



Sustainable and reliable robotics for part handling in manufacturing

Project no.: 610917

Project full title: Sustainable and reliable robotics for part handling in manufacturing

Project Acronym: STAMINA

Deliverable no.: 4.2.3

Title of the document: Conversion logic and high level operations scheduling

Contractual Date of Delivery to the CEC:	30.09.2015
Actual Date of Delivery to the CEC:	28.09.2015
Organisation name of lead contractor for this deliverable:	INESC Porto
Author(s):	César Toscano, Germano Veiga, Matthew Crosby, Ron Petrick
Work package contributing to the deliverable:	WP4
Nature:	R
Version:	1.0
Total number of pages:	36
Start date of project:	01.10.2013
Duration:	42 months – 31.03.2017

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 610917

Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

This deliverable is an output of Task 4.2 “Logistics Planning and EIS integration with ERP system and related sub-systems” and provides the specification of the logistic planning activities performed by the Logistic Planner component.

Document History

Version	Date	Author (Unit)	Description
0.1	31-08-2015	César Toscano (INESC Porto)	Creation of document and first contents.
0.2	08-09-2015	César Toscano (INESC Porto)	Completion of most chapters and distribution to UEDIN for section 3.4.4.
0.3	11-09-2015	Matthew Crosby (UEDIN), Ron Petrick (UEDIN)	Completed description of Mission Planner.
0.4	12-09-2015	César Toscano (INESC Porto)	Revised whole document. Submitted to project leadership for validation and for WP4 partners.
0.5	21-09-2015	Volker Krüger (AAU)	Revised/validated whole document.
1.0	21-09-2015	César Toscano (INESC Porto)	Generated final version.

Table of Contents

GLOSSARY	5
1 INTRODUCTION	7
1.1 SCOPE AND OBJECTIVES	7
1.2 DOCUMENT ORGANIZATION	7
1.3 REFERENCES	7
2 STAMINA HIGH LEVEL SYSTEM ARCHITECTURE	8
2.1 OVERVIEW	8
2.2 LOGISTIC PLANNER AND EIS INTERFACE	9
2.3 MISSION PLANNER.....	9
2.4 TASK MANAGER AND SKILLS MANAGER	9
2.5 SERVICES	10
3 LOGISTIC PLANNER.....	11
3.1 PURPOSE AND KEY FUNCTIONS.....	11
3.2 HIGH LEVEL SYSTEM ARCHITECTURE AND INFORMATION FLOW	11
3.3 LOGISTIC WORLD MODEL	12
3.3.1 <i>Introduction</i>	12
3.3.2 <i>Conceptual representation</i>	13
3.3.3 <i>Concrete representation</i>	16
3.3.4 <i>Physical objects</i>	17
3.3.5 <i>Publishing and real-time updating</i>	20
3.3.6 <i>Set up</i>	21
3.4 ORDER MANAGER.....	26
3.4.1 <i>Introduction</i>	26
3.4.2 <i>FAC and Kitting Orders</i>	26
3.4.3 <i>Mission and Skills</i>	29
3.4.4 <i>Planning kitting orders on the Mission Planner</i>	31
3.5 SOFTWARE IMPLEMENTATION	34

Figures

Figure 1 – STAMINA high level system architecture (components and information flow).	8
Figure 2 – World Model: conceptual representation.	14
Figure 3 – World Model: bounding volume.	15
Figure 4 – World Model: concrete representation (UML class diagram).....	16
Figure 5 – World Model: instance with rack and small box (UML object diagram).	17
Figure 6 – World Model: types of physical objects (UML class diagram).....	18
Figure 7 – World Model: EIS integration services for managing types of physical objects.	18
Figure 8 – World Model: physical objects (UML class diagram).	19
Figure 9 – World Model: service for retrieving skills of a robot.	20
Figure 10 – User interface: skills implemented by a robot	20
Figure 11 – World Model: runtime services.	21
Figure 12 – User interface: part types.	22
Figure 13 – User interface: large box types.	22
Figure 14 – User interface: reference point and horizontal line on a SLAM-based image.	23
Figure 15 - User interface: specification of container and navigation regions.	23
Figure 16 - User interface: implantation of racks and large boxes.	24
Figure 17 - User interface: association of parts to small boxes.	24
Figure 18 - User interface: association of parts to large boxes.	24
Figure 19 - User interface: association of small boxes to racks.....	25
Figure 20 – User interface: 3D view of the logistic world model.....	25
Figure 21 – Order Manager: eFAC and Kitting Order (UML class diagram).	27
Figure 22 – Order Manager: integration services.	28
Figure 23 – User interface: importing kitting orders from external files.....	28
Figure 24 – User interface: status of a kitting order.	29
Figure 25 – Order Manager: mission and skill (UML class diagram).	29
Figure 26 – User interface: identification of a mission just after its creation.	30
Figure 27 – User interface: identification of a mission's status history.....	30
Figure 28 – Order Manager: mission related runtime services.	31
Figure 29 – Mission Planner: an overview of its internal processing.....	32
Figure 30 – STAMINA components and ROS nodes (UML deployment diagram).	35
Figure 31 – Software organization within the Logistic Planner.	36

Glossary

Term	Definition
API	Application Program Interface, specifies the visible interface of a given software element.
EIS	Enterprise Information Systems. General category of systems including ERP, MES and other information systems used in the enterprise.
ERP	Enterprise Resource Planning.
FAC	”Fiche Accompagnement”, created by PSA MES system, identifies the parts to be built in the kitting zone in the form of a kit.
HTML	HyperText Markup Language.
HTTP	Hypertext Transfer Protocol.
JSON	JavaScript Object Notation, syntax for describing data objects in the java script language. It is the most popular data format for supporting RESTfull web services.
Kitting Order	Instruction to build a single Kit (a kitting order is built from the FAC object generated by PSA EIS systems; the FAC – “fiche d’accompagnement” - corresponds to the printed paper given to the kitting operator telling him what to do in order to build a kit).
Large box	Type of packaging mainly used for material handling of large items/goods. Large box containers are usually handled by forklifts.
MES	Management Execution System.
Rack	Furniture element in a kitting area, containing small boxes.
RESTfull Web-Service	Specific type of web-service implemented through the HTTP protocol and following the POST-GET-PUT-DELETE model.
Robot	Machine that is doing the work in the kitting zone, i.e., building a given kit (a single robot includes several distinguished elements responsible for navigation, manipulation, inspection, etc.).
Robot Mission	Identification of the skills that the robot will use to fulfil one or two Kitting Orders.
Robot Skill	Primitive based description of compliant robot motions (action primitives), described as simple, atomic movements that can be combined to form a task.
ROS	Robot Operating System.

Small box	Box-type packaging used for material handling of small items/goods. Small boxes are usually stored in specific Kanban racks.
Takt Time	Maximal time to realize a mission or an assembly task (in this case maximum time to provide a kit).
UML	Unified Modelling Language.
Web-Service	Software system designed to support interoperable machine-to-machine interaction over a network (World Wide Web Consortium – W3C definition). A Web-Service has a specific interface, described in the XML and WSDL languages.
XML	Extensible Markup Language, syntax for describing structured data vocabularies.
XSD	XML Schema Definition.

1 Introduction

1.1 Scope and Objectives

This deliverable is the third result of Task 4.2 “Logistics Planning and EIS Integration with ERP systems and related sub-systems”. The main objective of this task is to provide integration mechanisms between STAMINA system and the Enterprise information Systems (EIS), including the conversion logic with the higher level ERP planning definitions. The integration mechanisms are designed by following a service-oriented architecture style and by considering the type of information commonly available by EIS in general and by PSA EIS in particular.

This deliverable aims to specify the activities performed by STAMINA Logistic Planner component and reports its current stage of development.

Two other deliverables are related with this one in the following aspects:

- Deliverable D4.1.1 “Specification of SOA for robot fleets”, version 1.2, 31-05-2015, specifies the service-oriented architecture of STAMINA, providing the details of all the services and interfaces.
- Deliverable D4.2.2 “Interface with ERP systems, version 1.0, 30-09-2015, written at the same time as this report, describes the kitting processes at PSA and the integration services and interfaces with EIS elements (ERP, MES, ...). As this report is confidential but provides the most updated version of STAMINA (WP4) high level system architecture, a copy of this section is also included in this report.

1.2 Document organization

Following this introductory chapter, the document is organized as follows:

- Chapter 2 – description of STAMINA high-level system architecture, according to the present state of the project. The major STAMINA components (excluding the robot specific modules and algorithms) are identified in terms of functionality together with their key dependencies. This section is also present in Deliverable D4.2.2.
- Chapter 3 – description of the Logistic Planner component - logistic world model and order manager, including the planning of missions – and overview of its current software implementation.

1.3 References

STAMINA consortium, 2014a. “Deliverable D3.2 Revised report on skill architecture”, version 1.0, 31-05-2015.

STAMINA consortium, 2014b. “Deliverable D4.1.1 Specification of SOA for robot fleets”, version 1.2, 29-04-2015.

STAMINA consortium, 2014c. “Deliverable D4.2.2 Interface with ERP systems”, version 1.0, 30-09-2015.

2 STAMINA high level system architecture

2.1 Overview

Figure 1¹ identifies the components that comprise the STAMINA system and the corresponding information flow (colours are used to differentiate the two main categories of information elements). At the bottom level, each STAMINA robot, apart from all the hardware components, is managed by two ICT elements: a Task Manager and a Skills Manager, which encapsulate and hide all the details concerning the execution of the actions performed by the robot (see D3.2 “Revised report on skill architecture”). These two elements operate on top of the robot’s operating system (ROS), running as ROS Nodes, in the most autonomous way as possible. At the top level, the Logistic Planner and the Mission Planner have general visibility over the robot fleet. The second runs also as a ROS node while the Logistic Planner is more appropriately supported by both Windows or Linux. EIS represent all the external systems integrated into STAMINA (e.g. ERP and MES).

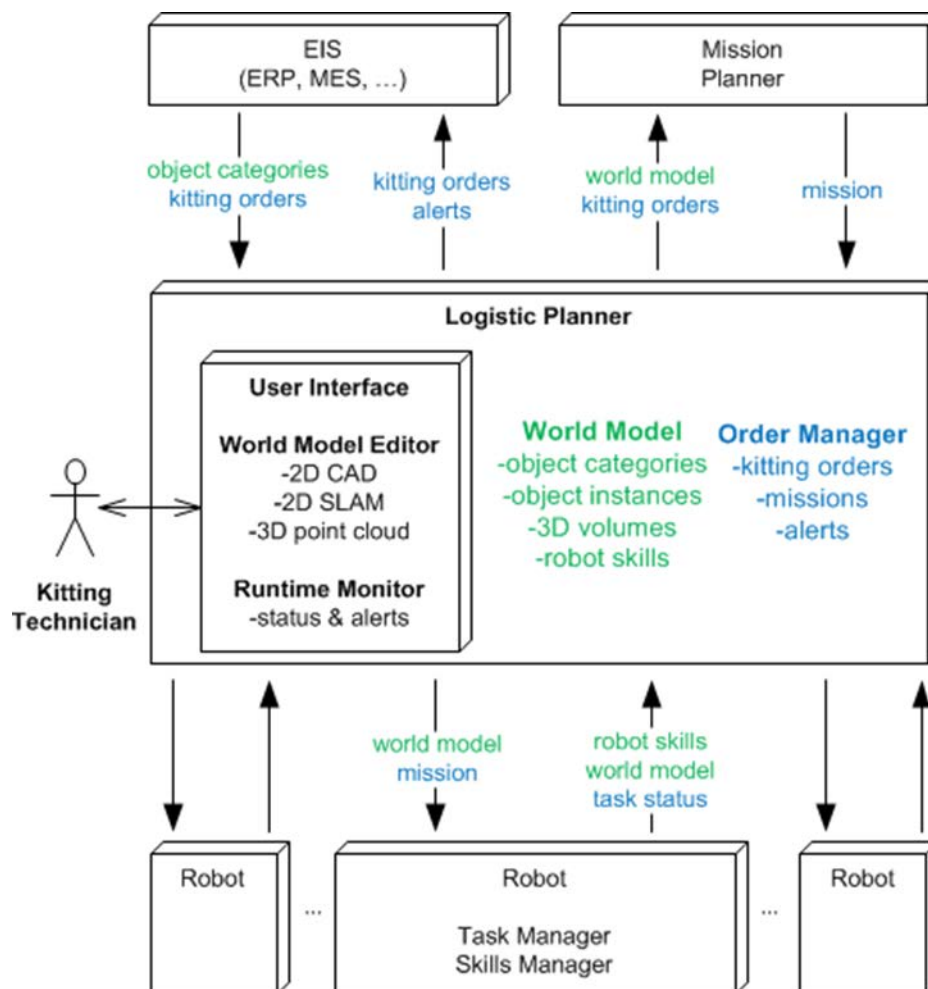


Figure 1 – STAMINA high level system architecture (components and information flow).

¹ Many relevant STAMINA elements comprising hardware and software components at a lower level than the Skills Manager are not shown in this high level perspective as they would increase the complexity of the diagram without adding any value to the goal of this report.

2.2 Logistic Planner and EIS interface

The Logistic Planner component aims at providing the integration mechanism with EIS elements. MES, ERP and any other information management systems that could be the source or destination of data are included in this EIS categorization. Conversion logic with higher level planning functions defined in ERP or MES type systems is the first main goal of the Logistic Planner. The semantics and format of data originating in the EIS are converted to a format understandable by the STAMINA components and, inversely, actions performed by these components are identified and converted back to a form understandable by the EIS. Thus, the Logistic Planner hides the remaining STAMINA components from these integration-related issues.

Moreover, the Logistic Planner aims to mediate between the functions performed by the remaining components in STAMINA, providing them with all the data about the working environment. This includes the identification of all the physical objects (racks, small boxes, large boxes, conveyors, kits, parts,) and their three dimensional models. This constitutes the output of the modelling done by the Kitting Technician (responsible for the kitting zone) in the Logistic Planner and is formally known as the world model (details in subsections below).

2.3 Mission Planner

The Mission Planner is responsible for generating mission assignments for individual robots in the fleet. Missions are created based on the status of the robots and the set of upcoming kitting orders. In the current version of the architecture, a mission is defined as a goal assignment to a specific robot for completing a single trip around the kitting zone. As the robot can carry two kitting boxes, a mission may involve the fulfilment of up to two kitting orders.

The Mission Planner interacts exclusively with the Logistic Planner, which provides details about the current set of kitting orders to be completed, the state of the robots in the fleet and their available skills, and information about the robots' operating environment (e.g., the world model). Missions are constructed using general-purpose automated planning techniques, by invoking a novel multi-agent planning algorithm. In addition, and as a by-product of the planning approach, a sequence of the skills a robot needs to execute for achieving each mission is also generated. Once generated, the set of mission assignments and skill sequences is returned to the Logistic Planner for delivery to individual robots in the fleet.

2.4 Task Manager and Skills Manager

Each robot in the fleet has an on-board Task Manager and Skills Manager. A Task Manager interacts with both the Logistic Planner and the Skills Manager. The Task Manager receives Mission assignments from the Logistic Planner, calls for the execution of skills (by the Skill Manager), and keeps track of the robot's progress on its mission. This progress is sent as updates to the Logistic Planner and used to invoke the Task Planner should re-planning be required.

Inside the Task Manager, a Task Planner provides automated planning services at the individual robot level, should further planning be required to fulfil a mission. For example, if the robot finds that its currently assigned sequence of skills is not applicable in the current environment, the Task Planner will be invoked to perform re-planning and find an alternative method of reaching the end goal of the mission (fulfilling one or two kitting orders). The Task Planner only interacts with the Task Manager and is invoked any time there is no conflict-free sequence of skills ready to achieve the robot's current mission.

The Skill Manager performs and monitors the execution of skills on the robot. In particular, skills requested by the Task Manager are executed by the Skill Manager by accessing the robot's skill database and associated world model. Precondition checks are performed to ensure that the conditions necessary for carrying out the requested skill have been satisfied, and post-condition checks verify whether the requested skill achieved its desired outcome successfully. The results of these checks are reported to the Task Manager, which can trigger re-planning from the Task Planner should a failure occur, or request the execution of the next skill if successful, or request help from the operator in case no skill sequence can be found.

2.5 Services

The task of dispatching a list of kitting orders (or list of FACs) created by the external EIS systems is a major goal of STAMINA. This involves the creation of a mission, its assignment to a given robot in the fleet, the planning of the skill sequences that are required to achieve a given mission and the execution of the plan by the robot by translating skills into low-level actions. These major functions are described and accessible in the form of services and constitute the **runtime services** of STAMINA.

Runtime actions are executed by different STAMINA components and they may (and generally do) require setup actions so that their runtime behaviour is as independent as possible from supporting information. **Setup services** are defined to implement this need.

The integration services with PSA EIS are available both as setup and runtime services.

3 Logistic Planner

3.1 Purpose and key functions

The Logistic Planner element aims at providing the integration mechanism with EIS elements. MES, ERP and any other information management systems that could be the source or destination of data are included in this EIS categorization. Conversion logic with higher level planning functions defined in ERP or MES type of systems is the first main goal of the Logistic Planner. The semantics and format of data originating in the EIS are converted to a format understandable by the STAMINA components and, inversely, actions performed by these components are identified and converted back to a form understandable by the EIS. Thus, the Logistic Planner hides the remaining STAMINA components from these integration-related issues.

Moreover, the Logistic Planner aims to mediate between the functions performed by the remaining components in STAMINA (the Mission Planner, and, on each robot, the Task Manager and the Skills Manager), providing them with all the data about the working environment. This includes the identification of all the physical objects (racks, small boxes, large boxes, conveyors, kits, parts,) and their three dimensional models. This constitutes the output of the modelling done by the Kitting Technician (responsible for the kitting zone) in the Logistic Planner and is formally known as the world model.

In this context, the Logistic Planner controls and monitors the higher level functions of STAMINA:

- the creation of Missions and Tasks,
- the assignment of Tasks and Skills to the robot team, and
- the consequent execution and control,

taking the major information inputs from PSA EIS and notifying it of exceptional events in STAMINA.

Information about the following key items is kept within the Logistic Planner:

- Parts (code references, description, ...);
- Packaging structure;
- Kit structure (cells, parts to place in each cell);
- Furniture structure in the kitting zone (shelves, levels, cells within each level);
- Implantation structure (for empty kits, full kits, large boxes, small boxes, parts and packaging elements) and CAD maps;
- Missions and Tasks (planned, active, finished);
- Abstract skills.

3.2 High level system architecture and information flow

Figure 1 identifies the components that comprise the STAMINA system and the corresponding information flow (colours are used to differentiate the two main categories of information elements). At the bottom level, each STAMINA robot, apart from all the hardware components, is managed by two ICT elements: a Task Manager and a Skills Manager, which encapsulate and hide all the details concerning the execution of the actions performed by the robot (see D3.2 “Revised report on skill architecture”). These two elements operate on top of the robot’s operating system (ROS), running as ROS Nodes, in the most autonomous way as possible. At the top level, the

Logistic Planner and the Mission Planner have general visibility over the robot fleet. The second runs also as a ROS node while the Logistic Planner is more appropriately supported by both Windows or Linux. EIS represent all the external systems integrated into STAMINA (e.g. ERP and MES). A detailed description of the high level system architecture is available on Deliverable D4.2.2.

Internally, the Logistic Planner comprises the following elements (see Figure 1):

- **Logistic World Model** - Data model specifying which physical objects (parts, racks, boxes, conveyors, and kits) are available in the physical environment (i.e. the kitting area), what are their relationships (e.g. which parts are in a given box) and their three-dimensional locations. The world model is initially built by the Kitting Technician on the World Model Editor and later on provided to the remaining STAMINA elements, Mission Planner, Task Manager and Skills Manager.
- **Order Manager** – Element responsible for managing the lifecycle of kitting orders: creation, planning, assignment, execution and monitoring.
- **User Interface** – Element through which the Kitting Technician interacts with the Logistic Planner allowing him to build and maintain the initial world model and to monitor the lifecycle of kitting orders. It comprises the World Model Editor and the Runtime Monitor.

The key information flowing in and out of the Logistic Planner support the following activities:

- EIS supplies initial data about parts, their packaging and kitting zone structure (object categories). This information is used by the Kitting Technician to build a model of the physical environment (world model). Skills from each available robot are retrieved and added to the model. These actions constitute the setup phase.
- During runtime, EIS injects FAC objects and/or kitting orders into the Logistic Planner. This may happen 15 to 20 minutes before the corresponding vehicles enter the assembly line. Subsequently, this is submitted to the Mission Planner together with the actual world model and robot missions are thereafter defined and kept in the Logistic Planner. A Mission is created for each one or two kitting order (depending on the robot skills).
- Missions are then sent to the Task Manager of the corresponding robots in concert with the world model. For each mission, the Task Manager specifies a task plan while interacting with the local Skill Managers. Tasks are subsequently executed and notifications sent to Logistic Planner to allow it to monitor the execution of Mission / Tasks. Update notifications allow the Logistic Model to keep the world model in sync. In this context, the major events in the execution of the plan are transmitted to EIS so as to allow them to follow the activities pursued in the kitting zone with special attention to abnormal/exceptional events (alerts).

3.3 Logistic World Model

3.3.1 Introduction

A crucial function of the Logistic Planner is to act as the major information provider for the high-level planning and execution elements in STAMINA: the Mission Planner at the highest level and the Task Manager and Skills Manager elements on each robot. The type of information communicated among these components mainly concerns the physical objects that are located in the kitting zone, namely major objects including: racks, small boxes, large boxes, conveyors, kits, parts and packaging elements. For each object, its characteristics (e.g., internal organization, geometry

and spatial location) are modelled in the Logistic Planner so as to provide this information to the above planning and execution elements. Modelling of these elements is accomplished in two phases: first, the representation of these objects are gathered from the external EIS system (see Deliverable 4.2.1) in a two dimensional format; second, a third dimension (the height) is added to provide a spatial representation of all physical objects in the kitting zone; finally, actual instances of objects are created and located in the space occupied by the kitting zone. This information model is defined as the world model and is provided to the Mission Planner and Task Manager elements when needed.

At the present stage of development (i.e. at the completion of the second test sprint), the world model is provided by the Logistic Planner to the Mission Planner and Task Manager in a single direction. However and as planned, the world model, as a “rough” approximation of reality, will have to consider the actions made by the robot fleet, taking into account the “things” they detect in the kitting zone, their real locations and the results of their normal behaviour (e.g., picking parts from a small box). This will involve updating the world model and ensuring its consistency in the working environment. Additional research concentrates on this aspect of the work.

3.3.2 Conceptual representation

The world model specifies the position, orientation and dimension of all physical objects available in space and their relationship to other objects. This is accomplished through an inverted hierarchical graph where a node represents a physical object and the link between two nodes represents a containment relationship between the two corresponding physical objects. Figure 2 depicts the organization of the world model and the types of nodes it may contain. In the containment relationship between any two nodes its cardinality is identified (by definition one object is contained in one instance of another object, its “parent”; inversely, any object may have more than one instance of another given object, known as its “children”).

Each node in the graph has the following common attributes:

- an **identifier**, for its unique identification in the graph;
- a **name**, specifying the type of physical object represented by the node;
- a **bounding volume**, specifying the 3D volume of the represented physical object and its location in the kitting zone (relative to the location of its container object);

Additionally, certain nodes in the graph may have **specific properties**, defining specific aspects of the represented object.

By definition, the node on the top of the hierarchy is the root node and represents the entire space (i.e., the kitting zone).

At the top level, the model specifies a **kitting region** as a delimited space in the kitting zone that has a specific purpose. By definition, a region may be decomposed in several regions, each one addressing a more specific and restricted purpose. Even though several levels of decomposition can be achieved, two levels of description are enough to model the space at hand. Thus, the entire kitting region is modelled as being comprised of container regions (regions containing racks, large boxes and conveyors) and navigation regions, modelling the space where the robot fleet navigates.

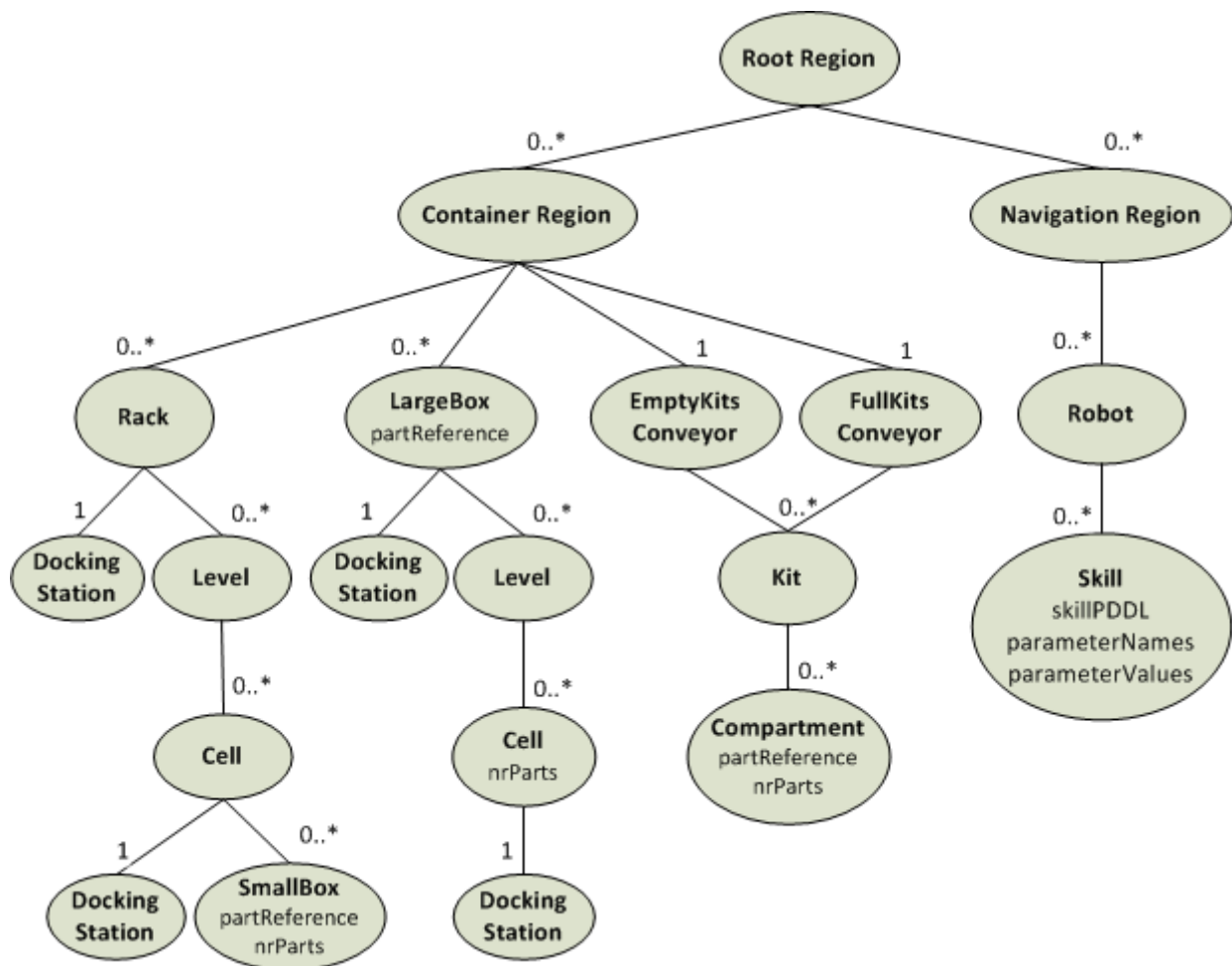


Figure 2 – World Model: conceptual representation.

The remaining nodes in the graph are used to model the following types of elements:

- **Kit** (i.e., kitting box) - box with compartments, where each compartment may have N instances of a given part. The compartment node specifies two specific properties: *partReference* and *nrParts*.
- **Largebox** - box organized in layers (levels) where each level can be organized in cells (compartments). Each cell may have N instances of a given part. Two specific properties are specified in this representation: *partReference* under the *largebox* node and *nrParts* under the *cell* node.
- **Smallbox** - box having N instances of a given part. Smallboxes are available inside a cell within a rack and contain N instances of a given part. Two specific properties are specified in this representation: *partReference* and *nrParts* under the *smallbox* node.
- **Rack** - object organized in levels and having cells on each level. Each cell may have one smallbox of a given type.
- **Conveyor** – equipment for keeping kits (empty and/or full kits) and allowing them to be moved in/out of a robot.
- **Docking Station** – proposed location for parking the robot and let him accomplish manipulation actions on the corresponding objects.

Parts are not represented as nodes in order to avoid excessive information in the graph. For example, a large box containing 20 Parts in each of their four levels would require the graph to have 80 nodes for representing each Part. Also, 3D models of the Part, even though available on the world model, are provided to the interested parties through specific transaction separately from the world model.

Bounding volume and transformation nodes

The world model aims to represent the 3D volume occupied by each and every object in the kitting zone. This is defined as its bounding volume. Most of the objects considered have parallelepiped geometry. The case of kitting boxes is the most complex case as they may have compartments with more complex geometries. In order to have a common volume specification, a 2D closed polyline extruded by height units along the z-axis (see Figure 3) is used to model any physical object in the kitting zone.

The definition of the 2D closed polyline is done in a local coordinate system, whose origin (0,0,0) is the centre point of the closed polyline. The origin of the local coordinate system is related to the origin of the parent's coordinate system (the exception is the root node that, by definition, does not have parent). A 3D vector specifies this translation. Additionally, the local coordinate system may suffer a rotation relative to the parent's coordinate system. A tuple specifying three rotations (roll, pitch and yaw) is used for this purpose. The default value for these two transformation vectors is (0,0,0). The points that define the polyline are defined in the clockwise direction (P_0 , P_1 , etc.).

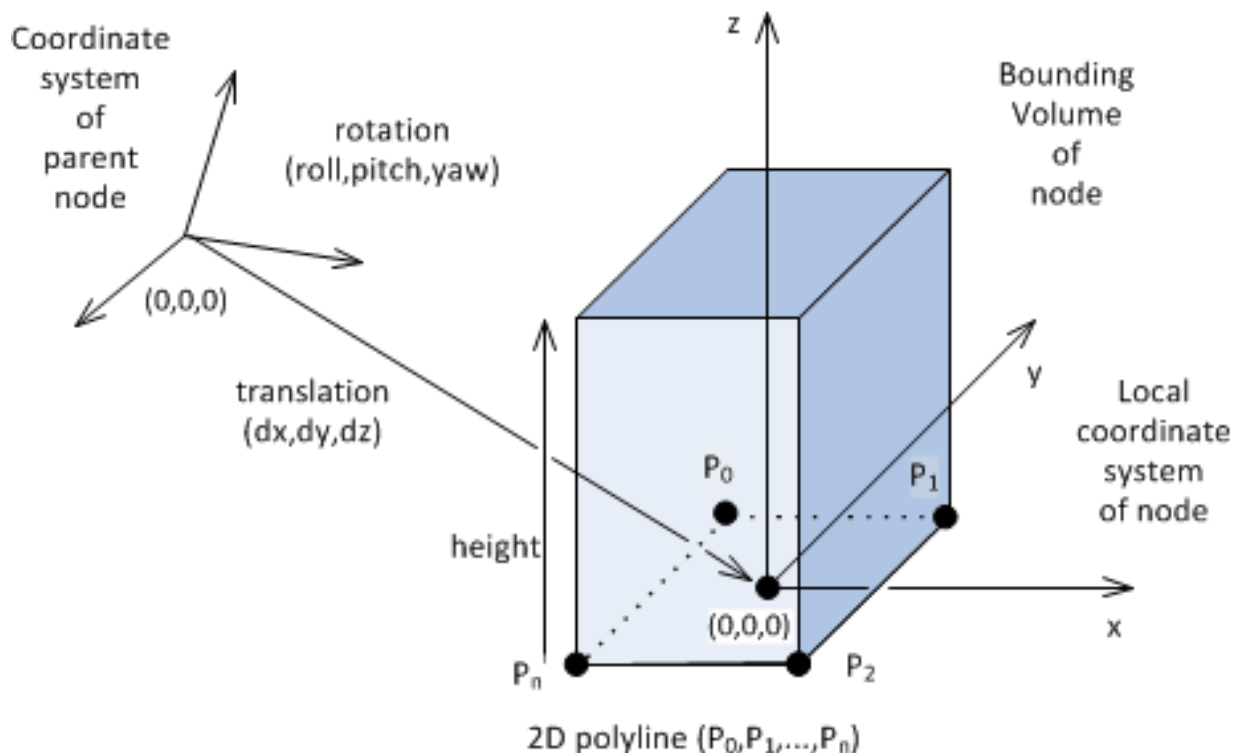


Figure 3 – World Model: bounding volume.

As some of the physical objects have a front face (e.g., the racks and large boxes) or a top face (e.g., the small boxes), by convention, the front face of every physical object is defined as the outer face obtained by extruding the first two points in the polyline (P_0 and P_1 , in Figure 3).

Most of the nodes in the graph represent space in the form of a given volume. Any two volumes may be related by a translation vector and by three rotations (roll, pitch and yaw). The model allows

a given volume to be empty (i.e. its closed polyline is not defined) so that in fact the node does not represent any physical object but is merely used to accomplish a transformation between any other nodes.

3.3.3 Concrete representation

The data format of the world model is specified through the UML class diagram presented below in Figure 4.

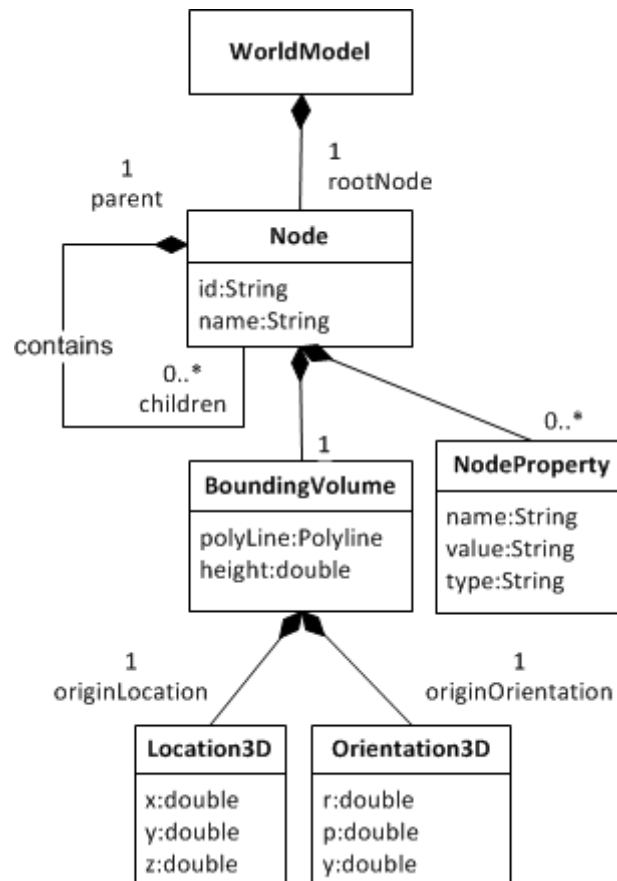


Figure 4 – World Model: concrete representation (UML class diagram).

Each node in the world model has the following attributes:

- **parent** - identifies the parent node (by definition, the root node has no parent node);
- **children** - specifies all the children nodes;
- **name** - identifies the type of physical object represented by the node;
- **id** - uniquely identifies the physical object;
- **nodeProperties** - list of specific properties of the represented physical object; each *NodeProperty* is modelled as a *(name,value,type)* tuple.
- **boundingVolume** - the 3D volume occupied by the physical object.

Taking the model of a rack as an example, a specific instance of the previous class diagram is presented in Figure 5. In this particular case, node 1 represents the entire kitting zone, node 2

represents the entire rack and nodes 3 and 4 the docking station and one level in the rack. This level is further decomposed into one cell containing a small box and a docking station.

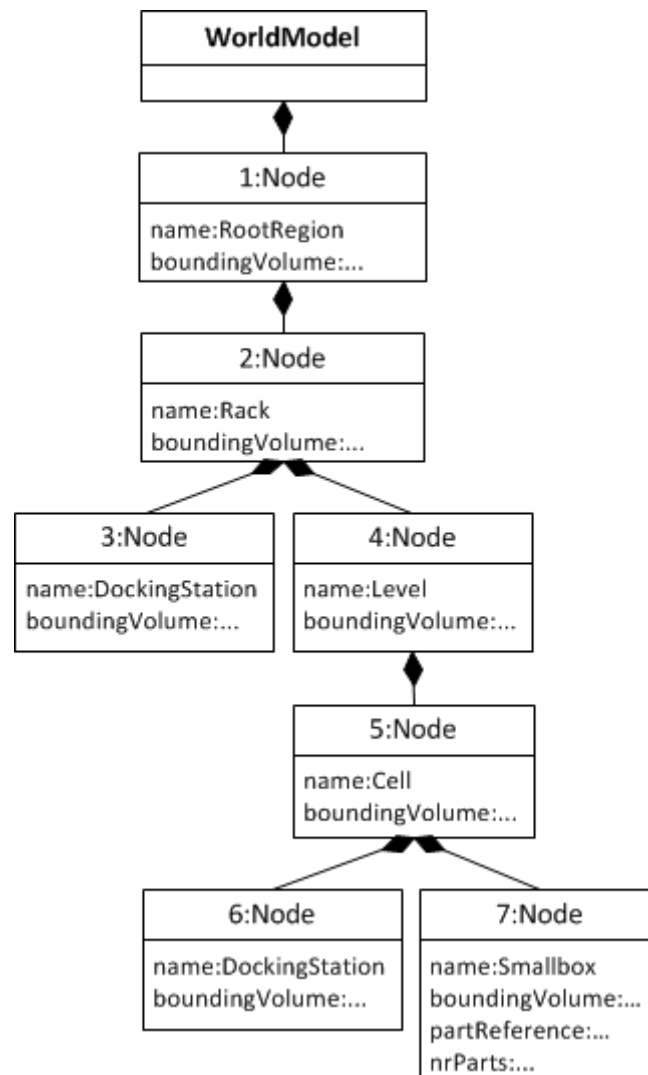


Figure 5 – World Model: instance with rack and small box (UML object diagram).

3.3.4 Physical objects

The object categories (types of physical objects) defined so far in the Logistic Model are defined in the UML class diagram of Figure 6 (each object class is represented by a rectangle with a title (e.g., *PhysicalObjectType*) and a set of attributes (e.g. *id* and *objectTypeId*); inheritance, composition and normal associations are represented). The classes are used to represent the types of physical objects that may be present in a kitting area for parts, racks, small boxes, large boxes, conveyors, kits and robots. They are named respectively as *PartType*, *Racktype*, *SmallboxType*, *LargeboxType*, *ConveyorType*, *KitConfig* and *RobotType*. All these class objects are subclasses of *PhysicalObjectType* thus inheriting all the attributes defined at the *PhysicalObjectType* level (this models the common attributes between the classes). Racks and large boxes have an internal hierarchical structure, supported by the *contains* relationship at the *PhysicalObjectType* class:

- a rack is modelled as a physical object having levels and in each level, space regions (cells) where small boxes are kept;

- a large box is modelled as a physical object having levels and in each level, space regions (cells) where single Parts are kept.

In this context, the *Location3D* class is used to specify the origin location of contained element relative to its container element as a three-dimensional translation vector.

The *KitConfig* class models the internal structure of each type of kit that can be built in the kitting area (cells, nr. or parts per cell, assignment of parts to cells). Even though PSA only has one single type of kits (only one kit configuration) the Logistic Planner allows one to model more than one type of kit configuration for other type of EIS.

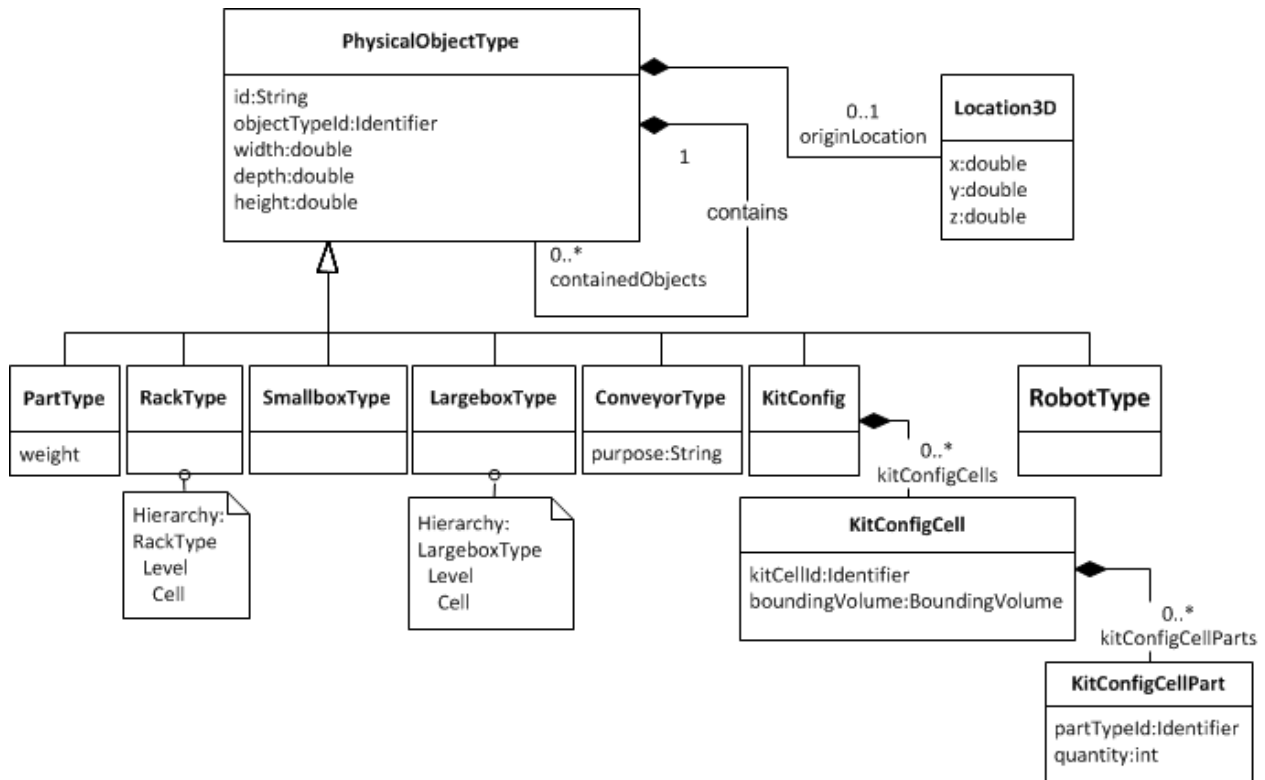


Figure 6 – World Model: types of physical objects (UML class diagram).

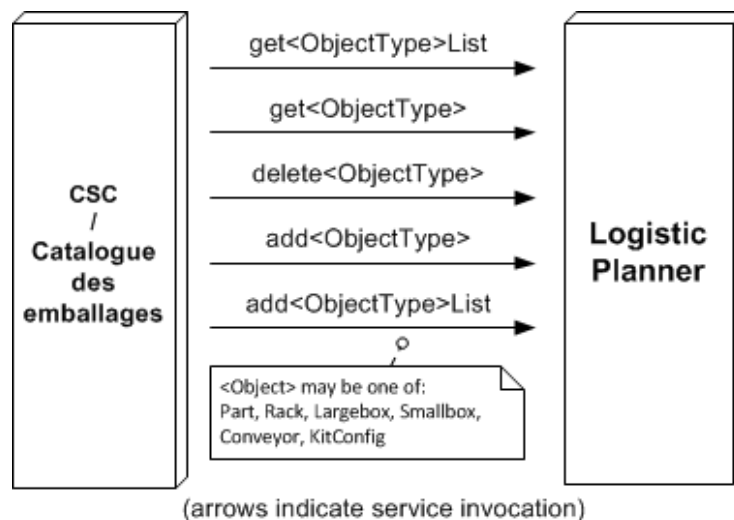


Figure 7 – World Model: EIS integration services for managing types of physical objects.

Interactions between the Logistic Planner and EIS allow the later to indirectly create these class objects (see Figure 7). For each type of physical objects, a set of RESTfull web services are provided by the Logistic Planner. Their invocation by EIS causes the Logistic Planner to create the corresponding objects in its internal repository. See Deliverable 4.2.2 for a detailed description of these integration transactions and services.

Figure 8 identifies the UML class diagram of the objects created inside the Logistic Planner and that represent the corresponding physical objects available in the kitting zone. In other words, for each physical object available in the kitting zone a corresponding model must be present in the Logistic Planner. These objects and their relationships comprise the world model and its setup is accomplished by following the methodology described in a later section of the report.

The classes *KittingRegion*, *Robot*, *Kit*, *Rack*, *Largebox*, *Smallbox* and *Conveyor* are defined as subclasses of *PhysicalObjectInstance*, that is, for each physical object a corresponding one in maintained. As there is no need to represent individual instances of Parts, the class *Part* is defined as a subclass of *PhysicalObjectInstanceGroup*. In common, these classes have the following properties, defined by the *PhysicalObject* class:

- *id* - Unique identifier of the object, created by the Logistic Planner;
- *objectId* – the Identifier of the object, generated upon the *objectId* of the corresponding type of object;
- *objectTypeId* – the identifier of the corresponding object type (generated by the Logistic Planner);
- *boundingVolume* – the bounding volume of the corresponding physical object;
- *specificProperties* – a list of specific properties of the object (e.g. *partReference* on small boxes, large boxes and kit cells).

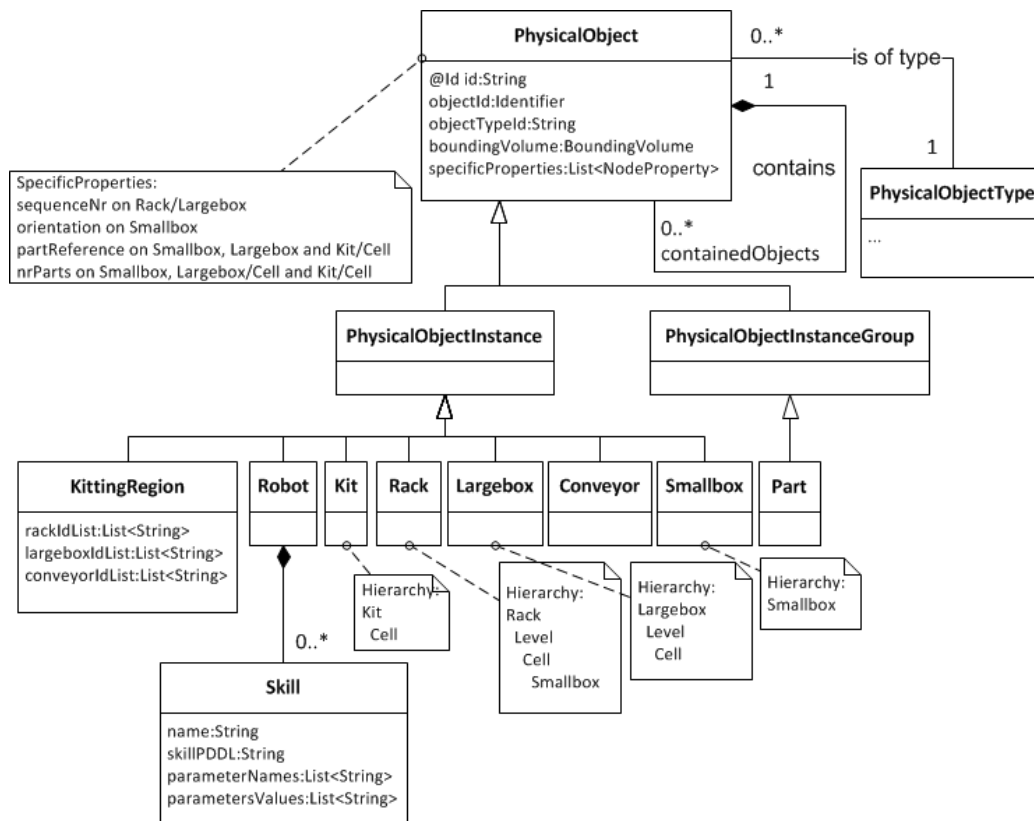


Figure 8 – World Model: physical objects (UML class diagram).

Furthermore, all objects may have a *contains* relationship, to express the composition of objects in space (e.g., a rack contains levels, a level contains cells and a cell contains a small box).

For each available robot in the system, the Logistic Planner creates an instance of the *Robot* class keeping a list of their skills, after querying the robot for such information. This is accomplished through the *getSkills* service (see Figure 9). Deliverable 4.1.1 version 1.2 details the service.

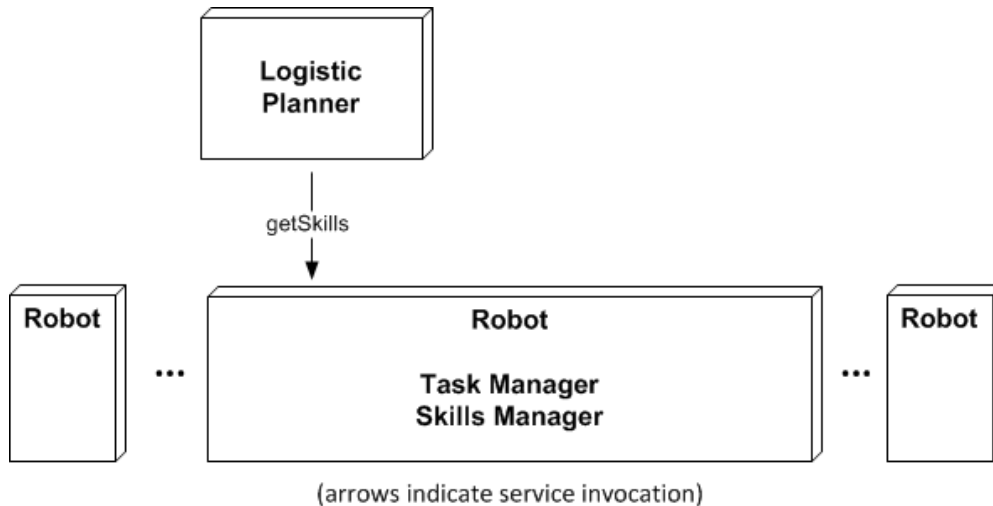


Figure 9 – World Model: service for retrieving skills of a robot.

The Logistic Planner’s user interface provides a screen to show the skills as retrieved from each robot. See Figure 10.

Robot II		Skills	Actions
/stamina_robot		drive - (.action drive .parameters (?containerToApproach - partcontainer ?arm - arm ?from - partcontainer) .precondition (and (arm-at ?arm ?from)) .effect (and (not (arm-at ?arm ?from)) (arm-at ?arm ?containerToApproach) (increase (total-cost) 10))) locate - pick - (.action pick .parameters (?part - part ?arm - arm ?pc - partcontainer ?tl - location) .precondition (and (arm-at ?arm ?tl) (container-at ?pc ?tl) (in-partcontainer ?pc ?part) (empty-handed ?arm)) .effect (and (arm-holding ?arm ?part) (not (empty-handed ?arm)) (increase (total-cost) 1))) place - (.action place .parameters (?placingLocation - kittingbox ?arm - arm ?p - part) .precondition (and (arm-carrying ?arm ?placingLocation) (arm-holding ?arm ?p) (not (in-kittingbox ?placingLocation ?p)) .effect (and (not (arm-holding ?arm ?p)) (empty-handed ?arm) (in-kittingbox ?placingLocation ?p) (increase (total-cost) 1)))	<div> </div>

Figure 10 – User interface: skills implemented by a robot

3.3.5 Publishing and real-time updating

The world model is dynamically provided to the interested parties (Mission Planner and TaskManager on each robot) throughout the *setWorldModel* and *getWorldModel* (see Figure 11). The specification of all the physical objects available on the kitting zone allow the Mission Planner to define and assign missions to a given robot in the fleet and the Task Manager to execute the tasks in a mission throughout its Skills Manager.

Actions having a physical effect on the kitting zone (e.g. a Part is taken from a small box) cause the robot to generate notifications informing the Logistic Planner of changes in the world model. This is accomplished through the *updateWorldModel* service. Additionally, this same transaction may be used by the robot to inform the Logistic Planner of adjustments in the coordinates that specify the location of any physical object in the kitting zone. By definition, the location and dimensions of physical objects may not be exact (see next section). The sensors that equip the robot may detect this discrepancy (e.g. a large box is detected to be 5 centimetres to the right of the position defined

in the world model). The *updateWorldModel* transaction should be used to adjust the world model to the sensed reality.

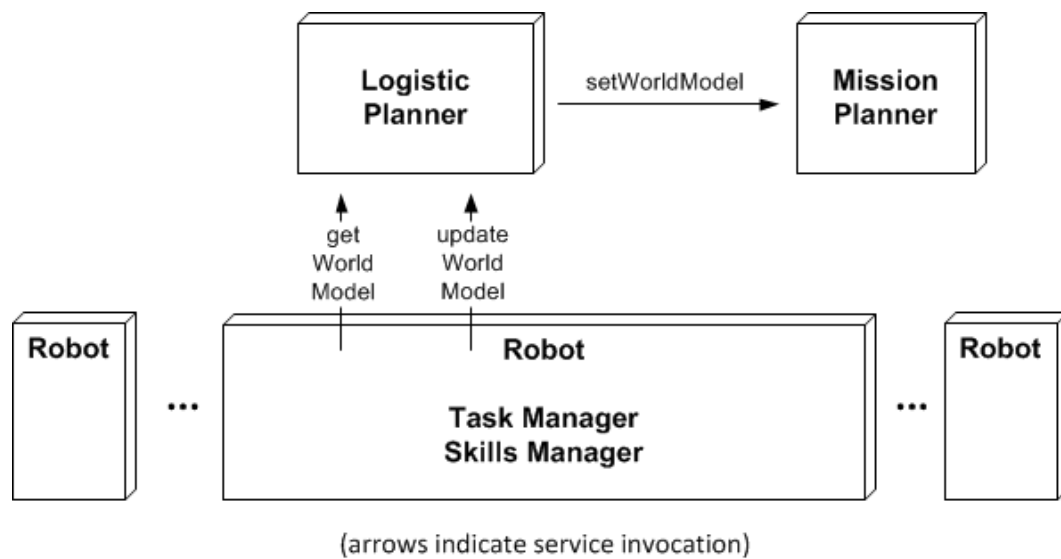


Figure 11 – World Model: runtime services.

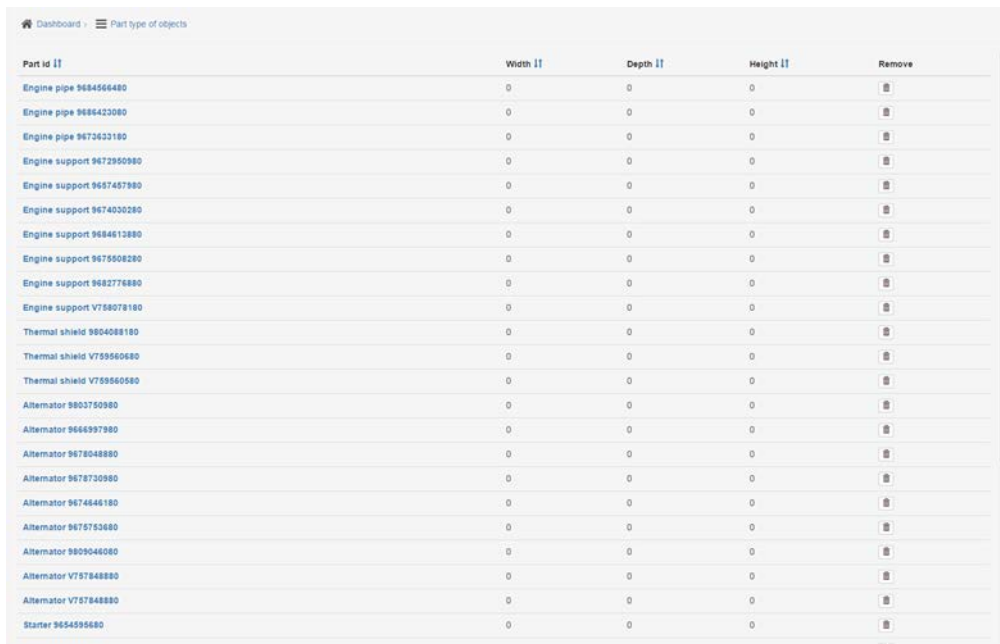
3.3.6 Set up

This section describes the methodology proposed for constructing the world model, i.e. the steps to follow in order to model the logistic area. The final goal of this set up is to have a concrete representation of each physical object present in the kitting area, specifying their characteristics and their 3D locations. This is achieved in two phases.

In the first set up phase, all types of object types are defined. This is accomplished by interacting with EIS (see Figure 6 and Figure 7) and the outcome is identification of the following properties for racks, large and small boxes, conveyors, parts and kits:

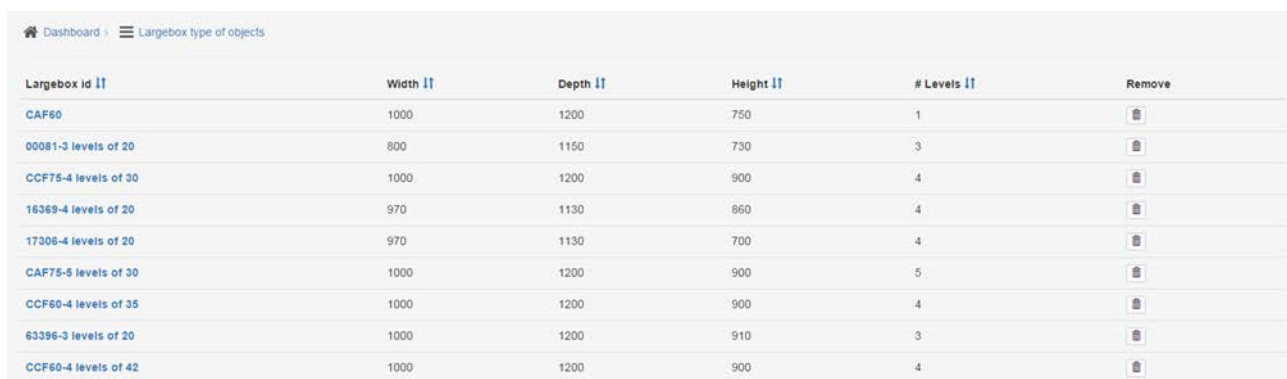
- system identifier;
- textual friendly name;
- width, depth and height of the object's volume;
- and internal organization in terms of levels and cells within each level (a rack is organized in levels and each level can have small boxes of a given type; a large box is also organized in levels and each level organized in cells containing usually an instance of a given part; a kitting box is organized in compartments to keep parts).

This information can be checked in the Logistic Planner, in its “Type of objects” area (see the following two examples - Figure 12 and Figure 13).



Part id	Width	Depth	Height	Remove
Engine pipe 9684566480	0	0	0	
Engine pipe 9686423080	0	0	0	
Engine pipe 9673632180	0	0	0	
Engine support 9672950980	0	0	0	
Engine support 9657457980	0	0	0	
Engine support 9674030280	0	0	0	
Engine support 9684612880	0	0	0	
Engine support 9675508280	0	0	0	
Engine support 9682776880	0	0	0	
Engine support V758078180	0	0	0	
Thermal shield 9804088180	0	0	0	
Thermal shield V759560680	0	0	0	
Thermal shield V759560580	0	0	0	
Alternator 9803750980	0	0	0	
Alternator 9666997980	0	0	0	
Alternator 9676048880	0	0	0	
Alternator 9678730980	0	0	0	
Alternator 9674848180	0	0	0	
Alternator 9675753680	0	0	0	
Alternator 9809046080	0	0	0	
Alternator V757848880	0	0	0	
Alternator V757848880	0	0	0	
Starter 9654595680	0	0	0	

Figure 12 – User interface: part types.



Largebox id	Width	Depth	Height	# Levels	Remove
CAF60	1000	1200	750	1	
00081-3 levels of 20	800	1150	730	3	
CCF75-4 levels of 30	1000	1200	900	4	
16369-4 levels of 20	970	1130	860	4	
17306-4 levels of 20	970	1130	700	4	
CAF75-5 levels of 30	1000	1200	900	5	
CCF60-4 levels of 35	1000	1200	900	4	
63396-3 levels of 20	1000	1200	910	3	
CCF60-4 levels of 42	1000	1200	900	4	

Figure 13 – User interface: large box types.

In the second set up phase, the goal is to identify which objects actually exist in the kitting area, which type of containment relationship relate some of the objects and where they are located in space (i.e., where the objects are implanted). This modelling work is done by the user (i.e. the Kitting Technician) within the Logistic Planner in the following way:

1. A 2D image of the kitting area is provided to the Logistic Planner. Two types of images can be used:
 - a. A 2D image created by the robot through its Simultaneous Localization and Mapping (SLAM) functionality where patterns of dots will visually indicate the technician the location and orientation of each physical object in the logistic area.
 - b. A 2D image created by a CAD application also visually specifying the location and orientation of each object in the logistic area.

In both cases, the location of racks and large boxes is easily deduced and these areas will allow the user to subsequently position instances of racks and large boxes. However, before going in that direction, the reference point of the map must be specified, i.e., the point with coordinates (x,y) equal to (0,0). Also, for the specific case of a SLAM-based image, a horizontal line may be

specified by selecting two points, allowing the drawing later on to be horizontally aligned in the user interface. The coordinate system of the robot and of the Logistic Planner Are assumed to be the same (x increases to the right, y increases to the top). Figure 14 identifies a SLAM-based image with a reference point in the top left and the two points that define the horizontality of the image.



Figure 14 – User interface: reference point and horizontal line on a SLAM-based image.

2. Having the 2D image horizontally aligned in the screen, the user specifies the area in it that corresponds to the whole kitting area (root region). Two types of region are subsequently defined: navigation regions, specifying the navigation areas for the robot fleet, and a set of container regions specifying the areas where racks and large boxes are located. This is shown in Figure 15 (with one navigation and two container regions).

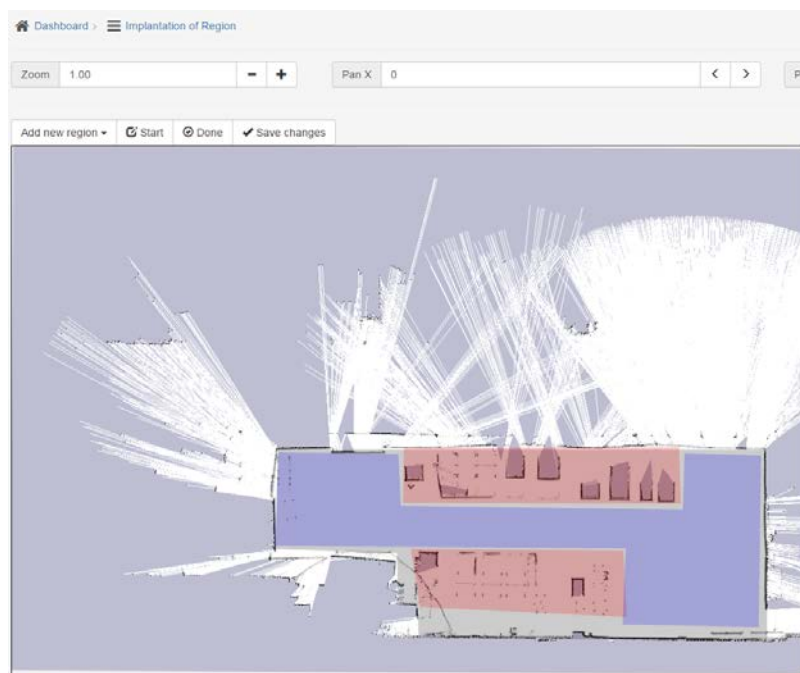


Figure 15 - User interface: specification of container and navigation regions.

3. An iterative process now begins where the user, according to the reality, instantiates objects representing the ones available in the kitting area (i.e., racks and large boxes) and locates them in the proper place in the logistic map. Figure 16 shows the implantation of five racks and two large boxes.

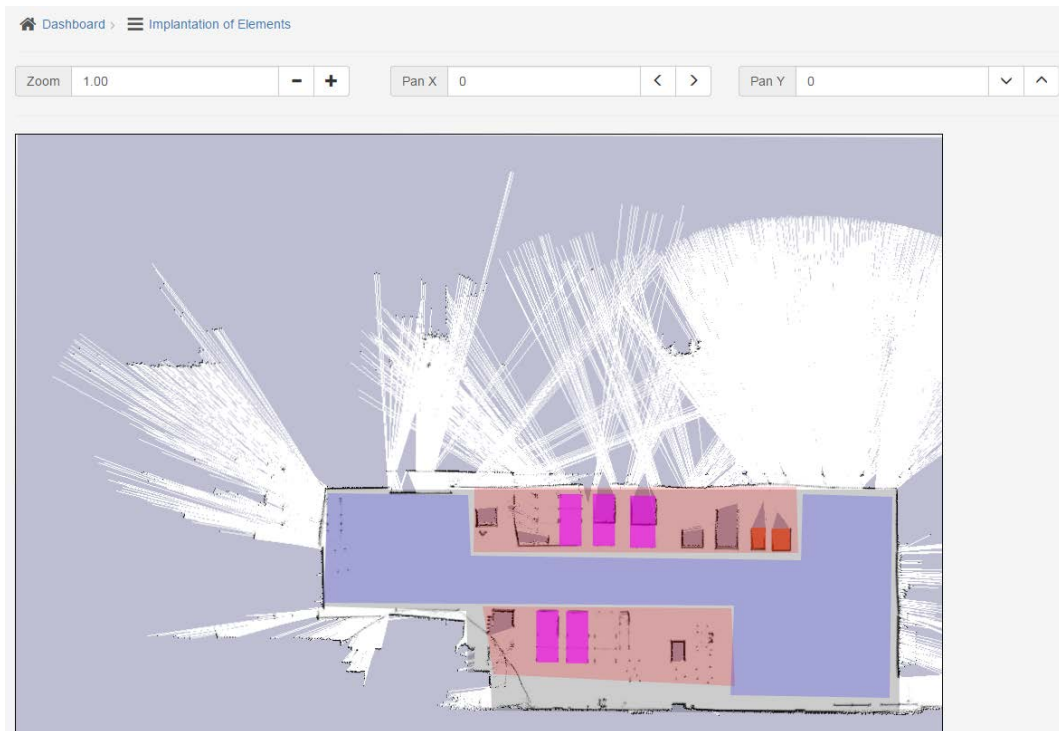


Figure 16 - User interface: implantation of racks and large boxes.

4. The final step in the implantation process, allows the user to specify the following containment relationships:

- what parts are contained by each small box (see Figure 17);

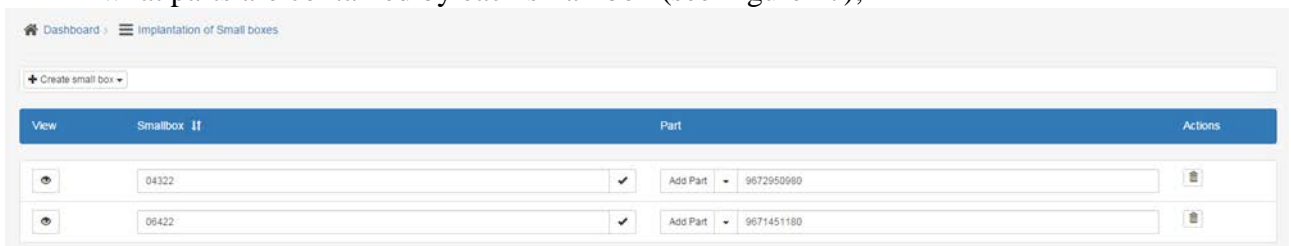


Figure 17 - User interface: association of parts to small boxes.

- what parts are contained by each large box (see Figure 18);

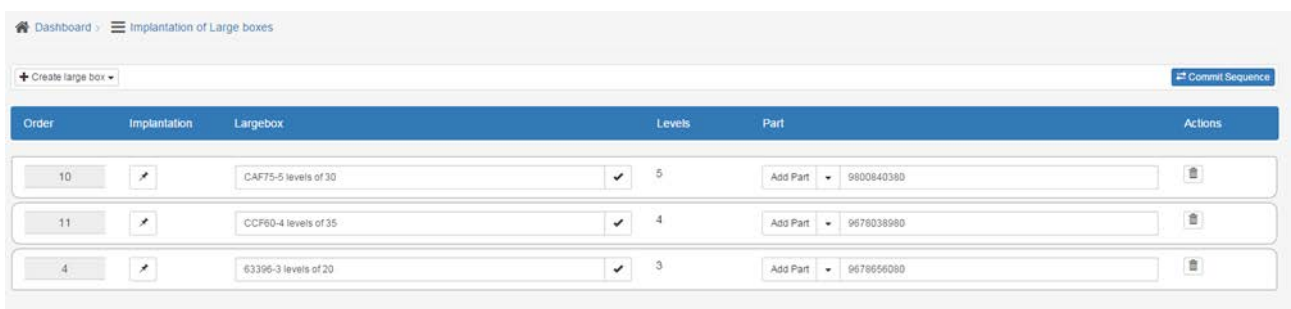


Figure 18 - User interface: association of parts to large boxes.

- and on which rack each small boxes is located (see Figure 19); in this case, small boxes are located in the proper level of the rack and they can be physically placed as transversal or longitudinal.

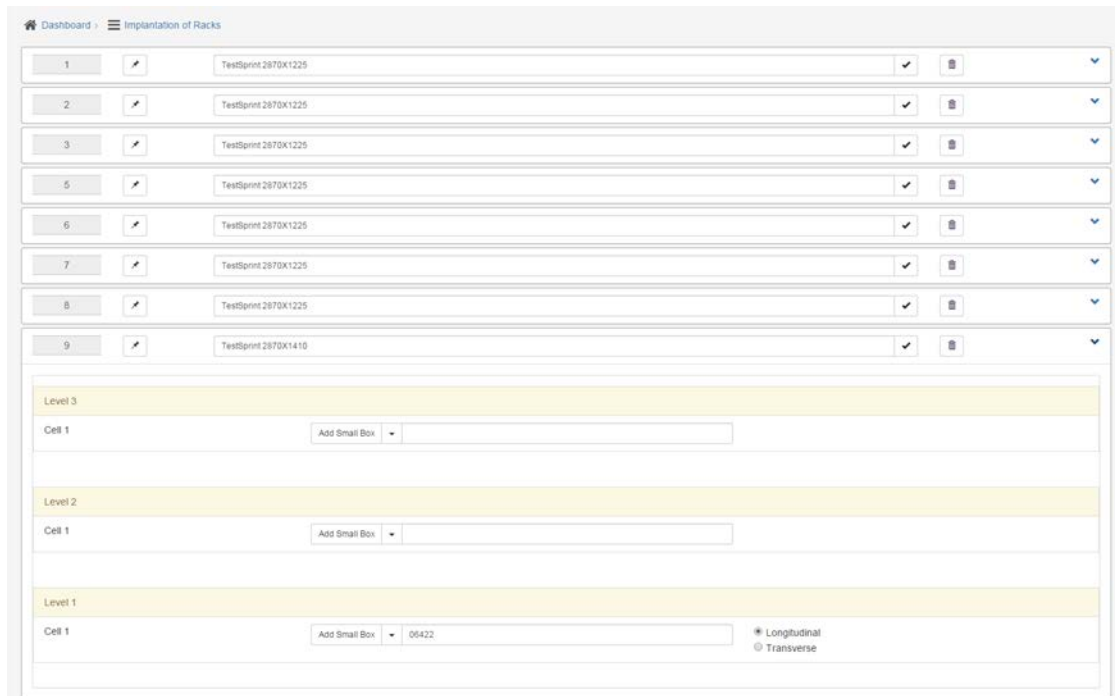


Figure 19 - User interface: association of small boxes to racks.

The following Figure 20 shows a 3D view of the modelling done within the Logistic Planner. Nine racks and three large boxes (orange colour) comprise the main objects. A small box containing parts was inserted on the right-most rack and each large box does contain a number of parts.

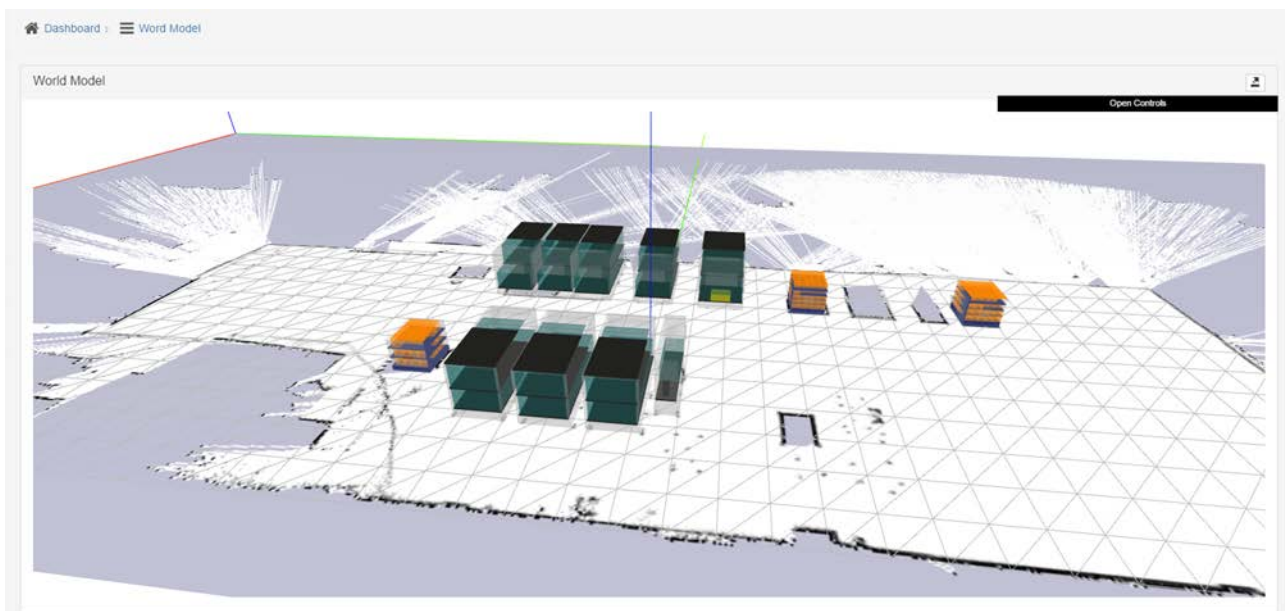


Figure 20 – User interface: 3D view of the logistic world model.

3.4 Order Manager

3.4.1 Introduction

The task of dispatching a list of kitting orders (or list of FACs) created by the external EIS systems is the key driver of STAMINA. This involves the creation of a mission, its assignment to a given robot in the fleet, the planning of the skill sequences that are required to achieve a given mission and the execution of the plan by the robot by translating skills into low-level actions. These major functions are described and accessible in the form of services and constitute the **runtime services** of STAMINA. This is achieved by the Order Manager component within the Logistic Planner and a Runtime Monitor user interface provides real-time information to the Kitting Technician.

3.4.2 FAC and Kitting Orders

As described in previous deliverables (Deliverables 4.2.1 and 4.2.2), a FAC object (“fiche d’accompagnement”) corresponds to the printed paper given to the kitting operator telling him what to do in order to build a kit. The FAC is created by PSA’s EIS 15 to 20 minutes before the corresponding vehicle enters the assembly line (a FAC is related to only one vehicle). The pool of FACs is maintained and transmitted to the Logistic Planner in the form of *eFAC* or *Kitting Order* objects. The model of these objects is shown on Figure 21 in the form of a UML class diagram. As described in Deliverable 4.2.2, the Logistic Planner deals primarily with Kitting Orders. These objects may be injected directly into the Logistic Planner or may be created following the reception of *eFAC* objects. In other words, *eFAC* objects are transformed into Kitting Order objects in order to provide a generalization able to be exploited in other types of non PSA EIS.

By definition, a kitting order is an instruction telling the system which kit is to be built and which parts are to be placed on which cells within the kit. It also specifies the car that will receive the kit. As such, a kitting order has the following properties (Figure 21):

- *id* – unique identifier generated by the Logistic Planner;
- *kittingOrderId* – external Identifier of the order, composed by two fields:
 - *systemId* – system identifier;
 - *friendlyName* – corresponding textual name, to be used on the user interface;
- *carSeqNumber* – sequence number of the car, in the assembly line, related to this kitting order; identifies the car that will get the kit after its completion;
- *kittingZoneId* – Identifier of the kitting zone where the kit is to be built (there can be several kitting zones doing the construction of kits that subsequently are transported to the car assembly line);
- *kitTypeId* – Identifier of the type of kit to be built. At PSA, only one type of kit is possible to build. However, the Logistic Planner supports several types of kits in order to have greater flexibility for more elaborated kitting zones;
- *kitId* – Identifier of the kit to be built. Following the design guideline of separating objects instances from object types, the *kitId* identifies the kitting box that will be manipulated by the robot.
- *missionId* – system identifier of the mission to execute on the robot (see next section for details).

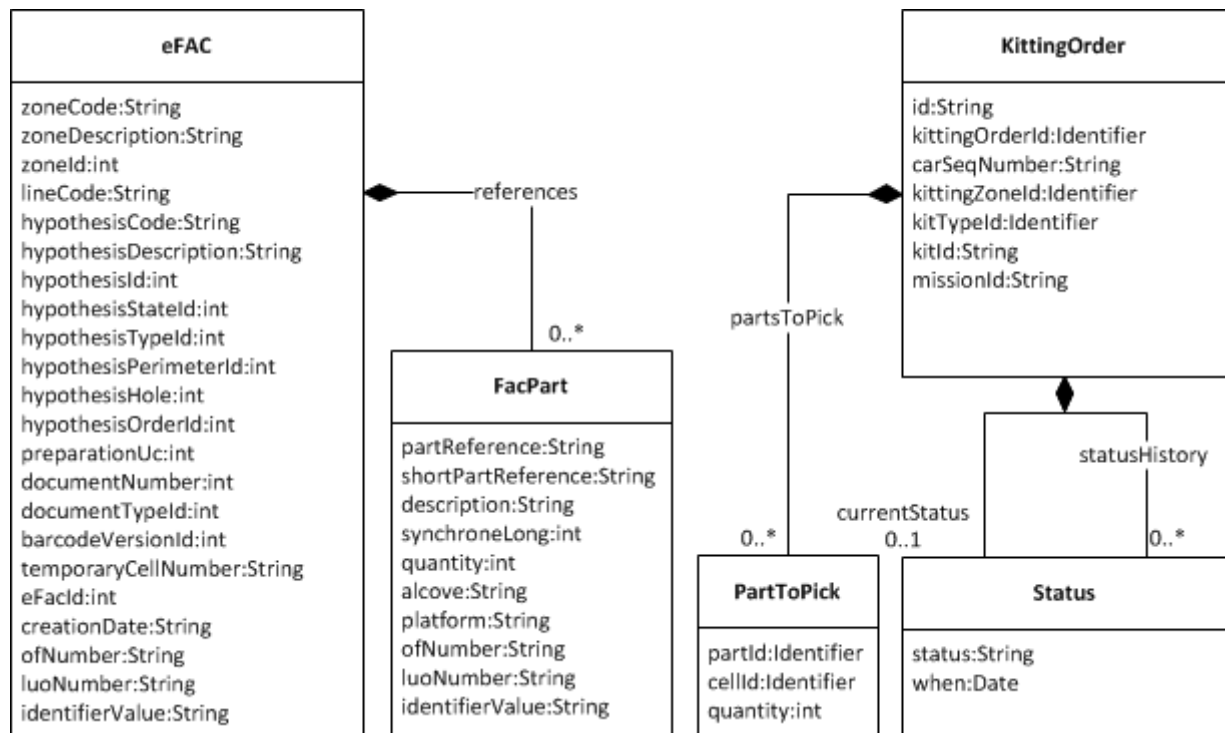


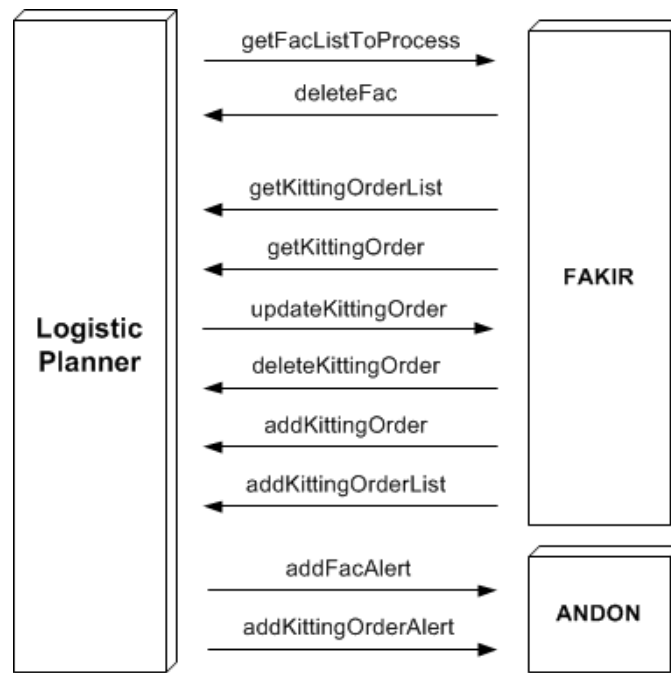
Figure 21 – Order Manager: eFAC and Kitting Order (UML class diagram).

The *PartToPick* class in the diagram specifies the number of Parts (*quantity* and *partId* properties) to be placed in a given cell (*cellId*) of the kit. The *partsToPick* relationship identifies the parts that should be placed in the kit.

The current status and time history of Kitting Orders are kept in the *currentStatus* and *statusHistory* properties. The *status* property may assume the following values: not planned, mission planned, mission started, mission executing, mission ended (successfully/unsuccessfully). More detailed status may be created in the near term (for the third STAMINA test sprint).

eFAC objects and Kitting Order objects are injected in the Logistic Planner through RESTfull web services (see Figure 22). These integration services were described in deliverable 4.2.2. Additionally, the services related with Kitting Orders are used by the user interface of the Logistic Planner, namely by the Runtime Monitor:

- Figure 23 shows the screen allowing the Kitting Technician to manually import JSON files containing Kitting Orders objects (thus simulating the creation of a pool of Kitting Orders).
- Figure 24 shows the history status (including the current status) of a kitting order.



(arrows indicate service invocation)

Figure 22 – Order Manager: EIS integration services.

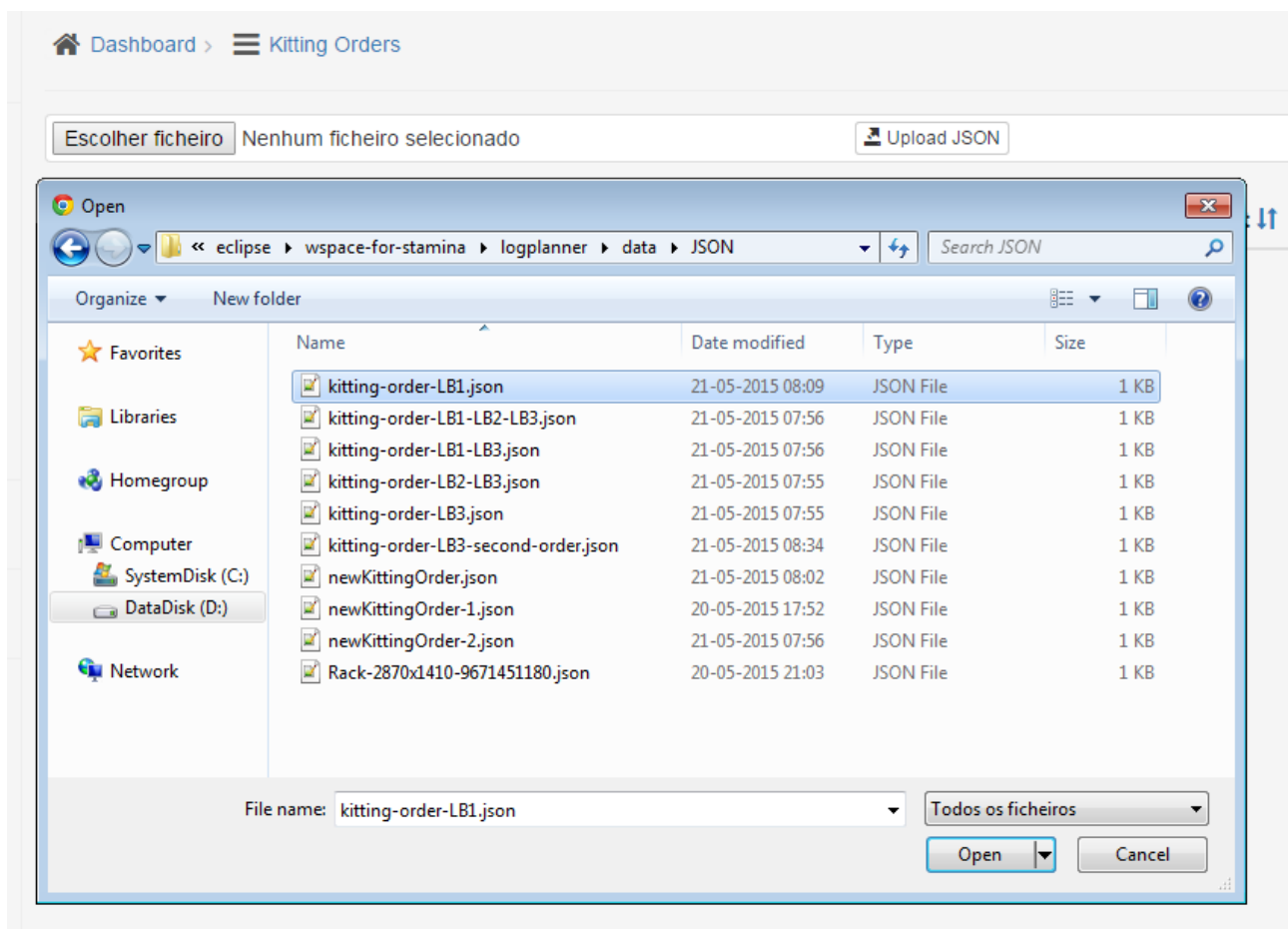


Figure 23 – User interface: importing kitting orders from external files.

Select	Kitting Order	Car Seq. Number	Kit	Status History	Parts To Pick	Actions
<input type="checkbox"/>	order1	272	Kit 2 - 557eec2b08f82203cd8f9fe3	Created - 4:15PM Sent to MP: ok - 4:16PM Sent to MP: ok - 4:17PM Sent to MP: ok - 4:18PM Planned/Added to mission. - 4:18PM	9678656080 / CellB / 1	

Figure 24 – User interface: status of a kitting order.

3.4.3 Mission and Skills

Skills and missions are key concepts within STAMINA. A skill is defined as a primitive based description of compliant robot motions (action primitives), described as simple, atomic movements that can be combined to form a task (see Deliverable D3.2). A mission is defined as the identification of the skills that the robot will use to fulfil one or two Kitting Orders (as the STAMINA robot is able to carry and consequently build two kits). The corresponding class objects in the Logistic Planner are shown in Figure 25.

A Mission object has the following properties:

- *id* – unique identifier, generated by the Logistic Planner, of the mission;
- *KittingOrderIdList* – list of kitting order identifiers. According to STAMINA robot features, a mission fulfils one or two kitting orders;
- *robotId* – the identifier of the robot to carry on the mission;
- *currentStatus* – the current status of the mission (*created*, *assigned*, *execution_started*, *execution_finished*, *execution_cancelled*);
- *statusHistory* – the history of all status changes in the mission;
- *plan* – the plan to be carried over by the robot (see above section 3.4.4 for details).

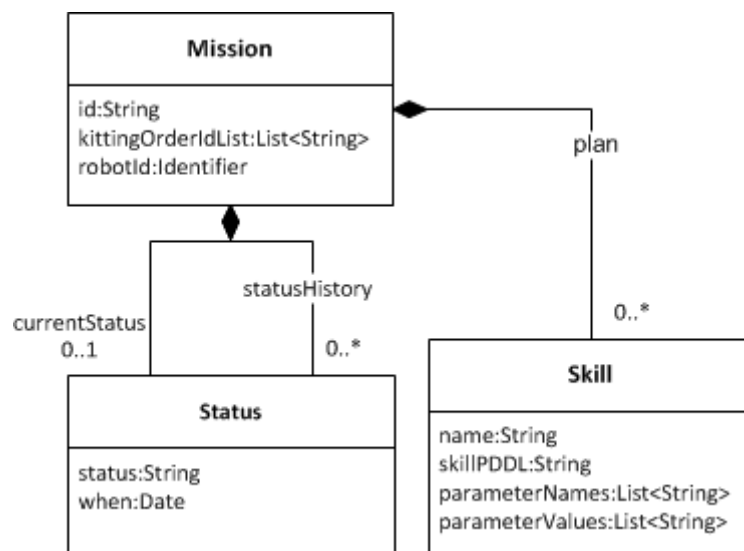
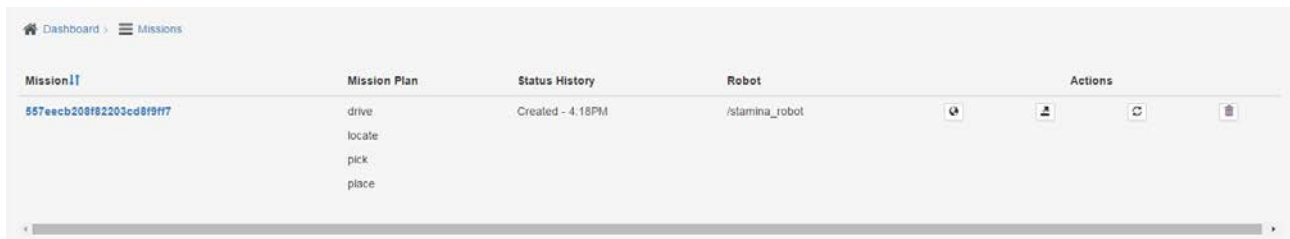


Figure 25 – Order Manager: mission and skill (UML class diagram).

As described in the previous chapter 3.3 “Logistic World Model”, skills are dynamically retrieved from each robot in the fleet and subsequently inserted in the World Model as nodes under the Robot node. A Skill object has the following properties:

- *name* – name of the skill.
- *skillPDDL*, *parameterNames* and *parameterValues* – the PDDL (Planning Domain Definition Language) program and corresponding parameters (see above section 3.4.4 for details).

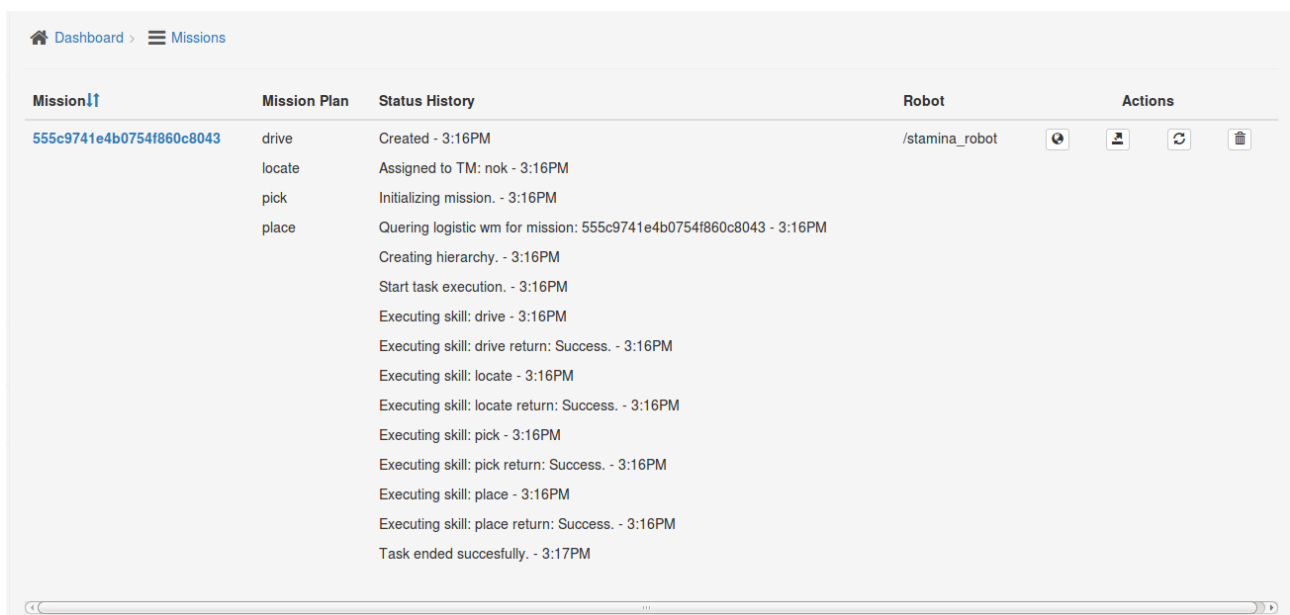
Mission objects are created by the Mission Planner component for fulfilling a set of Kitting Orders and later on provided to the selected robot for execution. A specific screen on the Logistic Planner’s user interface identifies the creation and progress of missions.



The screenshot shows a web application with a 'Dashboard' and 'Missions' menu. The 'Missions' page displays a table with the following data:

MissionID	Mission Plan	Status History	Robot	Actions
557eeeb208f82203cd8f9f7	drive locate pick place	Created - 4:18PM	/stamina_robot	[Refresh] [Assign] [Refresh] [Delete]

Figure 26 – User interface: identification of a mission just after its creation.



The screenshot shows the same web application, but the 'Status History' column for the mission 555c9741e4b0754f860c8043 is expanded, showing a detailed log of events:

MissionID	Mission Plan	Status History	Robot	Actions
555c9741e4b0754f860c8043	drive locate pick place	Created - 3:16PM Assigned to TM: nok - 3:16PM Initializing mission. - 3:16PM Querying logistic wm for mission: 555c9741e4b0754f860c8043 - 3:16PM Creating hierarchy. - 3:16PM Start task execution. - 3:16PM Executing skill: drive - 3:16PM Executing skill: drive return: Success. - 3:16PM Executing skill: locate - 3:16PM Executing skill: locate return: Success. - 3:16PM Executing skill: pick - 3:16PM Executing skill: pick return: Success. - 3:16PM Executing skill: place - 3:16PM Executing skill: place return: Success. - 3:16PM Task ended succesfully. - 3:17PM	/stamina_robot	[Refresh] [Assign] [Refresh] [Delete]

Figure 27 – User interface: identification of a mission’s status history.

A set of web services was specified and implemented that link the Logistic Planner with the Mission Planner, the Task Manager and EIS. They are described in detail in Deliverable 4.1.1, version 1.2. These transactions (see Figure 28) are related with the creation of missions, its assignment to robots, its execution and monitoring.

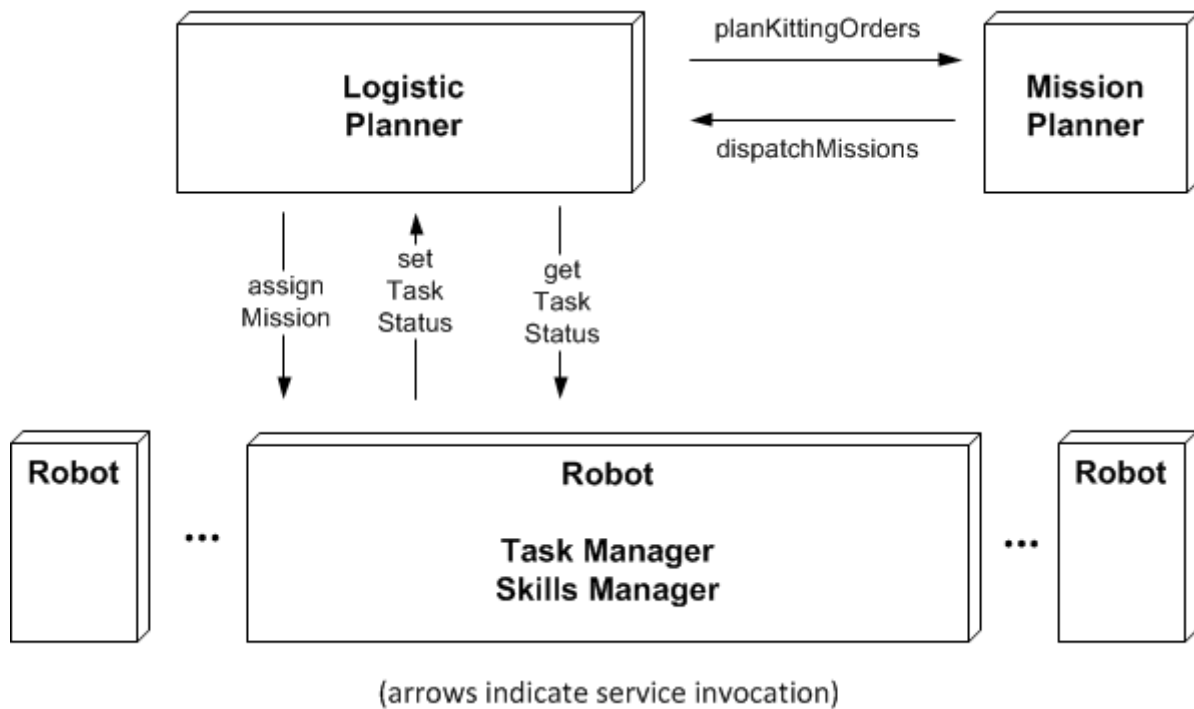


Figure 28 – Order Manager: mission related runtime services.

3.4.4 Planning kitting orders on the Mission Planner

This Section describes the integration of the Mission Planner at the second test sprint. The Mission Planner interacts with the Logistic Planner taking the world model, robot skills and kitting orders as input and returning plans to be assigned to the robots in the fleet (for the second test sprint, only a single robot was used).

The internal processing of the Mission Planner is shown in Figure 29. The skills information is combined with the world model to create a planning domain model described in the standard Planning Domain Definition Language (PDDL). This model includes a description of the possible actions available to the robot (the skills) along with the types of entities that exist in the robot's operating environment. The Mission Planner receives the world model from the Logistic Planner as a JSON string which it then converts to an internal representation suitable for building the planning domain. Only entities that appear in the world model are included in the domain. For example, if the world model contains no instantiated large boxes then the planning domain will not include any large boxes even when such objects exist in the ontology.

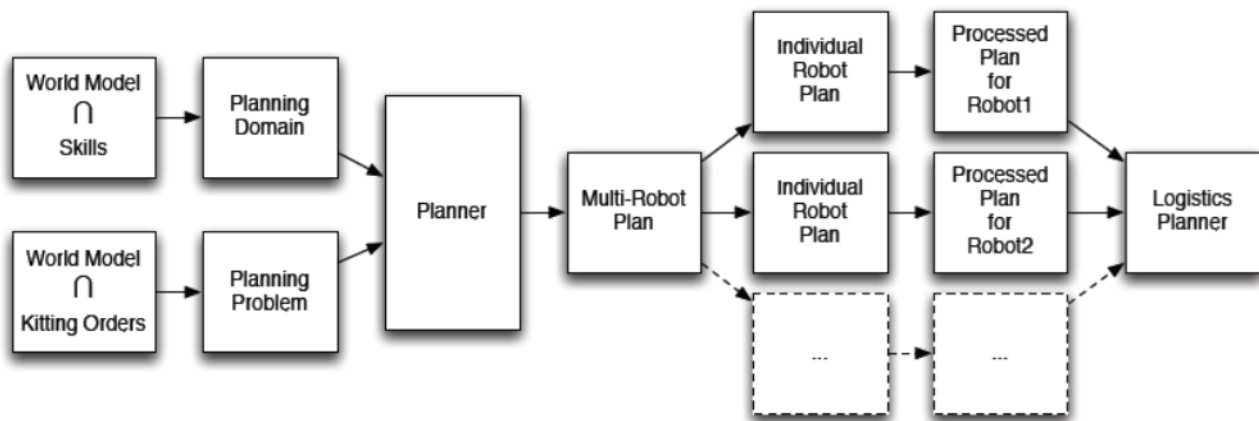


Figure 29 – Mission Planner: an overview of its internal processing

For the test sprint, the available skills when converted into PDDL are as follows:

```

(:action drive-macro
  :parameters (?r - robot ?a - box ?b - box ?x - location ?y - location)
  :precondition (and
    (robot-at ?r ?x)
    (robot-at-box ?r ?a)
    (before ?x ?y)
    (box-at ?b ?y))
  :effect (and
    (not (robot-at ?r ?x))
    (robot-at ?r ?y)
    (robot-at-box ?r ?b)
    (increase (total-cost) 10))
)
  
```

```

(:action drive-micro
  :parameters (?r - robot ?a - box ?b - box ?x - location)
  :precondition (and
    (robot-at ?r ?x)
    (robot-at-box ?r ?a)
    (box-at ?b ?x))
  :effect (and
    (not (robot-at-box ?r ?a))
    (robot-at-box ?r ?b)
    (increase (total-cost) 2))
)
  
```


)

(:action pick

:parameters (?r - robot ?p - part ?b - box ?x - location)

:precondition (and

(robot-at ?r ?x)

(robot-at-box ?r ?b)

(box-at ?b ?x)

(empty-handed ?r)

(in-box ?p ?b))

:effect (and

(holding ?r ?p)

(not (empty-handed ?r))

(increase (total-cost) 1))

)

(:action place

:parameters (?r - robot ?p - part ?c - cell ?k - kit)

:precondition (and

(holding ?r ?p)

(carrying ?r ?k)

(fits ?p ?c)

(cell-component ?c ?k))

:effect (and

(in-kit ?p ?k)

(not (holding ?r ?p))

(empty-handed ?r)

(increase (total-cost) 1))

)

Note that at the planning level, the Drive skill is partitioned into two separate skills, drive-micro and drive-macro, for use by the Mission Planner. The drive-micro skill is used to reposition the robot between two boxes that are at the same location (e.g., on the same rack), while the drive-macro skill is used to drive the robot between large objects that have different locations in the navigation waypoint map. This allows for the repositioning of the robot between grasps in order to retrieve parts from nearby containers. After a plan has been generated by the Mission Planner, all occurrences of the drive-micro and drive-macro actions in the plan are converted back to the Drive skill before being returned to the Logistic Planner. As the kitting orders used in the test sprint only

included large boxes, the drive-micro skill was not used in any of the planning problems. However, it has proven useful during internal testing of larger planning domains.

Kitting orders provide the goals for the Mission Planner's search algorithm: each part in a kitting order is added as a goal for the appropriate kit. For instance, an example goal might be:

```
(:goal (and
      (in-kit part1 kit1)
      (in-kit part2 kit1)
      (in-kit part2 kit2)
      (in-kit part3 kit2)
    ))
```

which represents a conjunction of facts such that kit1 will contain part1 and part2 and kit2 will contain part2 and part3. As with the planning domain, the planning problem only includes elements from the world model that are relevant to the current goals. For example, even if part4 was present in the warehouse it would not be modelled in the current planning problem as it does not appear in any of the kitting orders.

Once the planning domain and planning problem are constructed, they are used as input to the Mission Planner, which returns a plan that achieves the goal (if one exists). In multi-robot instances of the planning problem (not relevant for this test sprint), the generated plan is then separated into individual plans for each robot. There is also a final post-processing step before the plan is returned to the Logistic Planner, which converts any planner-specific names back to STAMINA names, combines the drive-micro and drive-macro actions, and removes parameters unnecessary for skill execution. For instance, the planner might generate the following (post-processed) plan using the example goal described above (i.e., kit1 should contain part1 and part2 and kit2 should contain part2 and part3):

```
DRIVE(robot1, startloc, box1)
PICK(robot1, part1, box1)
PLACE(robot1, part1, cell1, kit1)
DRIVE(robot1, box1, box2)
PICK(robot1, part2, box2)
PLACE(robot1, part2, cell2, kit1)
PICK(robot1, part2, box2)
PLACE(robot1, part2, cell2, kit2)
DRIVE(robot1, box2, box3)
PICK(robot1, part3, box3)
PLACE(robot1, part3, cell3, kit2)
```

3.5 Software implementation

As the Logistic Planner needs to integrate with external EIS systems and interactivity dialogues must be realized with the person responsible for the kitting zone in order to provide logistic

information to the remaining STAMINA components and to monitor the creation, planning and execution of kitting orders, the Logistic Planner was implemented as an autonomous software component, running as a server on top of a Java virtual machine, and providing a Java Script / HTML 5 user interface on standard Internet browsers. Being dependent on the Java 8 virtual machine, the server part run on both the Microsoft Windows (7 and 8) and Linux (Ubuntu) operating systems (see Figure 30).

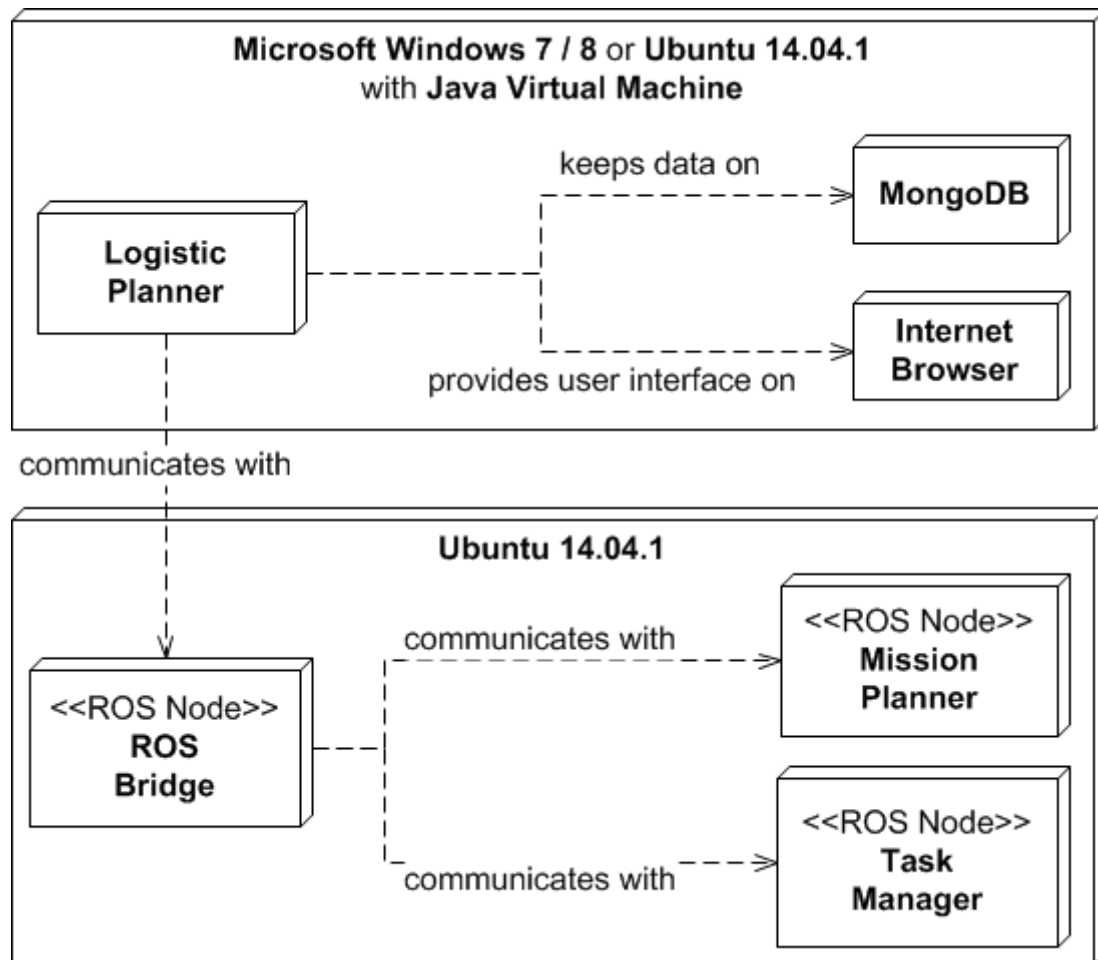


Figure 30 – STAMINA components and ROS nodes (UML deployment diagram).

Its data repository is provided by MongoDB (<http://www.mongodb.org>), a document-oriented data base management system (DBMS). As this DBMS keeps data in documents organized as data hierarchies while simultaneously offering high degrees of agility and scalability, this DBMS was chosen as the data repository of the Logistic Planner.

Communication with the ROS world, where the remaining STAMINA components do run, is provided through ROS Bridge. Details about ROS services, topics, data objects and service registry, are given by Deliverable 4.1.1, version 1.2.

Figure 31 identifies how the Logistic Planner is organized in terms of software implementation. Its user interface runs on HTML 5 compliant browsers as a Java Script application making use of several popular frameworks such as bootstrap (<http://getbootstrap.com/>), angularJs (<https://angularjs.org/>), jQuery (<https://jquery.com/>), requires (<http://requirejs.org/>) and WebGL (<https://www.khronos.org/webgl/>). Communication with the Logistic Planner's server element is achieved through the HTTP protocol by invoking RESTfull web services driven by JSON data

format. Invocation of these web services triggers the activation of Controllers and Actions inside the server, that validate data input and output and map JSON data into Java data. At a lower level, the required functions are encapsulated as Services and the Spring Data provides the mapping of Java objects into data base objects. All usual data base management systems are supported (Microsoft SQL Server, Oracle, Sybase, etc.) but MongoDB is the one selected. The Object Model specifies all the Java objects and relationships corresponding to Kitting Order, Mission, Skill, eFAC, etc. Several UML class diagram in the above chapters 3.3 and 3.4 describe this Object Model.

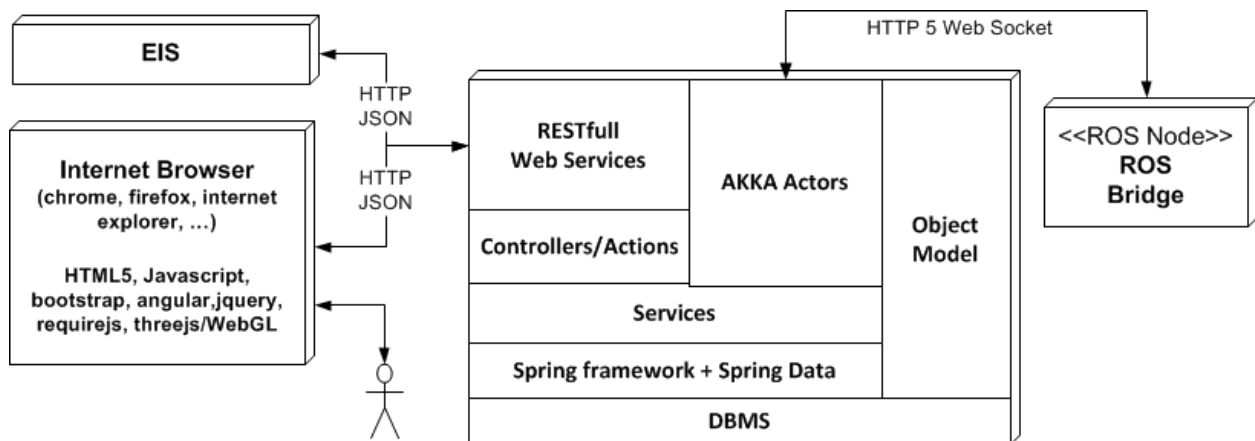


Figure 31 – Software organization within the Logistic Planner.

The reactive part of the server, implemented by the Controllers and Actions, follow the Play Framework (<https://www.playframework.com/>), a high-productivity Java and Scala web application framework integrating the components and APIs currently needed for developing modern web applications. The Play Framework is based on a lightweight, stateless, web-friendly architecture and features predictable and minimal resource consumption (CPU, memory, threads) for highly-scalable applications thanks to its reactive model, based on Iteratee IO. In this context, the Controllers and Actions within the Logistic Planner constitute the elements that are called to respond to RESTfull invocations: GET, POST, DELETE, PUT, HEAD, OPTIONS (see Deliverable 4.2.2 for details). However, apart from responding to RESTfull requests coming from the Internet (from the user interface or EIS), the Logistic Planner needs to have active threads in support of order lifecycle in order to interact with the Mission Planner and Task Manager without any human intervention. This is implemented by AKKA Actors, objects that encapsulate state and behaviour and that communicate exclusively by exchanging messages, having their own light-weight thread. In practice, they are used within the Logistic Model to control the planning of a set of kitting orders by the Mission Planner, their assignment to the Task Managers, and their execution and monitoring, besides other minor functions. As explicated in Figure 31, Controllers/Actions and AKKA Actors use in common the Services provided at a lower level.