Mobile dual arm robotic workers with embedded cognition for hybrid and dynamically reconfigurable manufacturing systems

Grant Agreement No	**:**	**723616**
Project Acronym	**:**	**THOMAS**
Project Start Date	**:**	**1st October, 2016**
Consortium	**:**	**UNIVERSITY OF PATRAS (LMS)** **PEUGEOT CITROEN AUTOMOBILES S.A. (PSA)** **SICK AG (SICK)** **FUNDATION TECHNALIA RESEARCH & INNOVATION (TECNALIA)** **ROBOCEPTION GMBH (ROBOCEPTION)** **DGH ROBOTICA, AUTOMATIZACION Y MANTENIMIENTO INDUSTRIAL SA (DGH)** **AERNNOVA AEROSPACE S.A.U. (AERNNOVA)** **INTRASOFT INTERNATIONAL SA (INTRASOFT)**

Title	**:**	THOMAS Service Oriented Network of Resources – Initial Prototype
Reference	**:**	D5.3
Availability	**:**	Public version
Date	**:**	31/03/2018
Author/s	**:**	INTRASOFT, LMS, TECNALIA, PSA
Circulation	**:**	EU, Consortium

Summary:

This report documents the interfaces, messages and system architecture for creating a service-oriented network of resources available in the THOMAS platform.

Table of Contents

1. LIST OF FIGURES 3
2. LIST OF TABLES 4
3. EXECUTIVE SUMMARY 5
4. INTRODUCTION 6
5. KEY FUNCTIONAL CONCEPTS SPECIFICATION 8
   5.1. SONAR – Service-Oriented Network Adjacent Resources 8
      5.1.1. Data Model 8
      5.1.2. Nodes 8
      5.1.3. Registries 9
6. SONAR ARCHITECTURE 10
   6.1. Prototype Environment 10
      6.1.1. Virtual Machine Specifications 10
   6.2. Resources Management 12
   6.3. Station Agent and Station Controller 15
   6.4. Services for WP2 Integration – Human Robot Interaction mechanisms 16
      6.4.1. Task Take Over Services 16
      6.4.2. Human Gesture/Posture Recognition Service 19
      6.4.3. Object pre-classification based on 2D data – 2D Human Detection Service 20
   6.5. Services for WP3 Integration – Perception for Process Context Awareness 21
   6.6. Services for WP4 Integration – Skill Engine 23
   6.7. Programmatic Interfaces 24
   6.8. Network of services prototype implementation 25
7. CONCLUSIONS 26
8. GLOSSARY 27
9. REFERENCES 28

1. LIST OF FIGURES

FIGURE 1: OVERALL NETWORK OF SERVICES CONCEPT 5
FIGURE 2: THOMAS SONAR CONCEPT 6
FIGURE 3: AGILE DEVELOPMENT APPROACH BETWEEN THOMAS WORKPACKAGE 7
FIGURE 4: ILLUSTRATION OF THE SONAR SYSTEM. NODES AND REGISTRY 9
FIGURE 5: VIRTUAL MACHINE DETAILED SETTINGS 10
FIGURE 6: VIRTUAL MACHINE RUNNING FOR DESKTOP 12
FIGURE 7: THOMAS WORLD MODEL [REF D5.1] 13
FIGURE 8: LAYERED ARCHITECTURE FOR MANAGEMENT OF RESOURCES 14
FIGURE 9: THOMAS WORLD MODEL VISUALIZATION – MRP 1 AND MRP2 15
FIGURE 10: UML SEQUENCE DIAGRAM OF THE TASK TAKE OVER FEATURE 17
FIGURE 11: HUMAN GESTURE/POSTURE RECOGNITION SERVICE 19
FIGURE 12: OBJECT DETECTION WORKFLOW DIAGRAM 20
FIGURE 13: SCREENSHOT FROM THE ROBOCEPTION GUI 21
FIGURE 14: PICK_UP TASK EXECUTION SEQUENCE DIAGRAM 22
FIGURE 15: DAMPER PART DETECTION AND GRASPING 22
FIGURE 16: COMMUNICATION INTERFACE OF STATION CONTROLLER/SKILL ENGINE INTEGRATION 23
FIGURE 17: XML STATION PROGRAM FROM PSA CASE OF STATION CONTROLLER 23
FIGURE 18: XML SKILL DEFINITION OF SKILL ENGINE FOR NAVIGATION 24
FIGURE 19: CLASS DIAGRAM OF THE CONTROL INTERFACES 24
FIGURE 20: IMPLEMENTATION OF INTERFACES FOR MRP 24
FIGURE 21: NETWORK OF SERVICES OVERALL INITIAL PROTOTYPE 25

2. LIST OF TABLES

TABLE 1: VIRTUAL MACHINE CONFIGURATION 10
TABLE 2: VIRTUAL MACHINE OPERATING SYSTEM PROPERTIES 11
TABLE 3: LAYERED ARCHITECTURE FOR RESOURCE SERVICES 14
TABLE 4: ROS SERVICES FOR RESOURCE MANAGEMENT 14
TABLE 5: STATION AGENT SERVICES 15
TABLE 6: STATION CONTROLLER SERVICES 16
TABLE 7: TASKTAKEOVER SERVICE REQUEST 16
TABLE 8: TASKTAKEOVER SERVICE RESPONSE 17
TABLE 9: TASKCOMPLETED SERVICE REQUEST 18
TABLE 10: TASKCOMPLETED SERVICE RESPONSE 18
TABLE 11: RESOURCELIST SERVICE RESPONSE 18
TABLE 12: RESOURCEINFOMSG DEFINITION 18
TABLE 13: TASKLIST SERVICE REQUEST 18
TABLE 14: TASKLIST SERVICE RESPONSE 18
TABLE 15: TASKINFOMSG DEFINITION 18
TABLE 16: RECTANGLEOBJECT MESSAGE DEFINITION 20
TABLE 17: OBJECTS MESSAGE DEFINITION 21
TABLE 18: BOXSIZE MESSAGE DEFINITION 21
TABLE 19: INTERFACE DESCRIPTIONS 24

3. EXECUTIVE SUMMARY

The present report describes the developed initial prototype of the common integration communication architecture of THOMAS system, namely the Network of Services. The initial prototype provides a communication infrastructure which implements initial prototype functionality for the message exchange, including the structure of the messages to enable the communication of the production elements such as humans and mobile robot platforms (MRPs). The design of the network of services enabling the inter components communication has been documented under D5.1 "Methods for dynamic work balancing of human robot collaborative environments - Design" submitted on M12, based on the reference architecture of deliverable D1.3 "THOMAS reference architecture design" submitted on M09.

In the framework of WP5 initial prototypes of the components described in the design of D5.1 "Methods for dynamic work balancing of human robot collaborative environments - Design" were developed, providing a subset of the functionalities that will be provided by the final prototypes. In particular:

- In order to develop the initial prototype in a suitable and portable environment a virtual machine was created and configured for hosting the developed software. Inside the virtual machine a set of needed servers and other required software have been installed.
- The first version of THOMAS Service Oriented Network Adjacent Resources (SONAR) - including the implemented data model, and resource interfaces has been developed
- An initial prototype of the Resource management in combination of the THOMAS World model resource manager prototype (see D5.2) has been implemented
- The services exposed for interfacing WP2, WP3 and WP4 developments have been deployed under a first round of integration testing among the different WPs.

The interactions of WP5 with other workpackages are shown in Figure 1.

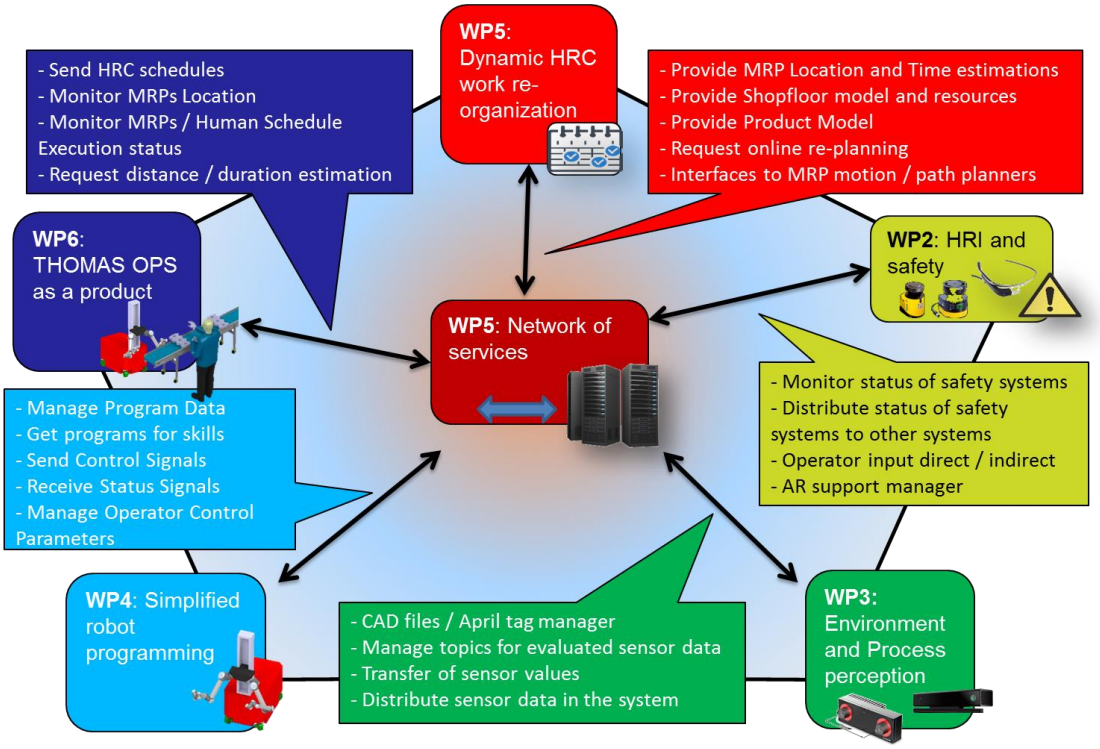

Figure 1: Overall Network of Services Concept

The developed initial prototype can demonstrate a first set of communication functionality between different THOMAS modules. The initial prototype will be further improved and will be used to integrate and test the THOMAS developments. The development in order to extend the provided functionality is continuous, since the D5.3 initial prototype will be used as a solid foundation for the development of thee final prototype to be deliverable under D5.4 "THOMAS service-based integration and communication network – Final version".

4. INTRODUCTION

This deliverable aims to describe the initial prototype of the common integration communication architecture of THOMAS system, namely the Network of Services. The design of the network of services enabling the inter components communication has been documented under D5.1 "Methods for dynamic work balancing of human robot collaborative environments - Design" submitted on M12, based on the reference architecture of deliverable D1.3 "THOMAS reference architecture design" submitted on M09.

The initial prototype of the common integration architecture is going to be further developed with the aim to offer the full functionality in D5.4 "THOMAS service-based integration and communication network – Final version". D5.3 has been developed in accordance with the D5.1 and in close cooperation with the development of other workpackages. More specifically:

- WP2 for the integration of the sensor data and human robot interaction mechanisms.
- WP3 for the integration of the sensor data for object detection and mobile resources navigation
- WP4 for the integration of the programming and program execution functionalities - Skills of robot resources which needs to be monitored and coordinated.
- WP5 for planning and online re-organization of the tasks and execution coordination and control.
- WP6 for the MRPs / Human execution monitoring.

THOMAS Service Oriented Network Adjacent Resources (SONAR) – concept implemented for deploying the designed network of services is visualized in Figure 2.

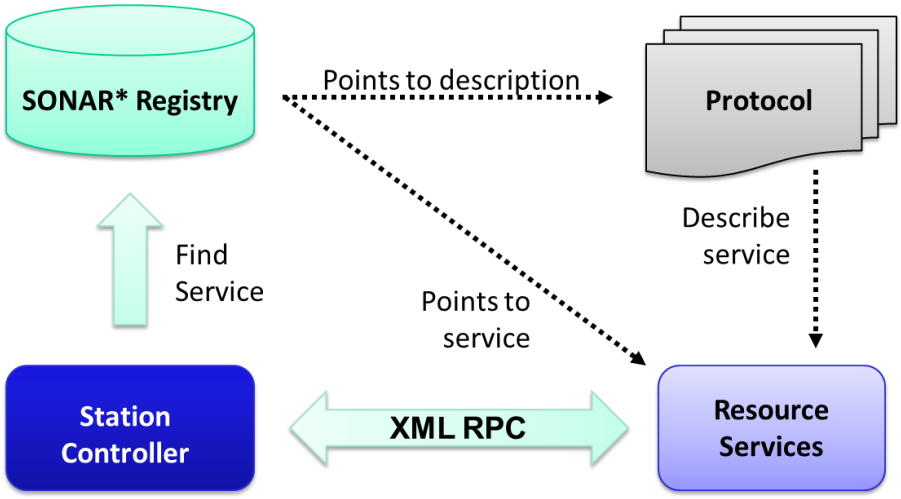

Figure 2: THOMAS SONAR Concept

The design, development, integration and testing process among the different workpackages of the THOMAS project follows an agile methodology approach. The process started with the design of the different developments then the different designs are developed in parallel within the framework of the workpackages. The developments are then integrated and tested, evaluated and then a new development cycle should start. The above process is described in Figure 3. For this reason and in order to make the integration tests possible remotely, a VPN server has been installed and configured in the virtual machine hosting the initial prototype of the THOMAS Network of Services. This agile approach allows the development in a closed loop where quality, compatibility and integration are controlled regularly, and the overall system design is constantly improved.

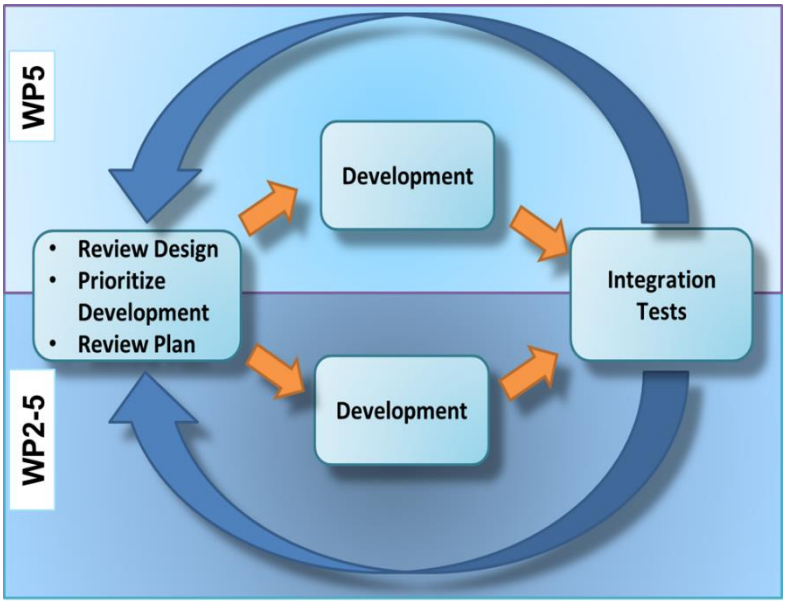

Figure 3: Agile Development Approach between THOMAS workpackage

5. KEY FUNCTIONAL CONCEPTS SPECIFICATION

5.1. SONAR – Service-Oriented Network Adjacent Resources

Service-Oriented Network Adjacent Resources (SONAR) is a system aims to support the dynamics of a Distributed Computer System, such as ROS nodes, and maintain up-to-date services currently running in the system. The system described is focused on the management of adjacent resources available on the network and their respective services. In the following sections are documented: a. the elements that consist of the data model, b. the types of nodes that fulfil certain roles of the whole system and c. the registry that centrally maintains the aforementioned services of the nodes.

5.1.1. Data Model

The data model consists of the following structures:

1. EntityDescription
2. ServiceDescription
3. BindingDescription

EntityDescription

EntityDescription structure is the center of the resource that contains the breakdown of available services, node locations and additional metadata for the resource and represents a single instance of a resource.

ServiceDescription

ServiceDescription structure represents a single service offered by an EntityDescription and includes information about how to connect to the service, its type.

BindingDescription

BindingDescription structure represents the technical specification required for the successful connection to a resource-service. For future purposes, it is allowed that multiple BindingDescription may be defined by a ServiceDescription so as to support future protocols.

5.1.2. Nodes

Nodes are considered all applications which either manage, provide or consume services into the system. The most important node in the SONAR architecture is the Central Registry which maintains the list of available services for clients to discover. Overall, the identified types of nodes in SONAR are given below:

1. **Central Registry**: a node that holds the available services available at any given instance.
2. **Service Providers** are considered the nodes that offer one or more services to the system
3. **Service Consumers** are considered the nodes that use one or more services to fulfil their purpose

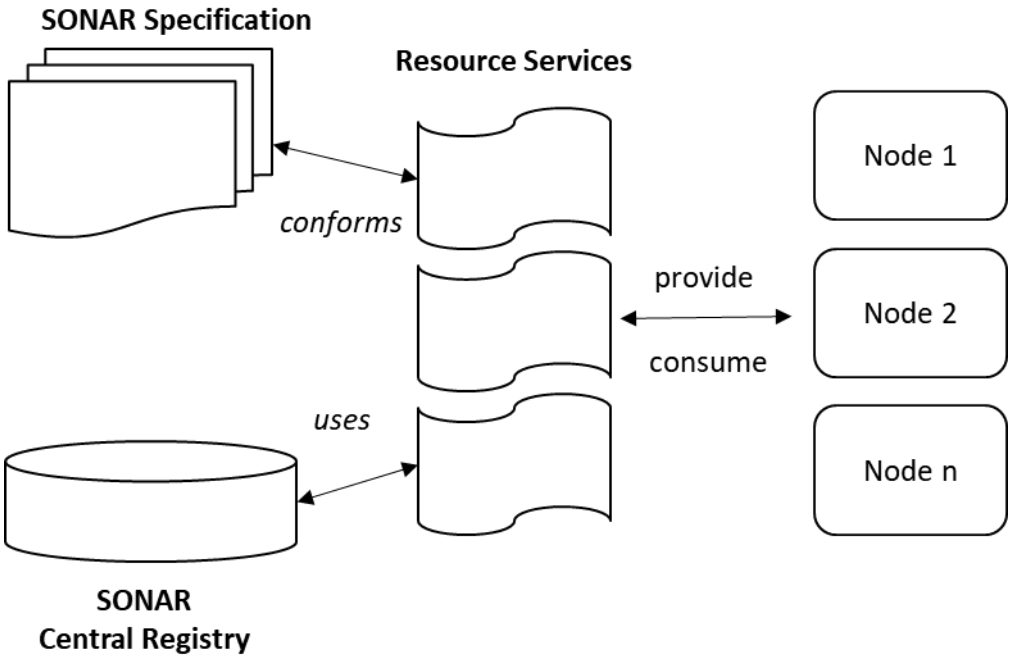

Figure 4: Illustration of the SONAR System. Nodes and Registry

Service Providers

Nodes that act as service providers utilize ROS communication protocol for data-transferring of requests and responses. For future-proof tolerance, the kinds of services provided are left to the discretion of each application.

Service Consumers

Nodes that act as Service Consumers will interact with the Central Registry to query the availability of the services required and, subsequently, they will interact directly with the specific Service Provider of that service. The Service Consumer is required to follow the service specification of any particular service is needed so as to be able to interface correctly.

5.1.3. Registries

The Centralized Registry holds the responsibility to maintain the list of available services at the runtime of the system and support clients for service registration and discovery.

The clients are classified into two (2) categories.

1. Service Provider: a client that offers one or more services to the system
2. Service Consumer: a client that uses one or more services to perform its purpose.

6. SONAR ARCHITECTURE

The current version of the software prototype is currently deployed in a virtual execution environment which is described in detail below. The virtual execution environment is a virtual machine which is currently hosted in an Oracle Virtual Box system. The virtual machine can be exported in the VHD (Virtual Hard Disk) format, which is commonly used format and accessible by many virtualization software applications. The software chosen in order to create the components of the virtual machine can be considered as the state of the art. The aforementioned section provides specific information about the current software used, including the versions, however the virtual machine could be possibly updated with a newer version of a software or additional software could be installed as needed.

6.1. Prototype Environment

6.1.1. Virtual Machine Specifications

The main specifications of the virtual machine are provided in Table 1. These specifications can be adjusted if needed by an appropriate tool. More details are shown in the settings provided in Figure 5.

Table 1: Virtual Machine Configuration

Component	Minimum Requirements
CPU	2 Processors
RAM	4932 MB
HDD	50 GB

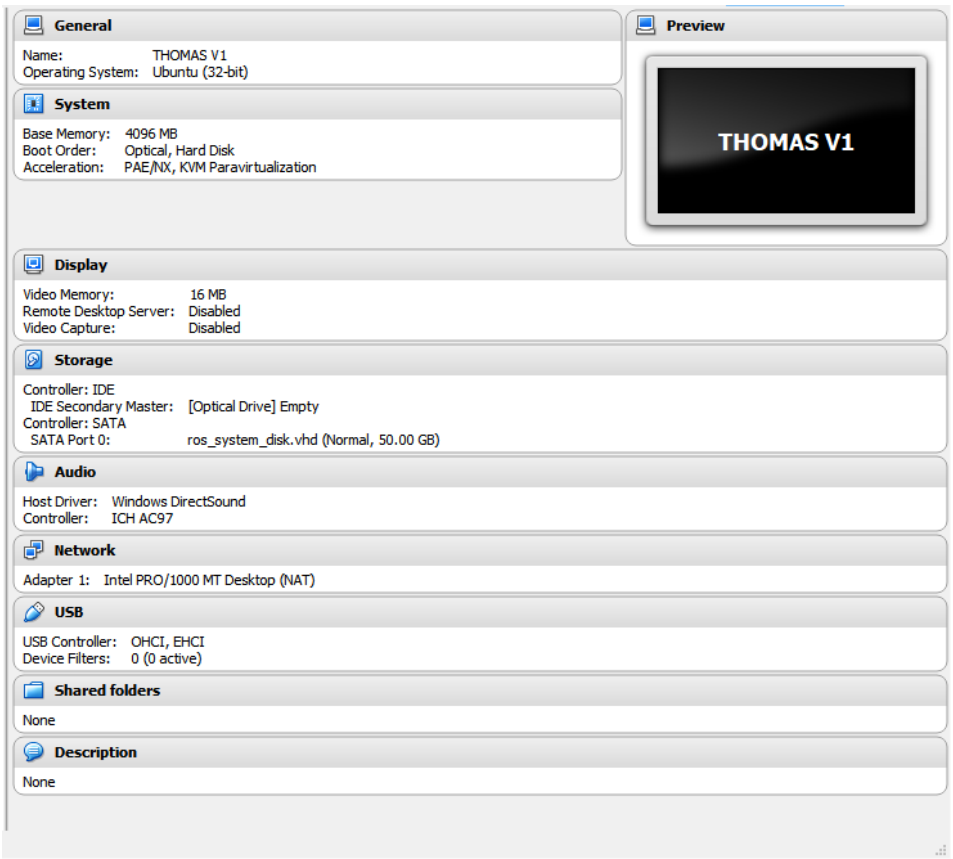

Figure 5: Virtual Machine Detailed Settings

6.1.1.1. Operating System

The choice of operating system has been made to ensure long term support, high performance and stability. Ubuntu 14.04 LTS is an open source long-term support operation system release. It has continuous hardware support improvements as well as guaranteed security and support updates until April 2019 [1]. The used Linux distribution and kernel release information are provided in Table 2.

Table 2: Virtual Machine Operating System Properties

Property	Value
Operating System Type	GNU/Linux
Linux Distribution	Ubuntu 14.04.01 LTS
Kernel Release	3.13.0-32-generic x86_64

The configuration of the virtual machine has started from an empty, blank virtual machine disk where the aforementioned Ubuntu Linux 14.04 LTS operating system has been installed and configured. Several configurations and installations have taken place such as:

- Users and rights configurations.
- Network configuration.
- Installation of additional software.
- Performance optimisation.

A summary of additional packages that have been installed and configured in order to enhance the performance of the virtual machine as well as provide additional functionality, which is useful for the execution of the system, is provided in the following sections.

6.1.1.1.1. Preload - adaptive read-ahead daemon

The virtual machine will be mainly used to host the servers' software that is described in detail. Therefore, the system shall benefit from an application that speeds up the execution of programs that run often. For this reason, the preload package has been installed in the virtual machine. The preload package offers an adaptive read-ahead daemon which monitors applications that users run, and by analysing this data, predicts what applications users might run, and fetches those binaries and their dependencies into memory for faster start-up times. Since the run applications will consistently be the aforementioned server software, the read-ahead daemon will consistently speed up the execution of the servers' software. Of course, the improvement of the performance due to the loading of the dependencies in the memory has a certain limit. However, the installation of the preload package is expected to improve the system performance.

The SONAR modules are integrated into the Virtual Machine (VM) Personal Computer (PC) environment to enable provisioning of multiple SONAR Registries. Additional nodes could be installed in the same VM, however, this is optional as the nature of the distributed architecture enabled the usage of SONAR registry from remote locations as long as they are located in the same subnet.

The minimum hardware operating requirements for a node operating the SONAR registry are given in the table below:

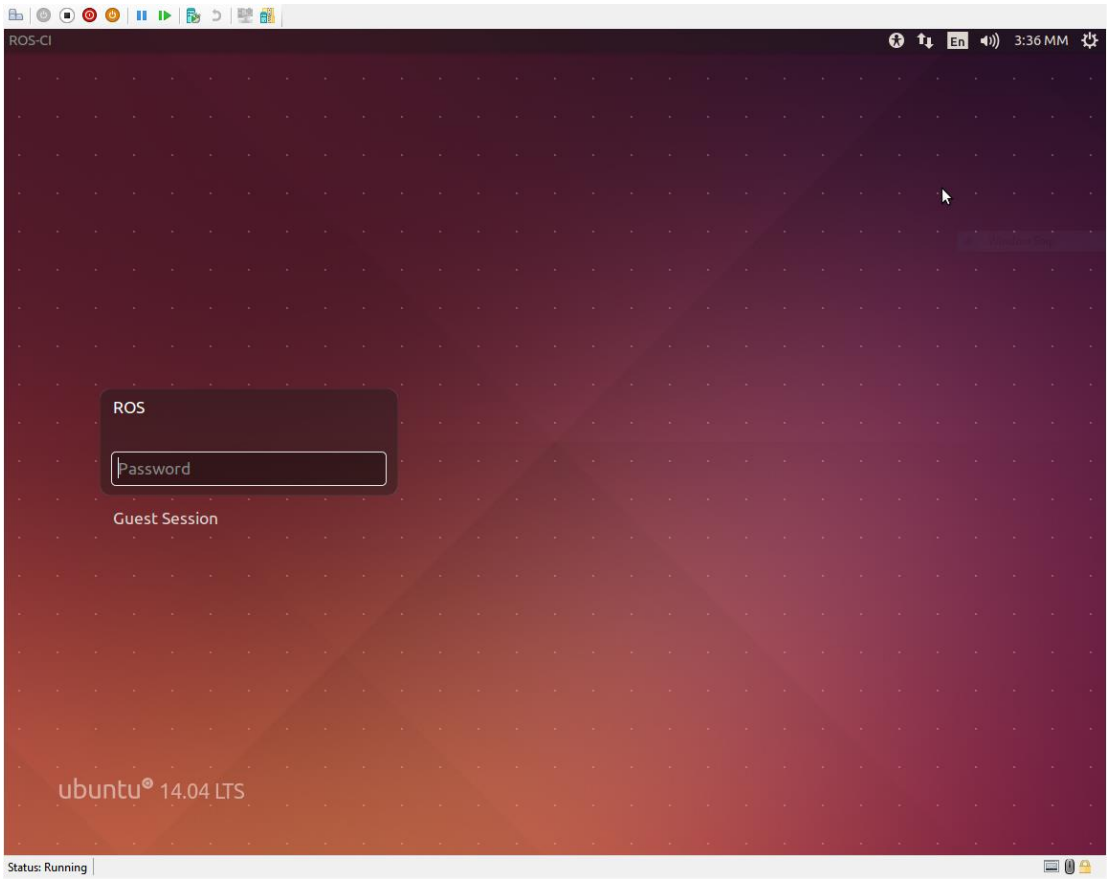

Figure 6: Virtual Machine Running for Desktop

6.2. Resources Management

This section documents the initial version of the Data Model of the THOMAS World Model that acts as the central registry for Resources among others. The current version of the Central Registry repository is a MySQL Community [2] edition which holds the information presented in D5.1 "Methods for dynamic work balancing of human robot collaborative environments – Design ", section 3.5.6 "THOMAS World Model". Moreover, for each of the corresponding model, the equivalent software layers were defined as messages and services so as the independent ROS nodes could access the data information.

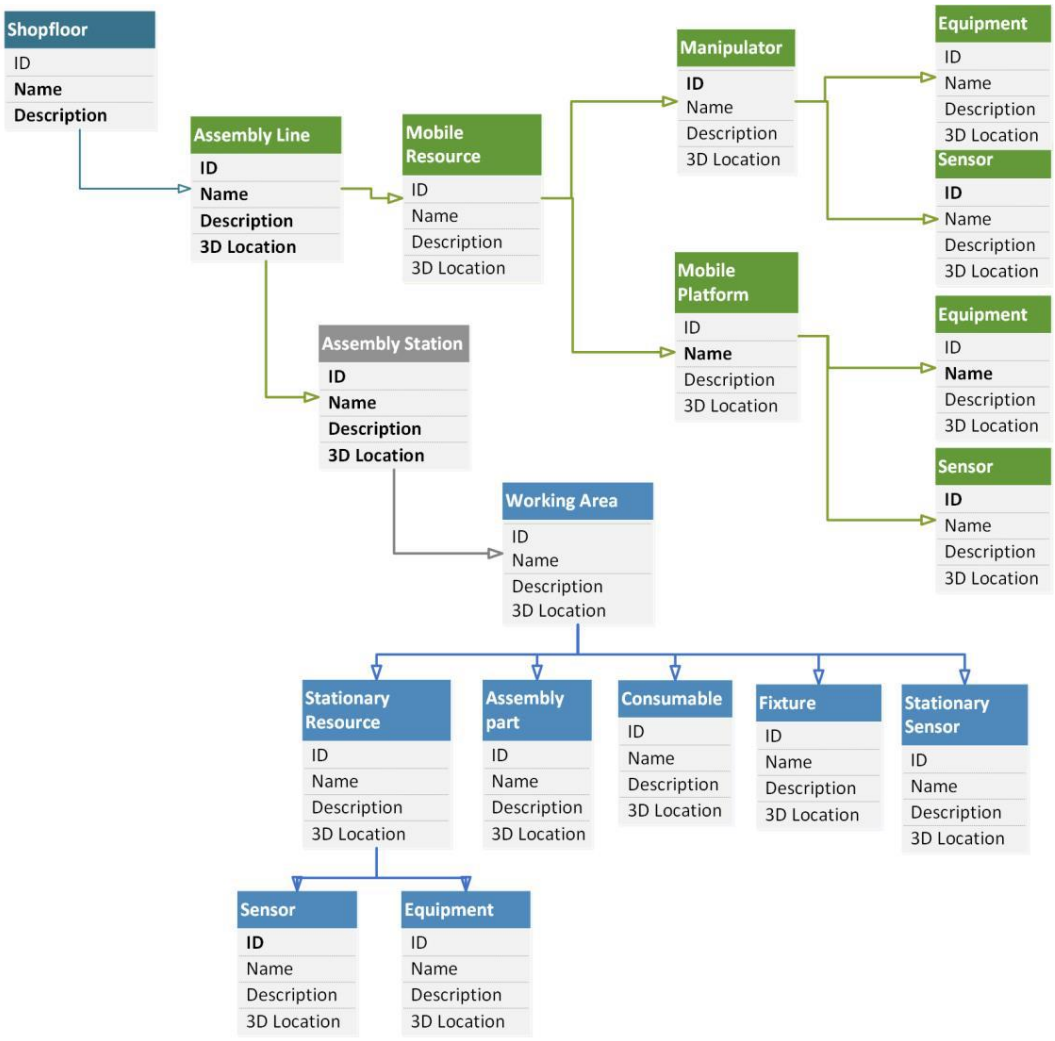

Figure 7: THOMAS World Model [ref D5.1]

In the following diagram, the layered architecture is presented for the Resource Management within the World Model as an example for the rest of the repository.

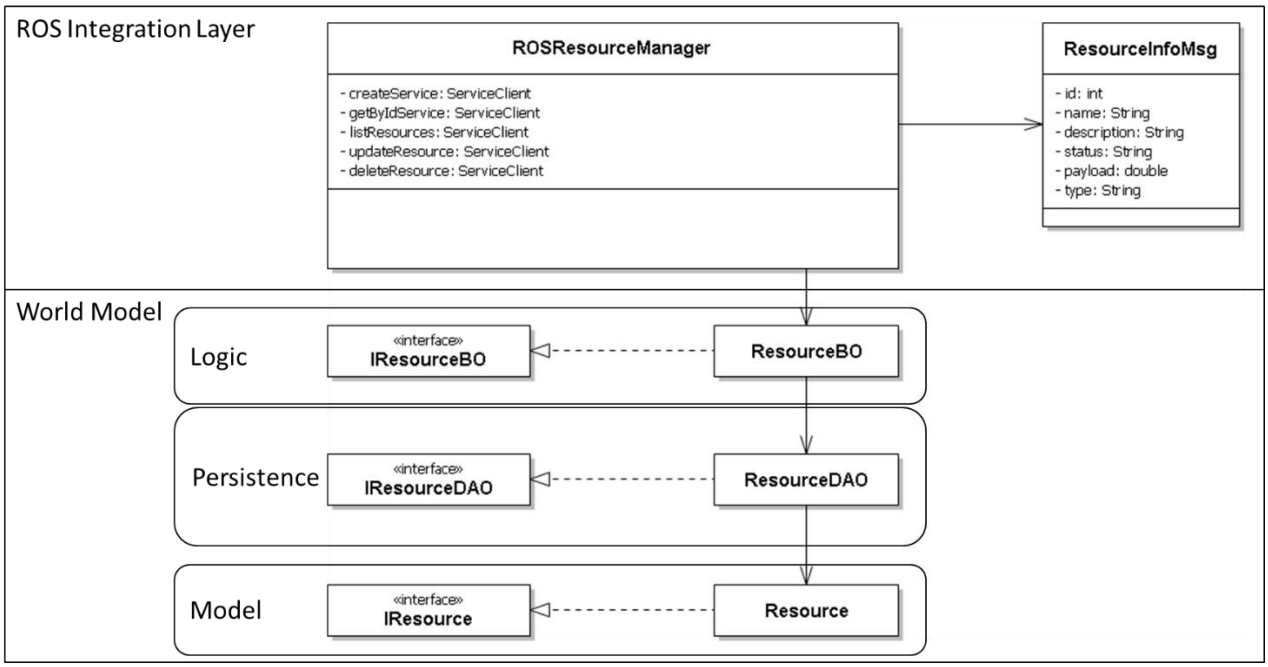

Figure 8: Layered Architecture for Management of Resources

In particular, the objects designed in this layered architecture have the following responsibilities:

Table 3: Layered Architecture for Resource Services

Layer	Description
IResource	The model interface for a single Resource instance.
IResourceDAO	The interface of Data Access Object (DAO) that manipulate resources in the Central Registry.
IResourceBO	The interface of Business Object (BO) that maintains the implementation for Resources Management
Resource	Implementaion class of the IResource interface for World Model
ResourceDAO	The implementation class for the IResourceDAO for storing resources in the MySQL Community Edition
ResourceBO	The implementation class of the IResourceBO for the management of resources
ROSResourceManager	The ROS Wrapper that exposes the functionality of the ResourceBO implementation as ROS services.
ResourceInfoMsg	The ROS Message for Resources used by the ROSResourceManager.

For the ROS integration layer, the following ROS Services were defined:

Table 4: ROS Services for Resource Management

Service	Description
ResourceCreate	Creates a new resource in the Central Registry.
ResourceGetById	Gets the information for a resource of the specified ID.

ResourceGetAll	Gets all the available resources in the Central Registry.
ResourceUpdate	Updates the resource information for an existing Resource.
ResourceDelete	Deletes an existing resource from the Central Registry.

6.3. Station Agent and Station Controller

A station agent is developed to accommodate the behavioral logic of the Station Controller for a single MRP. Such agents are able to run locally on each MRP, load instructions to them by the Execution Coordinator, and execute them by calling the respective service for each kind of action. Additionally, agents for Humans are also required so as to interact with the Station Controller, the MRPs or the MPPs.

In the following table, are shown the services created for a single agent:

Table 5: Station Agent Services

Service	Description
/{AGENT_NAME}/load_task	Service for loading programs to the station agent
/{AGENT_NAME}/start	Topic for starting execution of the agent's resource
/{AGENT_NAME}/stop	Topic for stopping execution of the agent's resource
/{AGENT_NAME}/agent_status	Topic for sending status messages of the agent
/{AGENT_NAME}/task/status	Topic for sending status messages about the task execution
/{AGENT_NAME}/gesture/detected	Topic for receiving detected gestured to be processed.

For instance, in the following picture illustrated 2 MRPs within the GAZEBO Simulation Environment under the current implementation of THOMAS World Model (see D5.2 "Dynamic work re-organization – Initial prototype" for more details on this prototype). Both MRPs are represented by a station_agent with a distinct name.

- The first agent is named /mrp_1 to manipulate the resource at the centre
- The second agent is named /mrp_2 to control the resource at the left

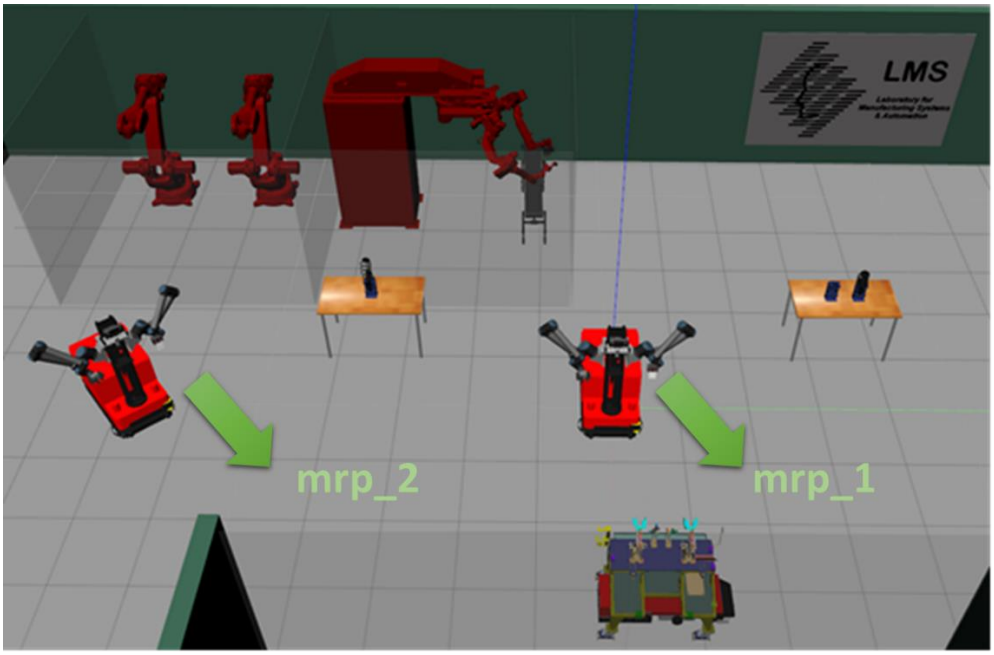

Figure 9: THOMAS world model visualization – MRP 1 and MRP2

Similarly, a station agent is required for Human resource respectively (ie. /worker_1) so as for a human to interact with the Station Controller as a whole.

Table 6: Station Controller Services

Service	Description
/station/resource_list	Service for acquiring the loaded resources into the station
/station/task_list	Service for acquiring the loaded tasks into the station
/station/TaskTakeOver	Service for interaction with Human Interfaces.
/station/resource/down	Service to notify station controller that resource is unavailable
/station/resource/up	Service to notify station controller that resource is now available.
/station/task/status	Service for notifying changes on tasks for all agents
/station/task/completed	Service for notifying controller that a task is completed (mostly used by Human Interfaces)

Task Take Over

The Task Take Over service in Station Controller is implemented to support the Human Robot interaction. In particular, the human is able to interact with the robot by taking the task off the robot in order for human to execute it whereas the robot may continue execution, if possible, to the next task.

Resource Down / Up

The resource down service in Station Controller is implemented so that an external ROS node involved in monitoring of the MRP to be able to notify the controller that it is not available anymore. Additionally, the same service may be used by the maintenance department of the factory. In such cases that any resource is down, the Station Controller will check for assigned work and re-assign the work to another resource.

Similarly, the Resource Up service is implemented so that external ROS nodes or humans indicate to the Station Controller that a resource is back available to the system. In this case, the controller will perform work balancing so as to distribute the work among all available resources at the time.

6.4. Services for WP2 Integration – Human Robot Interaction mechanisms

6.4.1. Task Take Over Services

For Human Robot Interaction, a Smartwatch application were developed, and the respective ROS services were defined and implemented to enable interaction with the agent of Station Controller. In particular, the Human interacts with the Robot using the aforementioned **Task Take Over** functionality implemented in the station agent that controls the MRP. Specifically, the **TaskTakeOver** service is defined as follows:

Table 7: TaskTakeOver Service Request

Attribute	Type	Description
taskId	String	The id of the task involved.

fromResourceId	String	The id of the resource to take the task from.
toResourceId	String	The id of the human to assign the task to.

Table 8: TaskTakeOver Service Response

Attribute	Type	Description
result	Bool	True, when the task was taken from the resource successfully, otherwise, false.

The Smartwatch utilizes few more services to bring the full potential of the functionality for the Human. In particular, it utilizes the services:

- **/station/resource_list**: to download the available resources at the current time instant and select the appropriate resource among them.
  The same service is used so the user can login by selecting the human resource she corresponds.
- **/station/task_list**: to download the assigned tasks for a particular resource
- **/station/TaskTakeOver**: to indicate that the task in interest should be re-allocated to the human
- **/station/task/status**: Received notification to start the task by the Station Controller
- **/station/task/completed**: Human notifies the Station Controller that the task is completed

Similarly, the execution sequence in typical conditions for the **TaskTakeOver** functionality is depicted in the following UML sequence diagram:

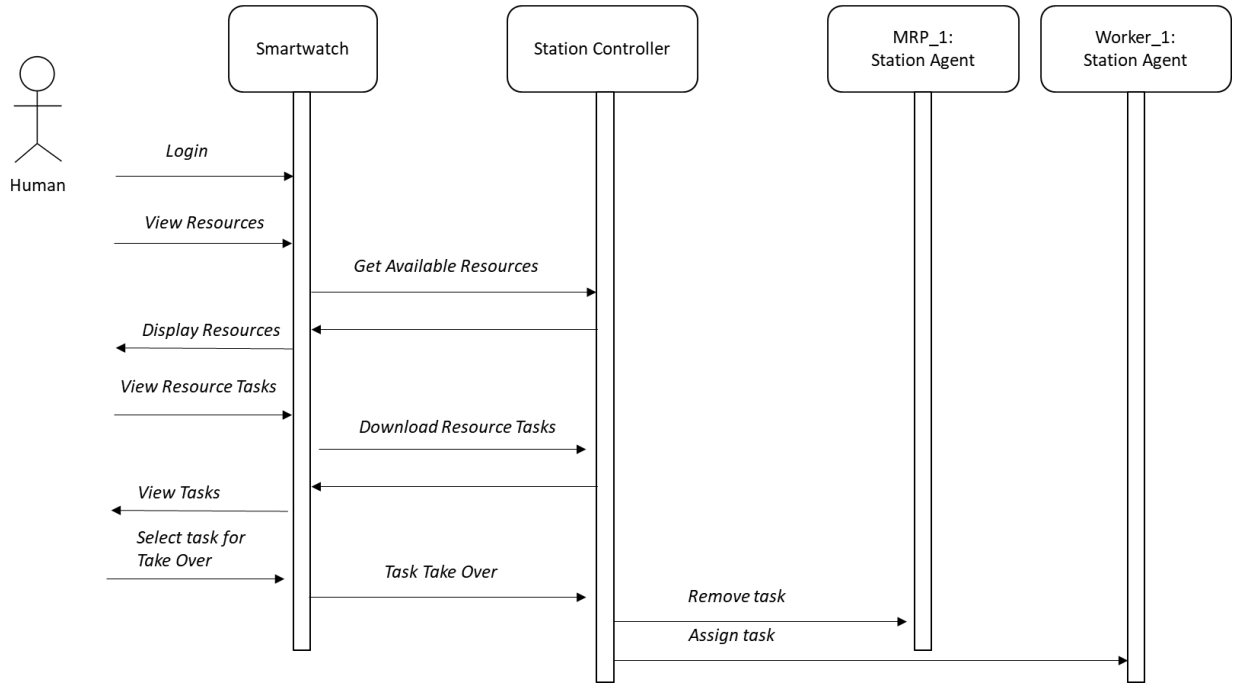

Figure 10: UML Sequence Diagram of the Task Take Over feature

The definitions of the remaining services are documented below:

Table 9: TaskCompleted Service Request

Attribute	Type	Description
taskId	String	The id of the task that is completed
resourceId	String	The id of the resource that completed the task
completed	String	True, if the task is completed successfully, otherwise, false.

Table 10: TaskCompleted Service Response

Attribute	Type	Description
result	Bool	True, when the request is being processed successfully

Table 11: ResourceList Service Response

Attribute	Type	Description
resources	ResourceInfoMsg[]	An array of Resource Info Messages (1 per resource)

Table 12: ResourceInfoMsg Definition

Attribute	Type	Description
resourceId	String	The id of the resource
displayName	String	The name to display for the resource
human	Bool	True, if the resource is human, otherwise, false.
robot	Bool	True, if the resource is robot, otherwise, false.

Table 13: TaskList Service Request

Attribute	Type	Description
resourceId	String	(Optional) The resource id to get the tasks for. If omitted, all loaded tasks are returned

Table 14: TaskList Service Response

Attribute	Type	Description
tasks	TaskInfoMsg[]	An array of TaskInfoMsg (1 per task)

Table 15: TaskInfoMsg Definition

Attribute	Type	Description
taskId	String	The id of the task
displayName	String	The name to display for the task
assignedResourceId	String	The id of the assigned resource

6.4.2. Human Gesture/Posture Recognition Service

Additionally, ROS services implemented to aid the manipulation of robotic arms by human gestures. Station agents are able to receive such gesture by respective WP2 module of Human Gesture/Posture Recognition using the sensor (Kinect v2) locating on the MRP's torso and translate it to commands for motion control. Subsequently, the motion command is delegated for control by the Skill Execution Engine from WP4.

The definition of the gesture topic is defined as textual commands as follows:

- NONE: No human detected in front of the MRP's sensor
- IDLE: Human is detected in front of the sensor but no gesture issued yet.
- MOVE_LEFT: A command issued to move left
- MOVE_RIGHT: A command issued to move right
- MOVE_UP: A command issued to move up
- MOVE_DOWN: A command issued to move down
- MOVE_BACK: A command issued to move back
- MOVE_FORWARD: A command issued to move forward

Additionally, the implementation of the Gesture Detector is enhanced to include the selection of robotic arm that the user wishes to perform the aforementioned commands. In particular, the arm selection command must be executed prior performing any other gestures for reference. Therefore, if the End-User requires to select the LEFT_ARM of the MRP, then he/she raises her left arm and respectively for the right arm. Any subsequent commands would be executed by the selected robotic arm. Details on the integration of the Human Gesture/Posture Recognition module are documented in D2.3 submitted on M18 of the project. Figure 11 visualizes the prototype implementation of the human gesture recognition handler integration as a service inside the 3D simulation set up based on the PSA pilot case. The digital MRP receives the input captured by the virtual Kinect based on MRP's torso. This Kinect v2 has been mapped to a physical Kinect v2 that records the 2D real view presented in the figure (see D5.2 for details on the managing of this sensor). Using the exposed depth data, the gesture handler translates human gesture into robot commands and through the station controller these commands are dispatched to the virtual MRP.

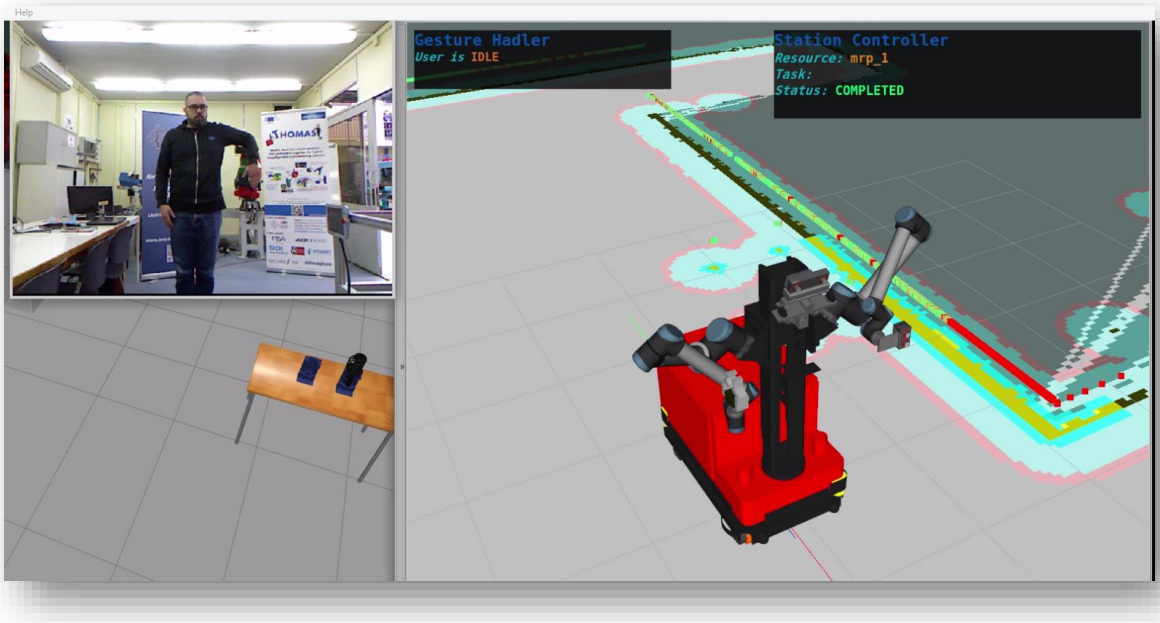

Figure 11: Human Gesture/Posture Recognition Service

6.4.3. Object pre-classification based on 2D data – 2D Human Detection Service

A ROS Wrapper is implemented for the 2D human detection module that acquires its information from a SICK MicroScan3 sensor. The 2D human detection module ROS service utilizes the SICK ECU in order to acquire raw data from the sensor as detailed under THOMAS World model initial prototype description on D5.2 submitted on M18 of the project. Then, it applies the algorithm to convert raw data to PointCloud and successively applies the object detection algorithm so as to cluster several Points to the identified Objects.

Moreover, the 2D human detection service provides backward-compatibility with applications require a typical PointCloud, without the additional Object Detection Messages, because it has been designed and implemented as a separate message. In the following diagram is depicted the workflow between the components (from left to right). On the left, the SICK MicroScan3 sensor is located on the MRPs were the driver is connected and feeds the Object Detection with data. Finally, the outcome message is published in addition to the standard ROS PointCloud message.

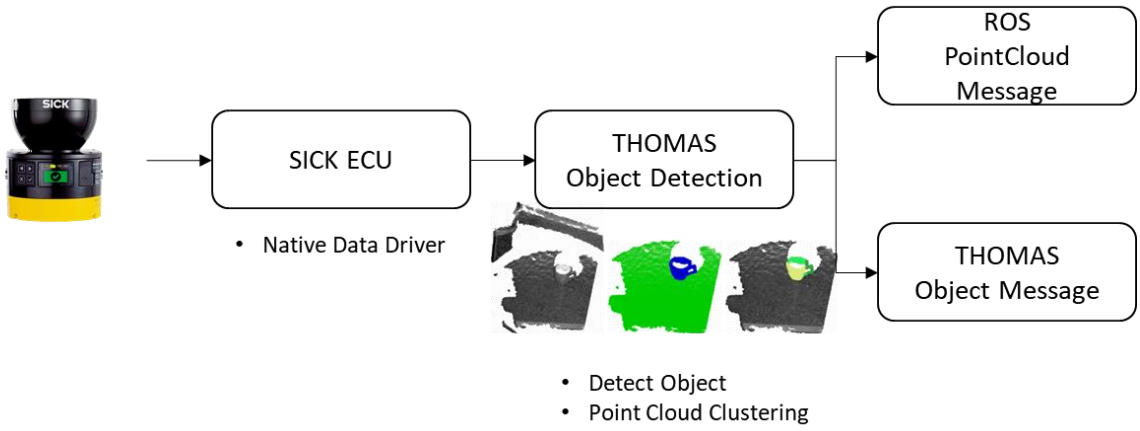

Figure 12: Object Detection Workflow Diagram

Hereafter are the message definitions implemented for the Object Detection Service.

Table 16: RectangleObject Message Definition

Attribute	Type	Description
deviceID	uint8	ID of the publishing service
objectID	uint16	Unique number for each object from the ECU
age	uint32	Number of scans this object has been tracked
hiddenAge	uint16	Number of scans an object has only been predicted without measurement update.
center	geometry_msgs/Point	Center point of the tracked object [in meters]
size	sick_ecu_msgs/BoxSize	The size of the object box [in meters]
velPose	geometry_msgs/Pose2D	Position and Orientation of the object
velMagnitude	float32	Magnitude of velocity [m/s]

Table 17: Objects Message Definition

Attribute	Type	Description
header	Header	ROS Timestamp Information
rectangles	RectangleObject[]	Array of detected objects

Table 18: BoxSize Message Definition

Attribute	Type	Description
x	float32	The X coordinate of the box
y	float32	The Y coordinate of the box

6.5. Services for WP3 Integration – Perception for Process Context Awareness

Services implemented for the perception module for the integration of rc_visard sensors. The perception module implemented is able to detect the part based on the system's training through the CAD model. Subsequently the module is able to update the THOMAS World Model for the location and orientation of the part.

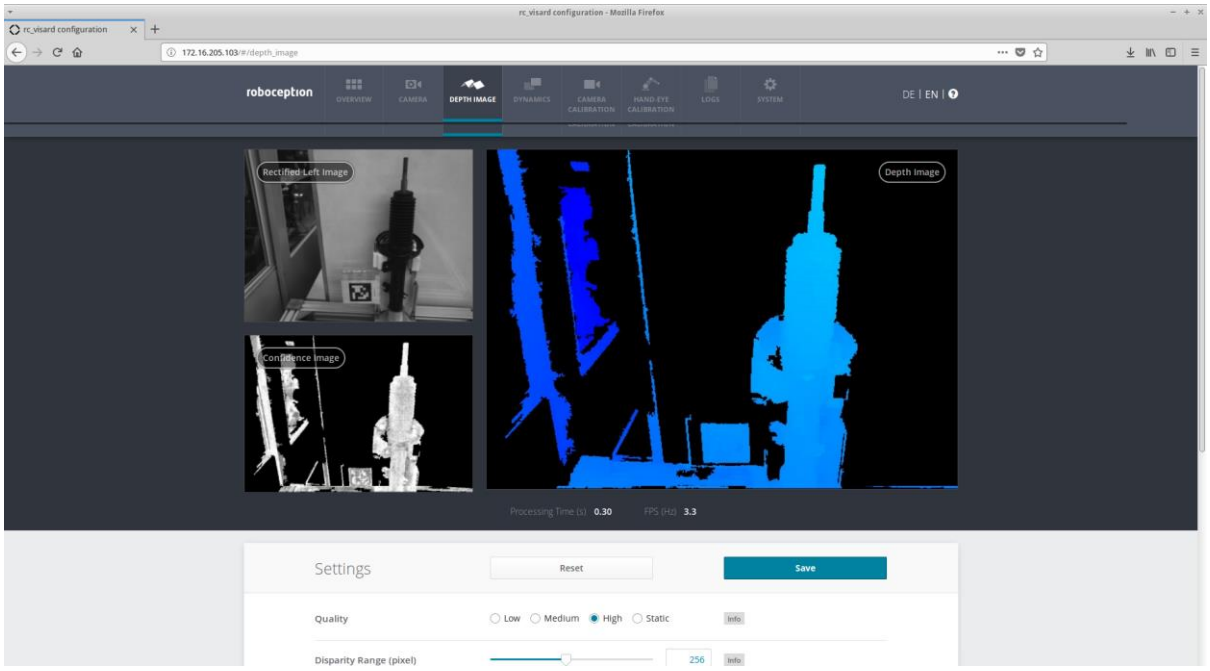

Figure 13: Screenshot from the ROBOCEPTION GUI

During the online execution phase, the process perception modules are subscribing to the respective stereo camera topics exposed by the THOMA World Model. Figure 14 presents the sequence diagram and information exchange among the different modules during the execution of a Pick_Up task, slightly updated with respect to the design presented in Deliverable D5.1, submitted on M12. The process perception modules are integrated in the system skill engine (see D3.3 for the communication interface) while the skill engine is triggered by the Execution Coordinator (see next section for the implemented communication interface) for the execution of the required tasks.

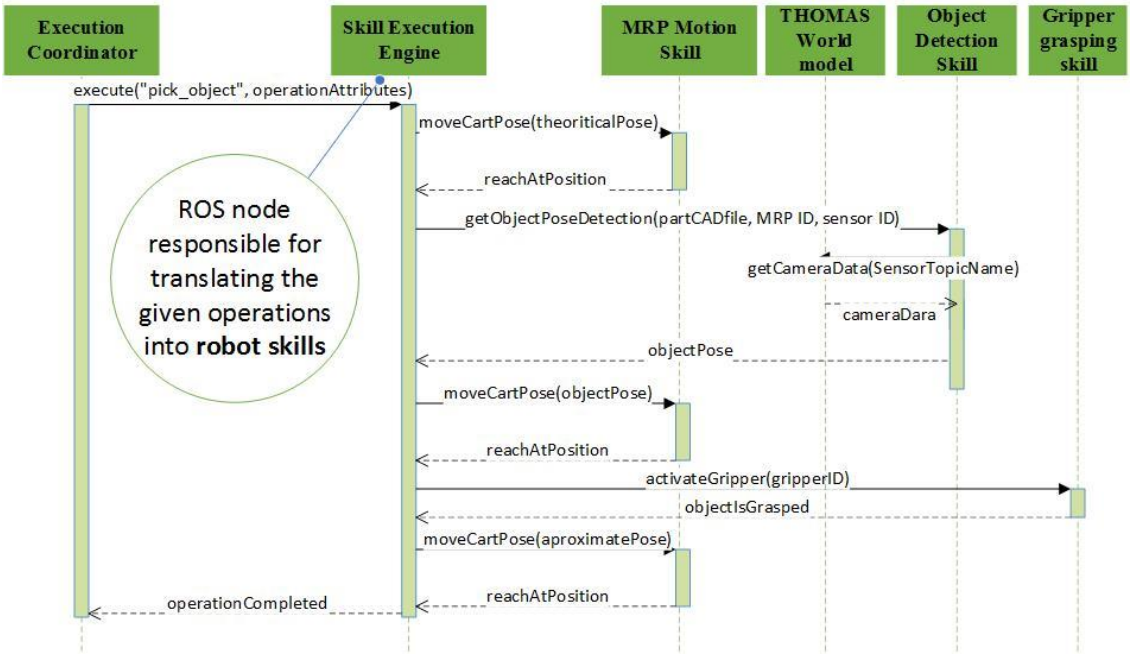

Figure 14: Pick_Up Task execution sequence diagram

When the Pick_Up operation is assigned to an MRP, the Execution coordinator calls the Skill Execution Engine component by sending a request to his action server. This action server is implemented with the concept of ROS-actions and is responsible for translating the assigned to MRP operation to robot skills. The requested goal is consisted of operationType which the unique description of the operation is and the operationAttributes which is a custom message containing the suitable information describing each operation. Figure 15 visualizes the initial implementation of the Object detection module as a service as deployed in the physical set up at LMS Machine Shop. A detailed description of the set up and the standalone application is provided in D3.3 document as well as the attached video demonstrator submitted on M18 of the project.

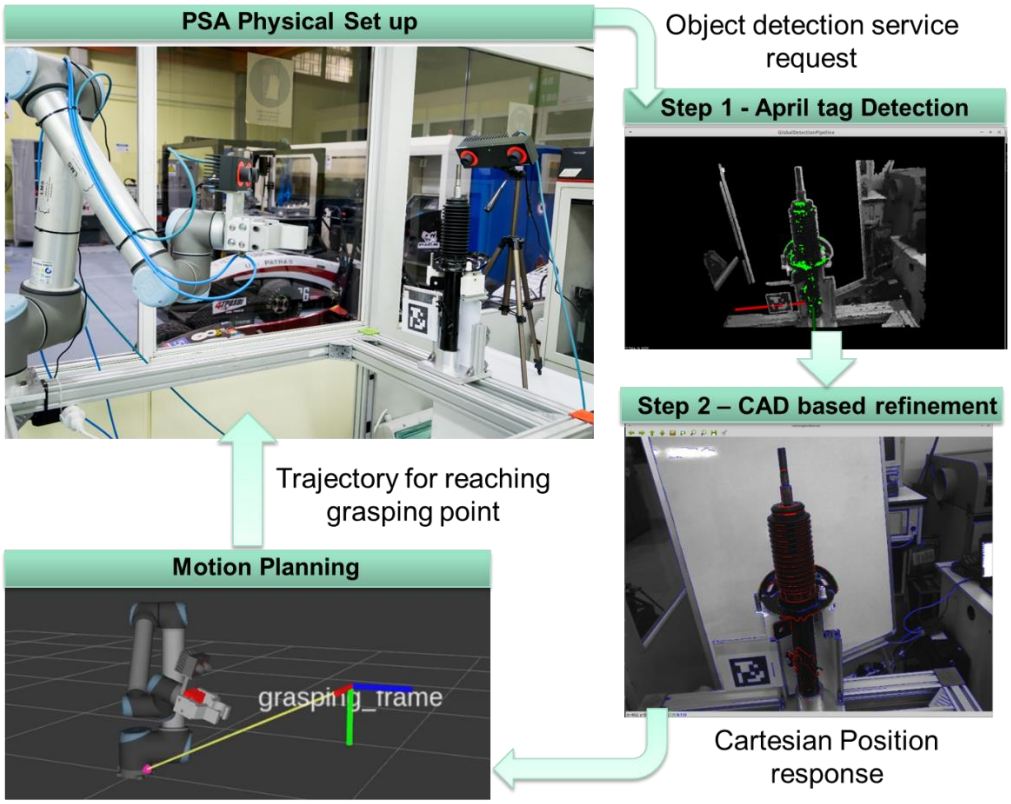

Figure 15: Damper part detection and grasping

6.6. Services for WP4 Integration – Skill Engine

For the integration of the Simplified Robot Programming and Skills application, a dedicated interface was implemented. The interface bridges the communication of WP5 Station Controller and the WP4 Skill Engine. In particular, the interface is being implemented based on ROS ActionLib [3] that enables both parties to be in total sync while a task or skill is executed. Moreover, an Action Server is implemented in the Skill Engine and an Action Client is implemented on the Station Controller.

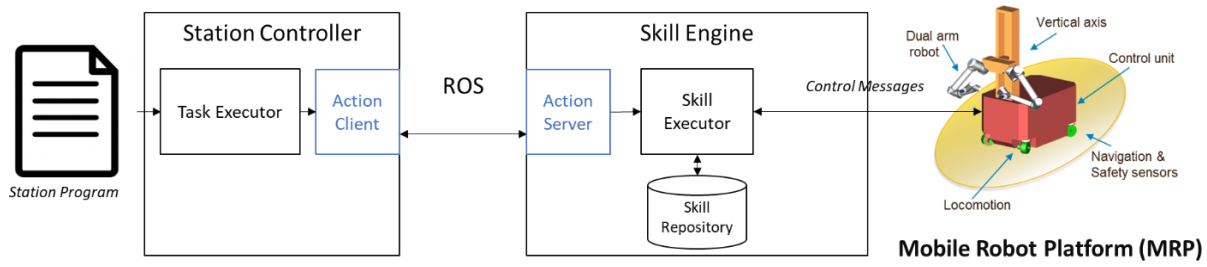

Figure 16: Communication Interface of Station Controller/Skill Engine Integration

Firstly, the station program must be loaded into the station controller so as to distribute the tasks among the available resources and their respective controllers. Once the End-User indicates that the production may start, the tasks begin execution by the respective *TaskExecutor*. For each of the task that needs to be executed by the Skill Engine, the *TaskExecutor* delegates them through the communication interface and tracks its status either for successful completion or if an error occurred.

A station program is defined by the series of tasks that will be executed by each resource's TaskExecutor where resource assignments are already calculated by the Work Re-organization module and its scheduling algorithm detailed in D5.2.

```
<?xml version="1.0" encoding="UTF-8" ?>
<program>
    <action name="psa_damper" description="[THOMAS] PSA Demo program for the assembly of damper" type="TASK" assigned="mrp_1">
        <actions>
            <action name="home" description="Go to HOME position" type="TASK">
                <actions>
                <action name="fold_arm" description="Move Arm near base" type="RIGHT_ARM_MOVE_JOINTS">
                    <inputs>
                        <input name="joints" value="1.57079632679 -1.57079632679 -2.74889357189 -1.9613 -1.8571 -0.75992" />
                    </inputs>
                    <outputs />
                </action>
                    <action name="home_nav" description="Go to HOME position" type="NAVIGATE">
                        <inputs>
                            <input name="target_frame" value="map" />
                            <input name="pose" value="0 0 0 0 0 0" />
                        </inputs>
                        <outputs />
                    </action>
                </actions>
        </action>
        <action name="pick_pre_assembly" description="Pick pre-assembly from table" type="TASK">
            <actions>
                <action name="pick_pre_nav" description="Go to pre-assembly table" type="NAVIGATE">
                    <inputs>
                        <input name="target_frame" value="pre_frame_tool" />
                        <input name="pose" value="0.4 1.5 -1.7 0.0 0.0 -1.5707" />
                    </inputs>
                    <outputs />
                </action>
                <action name="pick_pre_arm" description="Move Arm to grasp pre-assembly Waypoint 1" type="RIGHT_ARM_MOVE_JOINTS">
                    <inputs>
                        <input name="joints" value="1.11017 -2.62344 -1.89463 -1.66288 -1.94899 -0.75992" />
                    </inputs>
                    <outputs />
                </action>
                <action name="pick_pre_arm_2" description="Move Arm to grasp pre-assembly Waypoint 2" type="RIGHT_ARM_MOVE_LINEAR">
                    <inputs>
                        <input name="target_frame" value="pre_frame_tool" />
                        <input name="pose" value="0.0 0.0 -0.570 1.561 0.017 -0.043" />
```

Figure 17: XML Station Program from PSA Case of Station Controller

```xml
<?xml version='1.0' encoding='utf-8'?>
<process version="1.0">
  <action name="navigate">
    <parameters>
      <param name="goal">
        <value type="data">[0.78, -0.95, 0.0, 0.0, 0.0, 3.14]</value>
      </param>
    </parameters>
    <result/>
  </action>
</process>
```

Figure 18: XML Skill Definition of Skill Engine for Navigation

6.7. Programmatic Interfaces

To support the requirements within the Station Controller, programmatic interfaces were implemented for all the components to interact with. Figure 19 illustrates the inheritance of the interfaces implemented. In particular Table 19 provides descriptions for the interfaces.

Table 19: Interface Descriptions

Class	Description
IController	The base interface of all controllers.
IResourceController	The interface that must be implemented for all resources that require control during execution in the Station Controller
IGripperController	The interface to be implemented for grippers
IMobileController	The interface to be implemented for mobile resources
IGestureController	The interface to be implemented for translating gestures to control commands

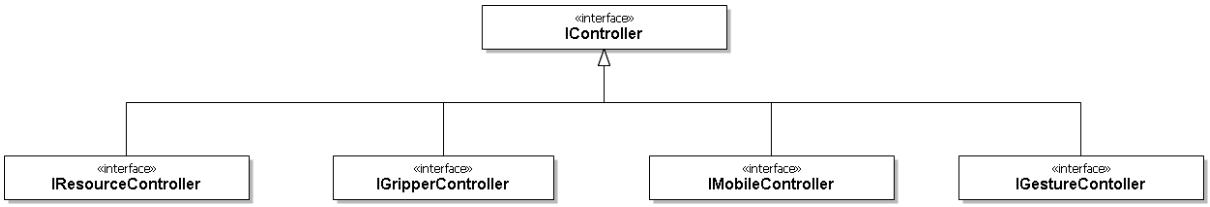

Figure 19: Class Diagram of the Control interfaces

For instance, we implemented the MRPController as a *IResourceController* so as to control the dual arm or each arm individually and simultaneously as a *IMobileController* to be able to move it within the Shopfloor.

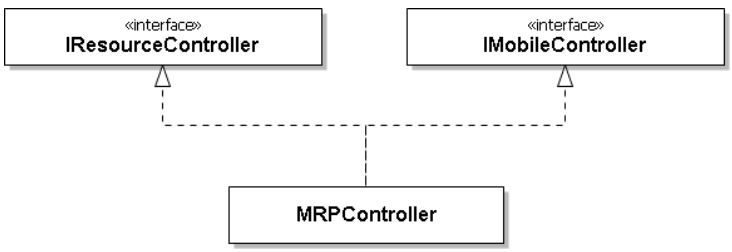

Figure 20: Implementation of interfaces for MRP

6.8. Network of services prototype implementation

Following the individual developments described in the previous section, Figure 21 represents the overall initial prototype developed for the network of services. In particular, the implemented scenario as the damper compression and assembly of the automotive pilot case of the project. The digital environment has been set up based on the LMS Machine Shop layout given that there will be deployed the first version of the automotive pilot case. A detailed description of the THOMAS world model set up is provided on D5.2 submitted on M18 of the project. Under this scenario the execution main tasks involved where pick and place tasks requiring the integration of the navigation, arm motion and gripper control services implementation. The navigation and arm motion services use the data exposed through THOMAS world model for ensuring collision free paths and trajectories. In addition, the human gesture recognition module has been integrated as a service allowing the physical human to interrupt the execution and directly instruct the robot. The object detection services in the current implementation has been tested in the physical set up but not in the simulation environment. Thus, the next step would be to integrate in this model the process perception modules as well.

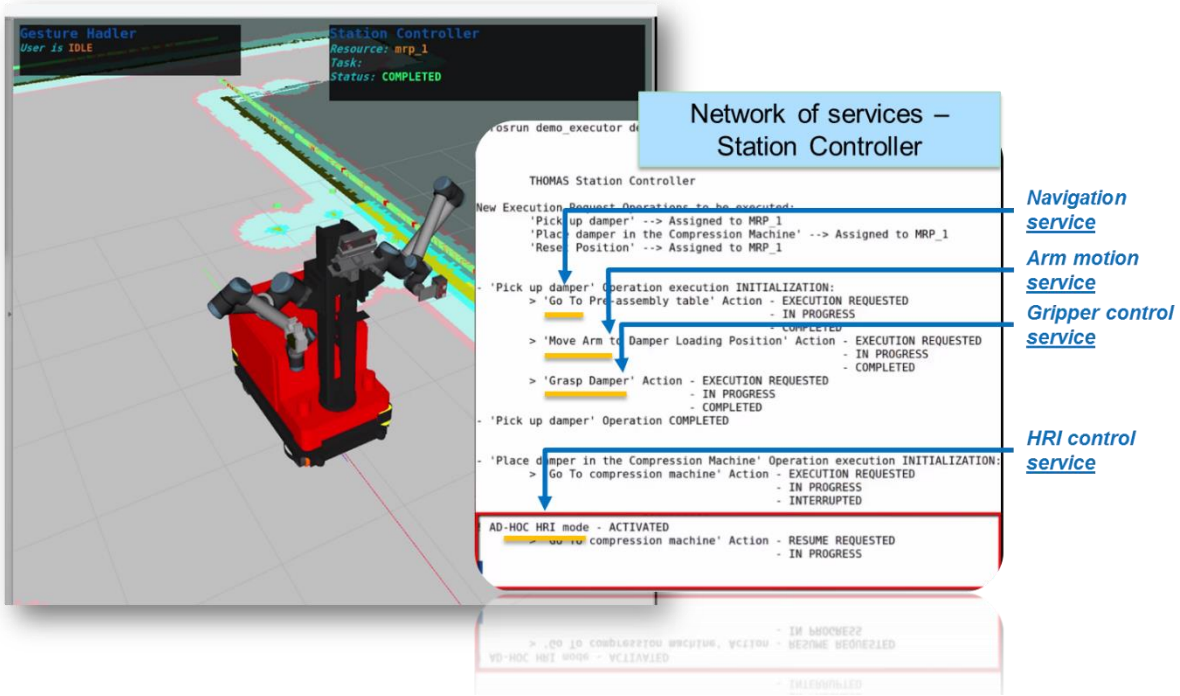

Figure 21: Network of services overall initial prototype

7. CONCLUSIONS

The document presented the initial prototype of the THOMAS Network of Services implementation. The prototype has been developed in accordance with the design that was documented under D5.1 and submitted on M12 based on the reference architecture design that was presented on D1.3.

The SONAR (Service Oriented Network Adjacent Resources) concept and initial prototyping has been detailed describing the mechanism that is employed for registering and exposing the available services to the entire THOMAS system. Following dedicated interfaces have been deployed for the integration / communication and data flow among the individual hardware and software components through the exposed services.

The initial prototype also includes the first implementation of the Station Controller through the deployment of multiple Station Agents mapped to the available resources. These agents are responsible for managing the execution control of the assigned tasks to the resources (see details in D5.2 "Dynamic work re-organization module – Initial prototype") while providing feedback on the execution status and their location. The Station Controller consumes the available services for realizing the scenario execution using the Environment / Process perception, the Safety system, the HRI mechanisms and the work re-organization module feedback.

For realizing the integration of these modules an agile approach for the developing – integrating – refinement loop has been defined. In particular, regular integration tests take place from this early beginning of the prototyping phase in order to accommodate on time the interfacing requirements leading to the THOMAS Open Production Station as a product.

For enabling the remote integration testing, the configured and customized virtual machine can be easily replicated and installed in local environments and can be used for testing as well as for integration of the developments. The current configuration is customized to allow the integration and testing in a remote, distributed environment by utilizing a VPN server.

The next steps in terms of the service-based integration and networking developed under workpackage 5 include the extension of the developed functionality in order to support the full functionality foreseen in the D5.1 and D1.3, as well as further integration and testing with the other developments of the project. The initial prototype will be further updated to support the integration and testing phase of the project. In addition, THOMAS Shared Data repository prototype is already under development as well as the THOMAS portal deploying all the required User Interfaces for user interaction with the central system. The final prototype of Network of services, offering the complete set of functionality will be provided on project month M36 with the deliverable D5.4 "THOMAS service based integration and communication network– Final Prototype".

8. GLOSSARY

MRP	Mobile Robot Platform
XML	eXtensible markup language
SONAR	Service Oriented Network Adjacent Resources
VPN	Virtual Private Network
ROS	Robot Operating System
VHD	Virtual Hard Disk
VM	Virtual Machine
SQL	Structured Query Language
DAO	Data Access Object
BO	Business Object
UML	Unified Modelling Language
ECU	Electronic Control Unit

9. REFERENCES

1. Ubuntu Linux Official Web Site, http://www.ubuntu.com/download/server, accessed 21.04.2015.
2. URL MySQL official site, Why MySQL, https://www.mysql.com/why-mysql/ , accessed 21.03.2018.
3. URL ROS actionlib http://wiki.ros.org/actionlib, accessed 21.03.2018