H2020 – FOF – 09 – 2015

Innovation Action

Smart integrated immersive and symbiotic human-robot collaboration system controlled by Internet of Things based dynamic manufacturing processes with emphasis on worker safety

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 680734

D4.2 Early version of the integrated platform and new Integration Plan

Report Identifier:	D4.2		
Work-package, Task:	WP4, Task 4.1	**Status – Version:**	1.00
Distribution Security:	CO	**Deliverable Type:**	D
Editor:	PROS		
Contributors:			
Reviewers:	TUM, CEA, ED		
Quality Reviewer:	ED		

Keywords:	
Project website: www.horse-project.eu	

Table of Contents

List of Tables

List of Figures

Abbreviations

CI	Continuous Integration
D2.1	Refers to HORSE Project deliverable D2.1 – System Requirements Specification
D2.2	Refers to HORSE Project deliverable D2.2 – Complete System Design
D3.2	Refers to HORSE Project deliverable D3.2 - Early prototype of the Manufacturing Process Management System (MPMS)
D3.4	Refers to HORSE Project deliverable D3.4 - Final Augmented Reality software for user assistance in a manufacturing work cell
D3.5	Refers to HORSE Project deliverable D3.5 - Final prototype of a process-execution environment for mixed-actor processes
D3.6	Refers to HORSE Project deliverable D3.6 - Early version of situation awareness software for human and non-human agents in a manufacturing work cell
D3.9	Refers to HORSE Project deliverable D3.9 - Final version of multi-modal monitoring sub-system
D3.10	Refers to HORSE Project deliverable D3.10 - HORSE Cross-Domain Messaging
D3.11	Refers to HORSE Project deliverable D3.11 - Final Version of HORSE Cross-Domain Messaging
D3.12	Refers to HORSE Project deliverable D3.12 - Middleware for HORSE Execution Domains
D3.13	Refers to HORSE Project deliverable D3.13 - Final Version of the Middleware for HORSE Execution Domains
D3.15	Refers to HORSE Project deliverable D3.15 - Final version of intuitive programming subsystem for robots and online, dynamic motion replanning
D4.1	Refers to HORSE Project deliverable D4.1 - Integration Plan and Description of the Integration Infrastructure
DBMS	Database management system
VCS	Version Control System

WP	Work package

Executive Summary

This deliverable describes the early version of the HORSE integrated platform. Various components and functions have been presented on the HORSE validation workshop in Munich in May 2017 and reported on the first annual review meeting in July 2017, while the final composition and verification was done in November 2015, indicating the successful pass of MS2. This early HORSE prototype validated the available functionality of the participating components demonstrating the optimal collaboration with each other (no boundary or crash tests performed). The integration strategy was adopted by the other components, who demonstrated basic collaboration capabilities.

The first introductory chapter describes the objectives and the scope of the document. The next chapter presents the components of the early prototype, their level of maturity and integration. Chapter 3 offers a short description of the target platform. Chapter 4 presents the taxonomy of event types used for creating of the HORSE messages, the key communication interface between the components. Chapter 5 describes the demonstration of the early prototype. The next chapters 6, 7 and 8 provide update of the integration plan, infrastructure and risks, that were introduced in D4.1. The document ends with a conclusion and list of next tasks for WP4.

1 Introduction

The HORSE platform is a flexible software system interacting with sensors, operators and various automation agents, engaged in an industrial process and enabling the safe collaboration of humans and robots. The key platform components have been described in HORSE deliverable D2.2 Complete System Design. These components have been implemented as part of Work package 3 activities and described in detail in the relevant WP3 deliverables.

The early HORSE prototype is a collection of HORSE components that interact with each other and demonstrate their capabilities by realizing a simple use case. This document contains description of the use case and the involved components. The document extends the integration plan presented in HORSE deliverable D4.1.

1.1 Objectives

The demonstrated early version of the HORSE platform marks the successful achievement of the following objectives:

Project health check

The HORSE project is a complex endeavour of a consortium of partners with different background and experience. The definition of internal milestones is vital for checking the health of the project and the achievability of the project goals. The early version of the HORSE integrated platform is such an important occasion for estimation of the progress, analysing the risks and designing and applying corrective measures.

Demonstration of the implemented features

The early integrated prototype should align the understanding of the technical partners about the further development and accommodation of the components they implement. Not of a lesser importance is the improved understanding of the non-technical partners about the platform features and limitations.

Validation of the collaboration of the modules

The early prototype contributed to the validation of the interfaces and the way the modules interact with each other. A critical prerequisite for a successful collaboration is the availability of a working middleware and persistent storage (databases).

Validation of the installation and configuration instructions

The activities to bring the components together contributed to a better understanding of the system requirements of the individual components and their installation and configuration specifics.

Validation and refinement of the integration plan

The integration of the early version of the HORSE platform helped validating the implementation strategy and plan published in D4.1.

1.2 Scope

The assembling of the early prototype of the HORSE platform is part of the integration efforts performed in Work package 4 “System integration, prototyping and technical verification”. The overall platform’s functionality, its context and scope are defined in the HORSE requirements specification (D2.1) and HORSE system architecture (D2.2). The section 3.4 of the initial integration plan (D4.1) denotes the bilateral relations of the components, defines several levels of maturity and proposes a timeline of the bilateral integration activities.

The early version of the HORSE platform is part of HORSE Milestone 2 and is vital for collection of the users’ feedback. This prototype, according to D4.1, should feature the implementation of the business logic for the high-priority interfaces and features (all HORSE components). The interfaces and business logic of the components should be implemented according to the implementation plan of WP3. The components should be able to exchange data and the received data is checked for consistency, but no complete handling of the deviations and exception will be expected.

The successful integration of the HORSE components depends on an elaborated and agreed taxonomy of the messages bearing the events and alerts, a reliably operating messaging middleware and the availability of the persistent data bases jointly used by the components. The HORSE partners under the guidance of TUE have specified and documented the HORSE alerts and events. The release of the HORSE Messaging Middleware, documented in D3.10 and D3.12 enables the transportation of the messages. The needed database structures have been defined and configured too.

The mapping of the integration phases to the system architecture is displayed on Figure 1.

The components in white are not part of the runtime system. Their interaction with the execution modules is limited to the provision of artefacts (scripts, models, data records) stored on the file repository or jointly used databases. The artefacts needed for the platform demonstration and validation will be created in advance in collaboration with the prospective users with the help of the available tools.

The components in blue have proved mature levels of implementation and integration as part of joint demonstrations.

The components in yellow demonstrated partial functionality and basic integration.

It needs to be pointed that the HORSE platform does not offer out of the box solution directly applicable to the end user needs, rather than the flexibility for customers to extend the available data structures, logical rules protocols to specific equipment.

In addition, the HORSE framework is just a prototype and does not provide complete handling of errors or boundary cases. The integration activities have been oriented to achieve the successful realisation of HORSE use cases and scenarios in their “best path of execution”.

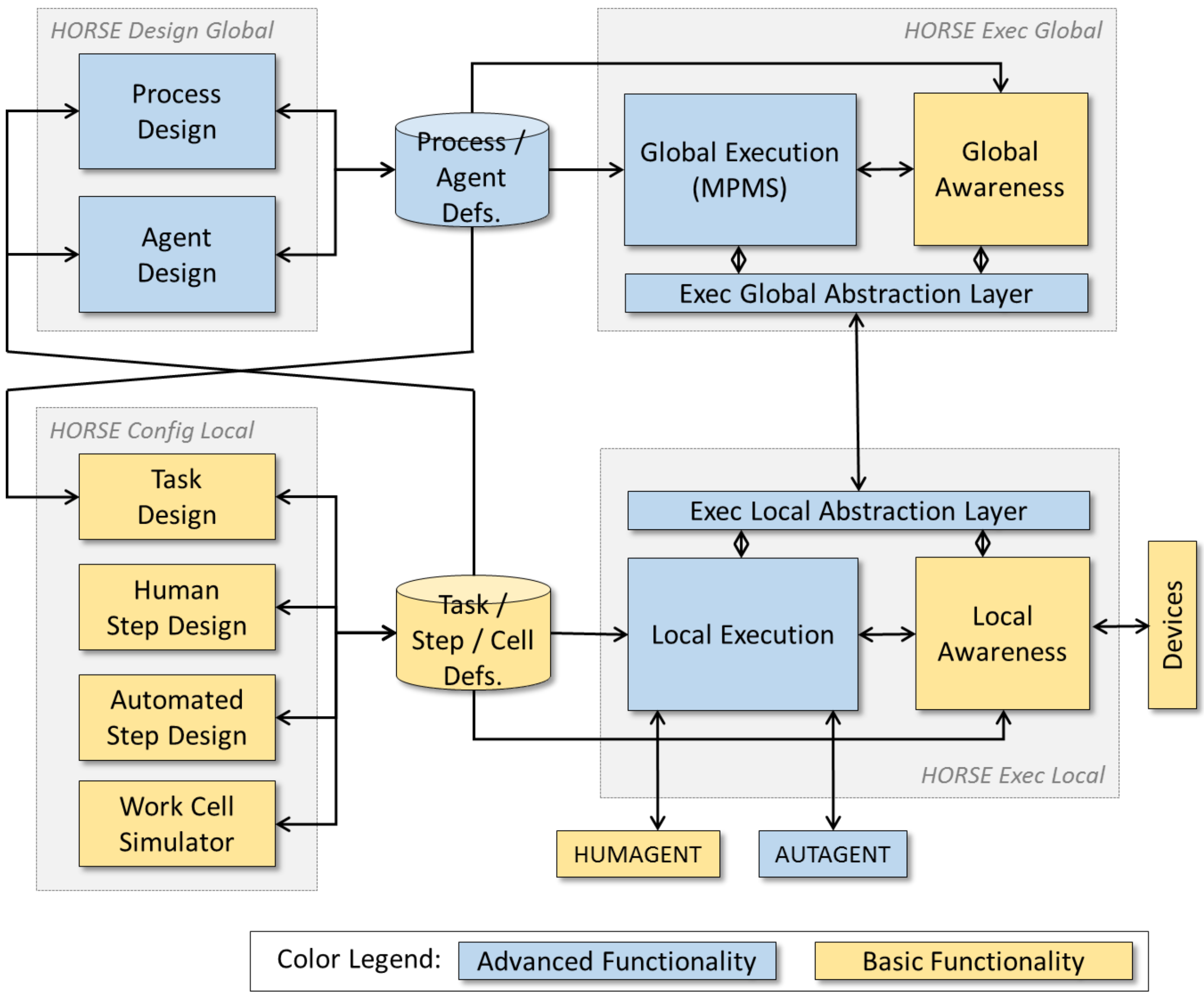


Figure 1: HORSE implementation and integration phases

The next table (**Error! Reference source not found.**) presents the functional relationships between the components (X) and indicates the level of integration achieved in the early prototype.

The components whose relationship is marked with "1" achieved the expected level of interaction and participated in the early use case.

The relationships marked with "2" point to partial integration demonstrated detached from the early use case.

	Middleware	Databases	Agent Mgr	Augmented Reality	AutAgent Step Exec	Cameras & Sensors	Conveyor Belt (BOS)	Deviation Monitor	Device Abstraction	Device Manager	Global Awareness	Global Execution	HumAgent Step Exec	Human Detection/Tracking	Human Machine Interface	Hybrid Task Supervisor	KUKA AutAgent INF	Local Safety Guard	Notification Beacon (BOS)	Object Detection/Tracking
Databases	X																			
Agent Mgr	2	X																		
Augmented Reality	2	X	X																	
AutAgent Step Exec	1		X																	
Cameras & Sensors				1																
Conveyor Belt (BOS)	2		X		2															
Deviation Monitor	2	X	X			2														
Device Abstraction	X					X														
Device Manager	X					X			X											
Global Awareness	2		X			2														
Global Execution	1	1	X								X									
HumAgent Step Exec	X		X	2																
Human Detection/Tracking	2	X	X			X			X											
Human Machine Interface	X		X	2									X							
Hybrid Task Supervisor	1	X	X		1		2					1	X							
KUKA AutAgent INF	1		X		1			X								1				
Local Safety Guard	2		X						X											
Notification Beacon (BOS)	X		X		X											X				
Object Detection/Tracking	2	X	X			1												2		
VisionControl (BOS)	2		X		2											2				

Table 1: Inter-component relations

2 Components of the Early HORSE Platform

The sections of this chapter provide a description of the implementation of the individual components of the early prototype.

Figure 2 presents the early prototype components and the realised interfaces. It needs to be pointed out that the Message Broker and the ROS Bridge are acting as message forwarders. The functional dependencies are depicted with blue arrows, while the paths that the messages travel between the functional components are given with red arrows.

As seen from the figure, some components are able to send messages, but they are either not capable to process them properly or the functional relations have not been fully demonstrated and for this reason these components have not been included in the integrated use case. Their functionality has been shown in bilateral demonstrations.

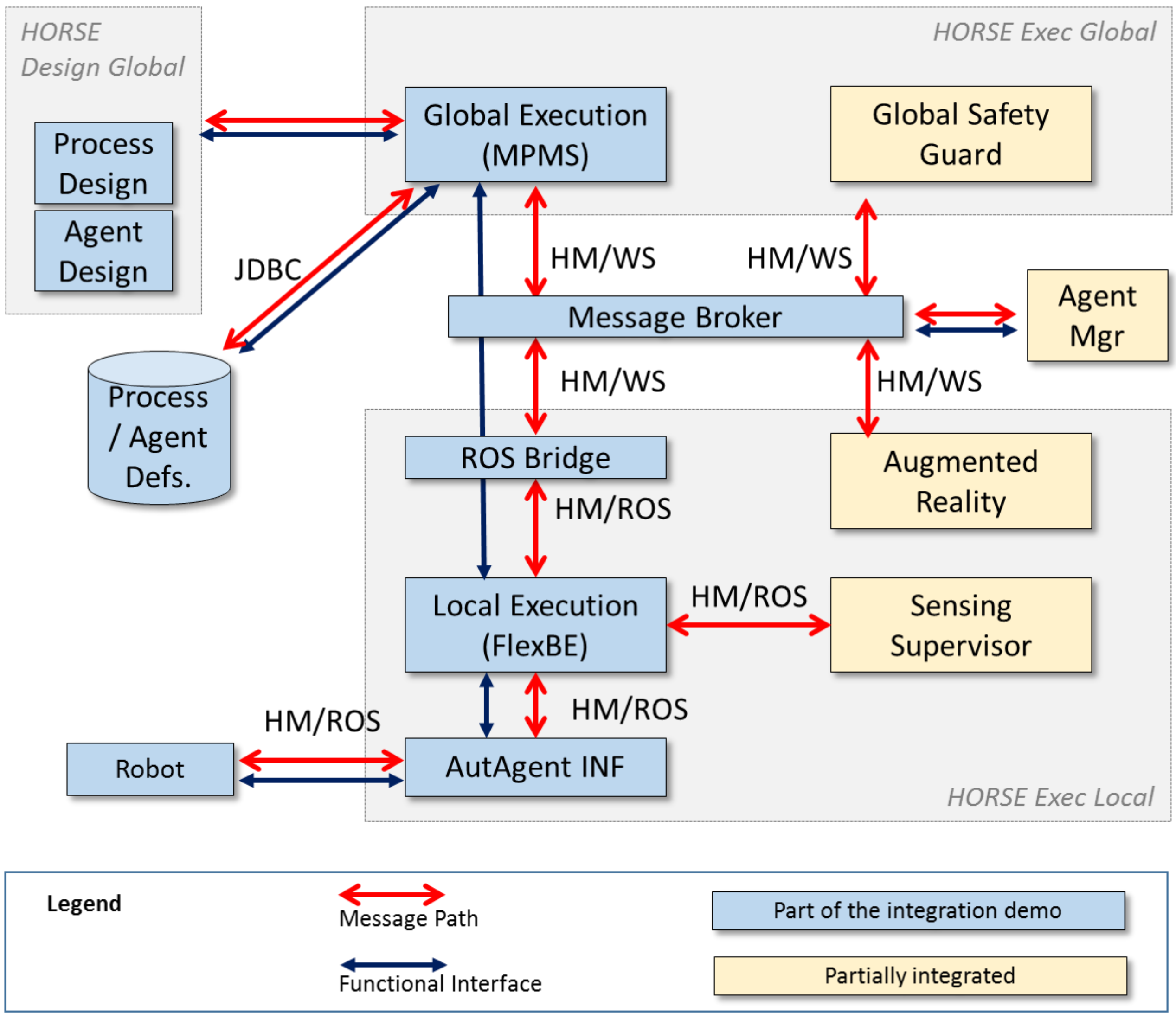


Figure 2: Early Prototype Components

2.1 Global Execution - MPMS

"The Manufacturing Process Management System (MPMS) is the collection of subsystems responsible to orchestrate the activities of agents in the manufacturing processes. Orchestration is dependent on the design of the processes and agents. The MPMS includes the functionality to design processes and agents, and execute the processes by allocating activities to agents." [D3.2]

The MPMS offers implementation of the components of the Global Execution functional group (part of the high level functional domain HORSE Exec Global, HEG). A detailed description of the component is provided in the HORSE deliverable "D3.5 Final prototype of a process-execution environment for mixed-actor processes". For this reason, the current section will only highlight the information and features that describe the integration of the module in the early HORSE prototype.

The MPMS version partaking in the early HORSE prototype provided functional maturity and could interact with the Messaging Broker and the components in HORSE Exec Local domain according to the specification.

2.1.1 Interfaces

2.1.1.1 Databases

In order to orchestrate the agents' tasks, the MPMS processes a workflow modelling the industrial process and communicates with the HORSE Exec Local components over the HORSE Messaging Middleware.

The workflow elements (processes, tasks, agents and products) are stored in separate tables of a RDBMS. For the early HORSE prototype these tables have been populated with sample data, but the structure of some tables (e.g. Product definitions) and the DB content need to be adapted to the end user scenario.

The MPMS is interacting with all tables listed in Section 2.6. The connection with the DB server is done JDBC protocol.

2.1.1.2 Messages

The MPMS is registered in the Messaging Middleware under the ID "'heg/global_execution/production_execution_control/" followed by an instance of the workflow execution.

The MPMS is capable of sending and processing the messages that follow the event taxonomy (Chapter 4). The next pages present few examples of the interaction with FlexBe (implementing the HybridTask Supervisor functionality).

2.1.1.2.1 Task assignment message from HEG (MPMS) to FlexBe

The MPMS instructs FlexBE (Hybrid Task Supervisor) to pick up an object via a task assignment message like this:

```
{
      'Topic': 'task_assigned',
      'Priority': '2',
      'SenderID':
'heg/global_execution/production_execution_control/91051701-fb73-
11e7-be4e-00059a3c7a00',
      'Receivers': 'rosbridge',
      'Type': '2',
      'Subtype': 'notification',
      'Timestamp': '20180117103317',
      'MessageID': '20180117103317',
      'ResponseMessageID': '',
      'Internal': 'true',
      'ExternalBrokers': '*',
      'SenderBroker': '',
      'Body': {
            'op': 'call_service',
            'service': '/task_request',
            'args': {
                  'agent_ids': '1',
                  'task_id': '3',
                  'task_instance_id': '504b5bbb-55a5-11e7-81c4-
2ae820524153',
                  'process_instance_id': '81d29cf3-fb6d-11e7-8d69-
5cab20524153'
            }
      }
}
```

The message is sent to the Messaging Broker, then forwarded to the ROS Bridge from there to FlexBe

2.1.1.2.2 Pickup task competition of reported by FlexBe to HEG (MPMS)

The task completion message from the Hybrid Task Supervisor is expected as:

```
{
      'Topic': 'task_completed',
      'Priority': '2',
      'ResponseMessageID': '',
      'Receivers':
'heg/global_execution/production_execution_control/91051701-fb73-
11e7-be4e-00059a3c7a00',
      'SenderID': 'rosbridge',
      'MessageID': '',
      'Subtype': 'notification',
      'Type': '2',
      'Timestamp': '20180117103347',
      'ExternalBrokers': '*',
      'Internal': 'true',
      'SenderBroker': '',
	'Body': {
		'EventID': 'GEV010',
		'Details': {
			'process_instance_id': '81d29cf3-fb6d-11e7-8d69-
5cab20524153',
			'task_instance_id': '504b5bbb-55a5-11e7-81c4-
2ae820524153',
			'task_id': '3'
		},
		'Variables': {},
		'Entity': 'Task',
		'State': 'Completed',
		'Event_Type': 'task_completed',
		'Event_Class': 'Progress'
	}
}
```

The message is sent by FlexBe through the ROS Bridge and the Messaging Broker.

2.1.1.2.3 Task assignment for visual check to FlexBe

With this HORSE Message the MPMS instructs FlexBe to perform Visual check

```
{
	'Topic': 'task_assigned',
	'Priority': '2',
	'SenderID':
'heg/global_execution/production_execution_control/91051701-fb73-
11e7-be4e-00059a3c7a00',
	'Receivers': 'rosbridge',
	'Type': '2',
	'Subtype': 'notification',
	'Timestamp': '20180117103348',
	'MessageID': '20180117103348',
	'ResponseMessageID': '',
	'Internal': 'true',
	'ExternalBrokers': '*',
	'SenderBroker': '',
	'Body': {
		'op': 'call_service',
		'service': '/task_request',
		'args': {
		'agent_ids': '3,1',
		'task_id': '4',
		'task_instance_id': '7cf90b89-fb6e-11e7-8d69-
5cab20524153',
		'process_instance_id': '81d29cf3-fb6d-11e7-8d69-
5cab20524153',
                'checkpoints': '6'
                }
        }
}
```

2.1.1.2.4 Visual check competition expected by FlexBe

The message on completion of visual check task is expected from the Hybrid Task Supervisor as:

```
{
        'Topic': 'task_completed',
        'Priority': '2',
        'ResponseMessageID': '',
        'Receivers':
'heg/global_execution/production_execution_control/91051701-fb73-
11e7-be4e-00059a3c7a00',
        'SenderID': 'rosbridge',
        'MessageID': '',
        'Subtype': 'notification',
        'Type': '2',
        'Timestamp': '20180117103405',
        'ExternalBrokers': '*',
        'Internal': 'true',
        'SenderBroker': '',
        'Body': {
                'EventID': 'GEV010',
                'Details': {
                        'process_instance_id': '81d29cf3-fb6d-11e7-8d69-
5cab20524153',
                        'task_instance_id': '7cf90b89-fb6e-11e7-8d69-
5cab20524153',
                        'task_id': '4'
                        },
                'Variables': {
                        'defect': {'value':true},
                        'defectimage': {'value': 'image'}
                },
                'Entity': 'Task',
                'State': 'Completed',
                'Event_Type': 'task_completed',
                'Event_Class': 'Progress'
        }
}
```

2.1.1.2.5 Placing task assignment to FlexBe

The MPMS instructs the FlexBe to place the object in the packaging box. The HORSE Message looks like this:

```
{
      'Topic': 'task_assigned',
      'Priority': '2',
      'SenderID':
'heg/global_execution/production_execution_control/91051701-fb73-
11e7-be4e-00059a3c7a00',
      'Receivers': 'rosbridge',
      'Type': '2',
      'Subtype': 'notification',
      'Timestamp': '20180117103406',
      'MessageID': '20180117103405',
      'ResponseMessageID': '',
      'Internal': 'true',
      'ExternalBrokers': '*',
      'SenderBroker': '',
      'Body': {
            'op': 'call_service',
            'service': '/task_request',
            'args': {
                  'agent_ids': '1',
                  'task_id': '5',
                  'task_instance_id': 'bfd23c21-fb6e-11e7-8d69-
5cab20524153',
                  'process_instance_id': '81d29cf3-fb6d-11e7-8d69-
5cab20524153'
            }
      }
}
```

2.1.1.2.6 Completion of the placing task

The MPMS expects from FlexBe a confirmation message upon the task completion like this:

```
{
      'Topic': 'task_completed',
      'Priority': '2',
      'ResponseMessageID': '',
      'Receivers':
'heg/global_execution/production_execution_control/91051701-fb73-
11e7-be4e-00059a3c7a00',
      'SenderID': 'rosbridge',
      'MessageID': '',
      'Subtype': 'notification',
      'Type': '2',
      'Timestamp': '20180117103415',
      'ExternalBrokers': '*',
      'Internal': 'true',
      'SenderBroker': '',
      'Body': {
            'EventID': 'GEV010',
            'Details': {
                  'process_instance_id': '81d29cf3-fb6d-11e7-8d69-
5cab20524153',
                  'task_instance_id': 'bfd23c21-fb6e-11e7-8d69-
5cab20524153',
                  'task_id': '5'
            },
            'Variables': {},
            'Entity': 'Task',
            'State': 'Completed',
            'Event_Type': 'task_completed',
            'Event_Class': 'Progress'
      }
}
```

2.1.2 Implementation Specifics

The MPMS is a Java product based on the Eclipse project Camunda. It could be started in Windows and Linux platforms. It requires a HTTP Server (e.g. Apache) and internal DB.

2.2 Messaging Middleware

The HORSE Messaging Middleware provides a uniform and platform neutral mechanism for exchanging JSON formatted messages between the HORSE components. The HORSE Architecture defines two types of functional domains for runtime components – a single HORSE Exec Global (acting as a Work Floor Manager) and one or more HORSE Exec Local (client platforms at the work cells). The communication within a single domain is documented in HORSE deliverable "D3.12 - Middleware for HORSE Execution Domains", while the message exchange between the domains is explained in "D3.10 - HORSE Cross-Domain Messaging".

All three components of the Messaging Middleware were available for integration and testing as follows:

- The Messaging Agent was implemented as part of MPMS, ROS Bridge, Augmented Reality, Global Safety Guard and KUKA Sunrise Platform, thus enabling their communication with the Messaging Broker;
- An instance of the Messaging Broker was deployed on the HORSE Integration Server, offering its functionality to all HORSE components capable of adequate participation in a joint test session. Due to their limited number it has been decided to perform the test sessions with just a single Broker and no Dispatcher.

- The Messaging Dispatcher has been tested separately with 3 Brokers and 3 dummy Agents.

2.2.1 Interfaces

2.2.1.1 Databases

The early versions of the Broker used no persistent databases.

2.2.1.2 Messages

The Messaging Broker receives all HORSE messages sent by the HORSE components by their implementation of the Messaging Agent. The Broker is processing only the message headers in order to determine the list of recipients. In order a component to be able to send and receive messages over the Messaging Middleware it should send a control message to the Broker. The format of this type of messages is elaborated in D3.11.

2.2.2 Implementation specifics

The Messaging Broker and the Messaging Dispatcher have been implemented as OSGi applications deployed and executed in the ProSyst mBS (an implementation of OSGi Framework). The mBS, being a Java based product, could be deployed in a great range of hardware platforms (from Raspberry Pi to powerful servers) with Java support.

2.3 Local Execution - FlexBe

This component provides an implementation of the Hybrid Task Supervisor, the key component in the HORSE Exec Local functional group. FlexBe is processing the task assigned by MPMS into a number of execution steps and orchestrates their execution by the available agents. This module has been implemented with Python and interacts with the other Python and C++ components (collision avoidance, object recognition…) and the robot interfaces via ROS. The communication with the non-ROS components and the HORSE Messaging Middleware is done though the ROS Bridge.

FlexBe could be used for designing the robotic operations and trajectories with the help of GPU-Voxel module (a C++ implementation).

FlexBe demonstrated a mature level of functionality and was capable to interact with the other HORSE components as expected.

This module has been documented in HORSE deliverable "D3.15 Final version of intuitive programming subsystem for robots and online, dynamic motion replanning".

2.3.1 Interfaces

2.3.1.1 Databases

This module does not interact with the DB server.

2.3.1.2 Messages

FlexBe is capable of receiving and processing task assignments from MPMS as described in Section 2.1.1.2 (MPMS Messages). These messages have been relayed through the Messaging Broker and the ROS Bridge. The messages sent to FlexBe should be addressed to "rosbridge" and should bear a parameter "service" with the value "/task_request" in the message body.

The communication with the automated agent (Universal Robot) has been done with the help of scripts of standard ROS commands like this:

```
/execute_saved_trajectory/goal
moveit_action_wrapper/ExecuteTrajectoryGoal
file_name:
'package://bosch_trajectories/trajectories/01_home_to_pickup.json
'
trajectory: ' '
```

2.3.2 Implementation specifics

This module is implemented in Python and requires Linux OS (Ubuntu 16.04). The Step Definitions utilised by FlexBe are presented in the form of ROS scripts, stored in the local file system.

In case the GPU-Voxel modul is used for design and tracking of the robot operations and movements, a GPU Voxel is needed.

2.4 ROS Bridge

This module translates the JSON formatted messages exchanged over the HORSE Middleware into ROS messages and vice versa. In this way the components already utilising ROS messaging are provided access to the non-ROS. The IDs of the ROS components are provided as parameters in the HORSE Messages and thus they could be globally addressed.

The component demonstrates mature functionality and interacts with the other components as expected.

2.4.1 Interfaces

2.4.1.1 Databases

The ROS Bridge does not utilise persistent data and thus no interaction with the DB server is needed.

2.4.1.2 Messages

The ROS Bridge component ID when exchanging messages over the HORSE Messaging Middleware is "rosbridge". Its only role is to forward these messages to the ROS proper component as provided in the "service" property in the message body.

Examples of such addressing could be seen in Section 2.1.1.2 (MPMS Messages).

2.4.2 Implementation specifics

The ROS Bridge is implemented in Python and requires Linux OS (e.g. Ubuntu 16.04)

2.5 Robot Interface

The FlexBe capabilities to interact with automated agents have been demonstrated with a Universal Robot. The robot was able to execute operations on pick-up of an object, its relocation and placement at the desired location. The operation details have been designed by FlexBe and commissioned to the robot via ROS messages.

2.5.1 Interfaces

2.5.1.1 Databases

This module does not interact with the HORSE DB.

2.5.1.2 Messages

The robot (automated agent) is not directly addressed via the HORSE Messaging Middleware. It interacts exclusively with the Hybrid Task Supervisor (FlexBe). This communication is done via messages within the ROS domain.

2.5.2 Implementation specifics

The Robot Interface utilises the ROS capabilities of the used robot.

2.6 Database Server

The definitions of objects, tasks and agents are stored in persistent database with the following structure.

In the early HORSE prototype, the DB server has been used mainly by MPMS.

2.6.1 TABLE ability

Description:

This table stores data describing the capabilities of the production agents, so that they are filtered and evaluated against the requirements of the tasks. As result, the agents are assigned to the proper tasks.

Fields:

ability_id bigint NOT NULL,

ability_name character varying(50),

ability_descr character varying(200)

2.6.2 TABLE agent

Description:

This table stores references to all agents.

Fields:

agent_id bigint NOT NULL,

agent_name character varying(50),

agent_role_id integer,

agent_operation_status character varying(20)

2.6.3 TABLE autoagent

Description:

This table stores the data of the automated agents.

Fields:

auto_agent_id bigint NOT NULL,

auto_agent_name character varying(255),

auto_agent_operation_status character varying(255),

auto_agent_role_id bigint,

auto_agent_team_id bigint

2.6.4 TABLE box

Description:

This table is specific for BOS pilot and stores the parameters of the boxes for packaging the WSAs.

Fields:

box_id bigint NOT NULL,

box_no character varying(20),

box_pn_no character varying(20),

box_pn_type character varying(20),

box_capacity integer,

box_layer_capacity integer

2.6.5 *TABLE humanagent*

Description:

This table stores the data of the human agents.

Fields:

human_agent_id bigint NOT NULL,

human_agent_name character varying(255),

human_agent_operation_status character varying(255),

human_agent_role_id bigint,

human_agent_team_id bigint

2.6.6 *TABLE partnowsas*

Description:

Contains the part numbers of the WSAs (BOS UC).

Fields:

pn_id bigint NOT NULL,

pn_no character varying(20) NOT NULL,

pn_quantity integer,

pn_type character varying(20),

pn_checkpoints integer

2.6.7 *TABLE processdef*

Description:

This table stores the process definitions.

Fields:

process_id bigint NOT NULL,

process_name character varying,

process_descr character varying

2.6.8 TABLE role

Description:

This table stores the roles of the agents.

Fields:

role_id bigint NOT NULL,

role_name character varying(50)

2.6.9 TABLE taskdef

Description:

This table stores the task definitions.

Fields:

task_id bigint NOT NULL,

task_name character varying(80),

task_process_id integer

2.6.10 TABLE tool

Description:

This table stores the data of the tools involved in the process.

Fields:

tool_id bigint NOT NULL,

tool_sn character varying(20),

tool_type character varying(50),

tool_capacity character varying(20)

2.6.11 TABLE wsa

Description:

This table stores the WSA data (BOS UC).

Fields:

wsa_id bigint NOT NULL,

wsa_sn character varying(20) NOT NULL,

wsa_pn_id integer

2.6.12 *TABLE wsabatch*

Description:

This table stores the data of the batches of WSAs (BOS Pilot).

Fields:

batch_id bigint NOT NULL,

batch_no character varying(255),

batch_quantity integer,

batch_type character varying(255)

2.7 *Design Global (MPMS)*

This tool has been used for designing the global production workflow and its artefacts (agents, products, processes). A basic implementation of this component is provided by Camunda (realising the MPMS). This module interacts only with MPMS and the DB Server. An extended description is provided by HORSE deliverable D3.5.

2.7.1 *Interfaces*

2.7.1.1 Databases

The module uses all tables used by MPMS and listed in Section 2.6

2.7.1.2 Messages

The module interacts only with MPMS and does offer or utilises any public interfaces.

2.7.2 *Implementation specifics*

The Design Global tool is part of MPMS.

2.8 *Augmented Reality*

This module helps the human agents (operators) to perform complex operations providing instructions and guidance through the process. The custom specific implementation efforts are very high and required proper equipment that was not available during the phase of initial integration. For this reason, the module functionality was demonstrated aside from the joint test session. The communication with the Messaging Middleware has been validated at level of exchanging of dummy messages. Further specification of the message payloads to the other HORSE components is in process.

An implementation of this module has been presented by TNO as documented in HORSE deliverable D3.4 Final Augmented Reality software for user assistance in a manufacturing work cell.

An alternative implementation of AR is being developed by TUM for the BOS Pilot site.

2.8.1 Interfaces

2.8.1.1 Databases

None of the AR implementations at that stage interacted with the DB Server.

2.8.1.2 Messages

The AR module of TNO successfully registered an Agent in the HORSE Messaging Middleware under the ID “hel/local_execution/humagent_step_execution”. It was able to receive and response to MPMS messages like these:

Task assignment (MPMS to AR)

```
{
     'Topic': 'task_assigned',
     'Priority': '2',
     'SenderID':
'heg/global_execution/production_execution_control/04c980ba-d1c0-
11e7-8d1e-dc3e20524153',
     'Receivers': 'hel/local_execution/humagent_step_execution',
     'Type': '2',
     'Subtype': 'notification',
     'Timestamp': '20170620115843',
     'MessageID': '20170620115843',
     'ResponseMessageID': '',
     'Internal': 'true',
     'ExternalBrokers': '*',
     'SenderBroker': '',
     'Body': {
          'agent_ids': '',
          'task_id': '19',
          'task_instance_id': '504b5bbb-55a5-11e7-81c4-
2ae820524153',
          'process_instance_id': 'b6276ad0-5033-11e7-be1e-
f65820524153'
     }
}
```

Report on task completion (AR to MPMS)

```
{
     'Topic': 'task_completed',
     'Priority': '2',
     'ResponseMessageID': '20170620115843',
     'Receivers':
'heg/global_execution/production_execution_control/04c980ba-d1c0-
11e7-8d1e-dc3e20524153',
     'SenderID': 'hel/local_execution/humagent_step_execution',
     'MessageID': '',
     'Subtype': 'notification',
     'Type': '2',
     'Timestamp': ' '20170620120343 ',
     'ExternalBrokers': '*',
     'Internal': 'true',
     'SenderBroker': '',
     'Body': {
          'EventID': 'EV108',
          'Details': {
               'process_instance_id': 'b6276ad0-5033-11e7-be1e-
f65820524153',
               'task_instance_id': '504b5bbb-55a5-11e7-81c4-
2ae820524153',
               'task_id': '19'
          },
          'Variables': {
               'aStringVariable': {'value':'aStringValue'},
               'aBooleanVariable': {'value': true}
          }
     }
}
```

2.8.2 Implementation specifics

The AR module provided by TNO has been implemented as Python application. The module utilises a MS Kinect sensor to track the hand movements of the operator and a beamer to project instructions.

2.9 Sensing Supervisor

This module utilises IR cameras to track the motion of objects and humans, GPU-Voxel to calculate and visualize images and trajectories in order to identify possible collisions and undertake measures for their avoidance as documented in HORSE deliverable D3.9 Final version of multi-modal monitoring sub-system.

The module has demonstrated a mature functionality and was capable of exchanging sample messages over the Messaging Middleware. A further specification of the messages according to the taxonomy provided in Chapter 4 in in process.

2.9.1 Interfaces

2.9.1.1 Databases

This module does not interact with the DB Server.

2.9.1.2 Messages

The module was able to send control messages to the Broker and a dummy payload message. A further specification of the payload messages is necessary in order to achieve a full integration with the other HORSE components.

2.9.2 Implementation specifics

This component has been implemented as C++ module.

Requires IR cameras and GPU-Voxel

2.10 Global Safety Guard

This module is capable of processing events and sensor data originating in multiple work cells and thus identify safety risks on global level. The early prototype of this module has demonstrated proper analysis and handling of the incoming data and was able to exchange sample messages over the HORSE Messaging Middleware. A detailed description of this module is provided in HORSE deliverable D3.6 - Early version of situation awareness software for human and non-human agents in a manufacturing work cell.

2.10.1 Interfaces

2.10.1.1 Databases

This version of the module does not interact with the DB server

2.10.1.2 Messages

The module was able to send sample alert messages. A further specification of the payload messages is necessary in order to achieve a full integration with the other HORSE components.

2.10.2 Implementation specifics

Not known

2.11 Agent Manager

This module is implemented as part of the Messaging Broker and provides information about the available messaging agents. It could be used by the MPMS to ensure dynamic assignment and reassignment of tasks to the available agents. Similarly, it could be used by the other members of the distributed execution environment to check if the intended recipient of their messages is available.

The module functionality has been successfully tested, but not utilised by the current versions of the other HORSE components.

2.11.1 Interfaces

2.11.1.1 Databases

This module does not handle with persistent data and thus it does not interact with the DB server.

2.11.1.2 Messages

The Agent Manager processes HORSE Messages of type System. An initial description is available in Section 3.4.3 Retrieving System Information of HORSE Deliverable D3.12

2.11.2 Implementation specifics

The Agent Manager is implemented as OSGi module executable in ProSyst mBS, a Java implementation of OSGI Framework specification.

3 Target Platform

The HORSE Framework provides a software system to be deployed in industry facilities. It features a Work Floor Manager (or HORSE Exec Global, i.e. MPMS) and HORSE Exec Local instances at the work cells. Depending on the number of work cells and number of HORSE modules requested by the end user as well as their physical distribution there are several deployment options.

3.1 Single platform

This setup is possible in case of fewer components interacting with agents and equipment in a close distance. In this case the system requirements of these components are not contradicting or competing with each other it is possible to deploy all components on a single HW platform. Such a deployment is planned for BOS pilot site. The characteristics of such platform are as follows:

- CPU – at least Intel core i5
- RAM – at least 8 GB
- Disk space – at least 10 GB
- GPU-Voxel if the Sensing Supervisor is needed.
- OS – Linux Ubuntu 16.04
- Java Runtime 8
- Apache Web Server
- postgresql database engine

Automated agents (robots), beamers, cameras, displays and additional sensors according to the needs of the end user.

3.2 Multiple platforms

In this case the HORSE Framework is deployed on several HW platforms, connected with each other via local network.

The HW platforms could have similar profile as the one above. Depending on the expected load, alternative implementations of the existing HORSE components, the need of special communication protocols (e.g. etherCAT), slight differences in one or more of the HW specifications should be expected.

Since the HORSE framework features only one Work Floor Manager, one of the platforms should host the MPMS. The recommended location of the DB server is on the same platform as MPMS, but a distribution of the bases is possible, if needed.

4 Taxonomy of Interfaces and Alerts

The messages related to the production or safety aspects are subject of strict format, where some of the properties are predefined as follows.

4.1 *Event Classes*

The event classes reflect the highest possible level of classification

Event Class	**Description**
Notification	Standard events related to the normal execution of the workflows
Compensation	Deviation of the expected behaviour
Critical	Critical condition with a defined resolution process
Failure	Identified failure of an assigned task
Undeveloped	Reserved for further specification, if needed

Table 2: Event Classes

4.2 *Event Types*

In addition to the event class each event is attributed to a given event type. The next subsections list the event types with respect of the following entities:

- Process
- Task
- Step
- Agent
- Product
- Object

4.2.1 *Process Event Types*

These are the events related to the progress of the production process.

State	**Event type (indicates that corresponding state is reached)**	**Event class**	**Condition**	**Action**	**Responsible module**
Draft (design in progress)	process_created	Notification	Design-time	Create new process entry in data store	Process flow modelling
Available (design approved)	process_defined	Notification	Design-time	Save process definition in data store	Process flow modelling

	process_started	Notification	Normal operation	Control process flow	Production execution control
Execution	process_resumed	Notification	Process is in waiting state	Control process flow	Production execution control
Waiting	process_paused	Notification	Normal operation	Wait until process_resumed event is received	Production execution control
Completed	process_completed	Notification	Normal operation	Close process instance	Production execution control
			Out-of-normal recognised according to process heuristics	Initiate corrective action according to process control flow	Production execution control
Out-of-normal	process_irregular	Compensation	Unknown issue detected	Calculate risk factor according to operating parameters	Global safety guard
			No safety risk	Initiate corrective action according to process control flow	Next task selection
Suspended	process_interrupted	Critical	Safety risk present	Halt production	Global safety guard
			Production incomplete	Notify production planner	Worklist delivery
			Production complete	Close process instance	Production execution control
Terminated	process_terminated	Failure	Safety risk present	Halt all production	Global safety guard

Table 3: Process Event Types

4.2.2 Task Event Types

The events of these types are related to the assignment and execution of the individual production tasks.

State	**Event type (indicates that corresponding state is reached)**	**Event class**	**Condition**	**Action**	**Responsible module**
Draft	task_created	Notification	Design-time	Create new task entry in data store	Task identification

Available	task_defined	Notification	Design-time	Save task definition in data store	Task parser
Active (queued in process logic)	task_activated	Notification		Select next task to be executed	Next task selection
Assigned (to a team)	task_assigned	Notification	Team formed	Send task instruction to agent(s)	Worklist delivery
	task_assigned_failed	Undeveloped	No appropriate agents found	Send notification to responsible person	Production execution control
Execution	task_started	Notification	Normal operation	Subscribe to task events	Event processing
Completed	task_completed	Notification	Task successfully completed	Follow process control flow	Production execution control
Suspended	task_interrupted	Critical	Out-of-normal	Initiate corrective action according to process control flow	Next task selection
			Safety risk present	Halt production	Local safety guard
Cancelled	task_cancelled	Failure	Out-of-normal	Notify responsible person	Agent selection

Table 4: Task Event Types

4.2.3 Step Event Types

The events of these type are related to the execution of the production steps.

State	**Event type (indicates that corresponding state is reached)**	**Event class**
Draft	step_created	Notification
Approved	step_defined	Notification
Active (queued in task logic)	step_activated	Notification
Execution	step_started	Notification
Suspended	step_interrupted	Critical
Completed	step_completed	Notification

Table 5: Step Event Tasks

4.2.4 Agent Event Types

The events of these types identify the availability and status of the production agents

State	Event type (indicates that corresponding state is reached)	Event class	Condition	Action	Responsible module
Draft	agent_created	Notification	Design-time	Create new agent entry in data store	Design agent
Defined (but not yet active)	agent_defined	Notification	Design-time	Save agent definition in data store	Design agent
Idle	agent_activated	Notification	Connected to service directory	Agent publishes its services as available	??
		Notification	No connection to service directory	Notify responsible person	??
	agent_released	Notification	Task successfully completed	Agent makes its services available again	??
Assigned (to a team)	agent_assigned	Notification	Agent currently idle	Assigned task added to agent tasklist	Worklist delivery
		Notification	Agent currently occupied	Assigned task added to agent tasklist	Worklist delivery
		Compensation	Agent currently stalled or stuck	Task assignment rejected	Production execution control
Occupied	agent_started	Notification	Normal operation	Perform normal task as dictated by process control flow	Hybrid task supervisor
		Compensation	Out-of-normal	Perform corrective action according to priority list	Hybrid task supervisor
		Notification	Teaching		
Inactive	agent_deactivated	Notification	Normal operation	Remove agent services from directory	??
Stalled	agent_halted	Critical	No other steps in queue	Attempt to resolve problem according to internal heuristics	Hybrid task supervisor
			Other steps available in queue	Attempt to perform alternative steps	Hybrid task supervisor

Stuck	agent_failed	Failure	Safety risk present	Halt all movement and sound alarm	Local awareness
			No safety risk	Attempt to resolve problem according to internal heuristics	Local awareness

Table 6: Agent Event Types

4.2.5 Product Event Types

The events of these types are related to the status of completion and quality of the products

State	Event type (indicates that corresponding state is reached)	Event class	Condition	Action	Responsible module
Draft	product_created	Notification	Design-time		
Defined	product_defined	Notification	Design-time		
In production	product_started	Notification	Normal operation		
In storage	product_stored	Notification	Normal operation		
Produced	product_completed	Notification	Normal operation		
Delivered	product_distributed	Notification	Normal operation		
Defective	product_nonconformance_detected	Critical	Repair possible	Initiate corrective action according to process control flow	Next task selection
			Repair impossible	Notify responsible person	Agent selection
Discarded	product_discarded	Failure	Out-of-normal	Notify responsible person	Agent selection
Discontinued	product_discontinued	Notification	Normal operation		

Table 7: Product Event Types

4.2.6 *Object Event Types*

The events of these types bear information about the objects and tools used in the production

Usable	object_activated	Notification
	object_released	Notification
In use	object_seized	Notification
Unusable	object_failed	Failure
	object_deactivated	Critical
Unknown	object_unknown	Undeveloped

Table 8: Object Event Types

4.3 *Event List*

The following events have been handled by the early prototype. They are based on the events specified for the Bosch pilot site.

Event ID	**Event name**	**Description**	**Event_type**
EV001	WSA arrives	The first wiper system assembly arrives for visual control and packaging.	process_started
EV002	Alert: Defect detected	Visual control camera indicates that the WSA is non-conforming.	process_irregular
EV003	1 minute without response	The operator fails to provide a response within one minute after a WSA is flagged as non-conforming.	process_irregular
EV004	Alert: Team leader notified of defect	A notification sent to the team leader if the operator fails to provide a response within one minute.	process_progress
EV005	Team leader received defect message	The team leader is informed that a defect was detected and not handled within the designated time.	process_progress
EV006	Intervention request accepted	The team leader indicates that he/she will handle the unhandled defect.	process_progress
EV007	Intervention request rejected	The team leader acknowledges the defect, but indicates that he/she can't assist with it.	process_progress
EV008	4 minutes without response	The team leader fails to provide a response within four minutes after a WSA is flagged as non-conforming.	task_interrupted
EV009	WSA placed in secondary rejected box	A true defect or unmanaged suspected defect is placed in a secondary box for later.	process_interrupted

EV010	Rejected box full	No space available in the secondary box.	task_interrupted
EV011	Production process halted	Production is stopped because of excessive defects or no space in the secondary box.	process_terminated
EV012	False negative recorded	The human agent determined that the defect is flagged incorrectly.	process_progress
EV013	Defect corrected	The human agent was able to repair the defect within the designated time.	process_progress
EV014	True defect evaluated	The human agent determined that the WSA was defective and determined the cause of the defect.	process_irregular
EV015	Predefined layer count reached	Depending on the packaging size, an alert is triggered an X number of WSA's in advance of a full layer.	process_progress
EV016	New layer available	The human agent indicates that a sheet has been placed and the layer is available	process_progress
EV017	Predefined box count reached	Depending on the packaging size, an alert is triggered an X number of WSA's in advance of a full box.	process_progress
EV018	New box available	The human agent indicates that a new packaging box is placed allowing packaging to continue.	process_progress
EV019	Layer full	The HORSE System detects that the layer is full, preventing further packaging	task_interrupted
EV020	WSA packaging completed	WSA successfully placed in packaging and process instance is completed	process_completed

Table 9: Event List

5 Prototype Demonstration

The components of the early prototype were engaged in the execution of a joint demonstration.

HORSE
Manufacturing Process Management
with Robot Task Synchronization
FZI
TU/e Technische Universiteit Eindhoven University of Technology
Where innovation starts
BOSCH
Invented for life

Figure 3: Early Prototype Demonstration - Banner with the participants of the demo

The workflow executed by MPMS contained several tasks and condition points. The task assignments were sent over the Messaging Broker and ROS-Bridge to FlexBe. FlexBe operated the robot and reported completion of the tasks back to the MPMS. The MPMS decided for the next task and the process was repeated.

As a very first step, the MPMS and the ROS Bridge registered their agents in the Messaging Broker.

The MPMS executed the workflow based on the model presented in the next figure.

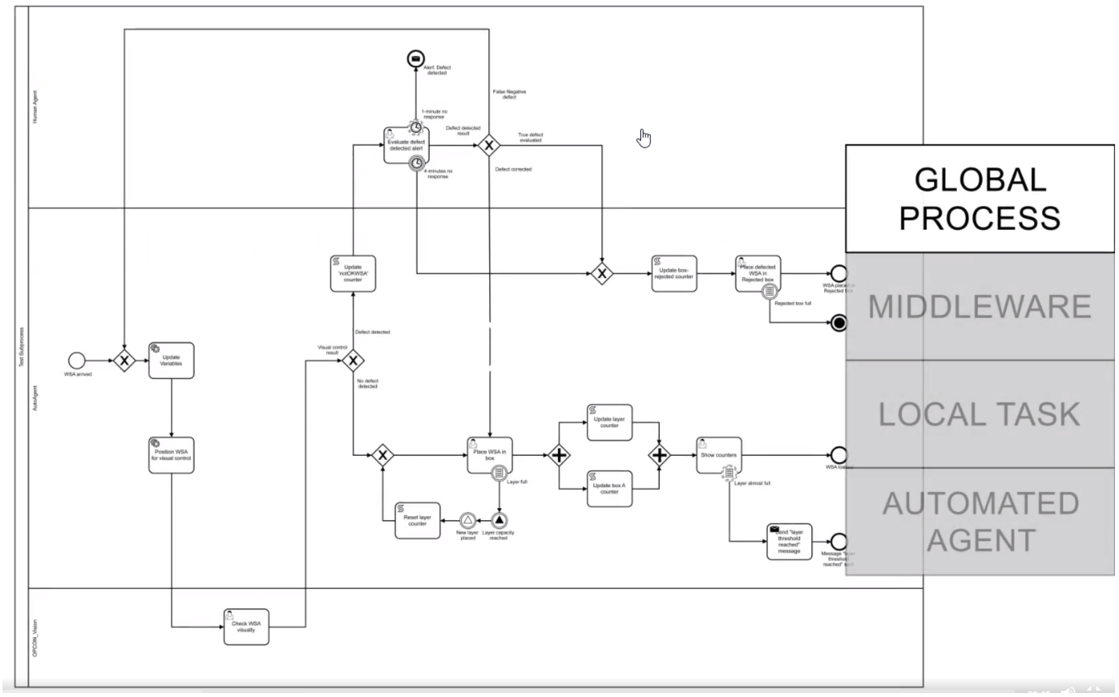


Figure 4: Early Prototype Demonstration - MPMS Workflow

The MPMS initiated the execution of a given task

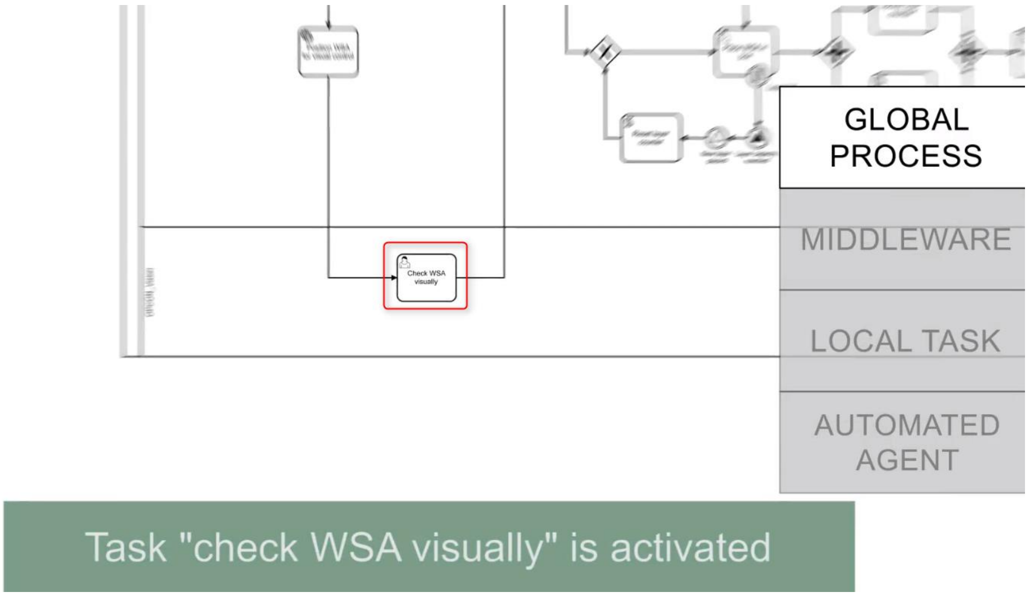
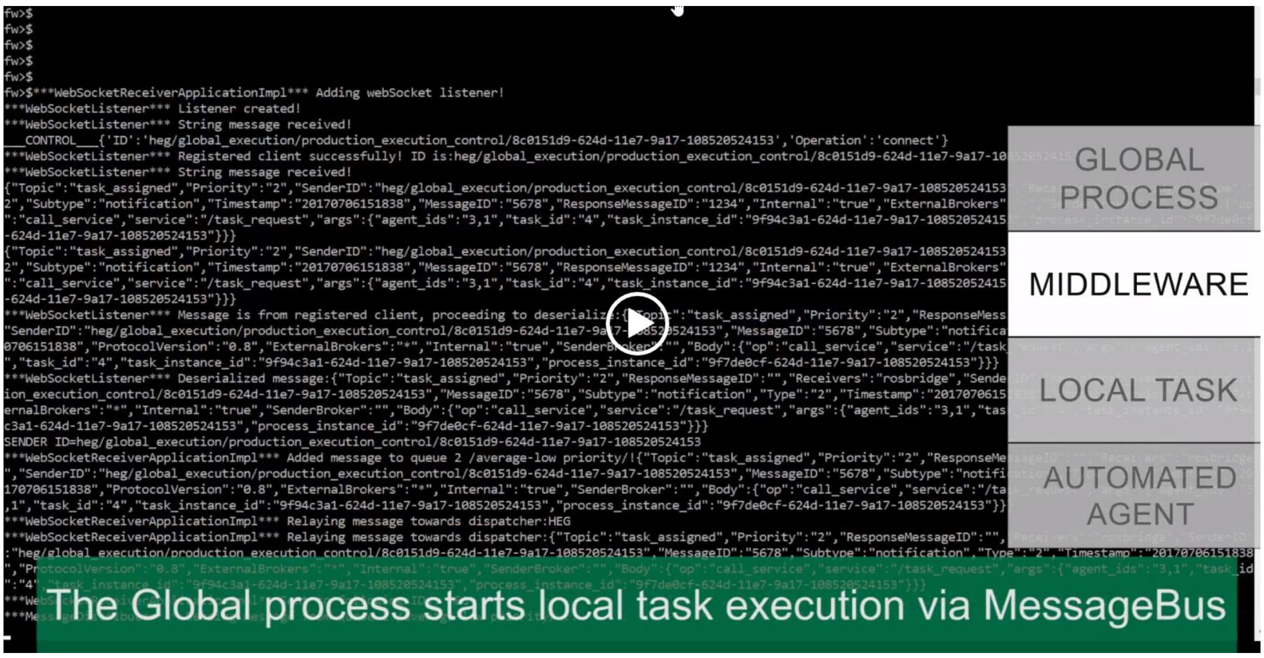


Figure 5: Early Prototype Demonstration - Task assignment by MPMS

The MPMS sent the task assignment through a payload message (like the one in Section 2.1.1.2.1) over the HORSE Messaging Middleware. The Broker received the message and determined its recipients.



```
fw>$
fw>$
fw>$
fw>$
fw>$
fw>$***WebSocketReceiverApplicationImpl*** Adding webSocket listener!
***WebSocketListener*** Listener created!
***WebSocketListener*** String message received!
___CONTROL___{'ID':'heg/global_execution/production_execution_control/8c0151d9-624d-11e7-9a17-108520524153','Operation':'connect'}
***WebSocketListener*** Registered client successfully! ID is:heg/global_execution/production_execution_control/8c0151d9-624d-11e7-9a17-10
***WebSocketListener*** String message received!
{"Topic":"task_assigned","Priority":"2","SenderID":"heg/global_execution/production_execution_control/8c0151d9-624d-11e7-9a17-108520524153
2","Subtype":"notification","Timestamp":"20170706151838","MessageID":"5678","ResponseMessageID":"1234","Internal":"true","ExternalBrokers"
":"call_service","service":"/task_request","args":{"agent_ids":"3,1","task_id":"4","task_instance_id":"9f94c3a1-624d-11e7-9a17-10852052415
-624d-11e7-9a17-108520524153"}}}
{"Topic":"task_assigned","Priority":"2","SenderID":"heg/global_execution/production_execution_control/8c0151d9-624d-11e7-9a17-108520524153
2","Subtype":"notification","Timestamp":"20170706151838","MessageID":"5678","ResponseMessageID":"1234","Internal":"true","ExternalBrokers"
":"call_service","service":"/task_request","args":{"agent_ids":"3,1","task_id":"4","task_instance_id":"9f94c3a1-624d-11e7-9a17-10852052415
-624d-11e7-9a17-108520524153"}}}
***WebSocketListener*** Message is from registered client, proceeding to deserialize:{"Topic":"task_assigned","Priority":"2","ResponseMess
"SenderID":"heg/global_execution/production_execution_control/8c0151d9-624d-11e7-9a17-108520524153","MessageID":"5678","Subtype":"notifica
0706151838","ProtocolVersion":"0.8","ExternalBrokers":"*","Internal":"true","SenderBroker":"","Body":{"op":"call_service","service":"/task
","task_id":"4","task_instance_id":"9f94c3a1-624d-11e7-9a17-108520524153","process_instance_id":"9f7de0cf-624d-11e7-9a17-108520524153"}}}
***WebSocketListener*** Deserialized message:{"Topic":"task_assigned","Priority":"2","ResponseMessageID":"","Receivers":"rosbridge","Sende
ion_execution_control/8c0151d9-624d-11e7-9a17-108520524153","MessageID":"5678","Subtype":"notification","Type":"2","Timestamp":"2017070615
ernalBrokers":"*","Internal":"true","SenderBroker":"","Body":{"op":"call_service","service":"/task_request","args":{"agent_ids":"3,1","tas
c3a1-624d-11e7-9a17-108520524153","process_instance_id":"9f7de0cf-624d-11e7-9a17-108520524153"}}}
SENDER ID=heg/global_execution/production_execution_control/8c0151d9-624d-11e7-9a17-108520524153
***WebSocketReceiverApplicationImpl*** Added message to queue 2 /average-low priority/!{"Topic":"task_assigned","Priority":"2","ResponseMe
","SenderID":"heg/global_execution/production_execution_control/8c0151d9-624d-11e7-9a17-108520524153","MessageID":"5678","Subtype":"notifi
170706151838","ProtocolVersion":"0.8","ExternalBrokers":"*","Internal":"true","SenderBroker":"","Body":{"op":"call_service","service":"/ta
,1","task_id":"4","task_instance_id":"9f94c3a1-624d-11e7-9a17-108520524153","process_instance_id":"9f7de0cf-624d-11e7-9a17-108520524153"}}}
***WebSocketReceiverApplicationImpl*** Relaying message towards dispatcher:HEG
***WebSocketReceiverApplicationImpl*** Relaying message towards dispatcher:{"Topic":"task_assigned","Priority":"2","ResponseMessageID":"",
:"heg/global_execution/production_execution_control/8c0151d9-624d-11e7-9a17-108520524153","MessageID":"5678","Subtype":"notification","Type":"2","Timestamp":"20170706151838
","ProtocolVersion":"0.8","ExternalBrokers":"*","Internal":"true","SenderBroker":"","Body":{"op":"call_service","service":"/task_request","args":{"agent_ids":"3,1","task_id
":"4","task_instance_id":"9f94c3a1-624d-11e7-9a17-108520524153","process_instance_id":"9f7de0cf-624d-11e7-9a17-108520524153"}}}
***We
***Me
```

GLOBAL PROCESS
MIDDLEWARE
LOCAL TASK
AUTOMATED AGENT
The Global process starts local task execution via MessageBus

Figure 6: Early Prototype Demonstration - the Broker processes the message

The Broker forwarded the message to the ROS Bridge.

The ROS Bridge processed the message body and based on the value of "Service" property determined the ultimate recipient of the message (in this case, FlexBe). Then the ROS Bridge converted the JSON formatted message into a ROS message and sent it to FlexBe.

FlexBe retrieved the model for execution of the given task and started the execution of its workflow by sending ROS instructions to the robot interface (like the one presented in Section 2.3.1.2 /FlexBe Messages).

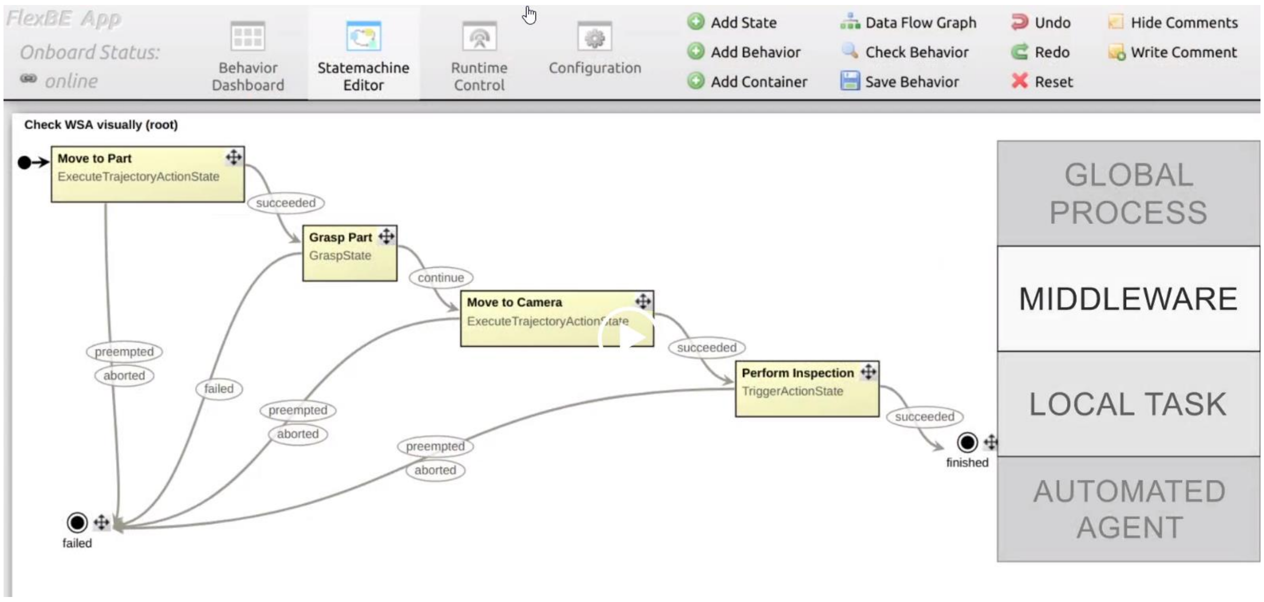


Figure 7: Early Prototype Demonstration - FlexBe state machine

The robot interface instructed the robot (automated agent) to perform a specific operation (pick up an object, move it and drop it in a specified location).

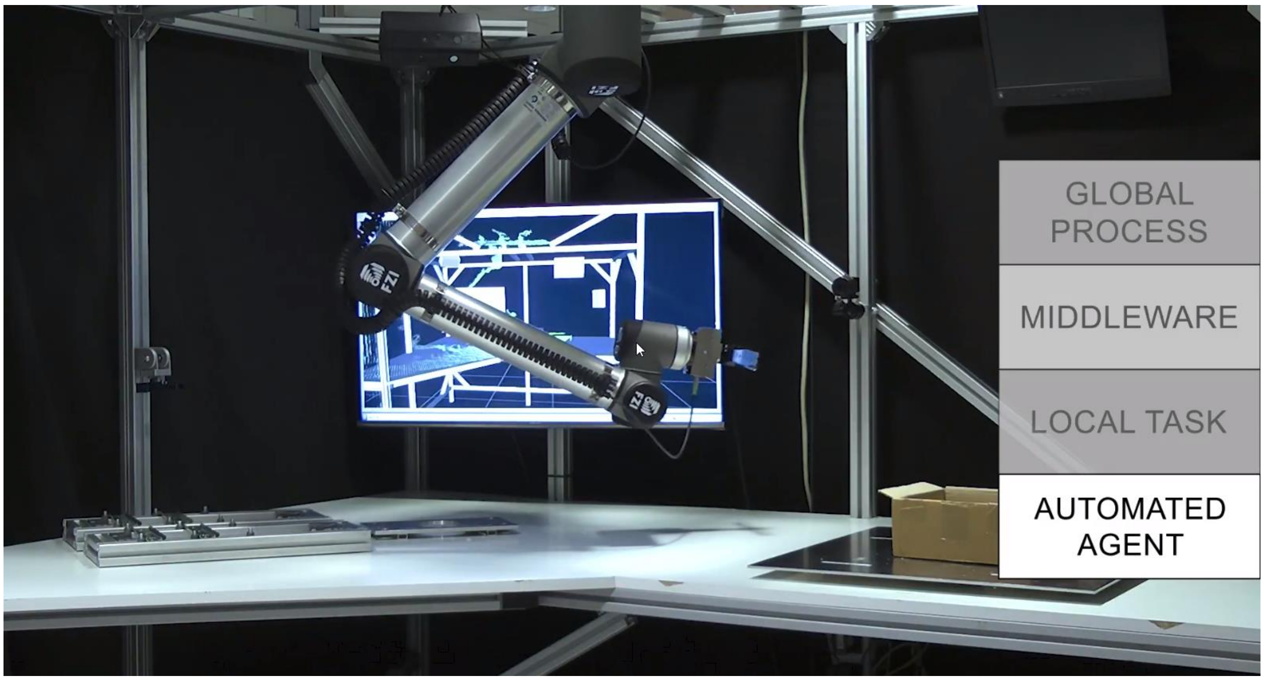


Figure 8: Early Prototype Demonstration - Robot operation

The results of the robot operations were processed by FlexBe and when the task was completed, the FlexBe sent a notification message (as the one in Section 2.1.1.2.2) up the communication channel back to MPMS.

The MPMS identified the progress, followed the decision path and initiated the next task.

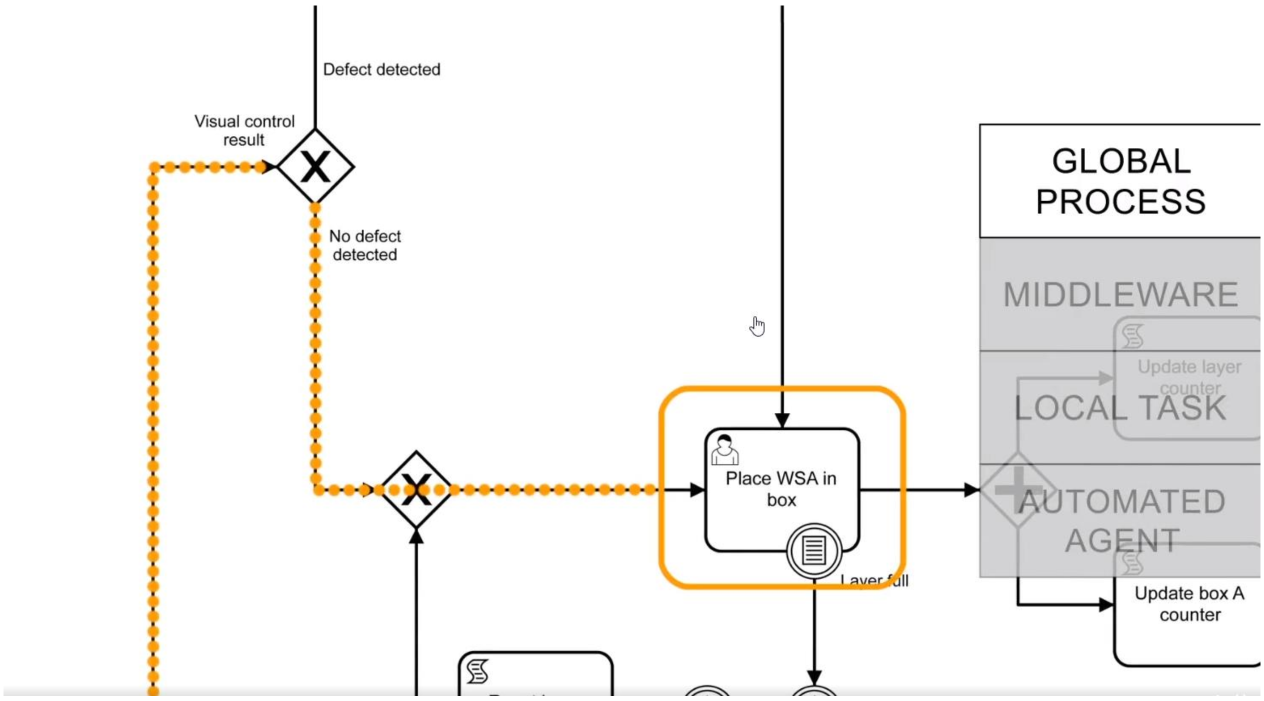


Figure 9: Early Prototype Demonstration - MPMS workflow progress

In parallel to the main process (the production track), a constant evaluation of the execution environment and participants has been performed. Two IR cameras monitored the work cell for human intrusion or deviations from the planned behaviour. Any disruptions have been reported to FlexBe.

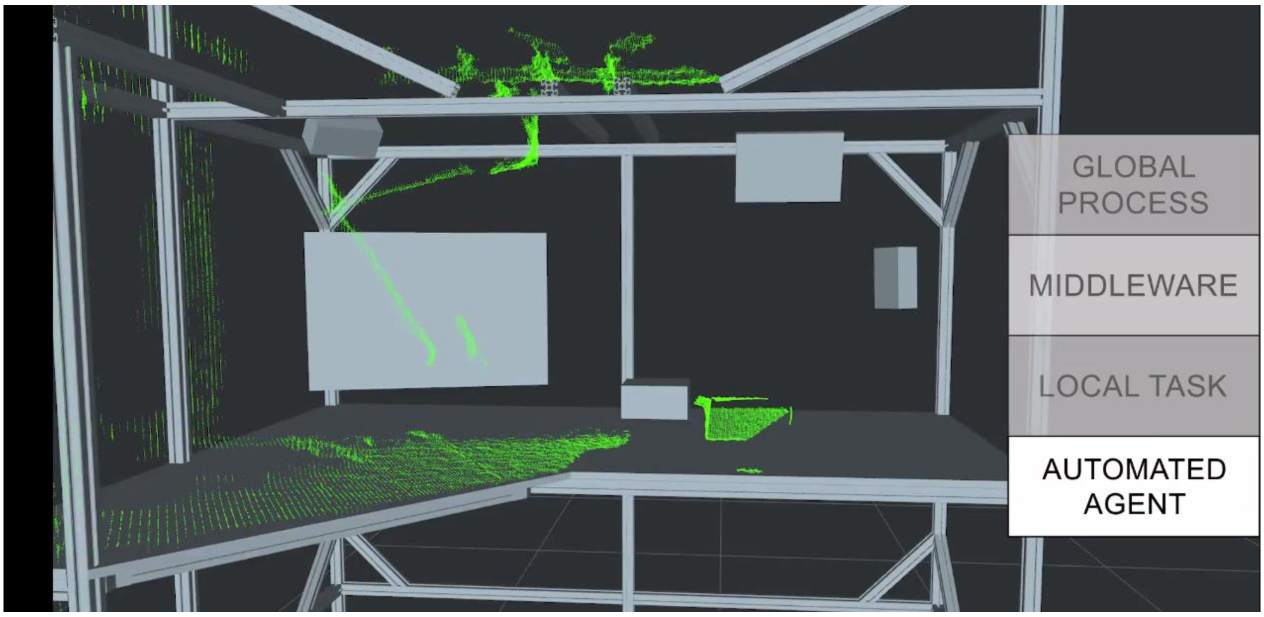


Figure 10: Early Prototype Demonstration - Situation awareness

Figure 10 shows the point cloud of the live environment constructed with the data of the IR cameras.

The next figure gives a schematic presentation of the demonstration steps presented above.

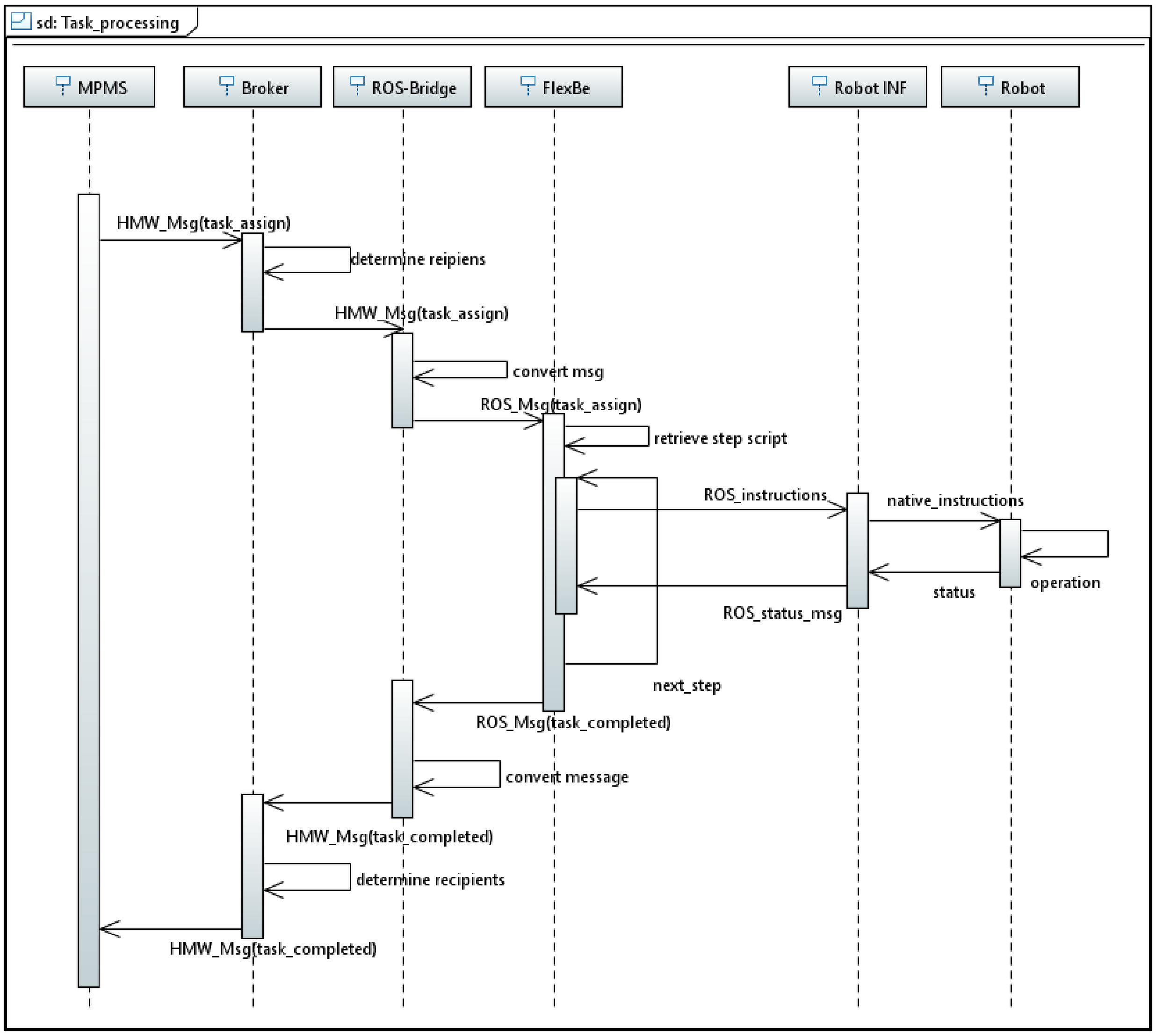


Figure 11: Early Prototype Demonstration - Task Execution

A video with the highlight of the demo could be watched at:
https://www.youtube.com/watch?v=hD1vqzykLkU

6 Integration Plan - Revised

This section presents the time and resource constrains of the WP4 activities, defined in the DoW. Then it introduces the integration plan for bringing together the WP3 components into a single platform.

6.1 Timeline

Figure 12 presents the timeline of the activities within Work package 4.

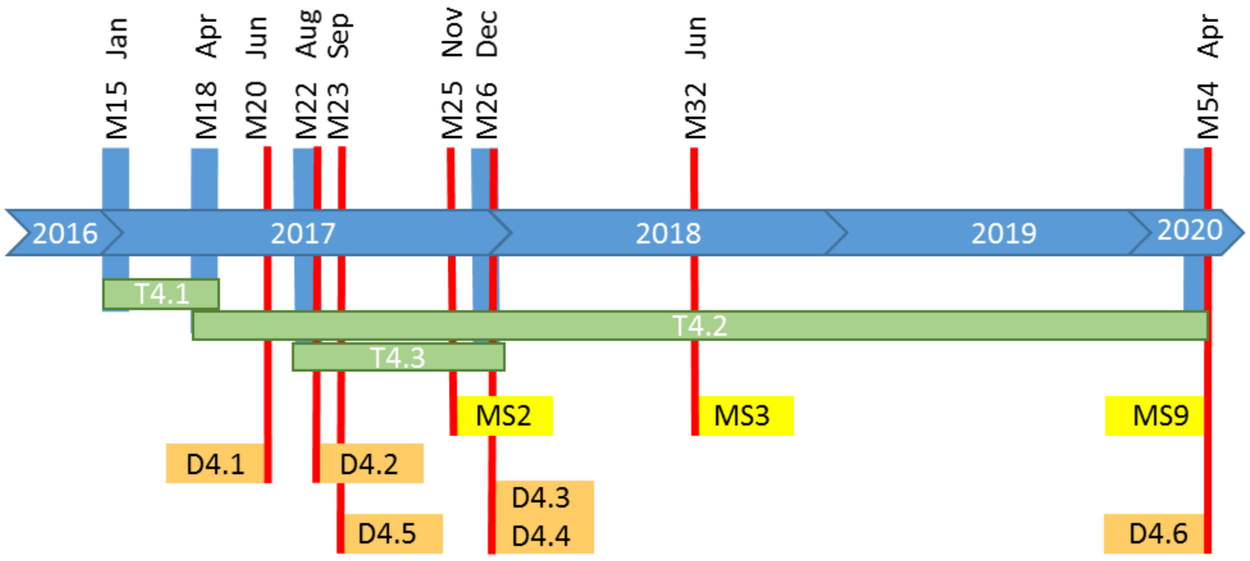


Figure 12: Timeline of WP4 activities

The activities in WP4 start at project month 15 (January 2017) and continue till the end of the project (project month 54, April 2020).

6.1.1 Milestones

There are three important milestones (the yellow boxes in the timeline), related to the product of this work package:

MS2 - First Integrated Prototype and Pilot Users Feedback

This milestone has been reached in November 2017 (project month 25). It marked the release of the early version of the integrated HORSE platform, its demonstration to the pilot users and the collection of their feedback. The key features of this release:

- Full functioning messaging middleware – the components should be able to send messages to the intended recipient components and these messages should be properly delivered - ACHIEVED;
- Databases that could be accessed by the components for retrieving and storing data. Availability of sample data for internal test purposes - ACHIEVED;

- The individual HORSE components should implement the functionality planned for this milestone as defined in their implementation plan - ACHIEVED;
- The integration of the devices, robots and machines should be at least of level of ability to communicate with the platform – PARTIALLY ACHIEVED.

The components that constitute the early prototype and their functional and integration maturity are presented in Chapter 2. More details are presented in Section 6.4 Integration Level.

MS3 - Final Integrated Prototype and Pilot Users Feedback

This milestone in June 2018 (project month 32). It marks the completion of the development process and the full integration of all components. The customised versions of the complete HORSE platform should be deployed at pilot sites and made available for field trials in industrial conditions. The definition of "complete HORSE platform" is presented in Section 6.4 Integration Level.

Update
The integration activities are on track and the milestone will be reached on time.

MS9 - Final HORSE

This milestone marks the completion of HORSE project, scheduled for April 2020. The integration efforts up to this moment comprise of updating the deployed instances of the platform with new releases of the HORSE modules, containing bug fixes and implementations of minor but critical changes whose need has been identified during the platform exploitation at the pilot sites and Competence Centres.

Update
No deviations of the plan are observed.

6.1.2 Work Organization

The implementation activities are divided into three tasks, as follows:

Number	**Name**	**Begin**	**End**
T4.1	Integration plan and infrastructure	M15	M18
T4.2	Integrated platform versions	M18	M54
T4.3	Integration Tests	M22	M26

Table 10: WP4 tasks

The bulk effort should be invested up to project month 32 (Milestone 3), when the final release of the integrated HORSE platform should be made available for field trials at the pilot test sites and centres of competence. From that moment on there will be no development and integration, but only support.

The partners' activities are synchronised in bi-weekly telephone conferences, remote and physical integration sessions. The work progress is reported to the rest of the consortium during the telephone conferences of HORSE Coordination Board and the relevant physical meetings.

A virtual platform has been setup as integration and test server in order to experiment with deployment and operation of the HORSE components.

6.1.3 *Deliverables:*

Number	Name	Short Description	Due
D4.1	Integration plan and description of the integration infrastructure	Initial integration plan, strategy and infrastructure.	Originally scheduled for M17 (Mar 2017) First release in M20 (Jun 2017) – rejected Revision in M25 (Nov 2017) –resubmitted
D4.2	Early version of the integrated platform and new integration plan	A demonstration of the early prototype and an update of the integration plan This document	Originally scheduled for M20 (Jun 2017) Rescheduled to M25 (Nov 2017) To be released in M29 (Mar 2018)
D4.3	Final Version of the integrated platform	Specification of the integrated platform (description of the hardware, modules and their versions, integrated sensors and actuators, communication channels and protocols) and demonstration of its final integration.	M26 (Dec 2017) Rescheduled to M32 (Jun 2018)
D4.4	Test Report	Integration tests report, containing description of the integration test cases with reference to their source code, log of execution of the integration test cases and registration of raised issues with their status	Originally scheduled for M26 (Dec 2017) Realistic release M32 (Jun 2018) with the release of the final prototype
D4.5	User Handbook	A guide for the HORSE platform. This deliverable should contain the description of the HORSE platform and its functionality for user point of view. The platform itself should be completed shortly after.	Originally scheduled for M22 (Aug 2017) Expected in M28 (Feb 2018), to be used by the pilot sites deployments
D4.6	Final HORSE Framework	Demonstration of the final HORSE framework, updated	M54 (Apr 2020)

		according to the final users feedback	

Table 11: WP4 deliverables

The schedule of the WP4 activities and the early deliverables suffered from a short delay. It is due to the technical problems on the setting up the integration infrastructure and experimenting with partners' modules and products from within Bosch network after the adoption of the Bosch security and communication restrictions on the acquired ProSyst Software GmbH.

In addition, not all HORSE components provided the required maturity as expected. The danger of delay in the release of the early integrated platform and the accompanying integration tests and thus missing the MS2, has been thwarted by investing of unplanned resources in aligning and negotiating the interfaces of the components. This resulted in reducing the efforts of the documenting the work done. This is the reason for the significant delay of this document.

The progress of the integration activities provides confidence that the next milestone will be reached on time.

6.1.4 Bilateral Integration Activities

The status of the realisation of the functional relationships is given in Section 1.2 and Chapter 2

6.2 Resources

The integration efforts are supported by the majority of the partners. The partners developing software modules or providing equipment would assist in the proper integration and testing of these components. The representatives of the pilot test sites would assist in the customisation and deployment of the HORSE Platform in their production facilities. They would also provide the valuable feedback from the field trials.

The next table and diagram present the partners resources per task and the share of each task in the entire efforts pool.

Partner \Task	**T4.1.**	**T4.2.**	**T4.3.**	**Sum**
ED		4,0		4,0
CEA	0,5	3,0	2,0	5,5
FZI	1,0	6,5	3,0	10,5
PROS	5,0	12,0	7,0	24,0
TUE	4,0	4,0	5,0	13,0
OPSA	1,0	5,0		6,0

KUKA	1,0	3,0	1,0	5,0
BOS	1,0	4,0		5,0
TUM	2,0	2,0	2,0	6,0
TNO	1,0	3,0	5,0	9,0
CET	1,5	2,0		3,5
Total	**18,0**	**48,5**	**25,0**	**91,5**

Table 12: Planned partners' resources in WP4 tasks

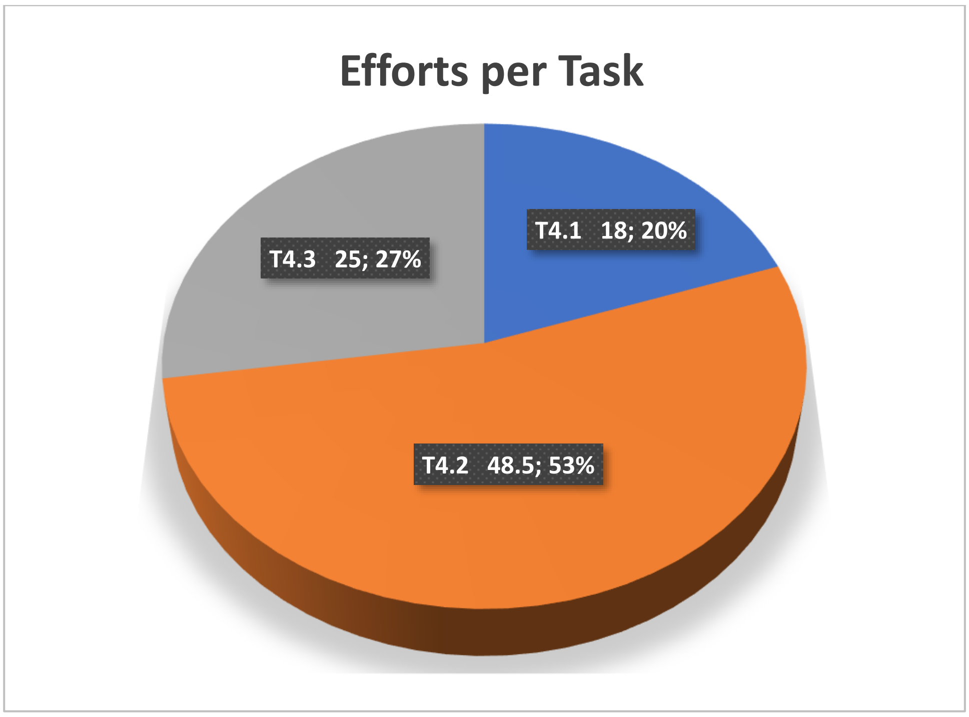


Figure 13: WP4 efforts distribution

As clearly seen on *Figure 13*, the greatest amount of the efforts is planned for actual integration, followed by designing, implementing and performing integration tests.

6.3 *Integration Process*

The HORSE integration process is an iterative one with increasing level of automation and functional coverage.

It could be divided into two major parts:

- Integration and Testing of the Internal Builds and Prototypes

- Integration, Update and Testing of the Pilot Platforms

.

6.3.1 *Integration and Testing of the Internal Builds and Prototypes*

This is the initial and internal part of the integration work. It includes all activities up to the release of a testable prototype for the pilot sites. During this period the components and the test cases are gaining in complexity and stability. Figure 14 visualises the process. The orange boxes denote interaction with the integration infrastructure.

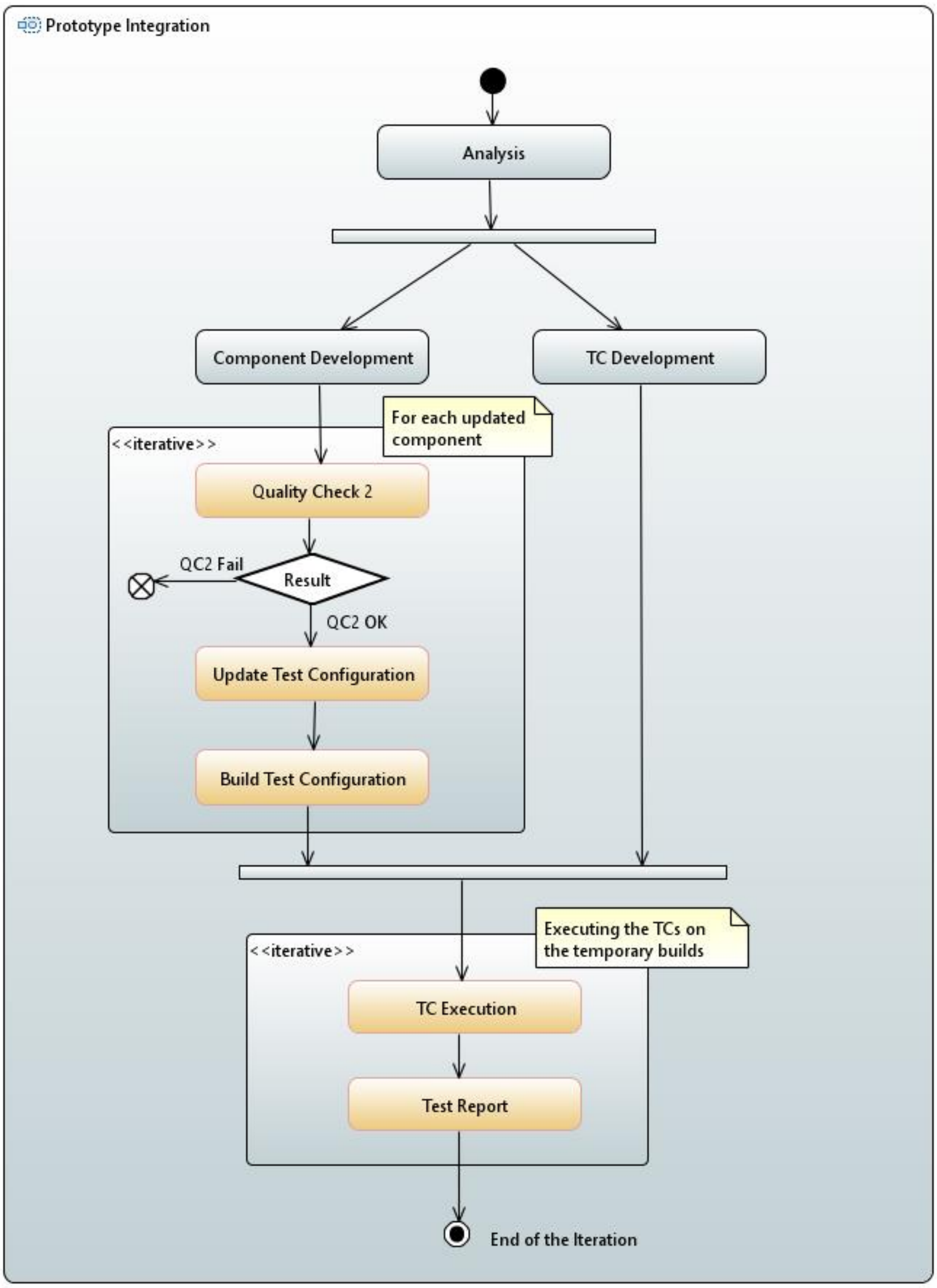


Figure 14: Integration of the internal prototypes

The process is iterative and incremental. Each iteration contains the following groups of operations:

A. Preparation and analysis
Each iteration starts with analysis of the results of the previous iteration. For the very first iteration the analysis is done based on the output of WP2. Special attention is paid on the reported problems, which are prioritised.

B. Development of HORSE components and features
The WP3 developers fix the identified problems, add new features and increase the stability of their components. They configure their development environment. The activities related to the Component Development are presented in next diagram.

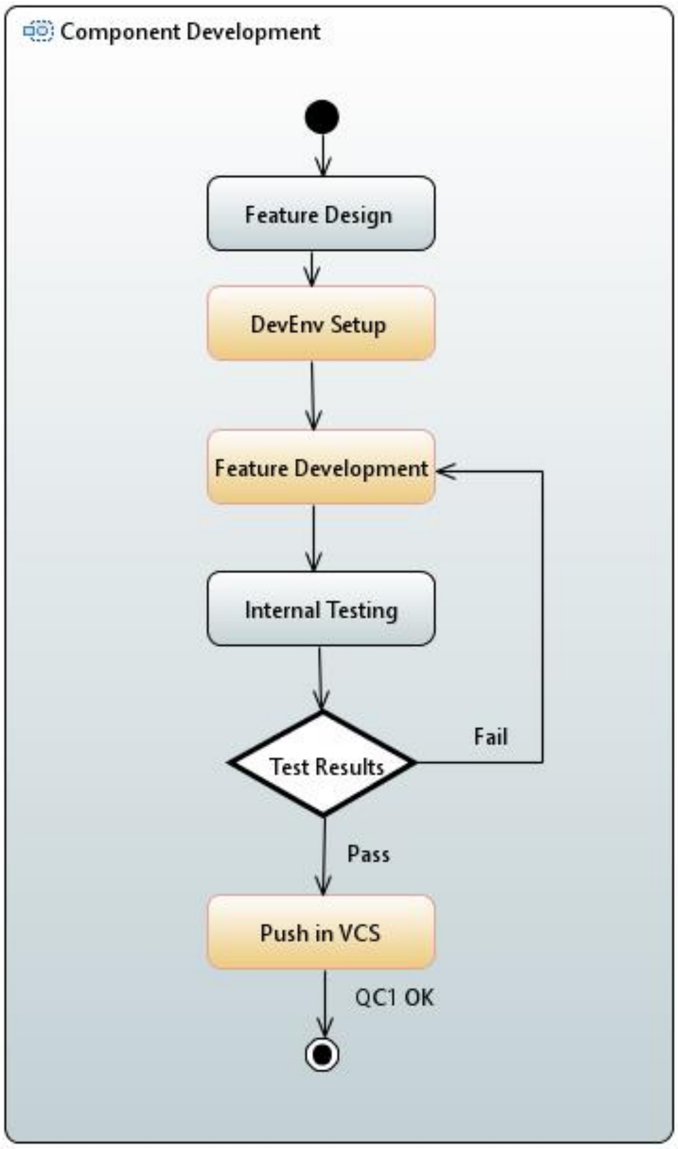


Figure 15: Component development

B.1 Feature Design.

In this step the features to be implemented or the changes to be applied are properly designed and documented.

B.2 Setup of the Development Environment.
The developers retrieve from the integration server (repository of source code or binary artefacts) the latest stable versions of the needed HORSE components and libraries.

B.3 Feature Development
The developers implement the assigned feature. For team work purposes the intermediate work could be stored in the source code repository.

B.4 Internal Testing
The partners responsible for the implementation of a given component (or its feature) are responsible for its quality. The team performs internal validation (unit tests, code review etc.) of the component before submitting it to the integration testing. In case of problems, several development iterations can be performed.

B.5 Push in the Version Control System
The healthy code is submitted into the Version Control System (VCS) with a mark "QC1 OK" (Quality Check 1) as ready for the integration testing.

C. Development of the test cases (TCs)
The TCs are auxiliary applications and scripts, aimed to verify and validate specific capabilities of the HORSE platform, respectively HORSE components. The steps are similar to those for the HORSE components' development. However, because the TCs are not part of the HORSE platform, they are tested only internally.

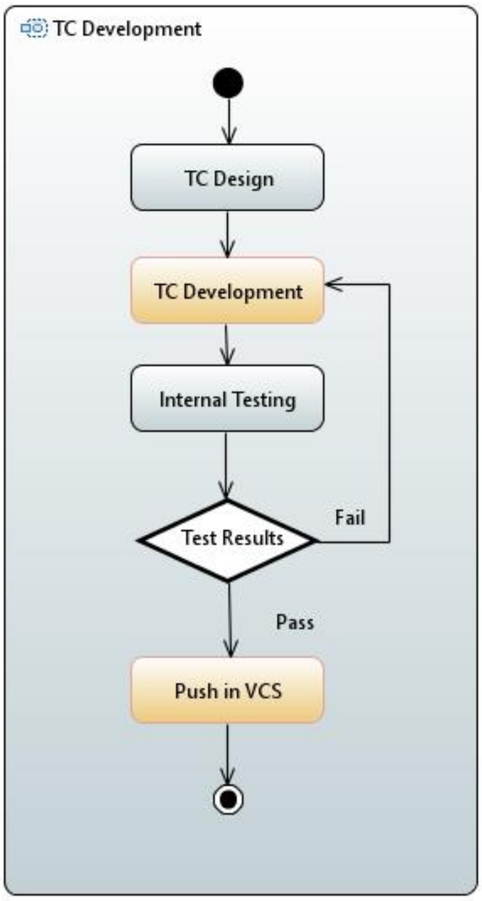


Figure 16: TC development

C.1 TC Design
The design of the TCs includes specification of the tested components, initial conditions and input data, interaction with the tested modules and expected output data. The TC definitions are formal descriptions of the HORSE use cases.

C.2 TC Development
The TCs are implemented with the help of the development tools. For team work purposes the intermediate work could be stored in the source code repository.

C.3 Internal Testing
This step aims to check the quality of the implemented TCs.

C.4 Push in VCS
The healthy TCs are made available for use by the testing environment

The steps in each group B or C are performed independently. Within each group multiple iterations are possible.

D. Update Test Configuration
The Test Configuration lists all components that are eligible for integration testing. At predefined periods (e.g. at night) or conditions (e.g. new QC1 update), the CI Server initiates the compilation and consistence check of the QC1 tagged component update. Upon success the update is added to the Test Configuration.
This group of operations is executed for each QC1 tagged component update.

D.1 Quality Check 2
The CI Server retrieves the QC1 tagged components updates from the VCS. It executes the building tools and scripts to compile and build temporary instances of these components. In case of success, the component is tagged as "QC2 OK" (Quality Check 2 – successful compilation by CI, ready for next step). Otherwise the update is tagged as "QC2 Fail" and a report is sent to the developer.

D.2 Update of the Test Configuration
The list of the test-ready components (the Test Configuration) is updated with the new version of the freshly certified component. A deployable build of the component is stored in the Artefacts Repository.

D.3 Build of the Test Configuration
The CI Server retrieves the deployable builds of Test Configuration components from the Artefacts Repository and deploys/installs them on the test platforms.

E. Integration Testing
The testing application attempts to run the available TCs on the temporary builds. For each TC the following steps are executed.

E.1 TC Execution
The TC feeds the testable components with the predefined input data, interacts with the system simulating the external systems, actors and devices, and keeps record of the processes and communication. The debug information is collected. A test protocol is created.

E.2 Test Report
Once the TC execution ends, a comparison of the observed end conditions and the expected ones is done. The test report is created and stored in the code repository. In case of problems or misbehaviour a ticket in the Issue Tracking System is created. If possible, the ticket is assigned to the developer of the component, causing the problem.

After the successful completion of all TCs, the Test Configuration is considered integrated prototype (i.e. ready for field trials at pilot sites).

6.3.2 Integration, Update and Testing of the Pilot Platforms

Once the individual HORSE components reach the required level of maturity and the integrated prototype passes the internal tests, it can be deployed at the test sites and validated in industrial conditions with real equipment. This is the initial point of a new cycle of iterative integration activities, as schematically depicted on Figure 17. These are applied on any stable system updates produced and internally tested, as described in the previous section

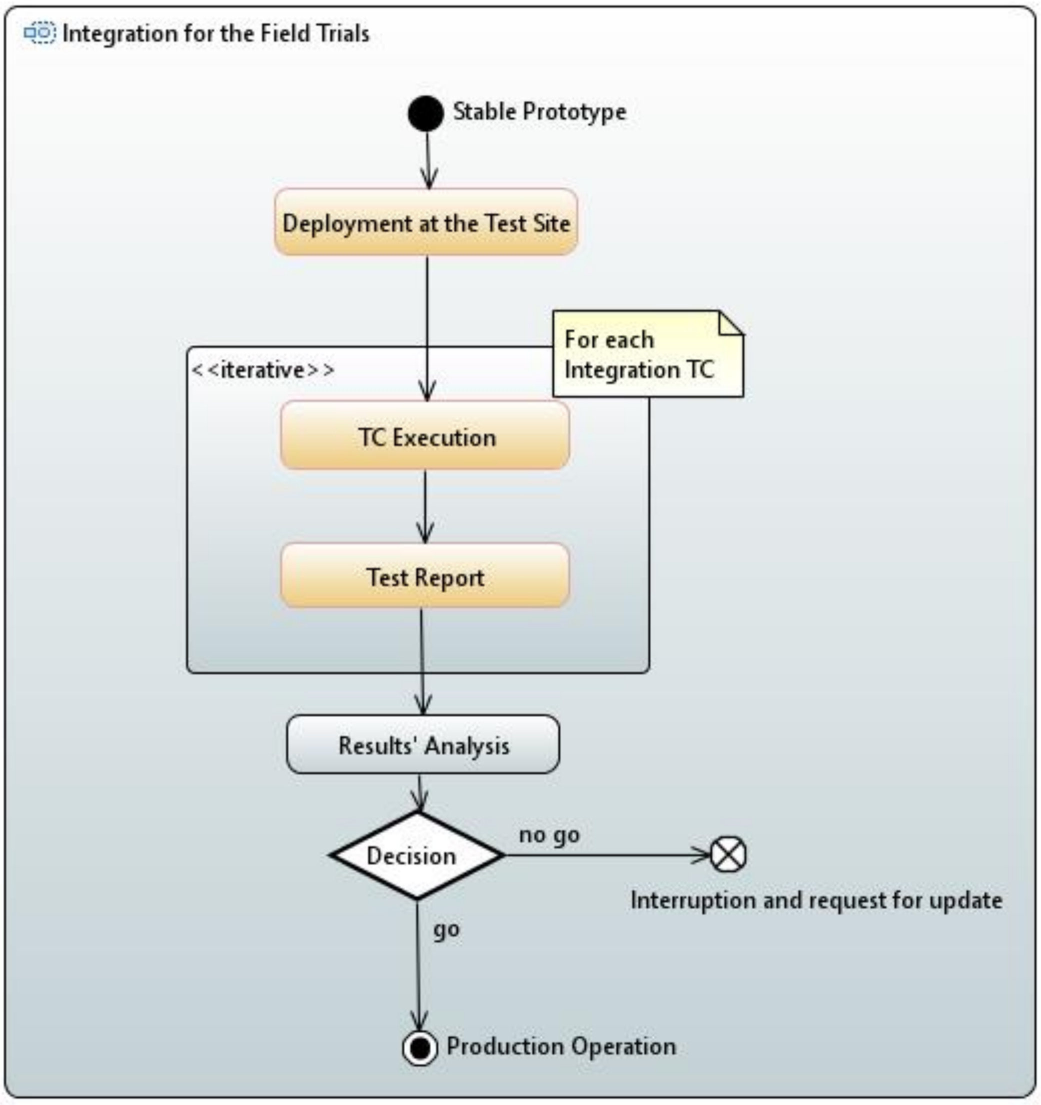


Figure 17: Integration and testing of the pilot platform

Using the deployment tools and scripts, the prototype components are installed at the target platforms at the pilot test sites. (It should be reminded that the HORSE Framework is a distributed solution, consisting of at least two physical entities, whose components are collaborating via a unified messaging infrastructure).

This step is followed by a cycle of execution sessions for each relevant integration TC. These are performed with real industrial equipment, factory operators and the IT infrastructure at the test sites. After the completion of each tests session, the test protocol is submitted to the integration server and the identified issues are reported to the development/support team by creating an issue in the ticketing system.

The results of all TCs are analysed and a decision if the prototype could be handed over for production operation is taken.

Once HORSE platform is accepted for production operation, any new system update undergoes the integration and validation processes described in sections 6.3.1 and 6.3.2 before updating the live platform.

6.4 Integration Level

The degree to which a component is capable to interact with the other HORSE components, actors and external software and hardware determines its integration level. The integration level depends on the maturity level of the component. The integration level of the entire HORSE platform is a direct product of the integration levels of its components.

In respect of integration the following maturity levels of the components developed in the HORSE project are identified:

Level	**Description**	**Integration aspect**
Mockup	No implementation available. The component's interfaces are identified and aligned with the other relevant components. The common data structures are defined.	The data and control flows are described. The integration is at design and planning stage.
Empty shell (Placeholder)	No business logic available. The component implements the HORSE messaging agent and is able to send and receive messages through the HORSE messaging middleware. The value of the sent message/request parameters is not relevant	Communication is realised on transport level only.
Dummy	No real business logic available The component is capable to: • compose messages/requests structured according to the interface specification and send them to the other components. The value of the sent message/request parameters could be predefined (hardcoded). • receive messages/requests from other components and retrieve their parameters and log them. No further processing of the received data is required.	The components are aware of each other. The execution of an exemplary scenario, in which the components are exchanging predefined data, could be demonstrated.

Development prototype	Partial implementation (core functionality and some of the high-priority interfaces) The component is capable to respond correctly to some of the received messages. It is assumed that the delivered data is meaningful and properly formatted, so no exception handling is done. Having in mind the limited functionality, the component can be used together with other components.	The TCs are designed to cover the available functionality. Not all UCs can be realised and validated.
(Part of) Early integrated platform =MS2	The business logic for the high-priority interfaces and features (all HORSE components). The interfaces and business logic of the component are implemented according to the implementation plan (WP3). The received data is checked for consistency. Deviations and exception are partially handled.	The TCs cover all components and their functionality (according to the WP3 implementation plan). End-to-end testing with properly selected pre-conditions and data completes without exceptions and the observed results (post-condition) match the expected ones
Final integrated platform =MS3	The component is implemented according to the specification at the agreed level of completion.	The TCs are successfully executed with great range of test data. The incorrect data is properly handled. The integrated platform can be validated at test sites.

Table 13: Maturity levels

Next table presents the expected functionality of HORSE components for the early and final versions of the integrate platform (resp. MS2 and MS3). An update of the status at MS2 is added.

Component name	**MS2 Features**	**MS3 Features**
Messaging Middleware	Complete implementation of the MW components (agents, dispatcher & broker); Other components can exchange messages; ACHIEVED	History Log added, Complete implementation
Databases	Initial version of the table structures;	Final table structures; Real data for the pilot sites;

	Dummy data; Scripts for creation of the tables and feeding the data; The individual components access the tables directly ACHIEVED	Updated and final scripts.
Agent Mgr	Functional implementation complete; Exchange of the correct data over the middleware; ACHIEVED Storing of dummy data in DB; DROPPED	~~Storing the correct data in DB~~ (DROPPED); Components can retrieve information about the available agents.
Augmented Reality (pilot specific)	Projecting of virtual controls and instructions; Execution of sample workflows; Exchange of dummy data over the middleware; ACHIEVED Storing of dummy data in DB; DROPPED	Implementation of the pilots' specific workflows complete; Exchange of the correct payload over the middleware; ~~Storing the correct data in DB~~ (DROPPED)

AutAgent Step Exec
(Pilot / HW specific)
	Simple interaction of KUKA robot over ROS;
	Simple interaction of KUKA robot over Messaging MW;
	ACHIEVED
		Interaction with pilots' robots over ROS and MsgMW.

Cameras & Sensors	The components' specific sensor and cameras are integrated in the respective components; ACHIEVED	Additional cameras and sensors according to the needs of the pilot sites.
Conveyor Belt (BOS)	Initial implementation of PLC communication (transport level);	Implementation of complete PLC communication (transport & payload);

	Exchange of dummy data over the middleware; ACHIEVED	Exchange of the correct payload over the middleware;
Deviation Monitor	Implementation of the processing logic complete; Operation with dummy data and rules; Exchange of dummy data over the middleware; ACHIEVED Storing of dummy data in DB; DROPPED	Implementation of pilot specific features (rules & data); Exchange of the correct payload over the middleware; ~~Storing the correct data in DB~~ (DROPPED)
Device Abstraction	Support of generic device classes; Exchange of generic sensor data and commands over the middleware; Initial version of the device model. Storing of dummy data in DB; MISSING (no UC and devices)	Support for pilot's specific devices and protocols complete Device model updated; Exchange of the correct payload over the middleware; Storing the correct data in DB TO BE RECONSIDERED
Device Manager	Support of device protocols ZigBee and Z-wave; Integration with Device Abstraction; Storing of device configuration data in DB; MISSING (no UC and devices)	Support of pilot specific protocols added; TO BE RECONSIDERED
Global Awareness	Implementation of the engine complete; Execution of sample rules; Exchange of dummy data over the middleware; Storing of dummy data in DB; PARTIALLY ACHIEVED	Implementation of pilot specific features; Rules for all pilots implemented; Exchange of the correct payload over the middleware; Storing the correct data in DB

Global Execution (MPMS)	Implementation of the engine complete; Execution of the initial versions of the pilots' workflows; Exchange of dummy data over the middleware; Storing of dummy data in DB; ACHIEVED	Implementation of pilot specific features; Workflow scripts for all pilots implemented; Exchange of the correct payload over the middleware; Storing the correct data in DB
HumAgent Step Exec (pilot specific)	Sample interaction with the Augmented Reality engine (feeding sample instructions); Simulation via messages for the missing HMI devices. Exchange of dummy data over the middleware; ACHIEVED Storing of dummy data in DB; MISSING (consider dropping)	Interaction with the HMI devices to be used in pilot tests. Exchange of the correct payload over the middleware; Storing the correct data in DB
Human Detection/Tracking	Implementation of the detection and tracking logic complete; Exchange of dummy data over the middleware; ACHIEVED Storing of dummy data in DB; DROPPED	Data models for the pilot specific environments; Exchange of the correct payload over the middleware; ~~Storing the correct data in DB~~ (DROPPED)
Human Machine Interface (UC specific)	Simulation via messages. MISSING	Implementation of pilot specific HMI; Exchange of the correct payload over the middleware;
Hybrid Task Supervisor	Implementation of the FlexBE engine complete; Execution of sample ROS scripts for steps execution; Exchange of dummy data over the middleware;	All ROS scripts implemented; Exchange of the correct payload over the middleware; ~~Storing the correct data in DB~~ (DROPPED)

	ACHIEVED Storing of dummy data in DB; DROPPED	
KUKA AutAgent INF (BOS)	Sending sample ROS instructions to a KUKA robot. MISSING	Sending of all BOS specific operations to KUKA robot, used by BOS pilot.
KUKA AutAgent INF (OPSA)	Exchange of dummy data (instructions & status) over the messaging middleware; MISSING	Exchange of the correct data with the KUKA robot, used by OPSA pilot;
KUKA AutAgent INF (TRI)	Exchange of dummy data (instructions & status) over the messaging middleware; MISSING	Sending of all TRI specific operations to KUKA robot, used by TRI pilot.
Local Safety Guard	Implementation of the core logic complete; Operation with dummy data and rules; Exchange of dummy data over the middleware; ACHIEVED Storing of dummy data in DB DROPPED;	Completion of the pilot specific features (data & rules); Exchange of the correct payload over the middleware; ~~Storing the correct data in DB~~ (DROPPED)
Notification Beacon (BOS)	Conceptual design of the OPC-UA communication; Specification of the communication with the HORSE Messaging Middleware. MISSING	Exchange of the correct payload over OPC-UA; Exchange of the correct payload over the middleware;
Object Detection/Tracking	Implementation of the detection and tracking logic complete; Exchange of dummy data over the middleware; Storing of dummy data in DB; MISSING	Data models for the pilot specific artefacts and environments; Exchange of the correct payload over the middleware; Storing the correct data in DB

VisionControl (BOS)	Initial implementation of etherCAT communication (transport level); Exchange of dummy data over the middleware; ACHIEVED	Sending of the correct payload over etherCAT; Handling of the check results and images; Exchange of the correct payload over the middleware; Storing the correct data in DB

Table 14: Completion Level per Component

7 Integration Infrastructure - Revised

The integration infrastructure should provide the means for the software developers, integrators and testers to perform their tasks described in Section 6.3 Integration Process.

Table 15 lists the integration infrastructure parts mapped to the individual parts of the integration process.

Operation	**Done by**	**Involved II Component**
Analysis of reported issues	Developer	Ticketing System
Setup of the Development Environment (Component Development)	Developer	Artefacts Repository (Nexus) Source Code Repository / Version Control System (Git)
Feature/TC Development (Component Development)	Component/TC Developer	Source Code Repository / Version Control System (Git)
Push in the Version Control System (Component Development)	Component/TC Developer	Source Code Repository / Version Control System (Git)
Retrieval from the Version Control System (QC2)	Automated Script	CI Server (Jenkins) Source Code Repository / Version Control System (Git)
Building of the component (QC2)	Automated Script	CI Server (Jenkins) Building tools (Maven, CMAKE Script) Artefacts Repository (Nexus)
Update of the Test Configuration	Integrator	Source Code Repository / Version Control System (Git)
Building Deployable Units	Integrator	Building tools (Maven, FZI Script; ProSyst mToolKit)
Configuration of the Test Environment	Integrator	Test Execution Framework (ProSyst TEE)
Deployment of the Test Components	Integrator / Scripts	Artefacts Repository (Nexus)

Execution of Internal Integration Tests	Test Engineer / Script	Test Execution Framework (ProSyst TEE)
Test Report	Test Engineer / Script	Test Execution Framework (ProSyst TEE) Source Code Repository / Version Control System (Git) Ticketing System

Table 15: Integration tasks and tools

The majority of the integration infrastructure components will be hosted and executed on a virtual machine provided by TUM and featuring Ubuntu operating system. The VCS is realised as dedicated Git project at the TUM public server with managed access.

8 Integration Risks - Revised

The following risks have been considered. An update is provided, when needed.

ID	Risk	Probability	Severity	Measure
IntR-1	Component implementation missing or delayed	~~Low~~ High	High	Regular progress checks; Intensive alignment
IntR-2	Design and implementation of test modules too complex resulting in insufficient test coverage.	High	Medium	Revision of the test priorities
IntR-3	New/changed requirements	Medium	Medium	Detailed requirements specification through an intensive collaboration of all stakeholders.
IntR-4	Critical equipment missing (e.g. due to the high acquisition & maintenance costs)	Low	High	Early specification of the needed equipment with cost estimation, procurement planning
IntR-5	Wrong time estimation (could result in IntR-1)	Medium	Medium	Regular progress checks and review of the priorities
IntR-6	Unexpected project scope expansion	Low	High	Understanding and agreement on the project scope by all stakeholders before the integration start
IntR-7	Wrong budget estimation (could result in IntR-1, IntR-2 & IntR-4)	Medium	Medium	Regular progress and costs checks
IntR-8	Dropping off a key contributor (person)	High	Medium	Proper knowledge management to enable seamless takeover of responsibilities by other contributor
IntR-9	Dropping off a partner (organization)	Low	High	Due the specialization of the partners a complete takeover of responsibilities by a single party is unlikely.

				However, the shared expertise of the consortium is sufficient to deal with partitioned responsibilities of the partners
IntR-10	Insufficient team communication	Medium	Medium	Regular synchronization calls, issue review, documentation
IntR-11	Technical problems (competency gap) by the integration of the existing equipment and technologies at pilot test sites or competence centers.	High	High	Intensive collaboration and commitment of all involved parties, especially the pilot hosts and competence centers hosts

Table 16: Integration risks

9 Conclusion and Next Steps

This document describes the early HORSE prototype and presents un update of the integration plan.

The following has been observed:

- Most of the components are implemented and integrated according to the initial plan
- The HORSE framework allows the existence of multiple solutions for one component. This allows the end user to select the one that fits his needs best.
- This official deliverable suffers from a significant delay.

The next steps in the WP4 activities include:

- Gap analysis of the D2.2 Use Cases and distribution of the responsibilities on designing and developing of the TCs,
- Deployment guide for all components
- Identification, design, implementation and execution of key TCs
- Release of the final integrated HORSE platform