5.1.2 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 end-points (see the following Figure). Under the hood, planning service uses Metric-FF planner and VAL validator [21], [22].

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. We use this to annotate the output plans for online monitoring of plan execution that would be proposed in D10.3.

The `/pddl` end-point serves OSLC documents created according to PDDL ontology. The `/pddlCreationFactory` end-point 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` end-point 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.

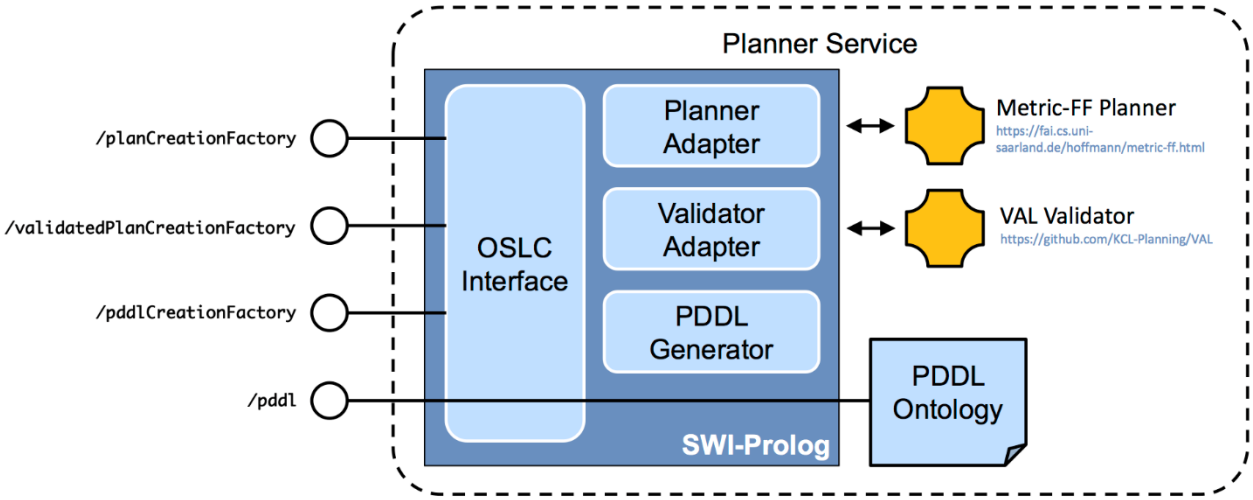

Figure 9. Components and interfaces of the planner service.

Planning process

The process and data flow within the planning service are shown in Figure 13. 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.

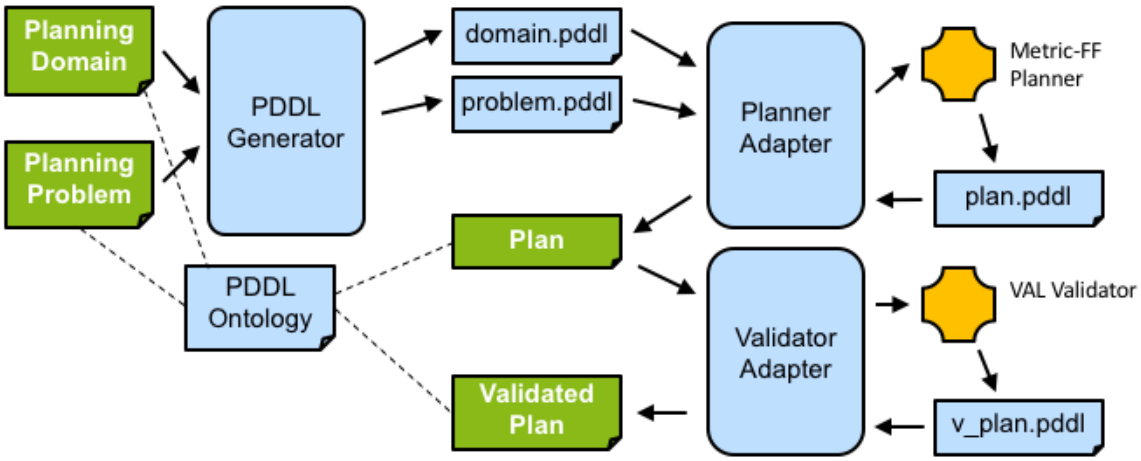

Figure 10. Plan synthesis process.

An example of PDDL generation can be seen in Figure 11. Output PDDL domain and problem from /pddlCreationFactory. The PDDL domain and problem were obtained through the REST interface endpoint `/pddlCreationFactory` of the Planner service.

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix oslc: <http://open-services.net/ns/core#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix pddl: <http://ontology.cf.ericsson.net/pddl/> .
@prefix : <http://ontology.cf.ericsson.net/pddl_example/> .

# --- Planning Domain
:warehouseDomain
  a pddl:Domain ;
  oslc:instanceShape pddl:DomainShape ;
  rdfs:label "warehouseDomain" ;
  pddl:type :Robot ,
            :Place ,
            :Waypoint ,
            :Object,
            :ObjType,
            :Charge;
```

Body Cookies Headers (3) Test Results Status: 200 OK

Pretty Raw Preview Text

```
(define (domain warehouseDomain)
  (:requirements :typing :equality :fluents)
  (:types Robot Place Waypoint Object ObjType Charge)
  (:predicates
    (on ?x - Robot ?y - Waypoint)
    (situated-at ?x - Place ?y - Waypoint)
    (is-on ?x - Object ?y - Place)
    (is-origin-on ?x - Object ?y - Place)
    (is-type ?x - Object ?y - ObjType)
    (carrying ?x - Robot ?y - Object)
```

Figure 11. 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 3.4.2 above (See Figure 15). 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 16).

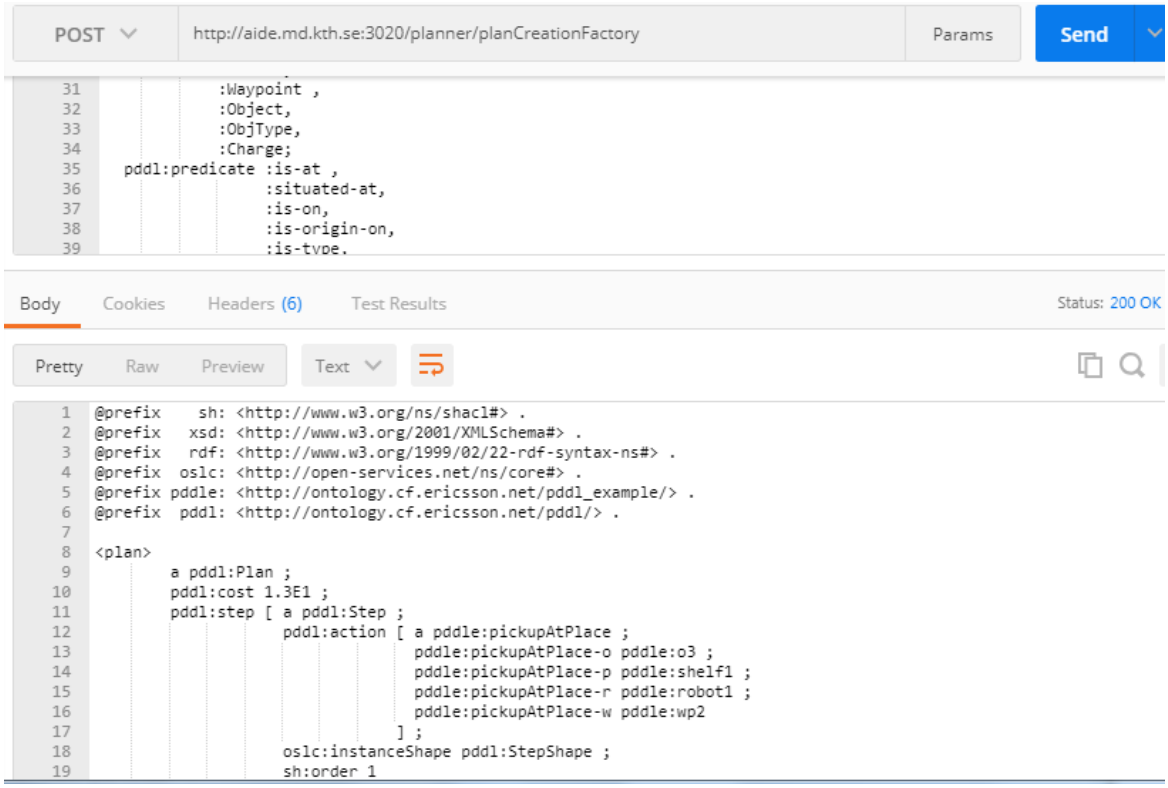

Figure 12. Output plan from /PlanCreationFactory.

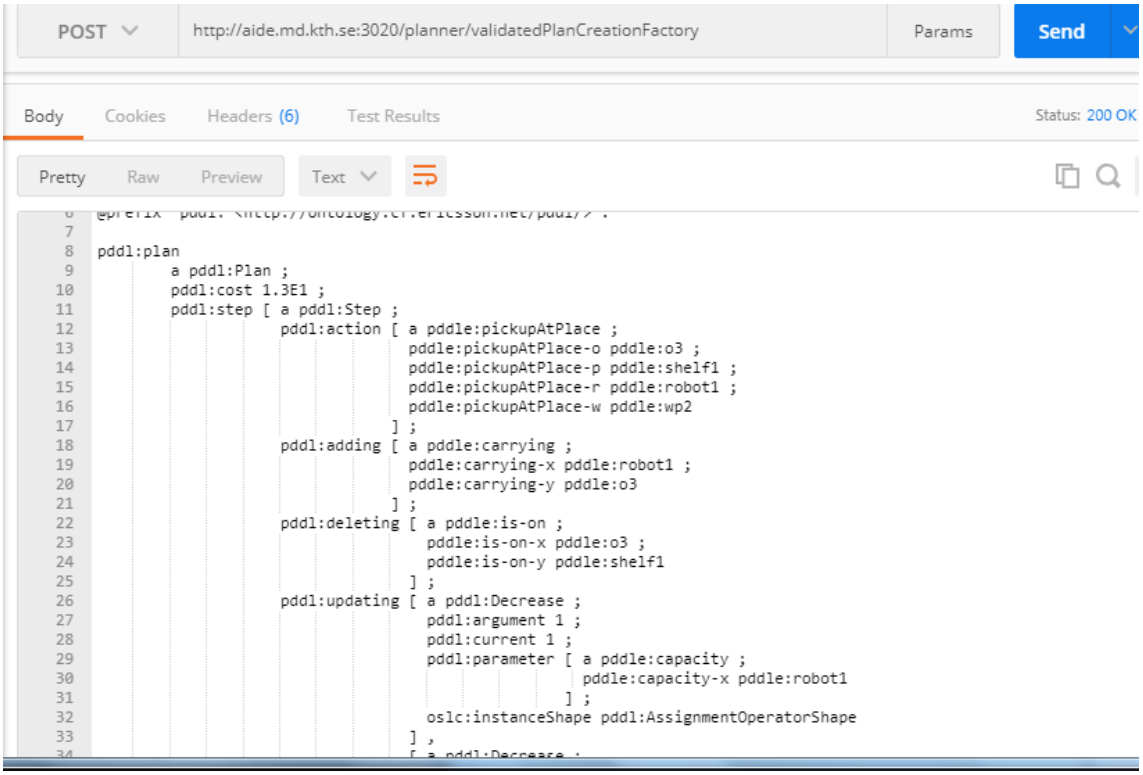

Figure 13. Output annotated plan from /ValidatedPlanCreationFactory.

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
```

3.5.1.3 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. Safety value computation is planned in D10.3.

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. The metric for the safety level of the plan will be added in the next iteration **following the work on the D28.3** [15]. It is expected that the warehouse manager takes a decision based on the completion-time and safety-level on whether to request for replan or not.

3.5.2 Digital Twins

Related building blocks	BB24.C_INTEGRA (Application layer protocols and cloud architectures)
Related requirements	REQ 644 Common Knowledge Representation REQ 651 Linked Data Platform
Implementation scope of the TBBs in the component	Digital Twins constitute a basic building element[1] of the Linked Data Platform. The Twins allow physical devices, such as robots, to be added to the system even if they rely on a protocol that is not investigated as part of the SCOTT project. The Twins are also designed to enable the implementation of the protocols investigated in BB24.C_INTEGRA for the communication with the rest of the system. Additionally, Twins allow for the model transformation of the data between the cloud system and the physical devices, enabling the use of a common knowledge representation in the cloud.

Table 4. TBB and requirement mapping for the digital twin component.

A digital twin is a software representation of a physical asset. On one hand, 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

[1] To avoid confusion with the term ‘building block’

of the rest of the system with high degree of accuracy and interface authenticity without involving physical assets.

In year 1, an initial prototype of a twin was developed to perform LWM2M communication with the ROS 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 3.4.4 for an example). Each twin has a corresponding ROS Node in order to communicate with the ROS Node of the corresponding *Thing*.

On the application layer, the twins implement OSLC interfaces and exchange notifications about change events to the tracked resources over MQTT, as detailed in the previous section about the Warehouse Controller. The Robot Twin (REST) service structure is shown in Figure 14.

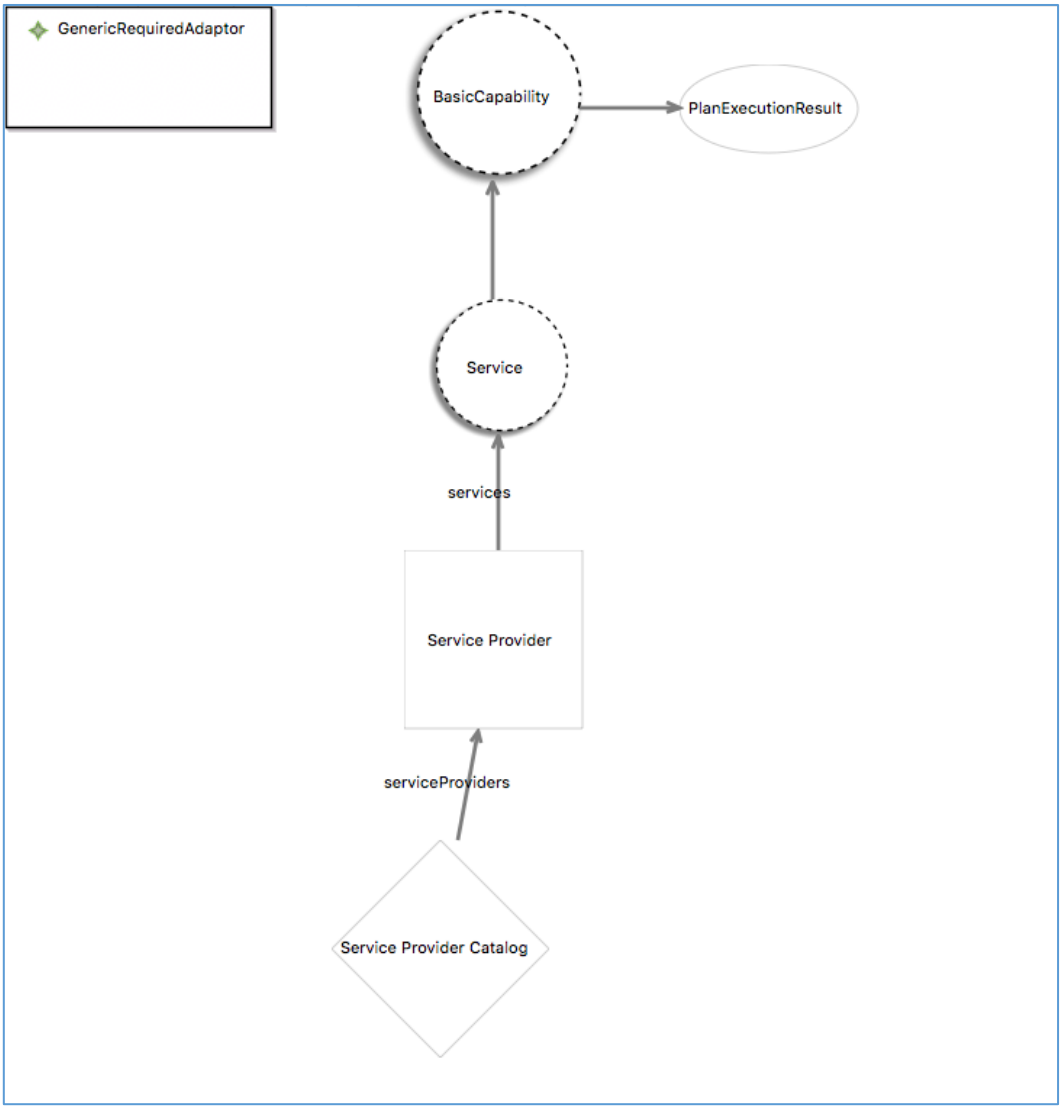

Figure 14. Robot twin internal service structure.

3.5.3 Robot and Scene ROS Nodes

The ROS framework 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 [12], 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.

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 [19] 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.
- **Physical Robot Scenario:** In this scenario, which will be implemented in the Sandbox Beta iteration, real robots are equipped with their own onboard computer that are 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, their should run the same algorithms of the simulated setup.

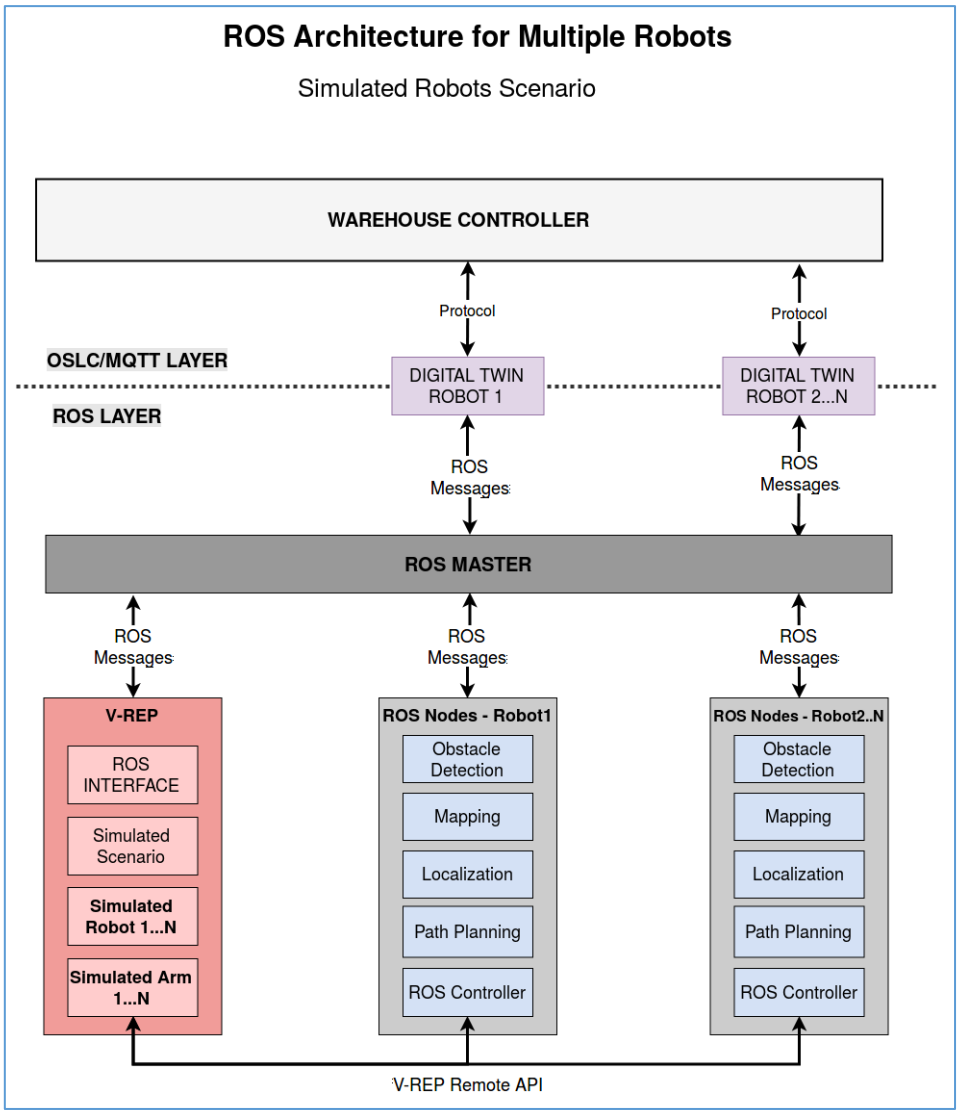

Figure 15. ROS-based architecture for the physical and simulated robots as well as their integration with the rest of the sandbox through the use of digital twins.

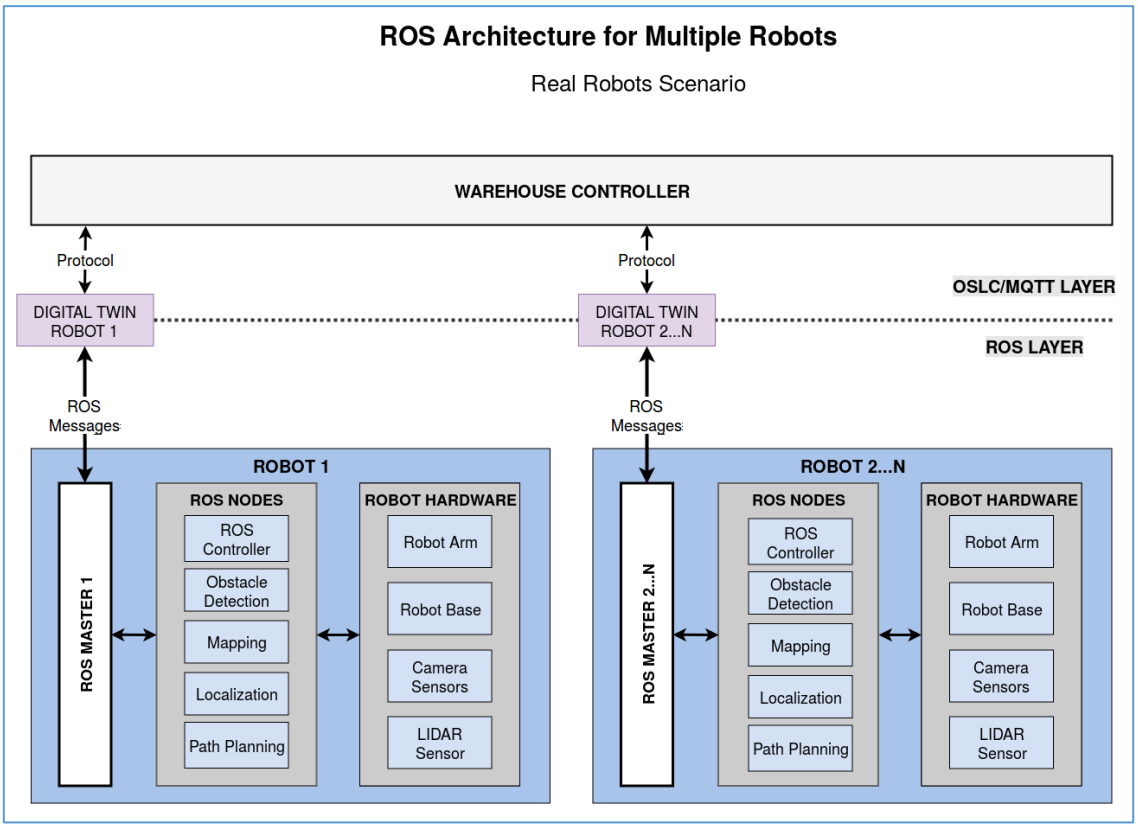

Figure 16. ROS-based architecture for the physical robot scenario.

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.

3.5.3.1 V-REP simulator

Before running experiments with real robots, it is interesting to validate the code in a simulated environment. The V-REP simulation environment can be seen in the following Figure.

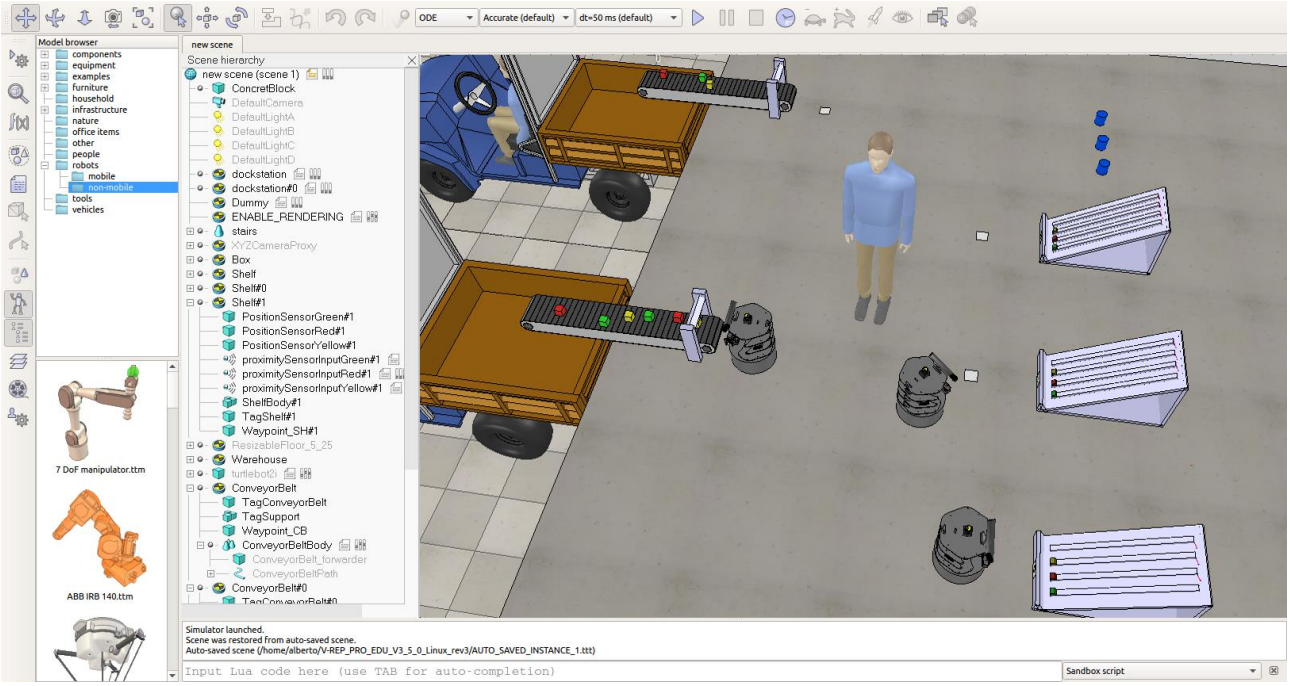

Figure 17. V-REP scene of the warehouse sandbox of the WP10.

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 [20], based on user experience).

3.6 Continuous Integration & Deployment (CI/CD)

Related building blocks	BB26.B_JIG (Cloud computing service platform) *BB24.E_JIG (Cloud computing services for novel connected mobility applications)*
Related requirements	REQ 649 Software upgrade in controllers (BB24.E_JIG)
Implementation scope of the TBBs in the component	The scope of the implementation of the BB26.B_JIG in WP10 lies within the use of the open-source core components needed for the CI/CD operations of the WP10 cloud services.

Table 5. TBB and requirement mapping for the CI/CD approach.

To ensure efficient development, continuous integration and deployment approach has been adopted. It is based on the internal lab installation of GitLab which is integrated with the main code repository at GitHub [10]. The source code of the project is divided into subprojects each of which may declare corresponding sections of the global manifest describing various stages and steps about applicable integration and deployment procedures. A commit to the project's GitHub repository triggers an event in the GitLab through a custom-built web service and spawns a CI/CD pipeline. Upon successful build and test, the pipeline also deploys the services in the project lab cluster(s) that are built with Docker Swarm on top of OpenStack-managed hardware.

Both stages are executed by the GitLab runners in fresh Docker containers. The source code is not taken from the Gitlab directly but from the original GitHub repository, though, GitLab is also used to mirror the code for redundancy purposes. In a subsequent build stage, a Docker container image is created. The image is subsequently pushed to the docker registry that is an integral part of the GitLab installation. Finally, the deployment of the Sandbox services to the Docker Swarm cluster is performed. It adopts subproject's `docker-compose.yml` manifest to instruct Docker Swarm manager about the configuration of the deployed service. In the depicted case, the exposed port of the services are adjusted prior to deployment.

3.6.1 Object Detection as CI/CD

Related building blocks	BB24.D_ITI (Big data analytics)
Related requirements	REQ 642 Knowledge transfer
Implementation scope of the TBBs in the component	Within the scope of BB24.D_ITI, the Objection Detection component is developed in WP10 in order to enable transfer of the machine learning models to enable analysis of the data (primarily, sensory) both in cloud services and on the physical devices.

Table 6. TBB and requirement mapping for the object detection component.

For simplicity, the process of maintaining object detection models is reduced to that of continuous deployment and continuous integration (CI/CD). This is done in order to leverage general purpose CI/CD as opposed to using machine learning framework bound techniques such as Tensorflow serving [11].

Assuming a Docker Swarm based approach to CI/CD, the following table describes the different aspects that would describe the CI/CD setup along with its individual components.

Deployments	Services
gitlab	gitlab-nodeport
postgres	postgres-nodeport
redis	redis-nodeport
gitlab-multi-runner	*not needed*
minio	minio-nodeport

Table 7. Mapping of deployments and services for object detection.

This deployment (of the CI/CD components) is meant to stay in a development cluster or namespace depending on hardware availability. The actual build task is carried out by gitlab-multi-runner which relies on each project's .gitlab-ci.yml file to deal with the specifics of building each project (compilation tasks), releasing (creation of corresponding docker image and pushing them to a docker registry), testing of the docker image with sample requests and deployment to the production cluster assuming all previous steps have completed successfully.

Figure 18 provides a high-level overview for the process of building and maintaining an object detection model. The process is broken down in two parts, bootstrapping and train/verify model.

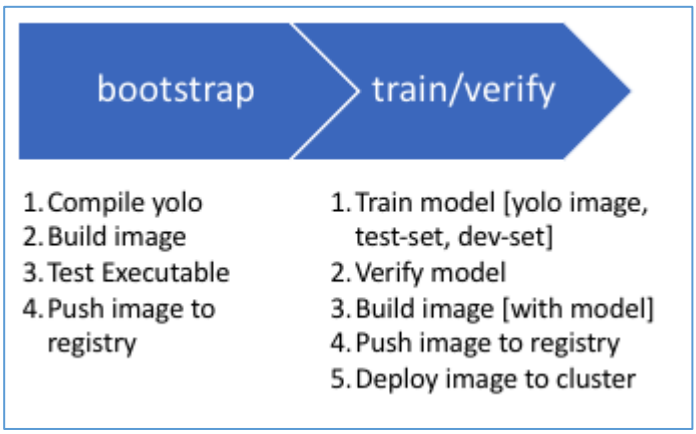

Figure 18. A two-part process for building and maintaining an object detection model.

3.6.1.1 Bootstrapping

This part of the process is tasked with creating a docker image that deals with the underlying machine learning software that we use for training a convolutional neural network (CNN). For simplicity, in our case we have chosen YOLO. The output of this model, if compilation is successful, would be a docker image in our local registry which can successfully run YOLO. Moreover, this setup can take care of compiling YOLO along with CUDA in order to benefit from hardware acceleration. Since this model only cares about YOLO – this part of the process will be triggered every time there is a change in the source code of YOLO.

3.6.1.2 Train/Verify model

This step of the process is meant to take care of the actual training of the model using the image that was pushed in the docker registry by the bootstrapping process. This part is data-driven and is triggered once there are new files (new images of objects to be detected). Every time new images are pushed into the corresponding repository this process is triggered. During the training process, the set of files is mounted as a volume into the docker image which is tasked for training the model. The verification phase is meant to estimate how good that model is by exposing the model to images that it did not see while it was trained. The trained model along with YOLO is then bundled into the same image which is pushed into the docker registry. Afterwards the new image is deployed in the cluster, thus making this model available to the robots in as-a-service approach. To benefit from hardware acceleration the deployment needs to target a compute node that supports hardware acceleration. From a cluster perspective this node needs to be protected from other kinds of workload that do not need such acceleration.

3.7 Evaluation

3.7.1 Deliverable Demo Evaluation

The evaluation of the deliverable demo in Year 1 will consist of two parts:

1. Completeness of the Warehouse Sandbox (alpha) demo description.
2. Quality of the Warehouse Sandbox (alpha) demo description.
3. Satisfaction of the demo requirements by the Warehourse Sandbox (alpha).

The completeness of the demo description is assessed by the coverage of the components that constitute a demo in Year 1. These components are highlighted in Figure 2. The following components were targeted for Year 1 demo:

1. Warehouse Controller – covered in Section 3.4.1
2. Digital Twin for the Robots and the Shelves – covered in Section 3.4.2
3. Planner Service – covered in Section 3.4.4
4. Estimator – covered in Section 3.4.5
5. ROS Master, ROS Nodes, and Simulation – covered in Section 3.4.3

Therefore, the completeness criteria is satisfied fully.

The quality of the demo description is ensured through the internal and formal reviews of this deliverable.

Satisfaction of the demo requirements is evaluated below according to the standard SCOTT requirement assessment scoring for the Use Case work packages in SP2 [23, §3.1.2]. Note that the adjusted scale [23, Table 2] for the non-waterfall model is used, as WP10 follows agile approach:

Requirement	Self-assessment score
DemoRQ 640 Monitoring and control	30%*
DemoRQ 641 Digital Twins	40%

Requirement	Self-assessment score
DemoRQ 643 Pubsub exchange	40%
DemoRQ 647 Warehouse instrumentation	30%**
DemoRQ 648 Robot and robot software	40%
DemoRQ 639 Operation Optimization	15%
DemoRQ 650 Testing framework for PDDL domains	15%
DemoRQ 681 Safe operations	15%
DemoRQ 682 Collaborative Operations	15%

Table 8. Demo requirements score self-assessment for WP10.

* DemoRQ 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.

** DemoRQ 647 will be implemented through a pair of individual components (Problem Planning Service and Planning Domain Service in Figure 2) but in this demo their functionality was prototyped inside of the Warehouse Controller component.

3.7.2 Overall Work Package Evaluation

According to the SP1 leadership guidelines, the WP10 has a progress score of 15%, which means that the D10.1 [1] specification has been finalised.

If an average score of all demo requirements is considered, the WP10 progress score as of D10.2 publication can be considered to equal 26%. This includes the requirements that were not targeted by this deliverable.

4 DISSEMINATION, EXPLOITATION AND STANDARDISATION

The code for the deliverable can be found under https://github.com/EricssonResearch/scott-eu

Modifications to the OSLC TRS specification in order to allow distribution of the change events over MQTT are being contributed to the OSLC Core Technical Committee [13] which operates under Organization for the Advancement of Structured Information Standards (OASIS).

The following two publications have been presented since the deliverable D10.1:

Rafia Inam, Elena Fersman, Klaus Raizer, Ricardo Souza, Amadeu Nascimento Junior, Alberto Hata. "Safety for Automated Warehouse exhibiting collaborative robots". Accepted, 28th European Safety and Reliability Conference (ESREL'18), June 17-21 2018, Trondheim, Norway.

The main focus of the paper is the presentation of our safety strategy for the collaborative operations performed in the warehouse. The strategy is modelled based on dynamic safety fields around the robot, which is consistent with important guidelines in collaborative robotics (i.e., ISO15066). We propose to perform safety analysis at two levels: a) Offline safety analysis is performed at the simulation level before the robot operation, to check the plan to be safe; b) Online safety analysis that will be performed at runtime during the robot operation for additional safety checks. We propose a risk assessment and risk management for online safety analysis.

Please note that safety requirements are part for Sandbox beta version, and we write it briefly here only to present a complete picture of warehouse architecture. The details and code will be presented next year.

Didem Gürdür, Aneta Vulgarakis Feljanb, Jad El-khourya, Swarup Kumar Mohalik, Ramamurthy Badrinath, Anusha Pradeep Mujumdar, Elena Fersman, "Knowledge Representation of Cyber-physical Systems for Monitoring Purpose", Accepted, Proceedings of the 51st CIRP conference on Manufacturing Systems(https://www.cirp-cms2018.org/), 16-18 May 2018, Stockholm.

During the warehouse operations, a number of key performance indicators (KPI) can be monitored to understand the proficiency of the warehouse and control the operations and decisions. It is possible to drive and monitor these KPIs by looking at both the state of the warehouse components and the operations carried out by them. Therefore, it is necessary to represent this knowledge in an explicit and formally-specified data model and provide automated methods to derive the KPIs from the representation. In this paper, we implement a minimalistic data model for a subset of warehouse resources using linked data in order to monitor a few KPIs, namely sustainability, safety and performance. We demonstrate that it is possible to develop the data models through Open Services for Lifecycle Collaboration (OSLC) resource shapes which enables compatibility with the declarative and procedural knowledge of automated warehouse agents specified in Planning Domain Definition Language (PDDL).

5 LINK TO TECHNOLOGY LINES

The link to the Technology Building Blocks is described in separate tables at the beginning of the relevant subsections of the Section 3. The summary of this links, and in particular, the related requirements covered in this deliverable, is given below.

5.1 WP24 Distributed Cloud Integration

5.1.1 BB24.C Appl. Layer Protocols

In Year 1, Ericsson and KTH have contributed to BB24.C in the following way:

- Contributed the REQ 644 Common knowledge representation.
- Contributed the REQ 651 Linked data platform.
- Provided an initial solution for REQ 644 in the information model covered in Section 3.3 of this deliverable.
- Provided an initial implementation of the REQ 651, covered in Section 3 of this deliverable and shared as open-source (Section 4).

5.1.2 BB24.D Big Data Analytics

In Year 1, Ericsson and KTH have contributed to BB24.D in the following way:

- Contributed the REQ 642 Knowledge transfer
- Contributed the REQ 645 Strategic and reactive planning
- Contributed the REQ 646 Plan verification for safety
- Provided a foundation for implementing REQ 642 through a two-part process for building and maintaining an object detection model detailed in Section 3.5.1
- Provided a foundation for implementing REQ 645 through the development of an initial prototype of the Warehouse Controller (Section 3.4.1) and the Planner Service (Section 3.4.4)
- Provided a foundation for implementing REQ 646 through the development of an initial prototype of the Estimator (Section 3.4.5)

5.1.3 BB24.E Cloud Services

In Year 1, Ericsson and KTH have contributed to BB24.E in the following way:

- Contributed to the definition of the REQ 649 Software upgrade in controllers
- Introduced CI/CD process to cover the REQ 649 (Section 3.5)

5.2 WP26 Reference Architecture / Reference Implementations

5.2.1 BB26.B Cloud Platform

In Year 1, Ericsson and KTH have contributed to BB26.B in the following way:

- Contributed to the definition of the REQ 260 Support IPv6 network protocol
- Contributed to the definition of the REQ 261 Auto Scaling System
- Contributed to the definition of the REQ 263 Connection restrictions
- Contributed to the definition of the REQ 264 Auto deployment
- Designed the architecture and the CI/CD process (Sections 3.2 and 3.5, respectively) to utilise containers and Docker Swarm to allow the requirements listed above to be implemented during Year 2.

6 INTEROPERABILITY

Interoperability is ensured at multiple levels. First, at the project level through the participation in Building Blocks and the alignment of the requirements, as described in the Section 5 above. 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, as described in the Section 3.4 above.
2. The development of a Linked Data architecture, described in the Section 3.3 above.

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. 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

During the first iteration during Year 1, WP10 and the involved partners:

- defined a set of requirements, both the demo requirements on the use-case itself and the requirements on the building blocks;
- defined the alpha version of the sandbox specification D10.1 [1];
- developed a detailed architecture of the sandbox;
- developed an approach for information modelling to allow interoperability between heterogeneous devices and systems, while providing a common ontology on which a planner can work to orchestrate the activity throughout the warehouse;
- developed an alpha version of the sandbox and made it publicly available on the Github platform;
- presented certain parts of its research in recognised conferences and journals;
- proposed the modifications to the OSLC TRS specification to the appropriate committee under the Organization for the Advancement of Structured Information Standards;
- contributed to building blocks listed in the Section 5 above;
- contributed to the Trust Framework deliverables D28.2 ([14], Section 3.5) and D28.3 ([15], Section 4.4).

In the next iteration, the sandbox specification will be improved in D10.3 [3] and the components planned for the next iteration will be developed. WP10 partners will continue collaboration within the building blocks to ensure reuse.

REFERENCES

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

[2] IEEE Standard 1872-2015 - Ontologies for Robotics and Automation.

[3] SCOTT Deliverable "D10.3 Sandbox Specification (Beta) - A Sandbox for Intelligent Warehouse Logistics", due M16

[4] OSLC Core library implementation in Prolog – https://github.com/EricssonResearch/oslc_prolog (last accessed: August 2018)

[5] SWI-Prolog Semantic Web Library 3.0 – http://www.swi-prolog.org/pldoc/doc_for?object=section(%27packages/semweb.html%27) (last accessed: August 2018)

[6] The ClioPatria Semantic Web Toolkit – https://github.com/ClioPatria/ClioPatria (last accessed: August 2018)

[7] PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains – http://icaps-conference.org/ipc2008/deterministic/data/fox-long-jair-2003.pdf (appendix A). (last accessed: August 2018)

[8] Bug 526362 "Allow exporting (and importing) specification and vocabulary as RDF" – https://bugs.eclipse.org/bugs/show_bug.cgi?id=526362 (last accessed: August 2018)

[9] Blocks world – https://en.wikipedia.org/wiki/Blocks_world (last accessed: August 2018)

[10] SCOTT WP10 source code repository for the Warehouse Sandbox implementation – https://github.com/EricssonResearch/scott-eu (last accessed: August 2018)

[11] Tensorflow Serving – https://www.tensorflow.org/serving/ (last accessed: August 2018)

[12] Turtlebot 2i (Trossen Robotics) – http://www.trossenrobotics.com/interbotix-turtlebot-2i-mobile-ros-platform.aspx (last accessed: August 2018)

[13] OASIS OSLC Lifecycle Integration Core (OSLC Core) TC – https://www.oasis-open.org/committees/oslc-core/ (last accessed: August 2018)

[14] SCOTT Deliverable "D28.2 Refined use case context definitions", v1.0, 2018-01-30

[15] SCOTT Deliverable "D28.3 "Use Case Vulnerabilities", v1.0, 2018-05-22

[16] Open Services for Lifecycle Collaboration – http://open-services.net (last accessed: August 2018)

[17] Eclipse Lyo – http://www.eclipse.org/lyo/ (last accessed: August 2018)

[18] Robot Operating System – http://www.ros.org (last accessed: August 2018)

[19] V-REP virtual robot experimentation platform – http://www.v-rep.eu (last accessed: August 2018)

[20] Gazebo – http://gazebosim.org (last accessed: August 2018)

[21] Metric-FF – https://fai.cs.uni-saarland.de/hoffmann/metric-ff.html (last accessed: August 2018)

[22] VAL, the plan validation system – https://github.com/KCL-Planning/VAL (last accessed: August 2018)

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

[24] SCOTT JU Grant Agreement incl. Description of Action (DoA), ECSEL Joint Undertaking, Grant Agreement No. 737422, Part B, 2018-07-11.

A. ABBREVIATIONS AND DEFINITIONS

Term	Definition
API	Application Programming Interface
BNF	Backus Naur form
CPS	Cyber-Physical Systems
CRUD	Create, Read, Update, Delete (operations)
DevOps	compound of "Development" and "Operations"
HLA	High-level architecture
HTTP	Hypertext Transfer Protocol
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
RDF	Resource Description Framework
REST	Representational State Transfer
ROS	Robot Operating System
TBB	Technical Building Block
TRS	OSLC Tracked Resource Set
V-REP	Virtual Robot Experimentation Platform, see [19]
WHC	Warehouse Controller